Classification Week 5 – Boosting

“Can a set of weak learners be combined to create a stronger learner?” – Kearns and Valiant 1988

Simple (weak) learners, like logistic regression with simple features, decision stumps or shallow decision trees have traits we like

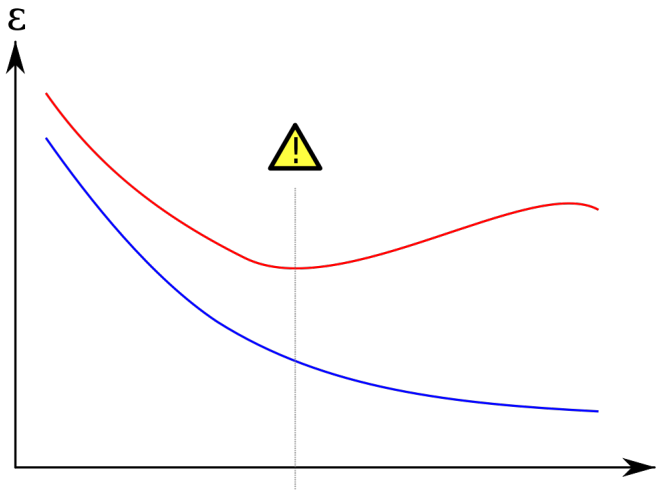
- They have low variance, so they tend to not overfit
- They can be learned quickly

But they have some problems

- Individual models have high bias – they tend to not fit the data as well as more complex models.

Our prior strategy for improving a weak classifier is to make it more complex, so it fits the training data better. We used validation to avoid overfitting.

Here is an illustration from this Wikipedia of how training error (the bottom, blue line) can decrease, but generalization error (the top, red line) can increase;



Error versus Model Complexity

The vertical line shows where the true error begins to climb, because the model becomes overfit to the training error.

In 1990 Rob Schapire and others addressed the Kearns-Valiant conjecture by producing an algorithm called boosting which uses multiple weak learners and a voting scheme to improve accuracy. This is now a very important and popular technique in machine learning and is the default method for many kinds of analysis.

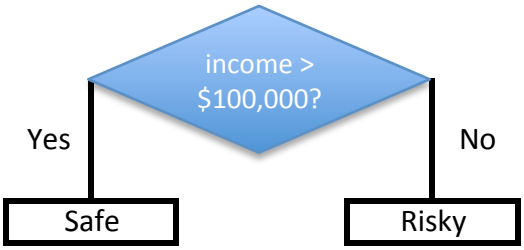
Ensemble classifiers

We can denote a weak classifier as f(x), so

$$\hat{y} = f(x)$$

which can take the values +1 or -1

We can use a decision stump as a weak classifier.



Boosting combines such simple classifiers into an ensemble classifier. The outputs of each individual weak classifier are combined to produce the final prediction;

Given the following input data x_i

income	Credit	Savings	Market
$120,000	Bad	$50,000	Good

This produces these outputs for each classifier;

f1(x) = +1 **f2(x) = -1** **f3(x) = -1** **f4(x) = +1**

Those individual outputs are then combined using coefficients which are learned. The results is an ensemble model denoted F(x)

$$F(x_i) = sign\big(w_1 f_1(x_i) + w_2 f_2(x_i) + w_3 f_3(x_i) + w_4 f_4(x_i)\big)$$

This is a weighted voting scheme that outputs +1 or -1. For example, given the following coefficients and the prior inputs

w	f(x)	output
2	1	2
1.5	-1	-1.5
1.5	-1	-1.5
0.5	1	0.5
	sum =	-0.5
	output =	-1

The sum ends up as -0.5, so we would predict -1. So predicting the output for a set of ensemble classifiers is;

$$\hat{y}_i = sign\left(\sum_{t=1}^{T} w_t f_t(x_i)\right)$$

So the goal with ensemble classifiers is the same as our other classifiers, given the data point x_i, predict the output y_i, which is +1 or -1.

In our prior classifiers we took a set of data, X and we created features h(x) from this data, then learned a set of coefficients w where each coefficient w_j was used to weight a feature $h_j(x)$.

With ensemble classifiers, we now learn two things;

- f(x), a set of weak classifiers
- w, a set of coefficients where each coefficient w_t is used to weight the output of a classifier $f_t(x)$

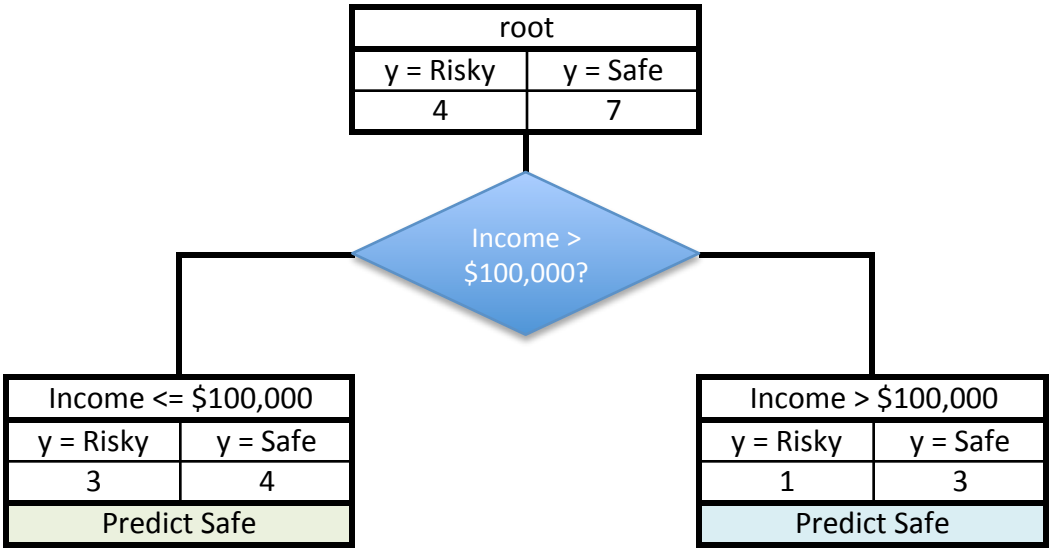
In effect, our set of weak classifiers are the features, but now we learn them from data as well as their weights.

Boosting

When doing boosting, our workflow includes learning the weak learners. In the case of classification, it means learning individual decision stumps. Given the following data;

Credti	Income	y
A	$130,000	Safe
B	$80,000	Risky
C	$110,000	Risky
A	$110,000	Safe
A	$90,000	Safe
B	$120,000	Safe
C	$30,000	Risky
C	$60,000	Risky
B	$95,000	Safe
A	$60,000	Safe
A	$98,000	Safe

We may learn the following decision stump;



On both sides of the decision we have predicted Safe, so it's not much a decision. Also, there are mistakes. We can do better. So what the boosting algorithm does is to learn another decision stump. This time around, it tries to do better with those rows that were misclassified – those rows are shown below;

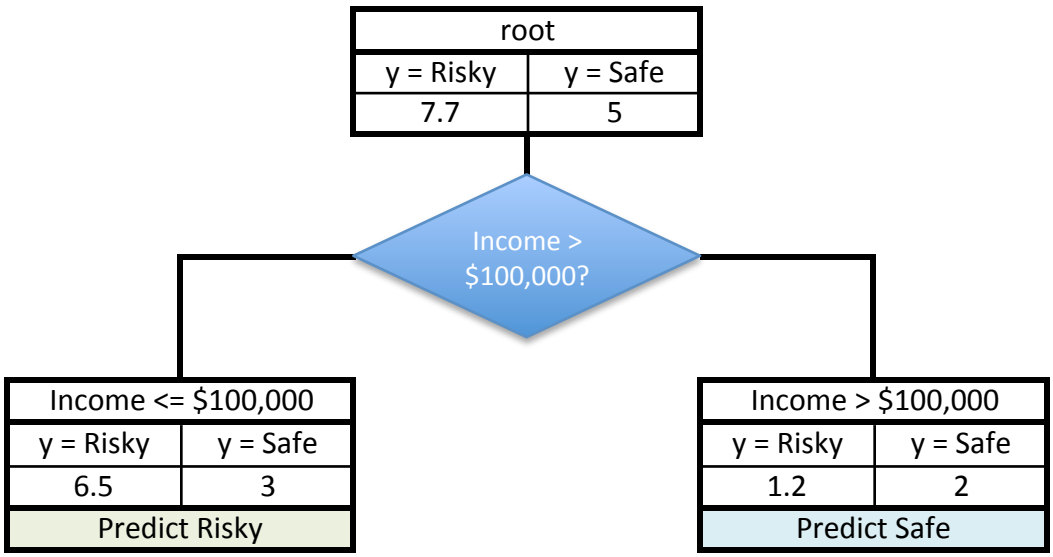
Credti	Income	y
A	$130,000	Safe
B	$80,000	Risky
C	$110,000	Risky
A	$110,000	Safe
A	$90,000	Safe
B	$120,000	Safe
C	$30,000	Risky
C	$60,000	Risky
B	$95,000	Safe
A	$60,000	Safe
A	$98,000	Safe

Since we know these rows are mistakes, then we can focus on 'fixing' them in the next decision stump that we learn. Boosting does this by adding a weight to each data point which determines it's relative importance in the decision. Larger weights make the data point more important. In the first decision stump, we can think of those weights as being exactly 1.0 for all points. But in this second decision stump, we might apply the following weights to the data points;

Credti	Income	y	weight
A	$130,000	Safe	0.5
B	$80,000	Risky	1.5
C	$110,000	Risky	1.2
A	$110,000	Safe	0.8
A	$90,000	Safe	0.6
B	$120,000	Safe	0.7
C	$30,000	Risky	3.0
C	$60,000	Risky	2.0
B	$95,000	Safe	0.8
A	$60,000	Safe	0.7
A	$98,000	Safe	0.9

We can see that those data points that were misclassified in the previous decision stump now have weights > 1 and the other data points that were classified correctly have weights < 1. If effect, we are making the rows that were incorrectly classified in this decision stump more important, so the next decision stump will focus on getting them right.

Remember in our prior decision tree work that we calculate the score as the sum of the data points. Now we calculate it as the sum of the data weights. In this case, the second decision stump looks like this;



Now the Income <= $100,000 predicts Risky, because the prior misclassified rows count more.

The weight is designated as α, the Greek letter alpha, so the weight associated with row i of the input data is α_i.

This use of weighted data is applicable to many machine learning algorithms. Remember the calculation of $w^{(t+1)}$ in the gradient ascent for logistic regression algorithm;

$$w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta \sum_{i=1}^{N} h_j(x_i)(1[y_i = +1] - P(y = +1|x_i, w))$$

To this, we simply multiply the weight α_i to the data point $h_j(x_i)$;

$$w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta \sum_{i=1}^{N} \alpha_i h_j(x_i)(1[y_i = +1] - P(y = +1|x_i, w))$$

Learning Ensemble Classifiers

Learning the ensemble is a series of iterative steps

- Training Data
- Learn First Classifier
- Predict $\hat{y} = sign(f_1(x))$
- Weighted Data (higher weights were $f_1(x)$ is wrong)
- Learn Second Classifier
- Predict $\hat{y} = sign(\hat{w}_1 f_1(x) + \hat{w}_2 f_2(x))$
- …
- Weighted Data
- Learn T-th Classifier
- $\hat{y}_i = sign(\sum_{t=1}^{T} w_t f_t(x_i))$

AdaBoost

(Freund and Schapire 1999)

- Start with uniform weights for all data points.
  - Can be an any number, 1 is common
  - We will use 1/N as it simplifies some things.
- For t = 1 to T
  - Learn $f_t(x)$ with data weights α_i
  - Compute coefficients $\hat{w}_t$
- Final model predicts using ensemble $\hat{y} = sign(\sum_{t=1}^{T} \hat{w}_t f_t(x))$

We need to know how to calculate two things

- We need to know how to re-calculate the data weights α_i before learning each classifier.
- We need to know how to compute the coefficient for each classifier. Trusted, Accurate classifiers have a larger coefficient.

Weighted Error

In AdaBoost, the coefficient for each classifier will be large if the classifier is accurate

- $\hat{w}_t$ is large if $f_t(x)$ has low training error
- $\hat{w}_t$ is small if $f_t(x)$ has high training error

So, as you would hope, accurate classifiers we weighted more heavily in the final vote than are inaccurate classifiers.

Importantly, training error is against the weighted data. So points that were wrong in prior iterations will themselves have higher α_i in iteration t, and so will contribute more to the training error that we are minimizing. In effect, minimization concentrates more on the points we got wrong in the last iteration.

Total Weight of all data points is

$$\sum_{i=1}^{N} \alpha_i$$

Total weight of mistakes is

$$\sum_{i=1}^{N} \alpha_i \times 1(\hat{y}_i \neq y_i)$$

Where

- α_i is the weight of i-th data point
- $1(\hat{y}_i \neq y_i)$ is 1 if prediction does **not** equal output
- $1(\hat{y}_i \neq y_i)$ is 0 if prediction **does** equal output

The weighted error is the fraction of the weight of mistakes:

$$WeightedError = \frac{total\ weight\ of\ mistakes}{total\ weight\ of\ all\ data\ points}$$

WeightedError

- Best value is 0.0, no weighted error
- Random classifier would produce weighted error of ~0.5
- Worst value is 1.0, everything in error

Note, if the classifier predicts everything exactly wrong (it's weightedError is close to 1.0), then if we can simply do the opposite, it becomes a good classifier. So an error around 0.5 is actually a bigger problem, because it is harder to correct.

AdaBoost formula for calculating coefficient $\hat{w}_t$ of classifier $f_t(x)$:

$$\hat{w}_t = \frac{1}{2}\ln\left(\frac{1 - weightedError(f_t(x))}{weightedError(f_t(x))}\right)$$

Here is what this formula produces for the 3 types of classifiers;

Classifier	weightedError(ft)	(1 - weightedError(ft)) / weightedError(ft)	wt
Good	0.01	99.00	2.30
Random	0.50	1.00	0.00
Bad	0.99	0.01	-2.30

See how the weight for the bad classifier is a negative weight, effectively inverting the output of the classifier and turning it into a good classifier.

Also note that the random classifier's weight is close to zero – it is really bad so we don't care about it, so it's weight in the ensemble is very small.

Reweighing data to focus on mistakes

AdaBoost updates the weight of data points α_i based on where the classifier $f_t(x)$ makes mistakes. More specifically;

- If $f_t(x_i)$ is correct, decrease α_i
- If $f_t(x_i)$ is wrong, increase α_i

AdaBoost formula for updating the weight of data points α_i:

$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\widehat{w}^{(t)}}, if\ f_t(x_i) = y_i \\ \alpha_i e^{\widehat{w}^{(t)}}, if\ f_t(x_i) \neq y_i \end{cases}$$

This formula has these effects;

f(x)==y?	Coefficient w	Multiply α by e^w	Implication for data point
Correct	2.30	0.10	Decrease importance
Correct	0.00	1.00	Keep the same
Wrong	0.00	1.00	Keep the same
Wrong	2.30	9.97	Increase Importance

So if we are right or wrong for a random-ish classifier (one that has a weight near zero), we don't change much. If have an accurate classifier that classifies a given point correctly then we decrease that data point's importance in the next iteration. If we have an accurate classifier that classifies a data point incorrectly, then we weight that point much higher in the next iteration.

Normalizing weights

As the AdaBoost algorithm proceeds, we are constantly changing the weights of data points. For those that are incorrectly predicted, we increase the weight by multiplying by a number > 1 and for those that were correctly predicted we decrease the weight by a number < 1. This can lead to some issues;

- For data points that are consistently predicted incorrectly, the weight can get very, very large.
- For data points that consistently predicted correctly, the weight can get very, very small.
- This can lead to numerical instability in the algorithm.

To avoid numerical instability, we normalize the weights of the data points in each iteration so that they add up to one. So for each data point, we divide it by the sum of all the weighted data points.

$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{i=1}^{N} \alpha_i}$$

So this is the final AdaBoost algorithm with normalization:

- Start with uniform weights for all data points.
  - We will use 1/N so weights are normalized to start
- For t = 1 to T
  - Learn $f_t(x)$ with data weights α_i
    - Pick the decision stump with the lowest weighted training error using data weights α_i
  - Compute coefficients $\widehat{w}_t$
  - Normalize data weights α_i
- Final model predicts using ensemble $\hat{y} = sign(\sum_{t=1}^{T} \widehat{w}_t\, f_t(x))$

Applying AdaBoost

The complexity of an ensemble is proportional to the number if individual classifiers. So in the case of AdaBoost, this is determined by the number of iterations through the algorithm.

If we run the AdaBoost algorithm long enough, we will achieve zero training error, but of course this will not generalize and so we end up with an overfit model.

Learning boosted decision stumps with AdaBoost
In each iteration t, we choose a decision stump f_t. We consider all features when choosing the stump, even if they have been used for prior decision stumps in the ensemble. We try a stump using each feature and choose the stump with the lowest training error on the weighted data (lowest weighted training error).

Once the decision stump is chosen, we calculate it's coefficient w_t in the ensemble;

$$\widehat{w}_t = \frac{1}{2}\ln\left(\frac{1 - weightedError(f_t(x))}{weightedError(f_t(x))}\right)$$

Next we calculate the weights of the data for the next iteration. We do this by making a prediction for each data point using the new decision stump we have learned. We then compare the prediction to the known value and update the weight of the data point based on correct or incorrect prediction;

For each data point i = 1 to N;

$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\widehat{w}^{(t)}}, if\ f_t(x_i) = y_i \\ \alpha_i e^{\widehat{w}^{(t)}}, if\ f_t(x_i) \neq y_i \end{cases}$$

We then normalize the weighted data;

totalWeightedData = $\sum_{i=1}^{N} \alpha_i$
For each data point i = 1 to N;

$$\alpha_i \leftarrow \frac{\alpha_i}{\text{totalWeightedData}}$$