1 Boosting

Boosting refers to a general and effective method of producing accurate classifier by combining moderately inaccurate classifiers, which are called weak learners. In the lecture, we'll describe three boosting algorithms: adaBoost, rankBoost and gradient boosting. AdaBoost is the first successful instance of Boosting algorithm in history. We'll first introduce AdaBoost.

2 AdaBoost

2.1 Intuition

AdaBoost is motivated by the following observation: a committee can often make accurate decisions, although each member only provides some weak judgment. AdaBoost solves learning problems by gathering wisdom from incompetent members. It assumes there is a weak learning algorithm available. For example, a weak learner can be a decision tree. Given some training data, a weak learner can produce a weak hypothesis which is not entirely trivial in the sense that it does at least a little bit better than random guessing. Weak learners trained with different data can produce different hypotheses. These weak learners serves as committee members. To gather wisdom from these members, adaBoost tries to solve two problems: First, how to select a diverse set of members with different strengths and weaknesses, so that they can compensate each other? Second, how to combine their inaccurate judgments together to get a better decision?

To solve the first problem, adaBoost picks its weak learners h in such a fashion that each newly added weak learner is able to infer something new about the data. To implement this idea, adaBoost maintains a weight distribution D among all data points. Each data point is assigned a weight $D(i)$ indicting its importance. When measuring a weak learner's performance, adaBoost takes into consideration each data point's weight. A misclassified high-weight data point will contribute more to the overall weighted error than a misclassified low-weight data point. To get a low weighted error, a weak learner must focus on high-weight data points and try to predict them correctly. Therefore by manipulating the weight distribution, we can guide the weak learner to pay attention to different part of the data. AdaBoost proceeds by rounds. In each

round t, we update weight distribution and find a weak learner that minimize the weighted error with respect to the current weight distribution D_t. At the first round, all weights are equal. On later rounds, we increase the weights of misclassified data points, and decrease the weights of correctly classified data points. So essentially, in each round, we ask the weak learner to focus on hard data points that previous weak learners cannot handle well.

Now we have trained a set of weak learners with different strength, how do we combine them in order to make predictions? Each weak leaner is trained with different weight distributions. So we can think that we've assigned different tasks to different weak learners and each weak learner tried its best to do its given task. Intuitively, when we take into account one weak leaner's judgment into the final prediction, we are willing to trust the weak leaner more if it did well on its previous task, and less if it did badly on its previous task. In other words, when we combine different weak learners' judgments, we take a weighted vote. We assign to each weak leaner a confidence value α_t, which depends on the weak learner's performance on its assigned task.

Before we go into the details of the algorithm, let's look at an example (Figure 1). Here we have 10 data points labeled as -1 or +1. Suppose our weak learners are vertical lines or horizontal lines. It's easy to see no single line can classify all 10 data points correctly. As we will see, adaBoost can do perfectly on the data set by combining three weak learners.

On the first round, each data point has equal weight. A weak learner h_1 is trained to minimize the weighted error. It mis-classifies three data points (circled in the figure 1). On round 2, these three data points' weights are increased (as indicted by the size), and other data points' weights are decreased. Now h_2 pays more attentions to these three data points and correctly classifies them. But h_2 mis-classified some other data points with lower weights. On round 3, h_3 only mis-classifies three data points with very low weights. The weighted vote of h_1, h_2 and h_3 is in Figure 2. As can be seen, the combined classifier makes no mistake.

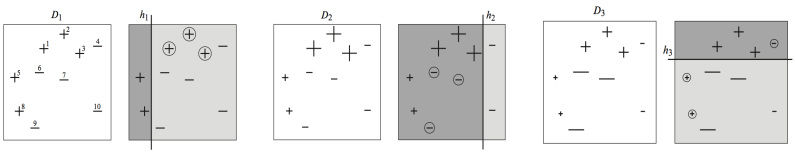

Figure 1: Weight distribution and weak learner in each iteration. Source: Figure 1.1 of [7]

2.2 Algorithm

Now we are ready to present the algorithm in detail.
Given $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, 1\}$.
Initialize $D_1(i) = 1/m$ for $i = 1, ..., m$.

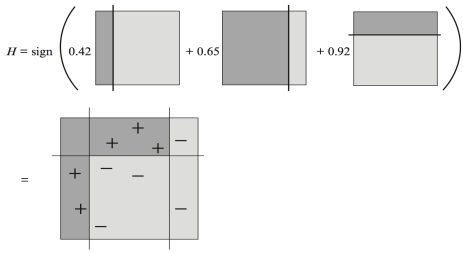

Figure 2: Ensemble of weak learners. Source: Figure 1.2 of [7]

For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis h_t: $\mathcal{X} \to \{-1, 1\}$.
- Aim: select h_t to minimize the weighted error:

$$\epsilon_t = Pr_{i \sim D_t}[h_t(x_t) \neq y_i]$$

- Choose $\alpha_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- Update

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases} \\ &= \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i)))}{Z_t} \end{aligned}$$

  where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

2.3 AdaBoost's Training Error

Let $\gamma = \frac{1}{2} - \epsilon_t$, and let D_1 be an arbitrary initial distribution over the training set. It can be shown [7] that the weighted training error of the combined classifier H with respect to D_1 is bounded as

$$Pr_{i \sim D_1}[H(x_i) \neq y_i] \leq \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2} \leq exp(-2 \sum_{t=1}^{T} \gamma_t^2)$$

Proof: Define $F(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$. We have

$$\begin{aligned} D_{T+1}(i) &= D_1(i) \times \frac{e^{-y_i \alpha_1 h_1(x_i)}}{Z_1} \times \cdots \times \frac{e^{-y_i \alpha_T h_T(x_i)}}{Z_T} \\ &= \frac{D_1(i) exp(-y_i \sum_{t=1}^{T} \alpha_t h_t(x_i))}{\prod_{t=1}^{T} Z_t} \\ &= \frac{D_1(i) exp(-y_i F(x_i))}{\prod_{t=1}^{T} Z_t} \end{aligned}$$

$$\begin{aligned} Pr_{i \sim D_1}[H(x_i) \neq y_i] &= \sum_{i=1}^{m} D_1(i) \mathbf{1}\{H(x_i) \neq y_i\} \\ &\leq \sum_{i=1}^{m} D_1(i) exp(-y_i F(x_i)) \\ &= \sum_{i=1}^{m} D_{T+1}(i) \prod_{t=1}^{T} Z_t \\ &= \prod_{t=1}^{T} Z_t \end{aligned}$$

$$\begin{aligned} Z_t &= \sum_{i=1}^{m} D_t(i) e^{-\alpha_t y_i h_t(x_i)} \\ &= \sum_{i: y_i = h_t(x_i)} D_t(i) e^{-\alpha_t} + \sum_{i: y_i \neq h_t(x_i)} D_t(i) e^{\alpha_t} \\ &= e^{-\alpha_t}(1 - \epsilon_t) + e^{\alpha_t} \epsilon_t \\ &= e^{-\alpha_t}(\frac{1}{2} + \gamma_t) + e^{\alpha_t}(\frac{1}{2} - \gamma_t) \\ &= \sqrt{1 - 4\gamma_t^2} \end{aligned}$$

Corollary:

There are two possibilities for ending AdaBoost training

1. Training error goes to 0

2. γ_t=0(equivalent $\epsilon = 0.5$). Boosting gets stuck: the boosting weights on training set are in such a way that every weak learner has 50 % error.

3 RankBoost

3.1 Learning to Rank

In this section, we describe the RankBoost algorithm, which is a special boosting algorithm used in ranking problems. Ranking problems arise in many different

domains. For example, in web search, a search engine ranks web pages according to their relevance to a user's query. In product recommendations, the system ranks products according to the chance that the viewer will like them or buy them. In learning to rank problems, the goal is to find good ranking rule. RankBoost is an efficient algorithm that can automatically learn a ranking rule based on some training data.

We first describe the setting of learning to rank problems. The objects to be ranked are called data points. The set of all data points are denoted as $\mathcal{X}$. The leaner is provided with a collection of training data $V \subset \mathcal{X}$. Usually, each training data point comes with its user rating, rated on some scale. For example, In a movie recommendation problem, we can have some movies, each of which rated from 1 star to 5 stars. A 5-star movie is preferred to a 4-star movie; but there is no direct comparisons between two 4-star movies. Formally, let $V = V_1 \cup V_2 \cdots \cup V_J$ be given training data points, where $V_1, ..., V_J$ are J disjoint subsets of $\mathcal{X}$. For $j < k$, all data points in V_k are ranked above all data points in V_j. We call such user feedback layered feedback. Let $E = \{(u, v) | v \text{ is ranked above } u\}$. E is the set of preference pairs. Then $E = \bigcup_{1 \le j < k \le J} V_j \times V_k$. Our aim is to find a good ranking rule that is maximally consistent with the training data. Mathematically, the ranking rule is a real-valued function $F : \mathcal{X} \to \mathcal{R}$ with the interpretation that F ranks v above u if $F(v) > F(u)$. The degree of inconsistency can be measured by the fraction of misorderings:

$$\frac{1}{|E|} \sum_{(u,v) \in E} \mathbf{1}\{F(v) \le F(u)\}$$

This is also called empirical ranking loss.

3.2 RankBoost Algorithm

To minimize empirical ranking loss, different versions of RankBoost algorithms are developed. The algorithm presented below reduces the ranking problem to binary classification problems. The reduction is inexact; but the resulting procedure is very efficient. For each data point x, we assign two labels -1 and $+1$ with different probabilities(or weights) $D(x, -1)$ and $D(x, +1)$. In each iteration, the distribution of weights is recomputed, and a weak learner (classifier) is trained to minimized the weighted error. To recompute weight distributions, first we make one scan of the data to get $S_{t,j}(y)$ based on the current predictions. Second we combine $S_{t,j}(y)$ to get $C_{t,j}(y)$. $C_{t,j}(y)$ is the same for all $x \in V_j$. Finally, $D_t(x, y)$ is computed based on $C_{t,j}(y)$. The algorithm is very efficient because each iteration takes $O(|V|)$ time, regardless of J. The full algorithm [7] is presented below:

Given: nonempty, disjoint subsets $V_1, ..., V_J$ of $\mathcal{X}$ representing preference pairs

$$E = \bigcup_{1 \le j < k \le J} V_j \times V_k.$$

Initialize: $F_0 = 0$.

For $t = 1, ..., T$ ::

- For $j = 1, ..., J$ and $y \in \{-1, +1\}$, let

$$S_{t,j}(y) = \sum_{x \in V_j} exp(yF_{t-1}(x))$$

and let

$$C_{t,j}(+1) = \sum_{k=1}^{j-1} S_{t,k}(+1),$$

$$C_{t,j}(-1) = \sum_{k=j+1}^{J} S_{t,k}(-1).$$

- Train instance-based weak learner using distribution D_t, where, for $x \in V_j$, $j = 1, ..., J$ and $y \in \{-1, +1\}$,

$$D_t(x, y) = \frac{1}{2Z} \cdot exp(-yF_{t-1}(x)) \cdot C_{t,j}(y),$$

and where Z_t is a normalization factor (chosen so that D_t is a distribution).

- Get weak classifier $h_t : \mathcal{X} \to \{-1, +1\}$.
- Aim: Choose h_t to minimize the weighted error:

$$\epsilon_t = Pr_{(x,y)\sim D}[h_t(x) \neq y]$$

- Choose $\alpha_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- Update: $F_t = F_{t-1} + \frac{1}{2}\alpha_t h_t$.

Output the final ranking: $F(x) = F_T(x) = \frac{1}{2} \sum_{t=1}^{T} \alpha_t h_t(x)$.

4 Gradient Boosting

4.1 Introduction

Gradient Boosting is a powerful and versatile machine learning algorithm. It can be used in regression, classification and ranking problems. In 2009, gradient boosting won Track 1 of the Yahoo Learning to Rank Challenge. As the names suggests, gradient boosting is a combination of boosting and gradient descent ideas.

4.2 Boosting as a forward stage-wise additive model fitting algorithm

As described in the previous section, AdaBoost fits an additive model (ensemble) $\sum_t \rho_t h_t(x)$ in a forward stage-wise manner. In each stage, AdaBoost introduces a weak learner to compensate the shortcomings of existing weak learners.In AdaBoost,"shortcomings" are identified by high-weight data points.

Gradient Boosting employs the same idea. Gradient Boosting fits an additive model (ensemble) $\sum_t \rho_t h_t(x)$ in a forward stage-wise manner. In each stage, Gradient Boosting introduces a weak learner to compensate the shortcomings of existing weak learners. In Gradient Boosting, "shortcomings" are identified by gradients. Both high-weight data points and gradients tell us how to improve our model.

Historically, AdaBoost is the first successful boosting algorithm [4, 3]. Later, Breiman formulated AdaBoost as gradient descent with a special loss function[2, 1]. Then Freiman generalized AdaBoost to Gradient Boosting in order to handle a variety of loss functions [5, 6].

4.3 Gradient Boosting for Regression Problems

Gradient Boosting can solve regression problems naturally. It solves classification and ranking problems by casting them into regression problems. So we will start with regression problems.

4.3.1 Regression with Square Loss

Let's play a game... You are given $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, and the task is to fit a model $F(x)$ to minimize square loss. Suppose your friend wants to help you and gives you a model F. You check his model and find the model is good but not perfect. There are some mistakes: $F(x_1) = 0.8$, while $y_1 = 0.9$, and $F(x_2) = 1.4$ while $y_2 = 1.3$... How can you improve this model?

Rule of the game:

- You are not allowed to remove anything from F or change any parameter in F.
- You can add an additional model (regression tree) h to F, so the new prediction will be $F(x) + h(x)$.

Here is a simple solution:

You wish to improve the model such that

$$\begin{aligned} F(x_1) + h(x_1) &= y_1 \\ F(x_2) + h(x_2) &= y_2 \\ &... \\ F(x_n) + h(x_n) &= y_n \end{aligned}$$

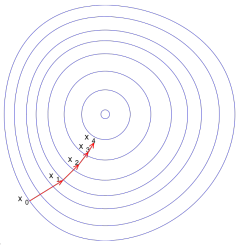

Figure 3: Gradient Descent. Source: `http://en.wikipedia.org/wiki/Gradient_descent`

Or, equivalently, you wish

$$\begin{aligned} h(x_1) &= y_1 - F(x_1) \\ h(x_2) &= y_2 - F(x_2) \\ &... \\ h(x_n) &= y_n - F(x_n) \end{aligned}$$

Can any regression tree h achieve this goal perfectly?

Can any regression tree h achieve this goal perfectly? Maybe not.... But some regression tree might be able to do this approximately. How?

Just fit a regression tree h to data $(x_1, y_1 - F(x_1)), (x_2, y_2 - F(x_2)), ..., (x_n, y_n - F(x_n))$. Congratulations, you get a better model!

$y_i - F(x_i)$ are called **residuals**. These are the parts that existing model F cannot do well. The role of h is to compensate the shortcoming of existing model F. If the new model $F + h$ is still not satisfactory, we can add another regression tree...

You may wonder, we are improving the predictions of training data, is the procedure also useful for test data? The answer is Yes! Because we are building a model, and the model can be applied to test data as well.

This is the basic idea of Gradient Boosting for regression problem. Clearly, it follows the boosting idea, which is about introducing a weak learner to compensate the shortcomings of existing weak learners. Let's look at how it is related to gradient descent.

Gradient descent algorithm minimizes a function by moving in the opposite direction of the gradient.

$$\theta_i := \theta_i - \rho \frac{\partial J}{\partial \theta_i}$$

In the Gradient Boosting algorithm, we have the square loss function $L(y, F(x)) = (y - F(x))^2/2$. Our goal is to minimize $J = \sum_i L(y_i, F(x_i))$ by adjusting

$F(x_1), F(x_2), ..., F(x_n)$. Notice that $F(x_1), F(x_2), ..., F(x_n)$ are just some numbers. We can treat $F(x_i)$ as parameters and take derivatives

$$\frac{\partial J}{\partial F(x_i)} = \frac{\partial \sum_i L(y_i, F(x_i))}{\partial F(x_i)} = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = F(x_i) - y_i$$

So we can interpret residuals as negative gradients.

$$y_i - F(x_i) = -\frac{\partial J}{\partial F(x_i)}$$

Compare this with gradient descent, we see:

$$\begin{aligned} F(x_i) &:= F(x_i) + h(x_i) \\ F(x_i) &:= F(x_i) + y_i - F(x_i) \\ F(x_i) &:= F(x_i) - 1\frac{\partial J}{\partial F(x_i)} \\ \theta_i &:= \theta_i - \rho\frac{\partial J}{\partial \theta_i} \end{aligned}$$

For regression with **square loss**,

$$\textit{residual} \Leftrightarrow \textit{negative gradient}$$

$$\textit{fit } h \textit{ to residual} \Leftrightarrow \textit{fit } h \textit{ to negative gradient}$$

$$\textit{update } F \textit{ based on residual} \Leftrightarrow \textit{update } F \textit{ based on negative gradient}$$

So we are actually updating our model using **gradient descent**!

It turns out that the concept of **gradients** is more general and useful than the concept of **residuals**. So from now on, let's stick with gradients. The reason will be explained later.

Let us summarize the algorithm we just derived using the concept of gradients. Negative gradient:

$$-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = y_i - F(x_i)$$

start with an initial model, say, $F(x) = \frac{\sum_{i=1}^n y_i}{n}$
iterate until converge:
calculate negative gradients $-g(x_i)$
fit a regression tree h to negative gradients $-g(x_i)$
$F := F + \rho h$, where $\rho = 1$

4.3.2 Regression with Other Loss Functions

The benefit of formulating this algorithm using gradients is that it allows us to consider other loss functions and derive the corresponding algorithms in the same way.

You may ask, why do we need to consider other loss functions? Isn't square loss good enough?

Square loss is:

✓ Easy to deal with mathematically

× Not robust to outliers

Outliers are heavily punished because the error is squared.

Let's look at an example:

y_i	0.5	1.2	2	5*
$F(x_i)$	0.6	1.4	1.5	1.7
$L = (y - F)^2/2$	0.005	0.02	0.125	5.445

Suppose the last data point is mislabeled. We can see the that data point alone contributes about 97 % to the overall square loss. The consequence is that the algorithm pays too much attention to outliers. It tries hard to incorporate outliers into the model. Doing so may degrade the overall performance.

There are some alternatives to square loss.

- Absolute loss (more robust to outliers)

$$L(y, F) = |y - F|$$

- Huber loss (more robust to outliers)

$$L(y, F) = \begin{cases} \frac{1}{2}(y - F)^2 & |y - F| \leq \delta \\ \delta(|y - F| - \delta/2) & |y - F| > \delta \end{cases}$$

y_i	0.5	1.2	2	5*
$F(x_i)$	0.6	1.4	1.5	1.7
Square loss	0.005	0.02	0.125	5.445
Absolute loss	0.1	0.2	0.5	3.3
Huber loss($\delta = 0.5$)	0.005	0.02	0.125	1.525

Following the Gradient Boosting idea, we can derive an algorithm for Regression with **Absolute Loss**:
Negative gradient:

$$-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = sign(y_i - F(x_i))$$

start with an initial model, say, $F(x) = \frac{\sum_{i=1}^{n} y_i}{n}$
iterate until converge:
calculate gradients $-g(x_i)$
fit a regression tree h to negative gradients $-g(x_i)$
$F := F + \rho h$

Following the Gradient Boosting idea, we can derive an algorithm for Regression with **Huber Loss**:
Negative gradient: Negative gradient:

$$\begin{aligned} -g(x_i) &= -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \\ &= \begin{cases} y_i - F(x_i) & |y_i - F(x_i)| \leq \delta \\ \delta sign(y_i - F(x_i)) & |y_i - F(x_i)| > \delta \end{cases} \end{aligned}$$

start with an initial model, say, $F(x) = \frac{\sum_{i=1}^{n} y_i}{n}$
iterate until converge:
calculate negative gradients $-g(x_i)$
fit a regression tree h to negative gradients $-g(x_i)$
$F := F + \rho h$

Now we can summarize the above three algorithms into a general procedure:
Regression with loss function L:
Give any differentiable loss function L

start with an initial model, say $F(x) = \frac{\sum_{i=1}^{n} y_i}{n}$
iterate until converge:
calculate negative gradients $-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$
fit a regression tree h to negative gradients $-g(x_i)$
$F := F + \rho h$

In general,

$$negative\ gradients \not\Leftrightarrow residuals$$

We should follow negative gradients rather than residuals. Why? Let's look at the updat rule for Huber loss.

If we update by negative gradient:

$$h(x_i) = -g(x_i) = \begin{cases} y_i - F(x_i) & |y_i - F(x_i)| \leq \delta \\ \delta sign(y_i - F(x_i)) & |y_i - F(x_i)| > \delta \end{cases}$$

If we update by residual:

$$h(x_i) = y_i - F(x_i)$$

We can see the difference: updating by negative gradient takes advantage of the robust loss function and pays less attention to outliers.

We didn't talk about how to choose a proper learning rate for each gradient boosting algorithm. For detailed discussions, see [6].

4.4 Gradient Boosting for Classification

We will discuss how to apply Gradient Boosting to classification problems through an letter recognition example. The task is to recognize the given hand written capital letter. In this multi-class classification problem, we have 26 classes, i.e., A,B,C...,Z. The letter recognition data set can be downloaded from `http://archive.ics.uci.edu/ml/datasets/Letter+Recognition`. There are 20000 data points, each of which has 16 extracted features.

In our Gradient Boosting algorithm, there are 26 score functions (our models): $F_A, F_B, F_C, ..., F_Z$. Give a data point x, each score function assigns a score for one class. For example, $F_A(x)$ assigns a score for class A. These scores are used to calculate class probabilities.

$$P_A(x) = \frac{e^{F_A(x)}}{\sum_{c=A}^{Z} e^{F_c(x)}}$$

$$P_B(x) = \frac{e^{F_B(x)}}{\sum_{c=A}^{Z} e^{F_c(x)}}$$

$$...$$

$$P_Z(x) = \frac{e^{F_Z(x)}}{\sum_{c=A}^{Z} e^{F_c(x)}}$$

Then the predicted label is just the class which has the highest probability.

Gradient Boosting works by approximating the true conditional probabilities $p(y = k|x)$. To measure how good the approximation is, we use KL-divergence. KL-divergence between two probability distributions is defined as

$$D_{KL}(p||q) = \sum_i p_i \log \frac{p_i}{q_i}$$

Here p is usually the true distribution, q is our estimated distribution. KL-divergence works as a loss function. More precisely, in our problem, we calculate the loss function for each data point using the following steps:

Step 1 turn the label y_i into a (true) probability distribution $Y_c(x_i)$

For example: y_5=G, $Y_A(x_5) = 0, Y_B(x_5) = 0, ..., Y_G(x_5) = 1, ..., Y_Z(x_5) = 0$. See Fig 2.

Step 2 calculate the predicted probability distribution $P_c(x_i)$ based on the current model $F_A, F_B, ..., F_Z$.

$P_A(x_5) = 0.03, P_B(x_5) = 0.05, ..., P_G(x_5) = 0.3, ..., P_Z(x_5) = 0.05$. See Fig 3.

Step 3 calculate the difference between the true probability distribution and the predicted probability distribution using $D_{KL}(Y||P)$.

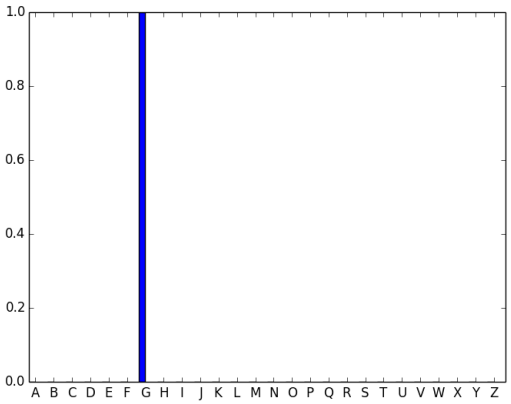

Figure 4: true probability distribution

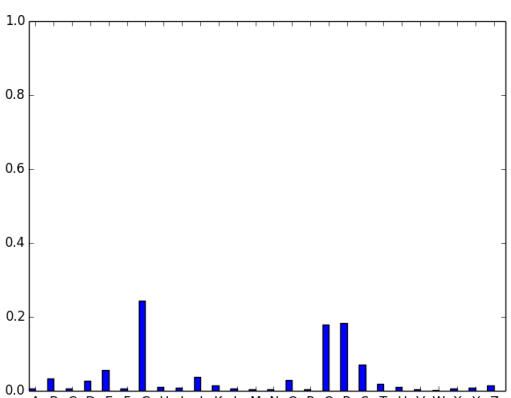

Figure 5: predicted probability distribution based on current model

As we can see, Gradient Boosting translates the classification problem into a regression problem. Our goal is to minimize the total loss (KL-divergence). For each data point, we wish the predicted probability distribution to match the true probability distribution as closely as possible. We try achieve this goal by adjusting our models $F_A, F_B, ..., F_Z$.

Comparing the setups of Gradient Boosting for classification and Gradient Boosting for regression, we can see now we have more parameters to optimize. The followings are the differences:

- $F_A, F_B, ..., F_Z$ vs F
- a matrix of parameters to optimize vs a column of parameters to optimize

$F_A(x_1)$	$F_B(x_1)$	...	$F_Z(x_1)$
$F_A(x_2)$	$F_B(x_2)$	...	$F_Z(x_2)$
...	...	...	...
$F_A(x_n)$	$F_B(x_n)$	...	$F_Z(x_n)$

- a matrix of gradients vs a column of gradients

$\frac{\partial L}{F_A(x_1)}$	$\frac{\partial L}{F_B(x_1)}$	...	$\frac{\partial L}{F_Z(x_1)}$
$\frac{\partial L}{F_A(x_2)}$	$\frac{\partial L}{F_B(x_2)}$	...	$\frac{\partial L}{F_Z(x_2)}$
...	...	...	...
$\frac{\partial L}{F_A(x_n)}$	$\frac{\partial L}{F_B(x_n)}$	...	$\frac{\partial L}{F_Z(x_n)}$

Given the above differences, one can imagine that the Gradient Boosting procedure for classification problems is very similar to that for regression problems, except that now we need to optimize each column of parameters.

start with initial models $F_A, F_B, F_C, ..., F_Z$
iterate until converge:
calculate negative gradients for class A: $-g_A(x_i) = -\frac{\partial L}{\partial F_A(x_i)} = Y_A(x_i) - P_A(x_i)$
calculate negative gradients for class B: $-g_B(x_i) = -\frac{\partial L}{\partial F_B(x_i)} = Y_B(x_i) - P_B(x_i)$
...
calculate negative gradients for class Z:$-g_Z(x_i) = -\frac{\partial L}{\partial F_Z(x_i)} = Y_Z(x_i) - P_Z(x_i)$
fit a regression tree h_A to negative gradients $-g_A(x_i)$
fit a regression tree h_B to negative gradients $-g_B(x_i)$
...
fit a regression tree h_Z to negative gradients $-g_Z(x_i)$
$F_A := F_A + \rho_A h_A$
$F_B := F_A + \rho_B h_B$
...
$F_Z := F_A + \rho_Z h_Z$

The following tables illustrate its update procedure:

round 0

i	y	Y_A	Y_B	Y_C	Y_D	Y_E	Y_F	Y_G	Y_H	Y_I	Y_J	Y_K	Y_L	Y_M	Y_N	Y_O	Y_P	Y_Q	Y_R	Y_S	Y_T	Y_U	Y_V	Y_W	Y_X	Y_Y	Y_Z
1	T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
2	I	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	D	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	N	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
5	G	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

i	y	F_A	F_B	F_C	F_D	F_E	F_F	F_G	F_H	F_I	F_J	F_K	F_L	F_M	F_N	F_O	F_P	F_Q	F_R	F_S	F_T	F_U	F_V	F_W	F_X	F_Y	F_Z
1	T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

i	y	P_A	P_B	P_C	P_D	P_E	P_F	P_G	P_H	P_I	P_J	P_K	P_L	P_M	P_N	P_O	P_P	P_Q	P_R	P_S	P_T	P_U	P_V	P_W	P_X	P_Y	P_Z
1	T	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
2	I	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
3	D	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
4	N	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
5	G	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

i	y	$Y_A - P_A$	$Y_B - P_B$	$Y_C - P_C$	$Y_D - P_D$	$Y_E - P_E$	$Y_F - P_F$	$Y_G - P_G$	$Y_H - P_H$	$Y_I - P_I$	$Y_J - P_J$	$Y_K - P_K$	$Y_L - P_L$	$Y_M - P_M$	$Y_N - P_N$	$Y_O - P_O$	$Y_P - P_P$	$Y_Q - P_Q$	$Y_R - P_R$	$Y_S - P_S$	$Y_T - P_T$	$Y_U - P_U$	$Y_V - P_V$	$Y_W - P_W$	$Y_X - P_X$	$Y_Y - P_Y$	$Y_Z - P_Z$
1	T	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	0.96	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04
2	I	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	0.96	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04
3	D	-0.04	-0.04	-0.04	0.96	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04
4	N	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	0.96	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04
5	G	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	0.96	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

round 1

i	y	Y_A	Y_B	Y_C	Y_D	Y_E	Y_F	Y_G	Y_H	Y_I	Y_J	Y_K	Y_L	Y_M	Y_N	Y_O	Y_P	Y_Q	Y_R	Y_S	Y_T	Y_U	Y_V	Y_W	Y_X	Y_Y	Y_Z
1	T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
2	I	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	D	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	N	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
5	G	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

i	y	F_A	F_B	F_C	F_D	F_E	F_F	F_G	F_H	F_I	F_J	F_K	F_L	F_M	F_N	F_O	F_P	F_Q	F_R	F_S	F_T	F_U	F_V	F_W	F_X	F_Y	F_Z
1	T	-0.08	-0.07	-0.06	-0.07	-0.02	-0.02	-0.08	-0.02	-0.03	-0.03	-0.06	-0.04	-0.08	-0.08	-0.07	-0.07	-0.02	-0.04	-0.04	0.59	-0.01	-0.07	-0.07	-0.05	-0.06	-0.07
2	I	-0.08	0.23	-0.06	-0.07	-0.02	-0.02	0.16	-0.02	-0.03	-0.03	-0.06	-0.04	-0.08	-0.08	-0.07	-0.07	-0.02	-0.04	-0.04	-0.07	-0.01	-0.07	-0.07	-0.05	-0.06	-0.07
3	D	-0.08	0.23	-0.06	-0.07	-0.02	-0.02	-0.08	-0.02	-0.03	-0.03	-0.06	-0.04	-0.08	-0.08	-0.07	-0.07	-0.02	-0.04	-0.04	-0.07	-0.01	-0.07	-0.07	-0.05	-0.06	-0.07
4	N	-0.08	-0.07	-0.06	-0.07	-0.02	-0.02	0.16	-0.02	-0.03	-0.03	-0.06	-0.04	-0.08	0.23	-0.07	-0.07	-0.02	-0.04	-0.04	-0.07	-0.01	-0.07	-0.07	-0.05	-0.06	-0.07
5	G	-0.08	0.23	-0.06	-0.07	-0.02	-0.02	0.16	-0.02	-0.03	-0.03	-0.06	-0.04	-0.08	-0.08	-0.07	-0.07	-0.02	-0.04	-0.04	-0.07	-0.01	-0.07	-0.07	-0.05	-0.06	-0.07
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

i	y	P_A	P_B	P_C	P_D	P_E	P_F	P_G	P_H	P_I	P_J	P_K	P_L	P_M	P_N	P_O	P_P	P_Q	P_R	P_S	P_T	P_U	P_V	P_W	P_X	P_Y	P_Z
1	T	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.07	0.04	0.04	0.04	0.04	0.04	0.04	0.04
2	I	0.04	0.05	0.04	0.04	0.04	0.04	0.05	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
3	D	0.04	0.05	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
4	N	0.04	0.04	0.04	0.04	0.04	0.04	0.05	0.04	0.04	0.04	0.04	0.05	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
5	G	0.04	0.05	0.04	0.04	0.04	0.04	0.05	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

i	y	$Y_A - P_A$	$Y_B - P_B$	$Y_C - P_C$	$Y_D - P_D$	$Y_E - P_E$	$Y_F - P_F$	$Y_G - P_G$	$Y_H - P_H$	$Y_I - P_I$	$Y_J - P_J$	$Y_K - P_K$	$Y_L - P_L$	$Y_M - P_M$	$Y_N - P_N$	$Y_O - P_O$	$Y_P - P_P$	$Y_Q - P_Q$	$Y_R - P_R$	$Y_S - P_S$	$Y_T - P_T$	$Y_U - P_U$	$Y_V - P_V$	$Y_W - P_W$	$Y_X - P_X$	$Y_Y - P_Y$	$Y_Z - P_Z$
1	T	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	0.96	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04
2	I	-0.04	-0.05	-0.04	-0.04	-0.04	-0.04	-0.05	-0.04	0.96	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04
3	D	-0.04	-0.05	-0.04	0.96	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04
4	N	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.05	-0.04	-0.04	-0.04	-0.04	-0.05	-0.04	0.96	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04
5	G	-0.04	-0.05	-0.04	-0.04	-0.04	-0.04	0.95	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

round 2

i	y	Y_A	Y_B	Y_C	Y_D	Y_E	Y_F	Y_G	Y_H	Y_I	Y_J	Y_K	Y_L	Y_M	Y_N	Y_O	Y_P	Y_Q	Y_R	Y_S	Y_T	Y_U	Y_V	Y_W	Y_X	Y_Y	Y_Z
1	T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
2	I	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	D	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	N	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
5	G	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

i	y	F_A	F_B	F_C	F_D	F_E	F_F	F_G	F_H	F_I	F_J	F_K	F_L	F_M	F_N	F_O	F_P	F_Q	F_R	F_S	F_T	F_U	F_V	F_W	F_X	F_Y	F_Z
1	T	-0.15	-0.14	-0.12	-0.14	-0.03	0.28	-0.14	-0.04	1.49	-0.07	-0.11	-0.08	-0.14	-0.17	-0.13	-0.13	-0.04	-0.11	-0.07	1.05	0.19	0.25	-0.16	-0.09	0.33	-0.14
2	I	-0.15	0.16	-0.12	-0.14	-0.03	-0.08	0.33	-0.04	-0.07	-0.07	-0.11	-0.08	-0.14	-0.17	-0.13	-0.13	-0.04	-0.11	-0.07	-0.11	-0.07	-0.15	-0.16	-0.09	-0.13	-0.14
3	D	-0.15	0.16	-0.12	-0.14	-0.03	-0.08	0.1	-0.04	-0.07	-0.07	-0.11	-0.08	-0.14	-0.17	-0.13	-0.13	-0.04	0.19	-0.07	-0.11	-0.07	-0.15	-0.16	-0.09	-0.13	-0.14
4	N	-0.15	-0.14	-0.12	-0.14	-0.03	-0.08	0.1	-0.04	-0.07	-0.07	0.46	-0.08	-0.14	0.5	-0.13	-0.13	-0.04	-0.11	-0.07	-0.11	-0.07	-0.15	0.25	-0.09	-0.13	-0.14
5	G	-0.15	0.16	-0.12	-0.14	-0.03	-0.08	0.33	-0.04	-0.07	-0.07	-0.11	-0.08	-0.14	-0.17	-0.13	-0.13	-0.04	0.19	-0.07	-0.11	-0.07	-0.15	-0.16	-0.09	-0.13	-0.14
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

i	y	P_A	P_B	P_C	P_D	P_E	P_F	P_G	P_H	P_I	P_J	P_K	P_L	P_M	P_N	P_O	P_P	P_Q	P_R	P_S	P_T	P_U	P_V	P_W	P_X	P_Y	P_Z
1	T	0.03	0.03	0.03	0.03	0.03	0.04	0.03	0.03	0.15	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.09	0.04	0.04	0.04	0.03	0.05	0.03	
2	I	0.04	0.05	0.04	0.04	0.04	0.04	0.06	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
3	D	0.04	0.05	0.04	0.04	0.04	0.04	0.05	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.05	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
4	N	0.03	0.03	0.03	0.03	0.04	0.04	0.04	0.04	0.04	0.06	0.04	0.03	0.06	0.03	0.03	0.04	0.04	0.04	0.04	0.04	0.04	0.03	0.05	0.04	0.03	0.03
5	G	0.03	0.05	0.04	0.04	0.04	0.04	0.06	0.04	0.04	0.04	0.04	0.04	0.04	0.03	0.04	0.04	0.04	0.05	0.04	0.04	0.04	0.04	0.03	0.04	0.04	0.04
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

i	y	$Y_A - P_A$	$Y_B - P_B$	$Y_C - P_C$	$Y_D - P_D$	$Y_E - P_E$	$Y_F - P_F$	$Y_G - P_G$	$Y_H - P_H$	$Y_I - P_I$	$Y_J - P_J$	$Y_K - P_K$	$Y_L - P_L$	$Y_M - P_M$	$Y_N - P_N$	$Y_O - P_O$	$Y_P - P_P$	$Y_Q - P_Q$	$Y_R - P_R$	$Y_S - P_S$	$Y_T - P_T$	$Y_U - P_U$	$Y_V - P_V$	$Y_W - P_W$	$Y_X - P_X$	$Y_Y - P_Y$	$Y_Z - P_Z$
1	T	-0.03	-0.03	-0.03	-0.03	-0.03	-0.04	-0.03	-0.03	-0.15	-0.03	-0.03	-0.03	-0.03	-0.03	-0.03	-0.03	-0.03	-0.03	-0.03	0.91	-0.04	-0.04	-0.03	-0.03	-0.05	-0.03
2	I	-0.04	-0.05	-0.04	-0.04	-0.04	-0.04	-0.06	-0.04	0.96	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04
3	D	-0.04	-0.05	-0.04	0.96	-0.04	-0.04	-0.05	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.05	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04
4	N	-0.03	-0.03	-0.03	-0.03	-0.04	-0.04	-0.04	-0.04	-0.04	-0.06	-0.04	-0.03	0.94	-0.03	-0.03	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.03	-0.05	-0.04	-0.03	-0.03
5	G	-0.03	-0.05	-0.04	-0.04	-0.04	-0.04	0.94	-0.04	-0.04	-0.04	-0.04	-0.04	-0.04	-0.03	-0.04	-0.04	-0.04	-0.05	-0.04	-0.04	-0.04	-0.04	-0.03	-0.04	-0.04	-0.04
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

We can see how the predicted probability distribution of a particular data point (with true label = G) gets updated.

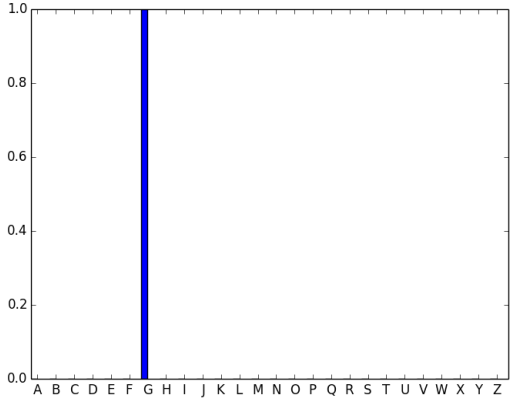

Figure 6: true probability distribution

References

[1] Leo Breiman. Prediction games and arcing algorithms. *Neural computation*, 11(7):1493–1517, 1999.

[2] Leo Breiman et al. Arcing classifier (with discussion and a rejoinder by the author). *The annals of statistics*, 26(3):801–849, 1998.

[3] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[4] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156, 1996.

[5] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Special invited paper. additive logistic regression: A statistical view of boosting. *Annals of statistics*, pages 337–374, 2000.

[6] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

[7] Robert E Schapire and Yoav Freund. *Boosting: Foundations and Algorithms*. MIT Press, 2012.

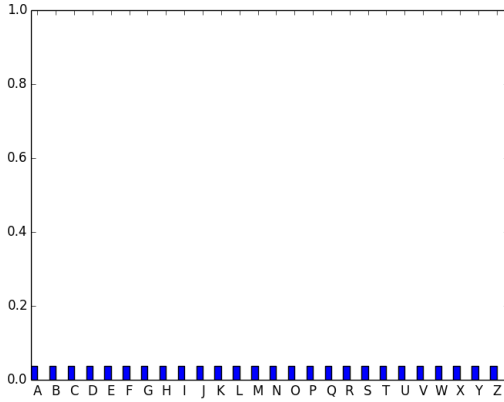

Figure 7: predicted probability distribution at round 0

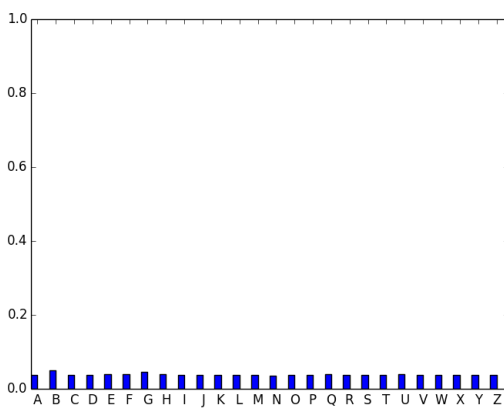

Figure 8: predicted probability distribution at round 1

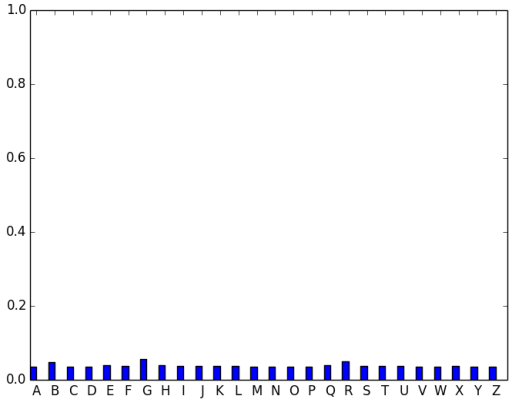

Figure 9: predicted probability distribution at round 2

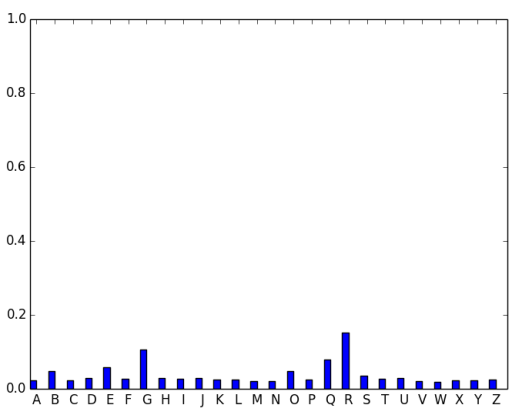

Figure 10: predicted probability distribution at round 10

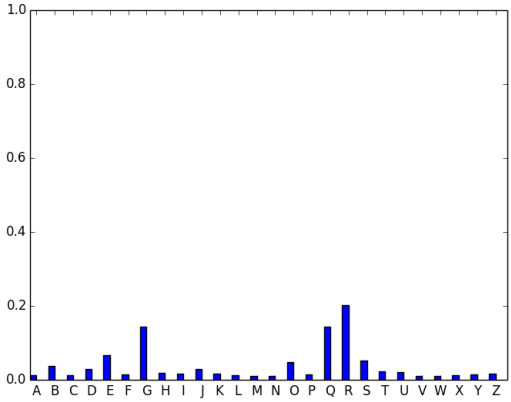

Figure 11: predicted probability distribution at round 20

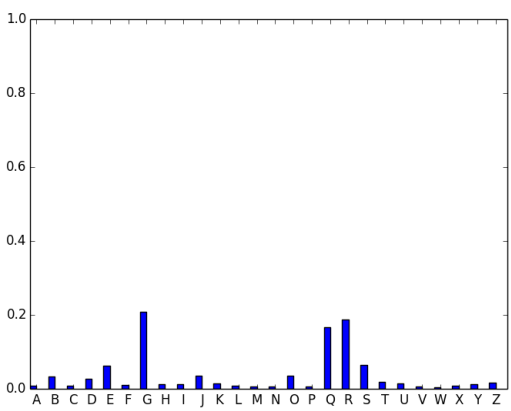

Figure 12: predicted probability distribution at round 30

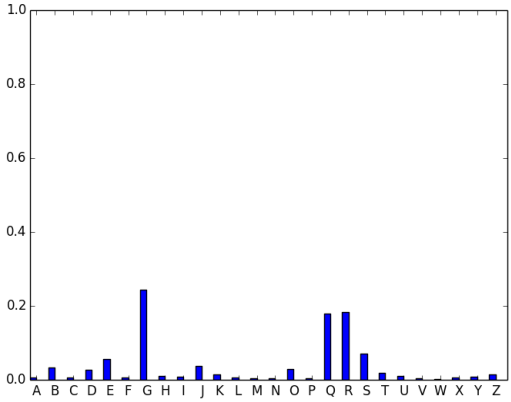

Figure 13: predicted probability distribution at round 40

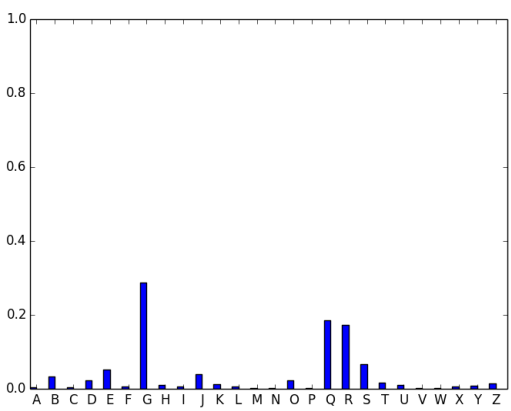

Figure 14: predicted probability distribution at round 50

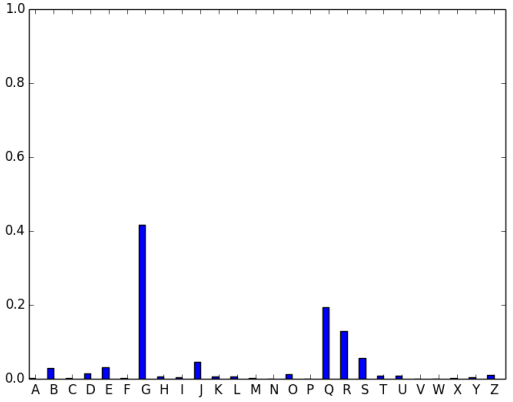

Figure 15: predicted probability distribution at round 100

Gradient Derivation in Gradient Boosting for Binary Classification

Setup

We consider binary classification with labels $y_i \in \{0, 1\}$, and a model output function $F(x)$ defined as an additive ensemble:

$$F(x) = \sum_{m=1}^{M} f_m(x)$$

The predicted probability uses the logistic (sigmoid) function:

$$p(x) = \frac{1}{1 + e^{-F(x)}}$$

The log-likelihood for a single data point is:

$$\ell_i(F) = y_i \log p(x_i) + (1 - y_i) \log(1 - p(x_i))$$

Gradient Derivation

To compute the gradient of the log-likelihood with respect to $F(x_i)$, we use the chain rule:

$$\frac{\partial \ell_i}{\partial F(x_i)} = \frac{\partial}{\partial F(x_i)} \left[y_i \log \left(\frac{1}{1 + e^{-F(x_i)}} \right) + (1 - y_i) \log \left(\frac{e^{-F(x_i)}}{1 + e^{-F(x_i)}} \right) \right]$$

$$= -y_i \cdot \frac{e^{-F(x_i)}}{1 + e^{-F(x_i)}} + (1 - y_i)(-1) + (1 - y_i) \cdot \frac{e^{-F(x_i)}}{1 + e^{-F(x_i)}}$$

$$= y_i \cdot \frac{e^{-F(x_i)}}{1 + e^{-F(x_i)}} - (1 - y_i) + (1 - y_i) \cdot \frac{e^{-F(x_i)}}{1 + e^{-F(x_i)}}$$

$$= \frac{e^{-F(x_i)}}{1 + e^{-F(x_i)}}(y_i + (1 - y_i)) - (1 - y_i) = \frac{1}{1 + e^{F(x_i)}} - (1 - y_i)$$

$$= (1 - p(x_i)) - (1 - y_i) = y_i - p(x_i)$$

Gradient Expression

The final gradient of the log-likelihood with respect to $F(x_i)$ is:

$$\frac{\partial \ell_i}{\partial F(x_i)} = y_i - p(x_i)$$

In gradient boosting, we fit the weak learner $f_m(x)$ to the negative gradient (residual):

$$r_i^{(m)} = -\frac{\partial \ell_i}{\partial F(x_i)} = p(x_i) - y_i$$

Summary Table

Concept	**Expression**
Predicted probability	$p(x) = \frac{1}{1+e^{-F(x)}}$
Log-likelihood	$\ell_i(F) = y_i \log p(x) + (1 - y_i) \log(1 - p(x))$
Gradient of loss	$\frac{\partial \ell_i}{\partial F(x_i)} = y_i - p(x_i)$
Boosting target	$r_i^{(m)} = p(x_i) - y_i$

Gradient Derivation in Multiclass Gradient Boosting

Setup

We consider multiclass classification with L classes. Each observation x_i has a true label $y_i \in \{1, 2, \ldots, L\}$.

Let $F_k(x)$ be the model score (logit) for class k. The predicted class probabilities are computed using the softmax function:

$$p_k(x) = \frac{e^{F_k(x)}}{\sum_{j=1}^{L} e^{F_j(x)}}$$

The negative log-likelihood (cross-entropy) loss for a single sample is:

$$\ell_i(\mathbf{F}) = -\sum_{k=1}^{L} \mathbf{1}_{\{y_i=k\}} \log p_k(x_i) = -\log p_{y_i}(x_i)$$

Gradient Derivation

We compute the gradient of the loss with respect to the score $F_k(x_i)$. There are two cases:

Case 1: $k = y_i$

$$\ell_i = -\log\left(\frac{e^{F_k(x_i)}}{\sum_{j=1}^{L} e^{F_j(x_i)}}\right) = -F_k(x_i) + \log\left(\sum_{j=1}^{L} e^{F_j(x_i)}\right)$$

$$\frac{\partial \ell_i}{\partial F_k(x_i)} = -1 + \frac{e^{F_k(x_i)}}{\sum_{j=1}^{L} e^{F_j(x_i)}} = -1 + p_k(x_i)$$

Case 2: $k \neq y_i$

Only the softmax denominator contributes:

$$\frac{\partial \ell_i}{\partial F_k(x_i)} = \frac{e^{F_k(x_i)}}{\sum_{j=1}^{L} e^{F_j(x_i)}} = p_k(x_i)$$

Final Result

Combining both cases:

$$\frac{\partial \ell_i}{\partial F_k(x_i)} = p_k(x_i) - \mathbf{1}_{\{y_i=k\}}$$

This is a vector-valued gradient for each training example.

Summary Table

Quantity	**Expression**
Softmax probability	$p_k(x) = \frac{e^{F_k(x)}}{\sum_{j=1}^{L} e^{F_j(x)}}$
Loss per sample	$\ell_i = -\log p_{y_i}(x_i)$
Gradient	$\frac{\partial \ell_i}{\partial F_k(x_i)} = p_k(x_i) - \mathbf{1}_{\{y_i=k\}}$