CS6200 Information Retrieval Homework1: Retrieval Models

Objective

Implement and compare various retrieval systems using vector space models and language models. Explain how and why their performance differs.

This assignment will also introduce you to elasticsearch: one of the many available commercial-grade indexes. These instructions will generally not spell out how to accomplish various tasks in elasticsearch; instead, you are encouraged to try to figure it out by reading the online documentation. If you are having trouble, feel free to ask for help on Piazza or in office hours.

This assignment involves writing two programs:

1. A program to parse the corpus and index it with elasticsearch
2. A query processor, which runs queries from an input file using a selected retrieval model

First Steps

- Download and install elasticsearch (http://www.elasticsearch.org), and the marvel/sense (http://www.elasticsearch.org/guide/en/marvel/current/_installation.html) plugin
- Download IR_data/AP89_DATA.zip from the Dropbox Data link. Contact the TAs if you need the password.

Document Indexing

Create an index of the downloaded corpus. The documents are found within the ap89_collection folder in the data .zip file. You will need to write a program to parse the documents and send them to your elasticsearch instance.

The corpus files are in a standard format used by TREC. Each file contains multiple documents. The format is similar to XML, but standard XML and HTML parsers will not work correctly. Instead, read the file one line at a time with the following rules:

1. Each document begins with a line containing `<DOC>` and ends with a line containing `</DOC>`.
2. The first several lines of a document's record contain various metadata. You should read the `<DOCNO>` field and use it as the ID of the document.
3. The document contents are between lines containing `<TEXT>` and `</TEXT>`.
4. All other file contents can be ignored.

Make sure to index term positions: you will need them later. You are also free to add any other fields to your index for later use. This might be the easiest way to get particular values used in the scoring functions. However, if a value is provided by elasticsearch then we encourage you to retrieve it using the elasticsearch API rather than calculating and storing it yourself.

Query execution

Write a program to run the queries in the file query_desc.51-100.short.txt, included in the data .zip file. You should run all queries (omitting the leading number) using each of the retrieval models listed below, and output the top 100 results for each query to an output file. If a particular query has fewer than 100 documents with a nonzero matching score, then just list whichever documents have nonzero scores.

You should write precisely one output file per retrieval model. Each line of an output file should specify one retrieved document, in the following format:

```
<query-number> Q0 <docno> <rank> <score> Exp
```

Where:

- is the number preceding the query in the query list
- is the document number, from the `<DOCNO>` field (which we asked you to index)
- is the document rank: an integer from 1-1000
- is the retrieval model's matching score for the document
- Q0 and Exp are entered literally

Your program will run queries against elasticsearch. Instead of using their built in query engine, we will be retrieving information such as TF and DF scores from elasticsearch and implementing our own document ranking. It will be helpful if you write a method which takes a term as a parameter and retrieves the postings for that term from elasticsearch. You can then easily reuse this method to implement the retrieval models.

Implement the following retrieval models, using TF and DF scores from your elasticsearch index, as needed.

Okapi TF

This is a vector space model using a slightly modified version of TF to score documents. The Okapi TF score for term w in document d is as follows.

$$okapi\_tf(w,d) = \frac{tf_{w,d}}{tf_{w,d} + 0.5 + 1.5 \cdot (len(d)/avg(len(d)))}$$

Where:

- $tf_{w,d}$ is the term frequency of term w in document d
- $len(d)$ is the length of document d
- $avg(len(d))$ is the average document length for the entire corpus

The matching score for document d and query q is as follows.

$$tf(d,q) = \sum_{w \in q} okapi\_tf(w,d)$$

TF-IDF

This is the second vector space model. The scoring function is as follows.

$$tfidf(d,q) = \sum_{w \in q} okapi\_tf(w,d) \cdot \log \frac{D}{df_w}$$

Where:

- D is the total number of documents in the corpus
- df_w is the number of documents which contain term w

Okapi BM25

BM25 is a language model based on a binary independence model. Its matching score is as follows.

$$bm25(d,q) = \sum_{w \in q} \left[\log\left(\frac{D + 0.5}{df_w + 0.5}\right) \cdot \frac{tf_{w,d} + k_1 \cdot tf_{w,d}}{tf_{w,d} + k_1 \left((1-b) + b \cdot \frac{len(d)}{avg(len(d))}\right)} \cdot \frac{tf_{w,q} + k_2 \cdot tf_{w,q}}{tf_{w,q} + k_2} \right]$$

Where:

- $tf_{w,q}$ is the term frequency of term w in query q
- k_1 , k_2 , and b are constants. You can use the values from the slides, or try your own.

Unigram LM with Laplace smoothing

This is a language model with Laplace (“add-one”) smoothing. We will use maximum likelihood estimates of the query based on a multinomial model “trained” on the document. The matching score is as follows.

$$lm\_laplace(d,q) = \sum_{w \in q} \log p\_laplace(w|d)$$

$$p\_laplace(w|d) = \frac{tf_{w,d} + 1}{len(d) + V}$$

Where:

- V is the vocabulary size – the total number of unique terms in the collection.

Unigram LM with Jelinek-Mercer smoothing

This is a similar language model, except that here we smooth a foreground document language model with a background model from the entire corpus.

$$lm\_jm(d, q) = \sum_{w \in q} \log p\_jm(w|d)$$

$$p\_jm(w|d) = \lambda \frac{tf_{w,d}}{len(d)} + (1 - \lambda) \frac{\sum_{d'} tf_{w,d'}}{\sum_{d'} len(d')}$$

Where:

- $\lambda \in (0, 1)$ is a smoothing parameter which specifies the mixture of the foreground and background distributions.

Think carefully about how to efficiently obtain the background model here. If you wish, you can instead estimate the corpus probability using $\frac{cf_w}{V}$.

Evaluation

Download trec_eval (trec_eval) and use it to evalute your results for each retrieval model.

To perform an evaluation, run:

```
trec_eval [-q] qrel_file results_file
```

The `-q` option shows a summary average evaluation across all queries, followed by individual evaluation results for each query; without the -q option, you will see only the summary average. The trec_eval program provides a wealth of statistics about how well the uploaded file did for those queries, including average precision, precision at various recall cut-offs, and so on.

You should evaluate using the QREL file named qrels.adhoc.51-100.AP89.txt, included in the data .zip file.

Create a document showing the following metrics:

- The uninterpolated mean average precision numbers for all five retrieval models.
- Precision at 10 and precision at 30 documents for all five retrieval models.

Be prepared to answer questions during your demo as to how model performance compares, why the best model outperformed the others, and so on.

Extra Credit

These extra problems are provided for students who wish to dig deeper into this project. Extra credit is meant to be significantly harder and more open-ended than the standard problems. We strongly recommend completing all of the above before attempting any of these problems.

Points will be awarded based on the difficulty of the solution you attempt and how far you get. You will receive no credit unless your solution is “at least half right,” as determined by the graders.

EC1: Pseudo-relevance Feedback

Implement pseudo-relevance feedback. The general algorithm is:

1. Retrieve the top k documents using one of the above retrieval models.
2. Identify terms in these documents which are distinctive to the documents.
3. Add the terms to the query, and re-run the retrieval process. Return the final results.

It is up to you to devise a reasonable way to choose terms to add to the query. It doesn’t have to be complicated, but you should be prepared to explain and justify your approach.

Evaluate your results using trec_eval and include similar metrics with your submission.

EC2: Pseudo-relevance Feedback using ElasticSearch aggs "significant terms"

Use ES API "significat terms" separately on each query term (stem root) to get a list of related words. The words you want to add to the query are:
  - related to more than one query term

- not stopwords
- high IDF
- other tricks you might need in order to only get interesting words

Add few of these words to the query and rerun your models.

Below is an example of this API in Sense for query term "atom":

```
GET /ap_dataset/document/_search
{
  "query" : {
    "terms" : {"TEXT" : [ "atom" ]}
  },
  "aggregations" : {
    "significantCrimeTypes" : {
      "significant_terms" : {
       "field" : "TEXT"
      }
    }
  },
  "size": 0
}
```

EC3: Bigram Language Models

Perform retrieval using a bigram language model. You should match bigrams from the query against bigrams in the document. You may smooth with a unigram language model, if you wish.

You should devise a matching score function for this task. Your matching score should assign smaller scores to documents when the two bigram terms appear at a greater distance from each other. Use term position information from elasticsearch to accomplish this. Be prepared to explain and justify your matching score function to the graders.

Evaluate your results using trec_eval and include similar metrics with your submission.

EC4: MetaSearch

Combine your models (all 5, or some of them) using a metasearch algorithm (../other_notes/AslamMo01.ps.pdf) for ranking fusion. You can try simple such algorithms (CombMNZ, Borda Count) or fancy ones like Condorcet. Your metasearch ranking results must be in the same format as before. Evaluate with trec_eval.

Submission and Grading

Submission checklist

- Your indexer's source code
- Your query program's source code
- Your results

Rubric

15 points	You index the documents correctly
15 points	You can retrieve term and document statistics from elasticsearch
50 points	You have implemented the retrieval models correctly
20 points	Your evaluation results are reasonable, and you can adequately explain your results during the demo.