Anytime Algorithms for Stream Data Mining

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften
der RWTH Aachen University zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Informatiker
Philipp Kranen

aus Willich, Deutschland

Berichter: Universitätsprofessor Dr. rer. nat. Thomas Seidl
Visiting Professor Michael E. Houle, PhD

Tag der mündlichen Prüfung: 14.09.2011

Contents

Abstract

Data is collected and stored everywhere, be it images or audio files on private computers, customer data in traditional or electronic businesses, performance or control data in production sites, web traffic and click streams at internet providers, statistical data at government agencies, sensor measurements in scientific experimentation, surveillance data, etc. There are countless examples, and the amount of data is tremendous. Data mining is the process of finding useful and previously unknown patterns in data. In the examples listed above, data mining can be used for automated recommendation of audio files, business analysis and target marketing, or performance optimization and hazard warnings. While early mining algorithms only considered static data sets, research and practice in data mining must nowadays deal with continuous, possible infinite streams of data, which are prevalent in most real world applications and scenarios.

Anytime algorithms constitute a special type of algorithm that is well suited to work on data streams. They inherit their name from their ability to provide a result after any amount of processing time. The amount of time available is not known to the algorithm in advance: anytime algorithms quickly compute an initial result and strive to improve it as long as time remains. When interrupted they deliver the best result obtained until that point in time.

In this thesis anytime classification is studied in depth for the Bayesian approach. New algorithmic solutions for anytime classification are developed and evaluated in extensive experimentation. The first anytime stream clustering algorithm is proposed, and an application to anytime outlier detection is presented. In addition to the algorithmic contributions, new meta-approaches are described that significantly widen the area of applications for anytime algorithms. The solutions and results of this thesis contribute to the state of the art in anytime algorithms and stream data mining research.

Zusammenfassung

Die rasante Entwicklung der Informationstechnologie hat zur Folge, dass in allen Bereichen der Gesellschaft und des täglichen Lebens große Mengen an Daten erzeugt und gespeichert werden. Beispiele reichen von Multimedia-Daten auf privaten Computern bis hin zu Messdaten in wissenschaftlichen Experimenten. Data Mining beschreibt die Aufgabe, in solchen Daten neue und interessante Muster zu finden. Diese können beispielsweise zur automatischen Empfehlung von Filmen genutzt werden oder helfen neue Zusammenhänge aufzudecken und Prozesse zu verstehen. Seit Beginn der Data Mining Forschung wächst die Größe der zu verarbeitenden Datensätze. Während Datensätze zunächst als statisch und vollständig gegeben angenommen wurden, generieren viele Anwendungen heute kontinuierliche und teilweise unendliche Datenströme.

Anytime-Algorithmen stellen eine Klasse von Algorithmen dar, welche sich besonders gut zum Einsatz auf Datenströmen eignet. Ihr Name rührt von ihrer Eigenschaft her, zu jeder Zeit ein Ergebnis liefern zu können. Die zur Verfügung stehende Zeit ist dem Algorithmus dabei nicht bekannt: er berechnet ein initiales Ergebnis und verbessert dieses solange zusätzliche Rechenzeit vorhanden ist. Wird der Algorithmus unterbrochen, so liefert er das beste Ergebnis zurück, welches bis zu diesem Zeitpunkt erzielt wurde.

In dieser Dissertation werden neue Anytime-Verfahren für die Bayes Klassifikation entwickelt, intensiv untersucht und evaluiert. Der erste Anytime-Algorithmus zum Clustern von Datenströmen wird vorgestellt und eine Anwendung für die Erkennung von Ausreißern wird diskutiert. Neben neuen Algorithmen werden zwei übergeordnete Verfahren entwickelt, die den Anwendungsbereich für Anytime-Algorithmen signifikant erweitern. Die in dieser Dissertation vorgestellten Ansätze und Resultate tragen zum Stand der Forschung im Bereich Anytime-Algorithmen und Data Mining auf Datenströmen bei.

Part I

Introduction

Chapter 1

The Need for Anytime Algorithms

The rapid development of computer and information technology fundamentally changed many processes in science, industry and daily life. Systems for collecting, storing and managing data evolved from primitive file processing systems to sophisticated and powerful database systems. The tremendous amounts of data soon exceeded the human ability of comprehension and thus called for advanced tools for analysis, compression and exploration. Knowledge discovery from data bases (KDD), or data mining, emerged as an interdisciplinary research field from the areas of data bases, machine learning, statistics, visualization and other areas. Many mining algorithms were invented that helped reducing data, automatically classifying new objects based on historical data or extracting rules and correlations that exhibit novel patterns and useful knowledge.

In early days of data mining research data sets were often considered static and given as a whole. This assumption allowed mining algorithms to perform random access on the data and to process objects multiple times during execution. With ever growing amounts of data the need for efficient processing yielded single pass algorithms and general solutions for fast access to large data sets were developed.

A data stream is a continuous, possibly endless sequence of data items that must be processed as they arrive. Many real world applications can be associated with data streams as large amounts of data must be processed

every day, hour, minute or even second. Examples include business/market data in companies, medical data in hospitals, statistical data in governmental institutions, experimental data in scientific laboratories, click streams in the world wide web, traffic/network data at web hosts or telecommunications companies, financial/stock market data in banking institutions, transaction/ customer profile data in e-commerce companies, etc.

Mining on data streams can basically perform the same tasks as mining on static data sets. Summarization or *clustering* are important tasks in stream mining, since they help to reduce the amount of data and can provide an overview of the data distribution. Other major tasks are stream *classification* and outlier detection on data streams. Figure 1.1 shows real world examples of data stream applications where such mining tasks are crucial. Sorting items on a conveyor belt is an application for stream classification (cf. top left of Figure 1.1). In the example the bottles correspond to the data items and the classifier must sort out the broken or dirty bottles while usable bottles can pass. Similar applications are continuous quality checks in production sites such as surface inspection in paper, fabric or foil production (cf. top row of Figure 1.1, middle and right).

The first example in the second row of Figure 1.1 shows an application for both classification and outlier detection. The image shows an offshore wind farm and is associated with remote monitoring of machinery in general. In these applications measurements such as temperature, pressure or frequency spectra are taken at regular intervals and sent to a remote monitoring center, where the data is analyzed. The analysis can facilitate classification of the current status, detection of abnormalities or prediction of future measurements for hazard warnings. The center image shows a buoy of a tsunami warning system, the image right to it illustrates a network of sensors spread on a glacier in Switzerland. In both cases environmental parameters are measured and sent to base stations where these measurements constitute a stream of data objects. While classification and outlier detection has priority in both applications, generating models and distributions of the incoming data can be of interest for researchers and decision makers.

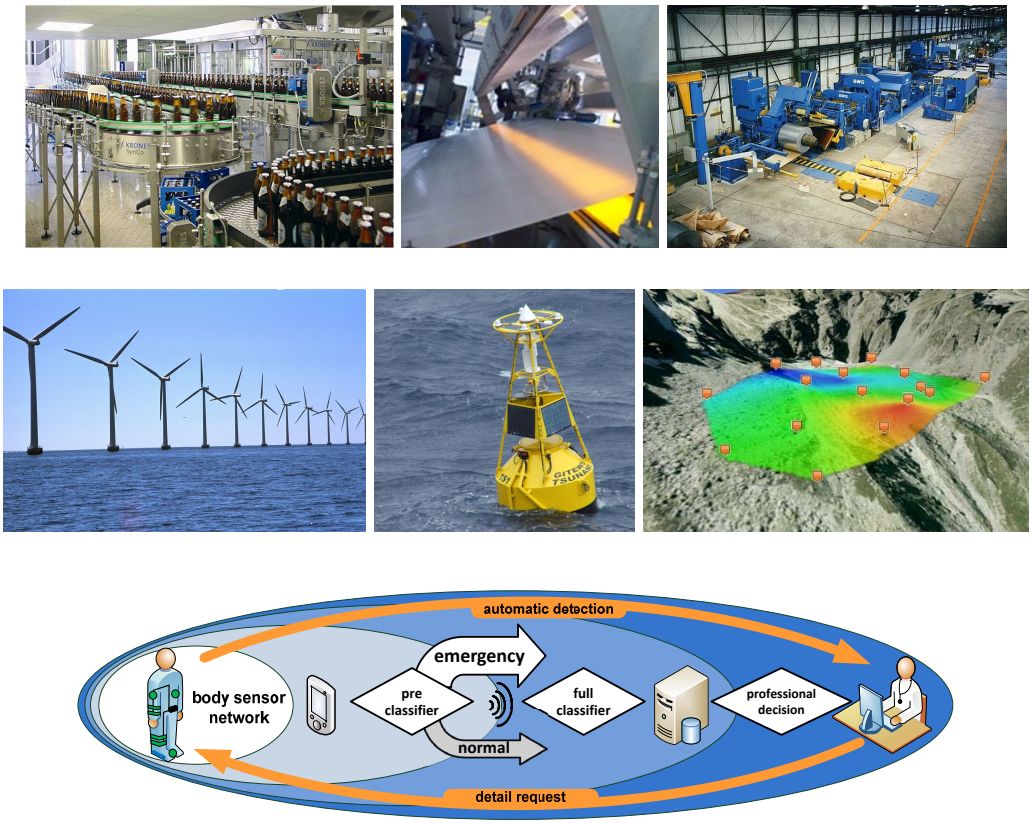


Figure 1.1: Examples for real data streams.

The bottom part of Figure 1.1 illustrates the HealthNet project that has been conducted at RWTH Aachen University within the UMIC research cluster (cf. Chapter 8 for details). A central part in the HealthNet project is a body sensor network that measures vital functions of patients or elderly people. Those measurements are transferred to a mobile device for a first analysis and then partially forwarded to a central server depending on the results of the local analysis. Again the above mentioned tasks of classifying, detecting, predicting and modeling such patient data are useful and important.

Summarizing the review of data streams in real world scenarios it becomes clear that basically everywhere where data is continuously measured, sent and analyzed, we find a potential application for stream data mining.

From the nature of a data stream and the properties of the system, on which the mining algorithm is executed, the following restrictions and consequential requirements emerge:

- endless stream – due to the fact that a data stream is possibly endless, the amount of data must be considered as infinite. As a consequence, algorithms are only allowed a single scan over the data. While some objects can be buffered to be processed again later on, nearly all items can be considered at most once. Moreover, the data must be processed in the order of its arrival; random access as in traditional data mining is missing in the streaming context.

- limited time – since data is continuously arriving, the time to process a single object is limited. Spending more time on an item by not processing other items (dropping, sampling) is prohibitive in many applications such as sorting or outlier detection. Therefore the available time is mostly dictated by the actual or average time between two consecutive stream items. To this end the algorithms must provide fast access and cheap functions to process incoming objects online.

- limited memory – the limitation of physical memory and the infinite amount of data naturally yield the need for a compact representation of historical data using effective data structures. Storing all data objects is in most cases practically infeasible.

- evolving data distribution – maintaining an up to date model of the data distribution is crucial for many applications and algorithms. However, in a data stream scenario the underlying model of the data distribution often changes over time; concepts may shift or disappear and novel concepts can emerge. To focus on recent data, algorithms must provide ways to update their models and also forget or weigh down older data. Besides maintaining a valid model, the detection of changes in the data distribution is a new task that emerged in stream data mining.

- noisy data – noise in data streams corresponds to improper data tuples that can result from faulty sensor readings, for example. It is important for a mining algorithm to be robust against noise and offer appropriate ways of noise handling.

- varying data rates – the majority of data streams do not exhibit the same data rate over the entire time of their existence. On the contrary, many applications produce data at massively changing rates, for example due to daytime or seasonal changes in transaction data. Even rather stable data streams show varying data rates over time such as changes in conveyor belt speed in production. The varying data rates imply varying time allowances, which are in many cases unforeseeable.

For some of the above aspects effective solutions have been proposed that are shared by many approaches. For limited time and limited space, established methods are available that are discussed in Chapter 3 along with common methods of dealing with evolving data distributions. Coping with single scan and missing random access is the core of any streaming algorithm. Also, the handling of noise is done in very individual ways (or even left out), since there is no clear definition of noise. Varying data rates and their implications are the focus in the remainder of this chapter.

The requirements for stream mining algorithms demand that items must be processed as they arrive. The available time for processing is naturally limited by the time between two consecutive items and may vary greatly in many applications. Referring back to the data stream applications from Figure 1.1 we take sensor networks as one example. Many sensor networks strive to reduce the amount of data that must be sent either to spare network bandwidth or energy resources, since sending data consumes much energy and often sensors are battery powered. Therefore, measurements are aggregated or only sent if they deviate significantly from the previous measurement. While this can yield unforeseeable inter-arrival times already for single sensors, it often results in data streams of massively changing data rates at base stations or monitoring centers. More moderate changes in stream rates can be expected at production sites and conveyor belt applications. However,

frequencies may differ as well on a daily, weekly or monthly basis. Finally, streams of business data or network traffic data exhibit high volatility for daytime or seasonal reasons.

As discussed previously, sampling or dropping of data items is prohibitive for many applications such as sorting or outlier detection. Since in varying data streams the duration of a burst or the number of objects that arrive within a certain time window is generally not bounded, simply buffering objects until the stream slows down is an option, neither. If the buffer size is exceeded, objects are lost. Moreover, buffering objects can largely delay the time of their processing and hence consecutive actions or reactions may not be initiated on time.

To process each item as it arrives traditional algorithms must base their computational model on the worst case assumption. More precisely, to be able to keep up even with the highest expected stream rate, the processing time for a single item must be strongly related to the corresponding smallest inter-arrival time. While parallel processing can speed up algorithms, it cannot break the dependency on the smallest time allowance. To finish the computation within a specific time budget, an algorithm can be restricted and tailored to the contracted budget. Clearly this yields large amounts of idle times, since the algorithm would finish his computation in the same time even if the available time is larger by orders of magnitude. While this is a drastic restriction for highly volatile streams, even for rather constant streams with smaller seasonal changes as in production sites, models must be built according to the minimal time allowance and, hence, the idle times add up and can become massive.

Optimally, an algorithm should be able to process an object in very short time and use any additional computation time to improve its outcome as long as permitted by the application. Consequently, idle times would be reduced or even avoided, the overall output of the algorithm would be improved, and the need to tailor algorithms to an expected or minimal time budget becomes obsolete. The idea of being able to provide a result regardless of the amount of available computation time lead to the development of anytime algorithms.

Definition 1.1 ***Anytime algorithm.*** *For a given input an anytime algorithm can provide a first result after a very short initialization time and it uses additional time to improve its result. The algorithm is interruptible after any time and will deliver the best result available at the point of interruption.*

Anytime algorithms have first been discussed in the artificial intelligence community by Thomas Dean and Mark Boddy in [DB88] and have thereafter been an active field of research [Bod91, Zil96, GZ96]. Recent work includes an anytime A* algorithm [LGT03] and anytime algorithms for graph search [LFG+08]. Related concepts include anyspace [RC05, YWKMN09] and anycost algorithms [EM11].

Anytime algorithms differ from online algorithms and stream algorithms. Online algorithms refer to computing with incomplete information; the input is given one piece at a time [BEY98]. In addition to that, stream algorithms consider limited resources explicitly, for example a fixed maximal amount of time or memory. In contrast to anytime algorithms, neither of the two categories requires the algorithm to be interruptible at any time.

Not every application needs an anytime algorithm. In many applications there may always be enough time to compute even the most detailed model. However, it is unquestioned that in many applications the available data largely increases in both size and complexity. This implies on the one hand that models become more complex, too, leading to higher computation times. On the other hand the increasing amount of data to process yields smaller time allowances per object. Multi-core processors and parallelization are not a one-fits-all solution to increasing complexity and decreasing time allowance. Large enterprises, especially web-scale endeavors, employ newest hardware and large scale server installations. While this may solve the case for some, others cannot afford giant hardware resources either due to economic reasons or contextual restrictions. Embedded systems, sensor networks, and mobile devices, for example, naturally own only very limited resources.

The anytime principle is applicable to many mining tasks and other algorithms. Classification is used as an example for the remainder of this section.

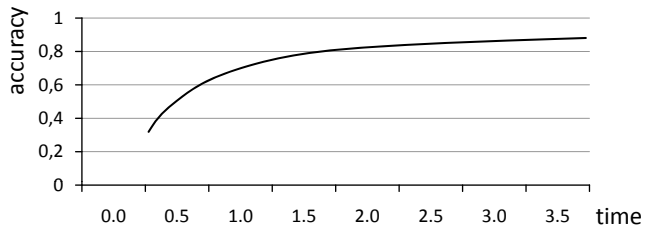


Figure 1.2: Anytime classification accuracy curve.

An anytime classifier can provide a first classification result for a given object after a short initialization. If it is not interrupted by the application, the algorithm continues processing the object to improve its classification decision; the accuracy of the result increases with the available computation time. Figure 1.2 shows the curve of a typical evaluation result for an anytime classifier (if the algorithm works correctly): very small time allowances typically lead to medium classification accuracy, with greater time allowances the accuracy increases asymptotically towards its maximum.

The evaluation principle of anytime algorithms is somehow opposite to evaluating traditional algorithms. Traditional algorithms are often compared in terms of their efficiency, stating how many resources (most often time) they need to achieve a certain goal. This corresponds to the minimum principle, or principle of thrift, known from economy, which strives to reach a certain output with the least possible input. Anytime algorithms rather correspond to the maximum principle, or principle of yield, which strives to yield the highest possible output given a certain input. Evaluating an anytime classifier hence asks the question ”how accurate is the classifier for a given amount of time?”. Taking the varying time allowances into account, not only the single time accuracy value pairs are of interest, but also the accuracy increase between two different time allowances as well as the average accuracy over a given distribution of time allowances. Details on these evaluation methods are provided in Chapter 4.

Anytime algorithms are obviously the best choice for data streams with varying data rates. Also on constant data streams, where every inter-arrival interval is always strictly the same, anytime algorithms can be beneficial and clearly outperform traditional budget algorithms. Consider the first applica-

tion from Figure 1.1, bottles on a conveyor belt in a brewery, for example. At some point the bottles pass a camera or the like, where features are extracted that are used in a classifier to decide whether the bottle must be sorted out or not. If the time between two bottles is known in advance, lets say 50 milliseconds, then a budget algorithm can be designed that delivers a classification result within that budget of 50 milliseconds. If the budget would be higher the resulting accuracy of the classifier is likely to be better, because it has more time and can process a more detailed model. However, the budget algorithm spends the same amount of time on each item, i.e. on each bottle. In reality there are bottles for which the decision is very clear even after a very short time, as in the case of a bottle that is broken or clearly wasted. At this point an anytime algorithm can be used to not spend any more time on these certain decisions and to take advantage of the gained extra time for other items, where a clear decision requires a more detailed model. This way, the overall performance of an anytime algorithm (the accuracy in this example) exceeds that of a traditional budget algorithm even on a constant data stream. In Chapter 9 different approaches to harness the strength of anytime algorithms on constant streams are proposed along with details and experimental analyses.

The abundance of data streams and applications and the superiority and usefulness of anytime algorithms on both varying and constant streams motivated the research that is presented in this thesis. Novel methods for anytime classification are developed and the first anytime algorithm for clustering on data streams is proposed. The outline of the thesis is provided in the next section.

1.1 Thesis structure

This thesis consist of five parts: I Introduction, II Anytime Stream Classification, III Anytime Stream Clustering, IV Summary and Outlook, and the appendices in Part V.

The introduction contains the motivation, the background, and the main concepts for stream data mining and anytime algorithms. We have seen that data mining algorithms are important in many applications and that streams are ubiquitous in all areas of our daily life. The special requirements of stream mining algorithms were discussed and anytime algorithms resulted as most flexible and the best choice for both varying and constant streams. In the remainder of the introduction the KDD process and general mining tasks are reviewed in Chapter 2 and specific tools and algorithms for stream data mining are discussed in Chapter 3.

In part II the focus is on anytime classification on data streams. Novel approaches for anytime stream classification are presented in Chapters 4 – 7 and an application of the proposed anytime classifier is shown in Chapter 8. Two concepts for using anytime algorithms on constant streams are proposed in Chapter 9, future work in the area of anytime stream classification is discussed in Chapter 10.

In Part III anytime clustering on data streams is introduced. Novel approaches for anytime stream clustering are proposed in Chapters 11 – 13, an application of the proposed technique for anytime outlier detection is shown in Chapter 14. Chapter 15 deals with the evaluation of stream clustering algorithms presenting a software framework for stream mining and a novel concept for the evaluation of clusterings on evolving data streams. Future work in the area of anytime stream clustering is discussed in Chapter 16.

Part IV summarizes the thesis and provides a general outlook for future research in the area of anytime algorithms and stream data mining.

The appendices in part V provide the bibliographic references as well as additional information on the author and his contributions in this thesis in the form of a statement of originality, a list of publications, and a curriculum vitae.

Chapter 2

Knowledge Discovery from Data

This chapter contains background information on the KDD process and introductions to basic data mining tasks, algorithms and challenges. Readers familiar with data mining may want to skip this chapter and proceed with Chapter 3, where more specific techniques for stream data mining are discussed.

2.1 The KDD process and data mining tasks

Knowledge discovery from data is an iterative process that contains multiple steps which are illustrated in Figure 2.1. Before the actual data mining is performed the desired data must be extracted and validated. This requires that a thorough preprocessing precedes the mining step.

Data cleaning refers to the removal of noise and inconsistencies in the raw data. These can be missing values or values that are outside the allowed range and can occur among others due to faulty measurements, human errors during data acquisition or deliberate manipulation. *Integration* of data refers to the combination of data from multiple sources, which can be different companies, different users or different time spans. Challenges in data integration include differences in naming conventions, missing attributes or a different schema in general. *Data selection* is the restriction to relevant data; only those tuples and attributes are retrieved that are inter-

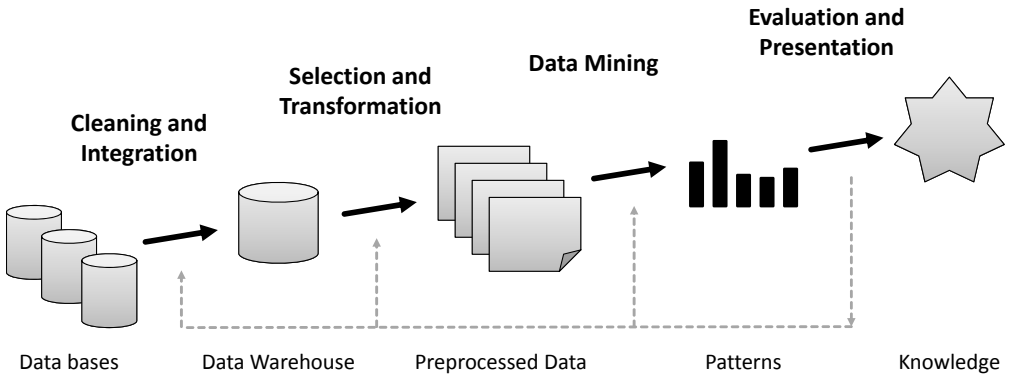


Figure 2.1: From data to knowledge: Illustration of the KDD process as described in [HK01] and others.

esting and necessary for the impending task. *Transformation* is the last step in the preprocessing where the data is altered to meet the requirements of the following mining algorithm. Normalization or unification of value ranges can be performed in this step as well as aggregation, binarization or other transformations.

The preprocessing is followed by the actual *data mining* step, where algorithms are applied that extract patterns from the data. These pattern are for example groups of data objects described by representatives, models or rules that are found by the algorithm. While representatives can be used for data reduction, models can identify *abnormal* objects that strongly deviate from the model, and rules can be employed to classify previously unknown data objects. Examples for data mining tasks are provided below.

The *evaluation* step is as crucial as the mining step and is often considered as a part of the data mining algorithm itself. The goal in this step is the identification of the truly interesting patterns; evaluation is similar to a filter or pruning step that reduces the amount of found patterns to those that are most important according to some interestingness measure. *Presentation* of the found patterns is most often done by visualization and knowledge representation techniques. It supports the user in exploring and interpreting the results and can provide hints for further mining and analysis steps. Finally, the found patterns can be integrated into the data repositories and used in further iterations of the KDD process.

Data mining is a central step in the KDD process. In the following the most recognized mining tasks are introduced along with brief explanations and examples. Since two tasks, namely classification (Part II) and clustering (Part III), constitute core parts of this thesis, a formal definition is provided for these as well. In the remainder of this thesis an object or point is referred to as a d-dimensional tuple or vector, where each dimension is called an attribute or a feature of the object. In general objects can be from an arbitrary metric space; if not mentioned differently, the d-dimensional Euclidean vector space $\mathbb{R}^d$ is assumed as the domain for all objects and the Euclidean distance between two points x and y is denoted as $\mathbf{d}(x, y)$.

Classification. Classification describes the process of assigning a previously unknown object to a category or *class* based on its features. To this end a classifier builds a so called model from the given training data in the training phase and uses this model to process new objects in the testing phase.

Definition 2.1 ***Classifier.*** *For an input space $\mathcal{Q}$ and a set of class labels $\mathcal{L} = \{l_1, \ldots, l_{|\mathcal{L}|}\}$ a classifier trains a set of internal model parameters Θ based on a set of training data $\mathcal{T} \subseteq \mathcal{Q} \times \mathcal{L}$ (and possibly external parameters). A classifier $\mathcal{C}$ is associated with a function $\boldsymbol{f}_{\mathcal{C}}(\Theta, q)$ that assigns an object $q \in \mathcal{Q}$ to a class label based on its parameters Θ.*

One well known family of classifiers are decision trees [BFOS84, Qui93], the left part of Figure 2.2 illustrates an example. A node of a decision tree is associated with an attribute, the outgoing branches correspond to an attribute value. In the example the risk for accidents is determined based on the two attributes ”car type” and ”age”, possibly useful for an insurance company. While the root node branches according to a match in the categorical attribute ”car type”, the second node applies a threshold on the continuous attribute ”age”. A thirty year old person driving a sports car would be assigned a high risk according to this decision tree. The tree is the actual model of the classifier that is learned in the training phase. A way to improve the accuracy of a classification method is to build an ensemble of several classifiers. For a decision tree one could for example build several trees from

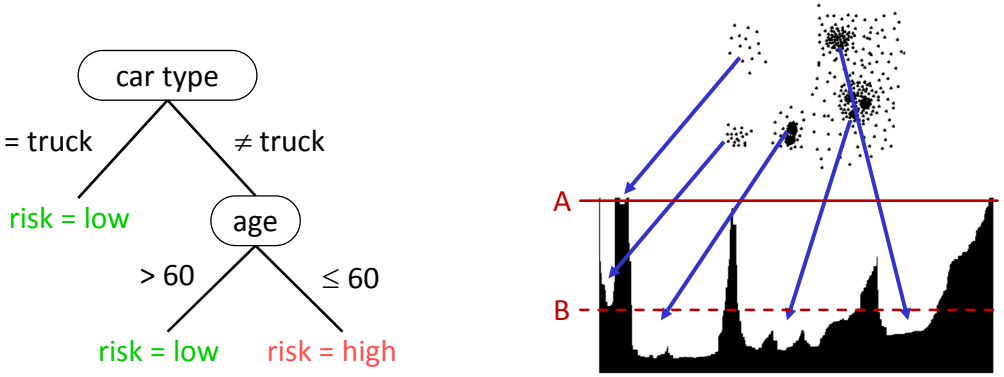


Figure 2.2: Left: decision tree classifier. Right: a result of the OPTICS clustering algorithm. (Examples taken form lecture slides corresp. to [HK01]).

different random samples of the training data. The classification decision can then be obtained by a majority vote on the results of all trained trees.

Prediction and regression are similar to classification. In regression the output of the algorithm is not from a limited set of class labels, but can be a numerical value. Prediction refers to finding the most probable future value based on a series of previous values or value tuples. Classification is also termed supervised learning, since the class labels of the training set are known in advance.

Clustering. Clustering refers to grouping of data objects, such that the objects within a group exhibit a high similarity and objects from different groups are dissimilar. Since in contrast to classification the labels of the objects are not known, clustering is also called unsupervised learning. A clustering algorithm takes a data set as an input and returns a set of groups called clusters.

Definition 2.2 ***Clustering.*** *Given a data set $\mathcal{O} \subseteq \mathcal{Q}$ from an input space $\mathcal{Q}$ a clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$ is a set of groups C_i called clusters. Each cluster C_i is a non-empty subset of objects from $\mathcal{O}$: $C_i \subseteq \mathcal{O}$ with $|C_i| > 1, \forall i = 1 \ldots k$.*

The most popular clustering methods are so called k-center clustering algorithms including k-means [Llo57], k-medoids [KR90] and their variants. They take the number of clusters k as an input parameter and try to find k representative centers such that a given objective is minimized. The objec-

tive is most often a variant of the sum over all squared distances from the objects to their closest representative. Since finding the optimal solution to this problem is NP-hard, clustering algorithms heuristically search a local optimum as their solution. Besides k-center clustering algorithms many other approaches have been proposed such as density-based clustering methods. One example is the OPTICS algorithm [ABKS99], the right part of Figure 2.2 shows a result of the algorithm. Each valley in the plot corresponds to a cluster in the data set, a horizontal line corresponds to a threshold value for the minimal cluster density. If A is chosen as a threshold, the two small groups on the left side are returned as two clusters and the remaining points are considered as a third cluster. Choosing B as a threshold results in three clusters as well (dense areas in the right group of points), but the objects on the left and the scattered points on the right are not included in a cluster.

Outlier analysis. Objects that deviate significantly from the rest of the data set are called outliers. There is no clear definition and, consequently, different approaches can find different outliers in the same data set. In the OPTICS algorithm described above, the objects that were not included in any cluster are considered outliers (cf. threshold B), since their region of the data space does not exhibit a sufficient density. Other algorithms determine outliers using distance-based [KNT00], angle-based [KSZ08] or statistical approaches [BL94].

Frequent patterns/Association rules. Market basket analysis is a common application for association rule mining or frequent pattern mining where sets of items are searched that frequently occur together. In the example they correspond to products that are often bought together. The goal is to find rules of the form "If customer X buys milk and bread he will also buy cheese (with a certain probability)". Technically, an association rule is of the form "head $\rightarrow$ body (support, confidence)" where body follows as a consequence from head, support is the number of times that the pattern was observed and confidence is the relative frequency of head compared to the support of the rule. Efficient methods to compute all association rules for given support and confidence values have been proposed in the literature [AIS93, AS94, HPY00].

Concepts description (characterization, generalization, discrimination). Characterization is a summarization of the general characteristics or features of a group of objects such as age, job and income of people that regularly go on a cruise. Generalization is an automated process that groups objects based on their characteristics. An example is attribute oriented induction [CCH91]. Discrimination finds characteristics that separate two groups of objects (contrasting classes), for example those customers that frequently buy product X and those that rarely buy it.

In this thesis new methods are mainly developed in the areas of classification and clustering. A more detailed review of existing approaches in those areas can be found in Sections 2.2 and 3.2 for classification and Sections 2.3 and 3.3 for clustering, where the corresponding section in Chapter 3 contains the stream specific methods. As an application of a proposed technique outlier detection is discussed in Chapter 14. The corresponding related work on outlier detection is discussed in that chapter. For a more detailed general introduction to data mining please refer to [HK06]. In the following section challenges in data mining are presented that arise due to certain properties of the data or requirements of the application. A broader introduction to those topics can be found in [KHY+09].

2.1.1 Challenges in data mining

Many mining algorithms rely on a notion of similarity between objects or representatives: clustering algorithms group objects based on similarity, a classification algorithm can assign labels based on similarities, etc. Efficient mining algorithms therefore often require methods that allow for fast similarity search in large data sets. Similarity is most often assessed by the use of a distance measure such as the Euclidean distance. To speed up similarity search a plenitude of methods and data structures have been proposed that facilitate fast access. In hierarchical index structures directory information is organized in a tree structure. When executing a similarity search query, these methods try to access only very small parts of the data by steering the search using the tree structure. Examples for index structures include the

popular B-trees [Com79], R-trees [Gut84, BKSS90], M-trees [CPZ97] and KD-trees [Ben75], as well as specialized methods such as the RI-tree [KPS00] or the TS-tree [AKAS08]. Besides hierarchical index structures many other approaches for fast similarity search have been proposed including bitmap indexes [CI98] or hashing techniques [DIIM04].

A major challenge are very high dimensional spaces. In high dimensional spaces distances lose their expressive power, since all distances tend to be similar. This problem is known as the curse of dimensionality, which affects similarity search and hence also mining algorithms. Specialized index structures for high dimensional spaces have been proposed such as X-trees [BKK96] and TV-trees [LJF94]. The VA-file [WSB98] constitutes a filter-and-refine approach that first computes a set of candidates using a lower dimensional representation of the data and refines the set in the full space in a second step. Other lines of research focus on approximate similarity search in high dimensional spaces [HS05] or try to avoid using distances and determine the association between objects by the means of shared-neighbor information [Hou08, HKK+10]. To circumvent the problems resulting from the similarity of distances in high dimensional spaces, subspace clustering algorithms search for solutions in different subsets of the attributes [AGGR98, APW+99, AY00, CFZ99, AKMS08]. Since the number of different subspaces is exponential in the dimensionality of the original space, these algorithm are faced with a huge combinatorial complexity.

Time poses different challenges on data mining algorithms, ranging from volatile data bases to dynamic and endless data streams. Time aspects, especially requirements and solutions for stream data mining, have been discussed in Chapter 1 and are the main topic throughout the thesis. A different notion of time in data sets and mining algorithms is introduced by time series data. Time series denote sequences of values over time which are available entirely before processing them in an algorithm. Hence, in contrast to dynamic data streams, time series constitute a static type of data. Mining on time series data includes searching for reoccurring or unusual patterns to discover trends or conspicuous behavior [AS95, HDY99, KGI+11].

A further challenge lies in the structure of the data. New technologies and new phenomena yielded complex data types such as multimedia data [Sub98], text data [SM84], graph data as in social networks [New03], gene sequences and molecular structures [BB01] and other domain-specific data types. For multimedia data even a single image is often represented by a large number of features describing color, texture or salient points in the image [HLZ02]. Specialized distance measures have been proposed that are claimed to reflect the human perception of similarity [SK97, RTG00]. Mining video data has many applications such as video copy detection, but remains a major challenge in data mining research [CZ06, AKS10]. Other complex data types pose their own difficulties but also offer new questions and tasks. From there many new directions in data mining research emerged such as graph mining or text and semantic mining [WM03, WIZD04, WF94].

Although pattern evaluation has always been a step in the KDD process (cf. Figure 2.1), it got new attention with the advent of complex methods and data types. One example is the exploration of subspace clustering results [MAK+08], where the number of found clusters can sometimes be larger than the actual size of the data set. This is due to the exponential number of different subspaces in which clusters can be found. More precisely, since generally objects can be in more than one cluster, the found clusters often largely overlap with respect to the contained objects. To this end measures for redundancy and interestingness have been proposed and included into algorithms either during the cluster search or as a post processing component [MAG+09a, AKMS08]. Similar issues and solutions apply for other mining tasks such as frequent item set mining [BEX02].

Finally there is a list of other issues that reveal limitations of mining algorithms and open up new research directions. Examples are privacy and security issues that either restrict the access to data or the validity of results [CM96, VBF+04]. Parallel and distributed data mining algorithms try to meet the requirements of growing data repositories by efficient strategies to share data and computation across large networks of servers and clients [PCY95, AS96, PHBB09]. The popularity of sensor networks, global positioning systems (GPS), cellular phones, other mobile devices and RFID tech-

nology yielded vast amounts of moving object data that calls for effective methods for analysis and knowledge extraction [DeC97, PZZ+07, CBB08]. The internet can be seen as a synonym for many of the challenges mentioned above. It represents graph data, text data and heterogeneous data, poses privacy and security issues, is massively distributed and parallel and has obviously a tremendous size.

2.2 Classification

In this section the main concepts and approaches proposed for classification on static data sets are reviewed. Stream classification algorithms are discussed in Section 3.2. Before going into detail on the individual approaches a more formal description of a classifier based on Definition 2.1 is provided.

Given an input space $\mathcal{Q}$ of dimensionality d and a set of labels $\mathcal{L} = \{l_1, \ldots, l_{|\mathcal{L}|}\}$, the extended input space of labeled objects is denoted as $\mathcal{Q}^+ = \mathcal{Q} \times \mathcal{L}$. The power set $\mathfrak{P}(\mathcal{Q})$ of $\mathcal{Q}$ contains all possible subsets of $\mathcal{Q}$, $\mathfrak{P}(\mathcal{Q}^+)$ analogously for $\mathcal{Q}^+$. The training set for a classifier can then be written as $\mathcal{T} \in \mathfrak{P}(\mathcal{Q}^+)$ and a test object for classification as $q \in \mathcal{Q}$. $\mathcal{T}_l = \{o \in \mathcal{T} | o = (o_1, \ldots, o_d, l)\}$ denotes the set of objects from $\mathcal{T}$ with label l.

Any classifier $\mathcal{C}$ can be described by a model M which is in turn defined as set of parameters $M = \{\psi_1, \ldots, \psi_{|M|}\}$. The domains of the parameters are denoted as $Dom(\psi)$, and $Dom(M) := Dom(\psi_1) \times \ldots \times Dom(\psi_{|M|})$ is the cross product of all parameter domains. A configuration or instantiation of the model M is a set of values $\Theta \in Dom(M)$ for its model parameters: $\Theta = (\theta_1, \ldots, \theta_{|M|})$ with $\theta_i \in Dom(\psi_i) \;\; \forall i = 1 \ldots |M|$.

To build a classifier, any approach must first *select* a model and second choose, or *optimize*, its parameter values. To this end it can consider the training set $\mathcal{T}$ and possibly a set of external parameters $\Phi = \{\phi_1, \ldots, \phi_{|\Phi|}\}$. Similar to the above the parameter domains are denoted as $Dom(\phi)$ and $Dom(\Phi) := Dom(\phi_1) \times \ldots \times Dom(\phi_{|\Phi|})$. A set of external parameter values is called $\Xi = \{\xi_1, \ldots, \xi_{|\Xi|}\}$ with $\xi_j \in Dom(\phi_j) \;\; \forall j = 1 \ldots |\Phi|$. The selected model is

$$M = \eta_{\mathcal{C}}(\Xi, \mathcal{T}) \tag{2.1}$$

where $\eta : Dom(\Phi) \times \mathfrak{P}(\mathcal{Q}) \mapsto \mathcal{M}$ is a model selection function that selects a model M from the model space $\mathcal{M}$ based on the training set and external parameters. The values for the model parameters are chosen in a second step as

$$\Theta = \pi_{\mathcal{C}}(M, \mathcal{T}) \tag{2.2}$$

where $\pi : \mathcal{M} \times \mathfrak{P}(\mathcal{Q}) \mapsto Dom(\mathcal{M})$ is a parameter optimization that is based on the selected model and the training set and $Dom(\mathcal{M}) = \bigcup_{M \in \mathcal{M}} Dom(M)$. Finally, the label of a test object $q \in \mathcal{Q}$ is determined by the classifier as

$$l = \mathbf{f}_{\mathcal{C}}(\Theta, q) \tag{2.3}$$

using a decision function $\mathbf{f}_{\mathcal{C}} : Dom(M) \times \mathcal{Q} \mapsto \mathcal{L}$ that assigns a label based on the model parameters Θ. Thus, without loss of generality, the general task of a classifier $\mathcal{C}$ is to determine

$$l = \mathbf{f}_{\mathcal{C}}(\pi_{\mathcal{C}}(\eta_{\mathcal{C}}(\Xi, \mathcal{T}), \mathcal{T}), q). \tag{2.4}$$

Finding the set of parameter values $\Theta = \pi_{\mathcal{C}}(\eta_{\mathcal{C}}(\Xi, \mathcal{T}), \mathcal{T})$ is usually referred to as the *training* of a classifier, also called *learning* or *supervised learning*, since the class labels of the training set are known. Applying the decision function on a set of test objects is often termed *testing* of a classifier.

The performance of a classifier is mostly measured in terms of its classification error err or classification accuracy $acc = 1 - err$ corresponding to the proportion of correctly classified objects. Examples for further analysis and performance measures like the confusion matrix, sensitivity, precision or the ROC curve can be found in [HK06]. To assess the performance of a classifier on some given data set $D \in \mathfrak{P}(\mathcal{Q}^+)$, an m-fold cross validation splits D into m equally large parts (folds) and uses successively one part for testing and the remaining parts for training. A different approach called bootstrapping uses m subsets D_i of size $|D_i| = |D|$ sampled randomly from D with replacement and uses D_i for training and $D \setminus D_i$ for testing for $i = 1 \ldots m$.

For the plethora of classification algorithms, different approaches have been suggested to categorize the proposed methods [HK01]. One approach

distinguishes lazy learners and eager learners, while the latter category is larger by far. Lazy learners spend little or no computation time on model selection and parameter optimization, but defer the work to the decision phase. Eager learners on the other hand exhibit more time consuming training procedures and are in turn often more efficient in the decision phase. A different categorization separates generative and discriminative methods. The objective in training generative classifiers is to build a model that best describes the classes $\mathcal{L}$ as given by $\mathcal{T}$, as if the model was *generated* by $\mathcal{T}$. This is opposed to discriminative methods, which seek to best separate, or *discriminate*, the different classes. Neither of these categorizations is followed here, but the terminology is referred to when describing the single methods. Among the presented classification approaches Bayesian classification is described in more detail, since it will be heavily used throughout the thesis. Support vector machines (SVM) and decision trees receive more attention that the remaining approaches, since both relate to parts of the methods presented in Part II. For SVMs a simple transformation to Gaussian mixture models is discussed and decision trees connect to the proposed methods by constituting a hierarchical approach.

Bayesian classification

Bayesian classification constitutes a statistical approach in which an object $q \in \mathcal{Q}$ is assigned to a class label $l \in \mathcal{L}$ based on membership probabilities. It is based on the Bayes theorem and one of the most frequently used methods for classification. The Bayes theorem for two events A and B is

$$P(A|B)P(B) = P(B|A)P(A), \tag{2.5}$$

where $P(A)$ is the prior or a priori probability of A and $P(A|B)$ is the posterior probability of A conditioned on B, also called the conditional probability of A given B. In the context of a classifier, given a set of labels $\mathcal{L} = \{l_1, \ldots, l_{|\mathcal{L}|}\}$ and an object $o \in \mathcal{Q}$, $P(l_i)$ and $P(o)$ are the prior probabilities for the labels and the object, respectively. $P(l_i|o)$ is the posterior probability of label l_i given object o and $P(o|l_i)$ is called the conditional prob-

ability of o given label l_i. Using Equation 2.5 the Bayes classifier assigns to an object o the label $\hat{l}$ that yields the maximal posterior probability:

$$\hat{l} = \mathbf{f}_{Bayes}(\Theta, o) = \arg\max_{l \in \mathcal{L}} \{P(l|o)\} = \arg\max_{l \in \mathcal{L}} \left\{ \frac{P(o|l)P(l)}{P(o)} \right\} \tag{2.6}$$

The probabilities on the right hand side of Equation 2.6 can be estimated from $\mathcal{T}$ as follows. The class priors are estimated as the relative frequency of the labels in $\mathcal{T}$:

$$P(l) = |\{o \in \mathcal{T} | o = (o_1, \ldots, o_d, l_o) \wedge l_o = l\}| \, / |\mathcal{T}| \tag{2.7}$$

The prior for the object is computed as

$$P(o) = \sum_{l \in \mathcal{L}} P(o|l)P(l). \tag{2.8}$$

However, as in the computation of $\hat{l}$ the object o is not varied in the decision set $\{P(l|o)\}$, the term $P(o)$ can be left out in equation 2.6, since it merely normalizes the probabilities but does not affect the decision (simultaneous classification of several objects is discussed in Chapter 9). The way of estimating the class conditional probability $P(o|l)$ defines different kinds of Bayesian classifiers. The approach known as naïve Bayes assumes class conditional independence of all dimensions in $\mathcal{Q}$, which allows to calculate the posterior probability for o as

$$P(o|l) = \prod_{i=1}^{d} P(o_i|l). \tag{2.9}$$

The class conditional probabilities $P(o_i|l)$ for the single dimensions can again be easily computed from $\mathcal{T}$ by counting. Estimating $P(o_i|l)$ requires different approaches for categorical and continuous attributes.

For a categorical attribute A_i with possible values $a_{i1}, \ldots, a_{iu}$ the probability is looked up in so called *conditional probability tables*, in which it is

precomputed for every possible value a_{ij} as

$$P(a_{ij}|l) = |\{o \in \mathcal{T} | o = (o_1, \ldots, o_d, l_o) \wedge l_o = l \wedge o_i = a_{ij}\}| \,/|\mathcal{T}_l| \tag{2.10}$$

Dropping the assumption of the naïve Bayes allows to introduce dependencies between the attributes by defining the probability of attribute A having value a_i conditioned on attribute B having value b_j. For categorical attributes the dependencies and the resulting probability distributions can be described by Bayesian networks, also termed belief networks or Bayesian belief networks. A Bayesian network $\mathcal{B} = \langle G, \Theta \rangle$ is defined by a directed acyclic graph G and a set of parameter values Θ. $G = (V, E)$ contains one vertex for each attribute and one vertex for $\mathcal{L}$. An edge $(v_1, v_2) \in E$ signifies that v_2 is conditionally dependent on v_1; there are exactly d edges in the Bayesian network for the naïve Bayes, one from $\mathcal{L}$ to each dimension (cf. Figure 2.3 left). For a vertex $v \in V$, let $pa(v)$ denote the parents of v in G. Then the joint probability from Equation 2.9 for a Bayesian network classifier is

$$P(o|l) = \prod_{i=1}^{d} P(o_i|l, \{o_j|v_j \in pa(v_i)\}). \tag{2.11}$$

Θ in $\mathcal{B}$ comprises conditional probability tables that store a probability per attribute v_i for each combination of values from v_i, $pa(v_i)$ and $\mathcal{L}$. Considering the example in the right part of Figure 2.3 lets assume $|\mathcal{L}| = 4$ and $|A_i| = 10$ $\forall i = 1 \ldots d$, then each attribute can have ten different values. This yields

$$|\Theta| = \underbrace{4}_{P(l)} + \underbrace{4 \cdot 10^2}_{A_1} + \underbrace{4 \cdot 10 \cdot 5}_{A_2,\ldots,A_6} + \underbrace{4 \cdot 10^3}_{A_7} = 4604 \tag{2.12}$$

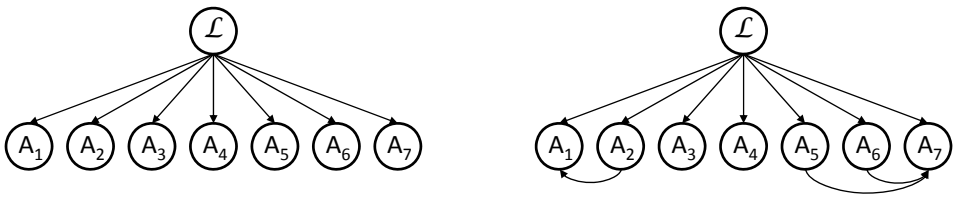


Figure 2.3: Examples for Bayesian networks for the naïve Bayes (left) and with additional dependencies (right).

which is opposed to $|\Theta| = 4 + 4 \cdot 10 \cdot 7 = 284$ for the naïve Bayes on the same example (cf. Figure 2.3 left). Besides higher space demands, the large number of parameters can frequently lead to *zero* probabilities, since certain value combinations do not occur in $\mathcal{T}$. These zero probabilities can cancel the effect of all other posterior probabilities; if in the above example six attributes would yield a high probability, a zero probability in the seventh attribute would spoil the output. To avoid this, a Laplace correction can be applied that initially sets each count in the probability tables to one and takes this into consideration during the normalization.

The model selection η (cf. Equation 2.1) for a Bayesian network classifier refers to the determination of the network topology (defining the edges in G). Proposed solutions are for example based on maximal correlation coefficients [Edw00, SS05] or use hill climbing methods to iteratively add the most promising edges [KP02]. The parameter optimization π (cf. Equation 2.2) for Bayesian networks as introduced above can simply compute the probabilities in a generative way from $\mathcal{T}$ (with or without Laplace correction) or can iteratively perform cross validation on $\mathcal{T}$ to find parameter values that lead to the best discrimination of the classes in $\mathcal{L}$. For Bayesian networks with hidden variables (not discussed in this thesis) there is an abundance of literature for network inference, introductions and examples can be found [RN95] and [Jen96].

For continuous attributes, the probabilities $P(o|l)$ and $P(o)$ in Equation 2.6 are substituted by density estimation functions denoted as $p(o|l)$ and $p(o)$. Usually the class conditional density $p(o_i|l)$ for a given object in dimension i is computed using a parameterized density function $g(o_i, \Theta_l)$ assuming a certain distribution of the attribute values. The parameter values Θ_l for the distributions of label l are derived from $\mathcal{T}$. The most popular example is the Gaussian distribution or normal distribution.

Definition 2.3 *The Gaussian distribution $\mathcal{N}(\mu_i, \Sigma_i)$ for a random variable X with mean value μ and variance σ^2 is described by the probability density function*

$$\boldsymbol{g}(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

For a set $\{X_1, \dots X_d\}$ of d random variables the joint probability distribution is

$$\boldsymbol{g}(\vec{x}, \vec{\mu}, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}}\sqrt{|\Sigma|}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu})\Sigma^{-1}(\vec{x}-\vec{\mu})^T}$$

where $\vec{x} \in \mathbb{R}^d$ is a d-dimensional vector, $\vec{\mu} = (\mu_1, \dots, \mu_d)$ is the vector containing the mean values of $X_1, \dots X_d$, Σ is the corresponding covariance matrix, $|\Sigma|$ its determinant and Σ^{-1} its inverse.

For convenience $\mathbf{g}(x, \mu, \Sigma)$ is used instead of $\mathbf{g}(\vec{x}, \vec{\mu}, \Sigma)$ for the d-dimensional case in the following. The joint probability distribution for continuous attributes naturally includes dependencies between all attributes via the covariance matrix Σ. For the naïve Bayes using Gaussian distributions, the covariance matrix constitutes a diagonal matrix $\Sigma = diag(\sigma_1^2, \dots, \sigma_d^2)$ and the posterior probability for an object o simplifies to

$$P(o|l) = \mathbf{g}(o, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}}\prod_{i=1}^{d}\sigma_i} e^{-\frac{1}{2}\sum_{i=1}^{d}\frac{(o_i-\mu_i)^2}{\sigma_i^2}} \tag{2.13}$$

Using a single Gaussian distribution to describe a set of objects is also called a unimodal model. A more flexible density estimation function is given if several Gaussian distributions are combined.

Definition 2.4 *A Gaussian mixture model (GMM) with k components is a weighted combination of k Gaussian distributions with parameters (μ_i, Σ_i), $i = 1 \dots k$ and is defined by the probability density function*

$$g_{GMM}(x, \Theta) = \sum_{i=1}^{k} w_i \cdot \boldsymbol{g}(x, \mu_i, \Sigma_i)$$

where $\Theta = \{\mu_1, \dots, \mu_k, \Sigma_1, \dots, \Sigma_k\, w_1, \dots, w_k\}$ is the set of all parameter values. w_i is the weight of the i-th Gaussian subject to $\sum_{i=1}^{k} w_i = 1$.

Besides unimodal models and mixtures of Gaussian, kernel density estimators constitute the most detailed model for density estimation based on training data. Kernel density estimation is detailed in Chapter 4, where kernels are employed in a novel anytime classification approach.

The model selection η for a Bayes classifier on continuous attributes can hence be viewed as selecting the type of density function, for example the number and type (naïve or not) of Gaussians. The parameter optimization π can again determine Θ from $\mathcal{T}$ in a generative way by using the EM clustering algorithm (cf. Chapter 6) or in a discriminative way by using max margin optimization for example (cf. Chapter 7). A more detailed introduction to Bayesian classification as well as further references can be found in [DHS01].

Support vector machines

Support vector machines (SVM) constitute discriminative classifiers that seek to find decision boundaries, called *separating hyperplanes*, which best separate the objects from different labels. A binary SVM considers objects from two different classes denoted as $+1$ and -1 ($l \in \{+1, -1\}$). The decision function of a binary SVM is

$$f_{SVM}(\Theta, q) = sign\left(\sum_{i=1}^{s} l_i \alpha_i K(x_i, q) - b\right), \tag{2.14}$$

where $\Theta = \{(x_i, \alpha_i) | i = 1 \ldots s\} \cup \{b, K(\cdot,\cdot)\}$ stores the bias b, the Kernel $K(\cdot,\cdot)$, the support vectors $x_i \in \mathcal{T}$ and their weight α_i. s is the number of support vectors, $l_i \in \{+1, -1\}$ is the label of x_i and examples for kernels $K(\cdot,\cdot)$ are

polynomial kernel of degree h:	$K(x_i, x_j) = (x_i \cdot x_j + 1)^h$
radial basis function (RBF) kernel:	$K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}$
sigmoid kernel:	$K(x_i, x_j) = \tanh(\kappa x_i \cdot x_j - \delta)$

The training of an SVM determines the weights α_i for all objects $o_i \in \mathcal{T}$. Those objects with corresponding $\alpha_i > 0$ are used as support vectors and are included in Θ. The actual training involves solving a constrained quadratic optimization problem, introductions and details to SVM can be found in [Vap95, Vap98, Bur98, Pla98].

In the case of $|\mathcal{L}| > 2$ multiple binary SVMs are combined. SVM classifiers for $|\mathcal{L}| > 2$ can for example train $|\mathcal{L}|$ binary SVMs, each separating the objects from one label against all other objects, or quadratically many binary SVMs, one for each pair of labels. The final decision can then be reached by a majority voting or interpreting the outputs of the single SVMs as probabilities.

On a side note, there is a simple transformation between an SVM using RBF Kernels and a Bayes classifier using Gaussian mixture models with equal variance σ for each Gaussian and each dimension, which yields the exact same decision boundaries for both classifiers [DHN10]. Using the notation from Equation 2.13 the transformation is

$$x_i = \mu_i \tag{2.15}$$

$$\alpha_i = P(l_i) \cdot w_i \cdot \frac{1}{(2\pi)^{\frac{d}{2}}\sigma^d} \tag{2.16}$$

$$\gamma_i = \frac{1}{2\sigma^2} \tag{2.17}$$

Transforming an SVM into GMMs, the bias b can be sufficiently well approximated by an additional Gaussian with arbitrary mean, very high variance and weight w proportional to b.

Decision tree classifier

An example of a simple binary decision tree was given in Figure 2.2, where the risk of an accident was determined based on the car type and the age of the driver. Generally, decision trees are not restricted to binary nodes but are allowed higher branching factors. The binary case is introduced below; the general case naturally follows from the description.

Definition 2.5 *A decision tree node $\nu = (Ai, a_{ij}, p_1, p_2)$ stores a splitting attribute A_i, a split value a_{ij} and two pointers p_1 and p_2 to the left and right subtree. A Leaf node ν is associated with a set of training objects $\mathcal{T}_{|\nu}$ and an inner node with the union of its subtree data sets $\mathcal{T}_{|\nu} = \mathcal{T}_{|\nu_1} \cup \mathcal{T}_{|\nu_2}$.*

For categorical attributes, a_{ij} splits $\mathcal{T}_{|\nu}$ into the two sets $\mathcal{T}_{|\nu_1} = \{o|o_i = a_{ij}\}$ and $\mathcal{T}_{|\nu_2} = \{o|o_i \neq a_{ij}\}$, for continuous attributes the two resulting sets are $\mathcal{T}_{|\nu_1} = \{o|o_i < a_{ij}\}$ and $\mathcal{T}_{|\nu_2} = \{o|o_i \geq a_{ij}\}$. For higher branching factors, additional values for the same attribute are stored in ν as well as additional pointers to the corresponding subtrees.

The training of a decision tree is often referred to as decision tree induction. Popular algorithms for decision tree induction are ID3 [Qui86], C4.5 [Qui93] and CART [BFOS84], which all follow a greedy top down approach to partition $\mathcal{T}$ recursively into smaller subsets. The algorithms seek to find the best split point, which is the best attribute A_i and attribute value a_{ij}, for the current node ν with respect to some objective function. A popular objective, which is also used in ID3, is the information gain

$$IG(\nu) = Ent(\nu) - \sum_{u=1}^{2} \frac{|\mathcal{T}_{|\nu_u}|}{|\mathcal{T}_{|\nu}|} Ent(\nu_u), \tag{2.18}$$

where $Ent(\nu)$ is the entropy of node ν defined as

$$Ent(\nu) = -\sum_{l \in \mathcal{L}} \frac{|\mathcal{T}_{l|\nu}|}{|\mathcal{T}_{|\nu}|} \log \left(\frac{|\mathcal{T}_{l|\nu}|}{|\mathcal{T}_{|\nu}|} \right). \tag{2.19}$$

$\mathcal{T}_{l|\nu}$ extends the notation of $\mathcal{T}_{|\nu}$ to $\mathcal{T}_l$. For higher branching factors the sum in Equation 2.18 considers all subtrees. Since this yields a bias towards nodes with a larger fanout, other methods such as C4.5 use the gain ratio as their objective, which overcomes the bias by normalizing the information gain with the potential information for the split [HK06]. The induction terminates if for example either all objects in $\mathcal{T}_{|\nu}$ have the same label or the same attribute values. There is a vast amount of literature on scaling decision tree induction to large data bases or improving decision trees, which is beyond the scope of this thesis.

During classification a query object $q \in \mathcal{Q}$ follows the path from the root of the decision tree to a leaf $\hat{\nu}$ according to its attribute values. The basic version of a decision tree then assigns a label based on a majority voting on $\mathcal{T}_{|\hat{\nu}}$. Advanced versions associate distributions with the leaves of the tree and apply for example a Bayes classifier using these distributions [Koh96].

Nearest neighbor classification

The nearest neighbor classifier (NN) [Das90, Sei09] is a simple classifier that belongs to the category of lazy learners. The idea is to assign to an object $q \in \mathcal{Q}$ the label $\hat{l} \in \mathcal{L}$ of the closest object $o \in \mathcal{T}$ with respect to a given distance function $dist(\cdot,\cdot) \in \Xi$:

$$f_{NN}(\Theta, q) = \hat{l} \quad \Leftrightarrow \quad \hat{o} = (\hat{o}_1, \ldots, \hat{o}_d, \hat{l}) \wedge \hat{o} = \arg\min_{o \in \mathcal{T}} \{dist(o, q)\}$$

Here, the training set forms the parameters: $\Theta = \mathcal{T}$. Ties can be broken assuming a canonical ordering of the labels. Variants of the nearest neighbor classifier use $k > 1$ objects, where $k \in \Xi$ is an external parameter. The decision based on the k resulting labels can be done using simple majority voting, weighting the frequencies with the class priors or weighting the influence of each object by $dist(o, q)^{-1}$.

Ensemble classifiers

Ensemble methods combine several classifiers and arrive at a decision based on the combined outputs. As an example, classifiers of different kinds can be trained on $\mathcal{T}$ and the label $l \in \mathcal{L}$ for an object $q \in \mathcal{Q}$ is determined via majority voting. *Bagging* (*b*ootstrap *agg*regation) [Bre96] is a popular ensemble method that trains k classifiers (usually of the same type) on k different bootstrap samples from $\mathcal{T}$. Another popular method is *Boosting* [FS97], which weights the results of the single classifiers with respect to their performance on $\mathcal{T}$. Moreover, in boosting the objects are weighted during training and the k classifiers are trained successively giving misclassified objects higher weights in subsequent trainings such that more attention is paid to them.

Neural networks

Neural networks are well known and very well researched. For a neural network classifier one must select a network topology and the type of neurons in a first step and learn or optimize the parameters of the network in a second

step. A frequently used training method for neural networks is backpropagation [RHW86, HB87, Jac88]. Using RBF-style neurons in a neural network yields the same decision hyperplane as an SVM with RBF kernels [HK06].

Besides the discussed classification algorithms there is a number of additional approaches including case-based reasoning and genetic algorithm as well as rough set and fuzzy set approaches; Please refer to [HK06] for a broader introduction. Each classification paradigm has its strengths and weaknesses. Bayesian classifiers are for example said to be less prone to overfitting since the employed probability distribution functions can provide an effective way of generalization. Clearly, there is no single solution that outperforms all other approaches on all domains and settings. Finding the best classifier for an actual application involves tuning and testing of various methods and is usually not addressed in the classification literature.

2.3 Clustering

Clustering tries to find groups of data, such that objects exhibit a high similarity within the groups and a low similarity between the groups (cf. Section 2.1). Research on clustering algorithms looks back at a long history. Since clustering is useful for many applications, approaches have been developed in different areas and from different perspectives. In this section the most prominent and frequently used clustering algorithms are reviewed.

The nature of clustering algorithms is often more heuristic compared to classification algorithms and their description appears therefore sometimes less formal. This holds true in a similar extent when it comes to the evaluation of clusterings that were found by an algorithm. A ground truth is hardly ever available; and even if, heuristics are needed to assess the quality of a found clustering. A wealth of measures can be found in the literature, some of which rely on a given ground truth clustering for comparison while other don't. The measures employed in Part III are commonly used and will be introduced where needed. The matter of clustering evaluation on evolving data streams is discussed in Chapter 15.

Definition 2.2 defines a clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$ as a set of groups C_i, each of which is a non-empty subset of objects from a given data set $\mathcal{O}$. This general definition allows both overlapping clusters (an object can be contained in more than one cluster) and unclustered objects, which are not contained in any cluster. Many clustering algorithms have been proposed and the approaches restrict these properties differently; some approaches allow only non-overlapping clusters, for example. An exhaustive survey is beyond the scope of this thesis. The following provides an overview of the main concepts and most important approaches as well as some details on those methods that are referred to in Section 3.3 and throughout the thesis.

k-center clustering

Algorithms that perform a k-center clustering on a given data set $\mathcal{O}$ start from an initial solution and try to optimize an objective by iterating the following two steps until a stopping criterion is met:

1. assign objects $o \in \mathcal{O}$ to a cluster w.r.t. the current cluster parameters

2. recompute the cluster parameters

The initial solution can either be chosen at random or given to the algorithm from the outside. Iterating the two steps can be stopped if between two iterations the improvement of an objective is less than a certain amount. The objective can for example be a similarity measure that is calculated between all objects from the same cluster. Another possible stopping criterion is a fixed number of iterations. Two well-known and commonly used approaches are k-Means and k-Medoids.

k-Means. k-Means [Llo57, Mac67] is a partitioning clustering methods, where every point is included in exactly one cluster: $\bigcup_{i=1}^{k} C_i = \mathcal{O}$ and $C_i \cap C_j = \emptyset, \forall i \neq j$. If no initial centers are provided, it chooses k objects randomly from $\mathcal{O}$ as the initial centers $\mu_1, \ldots, \mu_k$. In the assignment step it assigns each object $o \in \mathcal{O}$ to its closest center μ_i based on their distance and calculates the new centers as the resulting means in the recomputation

step. The objective that k-Means tries to be minimize is typically the sum of squared distances

$$SSQ = \sum_{i=1}^{k} \sum_{o \in C_i} \mathbf{d}(o, \mu_i)^2.$$

k-Medoids. k-Medoids is a partitioning method that uses objects as representative centers instead of cluster means. This allows on the one hand clustering in domains where no mean is defined (as in categorical data) and can on the other hand reduce the influence of outliers on the cluster center positions. Initialization and reassignment in k-Medoids is done as in k-Means. As the objective typically the total distance is used

$$TD = \sum_{i=1}^{k} \sum_{o \in C_i} \mathbf{d}(o, \mu_i).$$

Recomputing the cluster parameters (determining new representatives) can be done in several ways. The simplest solution is to randomly select a pair of representative and non-representative and swap them if the objective improves. The PAM algorithm (Partitioning Around Medoids) [KR90] tests each pair of representative and non-representative and swaps the pair that yields the largest improvement. CLARA (Clustering LARge Applications) [KR90] is a more efficient variant that applies PAM on several randomly drawn samples and returns the best found solution. Further solutions perform an interleaved resampling during the search [NH94]. A combination of k-Means and k-Medoids by interleaving iterations of both methods has been proposed in [GH07].

Expectation maximization. In contrast to partitioning methods the EM algorithm (Expectation Maximization) [DLR77] assigns each object to each cluster. More precisely, each object o belongs to each cluster C_i with a certain probability $P(C_i|o) > 0$. The underlying assumption is that the data is generated by a mixture of probability distributions. The standard EM algorithm assumes Gaussian mixtures (cf. Definition 2.4) and defines each cluster C_i as a Gaussian distribution $\mathcal{N}(\mu_i, \Sigma_i)$. Initialization can for example be done using a k-Means clustering result, keeping the means μ_i, computing the covariance matrices Σ_i from the points in cluster C_i by the standard formula

and setting the prior probabilities $P(C_i)$ to the proportion of points in C_i. In the reassignment step the *expected* cluster membership is calculated; each object o is assigned to each cluster C_i with probability

$$P(C_i|o) = \frac{P(C_i)\, p(o|C_i)}{p(o)} = \frac{P(C_i)\, \mathbf{g}(o, \mu_i, \Sigma_i)}{\sum_{j=1}^{k} P(C_j)\, \mathbf{g}(o, \mu_j, \Sigma_j)},$$

where $\mathbf{g}(o, \mu_i, \Sigma_i)$ is the Gaussian density function according to Definition 2.3. In the recomputation step the likelihood of the distribution is *maximized* with respect to the data by updating the cluster parameters according to

$$\begin{aligned} P(C_j) &= \frac{1}{n} \sum_{i=1}^{n} P(C_j|o_i) \\ \mu_j &= \frac{\sum_{i=1}^{n} o_i \cdot P(C_j|o_i)}{\sum_{i=1}^{n} P(C_j|o_i)} \\ \Sigma_j &= \frac{\sum_{i=1}^{n} P(C_j|o_i) \cdot (o_i - \mu_j) \cdot (o_i - \mu_j)^T}{\sum_{i=1}^{n} P(C_j|o_i)} \end{aligned}$$

The EM algorithm can easily be transformed into a partitioning method by assigning each object to the most probable cluster after the last iteration. k-Means then corresponds to a special case of the EM algorithm where the covariance matrix Σ_i is for all clusters fixed to the d-dimensional identity matrix. Due to the assignment of objects to their closest representative, the partitions induced by k-center methods always exhibit a convex shape (Voronoi cells). A drawback of these methods is that the user must specify the number of clusters k.

Density-based and Distribution-based methods

Density-based clustering methods search groups of data as connected and highly populated (dense) regions. They automatically determine the number of clusters based on density and connectivity and avoid at the same time the restriction to convex partitions.

DBSCAN and OPTICS. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [EKSX96] iteratively grows regions of sufficient den-

sity using the notions of core points and density-reachability. A point p is a core point if there are sufficient objects in its neighborhood as specified by two user parameters. Two points p and q are density-reachable if they are connected by a chain of neighboring core points. To find clusters in a given data set the algorithm iteratively searches for a core object that is not yet clustered and expands a new cluster from there on by adding the object and all its density-reachable objects to a new cluster.

To cope with the difficulty of finding a good parameter setting for DBSCAN on real data sets, the OPTICS algorithm has been proposed as a *visual* analysis method [ABKS99] (cf. Figure 2.2). Any choice for a neighborhood size (ϵ value in DBSCAN) corresponds to a horizontal line in the plot that cuts of the peaks and marks the objects above the line as noise. Each resulting valley underneath the line then represents one cluster.

Grid-based clustering. Grid-based clustering algorithms constitute a type of density-based clustering where the data space is divided into rectangular cells by defining split points along each dimension. Points are then included in their corresponding cell and noise or unpopulated regions of the data space can be removed by a simple density threshold on the resulting cells. Clusters can finally be determined as sets of neighboring cells that are populated or dense. STING [YM97] is a grid-based clustering algorithm that uses multiple resolutions of grids and stores additional statistical information in the grid cells. WaveCluster [SCZ98] uses multiple resolutions and employs a wavelet transformation on the feature space to find dense regions in the transformed space, in [BC00] different resolutions are used to determine clusters based on the fractal dimension.

DENCLUE. The DENCLUE algorithm [HK98] uses density distribution functions for clustering, which are basically kernels centered at the data points and the overall density function is a mixture of all kernels. Clusters are identified using the local maxima of the overall density function; a hill climbing algorithm is employed using the gradient of the density function to assign all points to one cluster that reach the same local maximum. Again noise can be removed by applying a density threshold for each point.

Figure 2.4: Data set, image from unordered objects and VAT image [BH02]. Lighter pixels correspond to larger distances, the dark squares along the diagonal correspond to dense regions in the data.

Hierarchical and specialized methods

Hierarchical clustering methods provide clusterings at different granularities. A popular example is the BIRCH algorithm [ZRL96]. It stores cluster representation in a tree structure where each entry summarizes all objects in its corresponding subtree. COBWEB [Fis87] is a hierarchical clustering algorithm that uses classification trees for clustering. Since BIRCH and COBWEB are discussed in the main part of the thesis, a more detailed description is provided in Chapter 3.

Generally, hierarchical methods either split the data set top down (*divisive*) or iteratively merge sets of objects bottom up (*agglomerative*). Examples of hierarchical methods can be found in [DE84]. A simple bottom up approach merges in each step the two closest clusters together. The distance between two clusters can be defined as the distance between their centers or using the minimum, maximum or average distance between included points. Using the minimum distance in agglomerative clustering is equivalent to building a minimal spanning tree where each point corresponds to one vertex in a graph. The analogue divisive method produces the same cluster hierarchy by starting from a minimal spanning tree and removing always the edge associated with the largest distance.

A method that uses the minimal spanning tree construction to "**v**isually **a**ssess the cluster **t**endency" (VAT) was proposed in [BH02]. The output of VAT for a data set $\mathcal{O} = \{o_1, \ldots, o_n\}$ is an $n \times n$ image where the intensity of the pixel at position (i, j) corresponds to the distance between the points o_i and

o_j, the lighter the color the larger the distance. The order of the objects is the order in which they are processed by the minimal spanning tree algorithm. Figure 2.4 shows an example of a VAT image. Several extensions of VAT have been proposed to deal with large data sets, arbitrarily shaped clusters or automated cluster detection from VAT images [HBH05, HBH06, WNB+10].

A more sophisticated graph-based clustering algorithm has been proposed in [KHK99]. It constructs a k-nearest-neighbor graph where each point corresponds to one vertex and an edge is inserted between two vertices if one of the points lies in the k-neighborhood of the other. The edges are weighted by the similarity between the vertices, which can be computed as the inverted distance, for example. The graph is then first partitioned into a relatively large number of small clusters, such that the edge cut (the total weight of all removed edges) is minimized. In a second step the resulting clusters are merged based on two criteria (interconnectivity and closeness) that exploit the graph structure of the clusters.

Further clustering approaches are for example frequent pattern-based clustering [BEX02] or MDL-based clustering [SVvL06]. Specialized solutions have been proposed for constraint-based clustering [TNLH01, THH01], subspace clustering and projective clustering in high dimensional spaces [AGGR98, APW+99, CFZ99, MSE06, AKMS08] or clustering of complex data types like time series [Lia05]. In Section 3.3 proposed methods for clustering on data streams are reviewed.

Chapter 3

Stream Data Mining

The requirements for stream mining algorithms have been discussed in Chapter 1. The previous chapter provided the background on the KDD process and reviewed established methods for classification and clustering which are not designed for working on data streams. This chapter deals with algorithms for stream mining. In Section 3.1 general tools and techniques are presented that deal with the special requirements in a data stream scenario. After that existing algorithms for stream classification and stream clustering are reviewed in Sections 3.2 and 3.3, respectively.

3.1 General Tools and Techniques

Two important requirements for stream mining algorithms are a compact representation of historical data and a mechanism that appropriately handles evolving data distributions. Solutions to these are detailed in Sections 3.1.1 and 3.1.2. A research area that is closely related to stream mining is *change mining* [BHS08], which concentrates on describing the changes and identifying or predicting when a change occurs. Section 3.1.3 provides an example for a change mining approach and outlines how the approach can be combined with the novel methods proposed in Part III.

Before going into detail about the general tools and individual algorithms, the following formally defines a stream of data objects.

Definition 3.1 ***Stream.*** *A stream* $\boldsymbol{S} : \mathbb{N}_0 \to T \times \mathcal{Q} : i \mapsto (t_i, o_i)$ *is an endless sequence of objects* $o_i \in \mathcal{Q}$ *from a* d*-dimensional input space* $\mathcal{Q}$ *and* $t_i \geq 0$ *is the arrival time of object* o_i*.*

3.1.1 Compression and the BIRCH Algorithm

Compression is a fundamental technique in stream mining algorithms. It results in a compact representation of the data, which yields on the one hand space efficiency and on the other hand time efficiency, since algorithms only process the compressed representation. Compression of a set of data objects can be achieved, for example, through clustering algorithms, statistical approaches or advanced methods that use wavelets or Fourier transformations. Clustering can be used for compression by storing only the representatives of a k-center approach and the resulting cluster parameters. Statistical approaches may use parameterized density functions or a mixture of these to describe data distributions in a compressed way. Another frequently used approach is a so called 'binning' of attribute values where attribute ranges are divided into intervals (of fixed or adaptive length) for which only counts are maintained.

An indispensable requirement for compression methods in a streaming scenario is the ability to incrementally update the maintained summary or representation. This requirement is obvious from the facts that new data is constantly arriving and that the distribution underlying the data may evolve over time. Considering the examples from above we find that simply maintaining cluster centers, weights and radii does not sufficiently meet the update requirement. While centers and weights can be updated, the radii (the maximal distance from the center to a contained point) cannot be calculated exactly when the centers change (deletions of points are common in data stream scenarios). Moreover, if the model is intended to reflect recent data more strongly, shrinking of radii due to the aging of points cannot be realized by this approach. For binning approaches the updating of counts can

be done easily. However, fixed binning approaches face the problem that, in the case of evolving data distributions, the number of populated cells may be exponential in d. Adaptive binning approaches are eventually forced to redefine the value intervals where they face the problem of redistributing the counts without actual knowledge about the original points.

The BIRCH clustering algorithm [ZRL96] introduces cluster feature vectors as a compact representation that allows for efficient incremental updates and simple computation of distribution parameters corresponding to the summarized data. Moreover, BIRCH enables fast access to the maintained summaries by organizing them in a hierarchical data structure. The BIRCH algorithm is described in the following including details of the aspects mentioned above.

Definition 3.2 ***Cluster feature vector.*** *For a set of objects contained in a cluster C the corresponding cluster feature vector $CF = (n, LS, SS)$ stores the number $n = |C| = \sum_{o_i \in C} 1$ of objects contained in C and their linear and quadratic sum per dimension: $LS = (LS_1, \ldots, LS_d)$, $LS_j = \sum_{o_i \in C} o_{ij}, \forall j = 1 \ldots d$ and $SS = (SS_1, \ldots, SS_d)$, $SS_j = \sum_{o_i \in C} o_{ij}^2, \forall j = 1 \ldots d$.*

n, LS and SS are the distributive measures from which the algebraic measures mean μ_j and standard deviation σ_j can be computed per dimension as follows:

$$\mu_j = LS_j/n \tag{3.1}$$

$$\sigma_j = \sqrt{(SS_j/n) - (LS_j/n)^2} \tag{3.2}$$

The latter follows directly from the linearity of expected values. In the BIRCH algorithm only the squared sum over all objects and all dimensions is stored in the cluster feature vector, which allows only the computation of the average standard deviation over all dimensions. A useful property of cluster features is their additivity.

Theorem 3.1 ***Additivity of cluster feature vectors.*** *For two disjoint clusters C_A and C_B with cluster feature vectors $CF_A = (n_A, LS_A, SS_A)$ and $CF_B = (n_B, LS_B, SS_B)$ assume that $C = C_A \cup C_B$ is the cluster that results from merging C_A and C_B. Then the resulting cluster feature vector for C is*

$$CF_A + CF_B = (n_A + n_B, LS_A + LS_B, SS_A + SS_B).$$

From the above theorem it becomes obvious that cluster features can be easily and efficiently updated by simply adding new points to all components of the CF vector. The BIRCH algorithm stores cluster features in a balanced hierarchical data structure called cluster feature tree. An inner node stores entries of the form $[CF, pointer]$, where the pointer corresponds to a child node and the cluster feature (CF) summarizes all entries in the child node (cf. Theorem 3.1). Leaf nodes only contain CF vectors without child pointers and store an extra two pointers to neighboring leaf nodes for efficient scans. The size of a node (the number of contained entries) is determined by the size of a page that is read from hard disc. The size of the complete CF-tree is determined by the amount of available memory.

To create the tree, new points are recursively added to the closest entry. New entries are created if an object reaches a leaf node and its insertion would cause the standard deviation (taken as the radius) of the corresponding CF vector (cluster) to exceed a given threshold τ. If a resulting node contains too many entries, it is split by choosing the farthest pair of entries as seeds and redistributing the remaining entries based on the closest criterion [ZRL96]. Once the CF-tree exceeds its maximal size, a new threshold value τ' is determined and a smaller tree is built from the current CF-tree using τ' before the clustering process can be continued with additional points. When the CF-tree is rebuilt, optionally leaf entries of low density can be stored separately, since those may correspond to noise or outliers. For these CF vectors it is regularly checked whether they can be reabsorbed into the CF-tree.

Cluster feature vectors constitute an excellent way of compression for streaming data and are indeed used in many stream clustering approaches (cf. Section 3.3). The BIRCH algorithm is a first step towards stream clus-

tering, since it processes each object only once; it uses a single linear scan. A drawback of BIRCH in a streaming scenario is the periodical rebuilding of the CF-tree and the reinsertion of possible outliers. Moreover, every object is considered equally important regardless of its age; the resulting clustering does not correspond to an up-to-date model in the sense that it reflects recent data more strongly. While BIRCH has been developed as an algorithm for clustering large data bases, for data stream mining the issue of evolving data distributions is of high importance. This aspect is detailed in the following section.

3.1.2 Mechanisms for aging data streams

The goal of the mechanisms presented in this section is to focus on data from a specific time interval. Therein the most recent data is of primary interest and is therefore given priority in all described approaches.

Sliding window

The simplest mechanism is the so called *sliding window* [GKS01, BBD+02, ZS03]. Two variants of sliding windows determine the set of contained objects either using a fixed size or a threshold for the age of the objects. For the former, objects that newly arrive on the data stream are added to the object set of the window and if the maximal capacity is reached, the oldest object drops out. If the age of the objects is used to determine their membership to the window, the actual size of the window is not fixed unless the data stream is constant.

All objects in the window are assigned an equal weight, usually 1; other objects are assigned a zero weight. As a consequence all objects in the window are equally important, since they are incorporated into the model of the actual algorithm with equal weights. While the sliding window is simple and easy to use, its weighting scheme does not allow for prioritization of the objects within the window according to their age. Moreover, there is no provision for an analysis of older data that already dropped out of the window.

Damped window

The *damped window* [AHWY04, CEQZ06] is similar to the sliding window, but introduces individual weights for the objects based on their age. In the literature the employed aging functions determine the weight for an object o as $w(o) = b^{-\lambda(t_{now}-t_o)}$, where t_{now} is the current time stamp and t_o the arrival time of o. The base b and the decay factor λ determine the shape of the aging function. If $b = 2$ then $1/\lambda$ is the half life of o. Using the above aging function, the total weight of the stream is bounded by

$$W = v \cdot \sum_{t=0}^{now} 2^{-\lambda t} = \frac{v}{1 - 2^{-\lambda}} \tag{3.3}$$

where v is the speed of the stream corresponding to the number of points arriving in one unit of time [CEQZ06].

The exponential aging function employed in the damped window gives more influence to recent data when building the model in an actual mining algorithm. The membership of the objects to the window can still be done either using a fixed size for the window or a threshold for the object weight (based on the age). As in the sliding window mechanism, an analysis of historical data cannot be done using a damped window alone.

Landmark window and tilted time frame

To enable the analysis of historical data, aggregates of the streaming data must be computed and maintained. The simplest mechanism is the *landmark window* [GKS01], which stores aggregates at regular time intervals. To compute a model in an actual mining algorithm it then uses the entire history from a specific time point called the 'landmark'. In a real data stream application, however, this model would fail since it does not provide any means to delete or reduce the amount of the stored aggregates, which is essential on infinite streams.

The *tilted time window* [CDH+02] stores aggregates at different granularities depending on the recency of the data; it stores fewer aggregates for older data and provides a more detailed view on recent data. Assume for example that for the most recent 15 minutes an aggregate is stored every minute. The older aggregates are compressed to one per quarter for the last hour. Even older data is represented by one aggregate per hour, then one per day, one per month, etc., yielding 15 snapshots for the minutes, 4 for the quarters, 24 for hours, 30 for days and 12 for months. In this example $15 + 4 + 24 + 30 + 12 = 85$ aggregates are stored for a whole year, while maintaining one aggregate per minute throughout would yield $60 * 24 * 30 * 12 = 518400$ aggregates.

Pyramidal time frame

The *pyramidal time frame* was introduced in [AHWY03]. It is the most advanced method presented here and can also be used in combination with the novel methods proposed in Part III. For these reasons it is described in more detail. The pyramidal time frame stores snapshots of clusterings as aggregates. The single clusters are represented by cluster features (cf. Definition 3.2) such that the difference between two snapshots can be computed using the additivity of cluster features (cf. Theorem 3.1). The principle of the pyramidal time frame, however, can be used for any type of additive aggregates.

Similar to the tilted time window, the pyramidal time frame stores recent aggregates at a high granularity and maintains only few aggregates for older data. The name is due to the employed data structure that resembles a pyramid: it stores the aggregates in levels of decreasing cardinality. The size of the pyramid is controlled by the two parameters α and β, where, roughly said, α controls the height and β the width of the pyramid (an example is provided below). The following steps determine whether and where an aggregate is stored in the data structure:

1. An aggregate that is taken at time t_{now} is stored on the highest level i with $t_{now} \bmod \alpha^i = 0$
2. If a level contains more than $\alpha^\beta + 1$ aggregates, the oldest is removed.

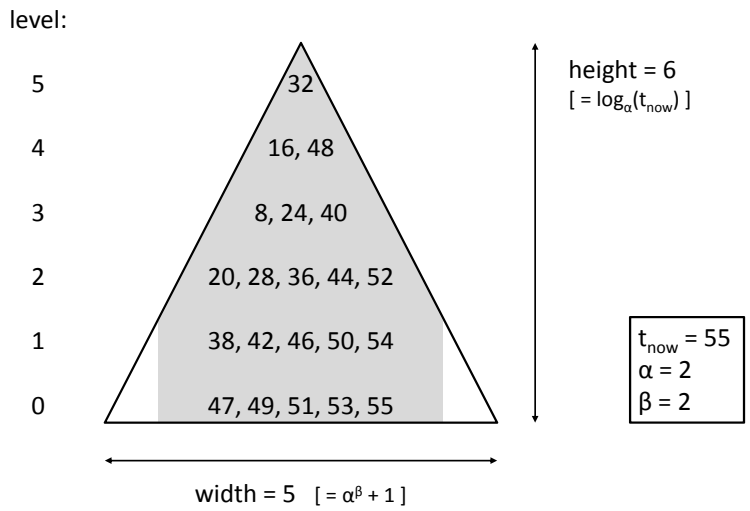


Figure 3.1: Example for the pyramidal time frame.

Consequently, at time t_{now} the height of the pyramid is at most $\lceil \log_\alpha(t_{now}) \rceil$ and the total number of stored aggregates is at most $(\alpha^\beta + 1) \cdot \lceil \log_\alpha(t_{now}) \rceil$. Figure 3.1 shows an example for the pyramidal time frame. In the example $\alpha = 2$, $\beta = 2$ and a snapshot is taken at every time unit. Hence, 55 snapshots have been taken in the example. The height of the pyramid is $\lceil \log_2(55) \rceil = 6$ and a level stores at most $2^2 + 1 = 5$ snapshots. Level 5 stores snapshots from time stamps which are divisible by 32, level 4 those that are divisible by 16 but not by 32, etc. The maximal number of snapshots per level cuts off the edges of the pyramid; the resulting structure therefore rather resembles a house or a tower (depicted by the gray area).

The flexibility that is given to the user by choosing the parameters α and β was shown in [AHWY03] by means of the following example. With $\alpha = 2$ and $\beta = 1$ one snapshot per second over 100 years would yield $(2 + 1)\lceil \log_2(60 * 60 * 24 * 365 * 100) \rceil = 96$ aggregates. This is on the one hand very efficient, but yields on the other hand only a very coarse representation. Allowing more snapshots per level by setting $\beta = 10$ in the same example yields $(2^{10} + 1)\lceil \log_2(60 * 60 * 24 * 365 * 100) \rceil = 32800$ aggregates.

To analyze the data from a time interval $[t_1, t_2]$ the two aggregates X_1 closest to t_1 and X_2 closest to t_2 are searched in the pyramid and a representation of the data between t_1 and t_2 is given by $X_2 - X_1$ using the additivity

of aggregates. An example for such an analysis is the change mining algorithm described in Section 3.1.3. Any interval $[t_{now} - h, t_{now}]$ that considers the most recent h time steps can at least be approximated with a factor of 2; a snapshot will always be found that lies at most at $t_{now} - 2 \cdot h$ as shown in [AHWY03]. Increasing the parameter β improves the approximation and a snapshot will be found within $(1 + \frac{1}{\alpha^{\beta-1}}) \cdot h$. In the above example of a 100 year old data stream with $\alpha = 2$ and $\beta = 10$ a snapshot will be found within $(1 + \frac{1}{2^{10-1}})h = 1.002 \cdot h$ for any h.

3.1.3 Change Mining

The mechanisms described in the previous section enable a stream mining algorithm to focus on more recent data and to maintain an up to date model. In contrast, research in change mining seeks to identify and describe changes in the data distribution. As one example a method called MONIC (modeling and monitoring cluster transitions) that was proposed in [SNTS06] is discussed here.

Given two clusterings $\mathcal{C}_1$ and $\mathcal{C}_2$ from two different time steps $t_1 < t_2$, MONIC distinguishes between internal and external transitions of a cluster from t_1 to t_2. For both groups a set of possible transitions is identified and formally described in [SNTS06]. As a prerequisite the overlap $overlap(C_1, C_2)$ of a cluster $C_1 \in \mathcal{C}_1$ to a cluster $C_2 \in \mathcal{C}_2$ is defined and the match for C_1 in $\mathcal{C}_2$ is derived as the cluster in $\mathcal{C}_2$ with the highest overlap (if the highest overlap is less than a threshold τ then C_1 does not have a match in $\mathcal{C}_2$). The identified external transitions are splitting, merging, disappearing, emerging and survival. For the latter the identified internal transitions are size transition (growing, shrinking), compactness transition (compacter, diffuser), location transition (mean shift, skew) and no change.

The overlap $overlap(C_1, C_2)$ of a cluster C_1 to a cluster C_2 in MONIC is based on the intersection $C_1 \cap C_2$. This requires that the identifiers of the actual objects that are summarized in the cluster need to be accessible to apply MONIC on two given clustering. The clusters maintained by a stream clustering algorithm, however, usually offer only a compact representation such

as cluster features (cf. Definition 3.2). The applicability of change detection methods to stream clustering outputs is nonetheless often given through so called micro clustering. Most stream clustering algorithm maintain a detailed micro clustering in an online component and defer the actual clustering to an offline component, which uses the stored micro clusterings. If the single micro clusters are given a unique ID by the online component, these can be utilized to compute overlaps, matches and transitions using MONIC.

3.2 Stream Classification

Research on stream classification seeks to develop algorithms that adhere to the requirements of stream data mining listed in Chapter 1.

Limited resources. Limited time and limited memory can be handled by restricting the classifier's model accordingly; limiting the number of support vectors in an SVM or the number of objects for a nearest neighbor classifier are two examples. In the following *budget* algorithm refers to an approach which restricts its model such that the classification decision is reached within a previously known time budget.

Evolving Data. Updating the model of a classifier in the presence of changing data distributions is less straightforward. Proposed methods are discussed in Section 3.2.1, some of which constitute general approaches while others are specialized solutions for a specific classification paradigm. To be able to adapt a model, new training data is constantly required. Hence, in the literature it is assumed that the objects in a data stream according to Definition 3.1 are interleaved with labeled objects $o \in \mathcal{Q}^+$ [WFYH03, KJ00]. In practice such labeled instances may result from supervised phases of the application, where an expert provides the ground truth labels manually.

Varying data rates. Anytime classifiers that adapt to varying data rates and flexibly use the available processing time are a more recent field of research. A review of the exisiting approaches is provided in Section 3.2.2, finding that their number is significantly smaller than that of conventional stream classification algorithms.

Many stream classification algorithms have been proposed, surveys and further references that go beyond the approaches discussed below can be found in [GZK05, Agg06, GG07].

3.2.1 Learning on evolving data streams

A general approach to building classifiers for evolving data streams has been proposed in [WFYH03]. The basic idea is to maintain a set of m classifiers trained on consecutive chunks of the stream and employ them in a weighted ensemble based on their classification performance. Newly arriving labeled objects are added to the current chunk $\mathcal{T}^x$ until a fixed amount is collected (*chunk size*). A new classifier is trained on $\mathcal{T}^x$ and the performance of the existing classifiers is computed using $\mathcal{T}^x$ as the test set. Each classifier is given a weight proportional to its accuracy and the m highest weighted classifiers are kept. The weights are used during classification in an ensemble of all m classifiers. The approach is applicable to any classification paradigm, provided that the times for training and classification meet the constraints of the actual stream setting.

Another approach that processes the data stream in chunks has been proposed in [KJ00]. The paper describes an implementation of the approach using support vector machines, but the idea can be generalized to a certain extent. As its input the approach assumes t chunks $\mathcal{T}^0, \ldots, \mathcal{T}^{t-1}$ of equal size m. It then trains t classifiers on all objects from the most recent h chunks for $h = 1 \ldots t$. The performance of the trained classifiers is assessed by testing on each object from the latest chunk in a leave-one-out fashion; each classifier must be trained m times. Since this procedure soon becomes prohibitively expensive, the error is estimated with a pessimistic bias, where the true error is overestimated on average. The output of the approach is the classifier which has the smallest estimated error. The error estimation thwarts the generality of the approach, since it either must be defined for other paradigms to be able to use the methodology or the leave-one-out performance of the paradigm must comply with the time constraints of the actual stream.

For the nearest neighbor classifier an adaptive version for evolving streams has been proposed in [BMH+05]. A fixed number of weighted representatives is maintained based on the prevailing time and space constraints. For a newly arriving labeled object o the closest representative $\hat{r}$ is determined according to some distance measure $dist(o, \hat{r})$. If $dist(o, \hat{r})$ is larger than a predefined threshold, o is stored as a new representative. Otherwise o is merged to $\hat{r}$ by moving its position taking the weight into account. If the labels of o and $\hat{r}$ are equal, the weight of $\hat{r}$ is increased by 1, otherwise 1 is subtracted. If the resulting weight is 0 then $\hat{r}$ is removed from the representatives. The classification decision for an object q is determined based on the k nearest neighbors among the representatives taking their weight into account.

Decision trees are an extensively researched paradigm in evolving data stream scenarios. In [DH00] a system called VFDT (Very Fast Decision Tree learner) has been proposed entailing a rich diversity of improvements and modified variants such as [HSD01, JA03, GMR04, BHP+09]. In decision tree induction leaf nodes are recursively replaced by test nodes (inner nodes) associated with a split attribute and one or several split values (cf. Section 2.2 and Definition 2.5). The core idea of VFDT is to consider only a small set of objects per leaf node to decide on the split. The size of the set is chosen such that, with high probability, the chosen attribute is the same that would have been chosen using infinite examples. To this end a statistical result known as Hoeffding bound (or additive Chernoff bound) [Hoe63, MM93] is employed, yielding the name Hoeffding trees for the decision trees used in VFDT and its variants. The Hoeffding trees grow top down by incrementally adding new labeled objects to the leaf nodes and eventually splitting them using the best attribute according to a splitting criterion of choice such as the information gain (cf. Equation 2.18). If the available memory is exceeded, the least promising leafs are deactivated in order to make room for new ones. The VFDT system includes further heuristics for solving ties between two attributes, increasing the efficiency by delaying computations of the splitting criterion, reducing the number of attributes or initializing VFDT to ensure a good early performance.

A common method for Bayesian classification on categorical attributes are Bayesian networks (cf. Section 2.2). Inference of a Bayesian network determines the network topology as well as the parameters for the conditional probability tables. While the latter naturally provide the means to efficiently add new training data or remove outdated training data by adjusting the entries in the conditional probability tables, determining the network topology is a more time consuming process. In [LW96] an anytime learning method for Bayesian networks based on state-space abstraction is discussed. State-space abstraction refers to restricting the number of possible values (categories, intervals) per attribute in order to reduce the size of the parameter set Θ (cf. Equation 2.12). This way the evaluation of a trial topology is accelerated. The restrictions on the state-space are iteratively loosened by refining one category for one attribute at a time until the process is interrupted or all states are elementary.

Both [DH00] and [LW96] declare their methods as anytime algorithms. In either case the term anytime refers to the training of the classifier: VFDT "is an anytime algorithm in the sense that a ready to use model is available at any time" [DH00] and the latter "considers an anytime procedure for approximate evaluation [inference] of Bayesian networks" [LW96]. While being able to learn in an anytime manner is certainly worthwhile, the ability to perform anytime classification is of higher value in streaming scenarios. Existing anytime classifiers are discussed in the next section.

3.2.2 Anytime classifiers

Given a test object $q \in \mathcal{Q}$, an anytime classifier $\mathcal{C}$ can provide a classification decision $\mathbf{f}_{\mathcal{C}}(\Theta, q, t)$ at any time $t \geq t_0$ after a very short initialization time t_0.

A simple and intuitive anytime classifier for nearest neighbor classification has been proposed in [UXKL06]. While the traditional nearest neighbor classifier is a lazy learner that essentially skips the training phase and exhibits high classification times especially on larger data sets* , the anytime

*Similarity search on large data sets can be sped up to some extend by index structures and other appropriate methods as discussed in Chapter 2.

nearest neighbor is quite the opposite. The basic idea is to compute a ranking on $\mathcal{T}$ during the training phase and process $\mathcal{T}$ in the resulting order during classification until the algorithm is interrupted or $\mathcal{T}$ is entirely read. Different ranking heuristics are discussed in [UXKL06]; the recommended heuristic assigns to each object $o \in \mathcal{T}$ a rank according to its performance in a leave-one-out classification on $\mathcal{T}$. The rank is computed as

$$rank(o) = \sum_j \begin{cases} 1 & \text{if } label(o) = label(o_j) \\ -\frac{2}{|\mathcal{L}|-1} & \text{otherwise} \end{cases} \tag{3.4}$$

where $o_j \in \mathcal{T}$ is an object having o as its nearest neighbor. Ties are resolved by sorting objects with the same rank according to their distance to their nearest neighbor of the same class. During classification the objects are processed in decreasing rank order and on interruption the label of the currently closest object is returned as the classification result. The training time of the anytime nearest neighbor is rather high, consider 53 minutes for $|\mathcal{T}| = 11340$ on a Pentium 4 with 3.0GHz as an example from [UXKL06]. Moreover, the employed ranking does not allow a straightforward incremental addition of new training data as desirable in the data stream context.

For support vector machines several anytime versions have been proposed [DeC02, DeC03, DM03]. The solution proposed in [DM03] relies on two main observations. The first observation is that for an SVM the same classification decision can often be reached by using only a small fraction of the nearest support vectors of a query in the decision function (cf. Equation 2.17). The second observation is that the ordering of the nearest neighbors can be approximate without harming the actual performance. Putting the second observation into practice the principal components of the support vectors are determined and the nearest neighbors for an object $q \in \mathcal{Q}$ are computed during classification using only the subspace spanned by the "top few principle component dimensions" [DM03]. The realization of the first observation follows directly from evaluating the support vectors incrementally with respect to their distance to q. Earlier anytime SVM versions iteratively compute tighter bounds on the SVM output using either distance

geometry [DeC02] or Cholesky factorization [DeC03]. In [DeC02] the support vectors are processed during classification in a fixed order (similar to [UXKL06]), which is precomputed via a greedy eigenvector analysis.

Anytime classification for the Bayes classifier on categorical attributes has been discussed in [YWKT07]. The system is called AAPE (anytime averaged probabilistic estimator) and consists of the following four steps:

1. Identify single improvement steps.
2. Order single improvement steps.
3. Invoke next single improvement step until time is exhausted.
4. Ensemble improvement steps to output class probability estimates.

Initially a naïve Bayes is evaluated to obtain a first classification result. As the single improvement steps SPODEs (superparent-one-dependent estimators) are used, which were introduced in [KP02] and frequently investigated in subsequent research [WBW05, ZW06, ZW07, YWC+07, FGMP09]. SPODEs assume that the attributes are independent of each other given the class label and a *common* attribute: the superparent. The SPODE considering the p-th attribute as superparent yields the posterior probability (cf. Equation 2.6) given by

$$P(l|o,p) = \frac{P(l,o_p)P(o|l,o_p)}{P(o)} = \frac{P(l,o_p)\prod_{i=1}^{d} P(o_i|l,o_p)}{P(o)} \tag{3.5}$$

Since any of the d attributes can be utilized as superparent, a total of d improvement steps are given in AAPE.

For the second step seven heuristics to determine the best order of SPODEs are discussed and evaluated in [YWKT07]. The recommended heuristic is called FSA (forward sequential addition), in which the SPODEs are successively added to an ensemble based on the resulting performance with respect to a leave-one-out evaluation on $\mathcal{T}$. Further heuristics include backward sequential elimination, random ordering and others. After computing the single posterior probabilities in step three until interruption, the

final decision is given by

$$\hat{l} = \arg\max_{l \in \mathcal{L}} \left\{ \frac{P(l)P(o|l) + \sum_{p \in I} P(l, o_p)P(o|l, o_p)}{P(o) \cdot (|I| + 1)} \right\} \tag{3.6}$$

averaging over the probabilities of the naïve Bayes and all evaluated SPODEs subsumed in I. While the probability tables used for the SPODEs can be incrementally updated with new training data, the order of the SPODEs requires more expensive recomputation, which cannot be done online in most cases.

3.2.3 Summarizing remarks

As noted in Section 2.2 different domains call for different classifiers. The development of anytime classifiers is about enabling the use of a specific classification paradigm in an anytime setting; it corresponds to designing an algorithm that follows Definition 1.1 for the paradigm at hand. Ideally a classifier supports both incremental learning and anytime classification. As described above anytime classifiers have been developed for SVMs, nearest neighbor classification and Bayesian classification for categorical attributes. In Part II the focus is on Bayesian anytime classification for continuous attributes. Therein first an anytime classifier is developed that can incrementally learn from new labeled instances arriving on the stream. Consequently the focus is on improving the method by building the classifier offline; to this end the general approach for evolving data streams can be employed that was proposed in [WFYH03] and discussed in Section 3.2.1.

3.3 Stream Clustering

Being one of the most frequently used data mining techniques, the development of clustering algorithms promptly followed the real world situation of data repositories; single scan algorithms were proposed when large data bases became prevalent and stream clustering algorithms followed soon after. For most of the clustering paradigms discussed in the Section 2.3 a streaming algorithm has been developed based on the static methods including stream variants of k-Means and DBSCAN, but also grid-based, graph-based and MDL-based approaches.

3.3.1 k-center methods

As in static clustering, k-center clustering approaches are numerous and among the most popular. Two algorithms are presented in greater detail and an overview of further k-center methods from the literature is provided thereafter.

STREAM

One of the first stream clustering algorithms called STREAM [OMM+02] uses a k-Medoids variant for clustering. Instead of processing each object directly as it arrives, the algorithm periodically collects a certain number of points and processes these as one chunk of data. The outline of the STREAM algorithm is as follows:

1. Perform k-Medoids clustering on the current chunk
2. Store the k resulting centers weighted by the number of assigned points
3. If no memory is left perform k-Medoids on the set obtained in step 2 and replace the set by the resulting centers

The output of STREAM is the k-Medoids result on the set of stored centers. The employed k-Medoids variant is called LSEARCH and is based on the facility location problem, where each cluster center represents a facility. In

contrast to the classical k-Medoids facility location imposes fixed costs ζ to every cluster (facility) and tries to minimize

$$FC(\mathcal{C}) = \zeta \cdot |\mathcal{C}| + \sum_{i=1}^{|\mathcal{C}|} \sum_{o \in C_i} \mathbf{d}(o, \mu_i).$$

Initially the points in the chunk are sorted randomly and a facility is opened at the location of the first point. At each following point a location is opened with probability d/f, where d is the distance of the current point to the closest existing facility.

To find the desired number of k clusters LSEARCH performs a binary search over the costs ζ. ζ is chosen between $\zeta_{min} = 0$ and $\zeta_{max} = \sum_{o \in \mathcal{O}} \mathbf{d}(p_0, o)$, where p_0 is the first point in the random ordering. The costs are in each iteration set to $\zeta = (\zeta_{min} + \zeta_{max})/2$ and the clustering $\mathcal{C}$ is computed. If $|\mathcal{C}| > k$ the cost were too small and the lower bound is updated by setting $\zeta_{min} = \zeta$, if $|\mathcal{C}| < k$ the cost were too large and the upper bound is updated by setting $\zeta_{max} = \zeta$. LSEARCH stops if either $|\mathcal{C}| = k$ or $FC(\mathcal{C})$ did not improve more than a given threshold. To speed up the algorithm a sampling approach is proposed in [OMM+02].

Considering the requirements for streaming algorithms discussed in Chapter 3 the main weaknesses of STREAM are the missing aging of historical points and the missing treatment of noise. However, the algorithm can easily be adapted to lay more focus on recent data by either decreasing the weights of centers from older chunks or removing them in step 3 instead of merging them.

CluStream

CluStream [AHWY03] introduces the notion of an online and an offline component. In the online component a larger number of so called micro-clusters MC are maintained and stored regularly using a pyramidal time frame (cf. Chapter 3). The offline component performs a k-Means clustering on the micro clusters from a user specified horizon. Micro-clusters in CluStream are represented by a temporal extension of cluster features as follows:

Definition 3.3 *An extended cluster feature* $CFT = (n, LS, SS, ls_t, ss_t)$ *stores in addition to a cluster feature* $CF = (n, LS, SS)$ *according to Definition 3.2 the linear sum* ls_t *and quadratic sum* ss_t *of the time stamps of all contained objects.*

Each micro-cluster is assigned a unique id upon creation. The total number of micro-clusters k' is a parameter of the algorithm, which is determined by the amount of available main memory according to [AHWY03]. Initially a fixed number ($InitNumber$) of points are collected and processed by the offline component to obtain k' micro-clusters. The value of $InitNumber$ is chosen to be as large as permitted by the computational complexity of a k-Means algorithm [AHWY03]. For a newly arriving point p the online component of CluStream performs the following operations:

1. Find the closest micro-cluster MC_i according to $\mathbf{d}(p, \mu_i)$
2. If $\mathbf{d}(p, \mu_i) < \alpha \cdot \sigma_i$ then add p to MC_i ($\alpha = 2$ in [AHWY03])
3. Else create a new micro-cluster containing only p and
   (a) delete the least recent MC, if its relevance stamp is below τ
   (b) else merge the two closest micro-clusters

If two micro-clusters are merged the resulting micro-cluster is assigned an id-list containing the member-ids of both. The relevance stamp for the delete check reflects the average time-stamp of the last $\hat{m}$ points. Using ls_i and ss_i it is calculated as the $\hat{m}/(2 \cdot n)$-th percentile of the time-stamp distribution assuming a normal distribution. The parameters of the distribution are calculated using ls_t and ss_t as in the feature case. τ and $\hat{m}$ are parameters of the algorithm.

In contrast to STREAM CluStream incorporates aging of points and micro-clusters through the extended cluster features and the deletion according to recency. Noise points are not directly discarded, since each point is included or assigned a new micro-cluster to allow the formation of new clusters and concepts. However, noise points that stay alone in their micro-cluster will be eventually be removed in step 3(b).

Other approaches

Besides the two described methods a variety of other k-center approaches has been proposed. E-Stream [URW07] maintains for each micro-cluster in addition to the cluster feature a histogram that is used to determine the best dimension for splitting the cluster. MovStream [TTD+08] uses a sliding window, adds a new point to its closest cluster and removes the oldest point from its cluster. OLINDDA [SdLFdCG07] stores a certain number of representatives per cluster such that the total number is constant and performs periodically a k-Means clustering on those objects. New objects that do not fall into an existing cluster are buffered and clustered separately to check for novelty and drift. StreamKM++ [ALM+10] computes a small weighted sample of the data stream using a coreset tree and performs a k-Means variant [AV07] on that sample. The sample size is experimentally shown to yield a good tradeoff between running time and clustering quality when set to $200 \cdot k$, where k is the number of resulting clusters. TRAC-STREAM [NR06] uses Chebyshev bounds to determine outliers and compatibility of clusters. In [Guh09] theoretical bounds are proven for maintaining k clusters of equal radius in a streaming scenario.

3.3.2 Density-based methods

Density-based stream clustering approaches claim to outperform k-center methods by finding clusters of arbitrary shape. This advantage is in most cases however only due to the employed offline component of the approaches. This issue is discussed at the end of this chapter in Section 3.3.4.

DenStream

The DenStream algorithm proposed in [CEQZ06] is based on DBSCAN (cf. Section 2.3). Similar to CluStream, micro-clusters are maintained in an online component and the DBSCAN algorithm is applied to these micro-clusters in the offline component upon user request. The online component differs

significantly from CluStream. It uses a time-weighted variant of cluster features.

Definition 3.4 *Let $w(o) = b^{-\lambda(t_{now}-t_o)}$ be a time weighting function where t_{now} is the current time and $t_o \leq t_{now}$ is the arrival time of an object o. Then the time-weighted cluster feature vector corresponding to a cluster C is $CF = (n^{(t)}, LS^{(t)}, SS^{(t)})$, with $n^{(t)} = \sum_{o_i \in C} w(o)$ and for each dimension $LS_j = \sum_{o_i \in C} w(o) o_{ij}, \forall j = 1 \ldots d$ and $SS_j = \sum_{o_i \in C} (w(o) o_{ij})^2, \forall j = 1 \ldots d$.*

DenStream distinguishes between core micro-clusters CMC with $n^{(t)} \geq \tau$ and outlier micro-clusters OMC with $n^{(t)} < \tau$. $n^{(t)}$ is the time weighted number of points contained in the micro-cluster and τ is a threshold parameter. The offline component is applied only on the core micro-clusters. To insert a new point p at time t the algorithm performs the following steps:

1. Find the closest core micro-cluster CMC_i
2. Add p to CMC_i if afterwards $\sigma_i \leq \epsilon$ holds
3. Else perform steps 1 and 2 to test the closest outlier micro-clusters
4. If p was added to an outlier micro-cluster OMC_j check whether OMC_j became a core micro-cluster
5. If p was not absorbed create a new OMC_p containing only p
6. Periodically after T_{per} time steps, if $t \, mod \, T_{per} = 0$,
   (a) delete all core micro-clusters CMC_i with $n^{(t)} < \tau$
   (b) delete all outlier micro-clusters that did not become a core micro-cluster within the last T_{per} time steps

Step 4 ensures that an outlier micro-cluster can evolve to a core micro-cluster over time, step 6 deletes noise and outdated micro-clusters. Given the maximal total weight W of the time-weighted stream according to Equation 3.3 the number of core micro-clusters is bound by W/τ and it is shown in [CEQZ06] that the number of outlier micro-clusters is also limited. The value

T_{per} for the periodical checks is determined using the decay factor λ and the density threshold τ as $T_{per} = \lceil \frac{1}{\lambda} \log(\frac{\tau}{\tau - 1}) \rceil$.

To initialize the list of maintained core micro-clusters the DenStream algorithm collects an initial number of points ($InitNumber$) in a temporary set. For a point p in this set, if the number of points in its ϵ-neighborhood is above τ, a new core micro-cluster is created containing these points and they are removed from the set.

In [CEQZ06] the core micro-clusters of the online component are called *potential c-micro-cluster* and τ is a product of two parameters. In their experimental evaluation the parameters were set to $\tau = 2$, $\epsilon = 16$, $InitNumber = 1000$ and $\lambda = 0.25$. For these parameters the value for periodical checks is $T_{per} = 4$.

DStream

DStream [CT07] uses a grid-based approach and divides each dimension into l partitions yielding a total number of $N = l^d$ grid cells. It exhibits some similarities with the previously described DenStream algorithm in that it uses a decay function for aging and checks periodically for noise or outdated regions. The online component maintains populated grid cells in a hash list. A grid cell GC is represented by its characteristic vector

$$(t_{update}, t_{spor}, n^{(t)}, label, status)$$

where $status$ can be $sporadic$ or $normal$ and $label$ is used for cluster membership. t_{update} and t_{spor} are the time stamps when GC was last updated or marked as $sporadic$ respectively. With t_p as the insertion time of point p the term $n^{(t)} = \sum_{p \in GC} b^{\lambda \cdot (t_{now} - t_p)}$ is the time weighted sum of all points in GC (density) similar to Definition 3.4. In DStream λ is set to 1 and the base b is used as a parameter to influence the aging. With this parameter choice the total weight of the stream is bounded by $W = \frac{1}{1-b}$ (cf. Section 3.1). This property is used in the online component.

DStream distinguishes three kinds of grid cells using the bounded total weight of the stream $W = \frac{1}{1-b}$ and the total number of grid cells $N = l^d$ as

follows:

- GC is a dense grid cell, if $n^{(t)} > \tau_{dense} \cdot \frac{W}{N}$
- GC is a sparse grid cell, if $n^{(t)} < \tau_{sparse} \cdot \frac{W}{N}$
- GC is a transitional grid cell, if $\tau_{sparse} \cdot \frac{W}{N} < n^{(t)} < \tau_{dense} \cdot \frac{W}{N}$

The two requirements for a sparse grid to be marked as sporadic are

$$S1: \; n^{(t)} < \frac{\tau_{sparse}}{N} \sum_{i=0}^{t_{now}-t_{update}} b^i$$

$$S2: \; t_{now} \geq (1+\beta) t_{spor}$$

$S2$ defines how fast a grid cell that has been marked as sporadic before can be marked again as sporadic. The larger β is chosen by the user the longer it takes for these grid cells to be removed. Moreover, since the comparison is absolute rather than relative, requirement $S2$ becomes more difficult to fulfill as time proceeds.

The main steps of the online component for any new point p arriving at time t are:

1. Determine the grid cell GC that p falls into
2. Add GC to the hash list if it is not already contained
3. Update GC with respect to p and t. If GC changed from sparse to another type, remove possible sporadic mark
4. Periodically after T_{per} time steps, if $t \, mod \, T_{per} = 0$,
   (a) delete all grid cells from the hash list that have been marked as $sporadic$ and did not receive new points within the last T_{per} steps
   (b) mark sparse grid cells as $sporadic$ if requirements $S1$ and $S2$ are met
   (c) adjust the clustering

It is shown in [CT07] that $S1$ and $S2$ ensure that no transitional or dense grid will be falsely deleted due to the removal of a sporadic grid cell.

Initially DStream collects incoming points for T_{per} time steps, inserts them into their corresponding grid cell and defines connected regions of dense or transitional grid cells as one cluster. In step 4(c) the clustering is updated to reflect the density changes of the grid cells, possibly merge newly connected clusters or split clusters that became unconnected.

The value T_{per} for periodical checks is set such that any change of a grid cell from dense to sparse or vice versa is recognized. This is achieved by

$$\begin{aligned} T_{per} &= \min\left\{\left\lfloor log_b \frac{\tau_{sparse}}{\tau_{dense}}\right\rfloor, \left\lfloor log_b \frac{N-\tau_{dense}}{N-\tau_{sparse}}\right\rfloor\right\} \\ &= \left\lfloor \log_b\left(\min\left\{\frac{\tau_{sparse}}{\tau_{dense}}, \frac{N-\tau_{dense}}{N-\tau_{sparse}}\right\}\right)\right\rfloor \end{aligned}$$

In the experimental evaluation in [CT07] the parameters are set to $\tau_{dense} = 3$, $\tau_{sparse} = 0.8$, $b = 0.998$, $\beta = 0.3$, l ranges from 20 to 50 and d ranges from 2 to 40. For these parameter settings the value for T_{per} ranges from 0 to 2.

Other approaches

In [PL04] a dynamic splitting of grid cells is used to cluster streaming data. To this end coarse initial grid cells are repeatedly split based on their data distribution using either a mean split in the dimension with the highest variance, or a split along the dimension with the lowest variance that keeps the elements around the mean in one segment. Grid cells are basically represented by their bounding box and a time weighted cluster feature. Splitting and deleting of cells is performed based on two user defined density threshold parameters. As in DenStream and DStream the algorithm periodically checks all maintained cells to remove noise or outdated regions. The initial cells are always maintained and cannot be deleted. An offline component is not employed. In [LC08] a grid-based stream clustering approach that uses the fractal dimension has been proposed.

3.3.3 Specialized methods

COBWEB [Fis87] is an incremental clustering method that stores only a compact representation of the data and can therefore be used for stream clustering. As a compact representation it uses classification trees, which differ from decision trees (cf. Chapter 2) in that they label nodes rather than branches and store probabilistic descriptors rather than logical ones. The construction of the classification trees is guided using probabilities of attribute values, a variant for continuous attributes using distributions was proposed in [GLF89]. To react to evolving data the algorithm allows merging and splitting of nodes. However, COBWEB, just as the other early method STREAM (cf. Section 3.3.1), does not incorporate aging of historical data.

For high dimensional data streams a projected clustering algorithm called HPStream has been proposed in [AHWY04]. It uses a variant of time weighted cluster features that represents clusters only in the most relevant dimensions. The relevant dimensions are determined globally by keeping the $k \cdot l$ dimensions over all clusters that have the smallest radii, where k is the number of clusters and l is a parameter. Subspace clustering algorithms for data streams have been proposed in [KPM06, PL07, LL08, PL08].

Further stream clustering algorithms use concepts mentioned in Section 2.3 such as MDL-based clustering [vLS08], graph-based clustering [LL09] or model-based clustering using kernel estimators [JZC06]. An extension of OPTICS to data streams has been proposed in [TRA07], a method that employs an immune system learning model was presented in [NUCG03].

3.3.4 Summarizing remarks

Many stream clustering algorithms have an online component that maintains detailed summaries of the streaming data and an offline component that produces the actual clustering. Other methods can be adapted to act as an online component: using a large k value for STREAM or performing step 4(c) in DStream (adjust clustering) only on user requests are examples. The offline components are often exchangeable; from any micro-clustering resulting from an online component a k-Means or DBSCAN clustering can

be calculated. However, the online components often exhibit a rather high complexity with respect to the number of maintained micro-clusters due to expensive checks for deleting or merging micro-clusters. This can be problematic on very fast or bursty streams. This is a first motivation for the development of new methods in Part III. Moreover, none of the proposed approaches for stream clustering constitutes an anytime algorithm; neither is interruptible or exploits additional time once its model has been processed.

Part II

Anytime Stream Classification

Chapter 4

The Bayes Tree

* Classification of streaming data faces three basic challenges: it must deal with huge amounts of data, it must make the best possible use of the varying time between two stream data items (anytime classification) and additional training data must be incrementally learned (anytime learning) for applying the classifier consistently to fast data streams. In this chapter a novel index-based technique is proposed that can handle all three of the above challenges using the established Bayes classifier on effective kernel density estimators. The proposed Bayes tree constitutes a hierarchy of mixture densities that represent kernel estimators at successively coarser levels. The proposed probability density queries adapt the employed mixtures efficiently to the individual object to be classified. Together with novel classification improvement strategies this allows for very effective classification at any point of interruption. Moreover, a novel evaluation method is introduced for anytime classification using Poisson streams, and the performance of the Bayes tree is evaluated empirically.

*This chapter has been published in the Proceedings of the 12th International Conference on Extending Database Technology (EDBT/ICDT 2009) [SAK+09].

4.1 Introduction and Preliminaries

As discussed in Chapter 3, existing classifiers that focused on anytime learning built a budget classifier [BH90, SK01, WFYH03], i.e. they are not able to perform anytime classification. Anytime classifiers on the other hand have so far not been able to profit from novel training data unless they were given a large amount of time to retrain their classifier. For a stream classification application it is important to support both anytime learning and anytime classification to allow consistent use on fast data streams.

The proposed classifier enables anytime Bayes classification using non-parameterized kernel density estimators (cf. Section 4.1.1). Consistent with classifying fast data streams the technique is able to incrementally learn from data streams at any time. The proposed Bayes tree indexing structure provides aggregated model information about the kernels in all subtrees on all hierarchy levels of the tree. In *probability density queries*, descending the tree is based on strategies that favor classification accuracy. As different granularities are available and compact models are located at the top levels, the probability density query can be interrupted early on (starting with a unimodal model at root level) and refines the model as long as time permits. The novel anytime classifier does not only improve its result incrementally when more time is granted, but also adapts the model refinement individually to the object to be classfied.

The design goals of the proposed anytime approach include

- **Statistical foundation.** The classifier uses the established *Bayes classifier* based on kernel estimators.
- **Adaptability.** The model is *adapted to the individual query object*, i.e. the mixture model is refined locally with respect to the object to be classified.
- **Anytime classification and anytime learning** for applying the Bayes tree consistently in applications on fast data streams.

The contributions on a technical level include

- **Novel hierarchical organization of mixture models.** The novel index structure provides a hierarchy of mixture density models and allows *fast access* to very fine-grained and individually adapted models.

- **Probability density queries.** Novel access strategies where traversal decisions are based on the *probability density of the query object* with respect to the current mixture model.

- **Poisson stream evaluation.** A novel evaluation method is introduced for anytime classification on data streams using *Poisson processes*.

In the following section first the evaluation of anytime classification is reviewed and Bayes classification using kernel density estimation is described. In Section 4.1.2 hierarchical indexing is discussed in the context of anytime learning.

4.1.1 Anytime classification and kernel density estimation

Anytime classifiers are capable of dealing with the varying time constraints and high data volumes of stream applications. The usefulness of an anytime classifier depends on its quality, which can be determined with respect to **accuracy** and **accuracy increase**. Accuracy refers to the proportion of correctly classified objects for a fixed time allowance, accuracy increase is the improvement of accuracy over the amount of available time (cf. Figure 1.2). In stream classification the accuracy of an anytime classifier is given by the average classification accuracy with respect to the inter-arrival rate (speed) of the data stream. The accuracy increase also influences the stream specific anytime classification accuracy, since a steeper increase yields a better accuracy even if marginally more time is available and thus a higher average accuracy for the same inter-arrival rate. In Section 4.3 a novel stream specific anytime classification evaluation method is introduced, which captures both aspects in a single measure.

Bayesian classification on continuous attributes requires a model for probability density estimation (cf. Chapter 2). A detailed approach to density

estimation are kernel density estimators, which do not make any assumption about the underlying data distribution (thus often termed "model-free" or "non-parameterized" density-estimation). Kernel estimators can be seen as influence functions centered at each data object. To smooth the kernel estimator a *bandwidth* h_i is used. Comparing Definition 2.3 with kernel densities, the bandwidth corresponds to the variance of a Gaussian component.

Definition 4.1 ***Kernel density estimation.*** *The class conditional probability density for an object $q \in \mathcal{Q}$ and class l_i based on a training set $\mathcal{T} \subseteq \mathcal{Q}^+$ and kernel $\mathbf{K}$ with bandwidth h_i is given by:*

$$p_{\mathbf{K}}(q|l_i) = \frac{1}{|\mathcal{T}_{l_i}| \cdot h_i^d} \sum_{o_j \in \mathcal{T}_{l_i}} \mathbf{K}\left(\frac{\|q - o_j\|}{h_i}\right)$$

Thus, the class conditional probability density for any object q is the weighted sum of kernel influences of all objects $\mathbf{o_j}$ of the respective class. In this thesis, Gaussian kernels $\mathbf{K}_{Gauss}(x) = \frac{1}{(2\cdot\pi)^{d/2}} e^{-\frac{\mathbf{x}^2}{2}}$ are used along with Gaussian mixture models in a consistent model hierarchy to support mixing of models and kernels in the Bayes tree.

In terms of classification accuracy, Bayes classifiers using kernel estimators have shown to perform well for traditional classification tasks [EW95]. Especially for huge training data sets the estimation error using kernel densities is known to be very low and even asymptotically optimal [Sil86]. In section 4.2 a novel indexing structure is proposed that enables kernel density estimation on huge data sets (as in stream applications) and incremental learning at any time. Moreover, it is shown how anytime density estimation using kernel density estimators can be realized with the proposed method. There has been some research on anytime density estimation [ZRL99, GM03]. It is however not described how to use these approaches for anytime classification. As we will clearly see in Section 4.3, this is not a trivial task. Neither a naive solution nor either of the straightforward solutions delivers competitive results.

4.1.2 Anytime learning using hierarchical indexing

Supervised classification is performed in two steps: learning and classification. So far we discussed the quality criteria for anytime classification. However, in stream applications it is often essential to efficiently learn from new labeled objects. As the time to learn from new objects is typically limited, classifiers are required which can be interrupted at any time during learning. The classification model which has been learned up to this point in time is then used for all further classification tasks. A classifier that can learn from new objects at any time is called an anytime learning algorithm in the following (also called incremental learning).

An important quality criterion for anytime learning algorithms is the runtime complexity of updating the learned model. While using lazy learning (e.g. nearest neighbor) allows for updating the decision set in constant time, the lack of a concise model of the data stops this approach from being effective in the anytime classification context. An anytime classifier based on the nearest neighbor approach has been proposed in [UXKL06] (cf. Chapter 3). However, to perform anytime classification, [UXKL06] uses a non-lazy learning method that has a superlinear complexity for new objects.

The method proposed in this chapter uses a hierarchy of models for Bayes classification that allows incremental refinement to meet the requirements of anytime classification and anytime learning. The Bayes tree indexes density models for fast access. Using multidimensional indexing techniques enables the Bayes tree to learn from new objects in logarithmic time.

Multidimensional indexing structures like the R-tree [Gut84] have been shown to provide substantial efficiency gains in similarity search in low dimensional spaces. The basic idea is to organize the data efficiently on disk pages such that only relevant parts of the database must be accessed. This is based on the assumption that in similarity search, query processing requires only a small portion of the data. This assumption does not hold in Bayes kernel classification. As the entire model potentially contributes to the class label decision, the entire database must be accessed in order to perform Bayes classification without loss of accuracy (see Section 4.2 for details). Con-

sequently, simply storing objects for kernel estimation in multidimensional indexes does not suffice for anytime classification.

The Gauss-tree [BPS06] builds on the R*-tree structure [BKSS90] to process unimodal probabilistic similarity queries. As the application focus is on similarity search, it does neither allow for anytime classification, nor for management of multi modal densities.

4.2 Indexing density models

Anytime algorithms and Bayes classification using kernel density estimation are the preliminaries for the proposed anytime classifier. The overall goal is a classification algorithm that can be interrupted at any given point in time and that produces a meaningful classification result that improves with additional time allowances. The Bayes classifier on kernel densities cannot provide a meaningful class conditional density $p(q|l_i)$ at an arbitrary (early) point in time, since the entire model may potentially contribute to the class label decision (cf. Section 4.1.2). In the next section an overview is provided of the technique for estimating the class conditional density before discussing the technical details in the following sections. Finally, it is described how the classification decision is made and improved using the novel index structure.

4.2.1 Outline

As discussed before, any anytime classifier must provide an efficient way to improve its results with additional time. Indexing provides means for efficiency in similarity search and retrieval. By grouping similar data on hard disk and providing directory information on disk page entries, only the relevant parts of the data are accessed during query processing. Similarity search queries are usually specified via a query object and a similarity tolerance given by a threshold range ε. Using this ε range, irrelevant parts of the data are pruned based on directory information during query processing. Thus the amount of data that must be accessed is reduced, which in turn greatly reduces runtime. This is illustrated in Figure 4.1 (top left). The query and

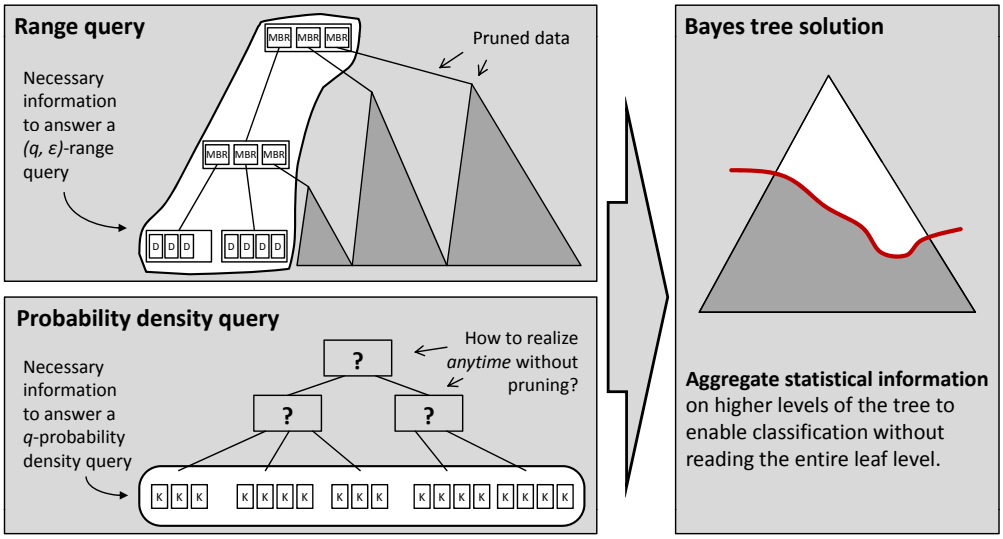


Figure 4.1: Pruning as in similarity search is not possible for probability density queries, since all kernels must be accessed to answer a q-probability density query.

the ε range allow for straightforward pruning of database objects whose directory entries are at more than ε distance (with respect to a given distance function). This scenario also applies for k nearest neighbor queries [SK98].

In the Bayes tree, the data objects are stored at leaf level as in similarity search applications. As classification requires reading all kernel estimators of the entire model, accuracy would be lost if a subset of all kernel densities was ignored. Consequently, there is no irrelevant data, and hence the pruning as in similarity search is infeasible when dealing with density estimation. As for the accuracy increase, accessing the entire kernel density model is not only inefficient but also not interruptible. Moreover, kernel densities provide only a single model of the data, i.e. no incremental improvement of classification is possible as required for anytime classification.

As illustrated in Figure 4.1 (bottom left), the upper levels of an index that supports anytime classification need to provide information that can be used for assigning class labels even before the leaf level containing the kernels is reached. The Bayes tree solves this problem by storing aggregated statistical information in its inner nodes (Figure 4.1 right).

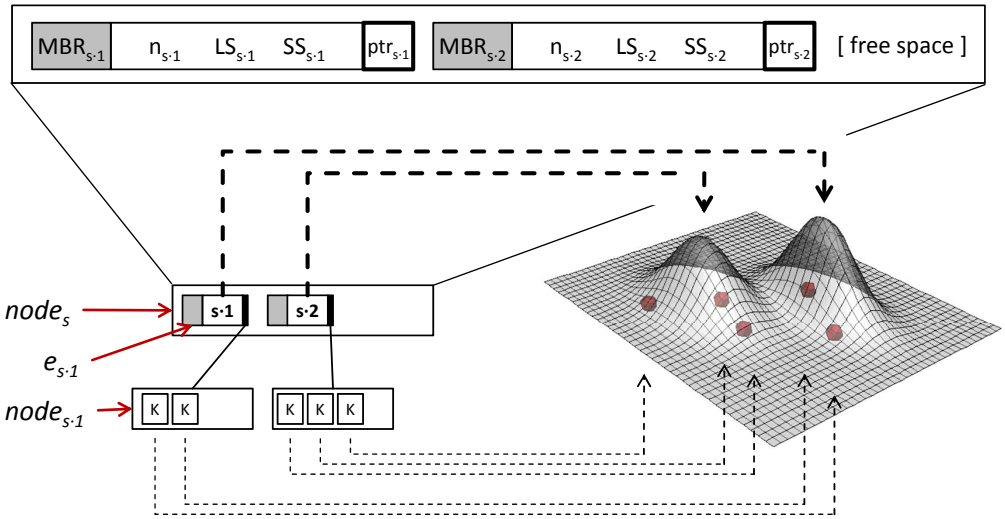


Figure 4.2: Nodes and entries in the Bayes tree (here: above the kernel level). Entries e_s store minimum bounding rectangles (MBR$_s$), child pointers (ptr$_s$), and additional information to compute mean and variance, i.e. number of objects (n_s) as well as their linear and quadratic sums per dimension. Kernel positions are depicted as dots, the higher level mixture density is represented as density curves.

The proposed approach is therefore a hierarchy of mixture models built on top of kernel densities. It provides a classification decision on interruption and allows for adaptive query-based refinement as long as more time is available.

4.2.2 Structure of the Bayes tree

In this section the Bayes tree structure is described for a single class l_i. Query processing and refinement on multiple classes is discussed in Section 4.2.3.

The Bayes tree builds on top of the R-tree and the CF-tree. Each entry is associated with a Gaussian that represents the data in its corresponding subtree. A cluster feature vector according to Definition 3.2 is stored per entry. This way the entries can be updated incrementally (cf. Theorem 3.1) and the parameters μ and σ of the Gaussian can be computed easily.

Definition 4.2 ***Bayes tree node entry.*** *A subtree T_s of a d-dimensional Bayes tree is associated with the set of objects stored in the leaves of the subtree: $T_s = \{o_{(s,1)}, \ldots, o_{(s,n_s)}\}$. An entry e_s then stores the following information about the subtree T_s:*

- *The **minimum bounding rectangle** enclosing the objects stored in the subtree T_s as $MBR_s = ((l_1, u_1), \ldots, (l_d, u_d))$*
- *A **pointer** ptr_s to the subtree T_s*
- *The **number** n_s of objects in T_s*
- *The vector LS_s containing the linear sum of all objects per dimension*
- *The vector SS_s containing the squared sum of all objects per dimension*

All objects stored in the leaves of the Bayes tree are d-dimensional kernels. Figure 4.2 illustrates the structure of a Bayes tree node entry. From the information stored in each entry according to Definition 4.2, mean and variance are computed according to Equations 3.1 and 3.2.

The Bayes tree extends the R*-tree to store model specific information in the following manner:

Definition 4.3 ***Bayes tree.*** *A Bayes tree with fanout parameters $\mathbf{m}$ and $\mathbf{M}$ and leaf node capacity parameters $\mathbf{l}$ and $\mathbf{L}$ is a balanced multidimensional indexing structure with the following properties:*

- *Each inner node $node_s$ contains between $\mathbf{m}$ and $\mathbf{M}$ entries (see Def. 4.2). The root has at least a single entry.*
- *Each inner node with ν_s entries has exactly ν_s child nodes*
- *Leaf nodes store between $\mathbf{l}$ and $\mathbf{L}$ observations (d-dimensional kernels).*
- *A path from the root to any leaf node has always the same length (balanced).*

This structure has some very nice and intuitive benefits: since the number of objects, their linear sum as well as their quadratic sum are all distributive measures, they can efficiently be computed bottom up. More precisely, we can use the build procedure of any standard R*-tree to create the hierarchy of mixture densities. Using an index structure from the R-tree family automatically allows both processing of huge amounts of data and incremental learning at any time.

Recall that R*-trees, or any other tree from the B-tree or R-tree family, grow in a bottom-up fashion. Whenever there are more entries assigned to any node than allowed by the fanout parameters $\mathbf{M}$, which in turn reflects the page size on hard disk, an overflow occurs. This overflow leads to a node split, i.e. the entries of this overfull node are separated onto two new nodes. Their ancestor node then stores aggregate information on these two nodes. In the R*-tree case, this was simply the MBR of the two nodes. In addition to that, for the Bayes tree the overall number, linear and quadratic sum must be computed and stored in the ancestor node. This way, in a very straightforward manner, coarser mixture density models are created.

Consequently, building Bayes trees is a simple procedure that starts from the original kernel density estimators that are stored in a leaf node until a split is necessary. This first split leads to creation of a second level in the tree, with a coarser mixture density model derived through R*-tree-style split [BKSS90]. Successive splitting of nodes (as more kernel densities are inserted) leads to further growth of the tree, eventually yielding a hierarchy of mixture density models as coarser representations of the kernel densities.

4.2.3 Query processing

Answering a probability density query requires a complete model as stored at each level of the tree. Besides these full models, local refinement of the model (to adapt flexibly to the query) provides models composed of coarser and finer representations. This is illustrated in Figure 4.3 on page 83. In any model, each component corresponds to an entry that represents its subtree. This entry may be replaced by the entries in its child node yielding a finer

representation of its subtree. This idea leads to query-based refinement in the proposed anytime algorithm.

A frontier is a model representation that consists of node entries such that each kernel estimator contributes and such that no kernel estimator is represented redundantly. To formalize the notion of a frontier, we need a clear definition for the enumeration of nodes and entries.

Definition 4.4 ***Enumeration of nodes and entries.*** *To refer to each entry stored in the Bayes tree we use a label s with the following properties:*

- *The initial starting point is labeled $e_\emptyset$ with $T_\emptyset$ being the complete set of objects stored in the Bayes tree.*
- *The child node of an entry e_s has the same label s as its parent entry, i.e. the child of e_s is $node_s$. T_s denotes the set of objects stored in the respective subtree of e_s.*
- *The ν_s entries stored in $node_s$ are labeled $\{e_{s\circ 1} \dots e_{s\circ \nu_s}\}$, i.e. the label s of the predecessor concatenated with the entry's number in the node. (recall Figure 4.2: $node_s$ has two entries $e_{s\circ 1}$ and $e_{s\circ 2}$ with their child nodes $node_{s\circ 1}$ and $node_{s\circ 2}$).*

A set $\mathcal{E} = \{e_i\}$ of entries defines a Gaussian mixture model GMM according to Definition 2.3, which can then be used to answer a probability density query.

Definition 4.5 ***Probability density query pdq.*** *Let $\mathcal{E} = \{e_i\}$ be a set of entries, $GMM_\mathcal{E}$ the corresponding Gaussian mixture model and $\mathbf{n} = \sum_i n_{e_i}$ the total number of objects represented by $\mathcal{E}$. A probability density query pdq returns the density for an object q with respect to $GMM_\mathcal{E}$ by*

$$pdq(q, \mathcal{E}) = \sum_{e_s \in \mathcal{E}} \frac{n_{e_s}}{\mathbf{n}} \cdot \boldsymbol{g}(q, \mu_{e_s}, \sigma_{e_s})$$

where μ_{e_s} and σ_{e_s} are calculated according to Equations 3.1 and 3.2.

For a leaf entry a kernel estimator as discussed in Section 4.1 is used and obviously μ_{e_s} is the object itself.

Figure 4.3 b) shows the resulting mixture density for the example frontier from part a). The leftmost Gaussian stems from the entry e_1 which is located at root level. The rightmost Gaussian and the one in the back correspond to entries e_{23} and e_{21} respectively, the remaining represent kernel densities at leaf level. Part c) of the image depicts the underlying R*-tree MBRs and the kernels as dots. The bigger blue dot and the vertical line represent the query object from which the above frontier originated.

Using the above enumeration a frontier can be derived from any prefix-closed set of nodes.

Definition 4.6 ***Prefix-closed subset and frontier.*** *A set of nodes $\mathcal{N}$ is prefix-closed iff:*

- $node_\emptyset \in \mathcal{N}$ *(the root is always contained in a prefix-closed subset)*
- $node_{s \circ i} \in \mathcal{N} \Rightarrow node_s \in \mathcal{N}$ *(if a node is contained in $\mathcal{N}$, its parent is also contained in $\mathcal{N}$)*

Let $\mathcal{E}(\mathcal{N})$ be the set of entries stored in the nodes that are contained in a prefix-closed node set $\mathcal{N}$: $\mathcal{E}(\mathcal{N}) = \bigcup_{node \in \mathcal{N}} (e \in node)$. The frontier $\mathcal{F}(\mathcal{N}) \subseteq \mathcal{E}(\mathcal{N})$ contains all entries $e \in \mathcal{E}(\mathcal{N})$ that satisfy

- $e_{s \circ i} \in \mathcal{F}(\mathcal{N}) \Rightarrow e_s \notin \mathcal{F}(\mathcal{N})$*: no predecessors of a frontier entry is in $\mathcal{F}(\mathcal{N})$* ***(predecessor-free)***
- $e_s \in \mathcal{F}(\mathcal{N}) \Rightarrow \nexists i \in \mathbb{N}$ *with $e_{s \circ i} \in \mathcal{E}(\mathcal{N})$: a frontier entry has no successors in $\mathcal{E}(\mathcal{N})$* ***(successor-free)***

Figure 4.3 a) illustrates both a prefix-closed subset of nodes and the corresponding frontier. The subset, separated by the curved red line, contains the root, $node_2$ and $node_{23}$, hence it is prefix-closed. Its corresponding frontier (highlighted in white) contains the entries e_1, e_{21}, e_{23}, e_{221}, e_{222} and e_{223} where the last three represent kernels. The frontier is both predecessor-free and successor-free.

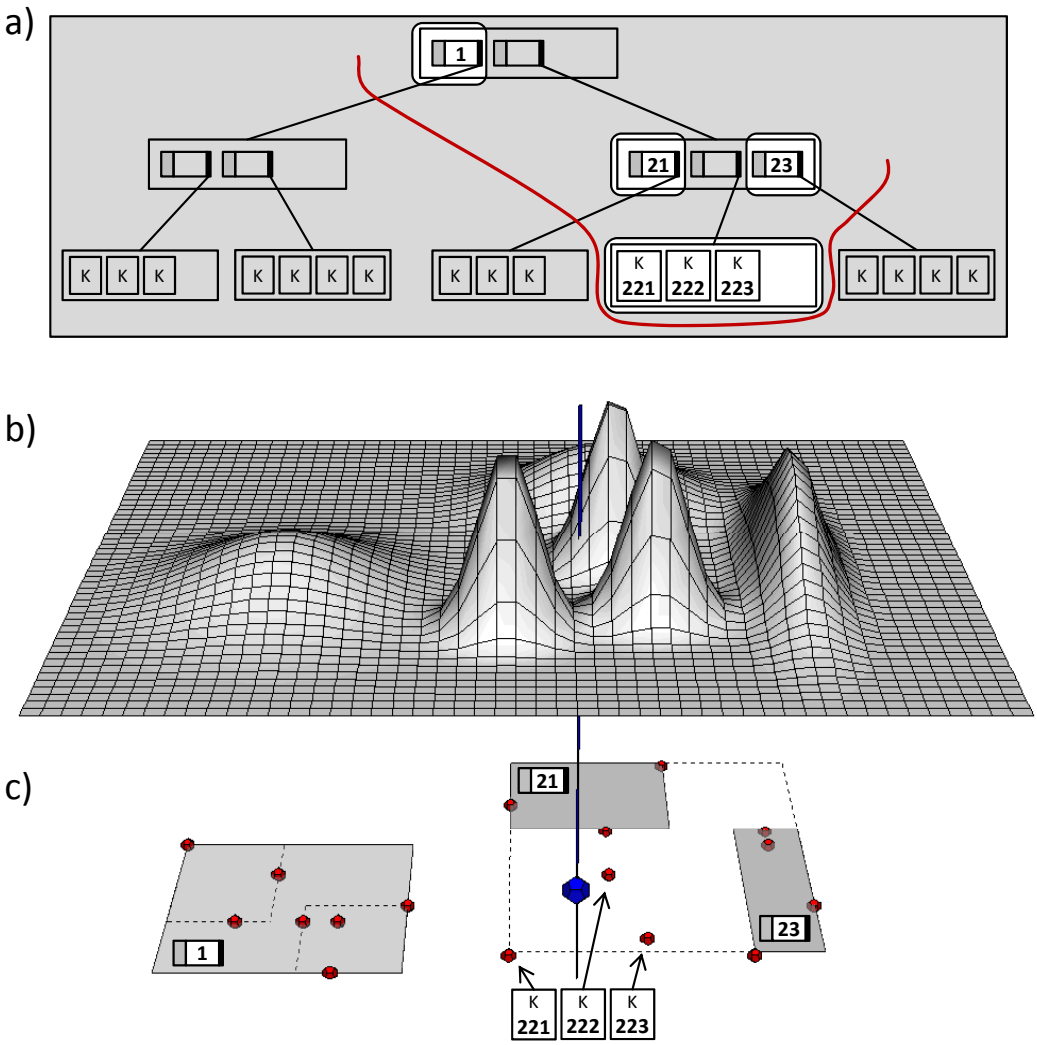


Figure 4.3: Tree frontier: In a) a frontier in a Bayes tree that corresponds to a model with mixed levels of granularity is depicted: nodes 2 (root level) and 22 (inner level) are refined, yielding 1 (root level), 21 (inner level), 221, 222, 223 (all leaf level), 23 (inner level). The resulting mixture density model is depicted in b), the actual kernel positions and the hierarchy are depicted in c).

Using the above definitions the following defines the processing of a probability density query on the Bayes tree.

Let $T_\emptyset$ be the set of objects stored in a Bayes tree containing t_{max} nodes. Anytime *pdq* processing for an object q processes a prefix-closed subset $\mathcal{N}_t$ in each time step t with $\mathcal{N}_0 = \{node_\emptyset\}$, i.e. the processing starts with the root node, and $|\mathcal{N}_{t+1}| = |\mathcal{N}_t| + 1$, i.e. in each time step one more node is read. The probability density for the query object q at time step t is then calculated using the mixture model corresponding to the frontier $\mathcal{F}(\mathcal{N}_t)$ of the current prefix-closed subset $\mathcal{N}_t$ as in Definition 4.5, that is $pdq(q, \mathcal{F}(\mathcal{N}_t))$. This way all objects of the tree are accounted for: $\sum_{e_s \in \mathcal{F}(\mathcal{N}_t)} n_{e_s} = |T_\emptyset|$.

Definition 4.7 ***Anytime pdq processing.*** *From time step t to $t+1$, $\mathcal{N}_t$ becomes $\mathcal{N}_{t+1}$ by adding the child node $node_s$ of one frontier entry $e_s \in \mathcal{F}(\mathcal{N}_t)$. If $node_s$ has ν_s entries, then the frontier $\mathcal{F}(\mathcal{N}_t)$ changes to $\mathcal{F}(\mathcal{N}_{t+1})$ by*

$$\mathcal{F}(\mathcal{N}_{t+1}) = (\mathcal{F}(\mathcal{N}_t) \setminus \{e_s\}) \cup \{e_{s\circ 1}, \ldots, e_{s\circ \nu_s}\}$$

i.e. e_s is replaced by its children. The probability density for q in time step $t+1$ is then calculated by

$$\begin{aligned} pdq(q, \mathcal{F}(\mathcal{N}_{t+1})) \;=\; & pdq(q, \mathcal{F}(\mathcal{N}_t)) \\ & - \frac{n_s}{|T_\emptyset|} \cdot \boldsymbol{g}(q, \mu_{e_s}, \sigma_{e_s}) \\ & + \sum_{i=1}^{\nu_s} \frac{n_{s\circ i}}{|T_\emptyset|} \cdot \boldsymbol{g}(q, \mu_{e_{s\circ i}}, \sigma_{e_{s\circ i}}) \end{aligned}$$

Following the above equation, the probability density for q in time step $t+1$ is calculated taking the probability density for q in time step t (row 1), subtracting the contribution of the refined entry's Gaussian (row 2) and adding the contributions of its children's Gaussians (row 3). Hence, the cost for calculating the new probability density for q after reading one additional node is very low due to the information stored for mean and variance. After that the entry has to be sorted into the frontier, which can be done in $O(log_2^2(r \cdot \mathbf{M}))$ using a heap (after r refinements the frontier contains maximally $r \cdot \mathbf{M}$ entries).

Note that after reading all t_{max} nodes $pdq(q, \mathcal{F}(\mathcal{N}_{t_{max}}))$ is a full kernel density estimation taking all kernels at leaf level into account.

Each possible frontier represents a possible model for the anytime algorithm. The number of possible models in any Bayes tree depends on the actual status of the tree, i.e. the degree to which it is filled and the height of the tree. For example, there are four possible models for a Bayes tree of height two and fanout two (root, leaves, and two mixed models). For realistic trees, with increasing height and fanout, the number of possible models is exponential in the height and in the fanout.

Choosing among all these models is crucial for anytime algorithms, where the available time (typically unknown a priori) should be spent such that the most promising model information is used first. For tree traversal three basic descent strategies are proposed to answer probability density queries on a Bayes tree. Descent in a breadth first (*bft*) fashion refines each level completely before descending down to the next level. Alternatively, descending the tree in a depth first (*dft*) manner refines a single subtree entirely down to the actual kernel estimators before refining the next subtree below the root. The choice of the subtree to refine is made according to a priority measure. The third approach, which is called global best first (*glo*), orders nodes globally with respect to a priority measure and refines nodes in this ordering.

The priority measure in R-Trees, denoted as *geometric*, gives highest priority to the node with the smallest (Euclidean) distance based on its minimum bounding rectangle.

In the Bayes tree, the focus is not on distances, but on density estimation. The proposed *probabilistic* priority measure awards priority with respect to the actual density value for the current query object based on the statistical information stored in each node. More specifically, at time step t, having read the prefix-closed set of nodes $\mathcal{N}_t$, the probabilistic approach descends at the next time step $t+1$ into the subtree $\mathbf{T}_{\hat{\mathbf{s}}}$ belonging to the entry $e_{\hat{s}}$ with

$$e_{\hat{s}} = \underset{e_s \in \mathcal{F}(\mathcal{N}_t)}{\arg\max} \left\{ \mathbf{g}(q, \mu_{e_s}, \sigma_{e_s}) \cdot \frac{n_s}{\mathbf{n}} \right\}$$

where q is the current query object and $\mathbf{g}$ is a Gaussian pdf as in Def. 2.3.

In Section 4.3 all combinations between the three descent strategies and the two priority measures are evaluated, where in the breadth first approach the priority measure determines the order of nodes per level.

The described descent strategies work on a single Bayes tree; they refine mixture models for just one class. In the following refinement with respect to several classes $\mathcal{L} = \{l_1, \ldots, l_{|\mathcal{L}|}\}$ is discussed. The time for classification is divided between several Bayes trees, one per class (cf. Section 4.2.2). The question is, having read x nodes so far, i.e. being at time step t_x, which of the $|\mathcal{L}|$ trees should be granted the right to read the next node in the following time step $t_x + 1$?

Initially, the coarsest model for each class l_i is evaluated, which is the model consisting of only one Gaussian probability density function representing all objects of l_i.

A naive improvement strategy chooses the classes in an arbitrary order $l_1, \ldots, l_{|\mathcal{L}|}$. First, the class l_1 is refined according to one of the above strategies. In the next step, l_2 is refined and so on. After $|\mathcal{L}|$ time steps all classes have been refined and the Bayes tree of the first class is processed again. As the classes are not sorted in any predefined way, this method is referred to as *unsorted*.

Intuitively, it is very likely that for an object q the true class l_i shows a high posterior probability independent of the time t_{l_i} spent descending its corresponding tree:

$$P(l_i|q) = pdq(q, \mathcal{F}(\mathcal{N}_{t_{l_i}})) \cdot P(l_i)/p(q).$$

Consequently, a second strategy for improving classification is suggested which only chooses the k classes having the highest a posteriori probability. This technique stores these k classes in a priority queue (*queue best* k). Assume the sorted priority queue contains the classes $(l_{i_1}, \ldots, l_{i_k})$. The next node is read in the Bayes tree of class l_{i_1} and the probability model is updated accordingly. If more time is permitted class l_{i_2} is refined and so on. After k steps all k classes have been improved and the queue is filled again using the new a posteriori probabilities. The influence of the parameter k is evaluated in the experiment section. Setting $k = 1$, always the model of the most prob-

able class is refined. For $k = m$ (assuming $m = |\mathcal{L}|$ in the rest of this chapter) all classes are refined in the order of their probability before the queue is refilled. The approaches are referred to as *qb1* and *qbm*, respectively, and *qbk* in general.

One observation which is also known from hierarchical indexing is that the root node often only contains a few entries. This free space is used for storing the entries from different classes on the same page if it does not exceed the page size. These new root pages are called the *compressed root*. The total size of the compressed root varies since the number of root entries in a single Bayes tree varies as in all index structures. Clearly, the total size also grows linearly in the number of dimensions and the number of classes.

4.3 Experiments

The experiments are performed on real world and synthetic data and evaluate the Bayes tree for both anytime classification and anytime learning. Table 4.1 summarizes the data sets and their characteristics. The synthetic data sets were generated using a random Gaussian mixture model as in [HTF02]. A small data set containing 4 dimensions, 2 classes and eleven thousand objects and one large data set containing 5 dimensions, 3 classes and one million objects are tested. Furthermore, real world data of various characteristics is tested which is taken from different sources such as the UCI KDD archive [HB99].

All experiments were performed using 4 fold cross validation. To evaluate the anytime performance the accuracy (averaged over the four folds) is reported after each node that is read (corresponding to one page). The first accuracy value corresponds to the classification accuracy of the unimodal Bayes classifier that is used as an initialization before reading the compressed root (cf. Section 4.3.1). Experiments were run using 2KB page sizes (except 4KB for Verbmobil and 8KB for the USPS data set) on Pentium 4 machines with 2.4 Ghz and 1 GB main memory. The bandwidth for the kernel estimators is set using a common method according to [Sil86].

name	size	classes	features	ref.
Synthetic (11K)	11000	2	4	
Synthetic (1M)	1000000	3	5	
Vowel	990	11	10	[HB99]
USPS	8772	10	39	[HTF08]
Pendigits	10992	10	16	[HB99]
Letter	20000	26	16	[HB99]
Gender	189961	2	9	[AS04]
Covtype	581012	7	10	[HB99]
Verbmobil	647585	5	32	[Wah00]

Table 4.1: Data sets used in the experiments.

The following first evaluates the descent strategies along with the priority measures introduced in Section 4.2.3. In Section 4.3.2 the various improvement strategies are analyzed to find that none of the three simple approaches leads to competitive results. After that a novel evaluation method is introduced for anytime classifiers on Poisson streams in Section 4.3.3. Finally the anytime learning performance of the Bayes tree is demonstrated for both anytime accuracy and stream accuracy.

4.3.1 Descent strategies

For tree traversal depth first (*dft*), breadth first (*bft*) and global best first (*glo*) strategies are evaluated. Combined with both the geometric (*geom*) and probabilistic (*prob*) priority measure six approaches are compared for model refinement in a Bayes tree. Figure 4.4 shows the results for all six approaches on the small and on the large synthetic data set using the *unsorted* improvement strategy. After 5 pages on the small synthetic data set the locality assumption of the depth first approach fails to identify the most relevant refinements for both geometric and probabilistic priority. For the large synthetic data set the accuracy even decreases after 17 pages. The processing strategy of the breadth first approach causes the classification accuracy to only increase if a next level of the tree is reached.

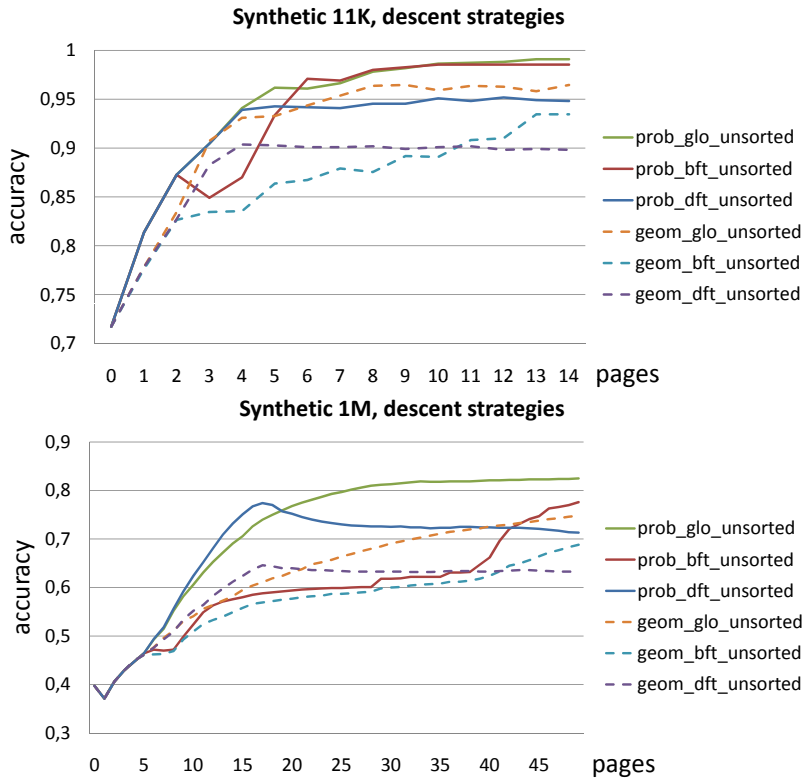


Figure 4.4: Comparing descent strategies along with priority measures on synthetic data.

The results suggest that the performance of the different descent strategies is independent of the employed improvement strategy (i.e. *unsorted* or *qbk*). As an example for the independence of the improvement strategy Figure 4.5 shows the performance on the Gender data set using *qbm*. The *glo* descent strategy with probabilistic priority measure shows the overall best anytime classification accuracy for all data sets. Further experiments use the global best first descent and the probabilistic priority measure.

4.3.2 Improvement strategies

First the improvement strategies *unsorted*, *qb1* and *qbm* are tested on all seven real world data sets from Table 4.1. Besides that *qb3* is used for the first evaluation (setting k to 3, 2 for Gender, respectively) as it can be seen

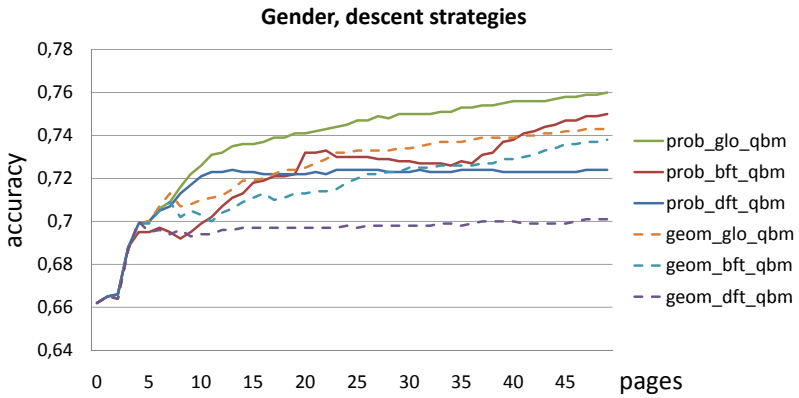


Figure 4.5: Comparing descent strategies along with priority measures on the Gender data set.

as corresponding to ”some relevant candidates”. (A thorough evaluation of various values for k follows later in this section.)

The results from Pendigits and Covtype as represent the two sorts of outcome that were found when analyzing the results (cf. Figures 4.6 and 4.7). The naive approach, i.e. the unsorted improvement strategy, does not provide a competitive solution (third rank on Pendigits and last rank on Covtype). However, after reading the entire compressed root this approach has read the same information as *qbm*. This can best be seen on the results for Covtype. After four page accesses the compressed root is read and both strategies deliver the same classification accuracy. From then on, their accuracy is the same after every seven steps, i.e. 11, 18, 25, ... since Covtype has seven classes. The graph for Pendigits does not show this behavior in such a clear fashion. This is due to the averaging over the four folds. The compressed root needs between seven and eight pages within each fold. Therefore the accuracy for *unsorted* and *qbm* is similar after seven to eight steps, yet not equal. This similarity occurs every ten steps from then on, since Pendigits has ten classes.

Regarding the other improvement strategies, one could expect that *qb1*, i.e. refining only the most probable class, delivers the best results, because it either strengthens the current decision by increasing the probability for the correct class or corrects the decision by decreasing the probability in case

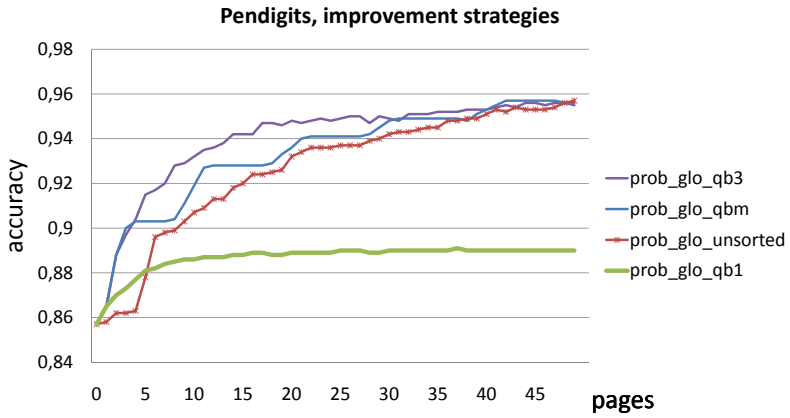


Figure 4.6: Comparing improvement strategies on Pendigits.

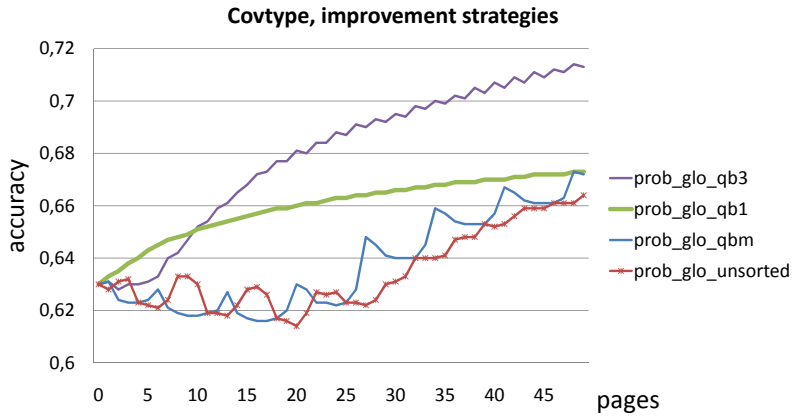


Figure 4.7: Comparing improvement strategies on Covtype.

a false class has currently the highest density. On Covtype *qb1* outperforms the *qbm* and *unsorted* strategies, but on Pendigits it performs way worse then both of them. Opposite success and failure would befall those in favor for *qbm*.

The results on Verbmobil and Gender were similar to those on Pendigits, i.e. *qb1* performed way worse than all other strategies. On the other hand, similar to the Covtype results, *qb1* performed better than *qbm* and *unsorted* on Letter and USPS. On all datasets all of the three strategies discussed above were clearly outperformed by the *qb3* approach.

These results prove that anytime density estimation does not extend in a naive or straightforward way to anytime classification with competitive re-

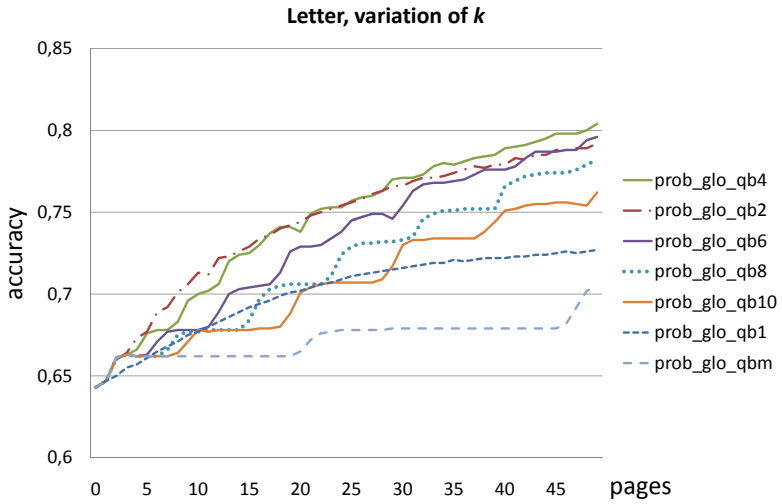


Figure 4.8: Variation of k on the Letter data set.

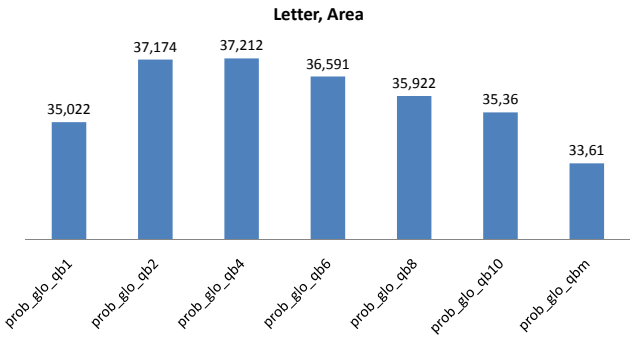


Figure 4.9: The area under the corresponding anytime curves in Figure 4.8.

sults. Moreover, the results pose the question, how the choice of k influences the anytime classification performance. Figure 4.8 shows the comparison of *qb1*, *qbm* and *qbk* for $k \in \{2, 4, 6, 8, 10\}$ on the Letter data set. As stated above, *qb1* performs better than *qbm* on this data set.

The accuracy of the *qbm* strategy in Figure 4.8 initially increases steeply in the first three to four steps. It is clearly visible that the compressed root needs on average 19 pages for the 26 classes (recall the 4 fold cross validation). After the compressed root is read, the accuracy of the *qbm* strategy again steeply increases for the next four to five steps. A similar increase can be observed 26 steps later when all classes have been refined once more.

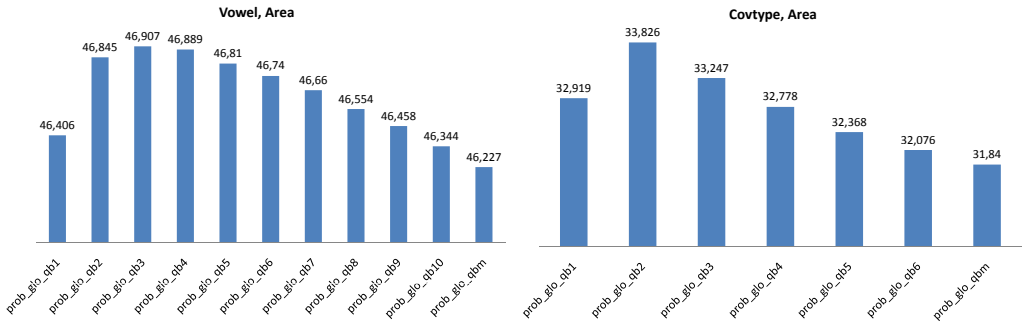


Figure 4.10: Area values for $k \in \{1 \dots m\}$ for Vowel (left) and Covtype (right).

Opposed to those three strong improvements are the rest of the steps, during which the accuracy hardly changes at all. For this data set we derive from Figure 4.8 that for an object to be classified there are on average five classes that are worth considering, the other 21 classes can be excluded.

The *qb10* strategy needs seven to eight pages to read the root information for the top ten classes from the compressed root. These steps show the same accuracy as the *qbm* approach, since the ordering according to the priority measure is equal. After that a similar pattern can be observed each ten steps, i.e. after reordering according to the novel density values.

Continuing with the *qb8* variant, the "ramps" (steep improvements) move even closer together and begin earlier. Similar improvement gains are shown by *qb6* and *qb4*. An even smaller k ($k = 2$ in Figure 4.8) does no longer improve the overall anytime accuracy and *qb1* shows an even worse performance than *qb10*.

For an easier comparison of the *qbk* performance when varying k, the area under the anytime curve is calculated for each k yielding a single number for each value of k. Since *qbk* curves mostly ascend monotonically, a larger area is an indication for a better anytime performance. Figure 4.9 shows the area values corresponding to the anytime curves in Figure 4.8. As in the above analysis *qb4* turns out to perform best according to this measure.

Evaluating *qbk* for $k \in \{1 \dots m\}$ on all data sets showed the same characteristic results. Figure 4.10 displays the results for Vowel and Covtype. The

results show that the maximum value is always between $k = 2$ and $k = 4$ on the data sets tested. Moreover, with an increasing number of classes, the k value for the best performance also increased, yet only slightly. An exception is the Gender data set, where $k = m = 2$ showed the best performance. This is due to the bad performance of $qb1$, therefore k is set to be at least 2. To develop a heuristic for the value of k experiments were performed on different data sets having different number of classes. It turned out that a good choice for k is logarithmic in the number of classes, i.e. $k = \lfloor \log_2(m) \rfloor$. Using this heuristic met the maximal performance for all the evaluations. Yet the minimum value for k is set to 2 as mentioned above. This improvement strategy is used in all further experiments along with the best descent strategy from Section 4.3.1.

4.3.3 Evaluating anytime classification using Poisson streams

So far we determined the best strategy for descending a Bayes tree to refine the density model and evaluated the proposed improvement strategies. Next a novel evaluation method is introduced for anytime classifiers using Poisson streams. Finally the anytime learning performance of the Bayes tree is demonstrated on various data streams.

To evaluate anytime classification under variable stream scenarios, the proposed evaluation method recapitulates a stochastic model that is widely used to model random arrivals [DHS01]. A Poisson process describes streams where the inter-arrival times are independently exponentially distributed. Poisson processes are parameterized by an arrival rate parameter λ:

Definition 4.8 ***Poisson stream.*** *A probability density function for the inter-arrival time of a Poisson process is exponentially distributed with parameter λ by $p(t) = \lambda \cdot e^{-\lambda t}$. The expected inter-arrival time of an exponentially distributed random variable with parameter λ is $E[t] = \frac{1}{\lambda}$.*

A Poisson process is used to model random stream arrivals for each fold of the cross validation. The method randomly generates exponentially dis-

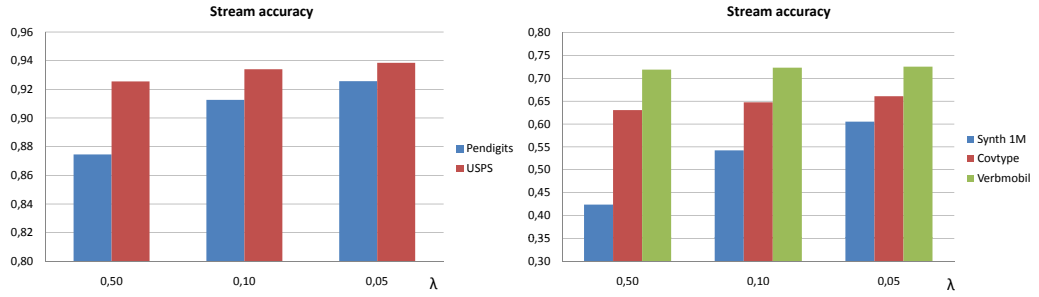


Figure 4.11: Stream classification accuracy using Poisson processes for data set Pendigits and USPS (left) and Synthetic 1M, Covtype and Verbmobil (right).

tributed inter-arrival times for different values of λ. If a new object arrives (the time between two objects has passed) the anytime classifier is interrupted and its accuracy is measured. This experiment is repeated using different expected inter-arrival times $\frac{1}{\lambda}$, where a unit corresponds to a page as in all previous experiments. It is assumed that any object arrives at the earliest after the initialization phase of the previous object, i.e. after evaluating the unimodal Bayes.

Figure 4.11 shows the results for the stream classification accuracy as described above for different values of λ on Pendigits, USPS, Covtype, Verbmobil and the large synthetic data set. All data sets show an improvement of accuracy as the expected inter-arrival time $1/\lambda$ increases and arrival times are distributed following a Poisson process. This is an excellent feature that contrasts anytime classifiers from budget or contract classifiers. Budget classifiers would be forced to use a time budget that is small enough to guarantee the processing of all arriving objects which would clearly result in worse performance. Moreover, evaluating anytime classifiers using Poisson processes yields a single accuracy value for each λ making it very easy to compare performance on different underlying streams.

4.3.4 Incremental learning

Incremental learning in stream classification applications originates from labeled objects that newly arrive in the data stream. If the Bayes tree started

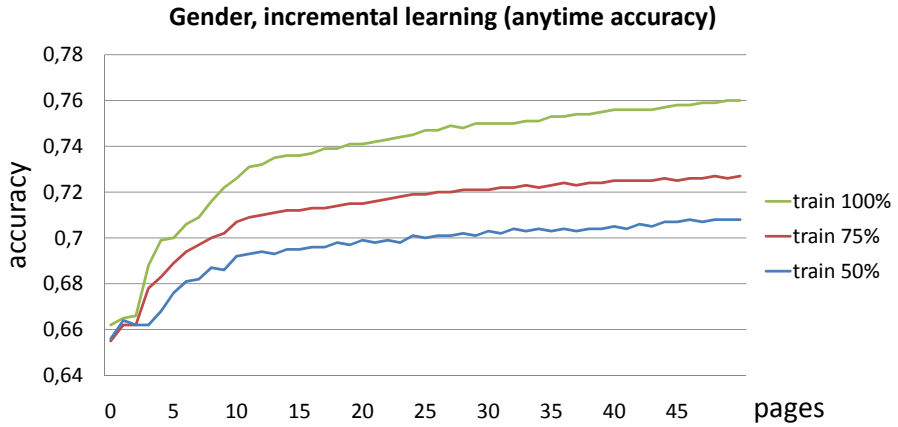


Figure 4.12: Incremental learning on Gender.

learning/inserting a new object, it always finishes the insertion for that object; no roll back will be performed. However, if the average inter-arrival time between two object is smaller than the time necessary for learning an additional object, the classifier cannot process all labeled objects online. Hence, the resulting classifier is based on a varying amount of training data depending on the stream speed, i.e. λ as in Definition 4.8. To evaluate the incremental learning performance of the Bayes tree different λ values are tested such that the classifier is only able to process a certain percentage of the training data. The results for 50%, 75% and 100% are reported in the following. First the anytime classification accuracy for the three classifiers is evaluated. To be able to compare the results, the classifiers classify each the same stream of test objects. As above the results are averages over 4 folds.

Figure 4.12 shows the resulting anytime accuracy of the three incremental learning classifiers on the Gender data set. Even the classifier trained on the fastest stream (train 50%) shows a good performance. On slower streams, the Bayes tree can exploit its incremental learning property and learn more objects before the training phase is over. The resulting anytime accuracy (train 75%) lies constantly above train 50% from page three onwards. The Bayes tree that was trained on the slowest stream (train 100%) performs consistently better then both the others, again highlighting the benefit of the incremental learning property.

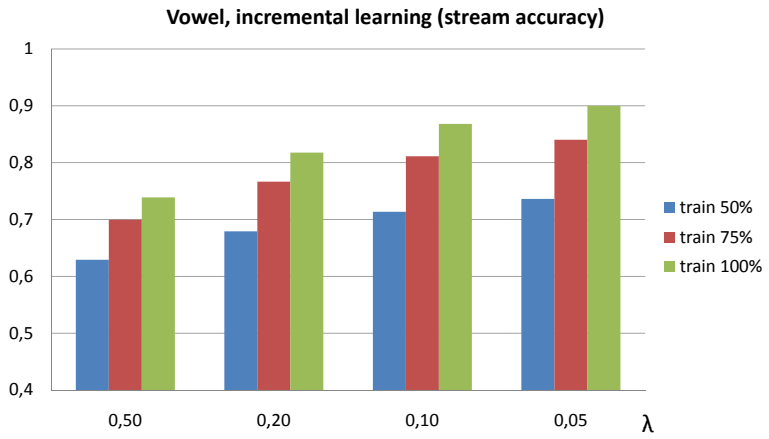


Figure 4.13: Incremental learning on Vowel.

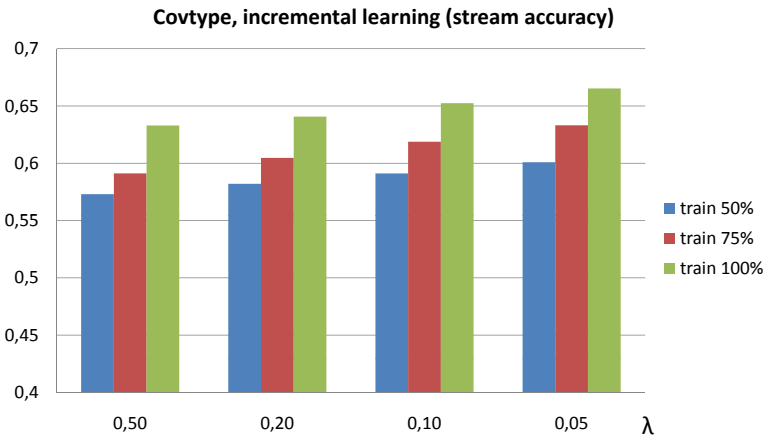


Figure 4.14: Incremental learning on Covtype.

Finally the incrementally trained Bayes trees are evaluated on Poisson streams for various λ. Figures 4.13 and 4.14 show the resulting stream accuracy values for Vowel and Covtype respectively. Again the same underlying stream data is used during classification to be able to compare the results. The anytime learning performance can be seen within each group of three bars, where immediate comparison of the classifiers is possible through the Poisson stream evaluation. For all of the evaluated λ values and data set the classifiers trained on slower streams perform better. Moreover, with smaller λ values during classification, the stream accuracy of each individual Bayes tree improves.

The evaluation shows that the Bayes tree efficiently supports incremental learning as well as anytime classification. Moreover, the underlying index structure ensures the ability to handle very large data sets as in the case of streaming data.

4.4 Conclusion

In this chapter a novel index-based classifier was proposed called Bayes tree. It supports anytime learning and anytime classification and can handle huge amounts of data, which makes it a consistent solution for classification on fast data streams. The Bayes tree constitutes a hierarchy of mixture densities that represent kernel estimators at successively coarser levels. The proposed probability density queries adapt the employed mixtures efficiently to the individual object to be classified. The time complexitiy of the r-th refinement is in $O(\log_2^2(r \cdot M))$, where M is the maximal fanout. Together with novel classification improvement strategies this allows for very effective classification at any point of interruption. Moreover, the anytime learning performance of the proposed classifier was demonstrated and a novel evaluation method for anytime classification was investigated using Poisson streams.

The following three chapters concentrate on improving the Bayes tree by different offline learning methods. The applicability to evolving data streams with changing distributions and new labeled data is given by combining the classifier with general approaches such as [WFYH03] as discussed in Chapter 3.

Chapter 5

The MC-Tree

* In this Chapter an approach to Bayesian anytime classification is presented that is called MC-Tree. As the Bayes tree it can provide a very fast first result after evaluating just one Gaussian normal distribution per class at the root level and it can improve the classification accuracy as long as time permits by refining its current model incrementally. On the finest level a kernel density estimator is evaluated for each object in the training set. In between, the MC-Tree stores a hierarchy of mixture models that allows effective and query adaptive anytime density estimation.

In contrast to the Bayes tree the mixture components contain objects from several (potentially all) classes. A top down approach for the tree construction is proposed and data transformations are used that take both the locality of the data and the class distribution into account. The proposed descent strategies, which exploit the entropy information available through the MC-Tree, achieve parallel model refinement for several classes. The experiments confirm the effectiveness of the MC-Tree and show significant improvements over previous approaches in terms of anytime classification accuracy.

*This chapter has been published in the Proceedings of the 22nd International Conference on Scientific and Statistical Database Management (SSDBM 2010) [KGFS10].

5.1 Combining Multiple Classes

In the following, firstly the structural differences of the MC-Tree and the novel construction approach are described in Section 5.1.1. The classification process with respect to the multi-class structure is presented in Section 5.1.2, classification refinement is discussed in Section 5.1.3.

5.1.1 Tree structure and construction

As in the Bayes tree the general idea of the Multi-Class Tree (MC-Tree) is to store a hierarchy of mixture densities. Figure 5.1 shows an MC-tree structure with three levels. The root node of the tree consists of two entries and represents the coarsest data model. The gray Gaussians in these entries represent the aggregated statistical information of the points in the subtrees. They are constructed from the Gaussians in the level below. For better illustration these lower-level components are shown once again in the entries of the root node but are actually not stored twice. The Gaussians of the first level on their part are constructed from kernel based Gaussian components in the leaves. As one can see at the left most entry in level 1, the MC-Tree permits to represent objects from different classes (A and B) in one Gaussian component. This is beneficial if the spatial similarity of the underlying objects is high and thus the model remains compact. Gaussians that comprise objects from several classes are represented by dotted lines in the figure. Furthermore, unbalanced MC-Trees are possible. If an area of the data space needs a more detailed representation, paths can become longer than for other areas.

Two types of entries are developed and analyzed for the MC-Tree. The first version separately stores the necessary information for calculating the mean and variance for each class contained in an entry. Thus if $O(e, l)$ is the set of objects from class l in the entry e we can derive the corresponding Gaussian. Furthermore we can calculate the mean and variance of the overall entry independent of the class information.

Definition 5.1 ***MC-Tree node entry (type A).*** *Let $O(e)$ be the objects that are represented by an entry e of the MC-Tree and $O(e, l) \subseteq O(e)$ the objects belonging to class $l \in \mathcal{L}$. An entry e of type A then stores the following information:*

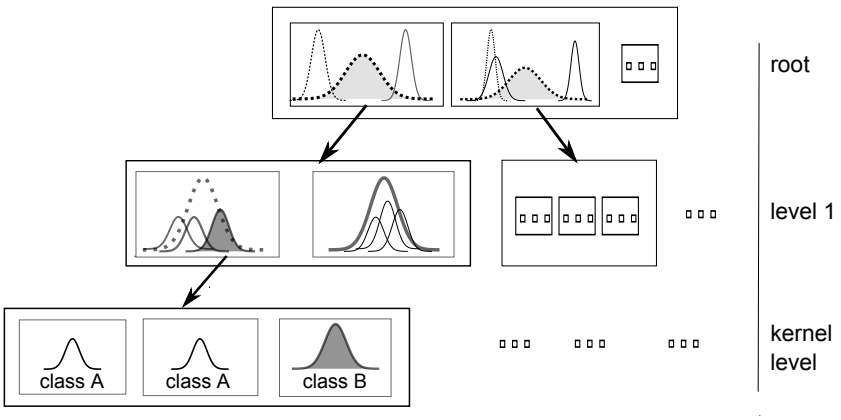


Figure 5.1: Example of an MC-Tree. Gaussian components of an entry may represents entities from various classes.

- *A pointer to its subnode* Sub_e
- *For each class* $l \in \mathcal{L}$ *a cluster feature* $CF_l = (n_{e,l}, LS_{e,l}, SS_{e,l})$ *representing the objects* $O(e, l)$

The algebraic measures mean and variance can be calculated out of the linear and squared sums (cf. Equations 3.1 and 3.2). Accordingly, we can calculate these values for the overall entry.

The second approach uses a technique known as variance pooling. Instead of storing the squared sums for each class, only the squared sums of all objects is stored in the entry. By this we assume for all classes the same variance within the entry, but we use less space. The linear sum, the number of objects and hence the mean are still used for each class on its own.

Definition 5.2 ***MC-Tree node entry (type B).*** *An entry* e *of type B stores the following information:*

- *A pointer to its subnode* Sub_e *(set of entries)*
- *For each class* $l \in \mathcal{L}$ *a the number* $n_{e,l} = |O(e, l)|$ *of objects and the vector* $LS_{e,l}$ *of their linear sums per dimension*
- *The squared sum* SS_e *for all objects* $O(e)$

An *inner* node of the MC-Tree is a set of the beforehand introduced entries. As in Chapter 4 a leaf node of the tree contains a set of kernels. The overall definition of the MC-Tree is:

Definition 5.3 ***MC-Tree.*** *Let $\mathcal{O}$ be a set of objects. An MC-Tree with fanout u is a tree with the following properties:*

- *each inner node contains maximally u entries (type A or B)*
- *each leaf node contains maximally u kernels*
- *the objects $O(e)$ represented by an entry e correspond to the objects of its subnode Sub_e, i.e. $O(e) = \bigcup_{e_i \in Sub_e} O(e_i)$*
- *the entries of a single node N represent disjoint object sets, i.e. $\forall e_i, e_j \in N: \ O(e_i) \cap O(e_j) = \emptyset$*
- *the root-node R represents the whole database, i.e. $\bigcup_{e_i \in R} O(e_i) = \mathcal{O}$*

Tree construction. With the definition of the tree structure we know how to represent a certain subset of objects. In the next step we must determine which objects should be grouped together to achieve a high classification accuracy based on the MC-Tree. The proposed method uses a top-down approach to divide the whole data set $\mathcal{O}$ in smaller subsets O_i. For each subset one entry e_i is constructed that represents the objects O_i. All the entries together represent an inner node of the MC-Tree. The subsets are recursively divided in smaller subsets and hence further inner nodes are constructed until the kernel level is reached.

For dividing a set of objects in reasonable subsets the EM clustering algorithm [Lau95] (cf. Chapter 2) is employed. The EM algorithm tries to represent objects with the best possible Gaussians. Because both approaches, the MC-Tree and the EM algorithm, use Gaussians to describe the data, the clustering results of EM are well suited for the index construction. Other methods like the density-based clustering DBSCAN [EKSX96] optimize different criteria to obtain the clusters. Thus, a subsequent description of these clusters by Gaussians in the MC-Tree may result in poor results. The number

of clusters used in the EM algorithm determines the fanout of the MC-Tree (explicit fanouts are provided in Section 5.2).

Using this clustering method, the MC-Tree possibly mixes several classes in one entry and can achieve compact representations of the data. However, very impure clusters with respect to their class labels could result in low classification accuracy even if the underlying objects show a similar spatial relationship. In an impure cluster we cannot discriminate between the classes and thus a precise class prediction is difficult. Hence it is preferable to find pure clusters with respect to their class labels if they also show good spatial compactness/similarity. The MC-Tree must seek a trade-off between pure entries with respect to the class labels and compact spatial clusters.

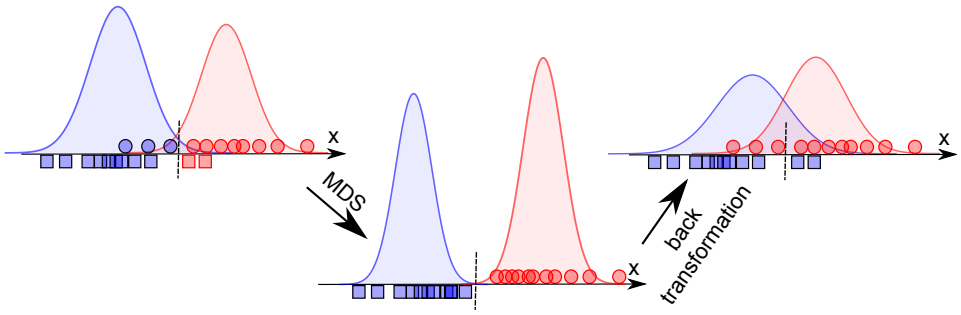
The EM algorithm does not consider the classes in the clustering process. It simply processes all objects neglecting their labels and hence pure clusters cannot be expected. To take care of this fact, objects from the same class must be considered as more similar than objects from different classes. Hence, a new distance function is used that combines the spatial similarity of the objects and their class similarity:

$$d_\delta(x,y) = \begin{cases} \delta \cdot \mathbf{d}(x-y) & \text{if } class(x) \neq class(y) \\ \mathbf{d}(x-y) & else \end{cases}$$

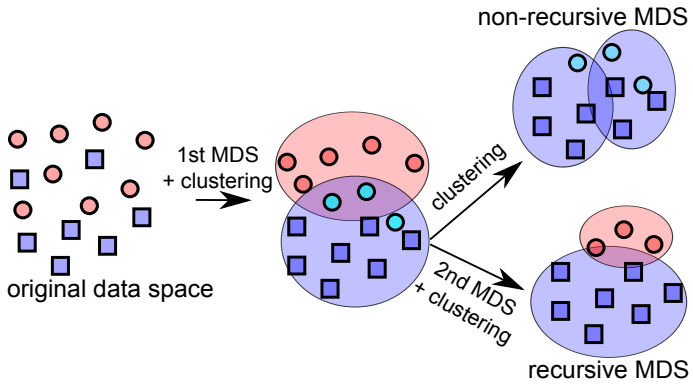
where $\mathbf{d}(x-y)$ is the Euclidean distance between the objects x and y and $\delta > 1$. However, EM cannot work on arbitrary distance functions and requires a vector space.

The proposed approach transforms the data space such that the class information is reflected by the spatial similarity and the Euclidean distance between x and y in the new space is approximately the value $d_\delta(x,y)$. As an advantage, we still have a vector space on which EM can work and we implicitly use the class information at the same time.

The proposed method makes use of multi-dimensional scaling (MDS [dL77]) to transform the data space. MDS is a non-linear transformation technique for representing or visualizing objects in any d-dimensional vector space. Often MDS is used to arrange and display objects with higher di-



(a) Transformation for better cluster detection; Gaussians in original space



(b) Recursive vs. non-recursive MDS

Figure 5.2: Multidimensional scaling (MDS) for class discrimination.

mensional features on a 2d screen. Given the original distances between the objects, MDS iteratively tries to find a mapping into the new d-dimensional space such that the distances are represented best possible. The method proposed here does not transform the objects to a lower-dimensional space but uses the same dimensionality as in the original space. It only changes the original distances to $d_\delta(x, y)$ before applying MDS. The initial coordinates which are used for the MDS algorithm are the original coordinates of the objects. By this most of the original spatial structure is preserved and is incorporated adjustments made by the MDS algorithm according to d_δ.

In Figure 5.2(a) (left) we see an example where two classes (squares/ circles) are mixed together in the 1d space. Its not possible for the EM to identify pure clusters, i.e. both clusters contain squares and circles. The clusters are marked in red and blue, respectively. After a transformation

with $\delta > 1$ the situation in the middle is obtained, for which EM yields the two marked clusters. Keep in mind that the transformation is only performed for the detection of clusters, i.e. which subsets should be grouped together. The Gaussians of the corresponding entries in the MC-Tree are still calculated in the original space. The resulting clusters/entries are presented in Figure 5.2(a) (right). Both clusters represent only objects from one class. Hence, Gaussians could overlap in the original space if by this we achieve purer clusters.

The proposed MDS technique is not only employed once at the beginning of the tree construction, but recursively for each subtree. By application of MDS only for a subset of objects a better discrimination of the classes in further steps is possible. An example is depicted in Figure 5.2(b), where it is not possible to separate all objects from class 1 to those from class 2 with the first transformation. If MDS is not performed recursively (right upper case) the clusters are still impure. However, if MDS is applied on the remaining smaller subset (right lower case), we can further discriminate the classes. With this recursive application Gaussians which represent only objects from one class are more likely to occur already in higher levels of the MC-Tree.

With the proposed method we can generate clusters and thus entries which account for the trade-off between compact spatial representations and pure cluster with respect to the class labels.

5.1.2 Classification

Given the MC-Tree we are able to perform classification of objects with unknown labels based on the mixture of densities. Two steps are distinguished in the classification process which are alternately performed. First, given a mixture model (a set of entries in the MC-Tree), we must determine the posterior probability of the contained classes w.r.t. the current object. Second, to realize the anytime property, in each step the mixture must be refined to receive a more fine-grained model. To this end an entry is replaced with the entries in its subtree. In the following the first step is discussed, the second step is detailed in Section 5.1.3.

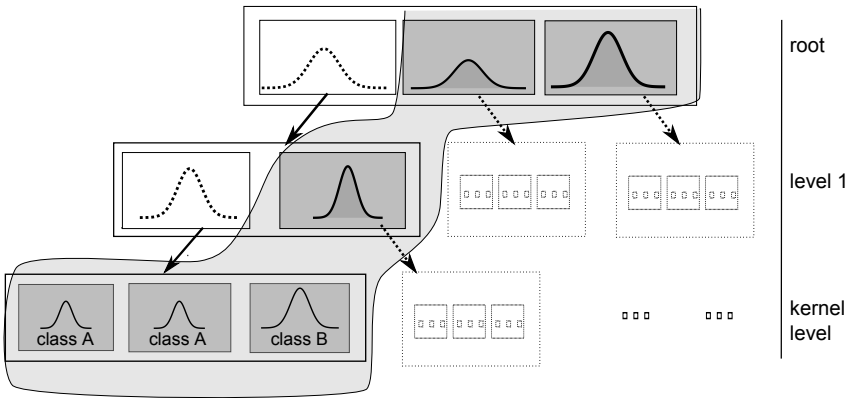


Figure 5.3: Example of a frontier. The gray entries contribute to the current mixture model for the density estimation and can be refined in further steps.

An example MC-Tree frontier is demonstrated in Figure 5.3. Given the frontier we can calculate the density of an object with respect to one class. Keep in mind that the MC-Tree can store objects from different classes in one entry. Hence, given an entry the algorithm must only use the class-specific information.

Definition 5.4 ***Probability density query (pdq) in MC-Tree.*** *Given a frontier $\mathcal{F} = \{e_1, \ldots, e_r\}$ and a class l. Let $n_{e,l}$ be the number of objects from class l in the entry e, then the pdq returns the density of an object x with respect to l and $\mathcal{F}$, i.e.*

$$pdq(x|l, \mathcal{F}) = \sum_{e \in \mathcal{F}} \frac{n_{e,l}}{|\mathcal{O}|} \cdot \boldsymbol{g}(x, \mu_{e,l}, \sigma_{e,l})$$

where $\mu_{e,l}$ and $\sigma_{e,l}$ are calculated based on the stored information within the entries (cf. Def. 5.1 or Def. 5.2).

This is a weighted mixture of densities according to distribution of number of objects from the current class. For the leaf entries a kernel estimator is used. The two different types of entries also yield two versions of the pdq.

Let n_{l_i} be the total number of objects from class l_i and let the a priori probability $P(l_i)$ be estimated from the training data as the relative frequency

of each class. Then it holds:

$$\begin{aligned} pdq(x|l_i, \mathcal{F}) &= \sum_{e \in \mathcal{F}} \frac{n_{e,l_i}}{|\mathcal{O}|} \cdot \mathbf{g}(x, \mu_{e,l_i}, \sigma_{e,l_i}) \\ &= \frac{n_{l_i}}{|\mathcal{O}|} \cdot \sum_{e \in \mathcal{F}} \frac{n_{e,l_i}}{n_{l_i}} \cdot \mathbf{g}(x, \mu_{e,l_i}, \sigma_{e,l_i}) \\ &= P(l_i) \cdot p(x|l_i) \end{aligned}$$

The term $p(x|l_i)$ is a valid probability density function because the weights associated to the Gaussians sum up to 1. With the MC-Tree the class-specific weighting $P(l_i)$ is directly integrated in the pdq. According to Bayes classification the class label assigned to an object can then be calculated as

$$\arg\max_{l_i \in \mathcal{L}} \{P(l_i) \cdot p(x|l_i)\} = \arg\max_{l_i \in \mathcal{L}} \{pdq(x|l_i, \mathcal{F})\}$$

5.1.3 Refinement

Given a frontier $\mathcal{F}$ the algorithm can guess the class of a new object. For anytime processing a chain of such frontiers must be generated, starting with the coarsest model from the root and down to the leaf nodes. In each step one entry $e \in \mathcal{F}$ is replaced with the entries in the subnode of e. This refinement of the underlying spatial data distribution corresponds to a refinement of the probability density query pdq.

Definition 5.5 ***Anytime pdq processing in MC-Tree.*** *Given a frontier $\mathcal{F}_i$. Let $e \in \mathcal{F}_i$ and $\{e_1, \ldots, e_m\}$ the entries of the subnode of e. The MC-tree algorithm refines $\mathcal{F}_i$ to $\mathcal{F}_{i+1}$ by removing e and inserting the child entries:*

$$\mathcal{F}_{i+1} = (\mathcal{F}_i \setminus \{e\}) \cup \{e_1, \ldots, e_m\}$$

The pdq of an object x with respect to a class l and the new frontier $\mathcal{F}_{i+1}$ is

calculated by:

$$pdq(x|l, \mathcal{F}_{i+1}) = pdq(x|l, \mathcal{F}_i) \quad - \quad \frac{n_{e,l}}{|\mathcal{O}|} \cdot \boldsymbol{g}(x, \mu_{e,l}, \sigma_{e,c}) \\ + \quad \sum_{i=1}^{m} \frac{n_{e_i,l}}{|\mathcal{O}|} \cdot \boldsymbol{g}(x, \mu_{e_i,l}, \sigma_{e_i,l})$$

As in the Bayes tree, from each time step i to $i+1$ the algorithm must decide which $e \in \mathcal{F}_i$ should be replaced. This decision is important for the performance in terms of anytime classification. If an entry with low information with respect to the current query is refined, the classification result may change only slightly. The time could better be spend for an improvement with more useful entries. In the following different strategies for choosing the next entry are presented.

Quality measure. During anytime processing the model is refined individually based on the current object to be classified. The classification is based on densities. A high density increases the probability of a class being selected. Hence a first approach, as presented in Chapter 4, is to refine the entry $e \in \mathcal{F}$ with the highest density. Doing this in the MC-Tree is not straightforward. Each entry subsumes several classes and densities of each class could vary. The question is, how to decide which entry is the best for refinement without favoring single classes. To make a fair selection, the density resulting from all objects is used disregarding their class labels. The next entry to refine is defined by

$$\underset{e \in \mathcal{F}}{argmax} \left\{ \frac{n_e}{|\mathcal{O}|} \cdot \mathbf{g}(x, \mu_e, \sigma_e) \right\}$$

with the mean μ_e, variance σ_e and number of objects n_e based on all objects in the entry e.

Similar to the tree construction this first approach only considers the spatial distribution of the data. During construction we considered the trade-off between pure clusters with respect to their class labels and the spatial similarity of the objects. A similar trade-off can also be defined for the refinement

method. Consider an entry e with several classes that splits up into complete pure clusters in its subnode, i.e. each sub-entry of e contains only objects from one class. Refining e yields a clear decision in the following steps and hence the accuracy could increase. This entry e should be preferred to an entry whose sub-entries are still impure. Using this intuition requires a measure that assesses the skew of the class label distribution in an entry and the possible gain if it is refined.

The MC-tree algorithm uses the well known information gain that is also used for decision trees (cf. Chapter 2). The information gain for an entry e with sub-entries $\{e_1, \ldots, e_m\}$ is defined as:

$$IG(e) = Ent(e) - \sum_{i=1}^{m} \frac{n_{e_i}}{n_e} \cdot Ent(e_i)$$

where $Ent(e)$ measures the entropy of the class label distribution in e:

$$Ent(e) = \sum_{l \in \mathcal{L}} \left(-\frac{n_{e,l}}{n_e} \cdot \log \left(\frac{n_{e,l}}{n_e} \right) \right)$$

The information gain measures the reduction of the entropy when e is refined; this corresponds to the reduction of the class label skew. This information can be calculated before the query processing, because it is independent of the actual query object x.

The higher the information gain the better is the refinement of e with respect to the class purity. The higher the beforehand defined density measure the better is the refinement of e with respect to the spatial similarity. Both measures are important for the MC-Tree; the trade-off is realized by building a linear combination of these terms for the proposed quality measure.

Definition 5.6 ***Refinement quality of an entry.*** *Given a query object x and a frontier $\mathcal{F}$, the refinement quality of an entry $e \in \mathcal{F}$ with respect to x and $\mathcal{F}$ is defined as:*

$$quality_\alpha(e, x) = \alpha \cdot \frac{IG(e)}{\log |\mathcal{L}|} + (1 - \alpha) \cdot \frac{n_e \cdot \boldsymbol{g}(x, \mu_e, \sigma_e)}{\max_{w \in \mathcal{F}} n_w \cdot \boldsymbol{g}(x, \mu_w, \sigma_w)}$$

Both measures are normalized to $[0, 1]$. The information gain is at most $\log |\mathcal{L}|$ if $\mathcal{L}$ is the set of all possible classes in the database. The user can control the influence of both measures by changing α.

Meta strategies. Formally the quality measure defines a ranking of the entries in the current frontier. To perform the refinement step the MC-tree algorithm selects the first entry out of this ranking:

$$\arg\max_{e \in \mathcal{F}} \{quality_\alpha(e, x)\}$$

Afterwards the ranking is adapted based on the newly inserted entries and the next best entry can be selected after updating the pdq. This method is referred to as *first-best*, since in each step the best entry is refined.

One possible problem of this approach is that only a small local area around the query object is refined. The algorithm can stick to one path of refined entries, while other entries with a possibly strong influence on the query are not refined. This problem is related to greedy algorithms which choose at each step the best local solution and can run into local optima not resulting in a global optimal solution. To avoid this two further meta strategies are presented for the selection of entries.

The *k-best* method is a direct extension of *first-best*. Instead of choosing the best entry this strategy marks the k best entries in the current frontier. While there are marked entries in the frontier the best out of these is selected to perform the refinement. Other non-marked entries are ignored even if they show better quality values. Only when all marked entries are processed for refinement, all entries in the frontier are considered again for marking the k currently best ones based on their quality measures. This method widens the search space because a currently second best entry is refined and can advance to a top choice for the following steps. Setting $k = 1$ yields the *first-best* method.

The *k-best* method simply chooses the k best entries based on their quality measures. The method does not consider the classes within the entries which is an important characteristic of the MC-Tree. Hence for *k-best* it is possible to select k entries all belonging to one class. This single class is fa-

vored in the classification process and all remaining classes are neglected. If the true class of the query object belongs to one of the neglected classes it is unlikely to reach a correct classification. Therefore the last proposed method tries to consider all classes equally within the refinement steps: in k successive refinement steps it favors k different classes. Similar to k-best the k best entries are marked with respect to the quality measure and the additional constraint that the strongest represented class of each marked entry is different among the k entries. This method is similar to the qbk heuristic, which yielded the best results in Chapter 4. Formally, for each class l_i that entry is selected which yields the highest quality and for which l_i is the strongest representative:

$$e^*_{l_i} = \arg\max_{e \in \mathcal{F}} \{quality_\alpha(e, x) \mid l_i = \arg\max_{l \in \mathcal{L}} \{n_{e,l}\}\}$$

Out of the set $\{e^*_{l_i}\}_{l_i \in \mathcal{L}}$ the k best entries are marked for the next refinement steps. The strongest represented class of each entry can be calculated before the actual query processing. This method, called *k-class*, accounts for different classes in the refinement and hence the classification decision is more likely to be changed to the correct class if it is underestimated so far.

5.2 Experiments

To evaluate the performance of the MC-Tree experiments are performed on different real world data sets with varying dimensionality, cardinality and number of classes (cf. Table 4.1). All experiments were run on Windows machines with 3 GHz and 2 GB RAM using Java 6.0. Mostly we will use an implementation invariant time measure (as used e.g. in [UXKL06]): in the graphs the classification accuracy is reported over the number of Gaussians that have been evaluated. When comparing against anytime nearest neighbor [UXKL06], SVM [WF05] and C4.5 [WF05] the actual time is provided on the x-axis. The goal in this chapter is to improve the accuracy of anytime Bayesian classification rather than showing statistical evidence for the superiority of one or the other classifier in different domains. 4-fold cross

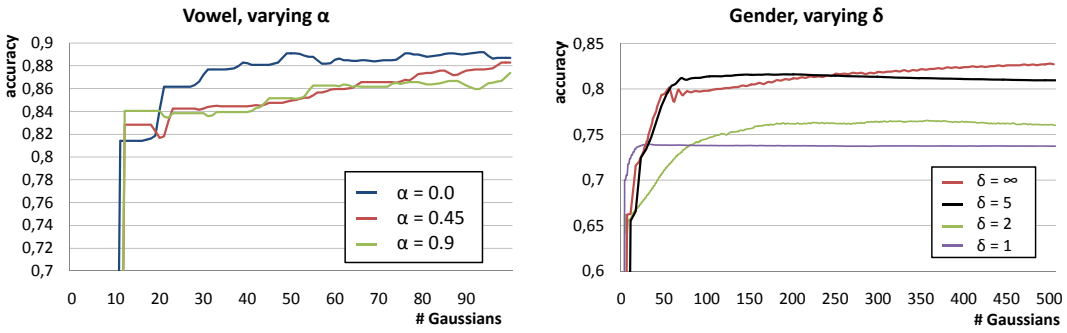


Figure 5.4: Left: Varying the influence of entropy during descent via the parameter α. Right: Varying the penalty for MDS construction via the parameter δ.

validation was performed on the data sets and the accuracy values are the corresponding averages. The tree structures are constructed once using all training data of the current fold. For updates of the trees using additional new training data one can employ the incremental insertion proposed in Chapter 4 or adapt other split strategies. Since the focus is on the anytime classification performance, these aspects are not studied here. In the next section first the influence of the parameters on the anytime classification accuracy of the MC-Tree are evaluated and in Section 5.2.2 the improvement of anytime Bayesian classification over previous results from Chapter 4 is shown.

5.2.1 MC-Tree parameter evaluation

To find a good parameter setting for the MC-Tree we start by evaluating the influence of the entropy level during descent by varying the parameter α in Figure 5.4 (left). The *k-class* strategy is used here, since it performed best among all meta strategies in primary experiments. Also, zero penalty was used for the MDS ($\delta = 1$) and the classification decision is based on $\mu_{e,c}$ and $\sigma_{e,c}$. The maximal number of entries in a node and hence the number of clusters for the EM algorithm is set to the number of classes in the respective data set.

We find a clear winner in the results indicating that $\alpha = 0$ yields the best

results. This means that the local density of the individual Gaussian components in the frontier is sufficient to best determine the next entry for a refinement. Discounting the importance of entries that yield a high probability density with respect to the query object by promoting other entries due to their higher entropy obviously delays beneficial model refinements and reduces the anytime accuracy performance. Hence, in the following only the entry's probability density is used and α is set to zero.

In Figure 5.4 (right) the influence of the penalty for the construction using MDS is shown on the Gender data set. The purple line corresponding to $\delta = 1$ (zero penalty) rises quickly to an accuracy of around 74%, but does not improve it afterwards. Similar results were found for most data sets when using the same parameter setting: steep increase early on and little improvement in later stages.

Increasing the penalty through a higher δ yields continuous improvement in terms of anytime accuracy. While the curve for $\delta = 1$ shows better performance for the first Gaussians that are read, the other settings can soon improve the accuracy significantly. The tree build with $\delta = \infty$ penalty shows an accuracy that is up to 9% higher for this data set. The superior performance for this setting was confirmed on the other data sets. Moreover, the stagnating behaviour disappeared throughout as will be shown in the next section.

$\delta = \infty$ constitutes a special case, since this transformation can be considered as dividing the database into subsets, such that all objects with the same class label are in the same subset. Afterwards the clustering is performed only on the separated classes. Hence we do not need the actual transformation but can directly cluster in the original space on these subsets. This special case is more similar to the Bayes tree, but the resulting hierarchy is still very different due to the top down clustering instead of the balanced R-tree split as in Chapter 4. We will see in the next section that the MC-tree shows significantly better anytime accuracy.

As a consequence of the high penalty and the resulting tree structure, all classification options provided by the different node types (cf. Def. 5.1 and 5.2) yield the same results and are therefore not displayed.

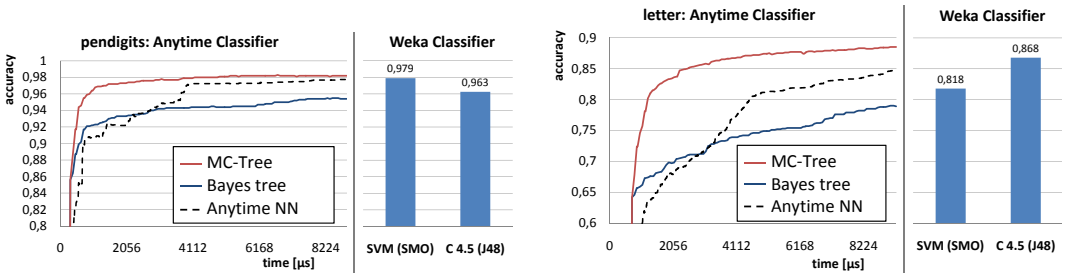


Figure 5.5: Classification accuracy on pendigits (left) and letter (right) for anytime classifiers and static classifier.

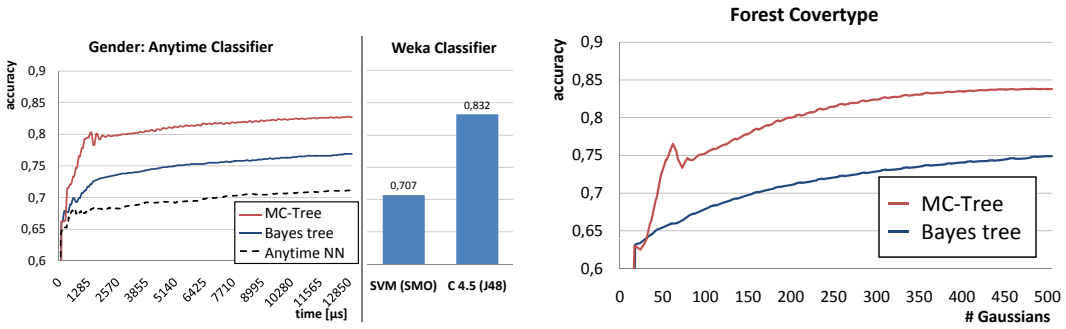


Figure 5.6: Classification accuracy on gender (left) and Forest Covertype (right).

5.2.2 Comparison: Improvement of anytime Bayesian classification

Next the MC-Tree is compared to the Bayes tree. For the Bayes tree the global best descent strategy and the qbk refinement strategy are used as they yielded the best results (cf. Chapter 4). For the MC-Tree the fanout (the maximal number of entries per node) is set to the same amount as in the Bayes tree where it is dictated through the page size. Eventually the results of both approaches will be equal, i.e. if the entire leaf level has been read the decision of both classifiers is based on the same model. However, the graphs focus on the interesting part: the anytime performance in the beginning of the classification process. On the right hand side of the Figures the corresponding accuracy reached by the SVM and C4.5 implementation from Weka [WF05] is reported. These methods do not constitute an anytime approach.

Figure 5.5 (left) shows the results for the Pendigits data set. The Bayes tree performs only slightly better than the anytime nearest neighbor in the beginning before it falls behind. The MC-tree shows a steep increase in the very beginning and outperforms the other two anytime classifiers throughout. While the accuracy of the MC-tree is similar to the nearest neighbor in later stages, it quickly shows a performance gain over the Bayes tree of three to four percent accuracy. Support vector machine (SVM) and decision tree (C4.5), which are considered very fast classifiers, perform well with 97.9% and 96.3% accuracy respectively. These approaches, however, cannot improve their accuracy once they computed their result while the anytime classifiers will use additional time for further computations.

Similar results can be seen for the letter data set in Figure 5.5 (right). For this domain C4.5 (86.8%) performs better than the SVM (81.8%). The results of the anytime NN and the Bayes tree are again similar, after a short time the nearest neighbor bypasses the accuracy of the Bayes tree. The MC-tree outperforms both approaches and reaches accuracy values which are up to 13% higher as compared to the Bayes tree constituting a major performance gain. Moreover, it soon reaches higher accuracy than both SVM and C4.5.

On the gender data set (cf. Figure 5.6 (left)) the Bayes tree shows constantly better performance than anytime NN. Once more, the MC-tree constantly outperforms both approaches by roughly 6% compared to the Bayes tree and 10% compared to anytime NN. As was the case for the letter data set, the decision tree (83.2%) reaches higher accuracy than the support vector machine (70.7%). This performance is once again met by the MC-tree after short time and additional time can be used for further improvement. More importantly, as was the major goal, the new concepts of the MC-tree prove to be effective through constantly better performance in comparison with the Bayes tree.

In Figure 5.6 (right) the results for the MC-tree and Bayes tree on the Forest Covertype data set are shown. The results for the anytime nearest neighbor, SVM and C4.5 could not be computed due to memory issues. In this Figure the number of Gaussians are reported that have been evaluated until

classification. As stated above, the goal in this chapter was to improve the performance of Bayesian anytime classifiers, which is again clearly reached in this experiment.

The accuracy curve of an anytime approach is desired to be a non-decreasing function. In Figure 5.6 we observe a slight up and down in the anytime curves for the MC-tree. Since in both cases $k = 2$ holds, the two most probable classes are refined in turns. Obviously the classification decision changes for some queries after each node that is evaluated. The amplitude of the oscillation indicates the proportion of queries behaving as just described. Those query points fall into a region of the data space where two classes overlap and where a correct decision is difficult to find. Refining the one class' model by reading an additional node increases its probability in that step, while in the next refinement the other class' model is refined (due to $k = 2$) and its corresponding probability prevails. However, the slight oscillation of the anytime curves does not diminish the dominance of the MC-Tree in terms of anytime accuracy.

5.3 Conclusion

In this chapter the MC-Tree was proposed which significantly improves the anytime classification performance over previous results from Chapter 4. Various strategies for tree construction, descent and classification were investigated. In experimental evaluation on real world data sets the MC-Tree outperformed previous Bayesian anytime classifiers and improved the accuracy constantly by up to 13%.

Starting from the initial idea of combining several classes in a single tree a novel way of constructing the tree was investigated through top-down clustering using the EM algorithm. While it turned out that separating the classes remains advisable, the EM construction showed very good results. Therefore, further alternative construction methods are investigated in the following chapter. Since we do not combine several classes in a single tree in the remainder of the thesis, we will not use the term MC-Tree in the following and only refer to the different construction as EMTopDown bulk loading.

Chapter 6

Bulk Loading the Bayes Tree

* In this chapter several methods are investigated for bulk loading mixture densities in the Bayes tree and the resulting performance is analyzed in experimental evaluation.

6.1 Bulk loading mixture densities

The goal in this chapter is to improve the performance of the Bayes tree. The accuracy of the Bayes tree results is based on the quality of the mixture densities stored in its entries. The iterative insertion performed in Chapter 4 does not consider the quality of the resulting Gaussian components. Several bulk loading approaches are developed and evaluated in this chapter that try to overcome this shortcoming and improve the quality of the mixture densities.

6.1.1 Machine learning and statistical approaches

Goldberger. Since the Bayes tree is a statistical approach to classification, statistical methods are investigated to create a smaller mixture model from a given mixture model. Starting bottom up with a mixture model that contains

*This chapter has been published in the Proceedings of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2010) [KKDS10].

a kernel estimator for each training set item successively coarser models are created that represent good approximations.

The first statistical approach is based on [GR04] and is called Goldberger in the following. The Goldberger approach assumes two initial mixture models f and g to be given, where f is the finer model with r components and g an approximation with s components, hence $r > s$. Each component is assigned a weight and is specified by its mean and covariance matrix. To measure the quality of the approximation [GR04] defines the distance between two mixture densities as follows.

Definition 6.1 *Let $f = \sum_{i=1}^{r} \alpha_i f_i$ and $g = \sum_{j=1}^{s} \beta_j g_j$ be two mixture densities containing r and s Gaussian components f_i and g_j with their respective weights α_i and β_j. The distance between f and g is then defined using the Kullback-Leibler divergence KL [CHOY08] as*

$$d(f,g) = \sum_{i=1}^{r} \alpha_i \cdot \min_{j=1}^{s} \{KL(f_i, g_j)\}$$

The optimal model $\hat{g}$ reducing f to s components is $\hat{g} = \arg\min_g(d(f,g))$. Since there is no closed form to compute $\hat{g}$, a local optimum is computed iterating the following two steps until the distance $d(f,g)$ no longer decreases. In that mapping $\pi(i) : \{1 \dots r\} \to \{1 \dots s\}$ is a mapping function that assigns each component in f to a component in g.

- Regroup: update π: $\pi(i) = \arg\min_{j=1}^{s} \{KL(f_i, g_j)\}$
- Refit: for each component g_j recompute weight β_j, mean μ_j and covariance matrix Σ_j as follows
  - $\beta_j \leftarrow \sum_{i,\pi(i)=j} \alpha_i$
  - $\mu_j \leftarrow \frac{1}{\beta_j} \sum_{i,\pi(i)=j} \alpha_i \mu_i$
  - $\Sigma_j \leftarrow \frac{1}{\beta_j} \sum_{i,\pi(i)=j} \alpha_i \left(\Sigma_j + (\mu_i - \mu_j)^2\right)$

A bulk loading technique is devised based on [GR04] as follows. To initialize the mixture g a first mapping π_0 is computed by assigning $0.75 \cdot M$

components from f to one component in g according to the z-curve [Sag94] order of their mean values. M is the Bayes tree fanout, which in turn is dictated by the page size. When no more changes occur in step 2, the resulting components g_j are converted to Bayes tree nodes containing the entries f_i with $\pi(i) = j$. Since the final π may map more than M components from f to a single component in g, strategies must be found to restrict the fanout to the given boundaries. Reformulating the regroup step into an integer linear program with constraints regarding the resulting fanout is a first option. However, primary experiments showed that for realistic problem sizes this approach takes way too long to compute a complete bulk loading. Hence, a post processing is performed after the mapping π is computed, which splits the nodes that contain too many entries. Therefore two representatives are computed by moving the mean along the dimension a with the highest variance σ_a by an $\epsilon = \sigma_a/2$ in both directions. A Gaussian is placed over the two representatives and the mapping of the entries to the representatives is computed as in the regroup step. If a node contains too few entries it is merged with the node closest to it in terms of the Kullback-Leibler divergence.

Virtual sampling. The second approach, called virtual sampling, uses the work presented in [VL98] and does not rely on the KL divergence. The virtual sampling approach assumes a given mixture model $f = \sum_{i=1..r} \alpha_i \cdot f_i$ containing r components and computes a coarser mixture model $g = \sum_{j=1..s} \alpha_j \cdot g_j$ with $s < r$ components. The components $f_i = \mathbf{g}(x, \mu_i, \sigma_i)$ (and g_j for g) constitute multivariate Gaussian normal distributions with their respective weight α_i. To derive an algorithm the following model is utilized: the mixture g can be computed using samples $R_1 \ldots R_r$ from each component in f with $R = \cup_{i=1..r} R_i$ and $|R_i| = \alpha_i \cdot |R|$. Assuming independence of the sample points from different components in f yields the assumption that they can be assigned to different components in g while samples from the same f_i are likely to be assigned to the same g_j. Based on this assumption hidden variables z_{ij} are introduced that indicate for each component f_i its assignment to the corresponding g_j. While the z_{ij} are binary during initialization, they can take values between 0 and 1 during the iterations. The hidden variables are used in a modified Expectation Maximization algorithm to compute the

coarser mixture g as follows (superscripts ^{f} and ^{g} are added for readability to indicate the origin of the components):

- Expectation: $z_{ij} \leftarrow \frac{\left[\mathbf{g}(\mu_i^f, \mu_j^g, \Sigma_j^g) e^{-\frac{1}{2} trace\{(\Sigma_j^g)^{-1}\Sigma_i^f\}}\right]^{|R_i|} \cdot \alpha_j^g}{\sum_{k=1}^{s}\left[\mathbf{g}(\mu_i^f, \mu_k^g, \Sigma_k^g) e^{-\frac{1}{2} trace\{(\Sigma_k^g)^{-1}\Sigma_i^f\}}\right]^{|R_i|} \cdot \alpha_k^g}$
- Maximization:
  - $\alpha_j^g \leftarrow \frac{1}{r}\sum_{i=1}^{r} z_{ij} \qquad \mu_j^g \leftarrow \frac{\sum_{i=1}^{r} z_{ij}|R_i|\mu_i^f}{\sum_{i=1}^{r} z_{ij}|R_i|}$
  - $\Sigma_j^g \leftarrow \frac{1}{\sum_{i=1}^{r} z_{ij}|R_i|}\left[\sum_{i=1}^{r} z_{ij}|R_i|\Sigma_i^f + \sum_{i=1}^{r} z_{ij}|R_i|\left(\mu_i^f - \mu_j^g\right)^2\right]$

The above equations are independent of the actual samples R_i and can be computed directly from the mixture components in f, hence *virtual sampling*. To use the described bottom up method for bulk loading an initialization is required for the hidden variables z_{ij}. The initialization of the mixture g is done as in the Goldberger approach described above. After getting the final values for z_{ij} from the virtual sampling algorithm, each f_i is assigned to that g_j with the maximum z_{ij} for all j. Moreover, the result must comply with the fanout parameters m and M of the Bayes tree. This is achieved through merging and splitting of the resulting components g_j. If a component g_j is assigned fewer than m components f_i, these components from f are assigned to the $g_{j'}$ with the second highest $z_{ij'}$. If more than M components f_i are assigned to one g_j, g_j is duplicated while moving the resulting two means in opposite direction along the dimension with the highest variance. The respective f_i are reassigned to the more probable candidate according to the density of their mean μ_i. After merging and splitting the corresponding mixture parameters are adapted following the above equations.

EMTopDown. Besides the above mentioned bottom up approaches the top down approach from Chapter 5 is evaluated that recursively splits the training set into several clusters. In contrast to the previous approach, where Gaussian components were merged and mapped, this approach operates solely on the data objects. More precisely, it starts with applying the EM [DLR77] algorithm to the complete training set. The desired number M of resulting clusters is always set to the fanout which is again given through

the page size. If the EM returns fewer than m clusters, the biggest resulting cluster is split again such that the total number of resulting clusters is at most M. In the rare case that the EM returns a single cluster, this cluster is split by picking the two farthest elements and assigning the remaining elements to the closest of the two. Finally, if a resulting cluster contains more than L objects (the capacity of a leaf node), the cluster is recursively split using the procedure described above. Otherwise the items contained in that cluster are stored in a leaf node, its corresponding entry is calculated and returned to build the Bayes tree. The EM approach may result in an unbalanced tree, which differs from the primary Bayes tree idea. However, as we will see in the experimental section, the results show that this is not a drawback but even leads to better anytime classification performance.

6.1.2 Data base driven approaches

Since the Bayes tree extends the R-tree, traditional R-tree bulk loading algorithms are employed for comparison. Two types of space filling curves are used in the experiments, namely Hilbert curve [Sag94] and z-curve. The following briefly describes the Hilbert curve approach, the z-curve bulk loading works analogously. The bulk loading according to the Hilbert curve is a bottom up approach where in the first step the Hilbert value for each training set item is calculated. Next the items are ordered according to their Hilbert value and put into leaf nodes w.r.t. the page size. After that the corresponding entry for each resulting node is created, i.e. MBR, cluster features (CF) and the pointer. These steps are repeated using the mean vectors as representatives until all entries fit into one node, the root node. Theory on creating multidimensional Hilbert curves can be found in [AN98], for implementation guide lines see [Law00].

Finally, the partitioning approach presented in [LEL97] is tested, which is called sort-tile-recursive. The basic idea is to build a hierarchy of rectangles which share per dimension approximately the same expansion in each dimension at the same level of the hierarchy. For details please refer to [LEL97].

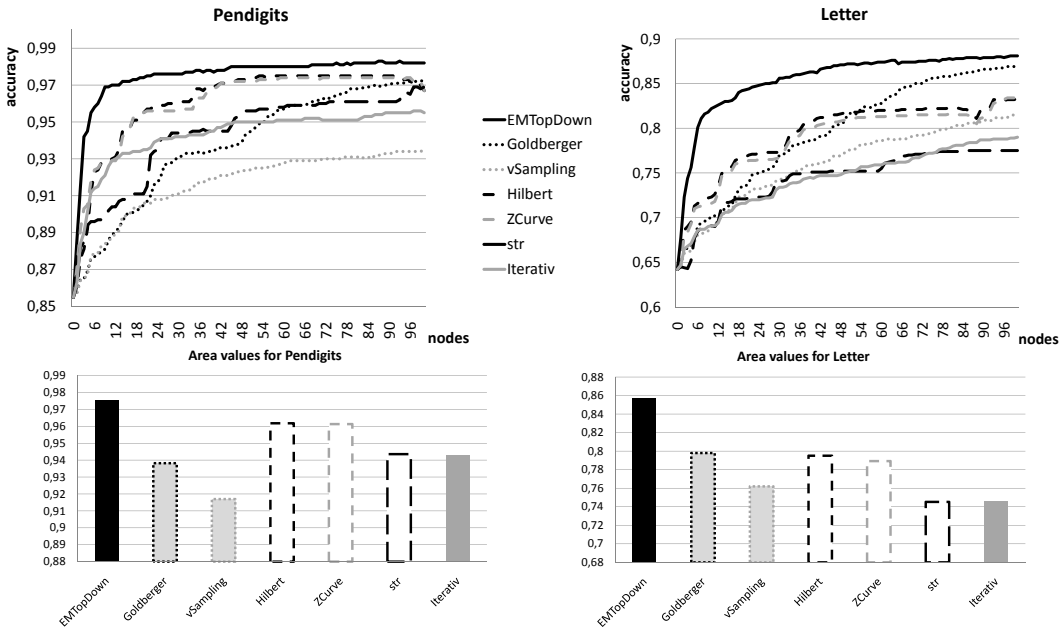


Figure 6.1: Anytime classification accuracy and a ranking of all approaches according to the area values for Pendigits (left) and Letter (right).

6.2 Experiments

The three proposed bulk loading techniques *Goldberger*, *virtual sampling* and *EMTopDown* are compared to the existing R-tree bulk loading approaches *Hilbert*, *z-curve* and *STR* and the previous results from Chapter 4 (called *Iterative* in the graphs since it performs iterative insertion of objects). The same settings as in Chapter 4 were used: 4-fold cross validation is performed on the same data sets and the classification accuracy is shown after each node averaged over the four folds. The employed strategies are the global best descent and the qbk improvement strategy as they showed the best results in Chapter 4. The bulk loading is done per fold once and offline and the resulting classifier is then used on the data stream. Since the focus is on anytime classification, the time performance of the bulk loading algorithm is not evaluated, but the performance of the resulting classifier in terms of its anytime classification accuracy.

The top left part of figure 6.1 shows the results for the pendigits data set. The Goldberger approach fails to improve the accuracy over the iterative in-

sertion for the first 50 nodes. After that it performs slightly better, but cannot increase the accuracy by more than 1%. Virtual sampling performs worst on this data set. The Hilbert and z-curve bulkload yield comparable results, their corresponding curves show a steep increase similar to the iterative insertion and show better performance in most cases. After falling behind during the first nodes, STR performs equaly well compared to Iterative. The EMTop-Down bulkload outperforms all other approaches and improves the accuracy over the iterative insertion constantly by 3% or more on this data set.

The performance of the Goldberger bulkload stayed below the iterative insertion in the majority of the experiments. Just on the Letter data set it improved the accuracy for larger time allowances (cf. Figure 6.1, right). For the first 40 nodes Goldberger and Iterative perform equally well, after that the accuracy of Iterative stays behind that of Goldberger. While the virtual sampling and STR bulkload shows similar performance to Iterative, Hilbert and z-curve (which are again in close proximity to each other) show constantly better accuracy than Iterative. The EMTopDown again constantly yields the best accuracy up to 13% better than the iterative insertion.

To facilitate an easier comparison between the different approaches Figures 6.1 (bottom) and 6.2 show the normalized area under the anytime curves for Pendigits, Letter, Vowel, USPS and Verbmobil. Throughout the data sets Hilbert and z-curve show nearly the same performance, while z-curve is usually slightly behind Hilbert except for the USPS data set. STR ranges between these two and the iterative insertion; it is never better than the former and never beaten by the latter. Surprisingly both statistical approaches exhibit the same weakness as STR, i.e. they never outperform the z-curve bulk load (except Goldberger on Letter) and several times show even worse performance than iterative insertion. This is especially interesting since both approaches are initialized using the z-curve, however, only with $0.75 \cdot M$ entries per node (cf. Section 6.1.1). We discuss the reasons for this shortcoming of the statistical approaches at the end of the section where we analyze the structure of the resulting trees. The EMTopDown bulk load shows constantly the best performance on all data sets despite the unbalanced resulting trees. Again we defer the analysis to the end of the section.

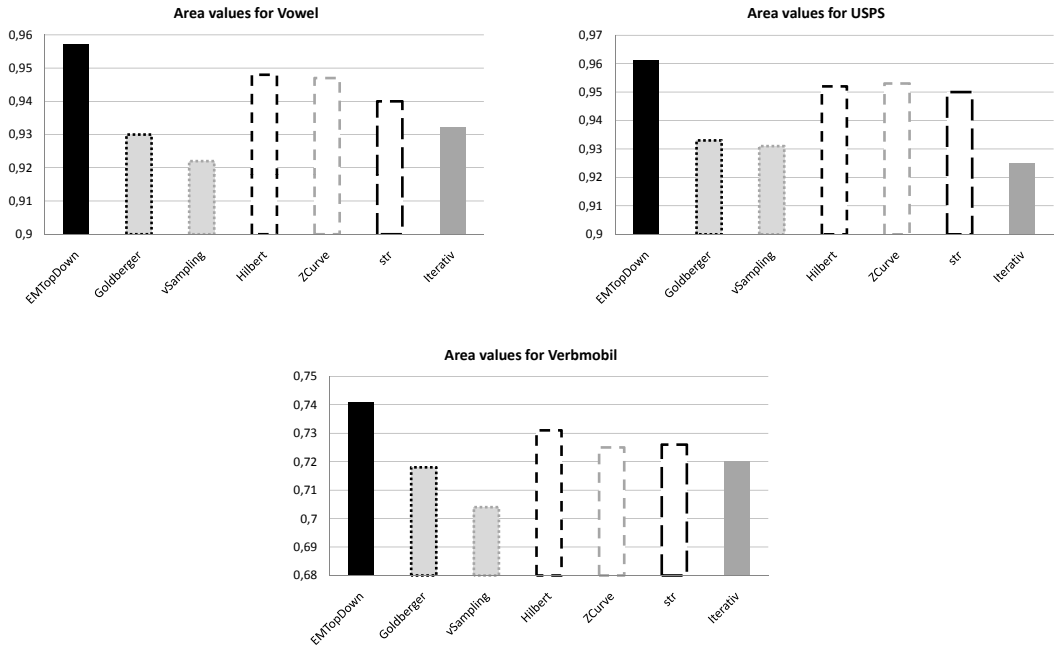


Figure 6.2: Comparison of all approaches on Vowel, USPS and Verbmobil.

Figure 6.3 shows the results for the gender and covertype data sets. For readability only the results for Hilbert and EMTopDown are shown. For both data sets $k = 2$ for the qbk improvement strategy. The graphs for EMTopDown and Hilbert using the global best descent (*glo*) show an oscillating behavior on both data sets. For comparison the breadth first traversal (*bft*) is recapitulated, the results are plotted as well. As was found in Chapter 4, the global best descent performs better than breadth first traversal. However, the graphs for bft do not show the oscillating behavior mentioned above. Since $k = 2$, there is obviously a certain proportion of object whose class decision changes in favor of (or against) the tree which is currently refined. More precisely, these objects are likely positioned on the decision boundary between the two most probable classes. In global best descend refining mixture components close to the objects, and hence close to the decision boundary, affects the corresponding posterior probabilities more heavily than refinement of a farther component as in breadth first traversal. If we assume the oscillation to be a higher frequency added to a smooth underlying anytime curve, the proportion of these borderline objects corresponds to the amplitude. How-

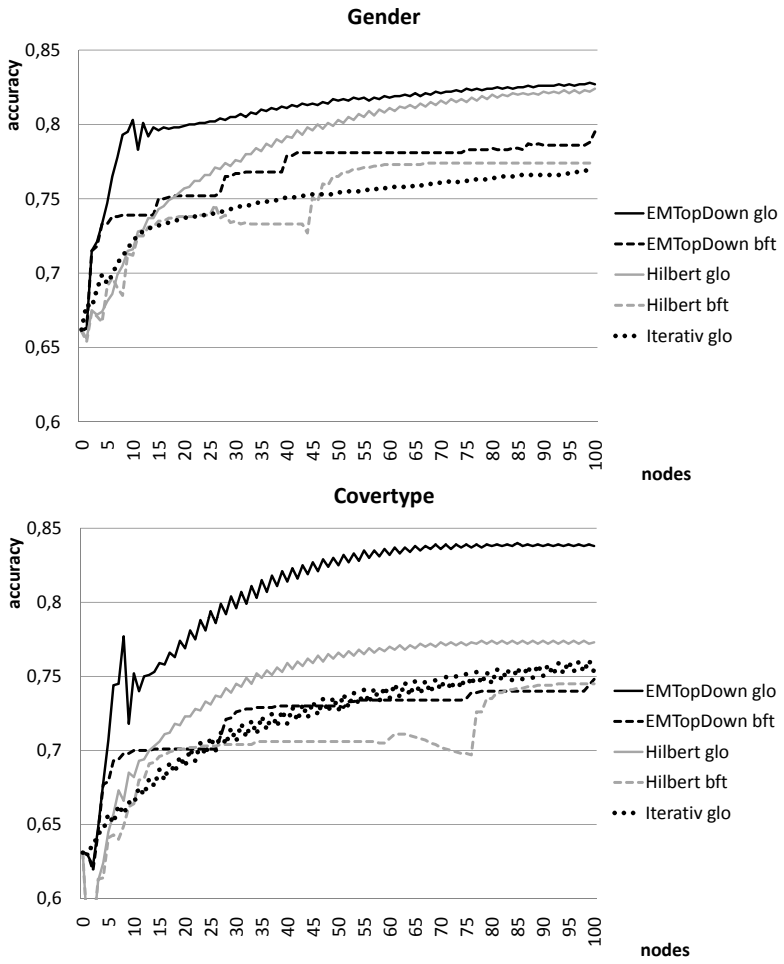


Figure 6.3: Anytime classification accuracy on Gender and Covertype.

ever, on balance, the oscillating behavior does not affect the superiority of the bulk loading over the iterative insertion.

To find reasons for the surprising ranking of the individual algorithms, the structure of the resulting trees is analyzed. Since more entries in a node correspond to more detailed information compared to fewer entries, the analysis focuses on the the degree to which the nodes were filled in the different approaches. To this end the average fanout was computed per level of the trees from root to leaf. However, the resulting figures did not reveal any correlation to the found ranking of the approaches. For example, both space filling curve approaches always fill the nodes to nearly 100% (except for the last

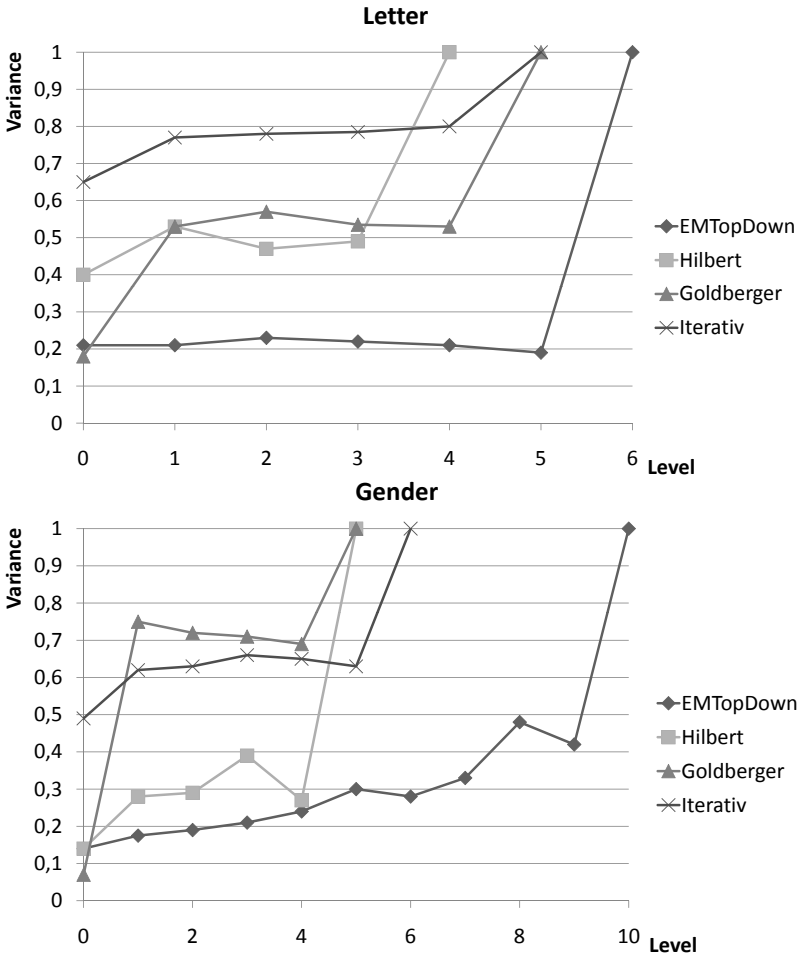


Figure 6.4: Variances per level for Letter and Gender.

one per level and the root), but the EMTopDown sometimes produces fewer than M entries for a given node.

A correlation can be found between the performance of the algorithm and the variance of the mixture components in the resulting trees. More precisely, the average variance of all entries is calculated per level, Figure 6.4 shows the resulting numbers for Hilbert, Goldberger, EMTopDown and Iterative on the Letter and Gender data sets. The variances are normalized by the variance of the entire data set per class. Level 0 corresponds to the leaves, Level 1 is above the leaves, etc. The trees resulting from different approaches can have different heights as can be seen in the graphs. Hilbert

bulk load fills each node to 100% if possible and consequently yields the smallest trees, while the unbalanced trees resulting from EMTopDown are up to twice as high on the Gender data set.

EMTopDown and Hilbert show significantly smaller average variances compared to the iterative insertion. While the corresponding variances for Goldberger are smaller than those of Iterative for the Letter data set they are larger on the Gender data set. This is in line with the observed anytime classification performances. Comparable similarities were found between the two measures on the other data sets. The average variances per level achieved by the EMTopDown bulk load were constantly amongst the lowest compared to all other approaches. This explains and underlines the superior performance of EMTopDown. The fact that the EMTopDown approach may yield unbalanced trees can lead to the assumption that this supports low variances. Specialized approaches, incremental or bulk loading, that strive to minimize the variance can be developed and tested to investigate this relation.

In general the EMTopDown shows the best results in terms of anytime classification accuracy on all tested data sets and continuously improves the accuracy over that of the previous results in Chapter 4 by up to 13%. This proves the effectiveness of the proposed bulk loading approach for hierarchical anytime classifiers.

6.3 Conclusion

In this chapter three bulk loading approaches were proposed for hierarchical mixture models to improve Bayesian classification on data streams using the Bayes tree. The approaches were compared to the previously proposed iterative insertion and three known R-tree bulk loading algorithms on a range of real world data sets. Experimental results showed that the EMTopDown bulk load constantly outperformed all other approaches and improved the accuracy by up to 13%. Surprisingly the two statistical approaches were outperformed by existing R-tree bulk loadings based on space filling curves. Further analysis attributed this shortcoming to a structural property of the resulting

Bayes trees. The results of the analysis were in line with the classification results found in the experiments confirming the superior performance of the new EMTopDown bulk loading in terms of anytime classification accuracy.

In the next chapter different popular classifiers are analyzed to identify ways to improve the Bayes tree by learning from related approaches. Transferring related concepts to the Bayes tree yields near perfect results on all data sets tested. Moreover, a remedy is found for the oscillating behavior seen in the results of Section 6.2.

Chapter 7

The Classifier Family: Learn from your Relatives

If two classification approaches are considered one can often find certain similarities, for example in the way that they build their model or how their parameters are optimized. Sometimes it is even possible to parameterize the individual approaches such that they yield the exact same decision boundaries. An example was shown for support vector machines and Bayesian classification using Gaussian mixtures in Chapter 2, other examples include the equality between nearest neighbor classification and Bayesian classification using kernel density estimators. This chapter investigates what the Bayes tree can learn from related approaches, for example other classification paradigms, to improve its anytime classification performance. For all processes of the Bayes tree, from construction to decision making, matching concepts are identified and transferred to alter the corresponding process. Extensive experimental results show the benefits of the concept transfer approach and yield outstanding results on all data sets tested.

7.1 Introduction

Combining classifiers to improve their performance has been frequently researched in the literature. The goal of the proposed methods is to build a

classifier that has a better performance than the given baseline approaches. While some methods focus on a specific application or the efficiency of classifiers [TLB04, AKKS99], most methods seek to build a general classifier that yields a high accuracy in various domains. One way to categorize the abundance of related work is to distinguish methods that use an ensemble of classifiers without modifying the single approaches from those methods that modify a classifier by transferring concepts from other approaches.

Ensemble methods train a set of classifiers and combine their results e.g. by majority voting or using individual weights for the single classifiers. Many approaches have been proposed including boosting [FS96, KSK02, ZWLL09], bagging [Bre96, LK05, LP03] and Random Forests [Bre01]. While some ensemble methods use only classifiers from the same paradigm, others combine the outputs of classifiers from different paradigms and achieve often higher robustness and better accuracy [KHDM98]. In [WKB97] four paradigms were combined (decision trees, neural networks, $k$nn and Bayes). In [LBB02] a genetic algorithm was used to find an appropriate rule for combining decision trees with neural networks. Similar research ideas tried to fuse discriminative with generative approaches like [DHN10], where the advantages of support vector machines and Gaussian mixture models were combined to create an image classification technique that benefits from both approaches. In [VS09] it was shown that using simple majority voting could negatively affect the overall performance of the ensemble of different paradigms and the idea of using dynamic weighting in such ensembles was introduced.

In contrast to the above mentioned ensemble methods, the approach that is followed in this chapter does only use the concepts from other approaches for improving the Bayes tree, i.e. it does not result in an ensemble of different classifiers. The output is an improved version of the original classifier.

The idea of modifying a given classifier by transferring concepts from other approaches was e.g. used in [PW10], where a method was suggested to optimize the performance of Gaussian mixtures-based Bayesian classifiers using the concept of margin maximization. In [TLB04] the variable selection for SVMs was done using Random Forests and vice versa. The issue of discriminative vs. generative concepts for learning Bayesian network classifiers

was discussed in [RH97]. In [GD04] a method for learning Bayesian network classifiers through a two-level optimization using the concept of maximizing the conditional likelihood was proposed.

In this chapter different options for combining the Bayes tree with related classifiers are identified and evaluated. Details of the individual concept transfers are provided in Section 7.2, the results are discussed in Section 7.3.

7.2 Learning from Relatives

For the concept transfer the three processes of building a classifier are considered. The employed notation was introduced in Section 2.2:

$M = \eta_{\mathcal{C}}(\Xi, \mathcal{T})$	Model selection (Eq. 2.1)
$\Theta = \pi_{\mathcal{C}}(M, \mathcal{T})$	Parameter optimization (Eq. 2.2)
$l = \mathbf{f}_{\mathcal{C}}(\Theta, q)$	Decision (Eq. 2.3)

The general idea of optimizing a classifier $\mathcal{C}$ by learning from relatives is to identify promising concepts of related approaches that can be harnessed to improve $\mathcal{C}$. More precisely, given a set of approaches $\mathcal{A} = \{A_1, \ldots, A_{|\mathcal{A}|}\}$ the optimization process combines the concepts of an approach $A \in \mathcal{A}$ with those of $\mathcal{C}$, denoted as $\mathcal{C} \diamond A$, and yields new functions $\eta_{\mathcal{C}\diamond A}$, $\pi_{\mathcal{C}\diamond A}$ and $f_{\mathcal{C}\diamond A}$. This is different from using A and $\mathcal{C}$ in an ensemble, since there only the outputs of $f_{\mathcal{C}}$ and f_A are combined, while the single concepts are left unchanged. The following chapters describe the investigated concepts for the three processes of model selection, parameter optimization, and decision.

The Bayes tree is referred to as BT in the remainder of this chapter and the following notation is used throughout.

Definition 7.1 *The model $M = \{\mathcal{N}_1, \ldots, \mathcal{N}_r\}$ of a Bayes tree is a set of connected nodes. Each node $\mathcal{N} = \{e_1, \ldots, e_s\}$ stores a set of entries with $2 \leq s \leq maxFanout$. An entry $e = \{p_e, n_e, \vec{LS}_e, \vec{SS}_e\}$ stores a pointer p_e to the first node $\mathcal{N}_e$ of its subtree, the number n_e of objects in the subtree and their linear and quadratic sums per dimension. The root node $\mathcal{N}_{root}$ stores exactly one entry e_o^l for each label $l \in \mathcal{L}$ that summarizes all objects with label l.*

$\mathcal{T}_{|e}$ refers to the set of objects stored in the subtree corresponding to entry e. In the Bayes tree, η_{BT} creates M from $\mathcal{T}$ using the iterative insertion process from R-trees (cf. Chapter 4) or the EMTopDown bulk load (cf. Chapters 5 and 6). Both approaches are used as baselines; they are referred to as R and EM, respectively.

7.2.1 Tree construction

The first process for building a classifier $\mathcal{C}$ is the model selection $\eta_{\mathcal{C}}$. The general approach followed in this chapter is to analyze the actions that are taken by the classifier and describe them by characteristic terms. For the Bayes tree, η_{BT} refers to determining the hierarchy of mixtures, i.e. deciding how to create the tree structure. Hence, any approach A that partitions $\mathcal{T}$ can be employed in this stage. Moreover, the Bayes tree always processes the hierarchy from the root, i.e. placing more valuable or important information in the upper levels may improve the performance of the approach. To this end, any approach A that ranks the objects in $\mathcal{T}$ with respect to their usefulness can be profitable. Therefore, approaches are searched that match the terms *partitioning* or *ranking*.

DT. Although decision trees do not maintain one hierarchy per class, the training of decision trees involves a partitioning of the data space based on the training data. Hence, the approach fits the term *partitioning* and is a candidate for transferring concepts to the Bayes tree. There are numerous ways to build decision trees; here the partitioning method proposed by Fayyad and Irani in [FI93] is adopted. Therein the information gain is used as the splitting criterion and the termination criterion is based on the minimal description length. The following node structure for a binary decision tree is used:

Definition 7.2 *A decision tree node $\nu = (A_i, a_{ij}, \delta, p_1, p_2)$ stores a splitting attribute A_i, a split value a_{ij}, a corresponding value for the information gain δ and two pointers p_1 and p_2 to the left and right subtree. A Leaf node ν is associated with a set of training objects $\mathcal{T}_{|\nu}$ and an inner node with the union of its subtree data sets $\mathcal{T}_{|\nu} = \mathcal{T}_{|\nu_1} \cup \mathcal{T}_{|\nu_2}$.*

Let $\mathcal{T}_l \subset \mathcal{T}$ be the set of all training objects with label l. Creating the hierarchical mixture models for a class $\hat{l}$ starts with all objects $\mathcal{T}_{\hat{l}}$ and the single partition induced by the root node ν_{root} of the decision tree: $part_0 = \{\mathcal{T}_{l|\nu_{root}}\}$. Next the set of partitions is expanded until $maxFanout$ non empty partitions are reached by

$$part_{k+1} = part_k \setminus \left\{\mathcal{T}_{l|\tilde{\nu}}\right\} \cup \left\{\mathcal{T}_{l|\tilde{\nu}} | \hat{\nu} \prec \tilde{\nu} \wedge \mathcal{T}_{l|\tilde{\nu}} \neq \emptyset\right\} \tag{7.1}$$

where $\hat{\nu} = \arg\max_{\nu' \in part_k}\{\delta'\}$ is the decision tree node in $part_k$ with the highest information gain and $\prec$ is the parent relation.

A Bayes tree node is generated by creating one entry from each partition, i.e. computing the corresponding cluster feature $(n, \vec{LS}, \vec{SS})$. For each class, this process is repeated recursively until the entire decision tree is read.

ANN. A classification method that creates a *ranking* of training objects is the anytime nearest neighbor classifier [UXKL06] (cf. Chapter 3). The basic idea is to search for the nearest objects by processing the training objects $x \in \mathcal{T}$ during classification in a precomputed order as long as time permits. The order is decreasing w.r.t. the rank of an object x, which is in [UXKL06] computed as

$$rank(x) = \sum_j \begin{cases} 1 & \text{if } l_x = l_{x_j} \\ -2/|\mathcal{L}| & \text{otherwise} \end{cases} \tag{7.2}$$

where x_j is an object that has x as its nearest neighbor based on a leave-one-out evaluation on $\mathcal{T}$.

To create a Bayes tree node for label $\hat{l}$ we pick the $maxFanout$ objects from $\mathcal{T}_l$ with the highest rank according to Equation 7.2 and assign all remaining objects from $\mathcal{T}_l$ to their closest representative. From the resulting subsets the cluster features are calculated. The steps are repeated recursively until the set fits into a leaf.

7.2.2 Parameter Optimization

For the second process we search for a set $\mathcal{A}_{II}$ of feasible approaches that can be used to improve the parameter optimization. π_{BT} in the Bayes tree deter-

mines the parameters (μ_e, Σ_e) for the Gaussians corresponding to the entries. The actions necessary to find good parameter values Θ can be described by *fitting* or *adjusting*. There are numerous approaches that match these terms. Detailed solutions are provided for margin maximization ($\pi_{BT\diamond MM}$), Bayesian networks ($\pi_{BT\diamond BN}$) and Bandwidth estimation ($\pi_{BT\diamond BW}$); further solutions are sketched for combining the Bayes tree with neural networks or SVMs.

MM. An approach for margin maximization that seeks to improve the classification performance of Bayesian classifiers based on Gaussian mixtures has been proposed in [PW10] (referred to as MM). Let Θ be the set of parameter values for the mixture models of all labels $l \in \mathcal{L}$ and $l_x \in \mathcal{L}$ the label of object x. Then MM approximates the multi class margin $d_\Theta(x)$ by

$$d_\Theta(x) = \frac{p(l_x, x|\Theta)}{\max_{l \neq l_x} p(l, x|\Theta)} \approx \frac{p(l_x, x|\Theta)}{\left[\sum_{l \neq l_x} p(l, x|\Theta)^\kappa\right]^{1/\kappa}}. \tag{7.3}$$

using $\kappa \geq 1$. In the global objective

$$D(\mathcal{T}|\Theta) = \prod_{x \in \mathcal{T}} h((d_\Theta(x))^\lambda) \text{ with } \lambda > 0 \tag{7.4}$$

the hinge function $\bar{h}(y) = \min\{2, y\}$ is approximated by a smooth hinge function $h(y)$ that allows to compute the derivative $\partial \log D(\mathcal{T}|\Theta)/\partial\Theta$. The derivatives w.r.t. to the single model parameters $\theta_i \in \Theta$ are then used to iteratively adjust the weights, means and variances of the Gaussians. The employed optimization algorithm is an extended Baum-Welch algorithm [GKNN91]

To determine $\pi_{BT\diamond MM}$ a heuristic is required to extract one mixture for each label $l \in \mathcal{L}$ from the Bayes tree and optimize these using MM. For a set of entries $\mathcal{E} \subset \bigcup_{\mathcal{N}_i \in M} \mathcal{N}_i$ let

$$\Theta(\mathcal{E}) = \bigcup_{e \in \mathcal{E}} \{\mu_e, \Sigma_e\} \tag{7.5}$$

be the corresponding set of parameter values. Initially $\mathcal{E}_0 = \mathcal{N}_{root}$; for each label $l \in \mathcal{L}$ the unimodal model describing $\mathcal{T}_l$ is represented by $\Theta(\mathcal{E}_0)$. After

applying MM to $\Theta(\mathcal{E}_0)$ the corresponding parameters are updated in the Bayes tree. Subsequent steps descend the Bayes tree and set

$$\mathcal{E}_{i+1} = \bigcup_{e \in \mathcal{E}_i} \begin{cases} \mathcal{N}_e & \text{if } p_e \neq \text{null} \\ \{e\} & \text{otherwise.} \end{cases} \tag{7.6}$$

As above, the sets $\mathcal{E}_i$ are optimized using MM and the Bayes tree parameters are updated. An additional parameter is the maximal number m of steps taken in Equation 7.6: for $m = 2$ only the upper two levels of the Bayes tree are optimized.

BN. The second studied approach are Bayesian networks to fit the Gaussians (μ_e, Σ_e) in the Bayes tree to the underlying training data $\mathcal{T}_{|e}$. For each entry e, π_{BT} computes the variance $\sigma_{e,ii}$ per dimension (cf. Equation 3.2) and sets all covariances $\sigma_{e,ij}$ to 0, i.e. Σ constitutes a diagonal matrix of a naive Bayesian approach. The space demand w.r.t. the dimensionality d is $O(d)$ compared to $O(d^2)$ for a full covariance matrix. Introducing full covariance matrices affects every single entry in the Bayes tree, i.e. $O(d^2)$ covariances must be stored per entry. Moreover, the time complexity for the evaluation of the Gaussian density function (cf. Definition 2.3) increases from $O(d)$ to $O(d^2)$. To avoid this quadratic time and space demand a hill climbing method for Bayesian network learning (as e.g. proposed in [KP02]) is adapted here to find the most important correlations.

The goal is to add important correlations at low time and space complexity. To this end a maximal block size s is fixed and Σ is constrained to have a block structure:

$$\Sigma = \text{diag}(B_1, \ldots, B_u) \tag{7.7}$$

where $B_i \in \mathbb{R}^{\mathrm{s}_i \times \mathrm{s}_i}$ are quadratic matrices of block size s_i and $\mathrm{s}_i \leq \mathrm{s}\ \forall i = 1 \ldots u$. The resulting space demand is in $O(d \cdot \mathrm{s})$. So is the time complexity, since the exponent in the Gaussian distribution (cf. Definition 2.3) can be decomposed into a sum as follows:

$$z\Sigma^{-1}z^T = \sum_{i=1}^{u} z[L_i..U_i]B_i^{-1}z[L_i..U_i]^T \tag{7.8}$$

where $z = (x - \mu)$, $L_i = 1 + \sum_{j=1}^{i-1} \mathbf{s}_j$, $U_i = L_i + \mathbf{s}_i - 1$ and $z[L_i..U_i]$ selects dimensions L_i to U_i from z.

A Bayesian network $\mathcal{B} = \langle G, \Theta \rangle$ is characterized by a directed acyclic graph G and a set of parameter values Θ. $G = (V, E)$ contains one vertex $i \in V$ for each dimension and an edge $(i, j) \in E$ induces a dependency between dimensions i and j. An undirected graph G' can easily be derived from G by removing the orientation. To derive Σ an edge $(i, j) \in E'$ is transferred to a covariance σ_{ij} and its symmetric counter part σ_{ji} in Σ. To ensure a block structure, a permutation P is applied to Σ by $P\Sigma P^{-1}$ that groups dimensions, which are connected in G', as blocks along the diagonal. P is then also applied to z before computing Equation 7.8. Since P is just a reordering of the dimensions, it can be stored in an array of size d and its application to z is in $O(1)$ per feature.

Creating the block covariance matrix for an entry e starts with an empty Bayes net; initially $E' = \emptyset$ and $\Sigma_e = \text{diag}(\sigma_{e,11}, \ldots, \sigma_{e,dd})$. From the edges that can be added to G' without violating the maximal block constraint in the resulting block matrix, iteratively the one is selected that maximizes the likelihood of e given $\mathcal{T}_{|e}$. Since all objects $x \in \mathcal{T}_{|e}$ have the same label l, the log likelihood is

$$LL(e|\mathcal{T}_{|e}) = \sum_{x \in \mathcal{T}_{|e}} \log\left(p(l, x|(\mu_e, \Sigma_e))\right) \tag{7.9}$$

The iterations stop when either the resulting matrix does not allow for further additions or no additional edge improves Equation 7.9.

In general, a block covariance matrix is determined for each entry in the Bayes tree. However, on lower levels of the Bayes tree the combination of single components is likely to capture already the main *directions* of the data distribution. This may render the additional degrees of freedom given by the covariances useless or even harmful, since they can lead to overfitting. In Section 7.3, in addition to $\mathbf{s}$, the influence of restricting the BN optimization to the upper m levels of the Bayes tree is evaluated.

BW. In its leaves the Bayes tree stores d dimensional Gaussian kernels. The kernel bandwidth h_i, $i = 1 \dots d$, is a parameter that is chosen per dimension by π_{BT} and that can be optimized by other methods for bandwidth estimation. A method from [Sil86] uses

$$h_i = \sqrt[d+4]{\frac{4}{(d+2) \cdot |\mathcal{T}|}} \cdot \widehat{\sigma}_i \tag{7.10}$$

where $\widehat{\sigma}_i$ is the standard deviation of $\mathcal{T}$ in dimension i. A second method [JL95] determines the bandwidth as

$$h_i = \frac{max_i - min_i}{\sqrt{|\mathcal{T}|}} \tag{7.11}$$

where max_i and min_i are the maximal and minimal values occuring in $\mathcal{T}$ in dimension i. Finally, a family of bandwidths

$$h_i = \alpha \cdot \widehat{\sigma}_i \tag{7.12}$$

is tested in Section 7.3, where a factor α is multiplied to $\widehat{\sigma}_i$. In $\pi_{BT \diamond BW}$ we refer to the three methods in Equations 7.10, 7.11 and 7.12 as *haerdle*, *langley* and *f* α respectively.

SVM and Neural networks. Combining Gaussian mixture models and SVMs has for example been studied in [DHN10], where a mapping from Gaussian mixtures to SVMs (and vice versa) is defined, that yields the exact same decision boundaries for both approaches. Using such a mapping $\pi_{BT \diamond SVM}$ can be designed similar to $\pi_{BT \diamond MM}$, i.e. one mixture model for each label is extracted from the Bayes tree, optimized using SVM and the new parameters are transferred back to the tree. Analogously back propagation as in neural networks using RBF kernels as neurons can by exploited to design $\pi_{BT \diamond NNet}$. For $|\mathcal{L}| > 2$, it must be ensured that the optimizer returns a single set of parameters per Gaussian. Also the bias resulting from SVM or $NNet$ training must be integrated into BT.

7.2.3 Decision Design

Let $\mathcal{F}_l(t)$ denote the frontier of label $l \in \mathcal{L}$ at time t. Then the decision function for the Bayes tree is

$$f_{BT}(\Theta, x, t) = \arg\max_{l \in \mathcal{L}} \left\{ P(l) \cdot \sum_{e \in \mathcal{F}_l(t)} \frac{n_e}{n_l} \mathbf{g}(x, \mu_e, \sigma_e) \right\} \tag{7.13}$$

To estimate the class conditional density for a class $\hat{l}$ at time t the entries that are stored in the current frontier $\mathcal{F}_{\hat{l}}(t)$ are evaluated and summed up according to their weight. Since that set of entries is a subset of all entries in the hierarchy, an appropriate term to describe the action necessary for the decision design is *selecting*.

NN. The nearest neighbor classifier finds a decision based on the object that is the closest to the query object x, i.e. it *selects* only the most promising object from $\mathcal{T}$. This concept can be transferred to the Bayes tree in a straightforward way by using the modified decision function

$$f_{BT \diamond NN}(\Theta, x, t) = \arg\max_{l \in \mathcal{L}} \left\{ P(l) \cdot \max_{e \in \mathcal{F}_l(t)} \left\{ \frac{n_e}{n_l} \mathbf{g}(x, \mu_e, \sigma_e) \right\} \right\} \tag{7.14}$$

Variants of the nearest neighbor classifier use k closest objects. The actual label can then be assigned based on a simple majority voting among the neighbors or taking their distance or the prior probability of the labels into account (cf. Chapter 2). In the experiments the standard nearest neighbor concept with $k = 1$ is tested for $f_{BT \diamond NN}$.

ENS. A second concept that matches the term *selecting* is the concept of ensemble classifiers. As discussed in Section 7.1 ensembles are a frequently used approach. The concept of ensembles is simple and can easily be transferred to any classification method. A straightforward way for the Bayes tree would be to build several tree structure, e.g. using different samples of the training data, and combine the individual outcomes to achieve a classification decision. The method we propose here uses a single Bayes tree and builds an ensemble over time.

In the Bayes tree so far only the most recent frontiers $\mathcal{F}_l(t)$ were used in the decision function. Transferring the ensemble concept to the Bayes tree, all previous frontiers are combined in the modified decision function

$$f_{BT\diamond ENS}(\Theta, x, t) = \arg\max_{l\in\mathcal{L}} \left\{ P(l) \cdot \sum_{s=0}^{t} \sum_{e\in\mathcal{F}_l(t-s)} \frac{n_e}{n_l} \mathbf{g}(x, \mu_e, \sigma_e) \right\} \tag{7.15}$$

The additional computational cost when using the ensemble decision from Equation 7.15 compared to the original decision from Equation 7.13 is only a single operation per class. More precisely, we just have to add the most recent density, which we compute also in the original Bayes tree, to an aggregate of the previous densities. Since the same amount of frontiers are summed up for each label, we can skip the normalization without changing the decision and do not have to account for additional operations.

7.2.4 Summary

Figure 7.1 summarizes the concept transfer approach as well as the results of the single steps in the Bayes tree. The steps are generally applicable to a classifer $\mathcal{C}$. The **first step** is to identify the processes of $\mathcal{C}$, which resulted in I – model selection, II – parameter optimization and III – decision design (cf. Eqs. 2.1 - 2.4). The **second step** is to analyze the actions necessary in $\mathcal{C}$ for the identified processes and characterize them by expressive terms (cf. Figure 7.1). The **third step** is to assign approaches $A_i \in \mathcal{A}$ from a set $\mathcal{A}$ of known approaches, whose concepts match the terms defined in step 2. The approaches discussed for the Bayes tree in the three processes I, II and III are $\{ANN, DT\} \subset \mathcal{A}_I$, $\{MM, BW, BN\} \subset \mathcal{A}_{II}$ and $\{NN, ENS\} \subset \mathcal{A}_{III}$ (cf. Figure 7.1). For the **fourth step** we first test the effects of the single improvements separately, e.g. $BT \diamond ANN$ or $BT \diamond MM$ (cf. Section 7.3). In further experiments we then seek to find the best combination of several improvements, e.g. $BT \diamond ANN \diamond BW \diamond ENS$.

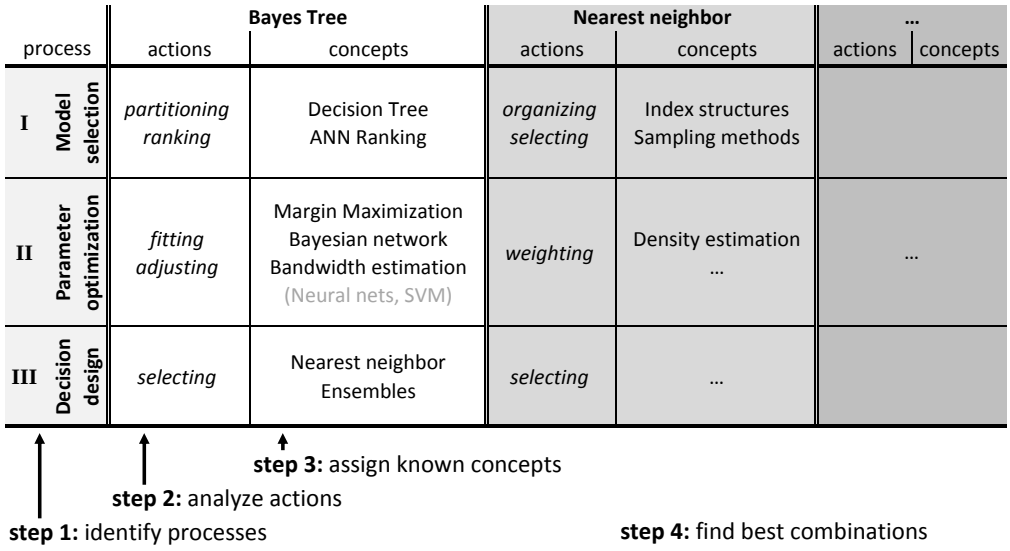


Figure 7.1: Summary of the concept transfer approach.

7.3 Experiments

In all experiments the results for the first $r = 200$ refinements are reported. To compare the different approaches the average accuracy avg is used as well as the maximal accuracy max over all refinements. The third employed objective, which penalizes descending or oscillating anytime curves, is the monotonicity

$$mon = 1 - \frac{1}{r}\sum_{n=1}^{r} \widehat{acc}(n) - \min\{\widehat{acc}(n), acc(n)\}$$

where $\widehat{acc}(n) = \max_{1 \leq n' < n} acc(n')$ is the maximal accuracy over all $n' < n$. When choosing best results they are selected according to a linear combination of all three measures with equal weights. For the Bayes tree the bandwidth heuristic from Equation 7.11 is used for the baselines and $maxFanout = 7$. The approaches are tested on the following data sets (available at [FA10] except [AS04] for gender):

name	#obj	d	$\lvert\mathcal{L}\rvert$	name	#obj	d	$\lvert\mathcal{L}\rvert$
page-blocks	5473	10	5	pendigits	10992	16	10
optdigits	5620	64	10	vowel	990	10	11
letter	20000	16	26	spambase	4601	57	2
segment	2310	19	7	gender	189961	9	2
kr-vs-kp	3196	36	2	covtype	581012	10	7

Model selection. The results for the modified model selections are shown in Figure 7.2, listing for both approaches the absolute differences in the objectives compared to the two baselines R and EM. Both ANN and DT show better performance than the R baseline for nearly all data sets and measures. Compared to the EM baseline an exceptional improvement (ANN:12.5% and DT:6.9% for avg) is reached by both approaches for the covtype data set, which is the largest tested data set. On the other data sets they are comparable or even slightly worse than the EM baseline.

	ANN-Ranking (BT ◊ ANN)						**Decision Tree** (BT ◊ DT)					
	diff. to R baseline			**diff. to EM baseline**			**diff. to R baseline**			**diff. to EM baseline**		
	avg	max	mon	avg	max	mon	avg	max	mon	avg	max	mon
page-blocks	**0.9%**	**0.5%**	-6.4%	-0.5%	-0.5%	-6.7%	**1.4%**	**1.0%**	-1.0%	0.0%	**0.0%**	-1.3%
optdigits	**4.0%**	**3.1%**	**9.7%**	-0.4%	-0.1%	-1.1%	**3.6%**	**2.8%**	**10.3%**	-0.7%	-0.4%	-0.7%
letter	**11.2%**	**11.3%**	-0.4%	-0.6%	-0.4%	**1.1%**	**9.2%**	**10.4%**	-0.2%	-2.6%	-1.3%	**0.8%**
segment	**4.7%**	**2.7%**	**20.3%**	**1.5%**	**1.1%**	**6.9%**	**2.3%**	**2.1%**	**5.1%**	-0.9%	**0.5%**	-8.3%
kr-vs-kp	**9.0%**	**5.0%**	**2.7%**	-0.5%	-0.1%	-0.2%	**9.7%**	**5.5%**	**2.7%**	**0.2%**	**0.4%**	-0.2%
pendigits	**5.2%**	**4.2%**	**11.4%**	**0.4%**	**0.1%**	**2.1%**	**4.4%**	**3.7%**	**9.3%**	-0.4%	-0.3%	**0.1%**
vowel	**0.2%**	0.0%	**1.1%**	**0.3%**	0.0%	-1.0%	**0.0%**	**0.5%**	**1.1%**	**0.2%**	**0.5%**	-0.6%
spambase	**1.1%**	**0.2%**	-7.5%	0.0%	**0.4%**	-11.8%	-0.3%	-0.7%	**6.0%**	-1.4%	-0.5%	**1.7%**
gender	**5.0%**	**6.1%**	**1.1%**	-2.4%	-1.8%	**1.1%**	**6.5%**	**7.8%**	**0.7%**	-0.9%	-0.1%	**0.6%**
covtype	**18.9%**	**22.4%**	**3.0%**	**12.5%**	**12.0%**	**4.9%**	**15.9%**	**18.9%**	**1.9%**	**6.9%**	**4.9%**	**3.6%**
averages	**6.0%**	**5.5%**	**3.5%**	**1.0%**	**1.1%**	-0.5%	**5.3%**	**5.2%**	**3.6%**	**0.1%**	**0.4%**	-0.4%

Figure 7.2: Model selection approaches.

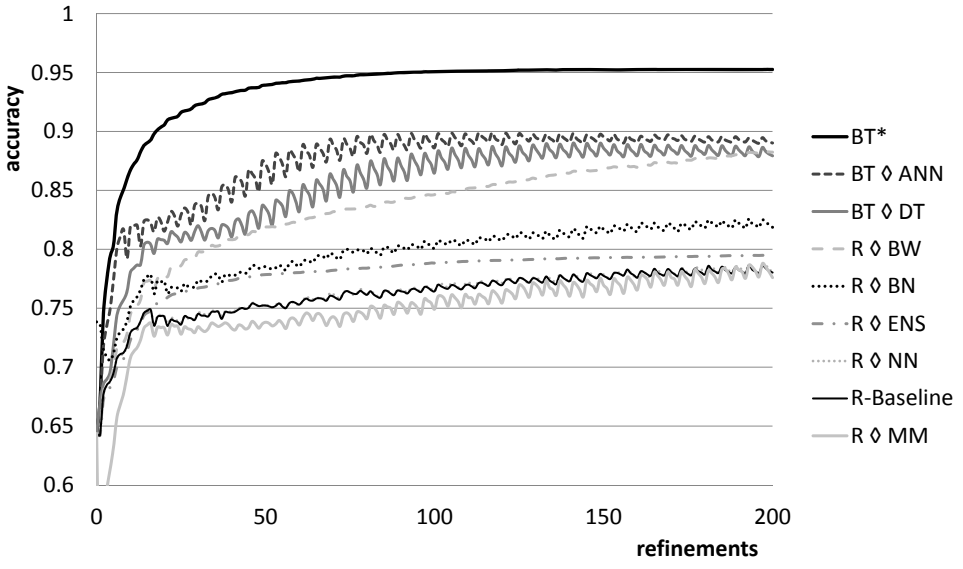


Figure 7.3: Effects of single approaches on letter.

	Bandwidth Estimation (BT ◊ BW)					
	diff. to R baseline			diff. to EM baseline		
	avg	max	mon	avg	max	mon
page-blocks	**1,5%**	**1,1%**	**5,2%**	**0,6%**	-0,3%	**9,8%**
optdigits	**0,7%**	**0,8%**	**2,4%**	0,0%	0,0%	0,0%
letter	**7,3%**	**9,5%**	**2,9%**	**5,5%**	**4,7%**	**3,7%**
segment	**3,9%**	**2,1%**	**24,8%**	**6,2%**	**3,1%**	**23,5%**
kr-vs-kp	0,0%	0,0%	0,0%	**1,1%**	**0,5%**	-1,4%
pendigits	**2,7%**	**2,9%**	**11,4%**	**1,1%**	**0,8%**	**3,0%**
vowel	**6,0%**	**4,1%**	**8,0%**	**4,9%**	**3,8%**	**6,4%**
spambase	**0,8%**	-0,3%	**1,2%**	0,0%	0,0%	0,0%
gender	**2,6%**	**3,9%**	**1,2%**	**3,5%**	**3,9%**	**1,3%**
covtype	**3,4%**	**5,6%**	**3,0%**	**14,3%**	**12,3%**	**5,4%**
averages	**2,9%**	**3,0%**	**6,0%**	**3,7%**	**2,9%**	**5,1%**

Figure 7.4: Approaches for parameter optimization I (BW).

Figure 7.3 shows the anytime accuracy results for the single approaches and the R baseline as a comparison. BT^* refers to the final combination that is found as the result of the evaluation and which is introduced at the end of the section. $BT \diamond ANN$ and $BT \diamond DT$ are located between the baseline and the final solution, i.e. they yield a significant improvement, but are outperformed by the combination in BT^*. Moreover, neither of the model selection approaches can diminish the oscillation of the anytime curve, which is achieved by other concepts as we see in the following.

Parameter optimization. Figures 7.4 and 7.5 show the results for the tested parameter optimization approaches. The shown results for the bandwidth estimation in Figure 7.4 are the best results over the three heuristics from Equations 7.10 to 7.12, where for the $f\alpha$ heuristic the following α values were tested: $\alpha \in \{0.001, 0.005, 0.01, 0.05, 0.1, 0.5\}$. In the 20 results ($EM$ and R on 10 data sets) *haerdle* and *langley* were both chosen three times, twice $f0.01$ was the best choice and the remaining results were achieved by $f0.05$. By the optimized bandwidth estimation all accuracy values, i.e. avg and max on both R and EM, could be improved with the exception of max for EM on page-blocks and max for R on spambase. The largest improvement is achieved for the monotonicity with the single exception of EM on kr-vs-kp. The anytime plot in Figure 7.3 illustrates the good performance of $BT \diamond BW$ in all three measures.

	Bayesian Network (BT ◊ BN)						**MaxMargin** (BT ◊ MM)					
	diff. to R baseline			**diff. to EM baseline**			**diff. to R baseline**			**diff. to EM baseline**		
	avg	max	mon	avg	max	mon	avg	max	mon	avg	max	mon
page-blocks	**0,2%**	**0,3%**	-18,0%	**0,1%**	-0,2%	-8,9%	-0,9%	-0,5%	-1,1%	-0,5%	-0,4%	**4,7%**
optdigits	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	-1,0%	-0,3%	**8,3%**	-0,6%	-0,3%	**0,2%**
letter	**3,6%**	**4,0%**	-1,7%	**1,5%**	**1,5%**	-1,1%	-1,3%	**0,1%**	**0,4%**	-1,5%	-0,6%	**1,3%**
segment	**0,0%**	-0,3%	**10,8%**	-0,1%	-0,1%	**6,4%**	-5,8%	-2,6%	**23,9%**	-1,3%	-0,7%	**20,2%**
kr-vs-kp	**0,0%**	0,0%	-0,5%	**0,0%**	0,0%	-0,1%	-0,5%	-1,1%	-2,3%	-18,7%	-13,1%	-17,4%
pendigits	**1,8%**	**2,0%**	-3,4%	**0,2%**	**0,1%**	-2,5%	0,0%	**0,3%**	**3,3%**	-0,6%	-0,5%	**1,1%**
vowel	**0,2%**	**0,9%**	-5,1%	**0,3%**	**0,1%**	-1,9%	-0,8%	-1,1%	**4,7%**	-1,8%	-1,5%	**4,4%**
spambase	0,0%	0,0%	**0,7%**	0,0%	0,0%	**0,1%**	-7,1%	-5,1%	-0,2%	-6,3%	-3,1%	-1,9%
gender	0,0%	0,0%	0,0%	**0,0%**	0,0%	-0,2%	**1,1%**	**1,3%**	**5,9%**	-4,7%	-3,3%	**0,7%**
covtype	-2,4%	-2,4%	**2,3%**	-0,3%	-0,3%	**1,7%**	-	-	-	-	-	-
averages	**0,3%**	**0,5%**	-1,5%	**0,2%**	**0,1%**	-0,7%	-1,8%	-1,0%	**4,8%**	-4,0%	-2,6%	**1,5%**

Figure 7.5: Approaches for parameter optimization II (MM and BN).

The performance of transferring the Bayesian network approach to BT is hardly better than any of the two baselines. As above, the results shown in Figure 7.5 are the best among all parameter settings for $BT \diamond BN$, i.e. over all block sizes $\mathbf{s}$ and numbers of levels m (cf. Section 7.2.2). The largest improvements are achieved on the letter data set with $\mathbf{s} = d$ and $m = 1$. The performance gain was less for smaller block sizes (results not shown). As assumed in Section 7.2.2, the additional degrees of freedom gained through the covariances are useless or even harmful on lower levels of the tree, i.e. the displayed results, which are the best among all m and $\mathbf{s}$, all use $m = 1$ and $\mathbf{s} = d$. Nonetheless, as stated above, the performance gain is only marginal on 9 of 10 data sets.

The margin maximization approach $BT \diamond MM$ does not add covariances but only seeks to improve the weights, means and variances of the Gaussians in the Bayes tree. The results on the 10 tested data sets are shown in Figure 7.5 (The results for covtype could not be achieved with 4GB RAM). As above the best results over all parameter settings are shown where $\kappa \in \{1..10\}$, $\lambda \in [0, 10]$ and m from 1 to the maximal tree height. On average $BT \diamond MM$ improves the monotonicity over the baselines, but neither of the accuracy measures. The detailed results show slight improvements over the R baseline on three data sets. This result is surprising at first sight, since the original concept from [PW10] is designed for improving Gaussian mixture models.

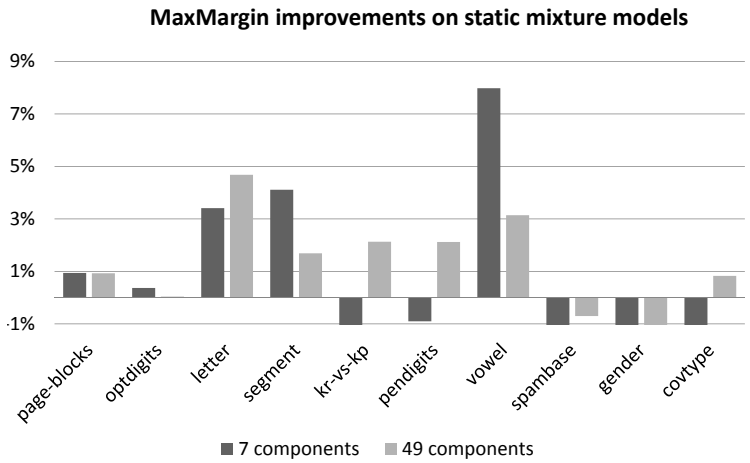


Figure 7.6: Results for MM on static mixture models of size 7 and 49.

To exclude the possibility that the poor performance of $BT \diamond MM$ is solely due to the data set characteristics, the implementation of MM is tested on static Gaussian mixture models. Using the same EM clustering that was used for constructing the Bayes tree in the EM baseline, for each data set two mixtures were created, each containing one model per class. In the first mixture each model has 7 components, in the second 49 components. (Multiples of 7 were chosen since they correspond to the chosen fanout of the Bayes tree.) The plots show the resulting absolute gain in classification accuracy from the optimized over the initial mixtures in Figure 7.6. MM improves the accuracy for at least one mixture on 8 data sets and for both mixtures on 5 of 10 data sets in the experiments. For the vowel data set the improvement is 3% for the 49 components and nearly 8% for the 7 components. However, neither avg nor max are improved by $BT \diamond MM$ on vowel (cf. Fig. 7.5). One reason for this is the fact that $\pi_{BT \diamond MM}$ optimizes the mixtures in the Bayes tree level by level, but the decision f_{BT} uses arbitrary mixtures that can contain components from many different levels of the tree. These components, or rather these mixtures, were never optimized together. Optimizing all possible mixtures is not feasible, since on the one hand the sheer number of possible mixtures makes the optimization computationally infeasible, and on the other hand such an approach does not yield a single set of parameters values per Gaussian.

	Ensemble (BT ◊ ENS)						**Nearest Neighbor** (BT ◊ NN)					
	diff. to R baseline			**diff. to EM baseline**			**diff. to R baseline**			**diff. to EM baseline**		
	avg	max	mon	avg	max	mon	avg	max	mon	avg	max	mon
page-blocks	**1.2%**	**0.4%**	**10.7%**	**0.8%**	-0.2%	**9.7%**	**0.7%**	**0.5%**	**3.8%**	-46.3%	-0.8%	-47.2%
optdigits	-1.1%	-2.0%	**11.4%**	-3.2%	-3.5%	**1.1%**	**0.0%**	0.0%	**0.1%**	0.0%	0.0%	0.0%
letter	**1.7%**	**0.7%**	**3.1%**	**1.6%**	**0.0%**	**3.8%**	**0.1%**	**0.2%**	-0.3%	-0.2%	**0.3%**	**1.5%**
segment	**5.9%**	**1.5%**	**37.7%**	**4.2%**	**0.5%**	**24.3%**	**0.0%**	0.0%	-0.1%	**4.4%**	**2.4%**	**18.6%**
kr-vs-kp	-1.7%	-3.4%	**2.0%**	-2.7%	-1.6%	**0.7%**	-0.1%	-0.1%	-1.1%	**0.1%**	**0.1%**	-0.2%
pendigits	**2.0%**	**1.1%**	**12.5%**	**0.3%**	-0.1%	**3.1%**	**0.1%**	**0.0%**	**0.8%**	**0.6%**	**0.3%**	**2.7%**
vowel	**2.4%**	**0.7%**	**8.1%**	**2.8%**	**1.1%**	**4.8%**	**4.9%**	**4.0%**	**7.4%**	**3.6%**	**3.9%**	**5.4%**
spambase	**2.1%**	-0.2%	**12.9%**	-4.0%	-5.7%	-3.8%	**0.0%**	0.0%	**0.0%**	-0.2%	-0.2%	-0.6%
gender	**0.4%**	**1.5%**	**1.3%**	**2.3%**	**1.7%**	**1.4%**	-2.1%	-1.7%	-4.4%	-3.4%	-0.8%	-17.3%
covtype	**2.3%**	**3.4%**	**3.0%**	**6.2%**	**1.8%**	**5.4%**	-0.5%	-0.1%	-3.2%	-11.9%	-6.9%	-37.9%
averages	**1.5%**	**0.4%**	**10.3%**	**0.8%**	-0.6%	**5.1%**	**0.3%**	**0.3%**	**0.3%**	-5.3%	-0.2%	-7.5%

Figure 7.7: Decision design approaches.

Decision design. For the process of decision design $f_{BT\diamond ENS}$ and $f_{BT\diamond NN}$ were tested; the absolute improvements for the three objectives are shown in Figure 7.7. Combining the ensemble concept with the Bayes tree (cf. Equation 7.15) yields on average a slight increase in the accuracy measures avg and max for the R baseline and is rather neutral on the EM baseline. The monotonicity, however, is drastically increased by $BT\diamond ENS$, on average by more than 10% compared to the R baseline and more than 5% compared to the EM baseline. This is underlined by the anytime accuracy plot of $BT\diamond ENS$ in Figure 7.3, which shows a smooth and monotonically increasing behaviour.

In contrast, the results of $BT \diamond NN$ hardly show any improvement over the R baseline except for vowel. The anytime plot for $BT \diamond NN$ is hardly visible in Figure 7.3, since it coincides with the curve of the R baseline. This is another surprising result. It signifies that taking per label only the one single Gaussian, which yields the highest class conditional density for the query object, results in almost exactly the same decisions as taking the entire mixture models. As can be seen in Figure 7.7, this strongly holds for 7 of the 10 tested data sets on the R baseline. Compared to the EM baseline the performance of $BT \diamond NN$ is worse on average.

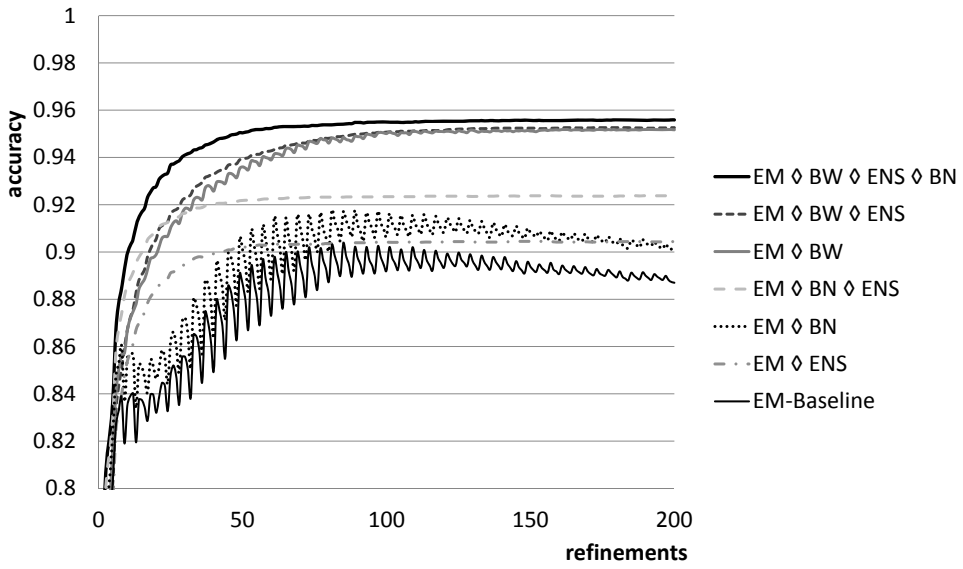


Figure 7.8: Anytime plots for combined approaches.

Combining approaches. Next we study the cumulative performance gain when combining BT with more than one concept. Figure 7.8 shows the anytime accuracy plots on letter for the EM baseline and the combined versions using ENS and/or BN and/or BW. The curve of the EM baseline exhibits a strong oscillation. $EM \diamond BN$ improves the accuracy throughout on this data set, but cannot diminish the oscillation. Near perfect monotonicity is reached when using ENS, either alone ($EM \diamond ENS$) or combined ($EM \diamond BN \diamond ENS$). The cumulation of the two positive effects, i.e. increased accuracy and monotonicity, is clearly expressed by the corresponding anytime plots. $EM \diamond BW$ pushes up the accuracy and can also improve the monotonicity. Adding ENS yields again near perfect monotonicity (cf. $EM \diamond BW \diamond ENS$). Finally combining the three concepts with EM yields the best results in this setting.

To find the globally best results all combinations are allowed and per data set the best setting is selected with respect to the linear combination of all three measures. This is denoted as *free choice*, the settings and corresponding results are shown in Figure 7.9 along with the two baselines and the results for BT^* (see below). The values shown are the absolute values for the measures. Highlighted cells indicate an improvement over both baselines.

	R baseline			EM baseline		
	avg	max	mon	avg	max	mon
page-blocks	94,0%	95,9%	87,0%	95,4%	96,8%	87,3%
optdigits	92,9%	94,6%	87,9%	97,2%	97,8%	98,8%
letter	76,1%	78,8%	96,8%	87,9%	90,5%	96,1%
segment	86,3%	92,0%	59,3%	89,4%	93,6%	72,6%
kr-vs-kp	84,4%	90,1%	96,1%	93,9%	95,2%	99,0%
pendigits	93,1%	94,6%	87,3%	97,9%	98,7%	96,6%
vowel	91,0%	94,4%	90,8%	90,8%	94,4%	92,5%
spambase	87,4%	91,5%	81,4%	88,5%	91,4%	85,7%
gender	73,1%	73,9%	98,4%	80,5%	81,8%	98,5%
covtype	62,4%	63,1%	96,0%	71,4%	77,2%	94,4%
averages	84,1%	86,9%	88,1%	89,3%	91,7%	92,2%

	BT*			free choice						
	avg	max	mon	avg	max	mon	buildup	bw	ens	opt
page-blocks	**96,6%**	**96,9%**	**98,2%**	**96,5%**	**97,0%**	**99,1%**	ANN	langley	yes	
optdigits	96,8%	97,0%	**100,0%**	96,9%	97,2%	**99,9%**	EM	f0.05	yes	MM
letter	**93,5%**	**95,3%**	**99,9%**	**94,5%**	**95,7%**	**99,9%**	EM	f0.05	yes	BN
segment	**94,4%**	**95,1%**	**96,4%**	**95,6%**	**96,1%**	**99,7%**	DT	langley	yes	
kr-vs-kp	**93,9%**	94,6%	**99,3%**	**98,0%**	**98,6%**	**99,3%**	DT	f0.05	yes	
pendigits	**99,0%**	**99,4%**	**99,7%**	**99,0%**	**99,4%**	**99,7%**	EM	f0.05	yes	
vowel	**97,4%**	**98,7%**	**99,6%**	**97,8%**	**98,6%**	**99,9%**	EM	f0.05	yes	MM
spambase	**91,9%**	**92,5%**	**98,5%**	**91,9%**	**93,0%**	**98,4%**	ANN	f0.05	yes	
gender	**84,3%**	**85,7%**	**99,9%**	**84,4%**	**86,2%**	**99,9%**	DT	f0.05	yes	
covtype	**79,8%**	**82,7%**	**99,8%**	**92,1%**	**94,9%**	**100,0%**	ANN	f0.05	yes	
averages	**92,8%**	**93,8%**	**99,1%**	**94,7%**	**95,7%**	**99,6%**				

Figure 7.9: Baseline results, results for BT*, and the globally best results and setups.

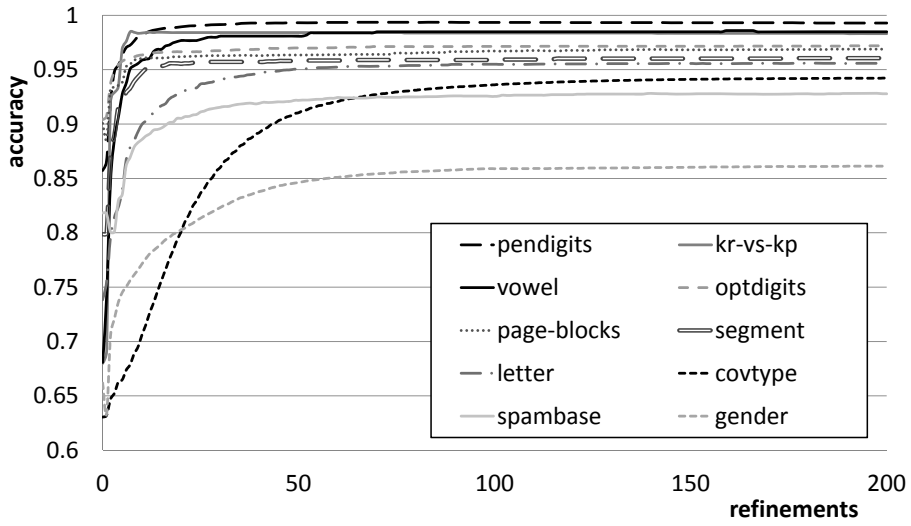


Figure 7.10: Anytime accuracy results for BT^*.

On all data sets all three measures are improved by the *free choice* except for avg and max on optdigits, where the actual values are slightly smaller (due to picking the best result according to the linear combination of all three measures). In the settings of *free choice*, ENS was used on all data sets and $f0.05$ was used eight times for BW, other parameter optimizations were rarely employed (once BN and twice MM). For the model selection, i.e. the *buildup* of the Bayes tree, EM, ANN and DT have a roughly equal shares.

To have one final setting that is used on all data sets $BT^* = EM \diamond f0.05 \diamond ENS$ is chosen. The choice of EM is due to the fact that the improvement of ANN and DT over EM on average is mainly only due to the covtype data set (cf. Figure 7.2). As can be seen in Figure 7.9, BT^* exhibits improvements over both baselines similar to *free choice* except for max on kr-vs-kp, where it shows slightly worse performance compared to EM, and for the covtype data set as discussed above.

Overall, improving BT to BT^* by transferring three concept yields very good results on all tested data sets. Figure 7.10 shows the anytime accuracy plots for BT^* which illustrate the great performance w.r.t. to all three measures. Figure 7.11 visually summarizes the average results of the single concepts, *free choice* and BT^*, underlining the effectiveness of the concept transfer approach to classifier improvement.

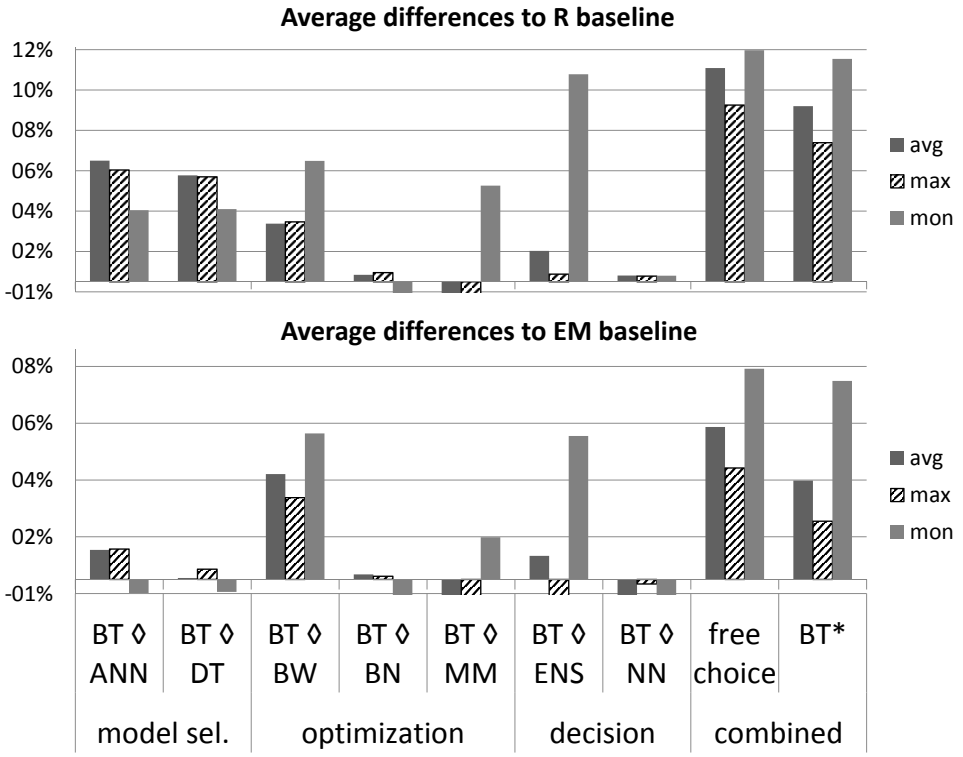


Figure 7.11: Average absolute differences over all 10 data sets for single and combined approaches.

7.4 Conclusion

In this chapter related concepts were transferred to the Bayes tree to improve its anytime classification performance. Experimental evaluation showed the effects of the single improvements and yielded a best combination of several concepts, that outperformed the baselines on a multitude of data sets. The improved version BT^* of the Bayes tree that resulted from the investigations in this chapter shows near perfect results on all tested data sets (cf. Figure 7.10). BT^* constitutes the final version of the proposed Bayesian anytime classifier as a first result of this thesis. In the remainder of Part II an application of the Bayes tree is presented in Chapter 8. In Chapter 9 novel approaches are introduced to harness the strengths of anytime algorithms on constant streams, future work in the area of anytime stream classification is discussed in Chapter 10.

Chapter 8

Application: Anytime Classification in HealthNet Scenarios

* Demographic change will pose a major challenge for our society as people live longer due to enhanced living conditions. Especially the increasing amount of elderly people which need medical assistance and supervision must be considered. In todays health environments medical supervision is an expensive task as it can only be achieved by health professionals in hospitals or other health facilities. A major contribution in this area is the development of remote monitoring systems that provide health assistance and supervision at home. Such a health monitoring provides not only economical savings as it can save expensive health professional costs, but it also increases the living quality of elderly people as they can stay in their familiar environment. One major task for such a remote health monitoring is to provide an emergency detection system. Based on a semi-automated classification of a patient's situation the system should automatically detect an emergency to pose an alert. This alert is then manually verified by a health professional. The applicability of this scenario to emergency detection has been shown as proof

*This chapter has been published in *Plant C., Bhm C. (eds.): Database Technology for Life Sciences and Medicine, World Scientific Publishing* [KMA+10] and presented at the 9th IEEE International Conference on Mobile Data Management (MDM 2008) [KKK+08].

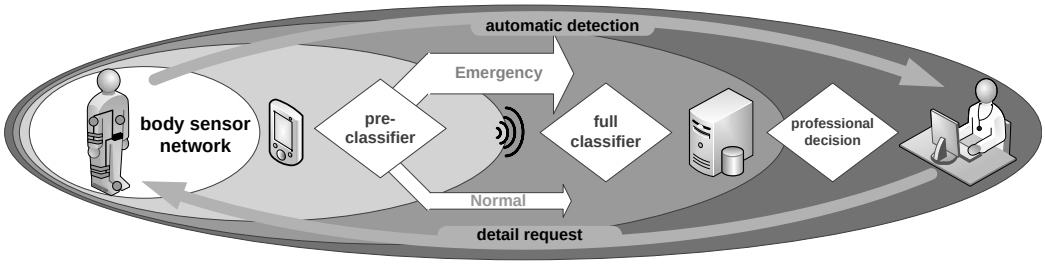


Figure 8.1: Multi-step patient health classification from the body sensor network up to the health professional.

of concept in a framework for "Mobile Mining and Information Management in HealthNet Scenarios" [KKK+08].

8.1 Scenario and Prototype

Health monitoring tasks pose major challenges for data processing. In general, automated monitoring produces a huge amount of stream data that cannot be manually handled by humans. An automated analysis to assist the health professionals is essential for a scalable system monitoring hundreds of patients. Although automated analysis cannot take the final decision about an emergency, it can dramatically reduce the number of patients by highlighting the most urgent cases. The health professional can then focus on a more detailed analysis of these patients, which leads to an overall scalable semi-automated processing.

For the data analysis components in such an emergency detection system there are special requirements derived out of the health surveillance application. Due to the huge amount of streaming data one must consider scalability issues for the automated analysis. While servers in a data farm may analyze terabytes of data, computers in hospitals can only process gigabytes of data. Mobile clients, which collect the data from sensors, have even less resources and may process megabytes of data. Even the sensors in a body sensor network have capabilities of processing some kilobytes of data, however only with very restricted algorithms. A requirement in today's het-

erogeneous architectures is thus to design an adaptive analysis which scales with the available resources in a multi-step approach.

The general aim of such a multi-step classification is to provide an adaptive classifier applicable on various layers of the health surveillance process. Especially for mobile devices with limited resources an efficient pre-classifier is required. Based on the decision of the pre-classification the required level of detail of the sensor measurements is decided. As shown in Figure 8.1 the full sensor information is sent only in emergency cases while only aggregated information is sent in normal situations. This way adaptive aggregation minimizes the communication cost and the energy consumption of the mobile devices. Every decision is refined on the next layer of the multi-step classification, since the pre-classification is tailored for limited resources and its lower classification accuracy may lead to false positives and false negative decisions.

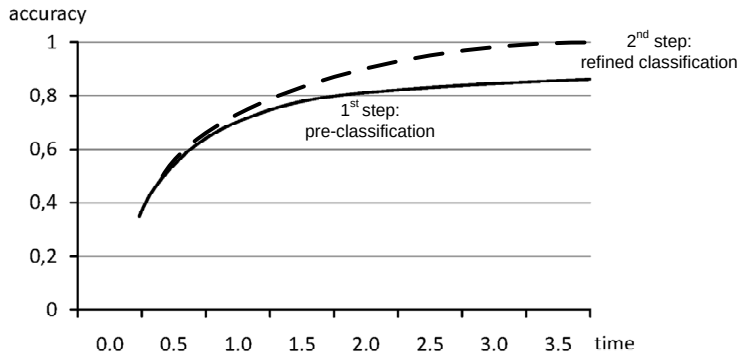


Figure 8.2: Multi-step anytime classification accuracy.

Each of the classifiers on the different levels must cope with streams of variable size and speed. This is on the mobile device due to the varying amount and frequency of data sent by the patients depending on their condition as preclassified locally on the sensor controller. The receiving server must classify all incoming patient data as accurately as possible. With many patients potentially sending aggregated or detailed data, the amount of data and their arrival rate may vary widely. Moreover, the number of patients currently connected to the system may also vary. To treat each patient in a

global emergency situation, e.g. an earthquake, classification for each individual observation must be performed quickly and be improved as long as time permits. A solution to these requirements is offered by algorithms that are based on the anytime paradigm.

A further challenging characteristic of medical data is the diversity of the data. Depending on the disease to be detected there are various possible measurements to be sensored. ECG, temperature and accelerometer data has already been included in a proof of concept [KKK+08]. However, there are far more possible non-invasive measures like respiration quotient, pulse oximetry, humidity, body posture, respiration frequency and many more complex medical measures. There can be also derived measures like features extracted out of the ECG signal [RGI05]. Especially the dependencies between some of these measures are important for an emergency detection, as respiration has a huge impact on ECG measurement artifacts, for example. The system must fulfill this requirement of multivariate classification by handling many different measurements.

Finally, since new patient data is constantly coming in and parts of these data, e.g. from supervised hospital situations, can be used as additional training data, the classifier should be able to incrementally learn from new training data. New training data can be learned while the patient is supervised and in the next moment the status can be switched back to classification.

In summary up, the overall classification task must fulfill three main requirements derived out of the health surveillance scenario:

- Anytime processing of varying stream rates
- Incremental learning based on multivariate data
- Multi-step processing for classification in a cascade of devices

The Bayes tree fulfills all of the above requirements. As an anytime classification approach it adapts to different classification times given for example by sensor measurement frequency. Furthermore, it can handle multivariate data provided by multiple different sensors. As the classifier includes an indexing structure for secondary storage it can handle huge amount of stream

data collected at medical facilities. Parts of this index structure can be used in multi-step approaches in a cascade of classifiers with increasing accuracy. Moreover, it also adapts to evolving data through an incremental learning of the underlying model.

8.1.1 Prototype

The achieved results on benchmark data (cf. Chapters 4 to 7) show the applicability of anytime classification, incremental learning and multi-step classification and suggest its use in future body sensor network scenarios. For a remote health monitoring system, this ensures scalability as huge amounts of information from multiple hospitals can be collected, stored and processed in data farms. Moreover, incorporating existing medical data bases through the presented bulk loading approaches improves the resulting anytime classification performance.

The anytime classification approach focuses on a multi-step patient health classification based on multivariate sensor data. As depicted in Figure 8.1 there are multiple steps from the body sensor network where sensor measurements are collected up to the health professional. Challenges for the overall system arise on multiple layers from the extraction of vital signals out of sensor measurements up to the health surveillance system for the health professional. A prototype addressing all of these layers showed the applicability of this concept. The developed prototype provides an overall mobile system with textile sensors as shown in Figure 8.3. The underlying sensor network is based on a MSP430 microcontroller (Texas Instruments, USA) which serves as a master controller for the deployed sensors. In the prototype three types of sensors are integrated: an ECG sensor, a temperature sensor and an accelerometer. The ECG sensor is used to monitor the electrical activity of the heart. The three-axis accelerometer is used to monitor the activity of the user and to detect and identify artefacts due to movement. For example, the ECG sensor is sensible to displacement of the electrodes. In this case the acceleration data can be correlated to distinguish movement artefacts from an irregular ECG due to disease related irregular patterns. As

Figure 8.3: Application of anytime classification in body sensor networks [KKK+08, MBZ+07, KBP+09, KCB+09].

textile sensors the shirt has three integrated electrodes for the monitoring of heart rate signals. The electrodes are made from silver coated polyamide yarn (Shieldex 110/34) which yields both highly conductive surfaces and elastic behavior for ease of use and wearing comfort. The garment must be close-fitting so that a permanent contact between the electrodes and the skin can be guaranteed both for direct and capacitive use [MBZ+07].

The master controller of this body sensor network is connected to the mobile device via a wireless communication channel. On the mobile device a J2ME application processes the sensor data stream and decides on the data to be forwarded to the back-end server. In depth analysis can then be done on the server and detailed data can be requested automatically or by inter-

action of a health professional. Thus, in addition to the pre-classification on the mobile device, further decision refinement about emergency situations is included on a central server with a more complex classifier. The integration of multiple classification steps into this process ensures a scalable emergency detection system. First, on a mobile device carried by the patient, which includes data aggregation based on the pre-classification. Second, classification on a central server in a health facility which collects data of various patients and performs a more accurate classification to refine the classification results of the mobile devices. Overall the system is controlled by a health professional taking final decisions about the patients situation.

8.2 Summary

In this chapter major challenges derived out of next generation mobile health surveillance were discussed. Semi-automated classification was highlighted for the emergency detection task and it was shown how novel index-based classifiers can build the core for multivariate multi-step classification in health surveillance. By supporting anytime learning and anytime classification the Bayes tree can handle huge amounts of data, which makes it a consistent solution for the described medical scenario. Moreover, as was laid out in this chapter, the Bayes tree fulfills all requirements which are crucial for classifying medical patient data in a scalable health surveillance.

Future challenges include extending the existing framework and evaluating the Bayes tree classifier based on sensor measurements in a broad health surveillance project. This project will include extensions of textile sensors, body sensors and preprocessing techniques as well as the integration and merging of sensor data in electronic health record systems. Emergency detection on multiple levels will show the benefits of multi-step classification and further enhance the scalability of emergency detection systems.

Chapter 9

Anytime Algorithms on Constant Streams

* Anytime algorithms have been proposed for many different applications, we discussed anytime algorithms for data mining in Chapters 3 and 4 - 7. Their strengths are the ability to first provide a result after a very short initialization and second to improve their result with additional time. Therefore, anytime algorithms are mostly used when the available processing time varies, as in the case of varying data streams. In this chapter the use of anytime algorithms on constant data streams is discussed, suggesting anytime algorithms for tasks with constant time allowance. Two approaches are introduced that harness the strengths of anytime algorithms on constant data streams and thereby improve the overall quality of the result with respect to the corresponding budget algorithm. A theoretic model for the expected performance gain is derived and the effectiveness of the proposed approaches is demonstrated using existing anytime algorithms on benchmark data sets.

*This chapter has been published in the Data Mining and Knowledge Discovery Journal, Special Issue on Best Papers from the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2009) [KS09a, KS09b].

9.1 Introduction

As discussed in Chapter 3 data streams can generally be divided into two groups, the first one being constant data streams, where the time between each two consecutive stream data items is equal (constant), and the second one being varying data streams, where the time between two items varies.

Items on a conveyor belt are an intuitive example for a constant stream. They are often evenly arranged before being sorted or before passing an automated quality check etc. We use this for illustration as a running example throughout the chapter (cf. Figure 9.1), keeping the abundance of applications from machine monitoring and sensor networks etc. in mind. Moreover, classification of stream data items is used as an example application. At point t_f features are taken for each item e.g. through cameras or sensors. At point t_d a classification decision must be made for the item passing to sort it into the correct bin.

In practice and in the literature it was so far not usual to employ an anytime algorithm on constant data streams, since there was no necessity to bother with additional implementation effort if the time budget is constant and known in advance. For any given time amount a budget algorithm can be developed that satisfies this allotted time. The goal which was set and achieved in the research presented here is to improve the quality of the result over that of the respective budget algorithm. In other words, for the abundance of constant data streams and mining applications the proposed methods achieve an improvement of the output quality.

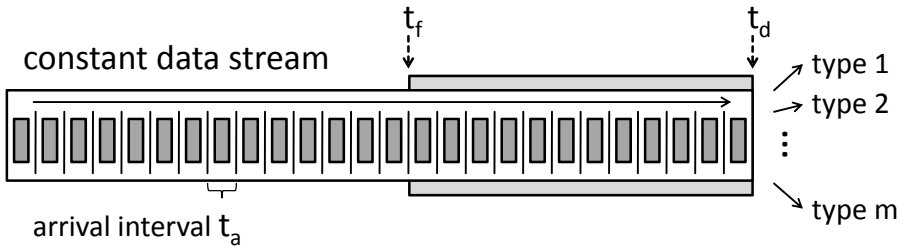


Figure 9.1: Items on a conveyor belt as a constant data stream. Features are extracted at t_f, a decision must be made at t_d.

The proposed approaches are not anytime algorithms by themselves. They constitute meta approaches that enable applications with constant data streams to benefit from the strengths of anytime algorithms. The basic idea underlying these meta approaches is to spend less time on data items that have a good result early on and use the acquired extra time on data items with poor results.[†] Both approaches are based on a quality measure for the current result of the anytime algorithm with respect to the current item. As stated above stream classification is used as a running example; the confidence of the current classification decision is used as a quality measure. The proposed approaches are described in Section 9.2 and evaluated in Section 9.3. Section 9.4 concludes the chapter.

9.2 Novel Approaches for Constant Data Streams

In this Section novel approaches are presented that allow for profiting from anytime algorithms on constant data streams. In Section 9.2.1 formal definitions are provided and the expected improvement is discussed using a theoretic model. Sections 9.2.2 and 9.2.3 introduce the batch and the fifo approach respectively.

9.2.1 Definitions and Models

Definition 9.1 *Let $\mathcal{S}$ be a constant data stream and $algo$ a stream mining algorithm working on $\mathcal{S}$. Then $t_f(o)$ is defined as the time when the features of an item $o \in \mathcal{S}$ become known to $algo$ and $t_d(o)$ is the time when $algo$ must provide a result for o. The* arrival time *between o_i and o_{i+1} is defined as t_a and is constant for all i.*

If $t_d - t_f \leq t_a$ a budget algorithm with a time budget of $t_d - t_f$ is the best choice. If $t_d - t_f > t_a$ the proposed approaches can improve the overall quality of the result. The running example assumes six items passing per

[†]The meta approaches presented in this chapter are applicable to both constant and varying streams. They are discussed and evaluated on constant streams to highlight the benefits over budget algorithms, which are commonly used in practice and literature.

second and two seconds between the camera (feature extraction at t_f) and the decision (classification at t_d). The proposed approaches can increase the number of correctly classified stream items (improve the overall classification accuracy). Besides the anytime algorithm they require a confidence measure that assesses the certainty of the current decision. This confidence measure must be provided by the user and is justified by its performance in an actual application.

Definition 9.2 *For a classifier $\mathcal{C}$ the confidence of its classification decision on an item o states how certain the decision is and ranges from 0 (no confidence) to 1 (certain):*

$$\mathit{conf}_{\mathcal{C}}(o) : o \rightarrow [0, 1]$$

A confidence measure is called *reliable* if the accuracy is monotonically increasing with the confidence: the higher the average confidence the better the accuracy of a classifier. In this section at first a linear dependency between confidence and accuracy is assumed, the influence of confidence measures is discussed at the end of the section.

To build a theoretical model, it is assumed that after initialization the average confidence of an anytime classifier's decision increases over time; it is the same confidence as that of the corresponding budget classifier with equal time allowance. Moreover, for any given time budget the confidence of the individual decisions for this budget are assumed to be scattered around the expected value. The amount of scattering is assumed to decrease over time. Figure 9.2 illustrates the assumptions (right). The time axis is normalized to $[0, 1]$; at $t = 1$ the anytime classifier has performed all possible improvements. The model assumes an anytime classifier that performs one initial step and n improvement steps that all take the same amount of time (the anytime nearest neighbor classifier [UXKL06] is an example, cf. Chapter 3).

Let $\zeta(x, t)$ denote the probability density function that describes the scattering of the confidences where t is the allowed time budget (figure 9.2 gives an example (left)). Then we can calculate the probability that the confidence

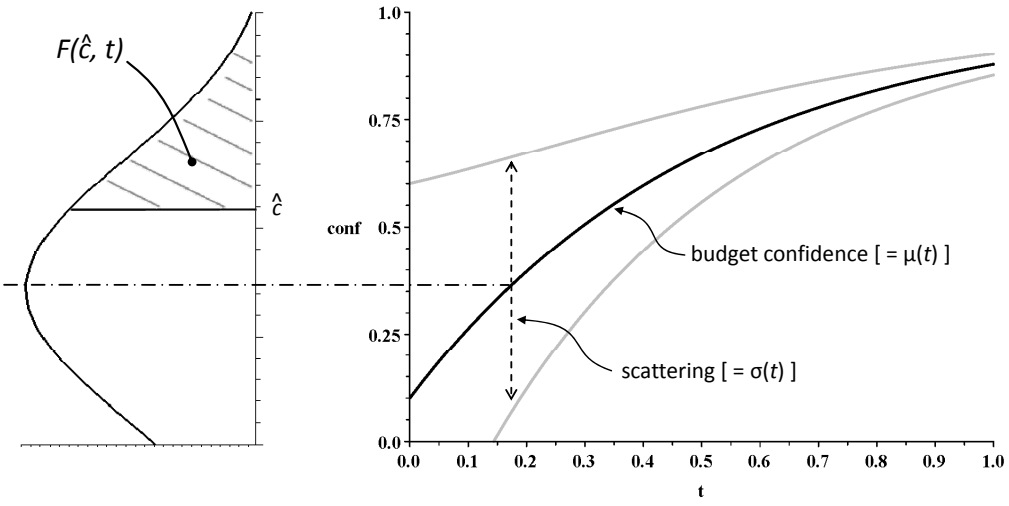


Figure 9.2: Expected confidence and scattering.

$c(t)$ at time t is larger than some confidence $\hat{c}$ by

$$F(\hat{c}, t) = p(c(t) \geq \hat{c}) = \int_{\hat{c}}^{\infty} \zeta(t, x)dx$$

Recall the n improvement steps $t_j, j = 1 \ldots n$, of the anytime classifier. If $F(c, t)$ is used as a cumulative distribution function, then the probability that we exceed $\hat{c}$ for the first time between t_{j-1} and t_j can be derived as $h(\hat{c}, t_j) = F(\hat{c}, t_j) - F(\hat{c}, t_{j-1})$. (Note that $F(\hat{c}, 0) + \sum_{v=1}^{n} h(\hat{c}, t_v) = 1$.) Now we can determine the expected time that is necessary to reach $\hat{c}$ by

$$t(\hat{c}) = F(\hat{c}, 0) \cdot \frac{1}{n} + \sum_{v=1}^{n} h(\hat{c}, t_v) \cdot \frac{v}{n} \tag{9.1}$$

As an example we assume the scattering to follow a Gaussian distribution $g(\mu, \sigma, x) = (1/(\sigma\sqrt{2\pi})) \cdot e^{(-0.5 \cdot (\frac{x-\mu}{\sigma})^2)}$ where the mean is the budget confidence following $\mu(t) = c_0 + (1 - c_0) \cdot (1 - e^{-\alpha \cdot t})$ and the variance (scattering) decreases by $\sigma(t) = \sigma_0 \cdot e^{-\beta \cdot t}$. $F(\hat{c}, t)$ is plotted in Figure 9.3 (left) for $\hat{c} = 0.3$. The gray line corresponds to the probability that we did not yet exceed $\hat{c}$. Figure 9.3 (right) shows the corresponding graph for $t(\hat{c})$ (*batch*) compared to the inverted *budget* confidence $\mu^{-1}(\hat{c})$. The inverse of equation 9.1 tells us

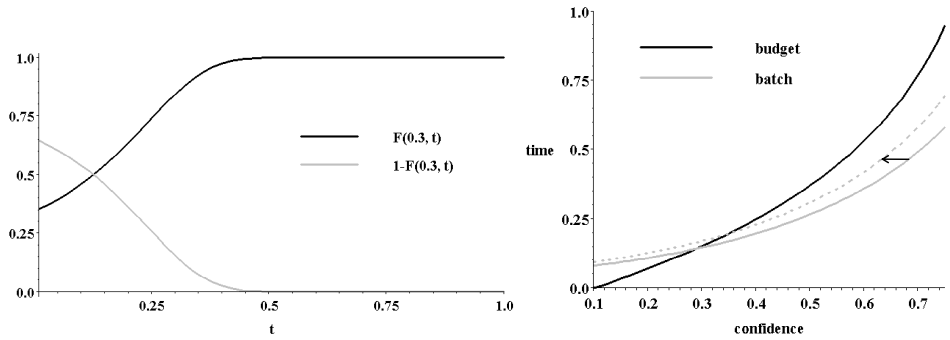


Figure 9.3: Left: $F(0.3, t)$ corresponds to the probability that we exceeded a confidence of 0.3 at time t. Right: The gray lines corresponds to the expected time we need to reach a given confidence when processing multiple objects at the same time.

which confidence can be achieved using an anytime classifier *in an average time*. More precisely, to be able to benefit from the above, several items must be processed simultaneously while striving to distribute the available time among them so as to optimize the resulting accuracy. This means fewer improvement steps should be spent on items that have a high confidence early on to use the acquired extra time to improve less certain decisions.

So far we assumed a reliable confidence measure, that is, a linear dependency between confidence and accuracy of a classifier. If the confidence is not fully reliable, then the average time needed to reach a certain accuracy will increase. This corresponds to a higher necessary confidence as is indicated by the dashed gray line in Figure 9.3 (right). Instead of elaborating this part for the theoretical model, the actual approaches are described in the next sections and evaluated in Section 9.3 using real algorithms and real confidence measures. Moreover, the confidence distributions resulting from the experiments are analyzed in comparison with the theoretical model in that section.

9.2.2 Batch Approach

As described in the last section several items must be processed simultaneously and the available time must be distributed among them to be able to

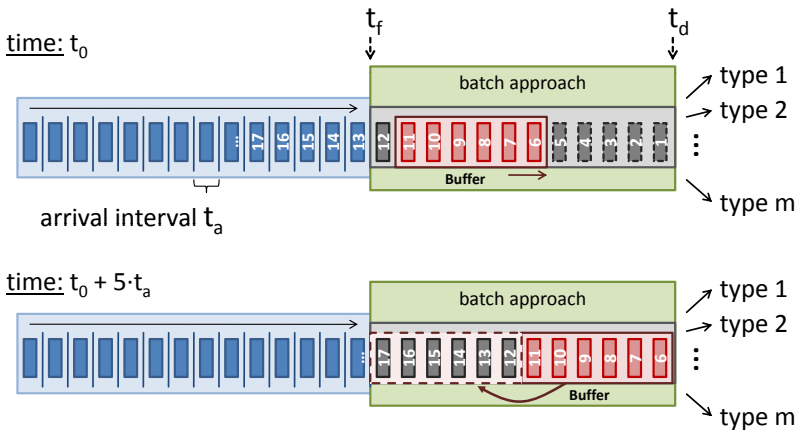


Figure 9.4: Illustration of the batch approach. When the classify buffer reaches t_d it is swapped (bottom).

improve the overall quality of the result. The first approach therefore uses an object buffer and processes the whole buffer until a decision must be made at t_d. This approach is referred to as the batch approach; it is illustrated in Figure 9.4.

The number of items between points t_f and t_d is $r = \lfloor (t_d - t_f)/t_a) \rfloor$, the capacity of the buffer is set to $\lfloor r/2 \rfloor$. In the running example $r = 12$, hence the buffer holds six items. At time t_0 (Figure 9.4, top) the buffer is filled ({6, 7, 8, 9, 10, 11}) and there is still $5 \cdot t_a$ time left to improve the classification on those six items. At time $t_0 + 5 \cdot t_a$ (Figure 9.4, bottom) the decision for item 6 is required and hence the buffer is cleared and the next six stream data items are added to it. The decisions for items 6 to 11 do not change thereafter and they are sorted into their corresponding bins when passing t_d.

The implementation of the batch approach uses two buffers, a *collectBuffer* to collect the next $\lfloor r/2 \rfloor$ items and a *classifyBuffer*, which is used to distribute the processing time among the included items according to their confidence. Figure 9.5 provides a pseudocode description of the batch approach. When a new item $\mathbf{s}_{new}$ arrives, it is initialized and stored in the *collectBuffer* (lines 5-7). Initialization is done using the actual anytime algorithm, hence a first result and a first confidence is available for $\mathbf{s}_{new}$ when it is stored in the buffer. When $\lfloor r/2 \rfloor$ items have been collected, the buffers

```
01 // Input: confidence measure conf, constant data stream S
03 collectBuffer = ∅, classifyBuffer = ∅;
03 capacity = floor( (t_d-t_f)/(2*t_a) );
04 while (true) {
05    if (new item s_new arrived) {
06       init(s_new);
07       collectBuffer.add(s_new);
08       if (collectBuffer.size() == capacity) {
09          classifyBuffer = collectBuffer;
10          collectBuffer = ∅;
11       }
12    } else {
13       minConf = ∞;
14       itemToImprove = null;
15       for (Item s : classifyBuffer) {
16          if (conf(s) < minConf) {
17             minConf = conf(s);
18             itemToImprove = s;
19          }
20       }
21       improve(itemToImprove);
22    }
23 }
```

Figure 9.5: Pseudocode description of the batch approach.

are swapped and the *collectBuffer* is cleared (lines 8-10). Hence, the *classifyBuffer* then contains $\lfloor r/2 \rfloor$ initialized items. If no new item has arrived, one item out of the *classifyBuffer* is improved, i.e. processed further using the anytime algorithm. For this purpose the object with the lowest confidence is chosen according to the given confidence measure *conf* (lines 12-21). After improving the item, its assigned class label may have changed as well as the confidence of the decision.

For classification on constant data streams using the batch approach the best would be to always refine that item in the buffer first, which requires the least number of improvements until it is classified correctly. Since this number is unknown, no confidence measure can be derived from it. In Section 9.3 details are provided about the actual confidence measures that are used and evaluated in the experiments. In the next section the second approach is described, which incorporates an additional time function for the priority decision.

9.2.3 FiFo Approach

In the batch approach the *classifyBuffer*'s capacity was set to $\lfloor r/2 \rfloor$ to enable a meaningful swapping; the second approach uses a FiFo queue that holds all r elements between t_f and t_d. As a consequence each item in the queue has an individual amount of time remaining for further improvement steps. In the batch approach each item contained in the *classifyBuffer* had the same remaining time due to the swapping of the two buffers. The FiFo approach uses the FiFo queue to store the items and to keep track of the order of arrival. The decision about which item in the queue is chosen to be improved is done separately by assigning priorities to the items as explained below.

The individual remaining time can be incorporated in determining the item to be improved next. Still, the confidence of the current result for the respective items is an important factor for the choice. If all items in the queue have the same confidence, then priority should be given to the item with the lowest remaining time. Even if the oldest item has a slightly higher confidence than the rest, we still prefer improving that item, because it is the next to be removed from the queue and we may therefore lose the opportunity of improving it once more. Hence, a weighting function is required that decreases with the remaining time:

$$\omega(t_r) : \mathbb{R}^+ \rightarrow \mathbb{R}^+ : t_r \rightarrow [\omega_{min}, 1] \tag{9.2}$$

with

$$t_1 < t_2 \;\Rightarrow\; \omega(t_1) \leq \omega(t_2).$$

In the above equation t_r is the remaining time and ω_{min} is the minimal weight. Using a weight function as in Equation 9.2 can artificially decrease the confidence for items with a small amount of remaining time by simply multiplying the confidence conf($\mathbf{s}$) of each item $\mathbf{s}$ with its weight $\omega(t_r(\mathbf{s}))$ according to the remaining time:

$$\text{conf}(\mathbf{s}) \cdot \omega(t_r(\mathbf{s}))$$

```
01 // Input: confidence measure conf, constant data stream S,
02 //        time weighting function weight
03 fifoQueue = ∅;
04 capacity = floor( (t_d−t_f)/t_a );
05 while (true) {
06    if (new item s_new arrived) {
07       init(s_new);
08       fifoQueue.put(s_new);
09       if (fifoQueue.size() > capacity)
10          fifoQueue.get();// remove oldest item from FiFo
11    } else {
12       minConf = ∞;
13       itemToImprove = null;
14       for (Item s : fifoQueue) {
15          tempConf = conf(s) * weight(remainingTime(s));
16          if (tempConf < minConf) {
17             minConf = tempConf;
18             itemToImprove = s;
19          }
20       }
21       improve(itemToImprove);
22    }
23 }
```

Figure 9.6: Pseudocode description of the FiFo approach.

Two common weighting functions are evaluated in the experiments:

$$\text{linear:} \quad \omega(t_r) = \omega_{min} + (1 - \omega_{min}) \cdot \frac{t_r}{(t_d - t_f)} \tag{9.3}$$

$$\text{exponential:} \quad \omega(t_r) = \omega_{min} + (1 - \omega_{min}) \cdot \frac{1 - e^{-\lambda \cdot t_r}}{1 - e^{-\lambda \cdot (t_d - t_f)}} \tag{9.4}$$

Figure 9.6 shows a pseudocode description of the fifo approach. Newly arrived items are initialized and added to the queue (lines 6-8). If the FiFo capacity is exceeded, the oldest item is removed from the queue (lines 9-10). If no new item arrived, an item to improve is chosen from the queue according to minimal time weighted confidence as described above (lines 14-20). As in the batch approach the chosen item is improved (line 21) possibly affecting its assigned class label and confidence.

9.3 Experiments

As introduced above, anytime classification is used as an example application. The employed algorithms and corresponding confidence measures are introduced in the next section. In Section 9.3.2 the effectiveness of the proposed approaches is demonstrated and their performance is compared. Section 9.3.3 provides an investigation of the confidence distributions in comparison with the theoretical model. A short discussion concludes this section.

9.3.1 Algorithms and Confidence Measures

The proposed meta approaches are evaluated using anytime versions of the nearest neighbor classifier, Bayes classifier and support vector machines (cf. Chapter 3). Simple confidence measures are employed here; more sophisticated measures can be developed but are not the focus of this research. Recent approaches to classification confidence are discussed in Section 9.3.4.

For the anytime nearest neighbor (knn for short), with $nn_i^{\mathbf{s}}$ being the nearest neighbors of the current object $\mathbf{s}$ and $d(\mathbf{s}, nn_i^{\mathbf{s}})$ the Euclidean distance between $\mathbf{s}$ and its i-th nearest neighbor, the following confidence measure assigns a higher confidence if the involved neighbors are closer to the object:

$$\text{conf}_{knn}(\mathbf{s}) = e^{-\sum_{i=1}^{k} d(\mathbf{s}, nn_i^{\mathbf{s}})}$$

To determine a confidence for the current decision of the anytime SVM, the difference between the highest output value $h_{j_1}(o)$ and the second highest output value $h_{j_2}(o)$ is computed to express the certainty. conf_{svm} is calculate as

$$\text{conf}_{svm}(o) = 1 - e^{-(h_{j_1}(o) - h_{j_2}(o))}.$$

Similar to the SVM the confidence of the current decision for the Bayes tree is calculated as the difference between the highest probability $P(C_{i_1}|o)$ and the second highest probability $P(C_{i_2}|o)$:

$$\text{conf}_{bt}(o) = P(C_{i_1}|o) - P(C_{i_2}|o)$$

These simple choices are justified empirically in the following.

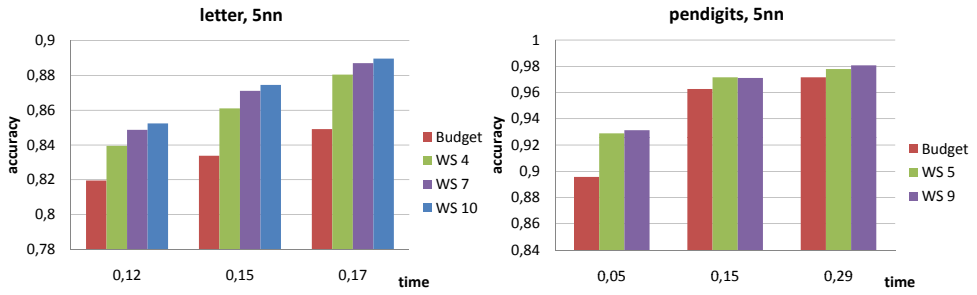


Figure 9.7: Evaluating the batch approach on different window sizes (WS) using anytime 5nn on the *letter* (left) and *pendigits* (right) data set.

9.3.2 Evaluation

As in section 9.2.1 the time is normalized to $[0, 1]$, where 0 represents the result just after initialization (i.e. zero improvements done) and 1 indicates that all information has been processed (i.e. no more improvements possible). All experiments use a 4-fold cross validation an benchmark data sets from [HB99] to evaluate the classification accuracy.

The first experiment evaluates the batch approach starting off with the anytime nearest neighbor with $k = 5$ (5nn). Figure 9.7 (left) shows the results on the *letter* data set (20000 object, 16 dimensions, 26 classes) for various window sizes. In all experiments, the corresponding budget classifier is trained for the respective time allowance such that each item is allowed the same processing time dictated by the arrival rate. The improvement of the budget classifier's accuracy when more time is given (slower stream) shows its effectiveness with respect to the allotted time.

For the same time allowance (one group of bars) the accuracy increases with growing window size, which proves the effectiveness of the proposed batch approach in this experiment. Similar results can be seen for 5nn on the *pendigits* data set (10992 objects, 16 dimensions, 10 classes) in Figure 9.7 (right). The relative accuracy increase due to the batch approach is the largest for small window sizes, i.e. the improvement from Budget to WS 4 is higher than the improvement from WS 4 to WS 7 on *letter* (compare Budget to WS 5 and WS 5 to WS 9 on *pendigits* respectively). On *pendigits* the accuracy for WS 9 is even slightly less than for WS 5 for $t = 0.15$ (i.e. when 15% of the model (training set) have been evaluated).

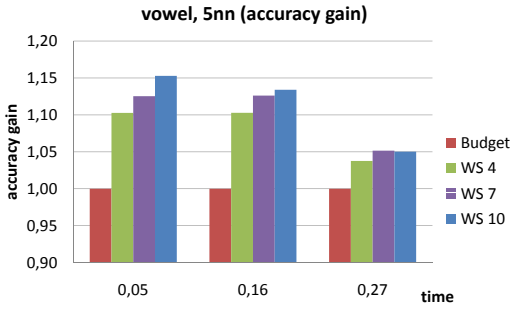


Figure 9.8: The relative accuracy gain w.r.t. the budget classifier for 5nn on *vowel* is up to 15%.

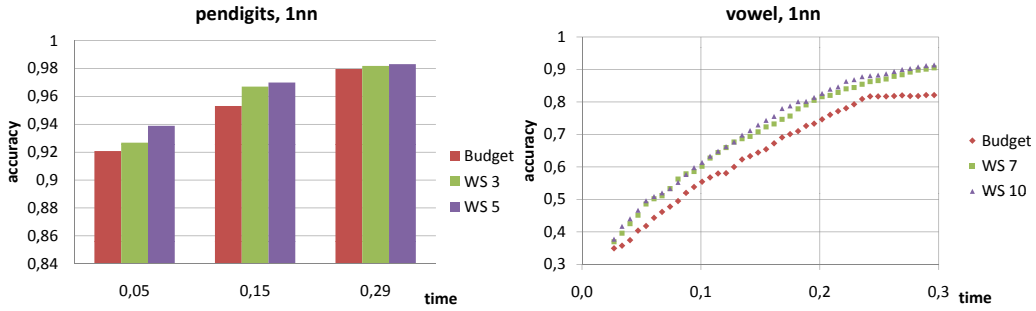


Figure 9.9: Performance of the batch approach.

Figure 9.8 shows the relative accuracy gain w.r.t. Budget for 5nn on the *vowel* data set (990 objects, 10 dimensions, 11 classes). Once more the accuracy increases with growing window sizes. Moreover, the accuracy gain with respect to the budget classifier is higher for faster streams (i.e. less time between objects): The accuracy increases by 15% using WS 10 at $t = 0.05$ while the increase is only 5% for the same window size at $t = 0.27$.

Setting $k = 1$ improves the result for the anytime nearest neighbor on *pendigits* (cf. Figure 9.9 left). Still the batch approach shows its effectiveness as the accuracy increases with the window size. Figure 9.9 (right) shows the performance of 1nn on *vowel* for a wide range of time intervals. The improvement from Budget to WS 7 is significant throughout while the improvement from WS 7 to WS 10 is rather small. This shows that the batch approach is very effective already for small window sizes, which implies that there are fewer objects that need to be 'helped out' by receiving additional processing than confident objects 'willing' to spare processing time.

Figure 9.10 shows the results for the SVM and Bayes tree classifiers on

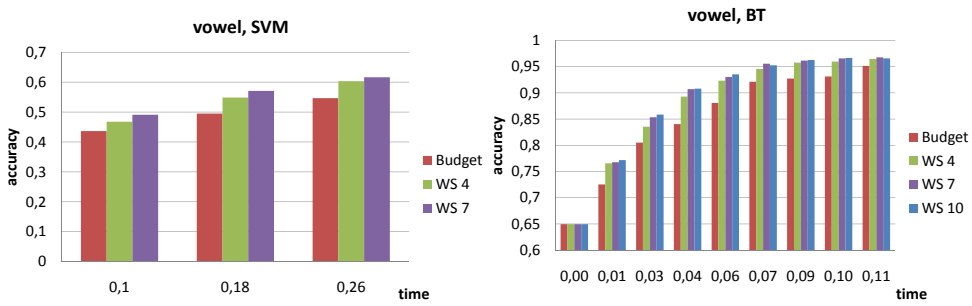


Figure 9.10: Testing the batch approach using anytime SVM (left) and Bayes tree (right) on *vowel*.

vowel. For all stream speeds both classifiers improve their accuracy with a larger window size. This proves the effectiveness of the batch approach also for different anytime algorithms. For the rest of the evaluations the *vowel* data set is used unless mentioned otherwise.

To relate to the theoretic assumptions made in Section 9.2.1 the dependency between confidence and accuracy for the three anytime classifiers is analyzed (additional analysis is provided in Section 9.3.3). Figure 9.11 (left) shows the accuracy over confidence plots for SVM, BT and 5nn respectively. Each point represents the accuracy and the corresponding average confidence for one experiment (one time interval tested using 4-fold cross validation as above). All three classifiers show a monotonic increase of accuracy over the respective confidence. Interestingly, for all tested classifiers and confidence measures the behavior is close to a linear dependency. Similar observations resulted from experiments on other data sets.

Completing the evaluation of the batch approach the expected time to reach a given confidence is reported in Figure 9.11 (right). These plots directly relate to Figure 9.3 from Section 9.2.1 (cf. page 164). The results show that, using the batch approach, the expected time decreases for all three anytime classifiers and for any given confidence. As above it can be seen that the largest impact appears for small window sizes (e.g. Budget to WS 4 for BT in Figure 9.11, middle). Due to the relationship between accuracy and confidence discussed above, similar plots and results can be obtained for the expected time to reach a given accuracy.

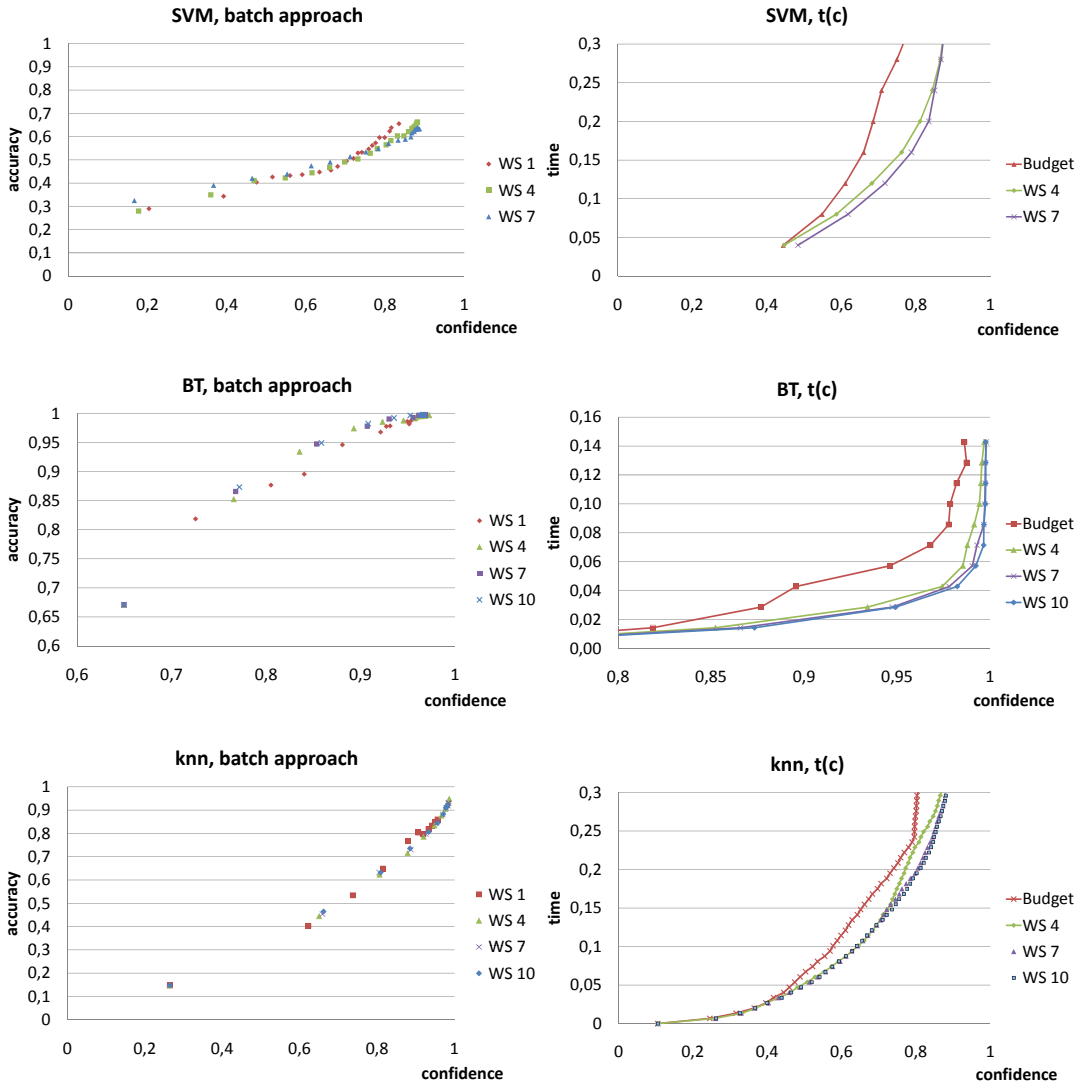


Figure 9.11: Left: Relation between the average confidence and the observed accuracy with the batch approach. Right: The time needed to reach a given confidence with the batch approach. Results are shown for all three classifiers tested on the *vowel* data set.

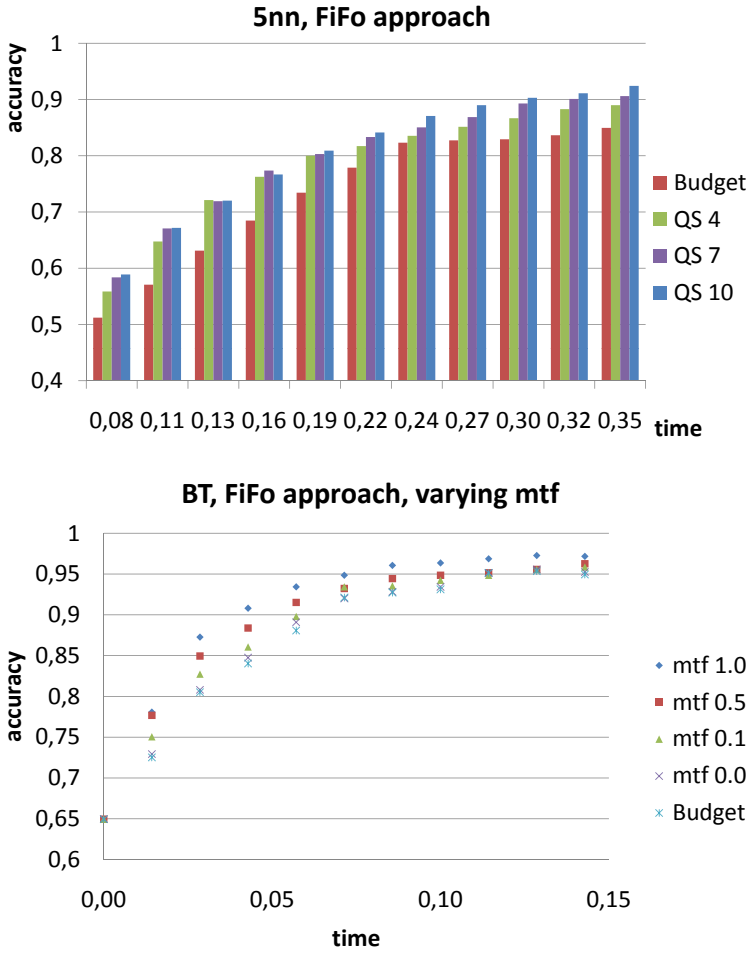


Figure 9.12: Top: Evaluating different queue sizes for the FiFo approach using the anytime nearest neighbor with $k = 5$. Bottom: Evaluating different time weighting functions for queue size 13 using the Bayes tree on *vowel*.

Next the FiFo approach is evaluated starting off again with the anytime nearest neighbor for $k = 5$. A linear time weighting function is used with $\omega_{min} = 0.5$ (cf. Section 9.2.3). Figure 9.12 (top) shows the results for four different queue sizes from faster streams ($t_a = 0.08$) to slower streams ($t_a = 0.35$). As in the batch approach the performance of the respective budget classifier is shown for comparison. The FiFo approach proves its effectiveness similar to the batch approach: with larger queue sizes the classification accuracy improves for all stream speeds. Similar results hold for the other

classifiers, details are omitted. The influence of the time weighting function is analyzed instead.

Figure 9.12 (bottom) shows the performance of the Bayes tree using the FiFo approach with varying time weighting functions. As a reference the results for budget classifier are given as well. The other results have been obtained using queue size 13. The employed time weighting functions are linear and the minimal value *mtf* (minimal time factor) varies from 0 to 1 (cf. ω_{min} in Section 9.2.3). mtf $= 1$ corresponds to a constant factor of 1, i.e. as if no time weighting would be applied.

All FiFo variants outperform the budget classifier. The linear time weighting function with $\omega_{min} =$ mtf $= 0$ does not yield the same results as the budget classifier, because there is still more than one improvement possible between two items (except for $t = 0$). Surprisingly the results improve constantly with growing mtf for all stream speeds. More precisely, a constant time weighting function, where time has *no* impact on the weighting, yields the best results. The same observation resulted for the exponential time weighting function, where $\omega_{min} = 1$ also corresponds to a constant function (cf. Equation 9.4). This indicates that simply using the confidence value is sufficient for distributing the available processing time even if items have individual remaining times. Therefore only the confidence value is used in the following when comparing the two approaches.

Figure 9.13 shows a comparison between the FiFo approach and the batch approach for anytime SVM and anytime nearest neighbor respectively. Both approaches outperform the budget classifier throughout. In the comparison the queue sizes are twice as big as the window sizes since the batch approach can only process half of the objects between t_f and t_d due to window swapping (cf. Section 9.2.2). For both classifiers the FiFo approach outperforms the batch approach slightly throughout different window sizes and different stream speeds. This seems obvious since the FiFo approach has more ”room” (objects) for optimization. Still, the batch approach is only slightly behind, showing that the approach is effective even for very small settings in terms of $t_d - t_f$.

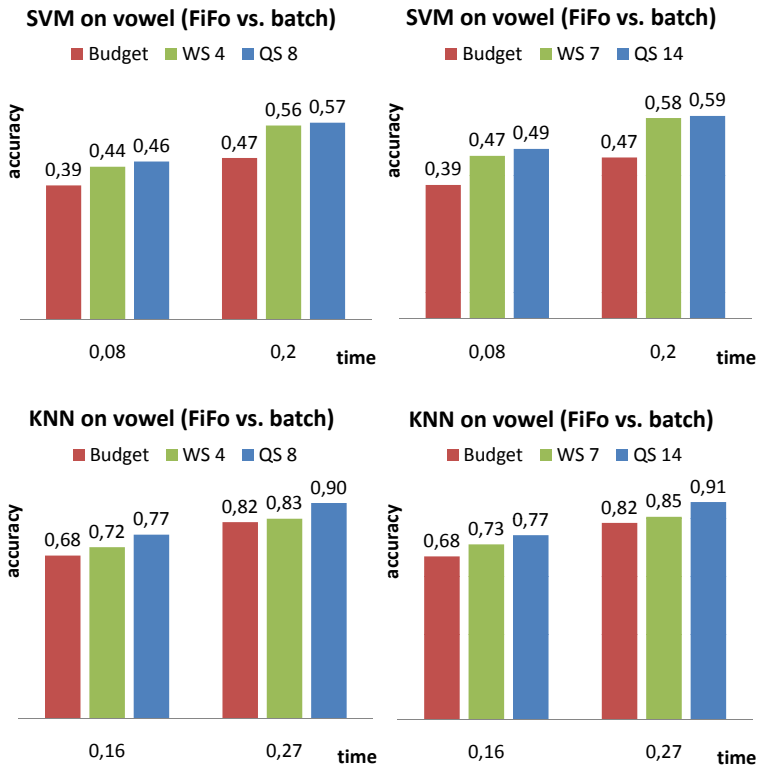


Figure 9.13: Evaluating FiFo versus batch approach using the anytime SVM (left) and 5nn (right) classifier. Both outperform the budget classifier.

9.3.3 Confidence distributions

One hypothesis in the theoretical model (cf. Section 9.2.1) is that the budget confidence (the average confidence over all classified objects) increases with increasing time allowance. The experimental results presented in the previous section confirmed this assumption for all three classifiers tested (cf. Figure 9.11). In the example of Section 9.2.1 we assumed that for a given time allowance the individual confidences follow a Gaussian distribution about their mean value. The following examines the confidence distributions obtained from the experimental evaluation.

Figure 9.14 shows the confidence distributions for the anytime nearest neighbor on three data sets (*letter*, *pendigits* and *vowel* from top to bottom), once in 3d-view (left) and the same plot seen from above (right). As can be seen on the very left edge of the 3d-plots, initially the confidences roughly

follow a normal distribution centered at 0.5 for *pendigits* and *vowel* and 0.7 for *letter*. While the mean increases over time, the scattering, i.e. the variance, decreases, even though only slightly. An explanation lies in the employed confidence measure: it increases with increasing proximity of the current nearest neighbors, which is naturally the case over time. These plots strongly resemble Figure 9.2 from the theoretical model confirming the assumptions made.

For the Bayes tree the confidence plots are shown in Figure 9.15 (top and center) for the *Forest Covertype* (left) and *Gender* data set (right). The bottom row shows the confidences of the anytime nearest neighbor on these data sets. The y-axis (count) is cut off to enhance visibility of the described aspects. The nearest neighbor plots show a similar behavior as described above. The confidences are higher on these two data sets, since their large size yields denser data distributions and hence smaller distances. The confidence distributions for the Bayes tree do not resemble Gaussian distributions per time allowance. The initial confidences seem to be rather equally distributed at lower confidences while a large fraction of the objects has a very high confidence early on. With increasing time fewer items exhibit a medium confidence while the large fraction of high confidences remains and a second accumulation grows at very low confidences. Similar results were obtained on other data sets. The reason lies once again in the employed confidence measure. For the Bayes tree the difference between the highest and the second highest probability is calculated in the exponent of the confidence measure (cf. Section 9.3.1). While for most objects the probability for a single class soon excels those of the remaining classes, a fraction of the objects lies at a boundary between two classes yielding similar probabilities the more the mixture models are refined over time. Nonetheless, the average confidence as well as the average accuracy of the Bayes tree increase over time.

The theoretical model is defined for arbitrary probability density functions (pdf) describing the confidence distribution. The results show that the actual pdfs for given confidence measures can be largely different. Nonetheless, the proposed meta approaches improved the accuracy in all tested scenarios.

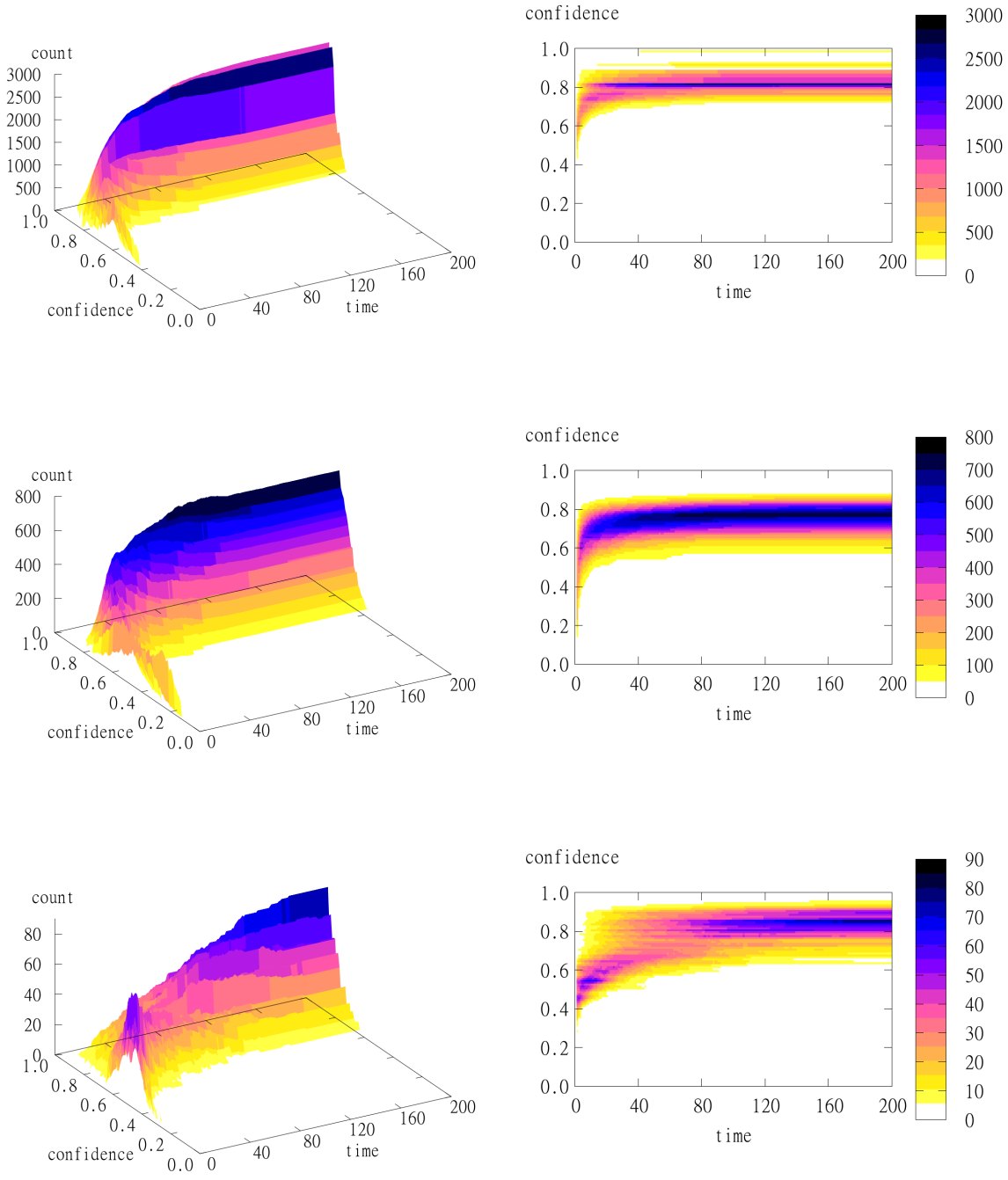


Figure 9.14: Confidence distributions for the anytime nearest neighbor on *letter* (upper row), *pendigits* (middle row) and *vowel* (bottom row). Left: 3d-view, Right: birds eye-view.

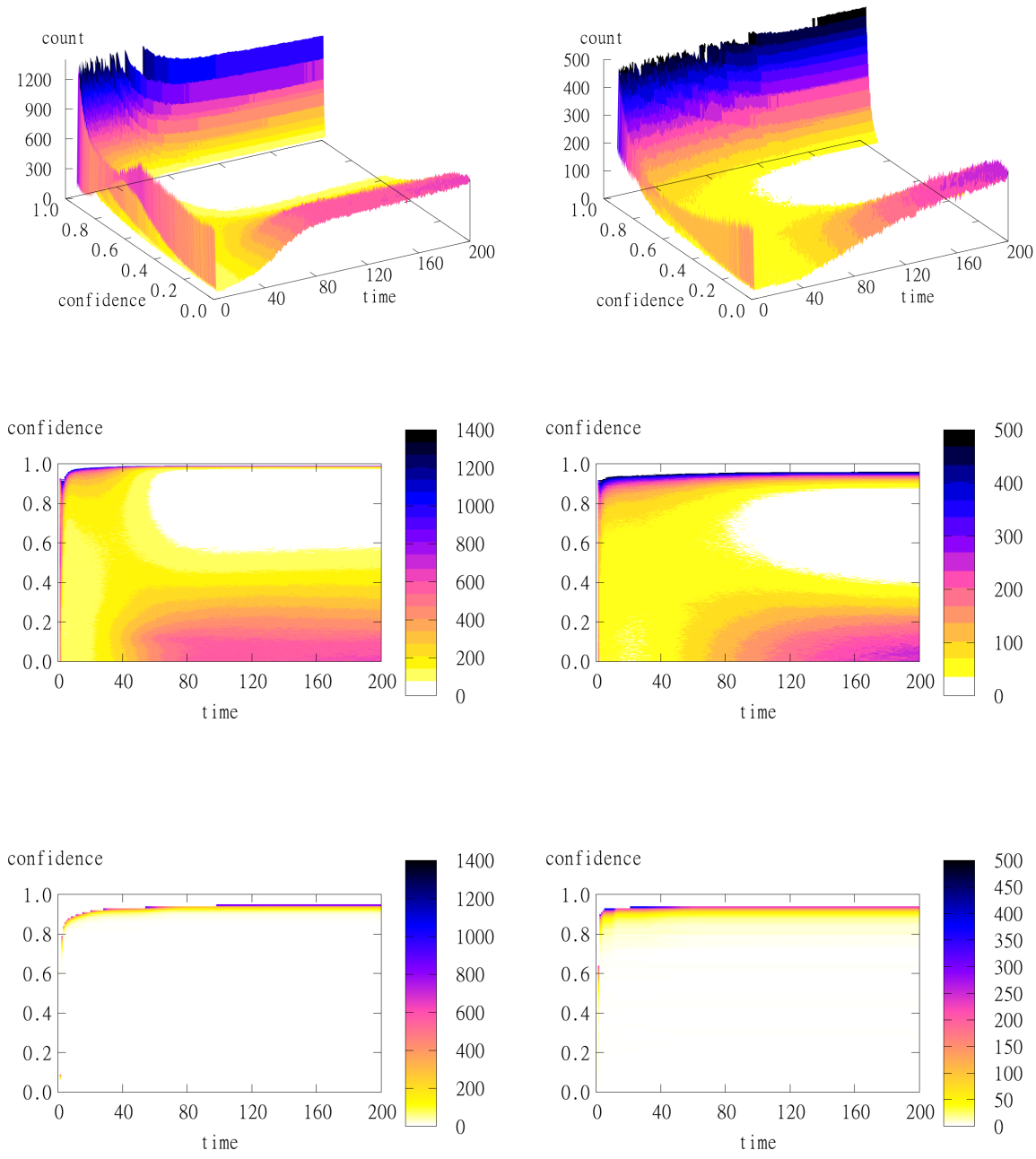


Figure 9.15: Confidence distributions on *Forest Covertype* (left) and *Gender* (right). The upper two rows show the resulting confidences for the Bayes tree (3d and birds eye), the bottom row shows the corresponding confidences of the anytime nearest neighbor (birds eye).

9.3.4 Discussion

We have seen that both the batch and the FiFo approach improve the classification accuracy of three prominent classification approaches (Bayes, SVM, nearest neighbor) on constant data streams even for small window or queue sizes. The small size implies a very small overhead in terms of space and time complexity, as can directly be seen from the provided pseudocodes. Moreover, the results presented in Section 9.3.2 prove that the small time overhead does not negatively affect the output quality, but improves the quality instead through the better distribution of available time.

While simple confidence measures were employed in the experiments, one may develop more sophisticated ones tailored to individual applications or classifiers. There are several approaches to classification confidence in the literature that either focus on specific classifiers such as neural networks [Wan90] or case based reasoning [Che00], or proposed solutions for a specific application like spam filtering [DCDZ05] or natural language processing [DCP08]. Along with the proposed approaches for classification on constant data streams this is an interesting area for further research.

9.4 Conclusion

In this chapter two novel approaches have been proposed that harness the strengths of anytime algorithms for constant data streams. The goal was to improve the quality of the result w.r.t. traditional budget approaches, which are used in an abundance of stream mining applications. Using anytime classification as an example application experimental results have shown for SVM, Bayes and nearest neighbor classifiers that both approaches improve the classification accuracy for slow and fast data streams. The results confirm the general theoretic model and show the effectiveness of the proposed approaches. The simple yet effective idea can be employed for any anytime algorithm along with a quality measure and motivates further research in classification confidence measures and anytime algorithms.

Chapter 10

Future Work

Many aspects of the algorithmic solution to anytime Bayesian classification on continuous attributes have been investigated and evaluated. The final version of the Bayes tree as it results from Chapter 7 shows very good results on various domains. An interesting future research objective is the combination of the Bayes tree with subspace methods. In a subspace Bayes tree the entries would store probability density functions that consider only a subset of the attributes. While a global dimensionality reduction can be achieved by preprocessing the data, for example using principle component analysis (PCA) or linear discriminant analysis (LDA), individual reductions per node can be achieved by applying subspace clustering in a top down manner for tree construction. Another interesting option for a subspace Bayes tree results from the concept transfer of Bayesian networks as used in Chapter 7: using an empty initialization for both the network structure and the covariance matrix before the hill climbing search can yield individual subspaces per node.

Further research questions regarding the Bayes tree include the investigation of different kernels such as the Epanechnikov kernel, and further construction methods such as those in X-trees or Ball-trees. The properties of created trees can be analyzed using statistical methods such as χ^2 tests. Characteristic values can be developed and evaluated that predict the expected performance based on the measured properties. Finally, to enable a consistent use on data sets with mixed attribute types the Bayes tree can be

combined with existing anytime classifiers for categorical attributes such as SPODEs [YWKT07] (cf. Chapter 3.2).

The benefits of the meta-approaches proposed in Chapter 9 can be investigated for other anytime algorithms and tasks. An example is provided in Chapter 14 for anytime outlier detection. The development of confidence measures for classifiers and other algorithms employed using the proposed meta-approaches is a promising research direction.

Finally, anytime classifiers can be further developed to serve as a static classifier. While this may sound absurd at first it offers interesting opportunities, since it can yield faster classification. Similar to their application on constant streams, a prerequisite is a confidence measure for the current decision. The classification process could then be terminated based on the current confidence value or the variance (or slope) of the most recent confidences. Further options include a fix number of improvement steps for the anytime algorithm, e.g. fix number of kernels or Gaussians for the Bayes tree. For the Bayes tree another possibility to terminate the refinement individually based on the current object is to determine lower and upper bounds for the densities per subtree and stop the descent if it cannot affect the decision anymore.

Part III

Anytime Stream Clustering

Chapter 11

Self-adaptive Anytime Stream Clustering

* Clustering data streams with varying inter-arrival times can profit from anytime algorithms. For clustering, this means that the algorithm is capable of processing even very fast streams, but also uses greater time allowances to refine the clustering model. In this chapter a parameter free algorithm is proposed that automatically adapts to the speed of the data stream. It makes best use of the time available under the current constraints to provide a clustering of the objects seen up to that point. The proposed approach incorporates the age of the objects to reflect the greater importance of more recent data. For efficient and effective handling the ClusTree is introduced as a compact and self-adaptive index structure for maintaining stream summaries. The ClusTree makes no a priori assumption on the size of the clustering model, but dynamically *self-adapts*. It is shown that the algorithm can be combined with existing techniques for aging objects in the stream using decay functions, reporting cluster snapshots at different points in time, and comparing views at different points in time (cf. Chapter 3).

*This chapter has been published in the Proceedings of the 9th IEEE International Conference on Data Mining (ICDM 2009) [KABS09].

11.1 The ClusTree Algorithm

The proposed self-adaptive anytime stream clustering relies on an index structure for storing and maintaining a compact view of the current clustering. The size of the clustering model automatically adapts to the stream speed, which cannot be achieved by any buffering outside the storage structure. Moreover, to preserve a complete model no data object is dropped; any object from the stream is inserted into the index and possibly merged with aggregates of previously inserted objects. The following describes strategies for dealing with varying time constraints for anytime clustering, i.e. how the process of inserting an object is interruptible at any time.

11.1.1 Micro-clusters and anytime insert

The proposed approach is based on micro-clusters [ZRL96] as compact representations of the data distribution. By maintaining measures for incremental computation of mean and variance of micro-clusters, the infeasible access to all past stream objects is not necessary (cf. Chapter 3). Existing micro-cluster approaches lack support for varying stream inter-arrival times. It is, however, crucial to provide the means for anytime clustering and self-adaptation to stream speed. The proposed solution maintains cluster features (CFs) by extending index structures from the R-tree family [Gut84, BKSS90, SAK+09]. Such hierarchical indexing structures provide the means for efficiently locating the right place to insert any object from the stream into a micro-cluster. The idea is to build a hierarchy of micro-clusters at different levels of granularity. Given enough time, the algorithm descends the hierarchy in the index to reach the leaf entry that contains the micro-cluster that is most similar to the current object. If this micro-cluster is similar enough, it is updated incrementally by the object's values. Otherwise, a new micro-cluster may be formed.

The important observation for anytime clustering of streaming data is, that there may not always be enough time to reach the leaf level to insert the object. Therefore novel strategies for anytime inserts are presented. There

are several possibilities for handling object arrivals before the current object insert reaches leaf level. The straightforward solution keeps a **global queue**. This approach is very simple, but it may require an infinite buffer. Also, we may never have the time to empty the queue, resulting in outdated clustering results. To reduce memory consumption, one could maintain a **global aggregate**, i.e. instead of the queue a single cluster feature. However, aggregating arbitrary objects loses too much information as they may be diverse.

To maintain the necessary information for clustering, and to ensure that any newly arriving object can be inserted at once, the proposed solution interrupts the insertion process. The object must be temporarily stored in a **local aggregate** from which we can continue at a later time. This yields foreseeable space demands like with a global aggregate, albeit slightly larger ones. For the invested space we obtain a greater accuracy. The great advantage of local aggregates over local queues is that we can easily use the time for regular inserts to take a buffered local aggregate along as a "hitchhiker". Moreover, they can be naturally integrated into the tree structure. We will discuss this in more detail shortly, after describing the structure of the ClusTree hierarchical index for maintaining the micro-cluster information.

11.1.2 The ClusTree data structure

The ClusTree approach consists of a hierarchy of entries that describe the cluster feature properties of their respective subtrees. The structure of an inner entry and a leaf entry is illustrated in the left part of Figure 11.1: Each entry contains a cluster feature of the number of objects n that were aggregated, their dimension-wise linear sum LS, and squared sum SS, as well as a pointer to the respective subtree. The ClusTree integrates local aggregates into the tree structure for temporary objects. More precisely, an inner entry provides an additional buffer b for temporary insertions of local aggregates (CFs). Leaf nodes' entries do not contain a buffer, since inserts at leaf level are final.

Definition 11.1 ***ClusTree.*** *A ClusTree with fanout parameters m, M and leaf node capacity parameters l, L is a balanced multi-dimensional indexing struc-*

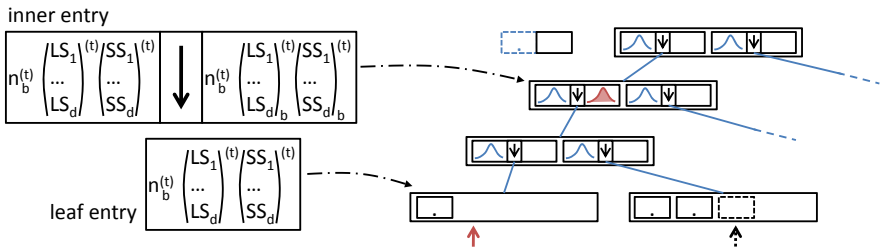


Figure 11.1: Left: inner node and leaf node structure. Right: insertion object, hitchhiker and buffer.

ture with the following properties:

- *an inner node $node_s$ contains between m and M entries. Leaf nodes contain between l and L entries. The root has at least one entry.*
- *an entry in an inner node of a ClusTree stores:*
  - *a cluster feature of the objects it summarizes.*
  - *a cluster feature of the objects in the buffer. (May be empty.)*
  - *a pointer to its child node.*
- *an entry in a leaf of a ClusTree stores a cluster feature of the object(s) it represents.*
- *a path from the root to any leaf node has always the same length (balanced).*

The tree is created and updated like any multidimensional index such as R-tree, R*-tree, etc. [Gut84, BKSS90, SAK+09]. Unlike the minimum bounding rectangles that they maintain in addition to the objects, the ClusTree stores only CFs. During insertion an object descends into the subtree with the closest mean with respect to Euclidean distance. Splitting is based on pairwise distances between the entries, where entries are combined into two groups such that the sum of the intra-group distances is minimal. We will see in Section 11.2 that $M = 3$ is a good choice, hence there are maximally six pairwise distances per node which yields a fast split operation.

The important property that reflects anytime capability of the ClusTree is its buffer in each entry. It serves as a temporary storage place for aggregates or objects that do not reach leaf level during insertion. Whenever insertion is interrupted, the current CF is simply stored in the buffer of the entry that corresponds to the subtree into which to descend next. At any future time when this subtree is next accessed, the temporary entry in the buffer is taken along as a "hitchhiker". This makes sure that future descent down the same subtree is used for continuing the insertion process. Whenever the descent destination of the current insertion CF and the hitchhiker differ, the latter is placed in the corresponding buffer again to wait for the next ride down the tree.

The right part of Figure 11.1 illustrates this process. Assume that the insertion object (drawn blue in the dashed box to the left of the root) belongs to the leaf that is marked by the dashed arrow (at the second leaf). Assume also, that the leftmost entry on the second level has a filled buffer (second distribution symbol in the entry), which belongs to a different leaf than the insertion object (indicated by the red solid arrow at the first leaf). The insertion object first descends to the second level, and next descends into the left entry. It picks up the left entry's buffer in its buffer CF for hitchhikers (depicted as the solid box at the right of the insertion object). The insertion object descends to level three, taking the hitchhiker along. Because the hitchhiker and the insertion object belong to different subtrees, the hitchhiker is stored in the buffer of the left entry on the third level (to be taken along further down in the future) and the insertion object descends into the right entry alone to become (part of) a leaf entry.

The buffer concept and the algorithmic idea of taking hitchhikers along are key to the proposed anytime clustering algorithm. It allows the algorithm to be interrupted at any point in time and making best use of future descents down the tree. Moreover, unlike global aggregates, objects are kept separate as long as time permits.

When a leaf node is reached and the insertion would cause a split, the algorithm checks whether there is still time left. If there is no time for a split, the closest two entries are merged. For tracking of concept drift, novelty, etc.

in the output clustering, leaf node entries contain a unique id. When they are created they are assigned a unique number in increasing order. When entries are merged, this is recorded as a pair of ids in a merging list.

The ClusTree can be initialized to improve the starting structure of the tree. Given an initial set of objects, each is transformed to a new CF. The leafs and internal nodes may be ordered for best structural properties through recursive top-down partitioning along the dimension with the largest variance and such that each partition contains equally many objects. Or any clustering algorithm, e.g. expectation maximization (EM) [DLR77] or k-means, can be used to group the objects in a top-down or bottom-up fashion to initialize the tree. However, the focus is not on optimizing the initialization phase, and the experiments are performed without it.

11.1.3 Maintaining an up-to-date clustering

In order to maintain an up-to-date view, we would like new objects to be more important than older objects. A common solution is to weigh objects with an exponential time-dependent decay function $\omega(\Delta t) = \beta^{-\lambda \Delta t}$ (cf. Chapter 3). The decay rate λ controls how much more one favors new objects compared to old ones. The higher λ is, the faster the algorithm "forgets" old data. In the ClusTree $\beta = 2$. For this basis the half life of objects is $\frac{1}{\lambda}$.

To incorporate decay, temporal information must be added to the ClusTree nodes. We ensure that the inner entries of the ClusTree still summarize their subtrees accurately by making elements of a cluster feature vector dependent on the current time t as in Definition 3.4:

$$\begin{aligned} n^{(t)} &= \sum_{i=1}^{n} \omega(t - ts_i), \quad LS^{(t)} = \sum_{i=1}^{n} \omega(t - ts_i) \cdot x_i, \\ SS^{(t)} &= \sum_{i=1}^{n} \omega(t - ts_i) \cdot x_i^2 \end{aligned}$$

n denotes the (unweighted) number of contributing objects and ts_i is the time stamp at which object x_i was added to the CF.

The additive properties of cluster features are preserved as well as the

temporal multiplicity [AHWY04]: If no object is added to a $CF^{(t)}$ during the time interval $[t, t + \Delta t]$ then

$$CF^{(t+\Delta t)} = \omega(\Delta t) \cdot CF^{(t)}.$$

Details and a proof of this property can be found in [AHWY04].

Each insertion object x carries the time stamp ts_x of its arrival time. Furthermore, each entry e_s has a time stamp $e_s.ts$ specifying its last update. It is used to compute the time that passed between the last update of an entry and $x.ts$ as the input of the decay function. Upon descending into a node, we update all entries e_s in the node to $x.ts$ by position-wise multiplication with the decay function and resetting the time stamp: $e_s.CF \leftarrow \omega(x.ts - e_s.\text{ts}) \cdot e_s.CF$, $e_s.\textit{buffer} \leftarrow \omega(x.ts - e_s.\text{ts}) \cdot e_s.\textit{buffer}$, $e_s.\text{ts} \leftarrow x.ts$. Entries in the same node always have the same time stamp, as we update all entries in the node we descend into.

The following shows that inner entries summarize their subtrees correctly. We derive an invariant that incorporates the time aspect. The cluster feature of a parent entry e_s that was last updated at $t + \Delta t$ equals the sum of the CFs of the entries in its child node updated from time t of their last update to the parent's time plus the parent's buffer.

Lemma 11.1 (ClusTree Invariant) *For each inner entry e_s with time stamp $t + \Delta t$ and decay function $\omega(\Delta t) = 2^{-\lambda \Delta t}$ it holds*

$$e_s.CF^{(t+\Delta t)} = (\omega(\Delta t) \cdot \sum_{i=1}^{\nu_s} e_{s \circ i}.CF^{(t)}) + e_s.\text{buffer}^{(t+\Delta t)}$$

Proof 11.1 *Each inner entry e_s is created first due to a split. Its summary is calculated directly as the sum of the cluster features in its child node entries $e_{s \circ i}$. The child node entries are all on the same time, because we update all entries in a node. The time stamp of the children is the insertion time $x.ts$ of the object x that caused the split. There can only be a change in one of the $e_{s \circ i}$, if there was first a change in e_s, because we always start from the root and descend downwards.*

Take the case of updating parent entry e_s (with filled buffer) to the new time $t + \Delta t + \Gamma t$, and addition of object y, where y descends into $node_s$ and gives the buffer a ride. Upon descending into $node_s$, all its entries $e_{s \circ i}$ are updated and y is added to the CF of exactly one of the $e_{s \circ i}$. e_s has a buffer, which y takes along. The buffer is also added to exactly one of the child entries' cluster features.

Following the above reasoning, we know that after updating e_s it holds that:

$$\begin{aligned} e_s.CF^{(t+\Delta t+\Gamma t)} &= (\omega(\Delta t + \Gamma t) \cdot \sum_{i=1}^{\nu_s} e_{s \circ i}.CF^{(t)}) \\ &\quad + e_s.\text{buffer}^{(t+\Delta t+\Gamma t)}. \end{aligned}$$

Because y descends into $node_s$, we update the child entries:

$$= \sum_{i=1}^{\nu_s} e_{s \circ i}.CF^{(t+\Delta t+\Gamma t)} + e_s.\text{buffer}^{(t+\Delta t+\Gamma t)}.$$

Now we give $e_s.\text{buffer}^{(t+\Delta t+\Gamma t)}$ a ride. Afterwards $e_s.\text{buffer}^{(t+\Delta t+\Gamma t)}$ contains zeros, and the values that it held before are added to the cluster feature of one of the child entries. Also adding y on both sides of the equation, once to $e_s.CF$ and once to the CF of one of the child node entries, leaves the invariant unchanged. This is also true for "hitchhiking" objects temporarily in a buffer o (replace $y.CF^{(t+\Delta t+\Gamma t)}$ with $y.CF^{(t+\Delta t+\Gamma t)} + \text{o}.CF^{(t+\Delta t+\Gamma t)}$), and if $node_s$ is a leaf node.

The last case to be check for violation of the invariant is a split. Let us consider the split of a leaf node $node_{s \circ i}$. Then two summaries in $node_s$ are computed from scratch. One overwrites the existing entry $e_{s \circ i}$ that pointed to the split node. The other one is the start of a new entry. The two new summaries naturally fulfill the invariant. The invariant also holds true for e_s, the entry pointing to $node_s$, because only the distribution of the summaries changed on the levels below e_s, not the total of the values.

Thus, maintaining a single additional field in each node with a time stamp value of its last update, and weighing according to the above scheme, ensures that decay with time is correctly captured. Weighing does not require additional memory; the weighted CFs simply replace the non-weighted cluster features.

Weighing with time provides an interesting way of avoiding splits to save valuable time. If a node is about to be split, the algorithm checks whether the least significant entry can be discarded, because it no longer contributes significantly to the clustering. Assuming that a snapshot of the ClusTree is taken regularly after t_{snap} time, the significance is tested by checking whether the entry $\hat{e}$ with the smallest $n_{\hat{e}}^{(t)}$ satisfies

$$n_{\hat{e}}^{(t)} < \beta^{-\lambda \cdot t_{snap}} \tag{11.1}$$

If this is the case, $\hat{e}$ is discarded, making room for the entry to be inserted, and avoiding a split. The summary statistics of $\hat{e}$ are subtracted from the corresponding path up to the root. Note that according to Equation 11.1 no entry is discarded if a new object has been added to it after the last snapshot has been taken. Moreover, Equation 11.1 guarantees that each entry is stored in at least one snapshot.

We discuss in Section 11.1.5 how the ClusTree results can be used to detect clusters of arbitrary shape and time horizon. Moreover, applicability of recent approaches to concept drift detection is shown.

11.1.4 Speed-up through aggregation

What happens to the index structure when it faces an exceptionally fast data stream? If insertion is interrupted at the top levels most of the time, the root and upper levels of the tree aggregate many objects in their buffers that have little chance of getting a ride down to a leaf. Worse yet, dissimilar objects which belong to different subtrees and leaves become inseparable in a buffer. The quality of the results is bound to deteriorate if we are constantly interrupted on higher levels.

The proposed solution is a speed-up through aggregation before insertion: if we do not insert each object individually, there is more time to descend deeper with an aggregate of objects. Naively, one could add up a certain number m of incoming objects, insert the aggregate, sum up the next m objects, and so on. This is essentially a global aggregate with the problem of merging arbitrary objects, even very dissimilar ones, to the same aggregate.

Clearly we need to exercise some control over which objects should – literally – "go together". Ideally, we want to aggregate objects that would end up in the same leaf if we could descend the tree with them. Most probably, the arriving objects are not all similar to each other, but we expect subgroups with inter-similarity – representatives of the clusters we also find in the tree.

The proposed solution is to create several aggregates for dissimilar objects. This makes sure that the objects summarized in the same aggregate are similar. To this end, a max_{radius} is set for the maximum distance of objects in the aggregate. max_{radius} does not need to be set by the user. The value is determined from the leaf level as the average variance of the leaves. This way, the aggregates for fast streams do not deteriorate the quality of the tree disproportionally.

For very fast streams, we store interrupted objects in their closest aggregate with respect to the distance to the mean, if this distance is below max_{radius}. If max_{radius} is exceeded, we open up a new aggregate. We insert aggregates, just as we insert single objects. Whenever the insertion thread is idle, it simply picks the next aggregate (ordered first by the number of objects in the aggregates and then by their age). The number of aggregates is limited by the stream speed, i.e. it cannot exceed the number of distance computations that can be done between two arriving items. In the case of a varying data stream the maximum number of aggregates must be set by the user, constituting the only parameter of the proposed approach. The aggregation is done by a different thread (second processor kernel), i.e. the insertion of aggregates works in parallel and is not affected in terms of processing time. If no aggregate violates the max-radius constraint, the fullest aggregate is inserted. If several aggregates are equally full, the oldest of these is inserted. Figure 11.2 summarizes the complete ClusTree algorithm.

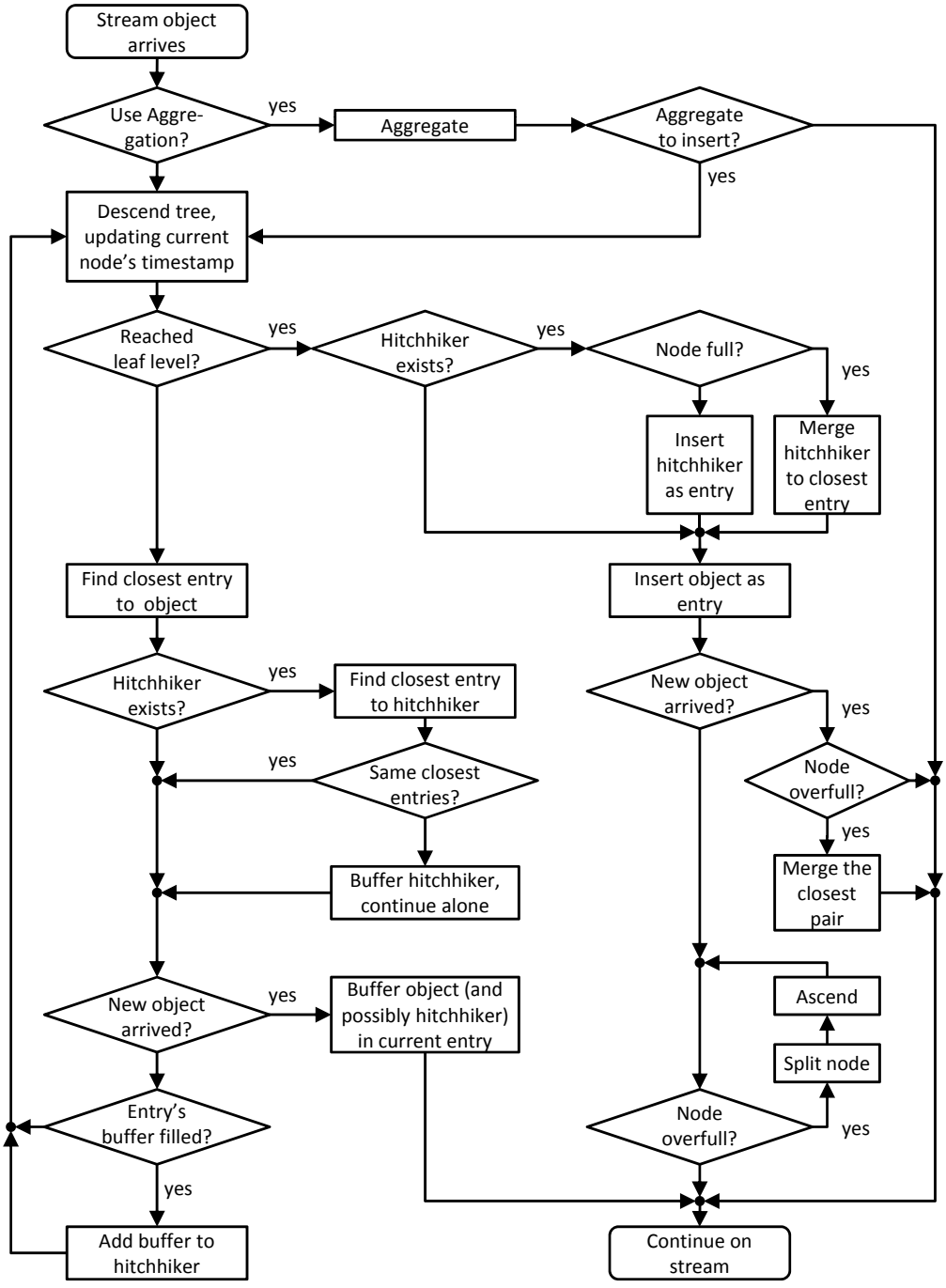


Figure 11.2: Flow chart of the ClusTree algorithm.

11.1.5 Cluster shapes and cluster transitions

The clustering resulting from the ClusTree is the set of CFs stored at leaf level, i.e. the finest representation maintainable w.r.t. the speed of the data stream. This can be seen as the online component and it allows for using various offline clustering approaches. Taking the means of the CFs as representatives we can apply a k-center clustering as in [OMM+02] or density based clustering as proposed in [CEQZ06] to detect clusters of arbitrary shape. One main advantage of the proposed approach is that it can maintain a way larger number of micro-clusters compared to other approaches [AHWY03, AHWY04, OMM+02, CEQZ06] and hence the offline clustering, e.g. density based, has finer input granularity.

Regarding cluster transitions, e.g. concept drift or novelty, many approaches proposed in the literature can directly be applied to the output of the ClusTree. Using the unique ids to every new leaf entry, we are able to track micro-clusters. Pyramidal time frames (cf. Chapter 3) allow the user to view clusterings of arbitrary time horizons. Furthermore, the ids allow us also to apply the transition detection and distinction techniques described in [SNTS06], including outlier, novelty and concept drift detection.

11.2 Analysis and experiments

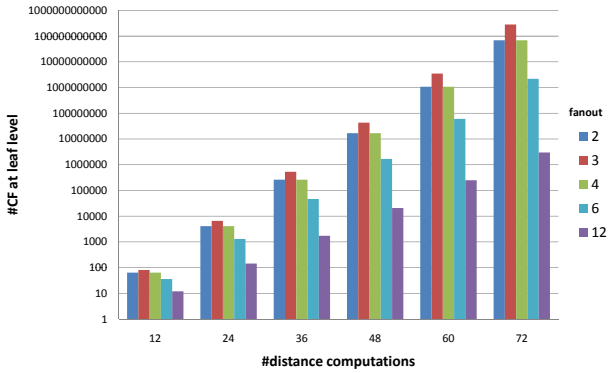
We assess the performance of the ClusTree in the following. First we examine the time and space complexity of building and maintaining a ClusTree in Section 11.2.1. In Section 11.2.2 we evaluate the anytime clustering property of the ClusTree and show the benefits of the speed-up through aggregation. Finally the adaptive clustering performance is demonstrated in Section 11.2.3 by comparing the obtained results against CluStream [AHWY03] and DenStream [CEQZ06]. The algorithms were implemented in C, all experiments were run on Windows machines with 3GHz.

11.2.1 Time and space complexity

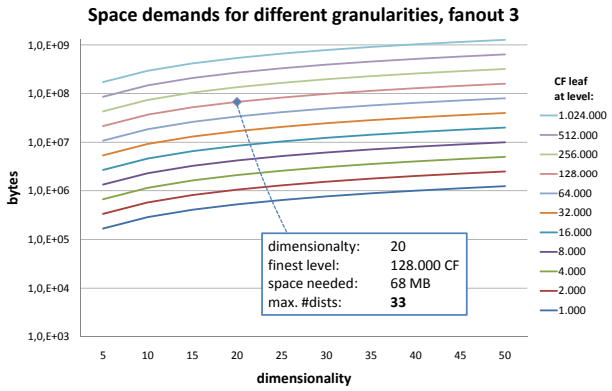
The goal for efficient and effective clustering is a high granularity with low processing costs. Therefore we investigate the effect of the fanout and of the number of distance computations required to insert an object from the stream on the granularity, i.e. the number of cluster features (CFs) at leaf level. Figure 11.3(a) shows the results for fanouts from 2 to 12. Depending on the speed of the stream, 12 to 72 distance computations are possible before interruption (multiples of 12 were chosen on the x-axis because it is the smallest multiple of all tested fanouts). As can clearly be seen in all groups of bars, a fanout of three yields the highest granularity independent of the number of distance computations, i.e. independent of the stream speed.

Next we evaluate the space demands with respect to the dimensionality and granularity in Figure 11.3(b). It shows the results for a fanout of 3 (assuming 4 Bytes per value). The space demands for the ClusTree are moderate even for high granularities and high dimensionality. For 128.000 CF at leaf level and 20 dimensions the ClusTree only needs 68 MB space, while the number of distance computations to reach the leaf level is only 33. Fanout 3, dimensionality 20 and one million CF at the finest level consume roughly 500 MB, i.e. still main memory, and the number of distance computations is still less than 40. This is opposed to any stream clustering algorithm that maintains one million micro-clusters and checks a new item against each of these. CluStream [AHWY03] for example stores q micro-clusters and must hence calculate q distances (plus possible delete $O(q)$ and merge $O(q^2)$ checks). The ClusTree only needs $O(log(q))$ many distance calculations and only stores $O(q)$ CF (cf. Figure 11.3).

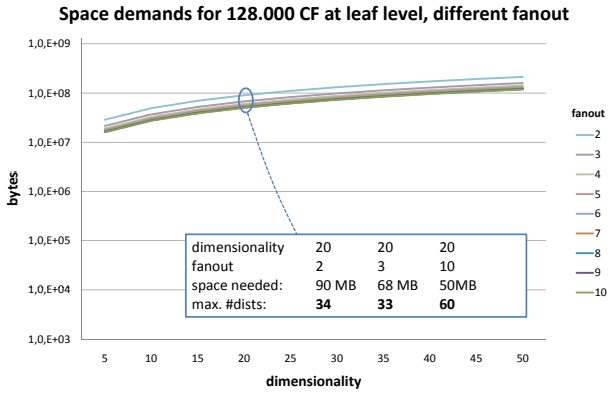
Figure 11.3(c) shows the space demand of the ClusTree for 128.000 CF at leaf level, different dimensionality and different fanout values. While a higher fanout yields less space demands, the number of distance computations that are necessary to reach the same granularity is significantly higher. Combining the results from Figure 11.3 we conclude that a fanout of 3 is the best choice in terms of time and space complexity.



(a)



(b)



(c)

Figure 11.3: a: Granularity (number of CF at leaf level) w.r.t. fanout and number of distance computations. b&c: Space consumption w.r.t granularity, fanout and dimensionality

Given the fanout of 3, the costs for a single split are low: 4 entries are present during split, hence 6 distances are calculated. A new node and one new entry for the parent node are created, and the old node and the old entry pointing to it are updated. In the worst case, the number of splits is equal to the height of the tree. Moreover, once the tree size is adapted to the stream speed and decay invalidates old entries, the number of splits is low.

11.2.2 Anytime clustering and aggregation

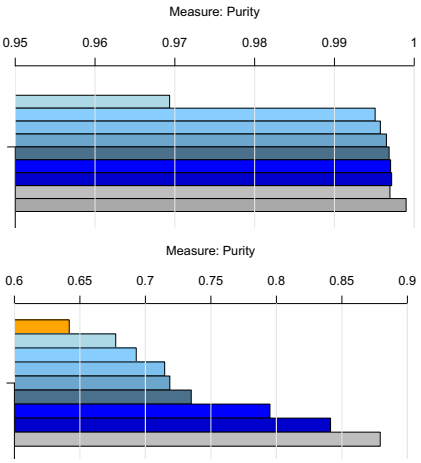
To evaluate the clustering quality of the ClusTree we evaluate the average purity of the clusters on the different levels of the tree. To determine the purity synthetic data is used as well as real world data that contains objects labeled with one of several classes. For a set K of CFs the purity is then calculated as the weighted average purity of all CFs in K: $\sum_{k=1}^{|K|} \frac{n_k}{n} \cdot \frac{\max_c(n_{ck})}{n_k} = \frac{1}{n} \sum_{k=1}^{|K|} \max_c(n_{ck})$, where n_k is the number of objects in the CF k, n_{ck} those belonging to class c and $n = \sum_{k \in K} n_k$. The real world data set Forest Covertype is available from [HB99] and contains roughly 580.000 objects from 7 classes and 10 continuous attributes. To investigate the scalability of the ClusTree in terms of dimensionality and the number of clusters the employed synthetic data sets contain 550.000 objects each (including 5% noise) and a varying number of attributes and classes. The clusters are generated as a hierarchy of Gaussians, where centers lie at a uniformly distributed angle and distance from their parents. To simulate a varying stream the arrival intervals are generated according to a Poisson process, a stochastic model that is often used to model random arrivals [DHS01] (cf. Chapter 4). For the anytime experiments the generated streams contained an expected number of 90000 points per second, i.e. $\lambda = 1/90000$.

Figure 11.4(a) shows the results for Forest Covertype (bottom) and the synthetic data set containing four classes and four dimensions (top). The results shown are the purity values after the complete data set has been processed. The top most bar (orange) represents the purity value at the root level and each following bar corresponds to the next deeper level. (For synthetic data the root level bar is not visible as the axis has been formatted

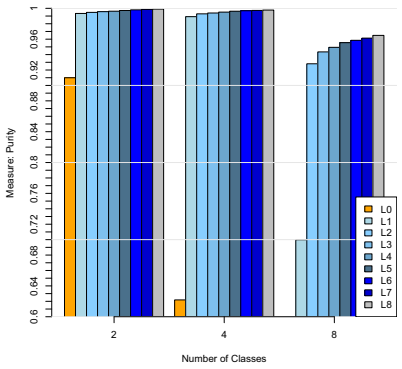
to show the difference on the lower levels.) The resulting ClusTree had ten levels for the synthetic data and 9 levels for the Forest Covertype data set. The most interesting purity value is that of the leaf level representing the finest micro-clustering granularity. It is above 99% for the synthetic data and still 88% for Forest Covertype. The purity values on the higher levels of the tree give an indication for the clustering quality for higher stream speeds. Further results on varying stream speed are shown in Section 11.2.3. Except for the leaf level, the purity values on the synthetic data set are above 95% on all levels, showing that the noise objects have been separated very well. The purity decreases more significantly for the Forest Covertype data, but is still above 70% even three levels underneath the root.

Figures 11.4(b) and 11.4(c) show the results regarding scalability using the same anytime stream as before. The number of classes varied from 2 to 8 at four dimensions and the number of dimensions from 2 to 8 using 4 classes (synthetic data). For 2 and 4 classes the quality is consistently high on all levels, just the root level purity drops at 4 classes, and further (below the shown area) for 8 classes. Increasing the number of classes to 8 shows a higher impact on the root level and also one level below the root. Although the purity decreases also on the other levels it is still above 95% on six levels, indicating a good separation of classes and even of noise objects. Comparing the results on different dimensionalities shows that the quality is similar for 4 to 8 dimensions, but lower in the 2-dimension case. This is due the fact that the overlapping of the classes is higher if the dimensionality decreases. However, once again the majority of the levels has a purity above 95%.

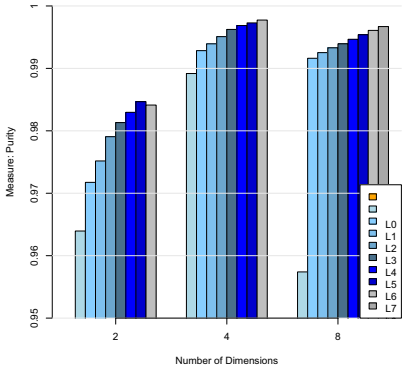
Finally, the expected number of points per second (stream speed in *pps*) was varied from 60000 to 150000. Figure 11.5 shows the resulting purity values for the leaf level and the middle level of the ClusTree for Forest Covertype. For the slowest stream the purity on the leaf level reaches 93%. While the purity is still very good (87%) at 120000 pps it drops below 70% for even faster streams with 150000 pps. For the proposed speed-up through aggregation (cf. Section 11.1), the results for 150000 pps are shown in the left part of Figure 11.5. Thanks to the aggregation the purity on the leaf level is significantly improved.



(a)



(b)



(c)

Figure 11.4: a: Clustering purity on synthetic data (top) and on Forest Covertype [HB99] (bottom). b: Scalability w.r.t. the number of clusters; c: Scalability w.r.t. the number of dimensions.

11.2.3 Adaptive clustering

To evaluate the adaptive clustering behavior of the ClusTree constant data streams were simulated with different numbers of points per second using the Forest Covertype data set. The obtained performance is compared against CluStream [AHWY03] and DenStream [CEQZ06]. For all approaches the results correspond to the online component, i.e. we analyze the properties of the resulting micro clusters and do not employ an additional offline component afterwards.

First of all we investigate the number of micro clusters that can be maintained by the individual approaches for different stream speeds. The results are shown in Figure 11.6, exact numbers are listed in Figure 11.7. As indicated, the ClusTree can maintain roughly 430000 micro clusters at 49,000 pps. With a stream speed of 140000 pps the ClusTree can still maintain 435 micro clusters. The competing approaches on the other hand can only process less than 10000 pps when maintaining 500 micro clusters. This drastic difference is due to the hierarchical structure of the ClusTree which yields only a logarithmic amount of distance computations. In other words, the number of micro clusters that can be maintained is exponential compared to CluStream or DenStream. This large number is beneficial, since the output of the online component is given to the offline component to compute the final clustering (using a clustering method of choice). A more detailed input to the final clustering enables more accurate results and detection of

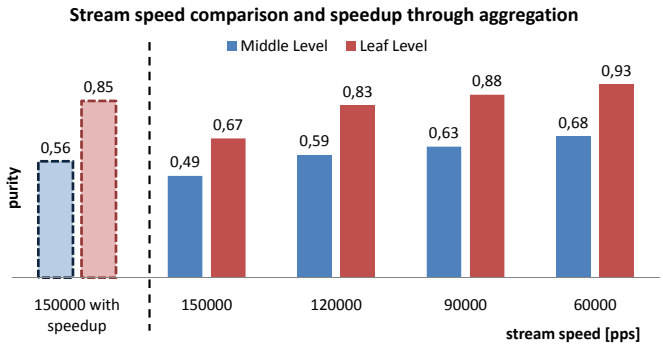


Figure 11.5: Purity with and without aggregation w.r.t. stream speed for Forest Covertype.

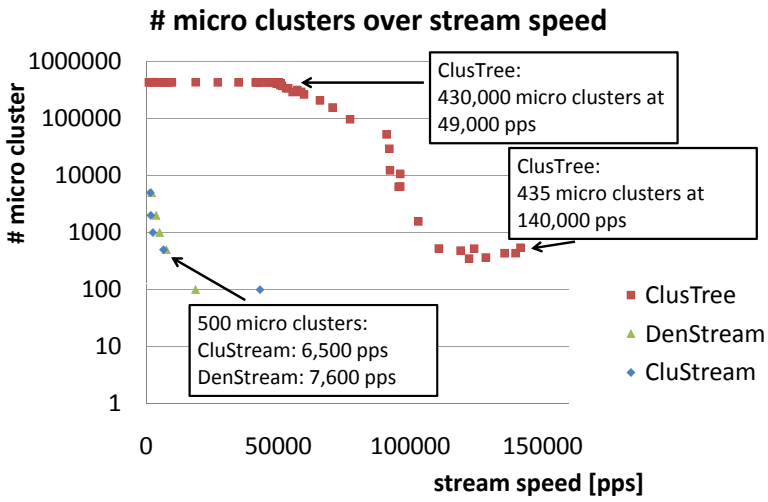


Figure 11.6: Number of micro clusters that can be maintained w.r.t. stream speed.

possible outliers. Moreover, the major advantage here is that the ClusTree automatically self-adapts to the stream speed without parametrization.

The question is at which price comes this benefit? Does the quality of the individual micro clusters deteriorate, because new points may not be added to the optimal micro cluster? To answer this question we evaluate the radius and the purity of the resulting micro clusters from all three approaches. Figure 11.8 shows the results for the ClusTree; results for CluStream and DenStream are listed in Figure 11.7.

# MC	pps	radius (median)	radius (max.)	purity
DenStream				
5000	2000	0.21	151.8	0.53
2000	3700	0.24	195.1	0.55
1000	5000	3.35	160.9	0.66
500	7600	14.01	83.7	0.53
CluStream				
5000	1500	0.02	37.4	0.70
2000	1700	0.03	224.4	0.87
1000	2500	0.33	238.8	0.90
500	6500	0.58	177.6	0.62
ClusTree				
5000	80,000	0.44	13.8	0.72
2000	94,000	0.51	18.9	0.71
1000	105,000	0.55	21.8	0.70
500	120,000	2.25	29.7	0.67

Figure 11.7: Overall results on Forest Covertype.

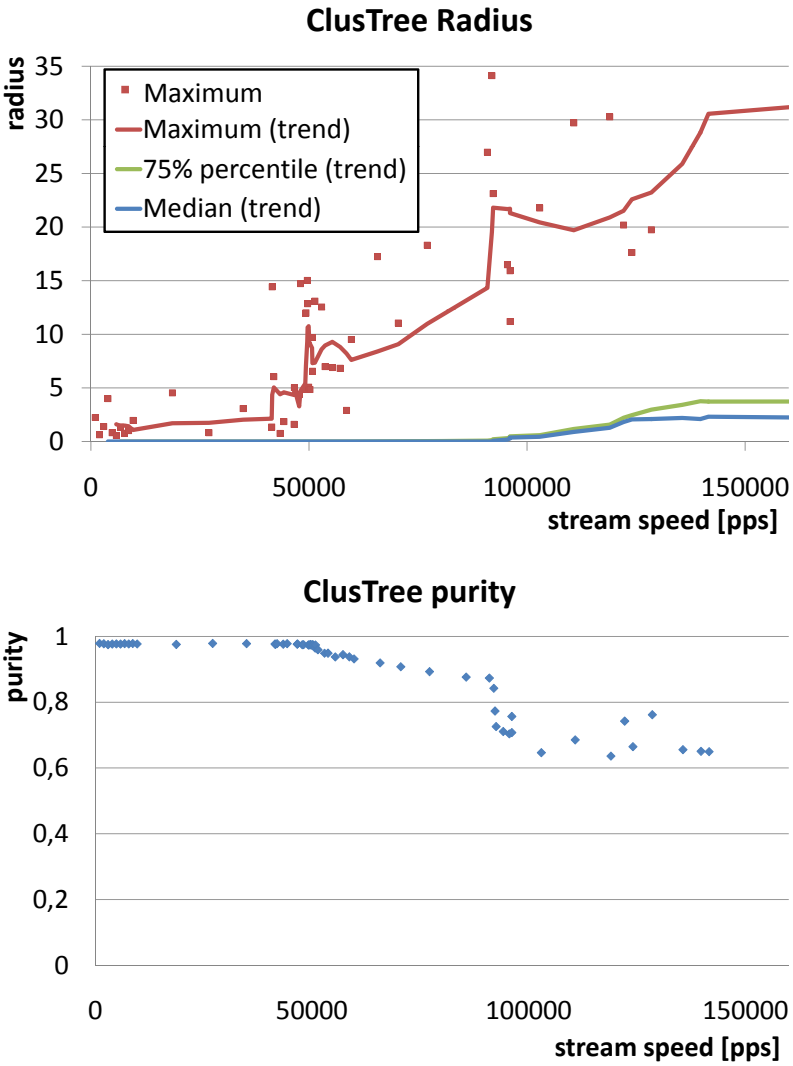


Figure 11.8: Radius (top) and purity (bottom) for ClusTree micro clusters w.r.t stream speed.

Figure 11.8 shows for the radius the maximum as well as the 75 percentile and the median. Since the actual numbers are skewed a moving average value is plotted. Naturally, with increasing stream speed, and hence decreasing number of micro clusters, the radii generally become larger. However, while we see a constant increase in the maximum value, the median and even the 75 percentile stays very low even for 100000 to 150000 pps. While DenStream produces larger micro clusters, CluStream shows a similar performance for the same amount of micro clusters. However, to maintain this amount of micro clusters CluStream can again only process slow streams where it is outperformed by the ClusTree.

The purity values for CluStream, DenStream and the ClusTree approach underline the above findings (cf. Figure 11.7). DenStream does not exceed an average purity of 70%. Clustream shows a higher purity than the ClusTree for 1000 micro clusters (90% for CluStream vs. 78% for the ClusTree), but again these numbers are not comparable due to the huge difference in terms of points per second. In conclusion it can be said that the ClusTree can maintain an equal amount of micro clusters on streams that are faster by orders of magnitude and that it can maintain an exponential amount of micro clusters at equal stream speed while providing good results in terms of cluster size (radius) and quality (purity).

11.3 Conclusion

Clustering streaming data is of increasing importance in many applications. In this chapter, a parameter free index-based approach was proposed that self-adapts to varying stream speed and is capable of anytime clustering. The ClusTree maintains the values necessary for computing mean and variance of micro-clusters. By incorporating local aggregates, i.e. temporary buffers for "hitchhikers", the ClusTree provides a novel solution for easy interruption of the insertion process that can be simply resumed at any later point in time. For very fast streams, aggregates of similar objects allow insertion of groups instead of single objects for even faster processing. In comparison to recent

approaches it was shown that the ClusTree can maintain the same amount of micro clusters at a stream speed that is faster by orders of magnitude and that for an equal stream speed the obtained granularity is exponential w.r.t. competing approaches. Moreover, a discussion was provided on the compatibility of the proposed approach with finding clusters of arbitrary shape and modeling cluster transitions and data evolution using recent approaches.

Chapter 12

Exploiting additional time in the ClusTree

* In this chapter novel descent strategies are proposed that improve the ClusTree's clustering result on slower streams as long as time permits.

12.1 Alternative descent strategies

Alternative strategies are suggested for inserting objects into micro-clusters. Assuming that the insertion process of an object is not interrupted, it continues down a single path. This path corresponds to always picking the child with the respective smallest distance between the object and the children reachable from the current node. This descent strategy down the tree can be considered a single-try *depth first* approach; it proceeds down a path that has been chosen and does not reconsider. More precisely, it does not explore continuing a path that branches further up the tree. The advantage of this approach is that the (unknown) time available to the anytime insertion process is spent on trying to reach a level as far down the tree as possible (cf. Figure 12.1). The further down the tree the object is inserted, the more fine-grained the resolution of the micro-clusters becomes. When the insertion

*This chapter has been published (together with the contents of the previous chapter) in the Knowledge and Information Systems Journal (KAIS 2011) [KABS11].

process reaches the leaf level and additional time is available, the leaf is split and hence the model size is automatically adapted to the stream speed.

For very fast streams frequent insertions on higher levels are prevented by the aggregation mechanisms (cf. Section 11.1.4). For slower data streams, however, we are very likely to often reach the leaf level and hence the model, i.e., the tree, continues to grow. As stated in the introduction, stream clustering algorithms naturally must cope with limited memory, i.e., there is a maximal model size, either dictated through the available memory or given by the user or the context program. Continuous growth of the model is thus limited by the maximal tree size and hence not ideal if more time is available.

Employing the proposed depth first strategy in this case would leave the algorithm idle once the leaf level is reached. And such idle times actually occur. One of the reasons is that the anytime processing is fast, so it reaches the leaf level after few computation steps. In the ClusTree algorithm the number of distance computations is only logarithmic in the number of maintained micro clusters, so time for further model improvement is often available even for larger model sizes. As we will see in the experimental section 12.2, maintaining 400000 micro clusters at a stream speed of 50000 points per second already yields idle times, which can be used for further computations and improvements of the clustering result.

In the following two tasks are considered: 1) finding alternative ways of choosing paths down the tree. 2) defining heuristics to exploit time that is available after reaching the leaf level.

12.1.1 Priority Breadth First Traversal

The first alternative descent strategy is a priority breadth first descent. The single path depth first descent does not perform any kind of backtracking and hence cannot correct any misguided choice that may occur due to overlapping entries on higher levels of the tree. In other words, even if we chose the closest entry on level k we cannot be sure that the closest entry on level $k+1$ is one of its children. To assure finding the closest entry on each level all entries per level are evaluated in the first heuristic. While doing this, the

entries are sorted by the distance of their corresponding parent to quickly find the closest option. Figure 12.1 b) illustrates the priority breadth first descent: instead of checking a single node on each level as in depth first descent, each level is evaluated entirely to find the closest entry by going through a list of entries sorted by the distance between the parent node and the current insertion object.

While at first the breadth first traversal sounds like many additional computations compared to linearly checking against all maintained micro clusters as in e.g. [AHWY03, CEQZ06], a closer look reveals advantages of this strategy. In the case of a binary tree the number of non-leaf entries is about equal to the number of leaf entries. However, for a higher fanout the number of inner entries is relatively smaller. Taking the fact into account that the entries are sorted according to the distance of their corresponding parent entry, we are likely to find the closest leaf level entry, i.e., the closest maintained micro cluster, within the first entries of the priority queue on the leaf level. Moreover, we are still able to perform anytime clustering, since the buffering strategies described in Section 11.1.2 are always used. The only change is that the entries on the final path are updated at the time of interruption and not as we go down the path. That means that we add the number, linear and quadratic sum when the object is inserted. Since maintaining 50000 micro clusters with a fanout of 3 yields a tree height of 10, the number of operations (additions) is negligible.

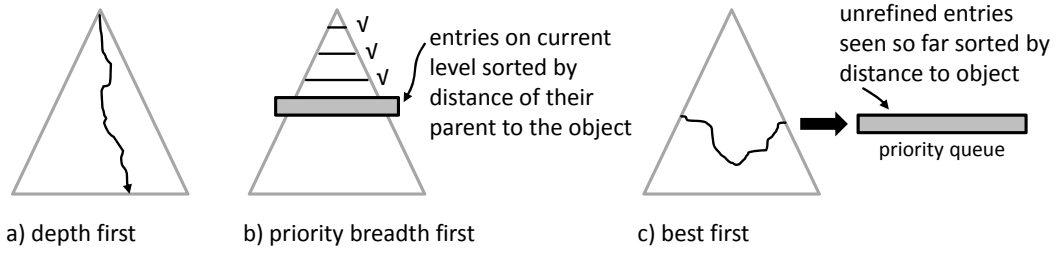


Figure 12.1: Descent strategies depth first, priority breadth first and best first.

12.1.2 Best First Traversal

As mentioned before, the underlying idea for the alternative descent strategies is based on the observation that by descending the tree depth first, we basically use a greedy approach that is not able to revise the decision for any given subtree. However, it is possible that the aggregate information on upper levels in the tree is misleading. Misleading in the sense that we may find at lower levels that an entry that had a short distance to the current object actually separates into micro-clusters at lower levels that have a comparatively high distance to the object.

In such a situation, it may be beneficial for the structure of the obtained micro-clusters to not continue on this path, but instead evaluate the situation at the children of the next-best choice. This means that we need to keep track of which options existed on the current path. This allows us to decide whether one of the branches further up on the current path has a distance to the object that is smaller than what we see for the current entry. If this is the case, we can go back and follow a different path. In query processing on indexing structures, this approach corresponds to a *best first* strategy [HS95, SAK+09] (cf. Chapter 4). To implement it, we need to maintain a priority queue while making the descent down the tree.

The priority queue contains the entries seen so far that have not been refined yet and their corresponding distance to the insertion object. Given the time to make the next step in descending down the tree, the best first approach always takes the first element from the queue, i.e., the entry which has the smallest distance to the object. The distances from the object to the entries in the corresponding child node are computed and inserted into the priority queue (refinement). This process continues until insertion is interrupted or until all nodes have been visited. As in the priority breadth first descent we keep the property of anytime clustering and update the path with cluster features (CFs) when we buffer or insert the object on interruption.

The best first strategy implies that the decision which node to refine is now based on all the information that the algorithm has at the time of the decision making. The next descent step is always to the entry that has the

smallest distance to the object, regardless of whether it yields the deepest path or not. In this sense, best first descent is a global strategy that takes all nodes into account, whereas depth first descent is local in the sense that a choice is only made among the children of the currently visited node. Figure 12.1 c) illustrates this strategy. As we can see, this algorithm maintains a priority queue of the lowest entries on all paths started so far, i.e., unrefined entries sorted by their distance to the object. The path is always continued on the path corresponding to the first entry in the queue.

12.1.3 Iterative Depth First Descent

In terms of anytime clustering, best first descent tries to optimize the selection of insertion nodes. A possible drawback of this strategy is that depending on how often the algorithm must go back and continue from upper nodes and on how soon it is interrupted, the algorithm may remain at the upper levels and buffer the object there. Similar drawbacks can be expected from the priority breadth first strategy. In contrast, depth first processing is most often able to reach leaf level.

Based on this analysis, an alternative descent strategy is suggested that tries to reach leaf level, and if more time is available uses this time to validate the decisions that were taken. This can be considered a compromise between depth first, priority breadth first and best first strategies. The approach is denoted as the *iterative depth first* descent strategy. The idea here is to start with the original depth first descent. Upon reaching leaf level we iteratively evaluate the alternatives for decisions taken at the nodes on the depth first path as long as time permits. This differs from iterative deepening as known in artificial intelligence, where a depth limit is iteratively grown to avoid ending up in infinite paths. In our case all paths are finite.

Figure 12.2 illustrates the strategy. The algorithm starts by descending down the tree as in the depth first approach (top left). Assuming that it is not interrupted, it then goes back to the root level and descends into the siblings of the entry chosen during the first iteration down the tree. Following down these alternatives to the leaf level, we eventually obtain two more candidate leaves (best fanout is 3, cf. Chapter 11) among which we can choose to insert

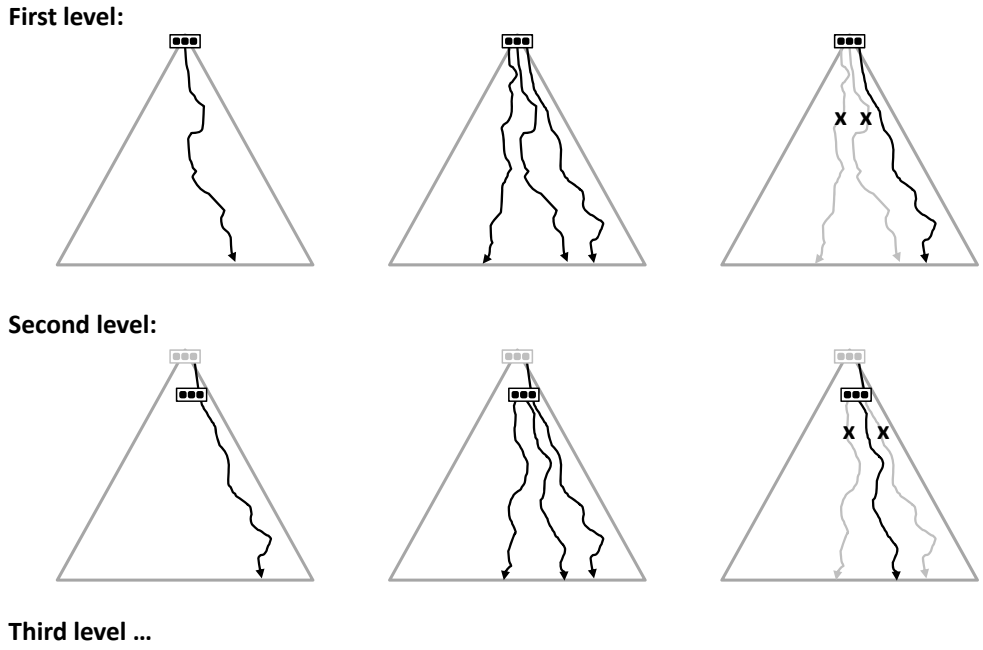


Figure 12.2: Iterative depth first descent. When the algorithm is interrupted the best leaf seen so far is chosen for insertion.

(top center of Figure 12.2). Among these three options, we now pick the best one as shown at the top right of Figure 12.2.

If there is still time, we repeat this process on the path leading to the current leaf (bottom left of Figure 12.2). We descend down the paths corresponding to the siblings of the node one level below the root on the currently chosen path (bottom center of Figure 12.2). This yields once again three options to choose from (bottom right of Figure 12.2). This process is continued until the algorithm is interrupted or until no more unchecked siblings on the path remain. On interrupt we buffer/insert and update as in the other strategies described above. All in all, using this strategy we will have at most $log^2(n)$ comparisons, e.g. for 50000 micro-clusters and a fanout of 3 about 100 comparisons, which is in stark contrast to 50000 comparisons as in e.g. [AHWY03, CEQZ06].

These alternative descent strategies complete the concept of anytime clustering in the sense that we can now use possible idle time to improve the insertion process. Whereas the depth first descent strategy stops once the

maximal model size is obtained and a leaf is reached, priority breadth first, best first and iterative depth first descent make use of additional time to check alternative insertion options. In this manner, the anytime clustering accounts for very short time spans per object through aggregation and for very long time spans through further improvement of inserts.

12.2 Evaluation of descent strategies

Due to the logarithmic number of distance computations that are necessary to maintain a certain number of micro-clusters in the ClusTree, in a given time frame we can maintain an exponential number of micro-clusters compared to linear approaches. This fact is inherent to the hierarchical ClusTree approach and was confirmed by the results in the previous chapter. If the model size, i.e., the tree size, is limited either through limited memory or user constraints, the ClusTree algorithm will be idle on slower streams once the maximal model size has been reached.

This can also be seen in some of the previous experimental results. For example, in Figure 11.6 (page 203) at the top left: the algorithm maintains approximately the same number of micro-clusters for a large range of stream speeds. It reaches the maximal number of micro-clusters at approximately 50000 pps. This means that for streams below this, the algorithm is idle once it has reached leaf level.

In this section the workings and benefits of the proposed alternative descent strategies are evaluated. *depth first*, *breadth first*, *best first* and *iterative depth first* are tested on the Forest Covertype data set using different tree heights and varying stream speeds. The tested tree heights 7, 9 and 11 correspond to roughly 2000, 20000 and 170000 possible micro-clusters at leaf level. The stream speed was varied from 600 pps to 60000 pps; on the x-axis the time per object (in μs/object) is reported such that there is more time per insertion from left to right. As in the previous experiments both the average purity values and the median of the resulting radii are measured. Figure 12.3 summarizes the results.

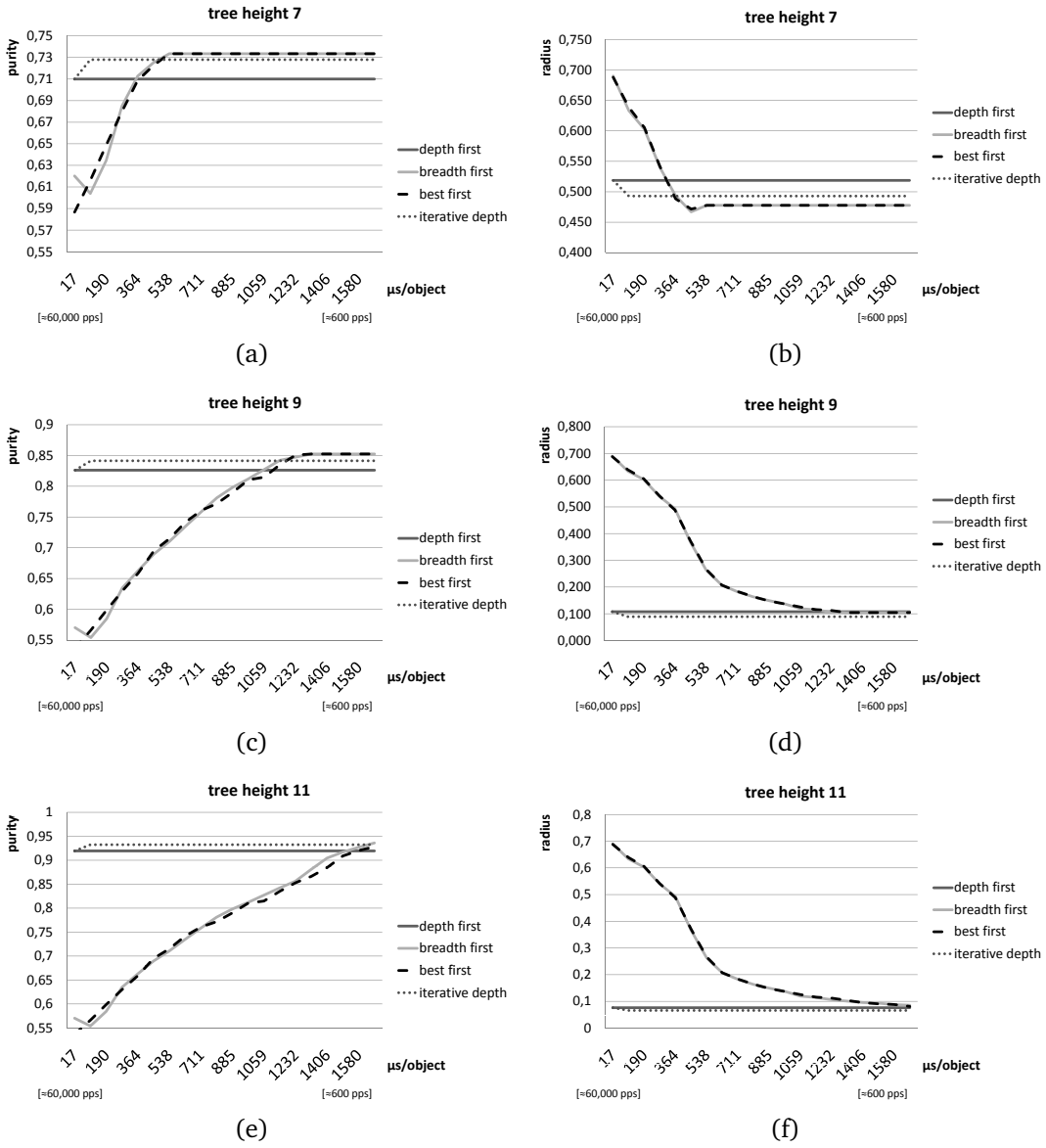


Figure 12.3: Left: purity values for different tree heights and varying stream speeds. Right: the corresponding radii (median).

Throughout the results in Figure 12.3 the primary descent strategy depth first and the novel iterative depth first descent show the same results on the fastest stream speed setting (17 μs/object, corresponding to roughly 60000 pps). This is due to the fact that both strategies start with the same initial solution. While the depth first approach stops once the leaf level is reached, the iterative depth first uses additional time and may find a better micro-cluster to insert the current object. For all tested tree heights the iterative depth first slightly improves in both measures for slower streams, i.e., higher purity values and smaller radii are achieved. Since the number of distance computations is in $O(\log^2(n))$, the iterative depth first strategy cannot profit from even more time per object. This means that for even slower streams the corresponding graphs show a stagnating behavior early on.

The *best first* and the *priority breadth first* strategies show nearly equal performance. Since neither of these strategies favors an initial descent to reach the leaf level, they process all (many) entries on the upper levels of the tree before continuing on the next level. As a consequence, these approaches do not reach the maximal tree size on faster streams. As can be seen in the left part of Figure 12.3, the full tree height is only reached at 600 μs/object for tree height 7 and 1200 μs/object for tree height 9. With this time allowance, the strategies evaluate all possible option and hence no further improvement is reached on even slower streams for the respective maximal model size. However, for all tested tree heights, both the *best first* and the *priority breadth first* strategy outperform the two depth first approaches in terms of the average purity. Regarding the achieved radii the depth first approaches are outperformed on the smaller tree and their performance is met on the larger trees for slower streams (cf. Figure 12.3 right). The results are summarized in the next section.

12.3 Conclusion

Summarizing the results for the four proposed descent strategies we see that the simple *depth first* descent yields already good results, especially on larger

tree/model sizes. The *best first* approach and the *priority breadth first* approach improved the results consistently on slower streams, which can be attributed to their strategy of testing all possible options (if time permits). The *iterative depth first* descent strategy constitutes an excellent alternative insertion strategy, since it starts with the same high performance as the depth first strategy, has very low runtime ($O(\log^2(n))$) and yet improves the initial solutions on all tested settings. Moreover, it finally reaches comparable high quality results compared to all other approaches.

Chapter 13

Robust Anytime Stream Clustering

* In this chapter the structure and working of the LiarTree are described. In the previously described ClusTree algorithm (cf. Chapter 11) the following important issues are not addressed:

- **Overlapping:** the insertion does not try to improve possible overlapping of inner entries (clusters).
- **Noise:** no noise detection is employed, since every point is treated equally and eventually inserted at leaf level. As a consequence, no distinction between noise and newly emerging clusters is performed.

The name LiarTree is due to the fact that the algorithm adds *fake* nodes to the hierarchy, which are not a direct result of aggregated streaming objects.

13.1 The LiarTree

The following describes how the issues listed above are tackled and how the drawbacks of the ClusTree are removed. Section 13.1.6 briefly summarizes the LiarTree algorithm and inspects its time complexity.

*This chapter has been published in the Proceedings of the 23rd International Conference on Scientific and Statistical Database Management (SSDBM 2011), [KRSS11].

13.1.1 Structure and overview

The LiarTree summarizes the clusters on lower levels in the inner entries of the hierarchy to guide the insertion of newly arriving objects. As a structural difference to the ClusTree, every inner node of the LiarTree contains one additional entry which is called the noise buffer.

Definition 13.1 ***LiarTree.*** *For $m \leq k \leq M$ a LiarTree node has the structure $node = \{e_1, \ldots, e_k, CF_{nb}^{(t)}\}$, where $e_i = \{ptr, CF^{(t)}, CF_b^{(t)}\}$, $i = 1 \ldots k$ are entries as in the ClusTree and $CF_{nb}^{(t)}$ is a time weighted cluster feature that buffers noise points. The amount of available memory yields a maximal height (size) of the LiarTree.*

The noise buffer consists of a single CF which does not have a subtree underneath itself. Its usage is described in Section 13.1.3.

Algorithm 13.1 illustrates the flow of the LiarTree algorithm for an object x that arrives on the stream. The variables store the current node, the hitchhiker (h) and a boolean flag indicating whether we encourage a split in the current subtree (details below). After the initialization (lines 1 to 2) the procedure enters a loop that determines the insertion of x as follows: first the exponential decay is applied to the current node in line 4. If nothing special happens, i.e. if none of the if-statements is true, the closest entry for x is determined (line 8) and the object descends into the corresponding subtree (line 24). As in the ClusTree, the buffer of the current entry is taken along as a hitchhiker (line 23) and a hitchhiker is buffered if it has a different closest entry (lines 9 to 12). Being an anytime algorithm the insertion stops if no more time is available, buffering x and h in the current entry's buffer (line 21). The issues listed above are solved in the four procedures, namely $calcClosestEntry$ (line 8), $liarProc$ (line 6), $noiseProc$ (line 14) and $leafProc$ (line 17). Details on the three methods to handle noise, novelty ($liarProc$) and drift ($leafProc$) are provided in subsections 13.1.3 to 13.1.5. The following describes how to descend and reduce overlapping of clusters using the procedure $calcClosestEntry$.

Algorithm 13.1: Process object (x)

```
1  currentNode = root; encSplit = false;
2  h = empty; // h is the hitchhiker
3  while (true) do                          /* terminates at leaf level latest */
4  |  update time stamp for currentNode;
5  |  if (currentNode is a liar) then
6  |  |  liarProc(currentNode, x); break;
7  |  end if
8  |  e_x = calcClosestEntry(currentNode, x, encSplit);
9  |  e_h = calcClosestEntry(currentNode, h, encSplit);
10 |  if (e_x ≠ e_h) then
11 |  |  put hitchhiker into corresponding buffer;
12 |  end if
13 |  if (x is marked as noise) then
14 |  |  noiseProc(currentNode, x, encSplit); break;
15 |  end if
16 |  if (currentNode is a leaf node) then
17 |  |  leafProc(currentNode, x, h, encSplit); break;
18 |  end if
19 |  add object and hitchhiker to e_x;
20 |  if (time is up) then
21 |  |  put x and h into e_x's buffer; break;
22 |  end if
23 |  add e_x's buffer to h;
24 |  currentNode = e_x.child;
25 endwhile
```

13.1.2 Descent and overlap reduction

The main task in inserting an object is to determine the next subtree to descend into. This is done by finding the closest entry; algorithm 13.2 illustrates the single steps. Besides determining the closest entry, the algorithm checks whether the object is classified as noise w.r.t. the current node and sets a $encSplit$ flag, if a split is encouraged in the corresponding subtree. The three blocks in the code correspond to the three tasks.

Algorithm 13.2: calcClosestEntry($node$, x, $encSplit$)
// returns closest entry and marks x as noise

```
1  if (node has an irrelevant entry e_irr) then
2  |  if (node is a leaf) then
3  |  |  return (e_irr, false, false);
4  |  end if
5  |  encSplit = true;
6  end if
7  calculate noise probability np(x);
8  if (np(x) ≥ noiseThreshold) then
9  |  mark x as noise;
10 end if
11 e_closest = closest entry;
12 if (!(node is a leaf)) then
13 |  e_1 = e_closest; e_2 = 2nd closest entry;
14 |  if (e_1 and e_2 overlap) then
15 |  |  look ahead: e_i* = closest entry in e_i's child;
16 |  |  reorganize: swap e_i* if radii decrease,
17 |  |  update the parent cluster features of e_i;
18 |  |  e_closest = e_1, if it contains the closest child entry; e_2 otherwise;
19 |  end if
20 end if
21 return e_closest;
```

In the first block (lines 1 to 6) we check whether the current node contains an irrelevant entry. This is done as in Chapter 11, i.e. an entry e is irrelevant if it is empty (unused) or if its weight $n_e^{(t)}$ does not exceed one point per snapshot. In contrast to the ClusTree, where such entries are only used to avoid split propagation, we explicitly check for irrelevant entries already during descent to actively encourage a split on lower levels. The reason is that a split below a node that contains an irrelevant entry does not cause an increase of the tree height, but yields a better usage of the available memory by avoiding unused entries. In case of a leaf node we return the irrelevant entry as the one for insertion (line 3), for an inner node we set the $encSplit$ flag (line 5).

In the second block (lines 7 to 10) we calculate the noise probability for the insertion object and mark it as noise if the probability exceeds a given threshold. The noise threshold $noiseThreshold$ constitutes a parameter of the algorithm and is evaluated in Section 13.2.

Definition 13.2 ***Noise probability.*** *For a node $node$ and an object o, the noise probability of o w.r.t. $node$ is*

$$np(o) = \min_{e_i \in node} \{\boldsymbol{d}(o, \mu_i)/r_i, 1\}$$

where e_i are the entries of $node$, r_i the corresponding radius (standard deviation in case of cluster features) and $\boldsymbol{d}(o, \mu_i)$ the Euclidean distance from the object to the cluster mean μ_i.

The last block (lines 11 to 20) finally determines the entry for further insertion. If the current node is a leaf node we return the entry that has the smallest distance to the insertion object. For an inner node we perform a local look ahead to avoid overlapping, i.e. we take the second closest entry e_2 into account and check whether it overlaps with the closest entry e_1 (line 14). Figure 13.1 illustrates an example.

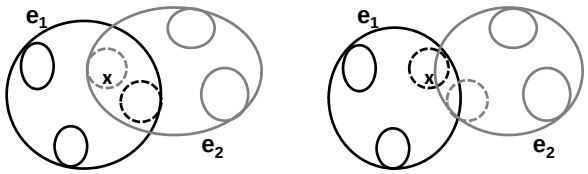


Figure 13.1: Look ahead and reorganization.

If an overlap occurs, we perform a local look ahead and find the closest entries e_1* and e_2* in the child nodes of candidates e_1 and e_2 (line 15, dashed circles in Figure 13.1 left). Next we calculate the radii of e_1 and e_2 if we would swap e_1* and e_2*. If they decrease, we perform the swapping and update the cluster features on the one level above (Figure 13.1 right). The closest entry that is returned is the one containing the closest child entry, i.e. e_1 in the example.

Algorithm 13.3: noiseProc ($node$, x, $encSplit$)
// determines whether a noise buffer has become a cluster

```
1  add x to node's noise buffer;
2  if (encSplit == true) then
3      n_avg = average weigth of node's entries;
4      ρ_avg = average density of node's entries;
5      ρ_NB = density of node's noise buffer;
6      if (gompertz(n_nb^(t), n_avg) · ρ_n ≥ ρ_avg) then
7          create a new entry e_new from noise buffer;
8          create a new empty liar root under e_new;
9          insert e_new into node;
10     end if
11 end if
```

The closest entry is calculated both for the insertion object and for the hitchhiker (if any). If the two have different closest entries, the hitchhiker is stored in the buffer CF of its closest entry and the insertion objects continues alone (cf. Algorithm 13.1 line 11).

13.1.3 Noise

As one output of algorithm 13.2 we know whether the current object has been marked as noise with respect to the current node. If so, the noise procedure is called, which is listed in algorithm 13.3. In this procedure it is regularly checked whether the aggregated noise within the buffer is no longer noise but a novel concept. Therefore, the identified object is first added to the noise buffer of the current node. To check whether a noise buffer has become a cluster, we calculate for the current node the average of its entries' weights $n^{(t)}$, their average density and the density of the noise buffer (lines 3 to 5).

Definition 13.3 ***Density.*** *The density $\rho_e = n_e^{(t)}/V_e$ of an entry e is calculated as the ratio between its weighted number of points $n_e^{(t)}$ and the volume V_e that it encloses. The volume for d dimensions and a radius r is calculated using the formula for d-spheres, i.e. $V_e = C_d \cdot r^d$ with $C_d = \pi^{d/2}/\Gamma(\frac{d}{2}+1)$ where Γ is the gamma function.*

Having a representative weight and density for both the entries and the noise buffer, we can compare them to decide whether a new cluster emerged. A cluster that forms on the current level should be comparable to the existing ones in both aspects. Yet, a significantly higher density should also allow the formation of a new cluster, while a larger number of points that are not densely clustered are further on considered noise. To realize both criteria the density of the noise buffer is multiplied to a sigmoid function, that takes the weights into account, before we compare it to the average density of the node's entries (cf. line 6). As the sigmoid function the Gompertz function [BGH+97] is used:

$$gompertz(n_{nb}, n_{avg}) = e^{-b(e^{-c \cdot n_{nb}})}$$

The parameters b (offset) and c (slope) are set such that the result is close to zero ($t_0 = 10^{-4}$) if n_{nb} is 2 and close to one ($t_1 = 0.97$) if $n_{nb} = n_{avg}$ by

$$b = \frac{\ln(t_0)^{\frac{1}{1.0-(2.0/n_{avg})}}}{\ln(t_1)^{\frac{2}{n_{avg}-2}}} \qquad c = -\frac{1}{n_{avg}} \cdot \ln(-\frac{\ln(t_1)}{b})$$

Definition 13.4 ***Noise-to-cluster event.*** *For a node* $node = (e_1, \ldots, e_k, CF_{nb}^{(t)})$ *with average weight* $n_{avg} = \frac{1}{k} \sum n_{e_i}^{(t)}$ *and average density* $\rho_{avg} = \frac{1}{k} \sum \rho_{e_i}$ *the noise buffer* $CF_{nb}^{(t)}$ *becomes a new entry, if*

$$gompertz(n_{nb}^{(t)}, n_{avg}) \cdot \rho_n \geq \rho_{avg}$$

We check whether the noise buffer has become a cluster by now, if the encourage split flag is set to true. A single inner node on the previous path with an irrelevant entry, i.e. old or empty, suffices for the encourage split flag to be true. Moreover, the exponential decay regularly yields outdated clusters. Hence, a noise buffer is likely to be checked.

If the noise buffer has been classified as a new cluster, we create a new entry from it and insert this entry into the current node. Next we create a new empty node, which is flagged as *liar*, and direct the pointer of the new entry to this node (cf. lines 7 to 9 in Algorithm 13.3). Figure 13.2 a-b) illustrate this noise to cluster event.

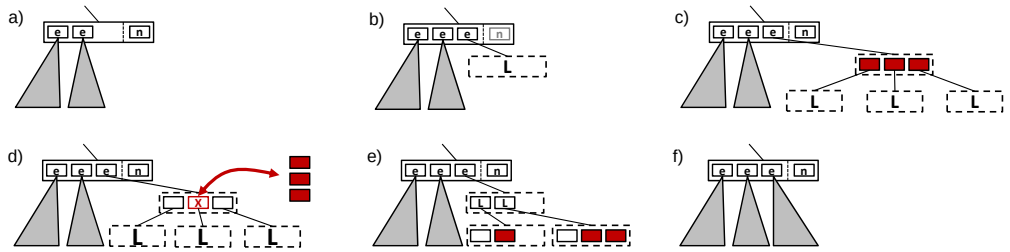


Figure 13.2: The liar concept: a noise buffer can become a new cluster and the subtree below it grows top down, step by step by one node per object.

13.1.4 Novelty

So far new nodes were only created at the leaf level, such that the tree grew bottom up and was always balanced. By allowing noise buffers to transform to new clusters, new entries and, more importantly, new nodes are created within the tree. To avoid getting an increasingly unbalanced tree through noise-to-cluster events, we treat nodes and subtrees that represent novelty differently. The main idea is to let the subtrees underneath newly emerged clusters (entries) grow top down step by step with each new object that is inserted into the subtree until their leaves are on the same height as the regular tree leaves. Leaf nodes that belong to such a subtree are called *liar nodes*, the root is called *liar root*. When we end up in a liar node during descend (cf. Algorithm 13.1), we call the liar procedure which is listed in Algorithm 13.4.

Definition 13.5 ***Liar node.*** *A liar node is a node that contains no entry. A liar root is an inner node of the liar tree that has only liar nodes as leafs in its corresponding subtree and no other liar root as ancestor.*

Figure 13.2 illustrates the liar concept, the image is referred to in the description of the single steps. A liar node is always empty, since it has been created as an empty node underneath the entry e_{parent} that is pointing to it. Initially the liar root is created by a noise-to-cluster event (cf. Fig. 13.2 b)). To let the subtree under e_{parent} grow in a top down manner, additional new entries e_i must be created (cf. red entries in Figure 13.2). Their cluster features CF_{e_i} must fit the CF summary of e_{parent}, i.e. their weights, linear and

Algorithm 13.4: liarProc ($liarNode$, x)
// refines the model to reflect novel concepts

1 create three new entries with dim dimensions $e_{new}[\,]$;
2 **for** *($d = 1$ to dim)* **do**
3 $\quad e_{new}[d \bmod 3].LS[d] = (e_{parent}.LS[d])/3 + \text{offset}_A[d]$;
4 $\quad e_{new}[(d+1) \bmod 3].LS[d] = (e_{parent}.LS[d])/3 + \text{offset}_B[d]$;
5 $\quad e_{new}[(d+2) \bmod 3].LS[d] = (e_{parent}.LS[d])/3 + \text{offset}_C[d]$;
6 $\quad e_{new}[d \bmod 3].SS[d] = F[d] + (3/e_{parent}.N) \cdot (e_{new}[d \bmod 3].LS[d])^2$;
7 $\quad e_{new}[(d+1) \bmod 3].SS[d] = F[d] + (3/e_{parent}.N) \cdot (e_{new}[(d+1) \bmod 3].LS[d])^2$;
8 $\quad e_{new}[(d+2) \bmod 3].SS[d] = F[d] + (3/e_{parent}.N) \cdot (e_{new}[(d+2) \bmod 3].LS[d])^2$;
9 **end for**
10 insert x into the closest of the new entries;
11 **if** *($liarNode$ is a liar root)* **then**
12 $\quad$ insert new entries into $liarNode$;
13 **else**
14 $\quad$ remove e_{parent} in parent node;
15 $\quad$ insert new entries into parent node;
16 $\quad$ split parent node (stop split at liar root);
17 **end if**
18 **if** *(non-empty liar nodes reach leaf level)* **then**
19 $\quad$ remove all liar flags in corresponding subtree ;
20 **else**
21 $\quad$ create three new empty liar nodes under $e_{new}[\,]$;
22 **end if**

quadratic sums must sum up to the same values. We create three new entries and assign each a third of the weight from e_{parent}. We displace the new means from the parent's mean by adding three different offsets to its mean (a third of its linear sum, cf. lines 3 to 5). The offsets are calculated per dimension under the constraint that the new entries have positive variances. We set one offset to zero, i.e. $\text{offset}_A = 0$. For this special case, the remaining two offsets can be determined using the weight n_e^t and variance $\sigma_e^2[i]$ of e_{parent} per dimension as follows

$$\begin{aligned} \text{offset}_B[i] &= \sqrt{\frac{1}{6} \cdot (1 - \left(\frac{1}{3}\right)^4)} \cdot (n_e^t) \cdot \sigma_e^2[i] \\ \text{offset}_C[i] &= -\text{offset}_B[i] \end{aligned}$$

The zero offset in the first dimension is assigned to the first new entry, in the second dimension to the second entry, and so forth using modulo counting (cf. lines 3 to 8). If we would not do so, the resulting clusters would lay on a line, not representing the parent cluster well. The squared sums of the three new entries are calculated in lines 6 to 8. The term $F[d]$ can be calculated per dimension as

$$F[d] = \frac{n_e^t}{3} \cdot \left(\frac{\sigma_e[d]}{3}\right)^4$$

Having three new entries that fit the CF summary of e_{parent}, we insert the object into the closest of these and add the new entries to the corresponding subtree (lines 11 to 17). If the current node is a liar root, we simply insert the entries (cf. Figure 13.2 c)). Otherwise we replace the old parent entry with the three new entries (cf. Figure 13.2 d)). We do so, because e_{parent} is itself also an artificially created entry. Since we have new data, i.e. new evidence, that belongs to this entry, we take this opportunity to detail the part of the data space and remove the former coarser representation. After that, overfull nodes are split (cf. Figure 13.2 d-e)). If an overflow occurs in the liar root, we split it and create a new liar root above, containing two entries that summarize the two nodes resulting from the split (cf. Figure 13.2 e)). The new liar root is then put in the place of the old liar root, whereby the height of the subtree increased by 1 and it grew top down (cf. Figure 13.2 e)).

In the last block we check whether the non empty leaves of the liar subtree already reach the leaf level. In that case we remove all liar flags in the subtree, such that it becomes a regular part of the tree (cf. line 19 and Figure 13.2 f)). If the subtree does not yet have full height, we create three new empty liar nodes (line 21), one beneath each newly created entry (cf. Figure 13.2 c)).

Algorithm 13.5: Leaf proc. ($leafNode$, x, h, $encSplit$)
// inserts object x and hitchhiker h (if any) into leaf node

```
1  if (time is up) then
2  |  insert x and h as entries, possibly merging closest pairs on
   |  overflow;
3  else
4  |  if (node is full and encSplit == false) then
5  |  |  merge hitchhiker to closest entry;
6  |  else
7  |  |  insert hitchhiker as entry;
8  |  end if
9  |  insert x as entry;
10 |  if (node is overfull) then
11 |  |  split node and propagate split;
12 |  end if
13 end if
```

13.1.5 Insertion and Drift

Once the insertion object reaches a regular leaf, it is inserted using the leaf procedure (cf. algorithm 13.5). If there is no time left, the object and its hitchhiker are inserted such that no overflow, and hence no split, occurs (line 2). Otherwise, the hitchhiker is inserted first and, if a split is encouraged, the insertion of the hitchhiker can also yield an overflowing node. This is in contrast to the ClusTree, where a hitchhiker is merged to the closest entry to delay splits. In the LiarTree splits are explicitly encouraged to make better use of the available memory (cf. Definition 13.1). After inserting the object we check whether an overflow occurred, split the node and propagate the split (lines 9 to 12).

Three properties of the LiarTree help to effectively track drifting clusters. The first property is the aging, which is realized through the exponential decay of leaf and inner entries as in the ClusTree. The second property is the fine granularity of the model. Since new objects can be placed in smaller and better fitting recent clusters, older clusters are less likely to be affected through updates, which gradually decreases their weight and they eventu-

ally disappear. The third property stems from the novel liar concept, which separates points that first resemble noise and allows for transition to new clusters later on. These transitions are more frequent on levels close to the leaves, where cluster movements are captured by this process.

13.1.6 Logarithmic time complexity

The LiarTree algorithm is summarized in the following and a proof of its worst case time complexity is sketched. **Summary:** To insert a new object, the closest entry in the current node is calculated. While doing this, a local look ahead is performed to possibly improve the clustering quality by reduction of overlap through local reorganization. If an object is classified as noise, it is added to the current node's noise buffer. Noise buffers can become new clusters (entries) if they are comparable to the existing clusters on their level. Subtrees below newly emerged clusters grow top down through the liar concept until their leaves reach the regular leaf level. If the insertion is interrupted the current object is buffered as in the ClusTree to allow for anytime clustering.

Lemma 13.1 ***LiarTree time complexity*** *The clustering model $\mathcal{M}$ of a liar tree consists of the micro clusters stored in its leaf nodes. A liar tree has by definiton a maximal height (cf. Def. 13.1) and hence its model has a maximal size $|\mathcal{M}| =: m$. The time complexity for inserting an object o into a liar tree of model size m is $O(\log m)$*

A sketch to prove lemma 13.1 is provided using Algorithm 13.1.

Proof 13.1 *(Sketch.) Let h be the height of the LiarTree, then h is logarithmic in m. The initialization takes constant time. The same holds for adding objects to cluster features (lines 11, 19, 21 and 23) and for the noise procedure $noiseProc$ (line 14). The two methods $liarProc$ (line 6) and $leafProc$ (line 17) basically have also constant complexity except for the split, which can be called maximally h times. Hence, these two methods are in $O(\log m)$. Since all three of the above methods are maximally called once per insertion object and afterwards the loop is left with a* ***break*** *statement (same lines), we are*

*still in $O(\log m)$. We still must prove the complexity of lines 8 and 9 and the termination of the **while** loop. Since the look ahead is local (one level only) the $calcClosestEntry$ procedure (lines 8 and 9) has a constant time complexity. The loop is called once per level (after each descent), i.e. it only depends on h and is therefore also in $O(\log m)$. Hence, the total time complexity of the LiarTree algorithm is logarithmic in the size of the clustering model, i.e. the number of maintained micro clusters at the leaf level.*

13.2 Experiments

To evaluate the performance of the LiarTree different stream scenarios are simulated to evaluate the radii of the resulting clusters as well as the recall, precision and F1 measure. Synthetic data is generated (details below) such that the ground truth is known for comparison. Precision and recall are calculated using a Monte Carlo approach: for the recall points are generated inside the ground truth and checked whether these are included in the found clustering. For the precision the process is reversed, i.e. points are generated inside the found clustering and checked whether they are inside the ground truth. In other words, the recall corresponds to the ground truth area that is found by the algorithm, precision corresponds to the proportion of the found area that is correct, i.e. without the unnecessary parts.

The synthetic data stream is generated using an radial basis function approach with additional noise. For a given number of clusters k and a given radius r points are generated equally at random within k hyperspheres of radius r. Noise is added equally distributed at random in the unit cube. Novelty is simulated by adding new clusters, drift is generated by moving the cluster means along individual vectors with a given drift speed. The drift speed sets the distance that a cluster moves every 1000 points (total). If a cluster is about to drift out of the unit cube, its corresponding movement vector is reflected such that it stays inside. If not mentioned differently the parameters are set to $k = 5$, $r = 0.05$ and drift speed$= 0.02$ at 20% noise in the four dimensional unit cube. Single parameters are varied when mentioned and the average values of the measures are reported per algorithm.

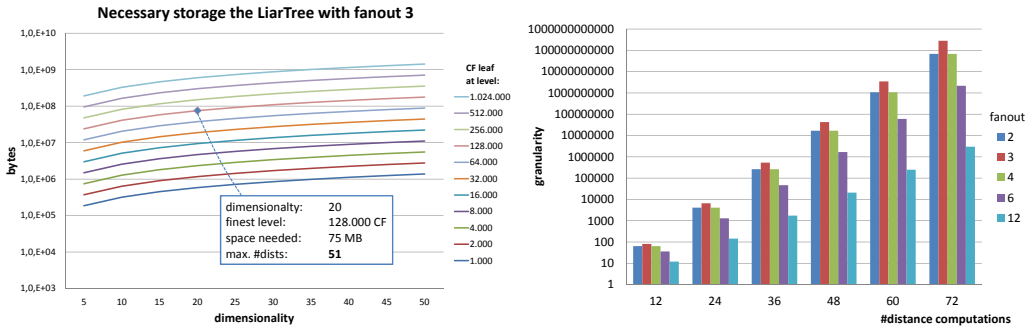


Figure 13.3: Influence of fanout and granularity on the LiarTree.

The liar tree is compared to the ClusTree algorithm on varying data streams using a Poisson process (cf. Def. 4.8, p. 94) as in previous chapters. On constant data streams, the liar tree is compared to DenStream [CEQZ06] and to the CluStream approach proposed in [AHWY03].

Figure 13.3 shows an evaluation of the influence of the fanout on the granularity and the number distance computations to reach the leaf level. Since the LiarTree extends the ClusTree, the results regarding time and space complexity are similar and can partly be transferred from the detailed analysis presented in Chapter 11. Due to the additional noise buffer the LiarTree needs one more distance computation per node and the additional functionality such as the liar concept are more expensive than the simple buffering in the ClusTree. However, as shown in Section 13.1.6 the additional methods are called maximally once per object and therefore the total descend is still logarithmic. As Figure 13.3 shows, a fanout of 3 yields the highest granularity at leaf level for the liar tree. This is in accordance with the results from

# MC	pps DenStream	pps CluStream	pps ClusTree	pps LiarTree
5000	2000	1500	80000	72000
2000	3700	1700	94000	84000
1000	5000	2500	105000	93000
500	7600	6500	120000	105000

Figure 13.4: The maximal points per second that can be processed by the approaches for differnet model sizes.

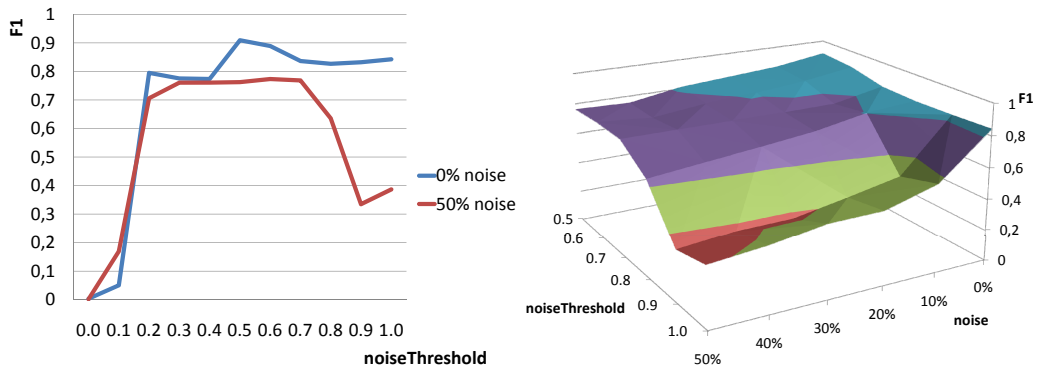


Figure 13.5: Robustness of the LiarTree to noise and the noise threshold parameter.

Chapter 11, where it yielded the best trade off between space demands and computation time. Hence, the fanout of the LiarTree is set to 3, i.e. three entries (plus noise buffer) per inner node of the tree.

Figure 13.4 shows for different model sizes (number of micro clusters #MC) the maximal number of points per second (pps) that can be processed by the individual approaches. For CluStream and DenStream the model size was fixed and the maximal pps were counted. For ClusTree and LiarTree the stream speed was fixed and the resulting maintainable model size was measure. The results are again in line with Chapter 11 and can be explained by the logarithmic time complexity of the tree approaches, i.e. at the same speed they can maintain a model that is larger by orders of magnitude.

To evaluate the noise threshold parameter of the LiarTree (cf. Section 13.1.2), the left part of Figure 13.5 shows the resulting F1 measure for 0% noise and 50% noise over the whole range of the noise threshold, the right part of the figure shows the corresponding values for all noise levels from 0% to 50% and noise thresholds from 0.5 to 1.0. The most important observation from this experiment is that the LiarTree shows good performances on a rather wide range, i.e. for a noise threshold from 0.2 up to 0.7 or 0.8. To both ends of the scale, i.e. close to zero or one, the performance drastically drops (except for 0% noise at a noise threshold close to 1.0). The performance drop for very low parameter values results from a decreasing recall, since nearly every point is considered noise in that case. For very high noise thresholds a

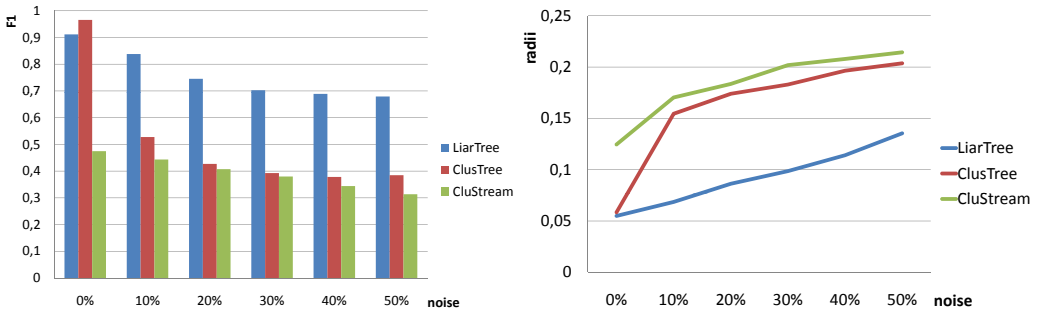


Figure 13.6: F1 measure and resulting radii for LiarTree, ClusTree and CluStream for different noise levels.

loss in precision causes the F1 measure to drop, since new points from drifting clusters are then more likely to be added to existing micro clusters rather than creating a new micro cluster using the liar concept. As a consequence the area covered by the older micro cluster increases and is likely to cover unnecessary parts of the data space. From the above results any choice between 0.2 and 0.8 for the noise threshold can be justified, 0.7 is used in the following. Summarizing Figure 13.5 we can notice that the LiarTree is rather robust against the choice of the noise threshold parameter.

Figure 13.6 (left) shows the F1 measures of LiarTree, ClusTree and CluStream for noise levels from 0% to 50%. To compare to the CluStream approach a maximal tree height of 7 was set and CluStream was allowed to maintain 2000 micro clusters. The parameters for the DenStream algorithm are difficult to set and greatly affect the quality of its results, such that it was only used for the performance comparison. Comparing LiarTree and ClusTree one can see that the ClusTree obtains an even slightly better F1 measure for 0% noise, but that it falls significantly behind that of the LiarTree in the presence of noise. CluStream meets the performance of the ClusTree in the presence of noise, but cannot profit from 0% noise and exhibits a significantly worse F1 measure in that case. The reason for the superiority of the LiarTree lies in its explicit handling of noise. As can be seen in the right part of Figure 13.6, the radii of the resulting offline clusters are more compact (compare to 0.05 ground truth) and hence the precision improves, i.e. less unnecessary covered area.

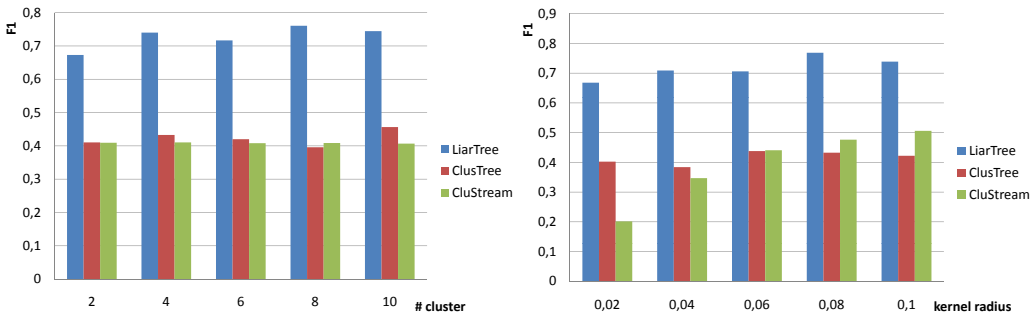


Figure 13.7: Varying the data stream's number of clusters and their radius.

Next the parameters k and r are varied, i.e. the number of ground truth clusters and their radius. Figure 13.7 shows the results for the three approaches. While the performance of the approaches obviously does not depend on the number of ground truth clusters, the results differ for varying radii (cf. Figure 13.7 right). The F1 measure for CluStream improves with an increasing radius. This is in line with the previous results. The missing noise handling causes the found clusters to be larger and consequently the precision drops due to unnecessarily covered areas, but with an increasing ground truth radius, the area covered by the ground truth increases and the above effect is slightly diminished. While this effect is not equally distinct for the ClusTree, the LiarTree clearly outperforms both approaches on all settings in this experiment.

To test the performance of the approaches in case of newly emerging clusters, data streams were generated where the number of clusters k is increased abruptly. We expect the recall of the approaches to drop significantly shortly after the introduction of novelty. However, the results did not show any salience in either measure, i.e. the transition was always smooth and the absolute values did not differ (cf. Figure 13.7 left). While all approaches seem capable to immediately react to newly emerging clusters, the overall performance of the LiarTree was above that of the other approaches regardless of the number of clusters (cf. Figure 13.7 left).

The next experiments evaluate the performance of the three approaches on data streams with varying drift speed. Figure 13.8 shows the resulting values for F1 and radii. As can be seen in the left part, both the LiarTree

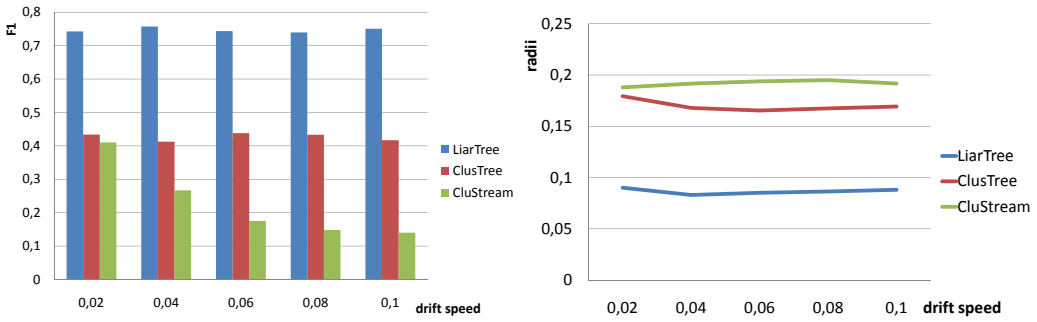


Figure 13.8: Varying the drift speed for LiarTree, ClusTree and CluStream.

and the ClusTree are not affected by higher stream speed, i.e. their F1 measure exhibits a stable value regardless of the speed. However, the LiarTree consistently outperforms the ClusTree, which proves the liar concept to be effective in the presence of drift and, as seen before, in the presence of noise. The main reason for the difference in the F1 measure is the poorer precision values of the ClusTree, details are provided below. The CluStream approach can compete with the ClusTree for slow drift speeds in this experiment, but falls significantly behind when the drift accelerates. Its drop in performance results from both decreasing recall and precision, while the latter has clearly the stronger influence.

Before looking at the details of precision and recall we shortly inspect the resulting radii over varying drift speeds in the right part of Figure 13.8. As before, the radii shown in the plot correspond to the offline component and must be compared to 0.05 for the ground truth. All three approaches show constant values over the various drift speeds, which is due to their property of removing older data to keep track of the more important recent data. The radii resulting from the CluStream approach are two to three times larger than the ground truth. Similar values are obtained by the ClusTree for this setting, i.e. allowing a comparable number of micro clusters to both approaches. As was detailed in Chapter 11, the ClusTree can maintain way more clusters in the same time due to its logarithmic time complexity in contrast to CluStream. The same holds for the LiarTree, since it has the same logarithmic complexity.

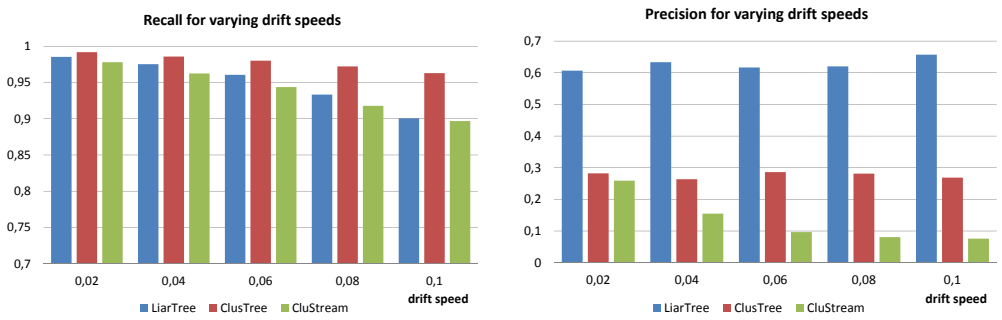


Figure 13.9: Precision and recall of the approaches for varying drift speeds.

Figure 13.9 details the precision and recall values of the approaches over varying drift speeds. The left part shows that the recall values for CluStream and LiarTree slightly decrease with faster drift speeds (mind the scale compared to the right part). The reason is that both approaches adapt to the drift and delete the oldest micro clusters in the process. The property of the LiarTree to actively encourage splits and create new entries can yield early outdated micro clusters in some cases. In contrast, in the ClusTree the new points are more likely to be added to existing concepts, which causes slightly increasing radii and therefore a higher recall value. However, this small benefit of the ClusTree is paid by a significantly worse precision compared to the LiarTree (cf. right part of Figure 13.9). While the ClusTree can maintain its level of precision with increasing drift speed, the CluStream approach suffers a severe loss in precision in the presence of noise and faster drifts. Once more the LiarTree clearly outperforms both approaches, showing the effectiveness of its new concepts.

13.3 Conclusion

In this chapter the LiarTree was introduced. It constitutes an anytime stream clustering algorithm, which avoids overlapping through local look ahead and reorganization and incorporates explicit noise handling on all levels of the hierarchy. It allows the transition from local noise buffers to new entries (micro clusters) and grows novel subtrees top down using its liar concept,

which makes it robust against noise and changes in the distribution of the underlying stream. Experimental evaluation showed for various data stream scenarios that the LiarTree outperforms competing approaches in the presence of noise and evolving data, proving its novel concepts to be effective.

This chapter concludes the algorithmic contributions to anytime stream clustering. In the following an application of the ClusTree is introduced in Chapter 14. An open source framework for stream data mining is presented in Chapter 15 along with a novel evaluation measure for clustering on evolving data streams. Future work in the area of stream clustering is discussed in Chapter 16.

Chapter 14

Application: Using Modeling for Anytime Outlier Detection

* Besides classification and clustering, outlier detection is another important mining task on data streams. Anytime outlier detection denotes the task of determining within any period of time whether an object is anomalous. The more time is available, the more reliable the decision should be. The solution presented in this chapter is based on the previously presented ClusTree. The effectiveness of the proposed method is shown in experimental evaluation on varying and constant streams.

14.1 Introduction

Outlier detection has been defined by Hawkins as the task of finding "an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism" [Haw80]. Based on this popular definition, different approaches for formalizing outliers and for algorithmically uncovering them have been proposed in the literature. In statistics, the general idea is to analyze the data by assuming that it follows a certain distribution [BL94]. In distance-based outlier detection, outliers

*This chapter has been presented at the International Workshop on Novel Data Stream Pattern Mining Techniques (StreamKDD 2010) in conjunction with ACM SIGKDD [AKBS10].

are defined as objects which do not have at least a certain number of objects within a given threshold distance [KNT00]. Another research direction is the use of clustering to identify the prevailing groups of data objects, which defines outliers as objects that are not clustered or far from cluster centers [EKSX96, HXD03].

For streaming data, some outlier detection algorithms have been proposed recently [Agg05, AF07, SPP+06, YiTWM04]. These algorithms successfully determine outliers for streams with constant inter-arrival rates. The approaches, however, are not capable of working effectively in an anytime environment. They may not be able to keep up with streams of varying speed, which leads to loss of data or unpredictable drop in accuracy for bursty streams. Also, if more time is available, these algorithms are not capable of using this time in order to improve the reliability of the result.

In anytime outlier detection the time available to the algorithm is used to refine the outlier detection and to increase the reliability of the result. The approach presented in this chapter employs the ClusTree to determine the outlier score based on the distance between object and cluster. Therein, until interrupted, more fine grained resolutions of the clustering structure are analyzed.

14.2 Related work

For outlier detection, several paradigms have been introduced in the research literature. In supervised outlier detection, a problem similar to unbalanced classification is studied [ZKF05, FZZ09]. However, supervised methods require training data with labeled outliers, which is often not the case in practice.

Unsupervised approaches do not assume the availability of training data but aim to identify outliers based on their deviation from the remainder of the data, similar in spirit to the widely known Hawkins definition [Haw80]. Among unsupervised outlier detection, different models for determining deviation exist. In statistical outlier detection, the core concept is to assume

that the data follows a certain data distribution, and to identify those objects that do not well fit this assumption [BL94]. Distance-based outlier detection finds objects that show a high distance to most other data objects [KNT00]. Parameters to this approach are the proportion of objects at a high distance, and a distance threshold. Clustering-based outlier detection uses clusters as a means to identify the inherent structure of valid data, and determines outliers as objects that are not clustered well [EKSX96, HXD03, MSS11].

While many traditional methods assume that data can be separated into outliers and inliers (valid data), finding such a clear boundary may prove difficult for many data sets. The Local Outlier Factor (LOF) method relaxed this requirement in favor of a scoring function that reflects the degree of deviation of an object [BKNS00]. The result of the outlier detection is then not a set of outliers, but a ranking of objects by their degree of outlierness. This idea has since been incorporated into other outlier detection methods, such as top-n outlier detection [JTH01]. An extension of LOF for very high dimensional spaces has recently been proposed in [dVCH10]. The basic idea is to first find candidate nearest neighbors in a random projection space of lower dimensionality and refine these in a second step in the original space.

Specialized techniques for different application areas study harmonization of local outlier factors to probabilities [KKSZ09] or focus on specific data types such as time series [MSV04]. While time series bear some resemblance with streaming data, the important difference is that time series data is assumed to be available entirely at the time of outlier detection (cf. Chapter 2). Moreover, the goal is not to identify outlying objects as in data streams, but outlying patterns that involve several values that are neighboring along the time axis. Thus, approaches for time series outlier detection are not applicable to the problem of finding outliers in streaming data.

Recently, many approaches for outlier detection on data streams have been proposed [Agg05, AF07, SPP+06, YiTWM04, FG08, ZGW08, VGN08, ELN+08, RWZH09, YRW09, ZGWW10, CZSC10, SG10]. However, all of these approaches assume streams of fixed arrival rates and do not meet the requirement of anytime outlier detection that more time leads to better accuracy.

14.3 Detecting outliers in streaming data

Before introducing the proposed AnyOut algorithm the problem of anytime outlier detection is introduced more formally.

14.3.1 The anytime outlier detection problem

In outlier detection, the goal is to identify objects which deviate from the remainder of the data. This is illustrated in a simplified example in Figure 14.1: The majority of data objects (depicted in black) is considered to represent valid patterns in the data, whereas singular objects such as the one at the top left (red colored) are typically assumed to be outliers. As discussed in the previous section, different research approaches for determining deviation have been proposed in the literature, and they arrive at different formalizations of the outlier notion or degree of outlierness. The formalization of the anytime outlier detection problem abstracts from the concrete notions of what constitutes an outlier. Different outlier detection paradigms may be followed in order to solve this problem. As stated above, the solution presented in Subsection 14.3.2 follows a cluster-based approach for solving anytime outlier detection.

As briefly sketched in Related Work as well, it may prove difficult to determine clear decision boundaries between outliers and inliers. Typically, objects deviate from the majority of the data objects to a varying degree. Consequently, many outlier detection techniques aim to capture this degree of deviation in a corresponding outlier score. In the following discussion it

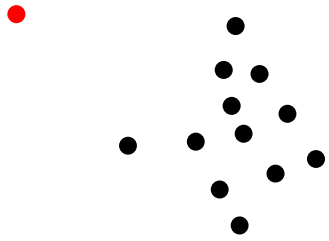


Figure 14.1: Cluster-based outlier detection: objects deviating from prevailing patterns in the data are considered outliers.

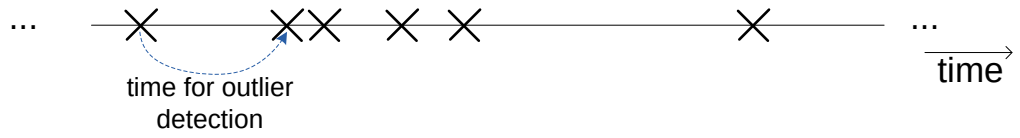


Figure 14.2: Streaming data with varying inter-arrival rates requires flexible anytime algorithms that are capable of processing each object within a time window that is unknown a priori, as time available is dictated by the arrival of the next object in the stream.

is assumed that outlier detection is concerned with the computation of an outlier score that expresses the degree of outlierness. This is without loss of generality, as algorithms that separate outliers and inliers in a binary fashion can be considered as special cases that compute just two distinct outlier scores.

Illustrated in Figure 14.2, as more objects arrive within shorter periods of time, decisions in outlier detection need to be taken in less time accordingly. Since the duration of a burst or the number of objects that arrive within a certain time window is generally not bounded, simply buffering objects until the stream slows down is not an option. If the buffer size is exceeded, objects are lost and detection accuracy drops unexpectedly and unpredictably. On the other hand, if the stream speed is slow, an ideal outlier detection algorithm should be capable of making use of this time. The accuracy of outlier detection should improve as more time is available, which leads to the definition of the anytime outlier problem.

Definition 14.1 ***Anytime outlier detection.*** *Given a data stream of data objects o_i arriving at a priori unknown inter-arrival rate, the anytime outlier detection problem is to compute an outlier score $s(o_i)$ in the time t_a between the arrival of o_i and its successor o_{i+1}. The larger t_a, the more accurate the outlier score $s(o_i)$ should be.*

The evaluation of the accuracy of outlier scores $s(o_i)$ is not straightforward. If synthetic or manually labeled data is available for empirical studies, this constitutes a ground truth for accuracy assessment. In practice, however, such ground truth is typically not available, and domain experts need

to judge the quality of the outlier scores that are assigned to objects in the stream. This issue is not specific to streaming environments, but affects outlier detection research in general.

14.3.2 Outlier detection using a cluster hierarchy

The general concept underlying the proposed anytime outlier detection approach is discussed in the following. As outlined in the problem definition above, there are two main requirements: Being capable of computing outlier scores even within very short time intervals and being able to improve the accuracy or reliability of the outlier score computed.

Clustering methods have been successfully used to identify prevailing patterns of the data that serve as an input to the actual outlier detection. In order to meet the requirements of fast initial response and improvement over time, the AnyOut algorithm employs the ClusTree as a hierarchical clustering method. Clusters at upper levels of the tree hierarchy subsume the more fine grained information at lower levels of the tree. The hierarchy of the tree hence provides a natural organization of the clusters that can be incrementally accessed in order to refine the outlier score of the object in question. Initially, the object is compared only to the root node that describes the data distribution using few clusters. This comparison can be performed efficiently; hence, the first requirement is met. Since more detailed information on the data distribution is available at lower levels of the tree, the reliability of the outlier scores is typically improved. This aspect is empirically studied in the experiments in Section 14.4.

Assessing the degree of outlierness in the AnyOut approach is based on the degree of accordance of the object with the closest cluster feature at the current level of resolution. Whenever a new object o_{i+1} arrives in the stream, this interrupts the descent down the tree with the current object o_i. The object o_i is then compared to the cluster feature to assess the outlier degree. The *mean outlier score* computes the degree of outlierness of an object o_i as the extent of deviation of o_i from the mean of the closest cluster feature. Since AnyOut operates directly on the ClusTree, the closest cluster feature

is simply the one that is reached through tree traversal until the point of interruption by the next data object in the stream.

Definition 14.2 ***Mean outlier score.*** *For any data object o_i, the mean outlier score $s_m(o_i)$ is defined as $s_m(o_i) := \mathbf{d}(o_i, \mu_{e_s})$, where μ_{e_s} is the mean of entry e_s in the ClusTree that o_i is inserted into when the next object o_{i+1} of the data stream arrives and $\mathbf{d}$ is the Euclidean distance.*

The mean outlier score thus assesses the deviation of the current object from the mean of the data distribution in the cluster feature of the current tree entry. By interpreting the cluster features as parameters of a Gaussian distribution of the data objects in the corresponding subtree, a second outlier score based on the probability density of the object can be defined.

Definition 14.3 ***Density outlier score.*** *For any data object o_i, the density outlier score $s_d(o_i)$ is defined as $s_d(o_i) := g(o_i, \mu_{e_s}, \sigma_{e_i})$, where e_s is the entry in the ClusTree that o_i is inserted into when the next object o_{i+1} of the data stream arrives and g is a Gaussian distribution according to Definition 2.3.*

Both the mean outlier score and the density outlier score reflect the degree of outlierness of the object at the point of interruption. In both cases, the data distribution of the closest cluster is the basis for the score computation. They differ in the way the cluster feature is interpreted: the mean outlier score only takes the center of mass of the data into account, whereas the density outlier score takes the overall data distribution into account, assuming a Gaussian distribution.

14.4 Experiments

Before the performance of the AnyOut algorithm is evaluated, the employed quality measures are described in Section 14.4.1. The quality of the approach is tested using the proposed outlier scores on the individual levels of the ClusTree. Besides the incremental insertion as in the ClusTree algorithm (cf. Chapter 11) the EM top down bulk loading discussed in Chapter 6 is

employed for constructing the tree. The results for the anytime performance as well as a comparison between incremental insertion and bulk loading are presented in Section 14.4.3. Section 14.4.4 contains the results achieved by AnyOut on constant streams.

14.4.1 Setup

In the experiments real world data sets of different characteristics from the UCI machine learning repository [HB99] were used in a four fold cross validation. One class was left out in the training set to generate outliers: the objects from the left out class which are contained in the test set are the outliers $\mathcal{O}$. The AnyOut algorithm is evaluated after all objects are processed yielding a complete ranking of all objects with respect to their assigned outlier score in descending order. $pos(o)$ gives the position of an object o in the ranking; the smaller $pos(o)$, the more likely o is an outlier. The ground truth information whether an object o was an outlier ($o \in \mathcal{O}$) or not is used to compute the quality measures, starting off with the position score

$$posScore = \sum_{o \in \mathcal{O}} pos(o)$$

The smaller the value for $posScore$, the better the quality of the ranking, since then the true outliers are on top of the ranking.

The ranking and the ground truth are further used to compute the ROC curve for the results, which plots the true positive rate (TPR) over the false positive rate for each prefix of the ranking. From the ROC plots the AUC value (area under the ROC curve [Bra97]) is derived as a standard measure. Finally two further standard measures for ranking quality are computed, namely the Spearman ranking coefficient [Spe04] and Kendall's Tau [Ken38]. In the plots generally the average performance values over all classes are reported and detailed results per class are shown for selected settings.

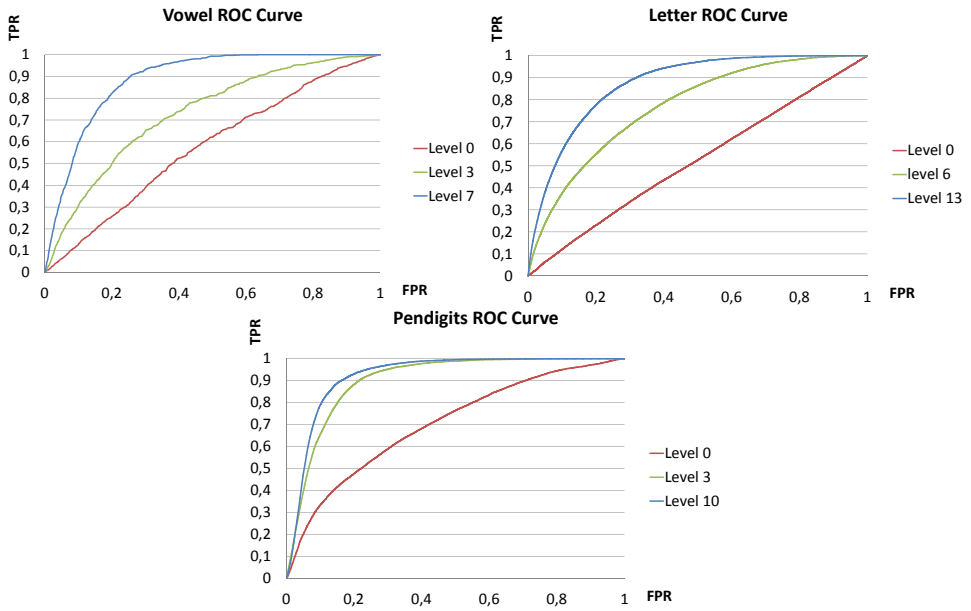


Figure 14.3: ROC curves for the *vowel*, *letter* and *pendigits* data sets.

14.4.2 Level analysis

To analyze the individual levels of the resulting tree structures in the AnyOut algorithm one ranking for each level of the tree is created and the measures introduced in Section 14.4.1 are computed. Figure 14.3 shows the ROC plots for the *vowel*, *pendigits* and *letter* data sets. Each plot contains a ROC curve for the root level, the leaf level and a level in the middle. The tree structures in this experiment were build using incremental insertion and the employed outlier score was the mean outlier score. What is clearly visible from the results on all data sets, is that the basic principle of the AnyOut algorithm is effective; the quality of the results improves if more time is available (deeper levels are reachable).

Similar observations can be made using the other quality measures. Figure 14.4 shows the results for Spearman's ranking coefficient, AUC and Kendalls Tau. The two plots show the mean outlier score and the density outlier score on the *vowel* data set using bulk loading to construct the tree. (Incremental insertion and bulk loading are compared in the next section.) The density outlier score yields slightly worse rankings than the mean outlier

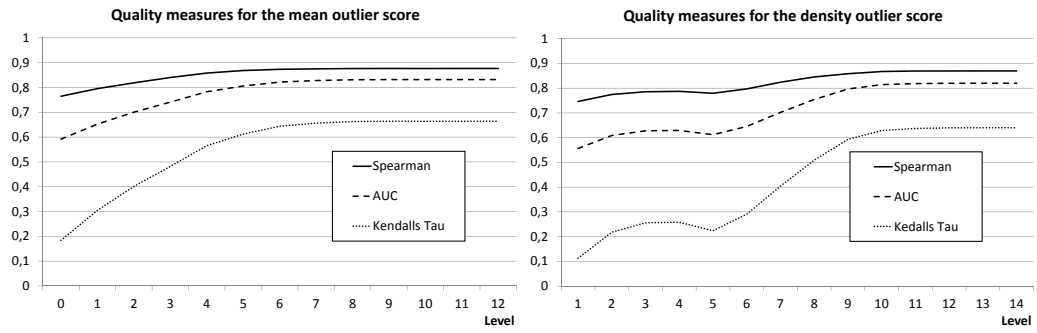


Figure 14.4: Quality Measures for mean and density outlier scores on *vowel*.

score. Moreover, the mean outlier score yields a constant quality improvement, whereas the density outlier score shows a slight reduction in ranking quality on intermediate levels. The mean outlier score showed better results throughout the data sets and is therefore employed in the following.

Figure 14.5 shows the results for the $posScore$ measure over the individual levels on the *pendigits* data set. Besides the average value, the values for each class that was left out are shown. While some classes are obviously easier to separate, the AnyOut principle shows its effectiveness in all cases; the more time is available to descend the tree structure, the better the resulting ranking reflects the true outliers.

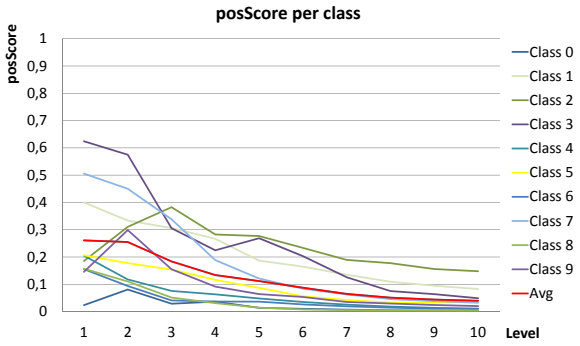


Figure 14.5: posScore per class and the averaged posScore for the *pendigits* data set.

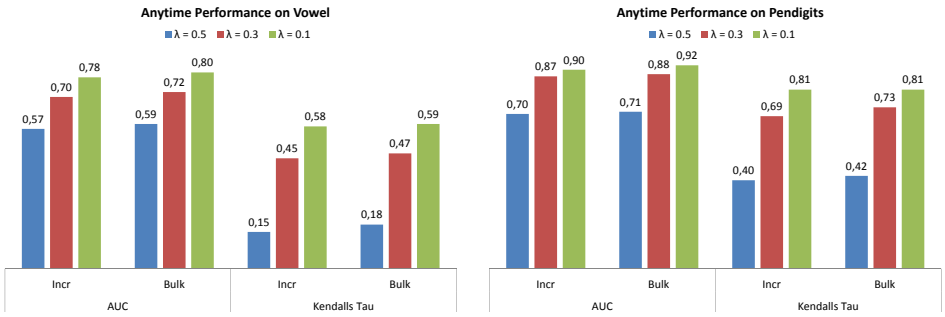


Figure 14.6: Results on anytime streams.

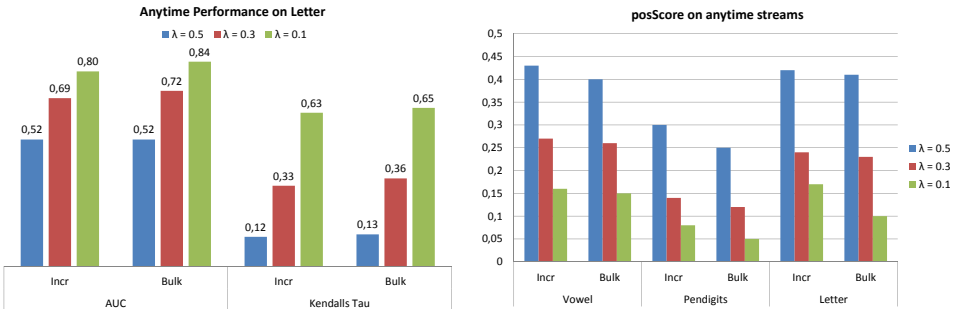


Figure 14.7: Results on anytime streams.

14.4.3 Anytime streams

In the anytime performance evaluation of AnyOut a unit corresponds to one level of the tree structure. Figures 14.6 and 14.7 show the results for Poisson streams (cf. Def. 4.8, p. 94) using the *vowel*, *pendigits* and *letter* data sets. In these experiments the performance of the incremental insertion is compared to the proposed bulk loading using the EM algorithm. In each group of bars the values for three different λ values are shown, while a smaller value corresponds to a slower stream with more expected time per object.

Starting with the *vowel* data set in Figure 14.6 (left) we see for the AUC measure and for Kendall's Tau, that both the incremental insertion and the bulk loading yield better qualities on slower stream. This is in line with the results from the level analysis in Section 14.4.2. Comparing the two tree construction methods we observe that on the one hand the bulk loading yields better results for any speed on both measures, but on the other hand the advantage is smaller than we expected.

Figure 14.8: Outlier Positions.

The results on *pendigits* (Figure 14.6 right) and *letter* (Figure 14.7 left) confirm both of the above findings. With a higher expected time per object (smaller λ value) the AnyOut algorithm shows its effectiveness and produces better rankings. The difference between incremental insertion and bulk loading performance are rather small but constant. To complete the analysis on anytime streams the $posScore$ values for all data sets are shown in Figure 14.7 (right). Summarizing the Evaluation we can conclude that the proposed AnyOut method is effective for anytime outlier detection.

14.4.4 Anytime outlier detection on constant streams

To test the performance of the AnyOut algorithm on streams with constant data rates a confidence measure for the current outlier score of an object is required (cf. Chapter 9). To this end the positions of the outliers in the ranking returned by the AnyOut algorithm are examined. Figure 14.8 shows the results for the vowel data set (middle level) using the mean outlier score. Optimally all outliers (green bars) would be at the far left. Obviously the objects with the highest outlier score are false alarms. Similar distributions

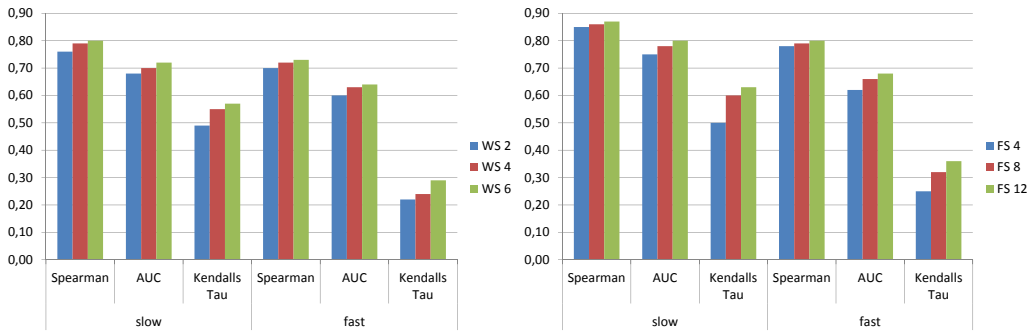


Figure 14.9: AnyOut using the window approach (left) and fifo approach (right) on constant data streams.

are observable for the other data sets. Therefore

$$conf(o) = 1 - e^{-s(o)}$$

is employed as a simple and straightforward confidence measure in the following, where $s(o)$ is the current outlier score of object o. Intuitively, we hence check for the putative outliers (the highest ranked objects) whether they are really outliers by giving them more computation time.

Figure 14.9 shows the corresponding results for the vowel data set using the window approach and fifo approach (cf. Chapter 9) respectively using three different window sizes (WS 2, 4 and 6) and the analogue fifo sizes (FS 4, 8 and 12). As in Section 14.4, four fold cross validation was used and the quality measures are based on the final ranking of all objects. The results from Figure 14.9 (left) show that the performance of the AnyOut algorithm improves with larger window sizes for slow and fast streams. The results using the fifo approach are even slightly better (cf. Figure 14.9 right), which is attributable to the greater flexibility according to the comparably larger set of objects. This is in line with the results from Chapter 9 for anytime classification on constant streams. The simple confidence measure proposed above already shows the effectiveness of AnyOut on constant streams. This motivates further research as in the case of anytime classification.

14.5 Conclusion

The anytime outlier detection problem for varying and constant data streams was introduced and studied in this chapter. A first algorithm called AnyOut was proposed and its structure and performance was analyzed on both types of streams. The algorithm is based on the methods proposed in the previous chapters and constitutes an application example for the ClusTree. The promising results of AnyOut motivate further research in both anytime clustering and anytime outlier detection. A simple and straightforward confidence measure was employed for performance improvement on constant streams. More sophisticated confidence measures can be investigated; using the variance of previous outlier scores for an object is one example. Also, anytime outlier algorithms can be developed using other paradigms such as density based approaches.

Chapter 15

MOA and CMM

[*] [†] Working on both classification (cf. Part II) and clustering (cf. Part III) reveals two very strong advantages of the former over latter when it comes to experimental evaluation: the chance of being provided with a ground truth on real world data sets and, given the ground truth, an undisputed objective evaluation measure (accuracy). The burden of finding data sets for the evaluation of clustering algorithms increases in the stream mining scenario, since a *gold standard clustering* for changing distributions in real world streaming data is hardly ever available. Therefore, evaluating and, especially, comparing stream clustering algorithms in a fair and meaningful way is a challenging task. The MOA framework presented in Section 15.1 constitutes a step towards easier and more repeatable evaluation and comparison of stream clustering algorithms. The matter of an appropriate evaluation measure for clustering evolving data streams is discussed in Section 15.2, where a short introduction to the cluster mapping measure (CMM) is provided.

[*]An Article on MOA has been published in the Journal of Machine Learning Research (JMLR) [BHP+10a] and MOA has been presented at different international conferences including ACM SIGKDD 2010 and IEEE ICDM 2010 [KKJ+10a, KKJ+10b, BHP+10b].

[†]A research paper describing the CMM has been published in the Proceedings of the 17th ACM International Conference on Knowledge discovery and Data Mining (KDD 2011) [KKJ+11].

15.1 The MOA Framework

In traditional data mining scenarios, evaluation frameworks were introduced to cope with the comparison issue. One of these frameworks is the well-known WEKA Data Mining Software that supports adding new algorithms and evaluation measures in a plug-and-play fashion [HFH+09, MAK+08, MAG+09b]. As data stream mining is a relatively new field, the evaluation practices are not nearly as well researched and established as they are in the traditional batch setting.

Massive **O**nline **A**nalysis (MOA) [BHKP10] is a framework for stream mining evaluation that builds on the work in WEKA. MOA contains state-of-the-art algorithms and measures for both stream classification and stream clustering and permits evaluation of data stream mining algorithms on large streams and under explicit memory limits. The main contributions and benefits of the MOA framework are:

- Analysis and comparison both for different approaches (new and state-of-the-art algorithms) and for different (large) streaming setting
- Creation and usage of benchmark settings for comparable and repeatable evaluation of stream mining algorithms
- Open source framework that is easily extensible for data feeds, algorithms and evaluation measures

In the following first the general architecture of MOA is introduced before classification and clustering on evolving data streams using MOA is described.

15.1.1 System architecture

A simplified system architecture is illustrated in Figure 15.1. It shows at the same time the work flow of MOA and its extension points, since all aspects follow the same principle. First a data feed is chosen, then a learning algorithm is configured (a stream classification or stream clustering algorithm)

and finally an evaluation method is chosen to analyze the desired scenario. The choice of streams, algorithms and especially evaluation methods differs between the classification and clustering parts and is therefore described separately in the following sections. For both tasks, users can extend the framework in all three aspects to add novel data generators, algorithms or evaluation measures. To run experiments using MOA users can chose between the command line or a graphical user interface.

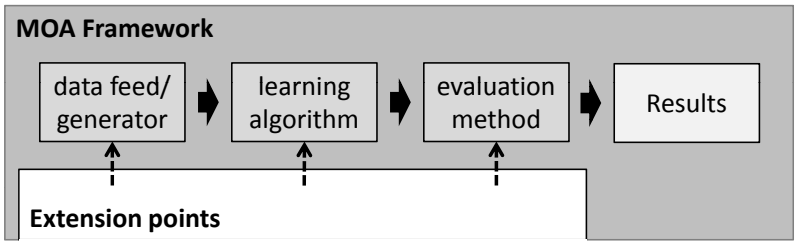


Figure 15.1: Architecture, extension points and work flow of MOA.

15.1.2 Classification

For the evaluation of stream classification algorithms MOA contains the following components.

Data stream generators for stream classification. MOA contains the data generators most commonly found in the literature. Streams can be built using generators, reading ARFF files, joining several streams, or filtering streams. They allow for the simulation of a potentially infinite sequence of data. The following generators are available in MOA for stream classification: SEA Concepts Generator [SK01], STAGGER Concepts Generator [SG86], Rotating Hyperplane [HSD01], LED Generator [BFSO84, HB99] and Function Generator [AGI+92].

Considering data streams as data generated from pure distributions, MOA models a concept drift event as a weighted combination of two pure distributions that characterizes the target concepts before and after the drift. Within the framework, it is possible to define the probability that instances of the

stream belong to the new concept after the drift. It uses the sigmoid function, as an elegant and practical solution [BHPG09, BHP+09].

Classifiers. For stream classification MOA contains a naïve Bayes classifier and the following variants of Hoeffding trees (cf. Chapter 3): Hoeffding Tree [DH00], Hoeffding Option Trees [PHK07], `ADWIN`[BG07], Bagging using `ADWIN`[BHP+09] and Adaptive-Size Hoeffding Trees [BHP+09].

Evaluation methods for stream classification. The evaluation procedure of a learning algorithm determines which examples are used for training the algorithm, and which are used to test the model output by the algorithm. When considering what procedure to use in the data stream setting, one of the unique concerns is how to build a picture of accuracy over time. Two main approaches arise:

- **Holdout**: When traditional batch learning reaches a scale where cross-validation is too time consuming, it is often accepted to instead measure performance on a single holdout set. This is most useful when the division between train and test sets has been predefined, so that results from different studies can be directly compared.

- **Interleaved Test-Then-Train or Prequential**: Each individual example can be used to test the model before it is used for training, and from this the accuracy can be incrementally updated. When intentionally performed in this order, the model is always being tested on examples it has not seen. This scheme has the advantage that no holdout set is needed for testing, making maximum use of the available data. It also ensures a smooth plot of accuracy over time, as each individual example will become increasingly less significant to the overall average [GSR09].

MOA is easy to use and extend. A simple approach to writing a new classifier is to extend `moa.classifiers.AbstractClassifier`, which will take care of certain details to ease the task.

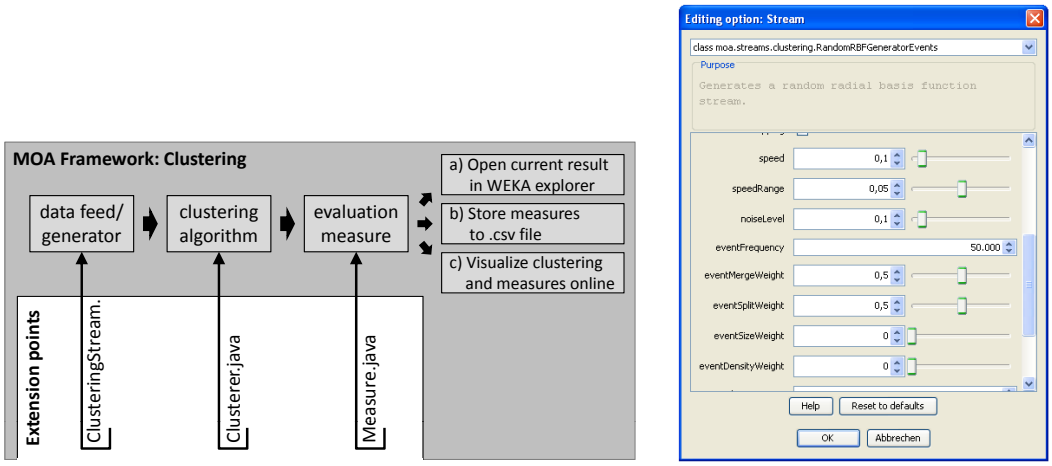


Figure 15.2: Left: Extension points and work flow of the MOA stream clustering framework. Right: Option dialog for the RBF data generator (by storing and loading settings benchmark streaming, data sets can be shared for repeatability and comparison).

15.1.3 Clustering

The stream clustering component of MOA has the following main features:

- data generators for stream clustering on evolving streams (including events like novelty, merge, etc. (cf. Chapter 3)),
- a set of state-of-the-art stream clustering algorithms,
- evaluation measures for stream clustering,
- visualization tools for analyzing results and comparing different settings.

The left part of figure 15.2 shows the extension points of MOA stream clustering and illustrates the architecture as well as the usage of the clustering component. First a data feed is chosen and configured, then a stream clustering algorithm and its settings are fixed, then a set of evaluation measures is selected and finally the experiment is run to obtain and analyze the result. The four aspects are detailed in the following subsections.

Data feeds and data generators

For stream clustering a new RBF data generator was added that support the simulation of cluster evolution events such as merging or disappearing of clusters.

The right part of figure 15.2 shows a screen shot of the configuration dialog for the RBF data generator with events. Generally the dimensionality, number and size of clusters can be set as well as the drift speed, decay horizon (aging) and noise rate etc. Events constitute changes in the underlying data model such as growing of clusters, merging of clusters or creation of new clusters [SNTS06]. Using the event frequency and the individual event weights, one can study the behavior and performance of different approaches on various settings. Finally, the settings for the data generators can be stored and loaded, which offers the opportunity of sharing settings and providing benchmark streaming data sets for repeatability and comparison. New data feeds and generators can be added to the MOA framework by implementing the ClusteringStream interface (further description and source code can be found on the MOA website `http://moa.cs.waikato.ac.nz/`).

Stream clustering algorithms

The MOA framework contains several stream clustering methods including StreamKM++ [ALM$^+$10], CluStream [AHWY03], ClusTree [KABS09], DenStream [CEQZ06], D-Stream [CT07] and CobWeb [Fis87]. As for stream classification, the set of algorithms is extensible through classes that implement the interface `Clusterer.java`. These are added to the framework via reflections on start up. The three main methods of this interface are

- `void resetLearningImpl()`: a method for initializing a clusterer learner
- `void trainOnInstanceImpl(Instance)`: a method to train a new instance
- `Clustering getClusteringResult()`: a method to obtain the current clustering result for evaluation or visualization

Internal measures	External measures
Gamma [BH75]	Rand statistic [Ran71]
C Index [HL76]	Jaccard coefficient [FM83]
Point-Biserial [Mil80]	Folkes and Mallow Index [FM83]
Log Likelihood [Har75]	Hubert Γ statistics [HA85]
Dunn's Index [Dun74]	Minkowski score [BSH+07]
Tau [Roh74]	Purity [ZK04]
Tau A [HL76]	van Dongen criterion [VD00]
Tau C [HL76]	V-measure [RH07]
Somer's Gamma [HL76]	Completeness [RH07]
Ratio of Repetition [HL76]	Homogeneity [RH07]
Modified Ratio of Repetition [HL76]	Variation of information [Mei05]
Adjusted Ratio of Clustering [HL76]	Mutual information [CT06]
Fagan's Index [HL76]	Class-based entropy [SZ08]
Deviation Index [HL76]	Cluster-based entropy [ZK04]
Z-Score Index [HL76]	Precision [vR79]
D Index [HL76]	Recall [vR79]
Silhouette coefficient [KR90]	F-measure [vR79]

Table 15.1: Internal and external clustering evaluation measures.

Stream clustering evaluation measures

For cluster evaluation various measures have been developed and proposed over the last decades. A common classification of these measures is the separation in so called internal measures and external measures. Internal measures only consider the cluster properties such as distances between points within one cluster or between two different clusters. External evaluation measures compare a given clusterings to a ground truth. Table 15.1 shows a selection of popular measures from the literature. MOA contains an extensible set of both internal and external measures that can be applied to both micro and macro clusterings. Comparative studies of clustering evaluation measures can be found in [Mil80, BSH+07, SZ08, WXC09], an outlook on evaluation of stream clustering algorithms is contained in Section 15.2.

To extend the available collection with additional or novel evaluation measures the Measure interface must be implemented. The main methods are:

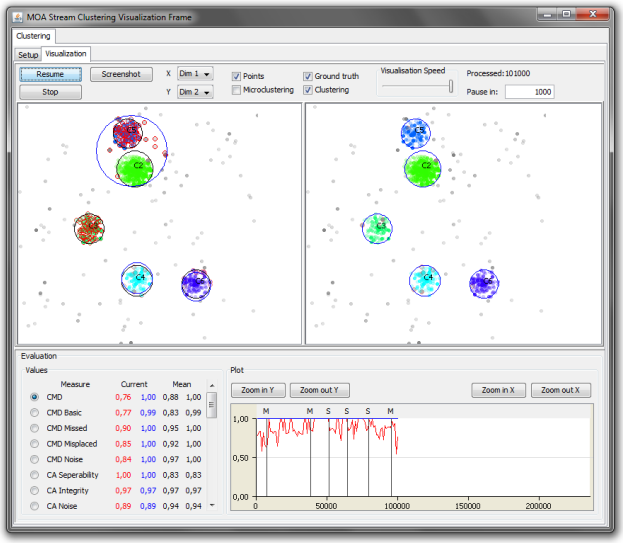


Figure 15.3: Visualization tab of the clustering MOA graphical user interface.

- `void evaluateClustering(Clustering clustering, Clustering trueClustering)`: uses the implemented measure to evaluate the given clustering w.r.t. to the provided ground truth.
- `double getLastValue()`: a method that outputs the last result of the evaluation measure.
- `double getMaxValue()`, `getMinValue()`, `getMean()`: methods that provide more statistics about the measure's distribution.

Analysis and visualization

After the evaluation process is started, several options for analyzing the outputs are given: a) the stream can be stopped and the current (micro) clustering result can be passed as a data set to the WEKA explorer for further analysis or mining; b) the evaluation measures, which are taken at configurable time intervals, can be stored as a .csv file to obtain graphs and charts offline using a program of choice; c) last but not least both the clustering results and the corresponding measures can be visualized online within the MOA framework. In the graphical interface MOA allows the simultaneous configuration and evaluation of two different setups for direct comparison of two different algorithms on the same stream.

The visualization component allows to visualize the stream as well as the clustering results, choose dimensions for multi dimensional settings, and compare experiments with different settings in parallel. Figure 15.3 shows a screen shot of the visualization tab. For this screen shot two different settings of the CluStream algorithm [AHWY03] were compared on the same stream setting (including merge/split events every 50000 examples) and four measures were chosen for online evaluation (F1, Precision, Recall, and SSQ). The upper part of the GUI offers options to pause and resume the stream, adjust the visualization speed, choose the dimensions for x and y as well as the components to be displayed (points, micro- and macro clustering and ground truth). The lower part of the GUI displays the measured values for both settings as numbers (left) and the currently selected measure as a plot over the arrived examples (right, F1 measure in this example). For the shown setting one can see a clear drop in the performance after the split event at roughly 160000 examples (event details are shown when choosing the corresponding vertical line in the plot). While this holds for both settings, the left configuration (red, CluStream with 100 micro clusters) is constantly outperformed by the right configuration (blue, CluStream with 20 micro clusters). A video containing an online demo of the system can be found at the MOA website along with more screen shots and explanations.

15.1.4 Conclusion

The MOA framework provides a set of data generators, algorithms and evaluation measures for stream data mining. Practitioners can benefit from this by comparing several algorithms in real world scenarios and choosing the best fitting solution. For researchers the framework yields insights into advantages and disadvantages of different approaches and allows the creation of benchmark streaming data sets through stored, shared and repeatable settings for the data feeds. The sources are publicly available and are released under the GNU GPL license. Although the current focus in MOA is on classification and clustering, envisioned extensions to the framework include regression and frequent pattern learning [Bif10].

15.2 Evaluation Measures for Stream Clustering

The experience from implementing and using the stream clustering component of MOA showed two major disadvantages of existing evaluation measures: First, none can properly handle the peculiarities of evolving data streams such as overlapping due to merging or drifting clusters or noisy data streams; As a consequence the measures cannot effectively reflect the occurring errors. Second, the vast majority of evaluation measures achieve suboptimal results even if the ground truth clustering is tested. The Cluster Mapping Measure (CMM) proposed in [KKJ+11] overcomes these shortcomings and enables effective evaluation of clustering results on evolving streams. This section provides a brief overview of the main idea behind CMM.

An evaluation measure for clustering on evolving data streams should take the following scenarios and circumstances into account:

1. **Aging / decay** Dealing with this property is probably the simplest task, because faults caused by a clustering algorithm can be weighted by the influence (age) of the corresponding points.

2. **Missed points** Moving clusters yield errors for missed points. These errors should reflect the seriousness, e.g. how close the point is to its actual cluster.

3. **Misplaced points** Evolution, merging, and splitting of clusters yields overlapping clusters and thereby easily misplaced points. A measure that punishes these misplaced points equally to misplaced points laying outside of any overlapping region, does not account for the special circumstances of evolving streams.

4. **Noise** Including noise in a found cluster is often inevitable in the model of the clustering algorithm and should be accounted for by an effective measure.

Summarizing these properties, three fault cases can be identified that have to be considered in depth, namely missed points, misplaced points, and

noise inclusion. The penalty for such errors of stream clustering algorithms should reflect their seriousness and take the age of the points as well as the clustering model into account. CMM is a normalized sum of the penalties for occurring errors that accounts for all aspects mentioned above. Two important prerequisites for the computation are a notion of how well an object fits into a cluster and a mapping from found clusters to ground truth classes.

The results of the experimental evaluation in [KKJ+11] show that the cluster mapping measure can precisely reflect various error types in evolving data streams. In comparison with known and widely used internal and external measures the CMM outperformed the competing approaches on real and synthetic data using both cluster generators and existing stream clustering algorithms. The CMM is included in the open source MOA framework for future experimentation and evaluation of novel approaches.

Chapter 16

Future Work

The ClusTree is the first anytime clustering algorithm for streaming data. As for the Bayes tree in Part II, a subspace variant of the ClusTree is an interesting research topic. The effects of different initializations can be evaluated, for example using a top down approach similar to the bulk loading proposed for the Bayes tree. For descending the tree alternative priority measures can be explored such as the Kullback Leibler divergence or the probability density of the object w.r.t. to the entry. Moreover, descending with the hitchhiker instead of the insertion object may be beneficial if the former fits the corresponding entry better. Finally, different ways to devise new anytime clustering algorithms can be investigated.

Anytime outlier algorithms can be developed using other paradigms such as density based outlier detection. For their application on constant data streams more sophisticated confidence measures can be investigated; using the variance of previous outlier scores for an object is one example.

The next step within MOA w.r.t. stream clustering is the inclusion of anytime clustering and corresponding evaluation mechanisms. Together with the proposed cluster mapping measure MOA offers a platform for an extensive evaluation study of stream clustering algorithms on evolving data. Another goal of the MOA project is to establish benchmark settings and collect real world benchmark data with drift, novelty and noise.

Part IV

Summary and Outlook

Part I. The introduction of this thesis provides a strong motivation for anytime algorithms. Many applications are discussed in Chapter 1 and the benefits of anytime algorithms are laid out. Chapters 2 and 3 provide background and related work on the KDD process and stream data mining.

Part II. Chapter 4 introduces the Bayes tree as a new anytime classifier for Bayesian classification on continuous attributes. The Bayes tree constitutes a hierarchy of mixture densities that represent kernel estimators at successively coarser levels. The proposed probability density queries adapt the employed mixtures efficiently to the individual object to be classified. Together with novel classification improvement strategies this allows for very effective classification at any point of interruption.

The Bayes tree uses an incremental insertion procedure and builds separate hierarchies for each class label. A different approach is followed by the MC-tree described in Chapter 5. Starting from the initial idea of combining several classes in a single tree a novel way of constructing the hierarchical models is investigated through top-down clustering using the EM algorithm. While it turns out that separating the classes remains advisable, the EM construction shows very good results.

In Chapter 6 further alternative construction methods are investigated for hierarchical mixture models in the Bayes tree. Experimental results show that the EMTopDown bulk load constantly outperforms other approaches and improves the accuracy by up to 13%. Surprisingly two proposed statistical approaches were outperformed by existing R-tree bulk loadings based on space filling curves. Further analysis attributed this shortcoming to a structural property of the resulting Bayes trees. The results of the analysis are in line with the classification results found in the experiments confirming the superior performance of the EMTopDown bulk loading in terms of anytime classification accuracy.

Chapter 7 concludes the in depth investigation of the Bayes tree. For construction, parameter optimization and decision design related concepts are analyzed and transferred to the Bayes tree to improve the corresponding process. A thorough experimental evaluation of the single improvements

as well as the combined approaches shows great potential of the concept transfer method. The improved version of the Bayes tree that results from Chapter 7 shows near perfect results on all tested data sets and constitutes the final version of the proposed Bayesian anytime classifier.

An application of anytime classification using the Bayes tree is presented in Chapter 8. The HeathNet scenario is introduced and the integration of the Bayes tree as well as its benefits are described. A proof of concept is provided by a prototype that has been developed within a UMIC research project at RWTH Aachen University.

Chapter 9 introduces two meta-approaches that harness the strengths of anytime algorithms for streams with constant data rates. The goal was to improve the quality of the result w.r.t. traditional budget approaches, which are used in an abundance of stream mining applications. Using anytime classification as an example application experimental results show for SVM, Bayes and nearest neighbor classifiers that both approaches improve the classification accuracy for slow and fast data streams. The results confirm the theoretic model that was introduced in that chapter and show the effectiveness of the proposed approaches. The simple yet effective idea can be employed for any anytime algorithm along with a quality measure and motivates further research in classification confidence measures and anytime algorithms.

Part III. The first anytime stream clustering algorithm is presented in Chapter 11. The proposed ClusTree algorithm self-adapts to varying stream speed and provides a novel solution for interruption of the insertion process that can be easily resumed at any later point in time. For very fast streams, aggregates of similar objects allow insertion of groups instead of single objects for even faster processing. In comparison to recent approaches it is shown that the ClusTree can maintain the same amount of micro clusters at a stream speed that is faster by orders of magnitude and that for an equal stream speed the obtained granularity is exponential w.r.t. competing approaches.

In Chapter 12 alternative descent strategies for the ClusTree are proposed that improve the resulting clustering on slower streams as long as time permits. The *iterative depth first* descent turns out as an excellent alternative

insertion strategy, since it starts with the same high performance as the original strategy, has very low runtime ($O(\log^2(n))$) and yet improves the initial solutions on all tested settings. Moreover, it finally reaches comparable high quality results compared to all other approaches tested.

The LiarTree presented in Chapter 13 improves the ClusTree w.r.t. overlapping of inner entries and incorporates explicit noise handling for robust anytime stream clustering. It allows for the transition from local noise buffers to new entries (micro clusters) and grows novel subtrees top down using its liar concept. Experimental evaluation shows for various data stream scenarios that the LiarTree outperforms competing approaches in the presence of noise and evolving data, proving its novel concepts to be effective.

Chapter 14 discusses an application of the ClusTree for anytime outlier detection. The performance of the proposed AnyOut algorithm is analyzed on both varying and constant data streams. The promising results of AnyOut motivate further research in both anytime clustering and anytime outlier detection.

Finally, results of ongoing stream clustering research are presented in Chapter 15. The MOA open source framework for stream data mining is introduced which contains evaluation methods for both stream classification and stream clustering. An extensible set of stream generators, mining algorithms and evaluation measures is contained in MOA and publicly available on the project homepage. In Section 15.2 the cluster mapping measure (CMM) is announced as a novel evaluation measure for clustering on evolving data streams. The CMM is the first measure that takes the special requirements of the streaming scenario into account.

Main contributions. The proposed methods enable for the first time anytime Bayesian classification on continuous attributes and thoroughly study possible heuristics and improvements. The Proposed meta approaches largely widen the application area for anytime algorithms and prove their effectiveness in previously uncommon terrain: constant data streams. The anytime principle is introduced for two additional mining tasks, namely clustering and outlier detection, and the first anytime stream clustering method is pre-

sented. Finally, many future research objectives are opened by the results of this thesis. First steps such as the MOA framework and the cluster mapping measure are documented, and further opportunities are discussed in Chapters 10 and 16 and the remainder of this section.

Outlook. Future research objectives specific to stream classification and stream clustering have been discussed in the last chapters of Parts II and III, respectively. Combining the results from both parts, a first option is to incorporate the decay from the ClusTree into the Bayes tree and evaluate the classification performance on changing distributions. Extending the proposed approaches to allow for semi-supervised learning or multi-label classification are interesting open topics.

The MOA framework constitutes work in progress. The framework can be used for the analysis of stream clustering algorithms and experimental comparison for newly developed algorithms; MOA provides a basis for thorough evaluation studies in the area of stream clustering. Many extensions of MOA can be thought of; including algorithms and evaluation methods for anytime classification and anytime mining in general are among the next steps. Further ongoing and open topics w.r.t the MOA framework are the integration of further mining tasks such as frequent pattern mining on evolving data streams.

Anytime algorithms can be further investigated for the tasks discussed in this thesis and beyond. Different approaches for clustering or outlier detection under the anytime paradigm can be developed, for example. Anytime algorithms can also be devised for related areas such as search and retrieval or relevance feedback systems.

Stream mining in general provides open research questions for performing more complex tasks on streams. Examples include subspace clustering on evolving data or graph mining on streams.

Collecting and creating benchmark data sets for stream mining, similar in spirit to the UCI repository, is an important step towards enabling more repeatable and more fair evaluation and comparison of proposed methods.

Part V

Appendices

Bibliography

[ABKS99] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 49–60, 1999.

[AF07] Fabrizio Angiulli and Fabio Fassetti. Detecting distance-based outliers in streams of data. In *Proceedings of the ACM Conference on Information and Knowledge Management, CIKM*, pages 811–820, 2007.

[Agg05] Charu C. Aggarwal. On abnormality detection in spuriously populated data streams. In *SIAM Data Mining*, 2005.

[Agg06] Charu C. Aggarwal. *Data Streams: Models and Algorithms*. Springer Verlag, 2006.

[AGGR98] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 94–105, 1998.

[AGI+92] Rakesh Agrawal, Sakti P. Ghosh, Tomasz Imielinski, Balakrishna R. Iyer, and Arun N. Swami. An interval classifier for database mining applications. In *Proceedings of the International Conference on Very Large Data Bases*, pages 560–573, 1992.

[AHWY03] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In *Proceedings of the International Conference on Very Large Data Bases*, pages 81–92, 2003.

[AHWY04] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for projected clustering of high dimensional data streams. In *Proceedings of the International Conference on Very Large Data Bases*, pages 852–863, 2004.

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993.

[AKAS08] Ira Assent, Ralph Krieger, Farzad Afschari, and Thomas Seidl. The ts-tree: Efficient time series search and retrieval. In *International Conference on Extending Data Base Technology*, pages 252–263, 2008.

[AKBS10] Ira Assent, Philipp Kranen, Corinna Baldauf, and Thomas Seidl. Detecting outliers on arbitrary data streams using anytime approaches. In *Proceedings of the International Workshop on Novel Data Stream Pattern Mining Techniques (StreamKDD 2010) in conjunction with 16th ACM SIGKDD*, pages 10–16, 2010.

[AKKS99] Mihael Ankerst, Gabi Kastenmüller, Hans-Peter Kriegel, and Thomas Seidl. Nearest neighbor classification in 3d protein databases. In *Proceedings of the International Conference on Intelligent Systems for Molecular Biology*, pages 34–43, 1999.

[AKMS08] Ira Assent, Ralph Krieger, Emmanuel Müller, and Thomas Seidl. Edsc: efficient density-based subspace clustering. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 1093–1102, 2008.

[AKS10] Ira Assent, Hardy Kremer, and Thomas Seidl. Speeding up complex video copy detection queries. In *International Conference on Database Systems for Advanced Applications*, pages 307–321. Springer, 2010.

[ALM+10] Marcel R. Ackermann, Christiane Lammersen, Marcus Märtens, Christoph Raupach, Christian Sohler, and Kamil Swierkot. Streamkm++: A clustering algorithm for data streams. In *Proceedings of the Workshop on Algorithm Engineering and Experiments*, pages 173–187, 2010.

[AN98] Jochen Alber and Rolf Niedermeier. On multi-dimensional hilbert indexings. In *Proceedings of the International Conference on Computing and Combinatorics*, pages 329–338, 1998.

[APW+99] Charu C. Aggarwal, Cecilia Magdalena Procopiuc, Joel L. Wolf, Philip S. Yu, and Jong Soo Park. Fast algorithms for projected clustering. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 61–72, 1999.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 487–499, 1994.

[AS95] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Proceedings of the International Conference on Data Engineering*, pages 3–14, 1995.

[AS96] Rakesh Agrawal and John C. Shafer. Parallel mining of association rules. *IEEE Transactions Knowledge Data Engineering*, 8(6):962–969, 1996.

[AS04] D. Andre and P. Stone. Physiological data modeling contest (ICML-2004): http://www.cs.utexas.edu/users/pstone/workshops/2004icml/, 2004.

[AV07] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035, 2007.

[AY00] Charu C. Aggarwal and Philip S. Yu. Finding generalized projected clusters in high dimensional spaces. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 70–81, 2000.

[BB01] Pierre Baldi and Søren Brunak. *Bioinformatics - the machine learning approach (2. ed.)*. MIT Press, 2001.

[BBD+02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the Symposium on Principles of Database Systems*, pages 1–16. ACM, 2002.

[BC00] Daniel Barbará and Ping Chen. Using the fractal dimension to cluster datasets. In *Proceedings of the ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 260–264, 2000.

[Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18:509–517, 1975.

[BEX02] Florian Beil, Martin Ester, and Xiaowei Xu. Frequent term-based text clustering. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 436–442, 2002.

[BEY98] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[BFOS84] Leo Breiman, Jerome Friedman, R. Olshen, and Charles Stone. *Classification and Regression Trees*. Wadsworth International Group, 1984.

[BFSO84] Leo Breiman, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. *Classification and Regression Trees*. Chapman & Hall, New York, 1984.

[BG07] Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *Proceedings of the SIAM International Conference on Data Mining*, 2007.

[BGH+97] Newton L. Bowers, Hans U. Gerber, James C. Hickman, Donald A. Jones, and Cecil J. Nesbitt. *Actuarial Mathematics*. Society of Actuaries, Itasca, IL, 1997.

[BH75] Frank B. Baker and Lawrence J. Hubert. Measuring the power of hierarchical cluster analysis. *Journal of the American Statistical Association (ASA)*, 70(349):31–38, 1975.

[BH90] Jack S. Breese and Eric Horvitz. Ideal reformulation of belief networks. In *Proceedings of the Annual Conference on Uncertainty in Artificial Intelligence*, pages 129–144, 1990.

[BH02] James C. Bezdek and Richard J. Hathaway. Vat: a tool for visual assessment of (cluster) tendency. In *Proceedings of the International Joint Conference on Neutral Networks*, volume 3, pages 2225–2230, 2002.

[BHKP10] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. MOA: Massive Online Analysis http://sourceforge.net/projects/moa-datastream/. *Journal of Machine Learning Research (JMLR)*, 2010.

[BHP+09] Albert Bifet, Geoffrey Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. New ensemble methods for

evolving data streams. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 139–148, 2009.

[BHP+10a] Albert Bifet, Geoffrey Holmes, Bernhard Pfahringer, Philipp Kranen, Hardy Kremer, Timm Jansen, and Thomas Seidl. MOA: Massive online analysis, a framework for stream classification and clustering. In *Journal of Machine Learning Research (JMLR)*, volume 11, pages 44–50, 2010.

[BHP+10b] Albert Bifet, Geoffrey Holmes, Bernhard Pfahringer, Philipp Kranen, Hardy Kremer, Timm Jansen, and Thomas Seidl. MOA: Massive online analysis, a framework for stream classification and clustering. In *Invited presentation at the International Workshop on Handling Concept Drift in Adaptive Information Systems, in conjunction with European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, 2010.

[BHPG09] Albert Bifet, Geoffrey Holmes, Bernhard Pfahringer, and Ricard Gavaldà. Improving adaptive bagging methods for evolving data streams. In *Asian Conference on Machine Learning*, pages 23–37, 2009.

[BHS08] Mirko Böttcher, Frank Höppner, and Myra Spiliopoulou. On exploiting the power of time in data mining. *SIGKDD Explorations*, 10(2):3–11, 2008.

[Bif10] Albert Bifet. *Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams*. Frontiers in Artificial Intelligence and Applications. Ios Pr Inc, 2010.

[BKK96] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The x-tree : An index structure for high-dimensional data. In *International Conference on Very Large Data Bases*, pages 28–39, 1996.

[BKNS00] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 93–104, 2000.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, 1990.

[BL94] Vic Barnett and Toby Lewis. *Outliers in Statistical Data*. Wiley, 3rd ed., 1994.

[BMH+05] Sanghamitra Bandyopadhyay, Ujjwal Maulik, Lawrence B. Holder, Diane J. Cook, Mohamed Gaber, Shonali Krishnaswamy, and Arkady Zaslavsky. On-board mining of data streams in sensor networks. In Lakhmi Jain and Xindong Wu, editors, *Advanced Methods for Knowledge Discovery from Complex Data*, Advanced Information and Knowledge Processing, pages 307–335. Springer London, 2005.

[Bod91] Mark S. Boddy. Anytime problem solving using dynamic programming. In *AAAI*, pages 738–743, 1991.

[BPS06] Christian Böhm, Alexey Pryakhin, and Matthias Schubert. The gauss-tree: Efficient object identification in databases of probabilistic feature vectors. In *Proceedings of the International Conference on Data Engineering*, page 9, 2006.

[Bra97] Andrew P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.

[Bre96] Leo Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[Bre01] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[BSH+07] Marcel Brun, Chao Sima, Jianping Hua, James Lowey, Brent Carroll, Edward Suh, and Edward R. Dougherty. Model-based evaluation of clustering validation measures. *Pattern Recognition*, 40(3):807–824, 2007.

[Bur98] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, 1998.

[CBB08] Leonardo Weiss Ferreira Chaves, Erik Buchmann, and Klemens Böhm. Tagmark: reliable estimations of rfid tags for business processes. In *International Conference on Knowledge Discovery and Data Mining*, pages 999–1007, 2008.

[CCH91] Yandong Cai, Nick Cercone, and Jiawei Han. Attribute-oriented induction in relational databases. In *Knowledge Discovery in Databases*, pages 213–228. AAAI/MIT Press, 1991.

[CDH+02] Yixin Chen, Guozhu Dong, Jiawei Han, Benjamin W. Wah, and Jianyong Wang. Multi-dimensional regression analysis of time-series data streams. In *International Conference on Very Large Data Bases*, pages 323–334, 2002.

[CEQZ06] Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. Density-based clustering over an evolving data stream with noise. In *Proceedings of the SIAM International Conference on Data Mining*, 2006.

[CFZ99] Chun Hung Cheng, Ada Wai-Chee Fu, and Yi Zhang. Entropy-based subspace clustering for mining numerical data. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 84–93, 1999.

[Che00] William Cheetham. Case-based reasoning with confidence. In *Advances in Case-Based Reasoning, (EWCBR)*, pages 15–25, 2000.

[CHOY08] Jia-Yu Chen, John R. Hershey, Peder A. Olsen, and Emmanuel Yashchin. Accelerated monte carlo for kullback-leibler divergence between gaussian mixture models. In *Proceedings of the IEEE International Conference on Acoustics*, pages 4553–4556, 2008.

[CI98] Chee-Yong Chan and Yannis E. Ioannidis. Bitmap index design and evaluation. In *International Conference on Management of Data*, pages 355–366. ACM, 1998.

[CM96] Chris Clifton and Don Marks. Security and privacy implications of data mining. In *Proceedings of the SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 15–20, 1996.

[Com79] Douglas Comer. Ubiquitous b-tree. *ACM Computing Surveys*, 11:121–137, 1979.

[CPZ97] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *International Conference on Very Large Data Bases*, pages 426–435, 1997.

[CT06] Thomas M. Cover and Joy A. Thomas. *Elements of information theory (2nd Edition)*. Wiley-Interscience, New York, NY, USA, 2006.

[CT07] Yixin Chen and Li Tu. Density-based clustering for real-time stream data. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 133–142, 2007.

[CZ06] Xin Chen and Chengcui Zhang. An interactive semantic video mining and retrieval platform–application in transportation surveillance video for incident detection. In *International Conference on Data Mining*, pages 129–138, 2006.

[CZSC10] Hui Cao, Yongluan Zhou, Lidan Shou, and Gang Chen. Attribute outlier detection over data streams. In *Database Systems for Advanced Applications*, pages 216–230, 2010.

[Das90] Belur V. Dasarathy. *Nearest neighbor (NN) norms: NN pattern classification techniques*. IEEE Computer Society Press, 1990.

[DB88] Thomas Dean and Mark S. Boddy. An analysis of time-dependent planning. In *AAAI*, pages 49–54, 1988.

[DCDZ05] Sarah Jane Delany, Padraig Cunningham, Dónal Doyle, and Anton Zamolotskikh. Generating estimates of classification confidence for a case-based spam filter. In *International Conference on Case-Based Reasoning*, pages 177–190, 2005.

[DCP08] Mark Dredze, Koby Crammer, and Fernando Pereira. Confidence-weighted linear classification. In *Proceedings of the International Conference on Machine Learning*, pages 264–271, 2008.

[DE84] William H. E. Day and Herbert Edelsbrunner. Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of Classification*, 1:7–24, 1984. 10.1007/BF01890115.

[DeC97] Dennis DeCoste. Mining multivariate time-series sensor data to discover behavior envelopes. In *International Conference on Knowledge Discovery and Data Mining*, pages 151–154, 1997.

[DeC02] Dennis DeCoste. Anytime interval-valued outputs for kernel machines: Fast support vector machine classification via dis-

tance geometry. In *Proceedings of the International Conference on Machine Learning*, pages 99–106, 2002.

[DeC03] Dennis DeCoste. Anytime query-tuned kernel machines via cholesky factorization. In *Proceedings of the SIAM International Conference on Data Mining*, 2003.

[DH00] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80, 2000.

[DHN10] Thomas Deselaers, Georg Heigold, and Hermann Ney. Object classification by fusing svms and gaussian mixtures. *Pattern Recognition*, 43(7):2476–2484, 2010.

[DHS01] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley, 2001.

[DIIM04] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computational Geometry*, pages 253–262, 2004.

[dL77] Jan de Leeuw. Applications of convex analysis to multidimensional scaling. In *Recent Developments in Statistics*, pages 133–146. North Holland Publishing Company, 1977.

[DLR77] Arthur P. Dempster, N. M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[DM03] Dennis DeCoste and Dominic Mazzoni. Fast query-optimized kernel machine classification via incremental approximate nearest support vectors. In *Proceedings of the International Conference on Machine Learning*, pages 115–122, 2003.

[Dun74] J.C. Dunn. Well separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4:95–104, 1974.

[dVCH10] Timothy de Vries, Sanjay Chawla, and Michael E. Houle. Finding local anomalies in very high dimensional space. In *Proceedings of the International Conference on Management of Data*, pages 128–137, 2010.

[Edw00] David M. Edwards. *Introduction to Graphical Modelling*. Springer Verlag, 2000.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[ELN+08] Manzoor Elahi, Kun Li, Wasif Nisar, Xinjie Lv, and Hongan Wang. Efficient clustering-based outlier detection algorithm for dynamic data stream. In *International Conference on Fuzzy Systems and Knowledge Discovery*, pages 298–304, 2008.

[EM11] Saher Esmeir and Shaul Markovitch. Anytime learning of anycost classifiers. *Machine Learning*, 82(3):445–473, 2011.

[EW95] Michael D. Escobar and Mike West. Bayesian density estimation and inference using mixtures. *Journal of the American Statistical Association*, 90(430):577–588, 1995.

[FA10] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[FG08] Conny Franke and Michael Gertz. Detection and exploration of outlier regions in sensor data streams. In *Workshops Proceedings of the IEEE International Conference on Data Mining*, pages 375–384, 2008.

[FGMP09] M. Julia Flores, José A. Gámez, Ana M. Martínez, and Jose Miguel Puerta. Gaode and haode: two proposals based on aode to deal with continuous variables. In *Proceedings of the International Conference on Machine Learning*, pages 40–47, 2009.

[FI93] Usama M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, pages 1022–1029, 1993.

[Fis87] Douglas H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172, 1987.

[FM83] E. B. Fowlkes and Colin L. Mallows. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78(383):553–569, 1983.

[FS96] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Internation Conference on Machine Learning*, pages 148–156, 1996.

[FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997.

[FZZ09] Andrew Foss, Osmar R. Zaïane, and Sandra Zilles. Unsupervised class separation of multivariate data through cumulative variance-based ranking. In *The IEEE International Conference on Data Mining*, pages 139–148, 2009.

[GD04] Daniel Grossman and Pedro Domingos. Learning bayesian network classifiers by maximizing conditional likelihood. In *Proceedings of the International Conference on Machine Learning,*, 2004.

[GG07] João Gama and Mohamed Gaber. *Learning from Data Streams – Processing Techniques in Sensor Networks*. Springer Verlag, 2007.

[GH07] Nizar Grira and Michael E. Houle. Best of both: a hybridized centroid-medoid clustering heuristic. In *Proceedings of the International Conference on Machine Learning*, pages 313–320, 2007.

[GKNN91] P. S. Gopalakrishnan, Dimitri Kanevsky, Arthur Nádas, and David Nahamoo. An inequality for rational functions with applications to some statistical estimation problems. *IEEE Transactions on Information Theory*, 37(1):107–113, 1991.

[GKS01] Johannes Gehrke, Flip Korn, and Divesh Srivastava. On computing correlated aggregates over continual data streams. In *Proceedings of the International Conference on Management of Data*, pages 13–24, 2001.

[GLF89] John H. Gennari, Pat Langley, and Douglas H. Fisher. Models of incremental concept formation. *Artificial Intelligence*, 40(1-3):11–61, 1989.

[GM03] Alexander G. Gray and Andrew W. Moore. Nonparametric density estimation: Toward computational tractability. In *Proceedings of the SIAM International Conference on Data Mining*, 2003.

[GMR04] João Gama, Pedro Medas, and Ricardo Rocha. Forest trees for on-line data. In *ACM Symposium on Applied Computing*, pages 632–636, 2004.

[GR04] Jacob Goldberger and Sam T. Roweis. Hierarchical clustering of a mixture model. In *Advances in Neural Information Processing Systems*, 2004.

[GSR09] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. Issues in evaluation of stream learning algorithms. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 329–338, 2009.

[Guh09] Sudipto Guha. Tight results for clustering and summarizing data streams. In *International Conference on Database Theory*, pages 268–275, 2009.

[Gut84] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD Conference*, pages 47–57, 1984.

[GZ96] Joshua Grass and Shlomo Zilberstein. Anytime algorithm development tools. *SIGART Bulletin*, 7(2):20–27, 1996.

[GZK05] Mohamed Medhat Gaber, Arkady B. Zaslavsky, and Shonali Krishnaswamy. Mining data streams: a review. *SIGMOD Record*, 34(2):18–26, 2005.

[HA85] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985.

[Har75] John Anthony Hartigan. *Clustering Algorithms*. John Wiley and Sons, New York, 1975.

[Haw80] Douglas M. Hawkins. *Identification of outliers*. Chapman and Hall New York, 1980.

[HB87] Stephen J. Hanson and David J. Burr. Minkowski backpropagation: Learning in connectionist models with non-euclidean error signals. In *Neural Information Processing Systems*. American Institute of Physics, 1987.

[HB99] Seth Hettich and Stephen D. Bay. The UCI KDD archive http://kdd.ics.uci.edu, 1999.

[HBH05] Jacalyn M. Huband, James C. Bezdek, and Richard J. Hathaway. bigvat: Visual assessment of cluster tendency for large data sets. *Pattern Recognition*, 38(11):1875–1886, 2005.

[HBH06] Richard J. Hathaway, James C. Bezdek, and Jacalyn M. Huband. Scalable visual assessment of cluster tendency for large data sets. *Pattern Recognition*, 39(7):1315–1324, 2006.

[HDY99] Jiawei Han, Guozhu Dong, and Yiwen Yin. Efficient mining of partial periodic patterns in time series database. In *Proceedings of the International Conference on Data Engineering*, pages 106–115, 1999.

[HFH+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[HK98] Alexander Hinneburg and Daniel A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 58–65, 1998.

[HK01] Jiawei Han and Michele Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.

[HK06] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques, Second Edition*. Morgan Kaufmann, 2006.

[HKK+10] Michael E. Houle, Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Can shared-neighbor distances defeat the curse of dimensionality? In *International Conference on Scientific and Statistical Database Management*, pages 482–500, 2010.

[HL76] Lawrence J. Hubert and Joel R. Levin. A general statistical framework for assessing categorical clustering in free recall. *Psychological Bulletin*, 83(6):1072–1080, 1976.

[HLZ02] Wynne Hsu, Mong-Li Lee, and Ji Zhang. Image mining: Trends and developments. *Journal of Intelligent Information Systems*, 19(1):7–23, 2002.

[Hoe63] Wassilij Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[Hou08] Michael E. Houle. The relevant-set correlation model for data clustering. In *Proceedings of the SIAM International Conference on Data Mining*, pages 775–786, 2008.

[HPY00] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1–12, 2000.

[HS95] Gísli R. Hjaltason and Hanan Samet. Ranking in spatial databases. In *4th International Symposium on Advances in Spatial Databases (SSD)*, pages 83–95, 1995.

[HS05] Michael E. Houle and Jun Sakuma. Fast approximate similarity search in extremely high-dimensional data sets. In *Proceedings of the International Conference on Data Engineering*, pages 619–630, 2005.

[HSD01] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106, 2001.

[HTF02] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning*. Springer, 2002.

[HTF08] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. Datasets for ”The Elements of Statistical Learning”: http://www-stat.stanford.edu/˜tibs/elemstatlearn/, 2008.

[HXD03] Zengyou He, Xiaofei Xu, and Shengchun Deng. Discovering cluster-based local outliers. *Pattern Recognition Letters*, 24(9-10):1641–1650, 2003.

[JA03] Ruoming Jin and Gagan Agrawal. Efficient decision tree construction on streaming data. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 571–576, 2003.

[Jac88] Robert A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1(4):295 – 307, 1988.

[Jen96] Finn V. Jensen. *An Introduction to Bayesian Networks*. Springer Verlag, 1996.

[JL95] George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 338–345, 1995.

[JTH01] Wen Jin, Anthony K. H. Tung, and Jiawei Han. Mining top-n local outliers in large databases. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 293–298, 2001.

[JZC06] Ankur Jain, Zhihua Zhang, and Edward Y. Chang. Adaptive non-linear clustering in data streams. In *CIKM*, pages 122–131, 2006.

[KABS09] Philipp Kranen, Ira Assent, Corinna Baldauf, and Thomas Seidl. Self-adaptive anytime stream clustering. In *Proceedings of the IEEE International Conference on Data Mining*, pages 249–258, 2009.

[KABS11] Philipp Kranen, Ira Assent, Corinna Baldauf, and Thomas Seidl. The ClusTree: Indexing micro-clusters for anytime stream mining. *Knowledge and Information Systems Journal (KAIS)*, 29(2):249–272, 2011.

[KBP+09] Saim Kim, L. Beckmann, M. Pistor, L. Cousin, Marian Walter, and Steffen Leonhardt. A versatile body sensor network for health care applications. In *Proceedings of the International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pages 175–180, 2009.

[KCB+09] Saim Kim, L. Cousin, L. Beckmann, Marian Walter, and Steffen Leonhardt. A body sensor network base support system for automated bioimpedance spectroscopy measurements. In *World Congress on Medical Physics and Biomedical Engineering*, 2009.

[Ken38] Maurice G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1-2):81–93, 1938.

[KGFS10] Philipp Kranen, Stephan Günnemann, Sergej Fries, and Thomas Seidl. MC-Tree: Improving bayesian anytime classification. In *Proceedings of the International Conference on Scientific and Statistical Database Management*, pages 252–269, 2010.

[KGI+11] Hardy Kremer, Stephan Günnemann, Anca Maria Ivanescu, Ira Assent, and Thomas Seidl. Efficient processing of multiple dtw queries in time series databases. In *International Conference on Scientific and Statistical Database Management*. Springer, 2011.

[KHDM98] Josef Kittler, Mohamad Hatef, Robert P. W. Duin, and Jiri Matas. On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:226–239, 1998.

[KHK99] George Karypis, Eui-Hong Han, and Vipin Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *IEEE Computer*, 32(8):68–75, 1999.

[KHY+09] Hillol Kargupta, Jiawei Han, Philip S. Yu, Rajeew Motwani, and Vipin Kumar. *Next Generation of Data Mining*. CRC Press, 2009.

[KJ00] Ralf Klinkenberg and Thorsten Joachims. Detecting concept drift with support vector machines. In *Proceedings of the International Conference on Machine Learning*, pages 487–494, 2000.

[KKDS10] Philipp Kranen, Ralph Krieger, Stefan Denker, and Thomas Seidl. Bulk loading hierarchical mixture models for efficient stream classification. In *Proceedings of the Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, Part II*, pages 325–334, 2010.

[KKJ+10a] Philipp Kranen, Hardy Kremer, Timm Jansen, Thomas Seidl, Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. Benchmarking stream clustering algorithms within the MOA framework. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2010.

[KKJ+10b] Philipp Kranen, Hardy Kremer, Timm Jansen, Thomas Seidl, Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. Clustering performance on evolving data streams: Assessing algorithms and evaluation measures within MOA. In *Proceedings of the IEEE International Conference on Data Mining, Workshops*, pages 1400–1403, 2010.

[KKJ+11] Hardy Kremer, Philipp Kranen, Timm Jansen, Thomas Seidl, Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. An effective evaluation measure for clustering on evolving data

streams. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 868–876. ACM, 2011.

[KKK+08] Philipp Kranen, David Kensche, Saim Kim, Nadine Zimmermann, Emmanuel Müller, Christoph Quix, Xiang Li, Thomas Gries, Thomas Seidl, Matthias Jarke, and Steffen Leonhardt. Mobile mining and information management in healthnet scenarios. In *Proceedings of the International Conference on Mobile Data Management*, pages 215–216, 2008.

[KKSZ09] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Loop: local outlier probabilities. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 1649–1652, 2009.

[KMA+10] Philipp Kranen, Emmanuel Müller, Ira Assent, Ralph Krieger, and Thomas Seidl. Incremental learning of medical data for multi-step patient health classification. In *Plant C., Böhm C. (eds.): Database Technology for Life Sciences and Medicine, World Scientific Publishing*, pages 321–344, 2010.

[KNT00] Edwin M. Knorr, Raymond T. Ng, and Vladimir Tucakov. Distance-based outliers: Algorithms and applications. *VLDB Journal*, 8(3-4):237–253, 2000.

[Koh96] Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 202–207, 1996.

[KP02] Eamonn J. Keogh and Michael J. Pazzani. Learning the structure of augmented bayesian classifiers. *International Journal on Artificial Intelligence Tools*, 11(4):587–601, 2002.

[KPM06] Maria Kontaki, Apostolos N. Papadopoulos, and Yannis Manolopoulos. Efficient incremental subspace clustering in

data streams. In *International Database Engineering and Applications Symposium*, pages 53–60, 2006.

[KPS00] Hans-Peter Kriegel, Marco Ptke, and Thomas Seidl. Managing intervals efficiently in object-relational databases. In *International Conference on Very Large Data Bases*, pages 407–418, 2000.

[KR90] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data An Introduction to Cluster Analysis*. Wiley Interscience, New York, 1990.

[KRSS11] Philipp Kranen, Felix Reidl, Fernando Sanchez Villaamil, and Thomas Seidl. Hierarchical clustering for real-time stream data with noise. In *Proceedings of the International Conference on Scientific and Statistical Database Management*, pages 405–413. Springer, 2011.

[KS09a] Philipp Kranen and Thomas Seidl. Harnessing the strengths of anytime algorithms for constant data streams. *Data Mining and Knowledge Discovery Journal*, 19(2):245–260, 2009.

[KS09b] Philipp Kranen and Thomas Seidl. Harnessing the strengths of anytime algorithms for constant data streams. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*, page 31, 2009.

[KSK02] Aleksander Kolcz, Xiaomei Sun, and Jugal Kalita. Efficient handling of high-dimensional feature spaces by randomized classifier ensembles. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 307–313, 2002.

[KSZ08] Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. Angle-based outlier detection in high-dimensional data. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 444–452, 2008.

[Lau95] Steffen L. Lauritzen. The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, 19:191–201, 1995.

[Law00] Jonathan K. Lawder. Calculation of mappings between one and n-dimensional values using the hilbert space-filling curves. In *Technical Report JL1/00, Birkbeck College, University of London*, 2000.

[LBB02] William B. Langdon, Steven J. Barrett, and Bernard F. Buxton. Combining decision trees and neural networks for drug discovery. In *Proceedings of the European Conference on Genetic Programming*, pages 60–70, 2002.

[LC08] Guopin Lin and Leisong Chen. A grid and fractal dimension-based data stream clustering algorithm. In *Proceedings of the International Symposium on Information Science and Engieering*, volume 1, pages 66–70. IEEE Computer Society, 2008.

[LEL97] Scott T. Leutenegger, Jeffrey M. Edgington, and Mario A. Lopez. Str: A simple and efficient algorithm for R-tree packing. In *Proceedings of the International Conference on Data Engineering*, pages 497–506, 1997.

[LFG+08] Maxim Likhachev, Dave Ferguson, Geoffrey J. Gordon, Anthony Stentz, and Sebastian Thrun. Anytime search in dynamic graphs. *Artif. Intell.*, 172(14):1613–1643, 2008.

[LGT03] Maxim Likhachev, Geoffrey J. Gordon, and Sebastian Thrun. Ara*: Anytime a* with provable bounds on sub-optimality. In *NIPS*, 2003.

[Lia05] T. Warren Liao. Clustering of time series data - a survey. *Pattern Recognition*, 38(11):1857–1874, 2005.

[LJF94] King-Ip Lin, Hosagrahar V. Jagadish, and Christos Faloutsos. The tv-tree: An index structure for high-dimensional data. *VLDB Journal*, 3(4):517–542, 1994.

[LK05] Aleksandar Lazarevic and Vipin Kumar. Feature bagging for outlier detection. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 157–166, 2005.

[LL08] Jae Woo Lee and Won Suk Lee. A coarse-grain grid-based subspace clustering method for online multi-dimensional data streams. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 1521–1522, 2008.

[LL09] Sebastian Lühr and Mihai Lazarescu. Incremental clustering of dynamic data streams using connectivity based representative points. *Data Knowl. Eng.*, 68(1):1–27, 2009.

[Llo57] Stuart P. Lloyd. Least squares quantization in PCM. IEEE Transactions on Information Theory 28(2):128-137, 1982. Original version: Technical Report, Bell Labs, 1957.

[LP03] Kelvin T. Leung and Douglas Stott Parker. Empirical comparisons of various voting methods in bagging. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 595–600, 2003.

[LW96] Chao-Lin Liu and Michael P. Wellman. On state-space abstraction for anytime evaluation of bayesian networks. *SIGART Bulletin*, 7(2):50–57, 1996.

[Mac67] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[MAG+09a] Emmanuel Müller, Ira Assent, Stephan Gnnemann, Ralph Krieger, and Thomas Seidl. Relevant subspace clustering: Mining the most interesting non-redundant concepts in high dimensional data. In *International Conference on Data Mining*, pages 377–386, 2009.

[MAG+09b] Emmanuel Müller, Ira Assent, Stephan Günnemann, Timm Jansen, and Thomas Seidl. OpenSubspace: An open source framework for evaluation and exploration of subspace clustering algorithms in weka. In *Proceedings of the Open Source in Data Mining Workshop in conjunction with the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 2–13, 2009.

[MAK+08] Emmanuel Müller, Ira Assent, Ralph Krieger, Timm Jansen, and Thomas Seidl. Morpheus: interactive exploration of subspace clustering. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1089–1092, 2008.

[MBZ+07] Guillermo Medrano, L. Beckmann, Nadine Zimmermann, T. Grundmann, Thomas Gries, and Steffen Leonhardt. Bioimpedance spectroscopy with textile electrodes for a continous monitoring application. In *International Workshop on Wearable and Implantable Body Sensor Networks*, volume 13, pages 23–28. Springer Berlin Heidelberg, 2007.

[Mei05] Marina Meila. Comparing clusterings: an axiomatic view. In *Proceedings of the International Conference on Machine Learning*, pages 577–584, 2005.

[Mil80] Glenn W. Milligan. An examination of the effect of six types of error perturbation on fifteen clustering algorithms. *Psychometrika*, 45(3):325–342, 1980.

[MM93] Oded Maron and Andrew W. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 59–66, 1993.

[MSE06] Gabriela Moise, Jörg Sander, and Martin Ester. P3c: A robust projected clustering algorithm. In *Proceedings of the IEEE International Conference on Data Mining*, pages 414–425, 2006.

[MSS11] Emmanuel Müller, Matthias Schiffer, and Thomas Seidl. Statistical selection of relevant subspace projections for outlier ranking. In *Proceedings of the 27th International Conference on Data Engineering*, pages 434–445, 2011.

[MSV04] Sambavi Muthukrishnan, Rahul Shah, and Jeffrey Scott Vitter. Mining deviants in time series data streams. In *Proceedings of the International Conference on Scientific and Statistical Database Management*, pages 41–50, 2004.

[New03] Mark E. J. Newman. The Structure and Function of Complex Networks. *SIAM Review*, 45(2):167–256, 2003.

[NH94] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. In *Proceedings of the International Conference on Very Large Data Bases*, pages 144–155, 1994.

[NR06] Olfa Nasraoui and Carlos Rojas. Robust clustering for tracking noisy evolving data streams. In *Proceedings of the SIAM International Conference on Data Mining*, 2006.

[NUCG03] Olfa Nasraoui, Cesar Cardona Uribe, Carlos Rojas Coronel, and Fabio A. González. Tecno-streams: Tracking evolving clusters in noisy data streams with a scalable immune system learning model. In *Proceedings of the IEEE International Conference on Data Mining*, pages 235–242, 2003.

[OMM+02] Liadan O'Callaghan, Adam Meyerson, Rajeev Motwani, Nina Mishra, and Sudipto Guha. Streaming-data algorithms for high-quality clustering. In *Proceedings of the International Conference on Data Engineering*, pages 685–694, 2002.

[PCY95] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu. Efficient parallel and data mining for association rules. In *Proceedings of the International Conference on Information and Knowledge Management*, pages 31–36, 1995.

[PHBB09] Biswanath Panda, Joshua Herbach, Sugato Basu, and Roberto J. Bayardo. Planet: Massively parallel learning of tree ensembles with mapreduce. *Proceedings of the VLDB Endowment*, 2(2):1426–1437, 2009.

[PHK07] Bernhard Pfahringer, Geoffrey Holmes, and Richard Kirkby. New options for hoeffding trees. In *Australian Conference on Artificial Intelligence*, pages 90–99, 2007.

[PL04] Nam Hun Park and Won Suk Lee. Statistical grid-based clustering over data streams. *SIGMOD Record*, 33(1):32–37, 2004.

[PL07] Nam Hun Park and Won Suk Lee. Grid-based subspace clustering over data streams. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 801–810, 2007.

[PL08] Nam Hun Park and Won Suk Lee. Memory efficient subspace clustering for online data streams. In *International Database Engineering and Applications Symposium*, pages 199–208, 2008.

[Pla98] John Platt. Fast training of support vector machines using sequential minimal optimization. In Schoelkopf, Burges, and Smola, editors, *Advances in Kernel Methods*. MIT Press, 1998.

[PW10] Franz Pernkopf and Michael Wohlmayr. Large margin learning of bayesian classifiers based on gaussian mixture models. In *European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 50–66, 2010.

[PZZ+07] Rong Pan, Junhui Zhao, Vincent Wenchen Zheng, Jeffrey Junfeng Pan, Dou Shen, Sinno Jialin Pan, and Qiang Yang. Domain-constrained semi-supervised mining of tracking models in sensor networks. In *International Conference on Knowledge Discovery and Data Mining*, pages 1023–1027, 2007.

[Qui86] John Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Qui93] John Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[Ran71] William M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.

[RC05] Fabio Tozeto Ramos and Fabio Gagliardi Cozman. Anytime anyspace probabilistic inference. *International Journal of Approximate Reasoning*, 38(1):53–80, 2005.

[RGI05] Jimena Rodríguez, Alfredo Goñi, and Arantza Illarramendi. Real-time classification of ecgs on a pda. *Transactions on Information Technology in Biomedicine*, 9(1):23–34, 2005.

[RH97] Y. Dan Rubinstein and Trevor Hastie. Discriminative vs informative learning. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 49–53, 1997.

[RH07] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the Joint Conference on Empirical Methods in*

Natural Language Processing and Computational Natural Language Learning, pages 410–420, 2007.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Robert J. Williams. Learning internal representations by error propagation. In *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1*, pages 318–362. MIT Press, 1986.

[RN95] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[Roh74] F. James Rohlf. Methods of comparing classifications. *Annual Review of Ecology and Systematics*, 5:101–113, 1974.

[RTG00] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40:99–121, 2000.

[RWZH09] Jiadong Ren, Qunhui Wu, Jia Zhang, and Changzhen Hu. Efficient outlier detection algorithm for heterogeneous data streams. In *International Conference on Fuzzy Systems and Knowledge Discovery*, pages 259–264, 2009.

[Sag94] Hans Sagan. *Space-Filling Curves*. Springer, 1994.

[SAK+09] Thomas Seidl, Ira Assent, Philipp Kranen, Ralph Krieger, and Jennifer Herrmann. Indexing density models for incremental learning and anytime classification on data streams. In *Proceedings of the International Conference on Extending Database Technology*, pages 311–322, 2009.

[SCZ98] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *Proceedings of International Conference on Very Large Data Bases*, pages 428–439, 1998.

[SdLFdCG07] Eduardo J. Spinosa, André Carlos Ponce de Leon Ferreira de Carvalho, and João Gama. Olindda: a cluster-based approach for detecting novelty and concept drift in data streams. In *Proceedings of the ACM Symposium on Applied Computing*, pages 448–452, 2007.

[Sei09] Thomas Seidl. Nearest neighbor classification. In *Liu L., Özsu M. T. (eds.): Encyclopedia of Database Systems.*, pages 1885–1890. Springer, 2009.

[SG86] Jeffrey C. Schlimmer and Richard H. Granger. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, 1986.

[SG10] Md. Shiblee Sadik and Le Gruenwald. Dbod-ds: Distance based outlier detection for data streams. In *International Conference on Database and Expert Systems Applications*, pages 122–136, 2010.

[Sil86] Bernard W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, London, 1986.

[SK97] Thomas Seidl and Hans-Peter Kriegel. Efficient user-adaptable similarity search in large multimedia databases. In *International Conference on Very Large Data Bases*, pages 506–515, 1997.

[SK98] Thomas Seidl and Hans-Peter Kriegel. Optimal multi-step k-nearest neighbor search. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 154–165, 1998.

[SK01] W. Nick Street and YongSeog Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 377–382, 2001.

[SM84] Gerard Salton and Michael McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Company, 1984.

[SNTS06] Myra Spiliopoulou, Irene Ntoutsi, Yannis Theodoridis, and Rene Schult. Monic: modeling and monitoring cluster transitions. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 706–711, 2006.

[Spe04] Charles Spearman. The Proof and Measurement of Association between Two Things. *The American Journal of Psychology*, 15(1):72–101, 1904.

[SPP+06] Sharmila Subramaniam, Themis Palpanas, Dimitris Papadopoulos, Vana Kalogeraki, and Dimitrios Gunopulos. Online outlier detection in sensor data using non-parametric models. In *Proceedings of the International Conference on Very Large Data Bases*, pages 187–198, 2006.

[SS05] Juliane Schäfer and Korbinian Strimmer. An empirical bayes approach to inferring large-scale gene association networks. *Bioinformatics*, 21(6):754–764, 2005.

[Sub98] V. S. Subrahmanian. *Principles of Multimedia Database Systems*. Morgan Kaufmann, 1998.

[SVvL06] Arno Siebes, Jilles Vreeken, and Matthijs van Leeuwen. Item sets that compress. In *Proceedings of the SIAM International Conference on Data Mining*, 2006.

[SZ08] Mingzhou (Joe) Song and Lin Zhang. Comparison of cluster representations from partial second- to full fourth-order cross moments for data stream clustering. In *Proceedings of the IEEE International Conference on Data Mining*, pages 560–569, 2008.

[THH01] Anthony K. H. Tung, Jean Hou, and Jiawei Han. Spatial clustering in the presence of obstacles. In *Proceedings of the In-*

ternational Conference on Data Engineering, pages 359–367, 2001.

[TLB04] Young Truong, Xiaodong Lin, and Chris Beecher. Learning a complex metabolomic dataset using random forests and support vector machines. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 835–840, 2004.

[TNLH01] Anthony K. H. Tung, Raymond T. Ng, Laks V. S. Lakshmanan, and Jiawei Han. Constraint-based clustering in large databases. In *International Conference on Database Theory*, pages 405–419, 2001.

[TRA07] Dimitris K. Tasoulis, Gordon J. Ross, and Niall M. Adams. Visualising the cluster structure of data streams. In *International Symposium on Intelligent Data Analysis*, pages 81–92, 2007.

[TTD+08] Liang Tang, Chang-jie Tang, Lei Duan, Chuan Li, Yexi Jiang, Chunqiu Zeng, and Jun Zhu. Movstream: An efficient algorithm for monitoring clusters evolving in data streams. In *The IEEE International Conference on Granular Computing*, pages 582–587, 2008.

[URW07] Komkrit Udommanetanakit, Thanawin Rakthanmanon, and Kitsana Waiyamai. E-stream: Evolution-based technique for stream clustering. In *International Conference on Advanced Data Mining and Applications*, pages 605–615, 2007.

[UXKL06] Ken Ueno, Xiaopeng Xi, Eamonn J. Keogh, and Dah-Jye Lee. Anytime classification using the nearest neighbor algorithm with applications to stream mining. In *Proceedings of the IEEE International Conference on Data Mining*, pages 623–632, 2006.

[Vap95] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, 1995.

[Vap98] Vladimir N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.

[VBF+04] Vassilios S. Verykios, Elisa Bertino, Igor Nai Fovino, Loredana Parasiliti Provenza, Yücel Saygin, and Yannis Theodoridis. State-of-the-art in privacy preserving data mining. *SIGMOD Record*, 33(1):50–57, 2004.

[VD00] Stijn Van Dongen. Performance criteria for graph clustering and Markov cluster experiments. *Report-Information systems*, 1(12):1–36, 2000.

[VGN08] Nguyen Hoang Vu, Vivekanand Gopalkrishnan, and Praneeth Namburi. Online outlier detection based on relative neighbourhood dissimilarity. In *Web Information Systems Engineering*, pages 50–61, 2008.

[VL98] Nuno Vasconcelos and Andrew Lippman. Learning mixture hierarchies. In *International Conference on Advances in Neural Information Processing Systems*, pages 606–612, 1998.

[vLS08] Matthijs van Leeuwen and Arno Siebes. Streamkrimp: Detecting change in data streams. In *European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 672–687, 2008.

[vR79] Cornelis J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979.

[VS09] Rosa M. Valdovinos and José S. Sánchez. Combining multiple classifiers with dynamic weighted voting. In *Proceedings of the International Conference on Hybrid Artificial Intelligence Systems*, pages 510–516, 2009.

[Wah00] Wolfgang Wahlster. *Verbmobil: Foundations of Speech-To-Speech Translation*. Springer, Berlin - Heidelberg - New York, 2000.

[Wan90] Eric A. Wan. Neural network classification: A bayesian interpretation. *IEEE Transactions on Neural Networks*, 1(4), 1990.

[WBW05] Geoffrey I. Webb, Janice R. Boughton, and Zhihai Wang. Not so naive bayes: Aggregating one-dependence estimators. *Machine Learning*, 58(1):5–24, 2005.

[WF94] Stanley Wasserman and Katherine Faust. *Social network analysis: Methods and applications*. Cambridge University Press, 1994.

[WF05] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques, 2nd Edition*. Morgan Kaufmann, 2005.

[WFYH03] Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–235, 2003.

[WIZD04] Sholom Weiss, Nitin Indurkhya, Tong Zhang, and Fred Damerau. *Text Mining: Predictive Methods for Analyzing Unstructured Information*. Springer, 2004.

[WKB97] Kevin Woods, W. Philip Kegelmeyer Jr., and Kevin Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:405–410, 1997.

[WM03] Takashi Washio and Hiroshi Motoda. State of the art of graph-based data mining. *SIGKDD Explorations*, 5(1):59–68, 2003.

[WNB+10] Liang Wang, Uyen T. V. Nguyen, James C. Bezdek, Christopher Leckie, and Kotagiri Ramamohanarao. iVAT and aVAT: Enhanced visual analysis for cluster tendency assessment. In *Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pages 16–27, 2010.

[WSB98] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *International Conference on Very Large Data Bases*, pages 194–205, 1998.

[WXC09] Junjie Wu, Hui Xiong, and Jian Chen. Adapting the right measures for k-means clustering. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 877–886, 2009.

[YiTWM04] Kenji Yamanishi, Jun ichi Takeuchi, Graham J. Williams, and Peter Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. *Data Min. Knowl. Discov.*, 8(3):275–300, 2004.

[YM97] Wei Wang 0010, Jiong Yang, and Richard R. Muntz. Sting: A statistical information grid approach to spatial data mining. In *Proceedings of the International Conference on Very Large Data Bases*, pages 186–195, 1997.

[YRW09] Di Yang, Elke A. Rundensteiner, and Matthew O. Ward. Neighbor-based pattern detection for windows over streaming data. In *International Conference on Extending Database Technology*, pages 529–540, 2009.

[YWC+07] Ying Yang, Geoffrey I. Webb, Jesús Cerquides, Kevin B. Korb, Janice R. Boughton, and Kai Ming Ting. To select or to weigh: A comparative study of linear combination schemes for superparent-one-dependence estimators. *IEEE Transactions on Knowledge and Data Engineering*, 19(12):1652–1665, 2007.

[YWKMN09] Lexiang Ye, Xiaoyue Wang, Eamonn J. Keogh, and Agenor Mafra-Neto. Autocannibalistic and anyspace indexing algorithms with application to sensor data mining. In *Proceedings*

of the SIAM International Conference on Data Mining, pages 85–96, 2009.

[YWKT07] Ying Yang, Geoffrey I. Webb, Kevin B. Korb, and Kai Ming Ting. Classifying under computational resource constraints: anytime classification using probabilistic estimators. *Machine Learning*, 69(1):35–53, 2007.

[ZGW08] Ji Zhang, Qigang Gao, and Hai H. Wang. Spot: A system for detecting projected outliers from high-dimensional data streams. In *Proceedings of the International Conference on Data Engineering*, pages 1628–1631, 2008.

[ZGWW10] Ji Zhang, Qigang Gao, Hai Wang, and Hua Wang. Detecting anomalies from high-dimensional wireless network data streams: a case study. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, pages 1–21, 2010.

[Zil96] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *The AI magazine*, 17(3):73–83, 1996.

[ZK04] Ying Zhao and George Karypis. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Machine Learning*, 55(3):311–331, 2004.

[ZKF05] Cui Zhu, Hiroyuki Kitagawa, and Christos Faloutsos. Example-based robust outlier detection in high dimensional datasets. In *Proceedings of the IEEE International Conference on Data Mining*, pages 829–832, 2005.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: an efficient data clustering method for very large databases. In *SIGMOD*, 1996.

[ZRL99] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Fast density estimation using cf-kernel for very large databases. In

Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 312–316, 1999.

[ZS03] Yunyue Zhu and Dennis Shasha. Efficient elastic burst detection in data streams. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 336–345, 2003.

[ZW06] Fei Zheng and Geoffrey I. Webb. Efficient lazy elimination for averaged one-dependence estimators. In *Proceedings of the International Conference on Machine Learning*, pages 1113–1120, 2006.

[ZW07] Fei Zheng and Geoffrey I. Webb. Finding the right family: Parent and child selection for averaged one-dependence estimators. In *Proceedings of the European Conference on Machine Learning*, pages 490–501, 2007.

[ZWLL09] Lei Zheng, Shaojun Wang, Yan Liu, and Chi-Hoon Lee. Information theoretic regularization for semi-supervised boosting. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1017–1026, 2009.

Statement of Originality

This thesis would not have been possible without the close collaboration in the group of Professor Seidl. Many of the presented ideas and techniques evolved from numerous fruitful discussions in the group. While the high level of collaboration both within the group and with the students constitutes a key factor of the productive environment I found at Professor Seidl's group, it makes it hard to pinpoint individual contributions. The order of authors on the publications gives certainly a good impression on the contributions in terms of novel ideas. The following provides some more detail on collaborations and support for the individual chapters. Parts I, IV and V as well as the future work chapters 10 and 16 constitute new material that I added in order to put the contents of the two main parts into context.

The Bayes tree was initiated by Professor Seidl. The first diploma thesis on the Bayes tree was done by Jennifer Herrmann, advised by Ira Assent and Ralph Krieger. The refinement strategies were evaluated in a later diploma thesis by Stefan Denker, whom I advised together with Ralph Krieger. These contents of Chapter 4 were published in [SAK+09]. The MC-Tree was developed in the diploma thesis of Sergej Fries, whom I advised together with Stephan Günnemann; the corresponding Chapter 5 was published in [KGFS10]. The bulk loading approaches presented in Chapter 6 were published in [KKDS10] and were investigated in the diploma thesis of Stefan Denker. Finally, the concept transfer approaches in Chapter 7 resulted from numerous discussion over the past years. The implementation and evaluation was greatly supported by the students from the lab course of winter semester 2010, which I advised together with Marwan Hassani.

The HealthNet project was conducted within the UMIC research cluster at RWTH Aachen University and constitutes joint work of four institutes: MedIt,

ITA, i5 and i9. The contents of Chapter 8 were published in [KKK+08] and [KMA+10].

The Meta approaches presented in Chapter 9 were published in [KS09a] and [KS09b]. Their implementation and evaluation was greatly supported by the students of the lab course of winter semester 2008, which I advised together with Emmanuel Müller. Subsection 9.3.3 and Figures 9.14 and 9.15 were not part of the publication in [KS09a]. The idea to evaluate the resulting confidence distributions in comparison to the theoretical model resulted from discussions with Michael Houle, Michael Wolf helped in experimental evaluation and the creation of the above mentioned Figures.

Anytime stream clustering was investigated in the diploma thesis of Corinna Baldauf, which I advised together with Ira Assent. The ClusTree presented in Chapter 11 was published in [KABS09]. Evaluating the alternative descend strategies for the ClusTree was greatly supported by Fernando Sanchez Villaamil and Felix Reidl; the corresponding contents of Chapter 12 were published in [KABS11]. Felix and Fernando also worked with me on the LiarTree, which is presented in Chapter 13 and published in [KRSS11].

Primary experiments for using the ClusTree for anytime outlier detection were done in the diploma thesis of Corinna Baldauf. Extensive experiments were greatly supported by Stephan Wels, the results of Chapter 14 were partly published in [AKBS10].

The extensions of MOA to stream clustering presented in Chapter 15 were initiated by me and build upon the MOA framework by Albert Bifet, Richard Kirkby, Geoff Holmes and Bernhard Pfahringer of the Waikato University in New Zealand. Large parts of it were implemented during the diploma thesis of Timm Jansen, whom I advised together with Hardy Kremer. An article on MOA was published in [BHP+10a], further appearances of MOA include [BHP+10b, KKJ+10a, KKJ+10b]. The cluster mapping measure was investigated in the thesis of Timm Jansen. It is introduced in Chapter 15 and published in [KKJ+11].

As stated above, the order of authors on the publications gives generally a good impression on the contributions in terms of novel ideas. The publications of the author are listed separately in the following for convenience.

List of Publications

Journal publications and book chapters

[KS09a] P. Kranen and T. Seidl. Harnessing the strengths of anytime algorithms for constant data streams. In *Data Mining and Knowledge Discovery Journal, Special Issue on Best Papers from ECML PKDD, (DMKD) Vol. 19, No. 2*, 2009.

[BHP+10a] A. Bifet, G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, and T. Seidl. MOA: Massive online analysis, a framework for stream classification and clustering. In *Journal of Machine Learning Research (JMLR) Vol. 11*, 2010.

[KABS11] P. Kranen, I. Assent, C. Baldauf, and T. Seidl. The ClusTree: Indexing micro-clusters for anytime stream mining. In *Knowledge and Information Systems Journal (KAIS) Vol. 29, No. 2*, 2011.

[WKAS10] M. Wichterich, P. Kranen, I. Assent, and T. Seidl. Efficient EMD-based similarity search in medical image databases. In *Plant C., Böhm C. (eds.): Database Technology for Life Sciences and Medicine, World Scientific Publishing*, pages 175–201, 2010.

[KMA+10] P. Kranen, E. Müller, I. Assent, R. Krieger, and T. Seidl. Incremental learning of medical data for multi-step patient health classification. In *Plant C., Böhm C. (eds.): Database Technology for Life Sciences and Medicine, World Scientific Publishing*, pages 321–344, 2010.

Peer reviewed full paper publications

[WAKS08] M. Wichterich, I. Assent, P. Kranen, and T. Seidl. Efficient EMD-Based Similarity Search in Multimedia Databases via Flexible Dimensionality Reduction. *ACM International Conference on Management of Data (SIGMOD)*, 2008.

[SAK+09] T. Seidl, I. Assent, P. Kranen, R. Krieger, and J. Herrmann. Indexing density models for incremental learning and anytime classification on data streams. *International Conference on Extending Database Technology (EDBT/ICDT)*, 2009.

[KS09b] P. Kranen and T. Seidl. Harnessing the strengths of anytime algorithms for constant data streams. *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Springer LNCS*, 2009.

[KABS09] P. Kranen, I. Assent, C. Baldauf, and T. Seidl. Self-adaptive anytime stream clustering. *IEEE International Conference on Data Mining (ICDM)*, 2009.

[MKN+10] E. Müller, P. Kranen, M. Nett, F. Reidl, and T. Seidl. Air-indexing on error prone communication channels. *International Conference on Database Systems for Advanced Applications (DASFAA), Springer LNCS*, 2010.

[KGFS10] P. Kranen, S. Günnemann, S. Fries, and T. Seidl. MC-tree: Improving Bayesian anytime classification. *International Conference on Scientific and Statistical Database Management (SSDBM), Springer LNCS*, 2010.

[KKJ+11] H. Kremer, P. Kranen, T. Jansen, T. Seidl, A. Bifet, G. Holmes and B. Pfahringer. An Effective Evaluation Measure for Clustering on Evolving Data Streams. *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2011.

Other peer reviewed publications

[KKK+08] P. Kranen, D. Kensche, S. Kim, N. Zimmermann, E. Müller, C. Quix, X. Li, T. Gries, T. Seidl, M. Jarke, and S. Leonhardt. Mobile mining and information management in HealthNet scenarios. *IEEE International Conference on Mobile Data Management (MDM)*, 2008.

[MKN+08] E. Müller, P. Kranen, M. Nett, F. Reidl, and T. Seidl. A general framework for data dissemination simulation for real world scenarios. *ACM SIGMOBILE International Conference on Mobile Computing and Networking (MobiCom)*, 2008.

[Kra09] P. Kranen. Using index structures for anytime stream mining. *PhD Workshop of the International Conference on Very Large Data Bases (VLDB)*, 2009.

[KKDS10] P. Kranen, R. Krieger, S. Denker, and T. Seidl. Bulk loading hierarchical mixture models for efficient stream classification. *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, Springer LNAI, 2010.

[KKJ+10a] P. Kranen, H. Kremer, T. Jansen, T. Seidl, A Bifet, G. Holmes, and B. Pfahringer. Benchmarking stream clustering algorithms within the MOA framework. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2010.

[AKBS10] I. Assent, P. Kranen, C. Baldauf, and T. Seidl. Detecting outliers on arbitrary data streams using anytime approaches. *International Workshop on Novel Data Stream Pattern Mining Techniques (StreamKDD) in conjunction ACM SIGKDD*, 2010.

[BHP+10b] A Bifet, G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, and T. Seidl. MOA: Massive online analysis, a framework for stream classification and clustering. *International Workshop on Handling Concept Drift in Adaptive Information Systems (HaCDAIS) in conjunction with ECML PKDD*, 2010.

[KKJ+10b] P. Kranen, H. Kremer, T. Jansen, T. Seidl, A. Bifet, G. Holmes, and B. Pfahringer. Clustering performance on evolving data streams: Assessing algorithms and evaluation measures within MOA. *IEEE International Conference on Data Mining (ICDM)*, 2010.

[KRSS11] P. Kranen, F. Reidl, F. Sanchez Villaamil, and T. Seidl. Hierarchical clustering for real-time stream data with noise. *International Conference on Scientific and Statistical Database Management (SSDBM), Springer LNCS*, 2011.

Curriculum Vitae

Name	Philipp Kranen
Academic Degree	Diplom-Informatiker
Born on	December 3rd, 1979
Born in	Willich, Germany
Nationality	German

Education

from 08/2007	RWTH Aachen University, Germany Computer science: Doctoral studies
10/2001 – 12/2006	RWTH Aachen University, Germany Computer science: Diplom Informatik
09/2004 – 02/2005	UNITECH scholarship at TU Delft, Netherlands
02/2003 – 09/2003	ERASMUS scholarship at UIB, Spain
10/2000 – 09/2001	RWTH Aachen University, Germany Mathematics studies
08/1990 – 06/1999	Lise-Meitner-Gymnasium Geldern, Germany Abitur

Professional Experience

from 08/2007	Computer Science 9, RWTH Aachen, Germany Research assistant
01/2007 – 06/2007	Siemens Corporate Research, Princeton, NJ, USA Intelligent Vision and Reasoning Department
03/2005 – 10/2005	Gründerkolleg Aachen, Germany Development and implementation of web applications
10/1999 – 08/2000	Gelderland Klinik, Germany Mandatory civilian service in lieu of military service