Chapter 8
Ensemble Methods

Slides (pdf) Video (Panopto)

So far in our exploration of Classification, we have covered Logistic Regression, Bayes Theorem, and Decision Trees as candidates for modeling data that is intended to be separated into classes. With the Decision Tree approach, we can model our data in a much more interpretable way, but Decision Trees are not known for reaching stellar accuracy. In this chapter we will discuss the modifications done unto Decision Trees to improve their performance. Specifically, we will discuss the Ensemble Method.

8.1 Ensemble Method

The basic idea behind the Ensemble Method is to combine a collection of models in hopes that their unity will achieve better performance rather than designing a new model from scratch. This collection of models is referred to as a **model ensemble**.

Definition 8.1: Model Ensemble

A **model ensemble** is a collection of (generally weak) models that are combined in such a way to create a more powerful model.

With trees, this is done in via either the **Random Forest (Bagging)** method, or **AdaBoost (Boosting)**.

8.1.1 Random Forest

The Random Forest method, also called Bagging, utilizes a collection of Decision Trees. Each decision tree casts a "vote" for a prediction based on an input and the ensemble predicts the majority vote of all of its trees. Each tree is built from a subset of the training dataset by random sampling the training dataset with replacement. This technique is also called **bootstrapping**.

When training the decision trees on the bootstrapped samples, the goal is to build very deep overfitting trees. Each tree thus has very high variance. As multiple models are used for the final prediction in the ensemble, the overall outcome is a model with low bias. This is because if many overfitting models are averaged out, the result is a model that does not overfit to any one sample so the ensemble has low bias and lower variance. So, while each tree is a robust modeling of a data sample, the united model ensemble is more powerful.

Example(s)

Random Forest has many applications in image processing and object detection. Microsoft used the Random Forest modeling technique in their Kinect system to identify the "pose" of a person from the depth camera.

The advantage of using a Random Forest is that it is versatile, and has many applications in classification, regression, and clustering. It is also low-maintenance and hence easily interpretable: There are few to none hyper-parameters in most cases, and generally more trees gives us better performance. Random Forest is also efficient, because the trees, as separate components, may be trained in parallel. So instead of waiting

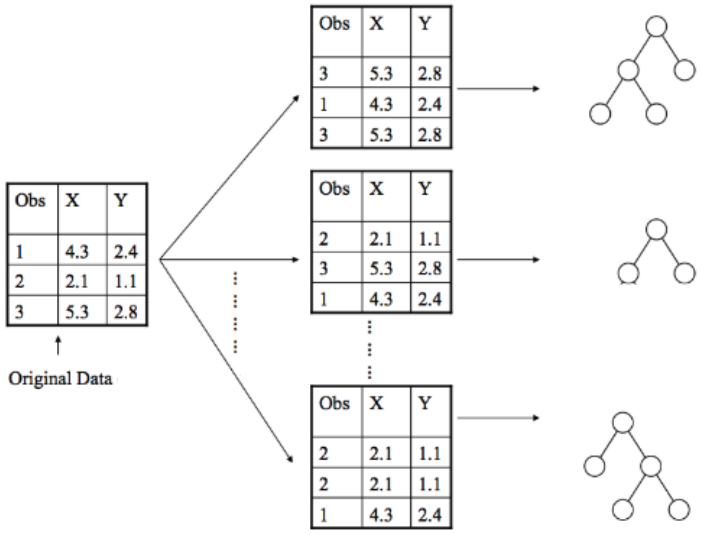

Figure 8.1: A subset of the original dataset is randomly selected with replacement for every tree we intend to build.

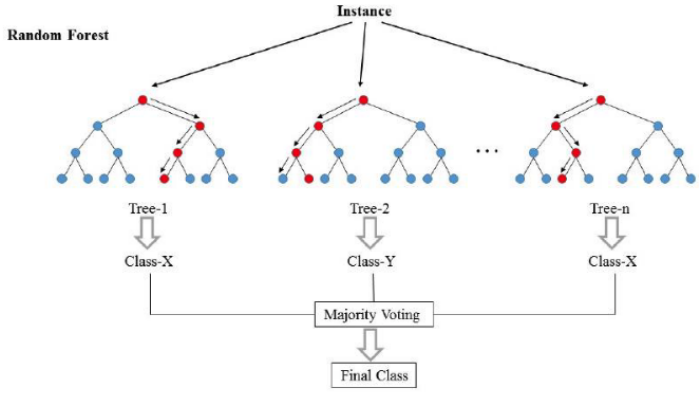

Figure 8.2: Each tree predicts its own final decision based on the input and the majority prediction is selected by the ensemble.

for a single complex model to train one step at a time as we may do with other models, we can assign each tree of the ensemble to a different machine so that multiple trees are trained simultaneously.

8.1.2 AdaBoost

Recall that a decision stump in a tree is a node that splits into multiple decisions. When we say "stump" we will refer to a tree of one-level. As opposed to growing very deep and overfitting trees in Random Forest, Adaboost utilizes stumps. So, the model ensemble for Adaboost is a collection of one-level trees. Each of these trees are assigned a weight. A weighted majority of votes from these decision stumps is what determines the ensemble's final prediction.

A key difference between Random Forest and AdaBoost is that AdaBoost is sequential, while Random Forest can train each of its trees in parallel. This is because in AdaBoost, each stump belongs to a certain order, and the first stump's decision impacts the second, which impacts the third, and so forth.

8.1.2.1 Training AdaBoost

When training AdaBoost, the error of the previous stump affects the learning of the next stump. To do this, two types of weights are stored:

1. $\hat{w}_t$: the weight of each stump t.
2. α_i: the weight of each datapoint i.

The training process of AdaBoost is as follows:

1. Decide on our stumps: the more features our data has, the more stumps we will need in AdaBoost since we will have to make at least one decision from each feature. We will call the total set of stumps T, and each stump in T is t.
2. Initialize the weights for all the datapoints in our training set α_i to be equal to each other.
3. For every stump t in T:
   (a) Learn a final prediction $\hat{f}_t(x)$ based on α, the weights of the datapoints used in t.
   (b) Compute the error of t's prediction $\hat{f}_t(x)$. As AdaBoost introduces weights to datapoints, we must compute a **weighted** error:
   $$WeightedError(f_t) = \frac{\sum_{i=1}^{n} \alpha_i || \{\hat{f}_t(x_i) \neq y_i\}}{\sum_{i=1}^{n} \alpha_i} \tag{8.1}$$
   This equation is simply taking the weighted sum of all the misclassified datapoints and then dividing it by the total weight of all the datapoints. In the weighted error, every datapoint's unique weight is used to compute the total error.
   (c) Based on the error rate of prediction $\hat{f}_t(x)$, update model t's weight $\hat{w}_t$. If t yields a low error, then we want $\hat{w}_t$ to be greater, but if the error is high, then the weight should be low. This is because we want more better performing models to have a higher influence in the final decision of the ensemble. The formula for computing the model weight is:
   $$\hat{w}_t = \frac{1}{2} ln(\frac{1 - WeightedError(\hat{f}_t)}{WeightedError(\hat{f}_t)}) \tag{8.2}$$
   (d) Recompute the datapoint weights α. AdaBoost will *increase* the weight of those datapoints it misclassified, and *decrease* the weights of those it classified correctly. This way, misclassified datapoints are more sensitively accounted for in the training of the next stump. This is the notation for updating the weights:
   $$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t} & if \hat{f}_t(x_i) = y_i \\ \alpha_i e^{\hat{w}_t} & if \hat{f}_t(x_i) \neq y_i \end{cases}$$
4. After we have computed $\hat{f}_t(x)$ for every stump t, we take the sign of the model outputs' weighted sum like so:
   $$\hat{y} = \hat{F}(x) = sign(\sum_{t=1}^{T} \hat{w}_t \hat{f}_t(x)) \tag{8.3}$$

8.1.2.2 Real-Valued Features

If we are splitting real-valued numeric features in our AdaBoost method, the algorithm is more or less the same but the splits must happen dependent on the weights of each numerical value.

8.1.2.3 Normalizing Weights

Generally, the weights for some datapoints can get very large or very small in magnitude due to how the data is laid out. If our numbers are on wildly different scales, we may run in to problems when running AdaBoost on our machine due to the finite precision of computers when it comes to real numbers. To resolve this, we normalize all the weights to sum to 1:

$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{j=1}^{n} \alpha_j} \tag{8.4}$$

8.1.2.4 Visualizing AdaBoost

Recall that all the weights for the datapoints are initialized to be equal to each other. After training on one stump, we compute new weights α_i based on the errors of $\hat{f}_1$.

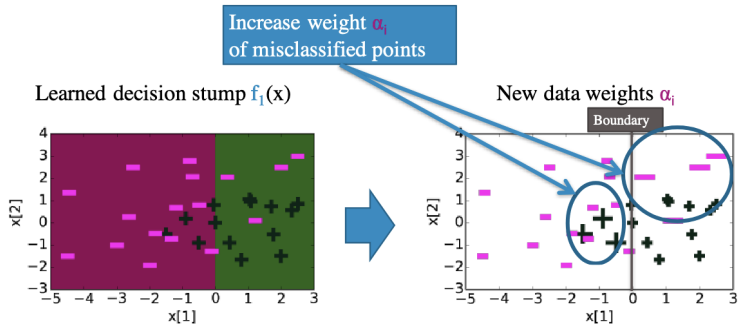

Figure 8.3: After training on the first stump, misclassified datapoints are given a greater weight (drawn larger).

The new weights α_i are used to learn the best stump that minimizes the weighted classification error. Then, update the weights again based on the error from $\hat{f}_2$.

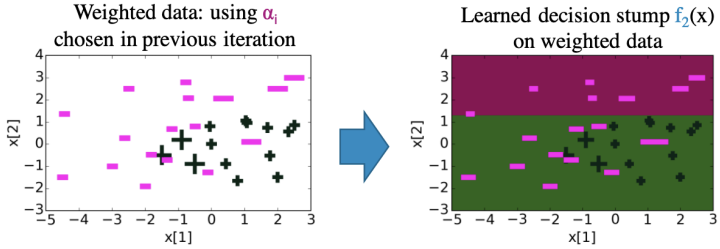

Figure 8.4: $\hat{f}_2$ is predicted based on the weights computed in the previous iteration.

If we plot what the predictions would be for each point, we get something that looks like this:

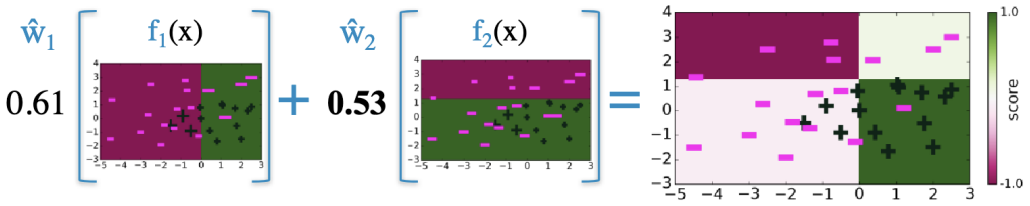

Figure 8.5: The weighted sum of the models $t = 1$ and $t = 2$ yields the weighted decision boundary on the right.

Eventually, we can achieve a training error of 0 with a large enough set of weak learners.

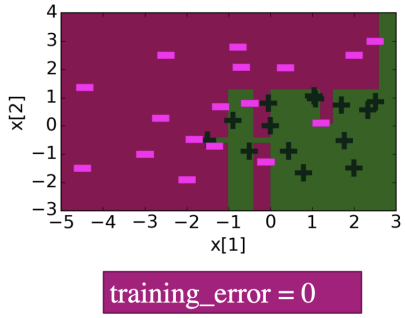

Figure 8.6: Decision boundary for $t = 30$.

The example above illustrates that AdaBoost is still capable of overfitting!

8.1.2.5 AdaBoost Theorem

The core idea behind the Ensemble Method is that a collection of weaker models combined will yield a more powerful model. This is the case with AdaBoost, where a series of weak single-stump trees can gradually reduce the training error of the overall model. Let's compare AdaBoost to a standard Decision Tree.

While a Decision Tree can significantly reduce the error of its training set, we see that this is at the expense of severe overfitting. So a Decision Tree lacks the ability to generalize to unseen test data. However, due to the internal mechanisms of AdaBoost, we avoid this severe overfitting and are more likely to achieve a lower test error. AdaBoost is also capable of overfitting as we have seen in the previous example, but in general the test error will stabilize. To determine the best T to avoid overfitting and underfitting, we must treat T as a hyper-parameter and compare different T's with the intent of minimizing the validation error.

8.1.2.6 Applications of AdaBoost

Boosting, AdaBoost, and other variants like Gradient Boost are some of the most successful models to date. They are extrmeley useful in computer vision, as they are the standard for face detection. Most industry Machine Learning systems use model ensembles.

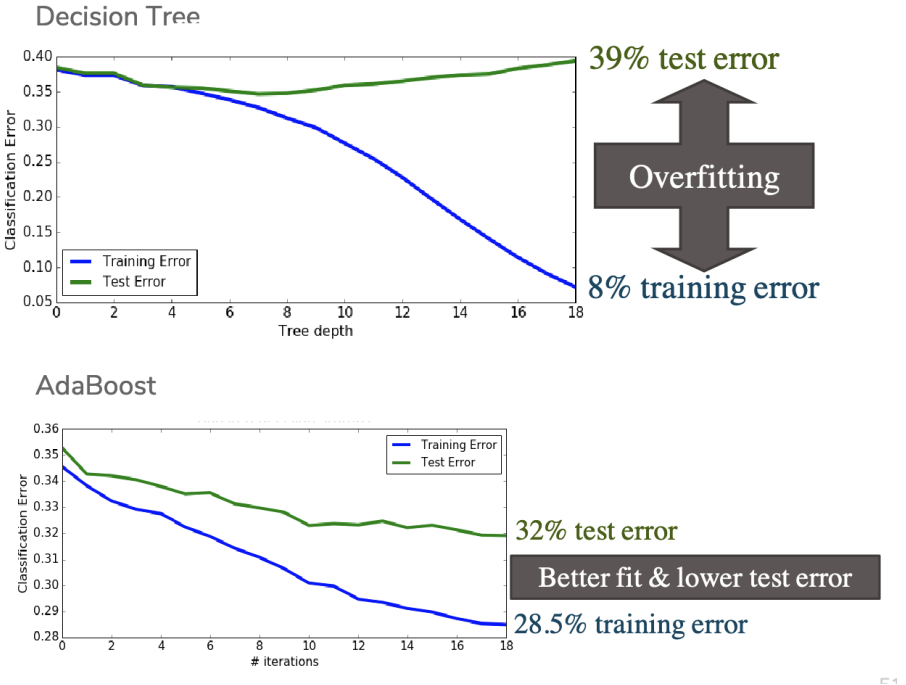

Figure 8.7: The train and test error curves for a standard Decision Tree and AdaBoost.

To conclude, AdaBoost is a powerful model ensemble technique and it typically does better than Random Forest with the same number of trees. However, while you do not have to tune parameters for AdaBoost (besides selecting T), boosting is sequential, so trees cannot be trained in parallel.