Engineering Applications of Machine Learning and Data Analytics

Adaboost

Contents

1 Introduction

We have seen several classifiers that are able to learn a complex decision boundary at the cost of increasing the complexity of a model. For example, learning a neural network with high accuracy on some problems can be achieved by increasing the complexity. Thus, generating a decision rule with a high accuracy is hard to achieve. On the other hand, generating a decision rule that is slightly better than flipping a coin is easy! We can think of this decision as more of a rule-of-thumb, which is probably not something you would trust with your retirement, but will work out quite well for boosting. The motivation is that the overall cost/complexity of learning a rule-of-thumb is easy, but a complex decision rule is hard to obtain!

Adaptive boosting was developed to convert a strategically learned set of "rules-of-thumb" into a high accuracy decision rule. We shall refer to this rule-of-thumb as a weak learning algorithm. For example, a decision stump is a weak learning algorithm. Boosting classifiers was originally developed from the computational learning theory literature, which showed that the aggregation of these weak learners can result in a very strong decision rule. In fact, this lecture will prove that the training error decreases exponentially with the number of weak learners added to the ensemble. The general idea is that classifiers are generated sequentially by focusing the learning problem on the samples that have not yet been sufficiently learned by the previous classifiers.

2 The Adaboost Algorithm

Let us begin with a very simple explanation of the Adaboost algorithm before we dive into the elegancy of the formulation of the approach. Algorithm 1 shows a high level pseudo-code for the approach. We begin with at training data set $\mathcal{S} := \{x_i, y_i\}_{i=1}^N$ of N samples with labels that we shall assume are in $\{\pm 1\}$ and the dimensionality is p for x_i. The "user" is expected to provided the number of learning rounds, which is referred to as T. Perhaps the vaguest term in Algorithm 1 is the hypothesis class $\mathcal{H}$. $\mathcal{H}$ represents the "class" of the classifier. For example if $\mathcal{H}$ is a decision stump then the class of functions is all of those that are thresholding functions. Another example would be that $\mathcal{H}$ is the set of all possible linear functions. The take-away point here is that $\mathcal{H}$ is not a strong decision rule such as a neural network or support vector machine. Rather, $\mathcal{H}$ is a class of decision rules that is at least correct 51% of the time it makes a prediction.

Adaboost holds a set of weight over the instances that can be viewed a the "cost" of misclassification. In the beginning, the cost of misclassifying a sample can be seen as equally likely since

Algorithm 1 Adaboost (Adaptive Boosting)

Input: $\mathcal{S} := \{x_i, y_i\}_{i=1}^N$, learning rounds T, and hypothesis class $\mathcal{H}$
Initialize: $\mathcal{D}_1(i) = 1/n$
1: **for** $t = 1, \ldots, T$ **do**
2: $h_t = \textsc{WeakLearn}(h, \mathcal{S}, \mathcal{D}_t)$
3: $\epsilon_t = \sum \mathcal{D}_t(i)[\![h(\mathbf{x}_i) \neq y_i]\!]$
4: $\alpha_t = \frac{1}{2} \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
5: $\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i)}{Z_t} \exp\left(-\alpha_t y_i h_t(x_i)\right)$
6: **end for**
7: **Output:** $H(x) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(x)\right)$

we do not know how difficult it is to classify a sample. Thus, choosing $\mathcal{D}_1(i) = 1/n$ will set the misclassification of sample i to be equal for all samples. Adaboost modifies these sample weights to make some samples' misclassification more costly than others.

Adaboot begins with $t = 1$. The first step is to find a decision rule/hypothesis/classifier h that minimizes the cost according to $\mathcal{D}_t$. This can easily be achieved in one of two ways:

- *Sampling*: Sample a new data set from the distribution $\mathcal{D}_t$, where instances with larger weight have a larger probability of appearing in the training data set to learn a classifier h_t.
- *Optimization*: Use $\mathcal{D}_t$ to directly optimize a cost-sensitive function where some samples take on a larger cost than others.

Keep in mind that h is a weak decision rule, not a complex one such as a neural network. After h_t is generated we learned the weighted error given by

$$\epsilon_t = \sum_{i=1}^N \mathcal{D}_t(i)[\![h(\mathbf{x}_i) \neq y_i]\!] \tag{1}$$

where ϵ_t is a weighted loss using $\mathcal{D}_t$ and $[\![\cdot]\!]$ is the indicator function. Thus, not all errors are weighted equally. The weight, or influence, of h_t in the ensemble prediction is given by the weight assigned to the classifier, which is given by α_t. This assignment should be rather intuitive. If a classifier has a small error than assign it a larger weight otherwise give it a small weight. The natural logarithm is there for reason that we will see soon.

The final step in Adaboost is adjusting the weights of the instances/samples. At a high level, if an instance was misclassified then increase the weight on the instance, otherwise decrease it. If an instance is classified correctly then $y_i h_t(x_i)$ is positive, thus decreasing the weight of the sample, otherwise it is increased. To see this, notice the sign in the exponent. Also, note that Z_t is a constant to assure that $\mathcal{D}_{t+1}$ sums to 1.

The ensemble hypothesis is given by

$$H(x) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(x)\right) \tag{2}$$

which is simply a linear combination of the decision on a sample x.

2.1 Error on the Next Round

We begin by a close examination of the of the normalization factor using to re-normalize the instance weights. Recall that this is the sum of all the updates to instance weights with the exponential factor. In order to make the interoperation of the normalization factor clear, we separate the normalization term into the instances that were correctly classified by h_t and those that were incorrectly classified.

$$\begin{aligned} Z_t &= \sum_{i:y_i=h_t(\mathbf{x}_i)} \mathcal{D}_t(i)\mathrm{e}^{-\alpha_t} + \sum_{i:y_i\neq h_t(\mathbf{x}_i)} \mathcal{D}_t(i)\mathrm{e}^{\alpha_t} \\ &= \mathrm{e}^{-\alpha_t} \sum_{i:y_i=h_t(\mathbf{x}_i)} \mathcal{D}_t(i) + \mathrm{e}^{\alpha_t} \sum_{i:y_i\neq h_t(\mathbf{x}_i)} \mathcal{D}_t(i) \\ &= (1-\epsilon_t)\mathrm{e}^{-\alpha_t} + \epsilon_t \mathrm{e}^{\alpha_t} \end{aligned} \tag{3}$$

where we have used the definition of ϵ_t. Now let us examine the exponential term. Recall that $\alpha_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$. Using this definition, we have:

$$\mathrm{e}^{-\alpha_t} = \mathrm{e}^{-\frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}} = \mathrm{e}^{\log\sqrt{\frac{\epsilon_t}{1-\epsilon_t}}} = \sqrt{\frac{\epsilon_t}{1-\epsilon_t}}$$

and

$$\mathrm{e}^{\alpha_t} = \mathrm{e}^{\frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}} = \mathrm{e}^{\log\sqrt{\frac{1-\epsilon_t}{\epsilon_t}}} = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$$

Substituting these results into equation (3)

$$\begin{aligned} Z_t &= (1-\epsilon_t)\sqrt{\frac{\epsilon_t}{1-\epsilon_t}} + \epsilon_t\sqrt{\frac{1-\epsilon_t}{\epsilon_t}} \\ &= 2\sqrt{\epsilon_t(1-\epsilon_t)} \end{aligned} \tag{4}$$

which is a simple expression for the normalization factor. Now let us examine the error that h_t has on $\mathcal{D}_{t+1}$. Again, the error on $\mathcal{D}_{t+1}$ is simply the sum of the instance weight that h_t incorrectly classifies. That is:

$$\begin{aligned} \epsilon_{t+1} &= \sum_{i:y_i\neq h_t(\mathbf{x}_i)} \mathcal{D}_{t+1}(i)\mathrm{e}^{\alpha_t} \\ &= \frac{1}{Z_t} \sum_{i:y_i\neq h_t(\mathbf{x}_i)} \mathcal{D}_t(i)\mathrm{e}^{\alpha_t} \\ &= \frac{\epsilon_t \mathrm{e}^{\alpha_t}}{2\sqrt{\epsilon_t(1-\epsilon_t)}} \\ &= \frac{\epsilon_t\sqrt{\frac{1-\epsilon_t}{\epsilon_t}}}{2\sqrt{\epsilon_t(1-\epsilon_t)}} \\ &= \frac{1}{2} \end{aligned}$$

Thus concluding the derivation.

2.2 Derivation of Adaboost's Error Bound

In this section we show that the error of the ensemble learned using the Adaboost strategy, can the upper bounded and intuitively interpreted. We begin this discussion with a careful examination of the weights $\mathcal{D}_{t+1}(i)$. Recall from the pseudo-code is given by:

$$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i)}{Z_t} \exp\left(-\alpha_t y_i h_t(x_i)\right)$$

where Z_t now has a "nice" definition from (4). $\mathcal{D}_{t+1}(i)$ can be further decomposed by recognizing that $\mathcal{D}_t(i)$ can be expressed as $\mathcal{D}_{t-1}(i)$. Thus,

$$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_{t-1}(i)}{Z_t Z_{t-1}} \exp\left(-\alpha_t y_i h_t(x_i)\right) \exp\left(-\alpha_{t-1} y_i h_{t-1}(x_i)\right) \tag{5}$$

Continuing this decomposition gives us.

$$\begin{aligned}
\mathcal{D}_{t+1}(i) &= \frac{\mathcal{D}_{t-2}(i)}{Z_t Z_{t-1} Z_{t-2}} \exp\left(-\alpha_t y_i h_t(x_i)\right) \exp\left(-\alpha_{t-1} y_i h_{t-1}(x_i)\right) \exp\left(-\alpha_{t-2} y_i h_{t-2}(x_i)\right) \\
&= \frac{D_1(i)}{\prod Z_t} \exp\Big(-\sum \alpha_t y_i h_t(x_i)\Big) \\
&= \frac{1}{N \prod Z_t} \exp\Big(-\sum \alpha_t y_i h_t(x_i)\Big)
\end{aligned} \tag{6}$$

If the final classifier makes a mistake on the point x_i, then

$$\text{sign}\Big(\sum \alpha_t h_t(x_i)\Big) \neq y_i \text{ and sign}\Big(y_i \sum \alpha_t h_t(x_i)\Big) = -1$$

then $-y_i \sum \alpha_t h_t(x_i) \geq 0$. Hence,

$$\exp\Big(-y_i \sum \alpha_t h_t(x_i)\Big) \geq 1 \text{ if } H(x_i) \neq y_i \tag{7}$$

and since $\exp(x) \geq 0$, we have

$$\exp\Big(-y_i \sum \alpha_t h_t(x_i)\Big) \geq 0 \text{ if } H(x_i) = y_i$$

We are now ready to put an upper bound on the error by using serval of the observations and connections between terms that we have already defined/proved. Using all of these facts, we finish the proof by writing the misclassification error as

$$\widehat{\text{err}}(H) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}[H(x_i) \neq y_i] \leq \frac{1}{N} \sum_{i=1}^{N} \exp\Big(-y_i \sum \alpha_t h_t(x_i)\Big) \tag{8}$$

The inequality can be applied since we known that the indicator function returns "1" for a misclassified data point and we know that the inequality in (7) holds. To continue the proof, we observe that the inequality in (6) can be re-arranged to bound (8) further. Using this fact

$$\begin{aligned}
\widehat{\text{err}}(H) &= \frac{1}{N} \sum_{i=1}^{N} N \mathcal{D}_{t+1}(i) \prod_{t=1}^{T} Z_t \\
&= \sum_{i=1}^{N} \mathcal{D}_{t+1}(i) \prod_{t=1}^{T} Z_t \\
\widehat{\text{err}}(H) &\leq \prod_{t=1}^{T} Z_t
\end{aligned} \tag{9}$$

This gives us the final error bound of

$$\begin{aligned}\widehat{\text{err}}(H) &= \frac{1}{N}\sum_{i=1}^{N} N\mathcal{D}_{t+1}(i)\prod_{t=1}^{T} Z_t \\ &= \sum_{i=1}^{N}\mathcal{D}_{t+1}(i)\prod_{t=1}^{T} Z_t \\ &= \prod_{t=1}^{T} Z_t \\ &= 2^T\prod_{t=1}^{T}\sqrt{\epsilon_t(1-\epsilon_t)}\end{aligned}$$

2.3 Derivation of Adaboost's Weights

Finding the weights of Adaboost can easily be determined with a little bit of thinking about what it is we are trying to achieve and a little bit of Calculus I. Our goal is to have an ensemble that has a small error and (9) and we have a definition of Z_t that can be written as a function of the weights. Therefore, minimizing the bound on Z_t given us

$$\begin{aligned}Z_t &= (1-\epsilon_t)\exp(-\alpha_t) + \epsilon_t\exp(\alpha_t) \\ \frac{\mathrm{d}Z_t}{\mathrm{d}\alpha_t} &= -(1-\epsilon_t)\exp(-\alpha_t) + \epsilon_t\exp(\alpha_t) = 0\end{aligned}$$

We can use basic algebra and properties of exponentials to further simplify the expression for α_t

$$\begin{aligned}-(1-\epsilon_t)\exp(-\alpha_t) + \epsilon_t\exp(\alpha_t) &= 0 \\ [-(1-\epsilon_t)\exp(-\alpha_t) + \epsilon_t\exp(\alpha_t)]\exp(-\alpha_t) &= 0\cdot\exp(-\alpha_t) \\ -(1-\epsilon_t)\exp(-2\alpha_t) + \epsilon_t &= 0 \\ \exp(-2\alpha_t) &= \frac{\epsilon_t}{1-\epsilon_t}\end{aligned}$$

Therefore,

$$\alpha_t = \frac{1}{2}\log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

3 Applications and Implementations

In the Figure 1, AdaBoost algorithm is used to separate out two non-linearly separable classification composed of two "Gaussian quantiles" clusters. As seen in the code, we first create two classes with two features with values $(0, 1)$. We use 200 estimators of DecisionTreeClassifier in the scikit-learn package to obtain the decision boundary seen in the Figure . We plot the decision scores for the points belonging to both classes. where the magnitude of score is the degree of likeliness that the point belongs to the class label. From the decision scores we can predict that samples with scores greater than zero are classified as B and those with less than zero are classified as A.

Figure 2 takes us through a visualization of the adaboost algorithm and how decision boundaries are created along with the assignments of the weights for each iteration. The classifier that is chosen is only 51% accurate, which means we will be classifying samples with error just less than 50%. For m=1, the classifier misclassifies 5 points (2 Blue, 3 Red), which have increased weights in the next

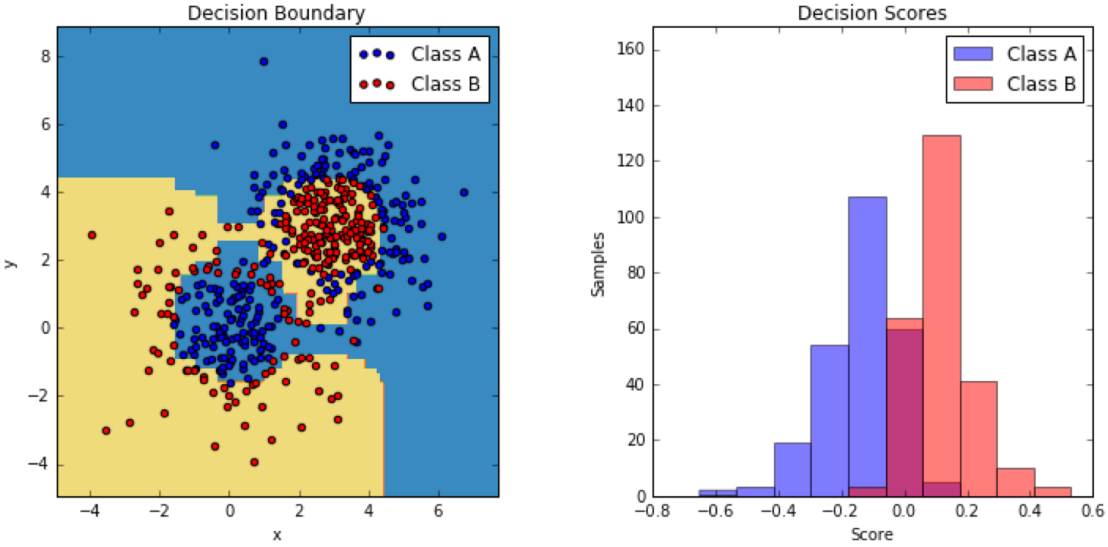

Figure 1: Two Class AdaBoost decision boundary and Decision Scores for each class

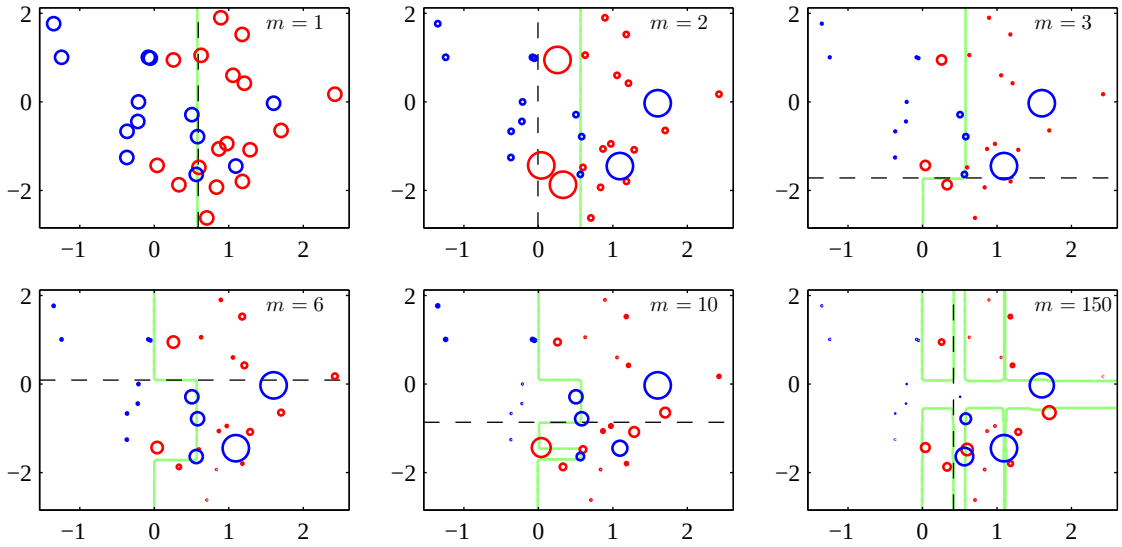

Figure 2: Examples showing the decision boundary and the weights assigned to each sample point based on previous error.

iteration, while the other points have reduced weights in the next distribution. Now the classifier choses a boundary such that the classification error is least for the new distribution of weights. Hence we see that the decision boundary is such that it misclassifies only 2 Blue points which is the least that the classifier can achieve. Thus the weights of the misclassified sample points is further increased and those of other points is decreased in the next distribution. For m=3 a horizontal decision boundary is chosen as a vertical one will error out many Red sample points. Thus with each iteration the decision boundary is modified to get a non-linear boundary for the sample. Final hypothesis is a weighted average of the decision boundary and the alpha values for them as seen in equation (2).

Acknowledgement

An initial draft of this document was prepared by students in *Engineering Applications of Machine Learning and Data Analytics* in 2018.

References

[Bishop, 2006] Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer.

[Duda et al., 2001] Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification*. John Wiley & Sons, Inc., 2nd edition.

[Freund and Shapire, 1997] Freund, Y. and Shapire, R. (1997). A decision-theoretic generalization of online learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139.

[Freund and Shapire, 1999] Freund, Y. and Shapire, R. (1999). A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771–780.

[Murphy, 2012] Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.

[Shapire and Freund, 2012] Shapire, R. and Freund, Y. (2012). *Boosting: Foundations and Algorithms*. The MIT Press.