

Machine Learning



Machine Learning

Lecture: Ensemble Learning - Bagging

Ted Scully

Introduction to Ensemble Learning

- ▶ In machine learning, ensemble methods utilize/**combine the results** from a number of learning algorithms to obtain a **better predictive performance** than could be obtained from any single one of the learning algorithms learning algorithms
- ▶ Instead of learning one model, we learn several and combine them. Such combinations are known as model ensembles.
 - ▶ Note the base learners could be the same algorithm or different
- ▶ They are among the most powerful techniques in machine learning. Typically improve accuracy (they can often significantly improve performance)
- ▶ **Downside** is increased **algorithmic and model complexity** (take much longer to run than a single model).

Introduction to Ensemble Learning

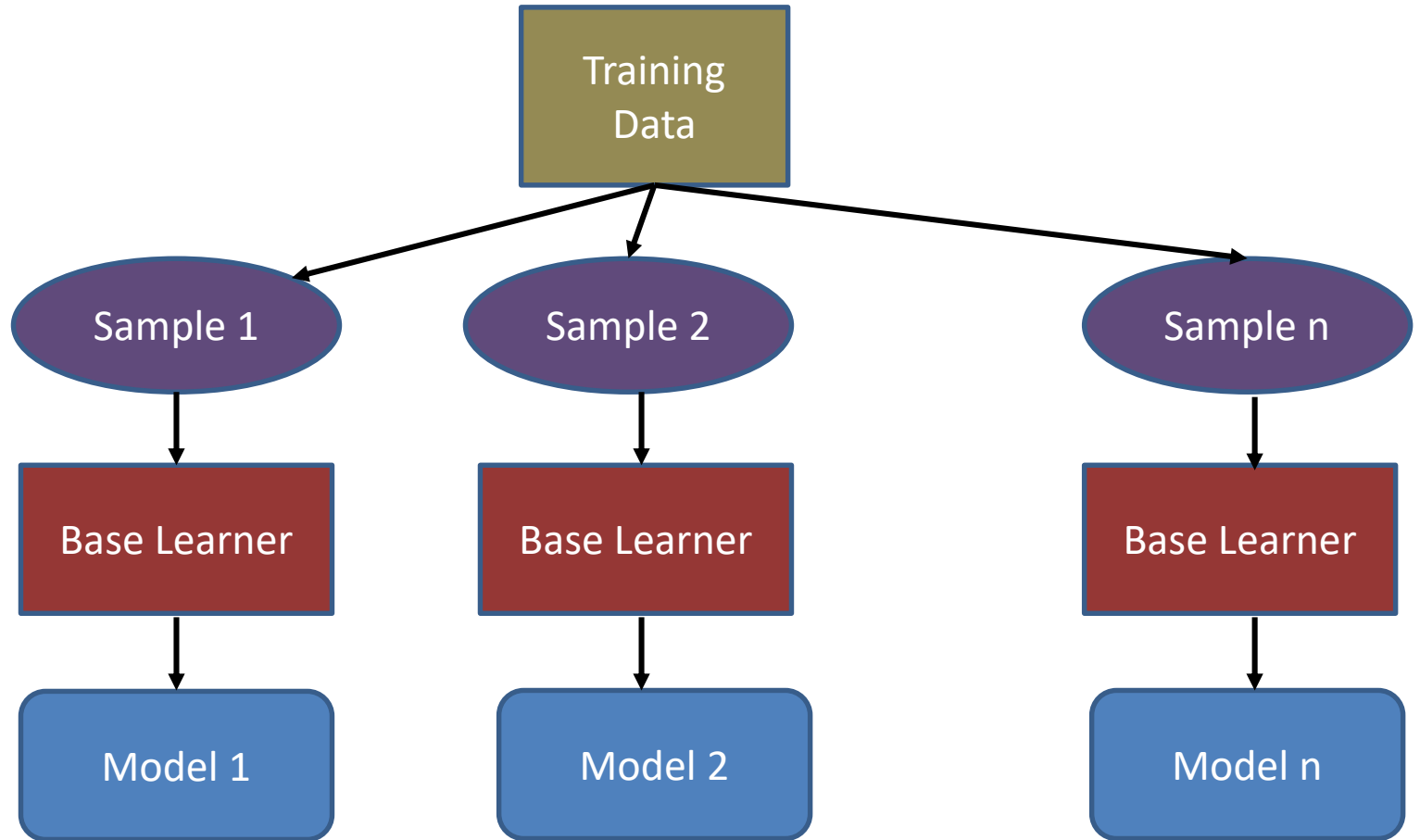
- ▶ Ensemble methods have the following two factors in common:
 - ▶ They construct **multiple, diverse predictive models** from adapted versions of the training data.
 - ▶ They **combine the predictions** of these models in some way, often by simple averaging or voting
- ▶ We will look at two different types of ensembles
 - ▶ Bagging (RandomForests)
 - ▶ Boosting (AdaBoost)

Introduction to Bagging

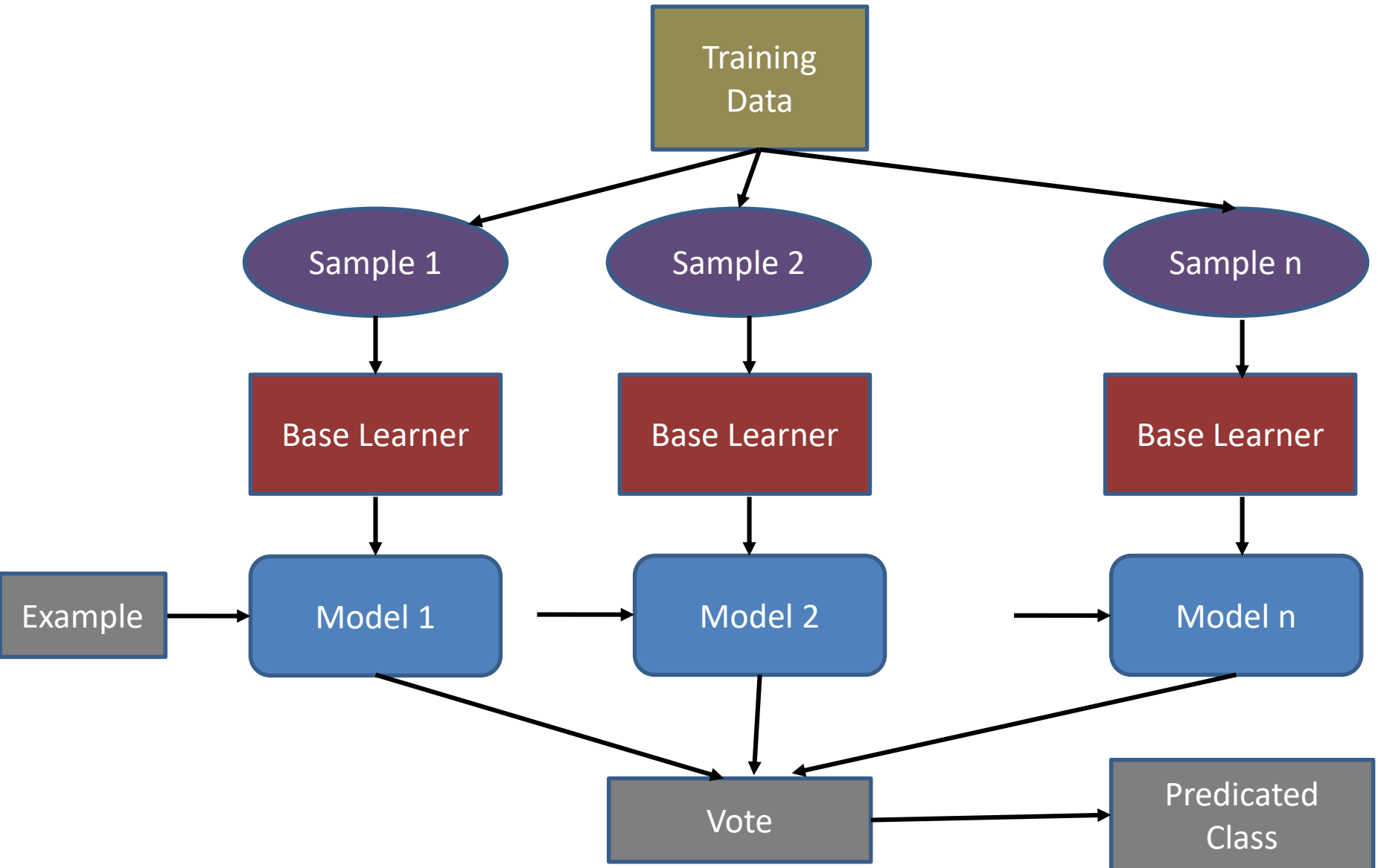
- ▶ **Bagging**, short for “**bootstrap aggregating**” is a simple but effective technique that creates different models on **random samples** of the original data set.
 - ▶ These samples are taken uniformly with replacement and are known as **bootstrap samples**.
- ▶ Learn one model for each sample
- ▶ Because samples are taken at random with replacement each sample will normally contain **duplicates** and hence some of the original data will be missing from your sample.
- ▶ This technique has shown that **statistically each of the samples look very similar** (variance, mean, etc)

Creating a Bootstrap Sample

Introduction to Bagging



Introduction to Bagging



Decision Trees and Subspace Sampling

- ▶ Bagging is particularly effective with **decision tree models**.
- ▶ Decision trees can be **very sensitive to the underlying data**. Changing a small percentage of the training data could produce a very different looking tree (in particular the lower levels).
- ▶ Other methods can also be used to inject additional diversity into the decision trees. One common method is when selecting a node for a tree using a **random subset of the features**. For each node we build in the tree we base on decision on a different subset of features each time.
- ▶ This is sometimes called **random subspace sampling**.

Random Subspace Sampling

Random Forests

- ▶ Random Forest is an ensemble techniques that combines both bagging and subspace sampling.

Input: data set D ; ensemble size T ; subspace dimension d

Function RandomForest (D, T, d)

for $t=1$ to T **do**:

- Build a bootstrap sample D_t from D by sampling $|D|$ data points with replacement
- buildRandomTree(D_t, d)

Function BuildRandomTree(D_t, d)

- **At each node:**
 - f = Select d features at random and reduce dimensionality of D_t accordingly
 - Split on best feature in f

return learned tree

Machine Learning



Machine Learning

Lecture: Ensemble Learning - Boosting

Ted Scully

Boosting

- ▶ Boosting is one of the most famous and successful categories of machine learning algorithms
- ▶ Basis came from a proof which showed that a weak learner (a learning algorithm that is only slightly better than chance – $50\% + \epsilon$ accurate) then you can boost that weak learner into a strong learner.
- ▶ This theory was then applied in Bell labs and lead to very significant jumps in accuracy.
- ▶ Over the following slides we are going to focus on AdaBoost.

Boosting - AdaBoost

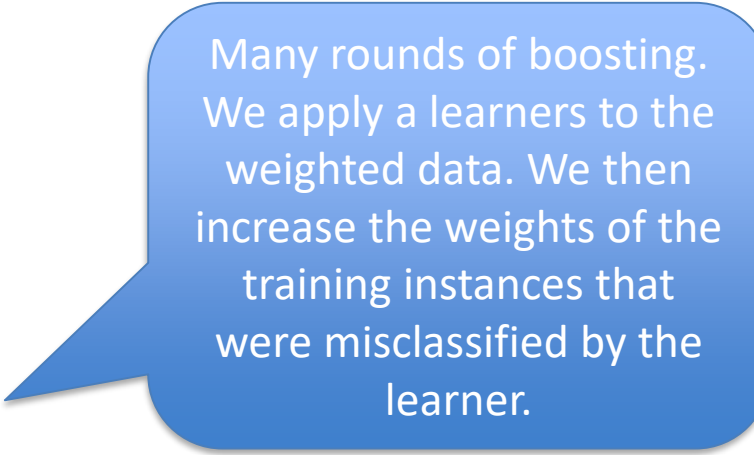
- ▶ As with a random forest, in AdaBoost we create many models. However, there are a number of important differences between random forest and AdaBoost:
 1. In a RF we generate bootstrap samples for each base learner. In contrast with AdaBoost we apply **weights to each of the training instances**. When we build an individual base learner we pass the training data and the weights to the base learner. The base learner will build a model that is biased towards those training instances that have a higher weight. In other words the model will put more importance on correctly classifying higher weighted training instances.
 2. In a RF each individual model is fully grown decision tree. However, in boosting each model is what we refer to as a **decision stump**. A decision stump is a decision tree that contains just one node (a root node) connected to leaf nodes. A decision stump is a weak learner.

Boosting - AdaBoost

- ▶ As with a random forest, in AdaBoost we create many models. However, there are a number of important differences between random forest and AdaBoost:
- 3. In a RF we combine the predicts of all base learners equally when arriving at a final prediction. In boosting we assign a **weight to each base learner**. The base learners with a higher weight will contribute more to the prediction of the final target compared to those with a lower weight.
- 4. Finally in a RF we create each learner independently from every other learner (the order with which we create each decision tree doesn't matter). In AdaBoost the process of creating each learning is **sequential and interconnected**. The creation of one decision stump base learner influences the creation of the next decision stump base learner. More specifically the errors by the first stump influence how we create the second stump and son on.

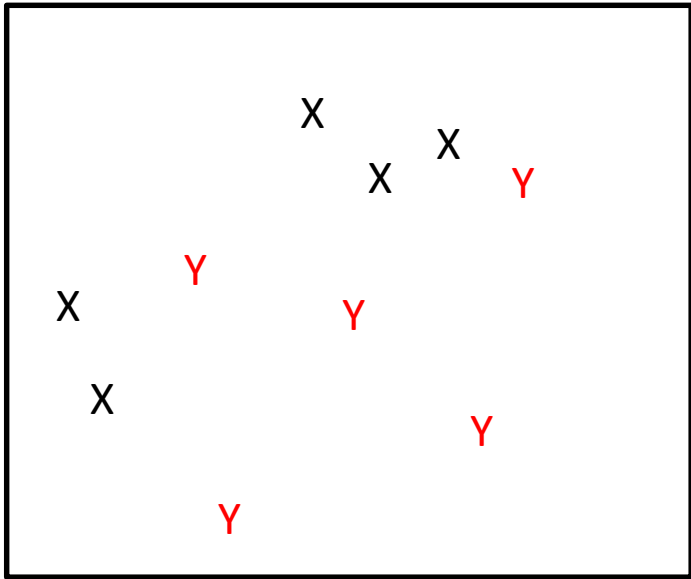
Boosting

- ▶ 1. Maintain a vector of weights for the training instances
- ▶ 2. Initialize with uniform weights
- ▶ Loop
 - ▶ Build ML Model with weighted examples
 - ▶ Adjust weights
- ▶ Combine models by weighted voting

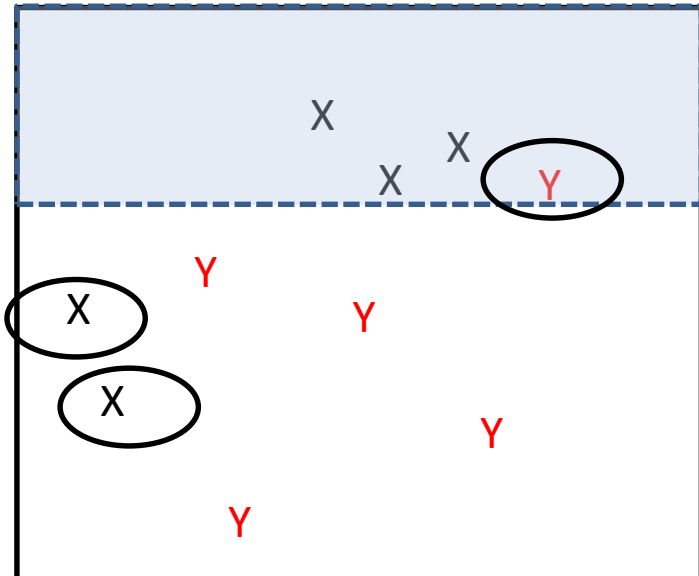


Many rounds of boosting. We apply a learners to the weighted data. We then increase the weights of the training instances that were misclassified by the learner.

Boosting Example

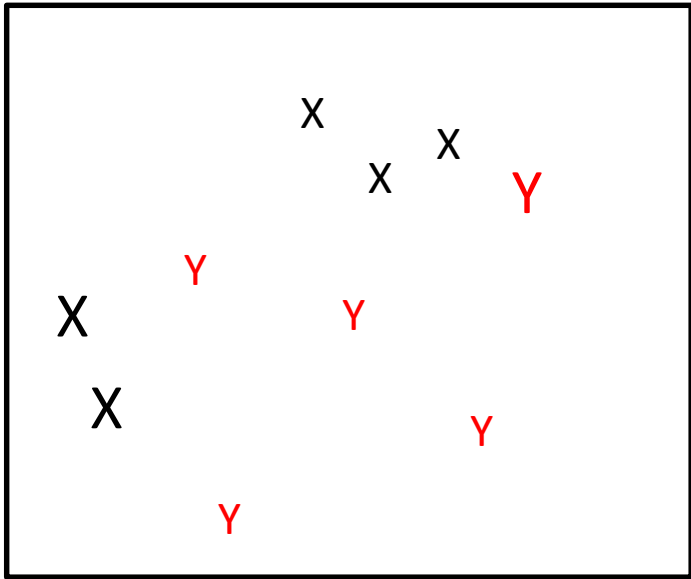


In this example we depict a dataset with two classes X and Y.

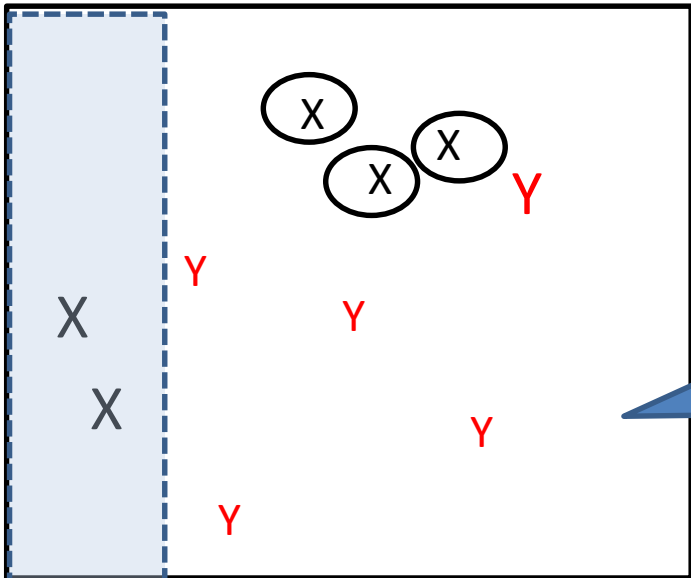


When we apply the algorithm we get the following model. Notice it classifies one instance of Y incorrectly and two instances of X. Boosting will focus the next trainer of those instances by increasing the weight of the instances we got wrong and decreasing the weight of those that we got right.

Boosting Example

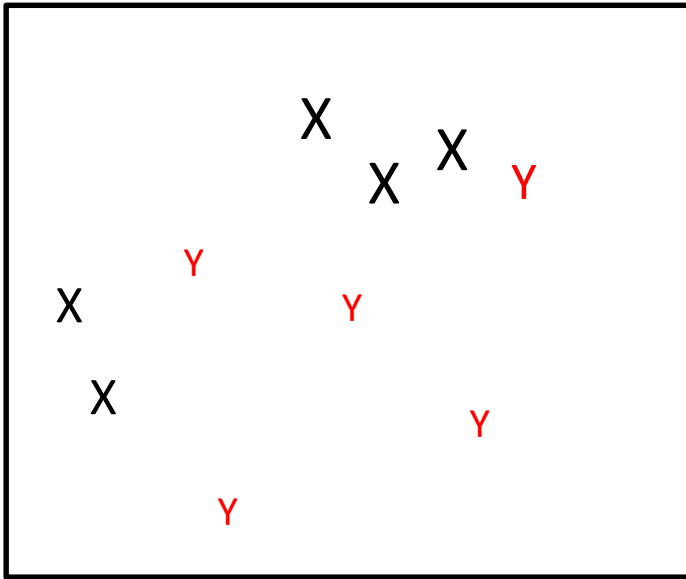


Notice the instances that we got wrong are depicted as being visually larger and those we got right slightly smaller. We train another classifier on this new data

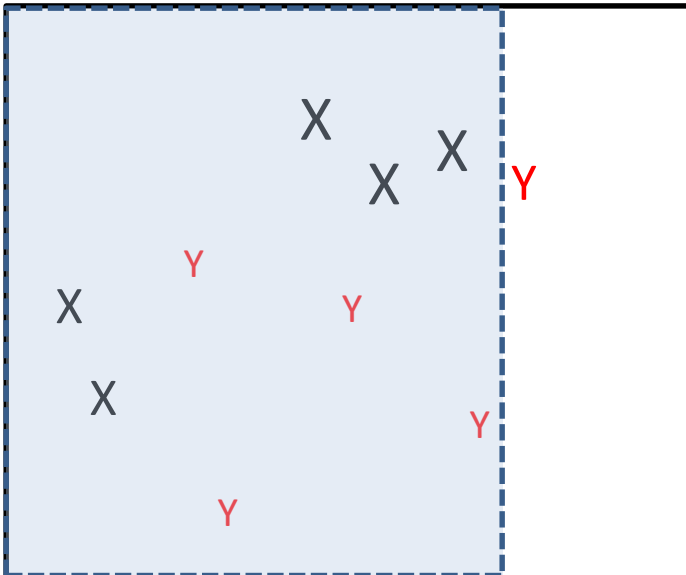


When we apply the weak learner we get the following model. Notice the model is more focused on getting the instances with the higher weight right. (Notice it gets 3 of the X instances wrong so we increase the weights of these examples and decrease the weight of the instances got right for the next round)

Boosting Example



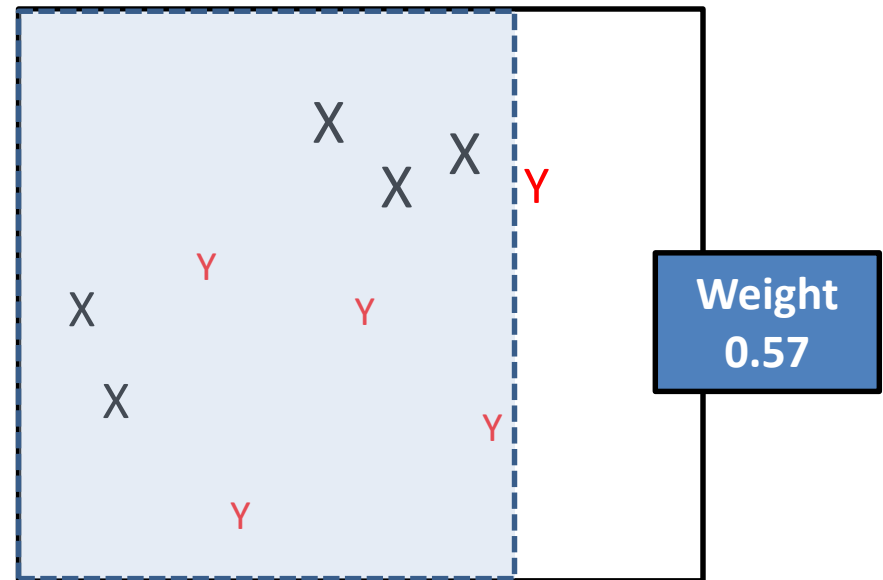
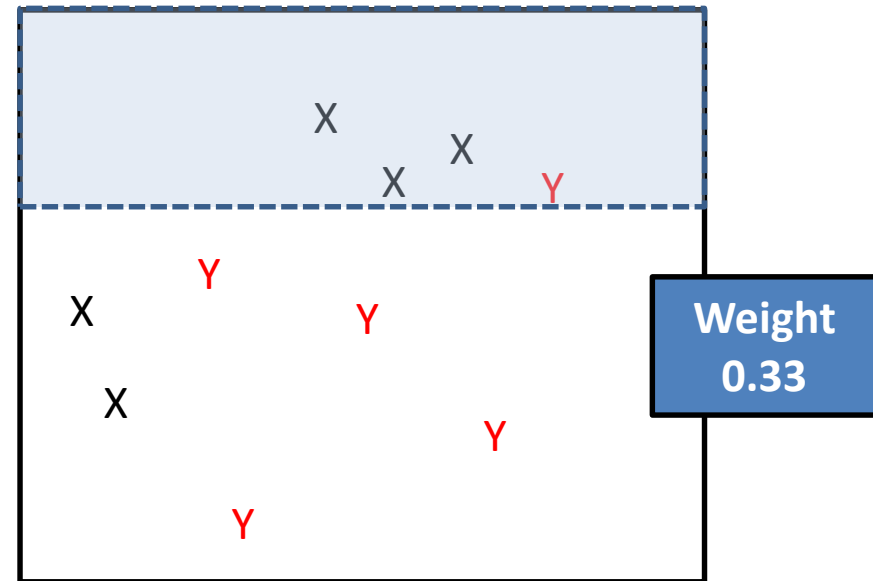
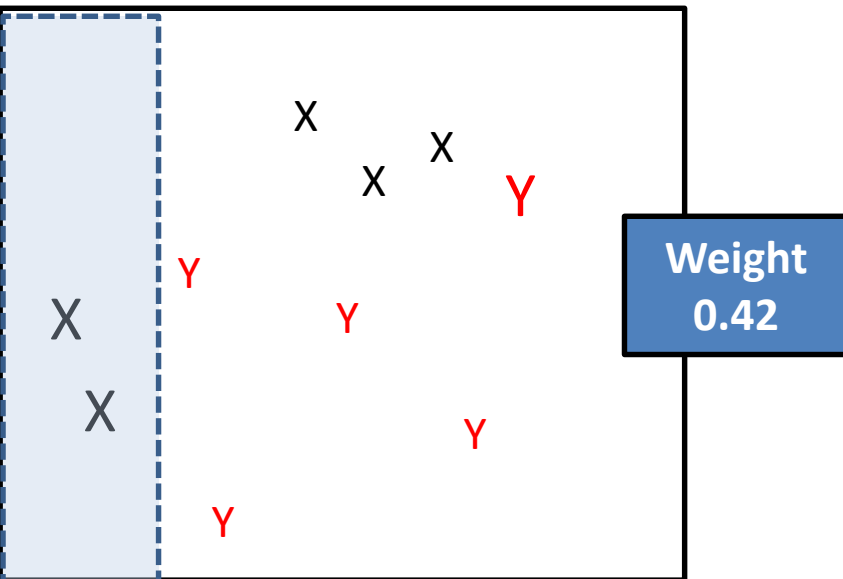
Notice here we have increased the weight of the instances classified incorrectly and decrease the weight of those classified correctly.



The classifier is focused primarily on getting the larger weighted data points correct. It gets all the X instances correct but makes a mistake on a number of small weight Y instances.

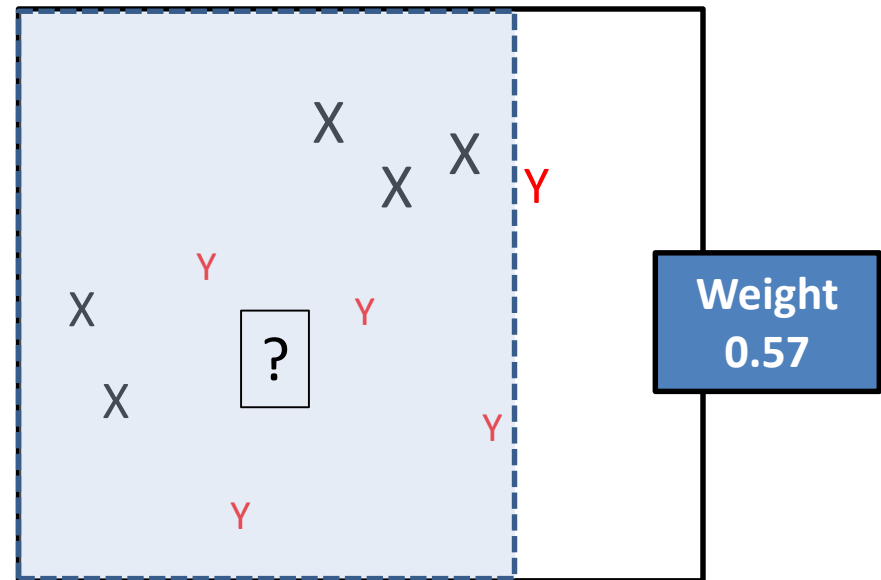
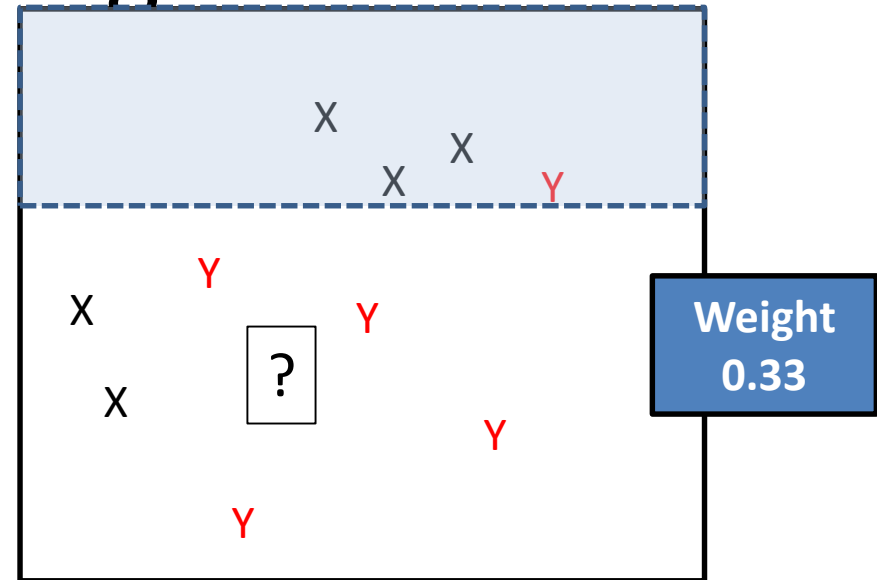
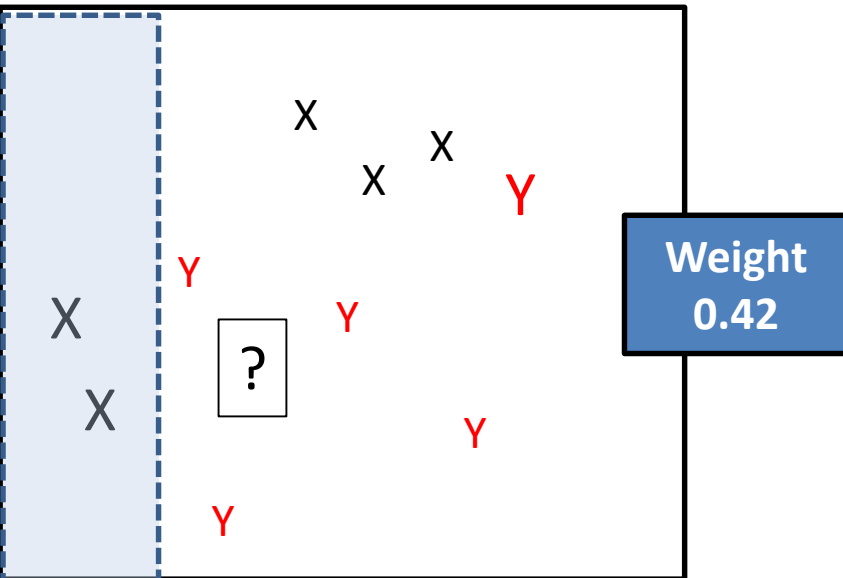
Boosting

- ▶ Each of the classifiers we created is assigned a weight and their weighted combination becomes an overall predictor.
- ▶ Classifiers that are more accurate have a higher weight



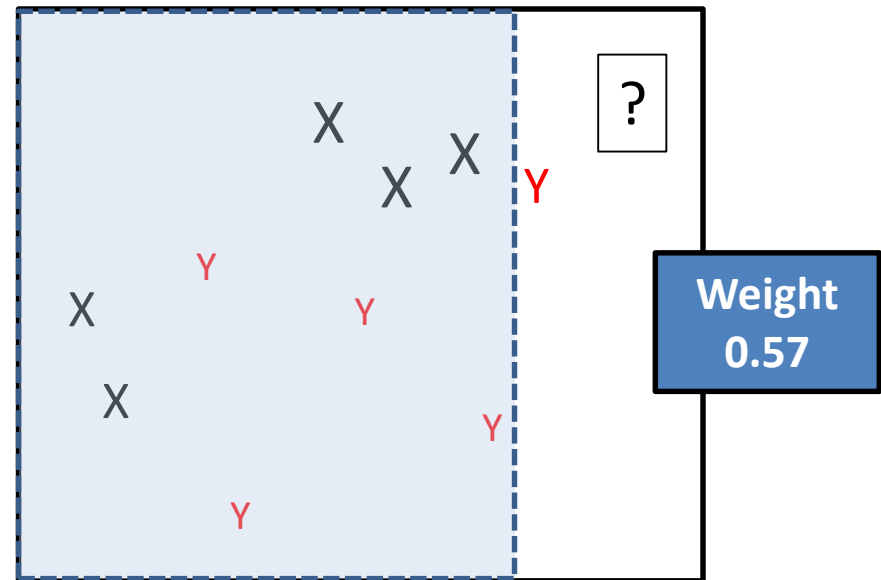
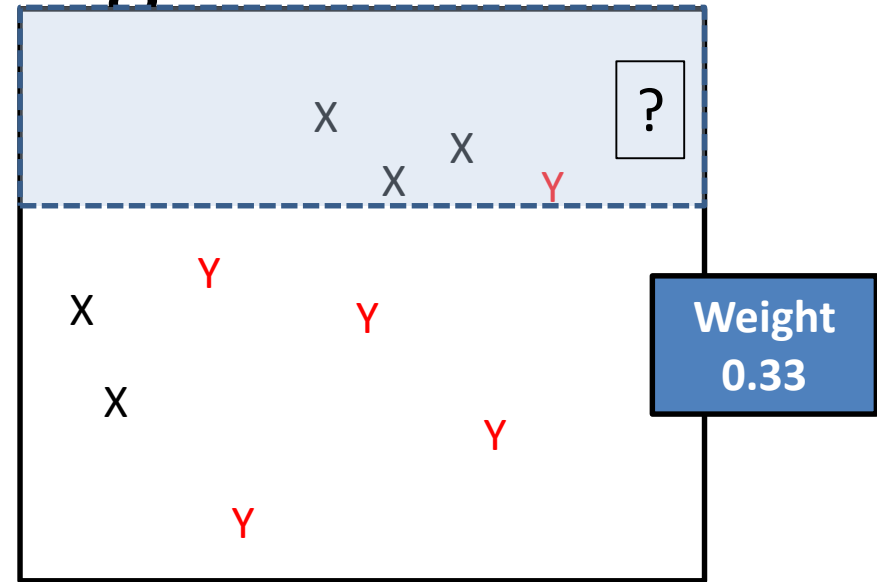
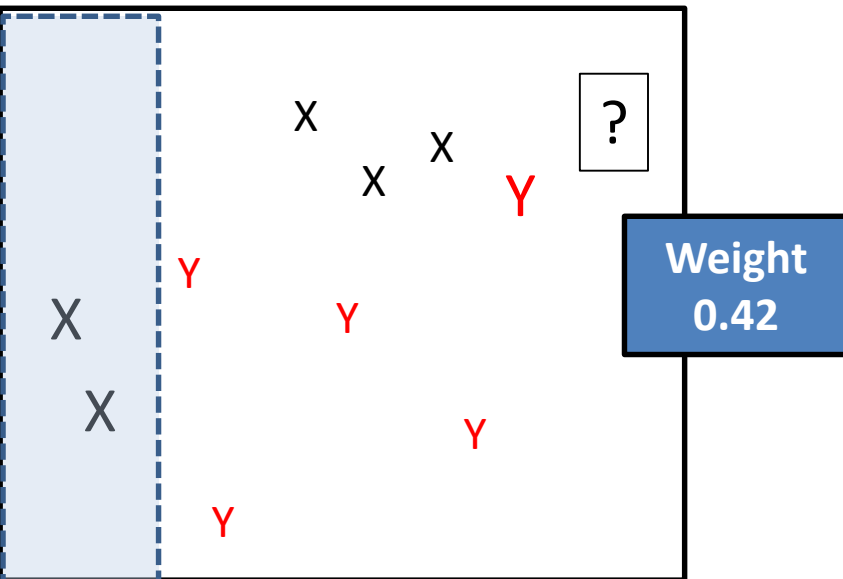
Boosting

- ▶ What class is assigned to the query instance below.
- ▶ Vote for Y: $0.42 + 0.33 = 0.75$
- ▶ Vote for X: 0.57
- ▶ Classed as Y



Boosting

- ▶ What class is assigned to the query instance below.
- ▶ Vote for Y: $0.42 + 0.57 = 0.99$
- ▶ Vote for X: 0.33
- ▶ Classed as Y



Pseudocode for AdaBoost

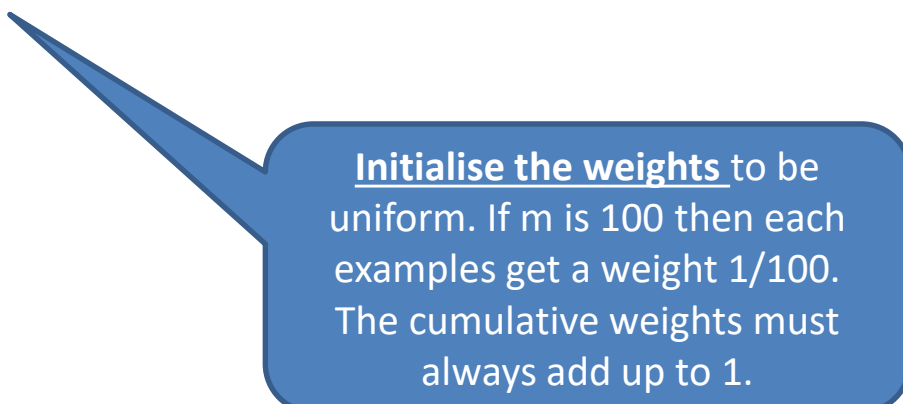
ADABOOST(S , $Learn$, k)

S : Training set $\{(x_1, y_1), \dots, (x_m, y_m)\}$, $y_i \in Y$

$Learn$: Learner(S , weights)

k : # Rounds

For all i in S : $w_1(i) = 1/m$



Initialise the weights to be uniform. If m is 100 then each examples get a weight $1/100$. The cumulative weights must always add up to 1.

Pseudocode for AdaBoost

ADABOOST(S , $Learn$, k)

S : Training set $\{(x_1, y_1), \dots, (x_m, y_m)\}$, $y_i \in Y$

$Learn$: Learner(S , weights)

k : # Rounds

For all i in S : $w_1(i) = 1/m$

| Training Instance | w1 |
|-------------------|-----|
| x^1 | 1/7 |
| x^2 | 1/7 |
| x^3 | 1/7 |
| x^4 | 1/7 |
| x^5 | 1/7 |
| x^6 | 1/7 |
| x^7 | 1/7 |

Initialise the weights to be uniform. If m is 100 then each examples get a weight $1/100$. The cumulative weights must always add up to 1.

ADABOOST($S, Learn, k$)

S : Training set $\{(x_1, y_1), \dots, (x_m, y_m)\}$, $y_i \in Y$

$Learn$: Learner(S , weights)

k : # Rounds

For all i in S : $w_1(i) = 1/m$

For $r = 1$ to k do

For all i : $p_r(i) = w_r(i) / \sum_i w_r(i)$

$h_r = Learn(S, p_r)$

Next build a ML model. Notice the ML model uses both the training examples and their weights in order to build the model. The learner algorithm and it returns a hypothesis (model) h for the round r . Remember this model put more importance on the instances with higher weights

At the start of each new round we renormalize the weights (so they will again all weight up to 1).

For the round r we get $p_r(i)$ for every training example i and this is just the weight of i divided by the cumulative weight of all i .

ADABOOST($S, Learn, k$)

S : Training set $\{(x_1, y_1), \dots, (x_m, y_m)\}$, $y_i \in Y$

$Learn$: Learner(S , weights)

k : # Rounds

For all i in S : $w_1(i) = 1/m$

For $r = 1$ to k do

For all i : $p_r(i) = w_r(i) / \sum_i w_r(i)$

$h_r = Learn(S, p_r)$

$\epsilon_r = \sum_i p_r(i) \mathbf{1}[h_r(i) \neq y_i]$

Next we measure the error rate of that classifier.

So we add up the weights of every example where the classifier got the prediction of the class wrong.

Remember this value will be between 0 and 1.

Think of epsilon as the total probability of the current examples the classifier is getting wrong.

| Training Instance | w1 | Base Learner 1 Accuracy |
|-------------------|-----|-------------------------|
| X^1 | 1/7 | F |
| X^2 | 1/7 | T |
| X^3 | 1/7 | T |
| X^4 | 1/7 | T |
| X^5 | 1/7 | F |
| X^6 | 1/7 | F |
| X^7 | 1/7 | T |

$$\varepsilon_1 = \frac{1}{7} * 3 = 0.42$$

ADABOOST($S, Learn, k$)

S : Training set $\{(x_1, y_1), \dots, (x_m, y_m)\}$, $y_i \in Y$

$Learn$: Learner(S , weights)

k : # Rounds

For all i in S : $w_1(i) = 1/m$

For $r = 1$ to k do

For all i : $p_r(i) = w_r(i) / \sum_i w_r(i)$

$h_r = Learn(S, p_r)$

$\epsilon_r = \sum_i p_r(i) \mathbf{1}[h_r(i) \neq y_i]$

If $\epsilon_r > 1/2$ then

$k = r - 1$

Exit

If the error is greater than 0.5 then the algorithm is in trouble because it is not even a weak learner. So the algorithm will just stop at this specific round and will return $r-1$ rounds of boosting.

ADABOOST($S, Learn, k$)

S : Training set $\{(x_1, y_1), \dots, (x_m, y_m)\}, y_i \in Y$

$Learn$: Learner(S , weights)

k : # Rounds

For all i in S : $w_1(i) = 1/m$

For $r = 1$ to k do

For all i : $p_r(i) = w_r(i) / \sum_i w_r(i)$

$h_r = Learn(S, p_r)$

$\epsilon_r = \sum_i p_r(i) \mathbf{1}[h_r(i) \neq y_i]$

If $\epsilon_r > 1/2$ then

$k = r - 1$

Exit

$\beta_r = \epsilon_r / (1 - \epsilon_r)$

Beta measures the incorrect prediction weights relative to the correct prediction weights.

Beta for the current round of boosting is the weights of the error expressed as a fraction of the weights of those instances correctly classified.

This value will always be between 0 and 1.

As epsilon gets smaller then so too does beta. Clearly we would like epsilon to be as large as possible

ADABOOST($S, Learn, k$)

S : Training set $\{(x_1, y_1), \dots, (x_m, y_m)\}, y_i \in Y$

$Learn$: Learner(S , weights)

k : # Rounds

For all i in S : $w_1(i) = 1/m$

For $r = 1$ to k do

For all i : $p_r(i) = w_r(i) / \sum_i w_r(i)$

$h_r = Learn(S, p_r)$

$\epsilon_r = \sum_i p_r(i) \mathbf{1}[h_r(i) \neq y_i]$

If $\epsilon_r > 1/2$ then

$k = r - 1$

Exit

$\beta_r = \epsilon_r / (1 - \epsilon_r)$

The final weight we assign to each base learner is directly related to the value of beta for that base learner. The higher the value of beta the lower the weight associated with the base learner (and vica versa)

Beta measures the incorrect prediction weights relative to the correct prediction weights.

Beta for the current round of boosting is the weights of the error expressed as a fraction of the weights of those instances correctly classified.

This value will always be between 0 and 1.
As epsilon gets smaller then beta gets smaller. As the value of epsilon gets larger so does beta.

| Training Instance | w1 | Base Learner 1 Accuracy |
|-------------------|-----|-------------------------|
| X^1 | 1/7 | F |
| X^2 | 1/7 | T |
| X^3 | 1/7 | T |
| X^4 | 1/7 | T |
| X^5 | 1/7 | F |
| X^6 | 1/7 | F |
| X^7 | 1/7 | T |

$$\varepsilon_1 = \frac{1}{7} * 3 = 0.42$$

$$\beta_1 = \frac{0.42}{(1-0.42)} = 0.72$$

ADABOOST($S, Learn, k$)

S : Training set $\{(x_1, y_1), \dots, (x_m, y_m)\}$, $y_i \in$

$Learn$: Learner(S , weights)

k : # Rounds

For all i in S : $w_1(i) = 1/m$

For $r = 1$ to k do

For all i : $p_r(i) = w_r(i) / \sum_i w_r(i)$

$h_r = Learn(S, p_r)$

$\epsilon_r = \sum_i p_r(i) \mathbf{1}[h_r(i) \neq y_i]$

If $\epsilon_r > 1/2$ then

$k = r - 1$

Exit

$\beta_r = \epsilon_r / (1 - \epsilon_r)$

For all i : $w_{r+1}(i) = w_r(i) \beta_r^{1 - \mathbf{1}[h_r(x_i) \neq y_i]}$

In this step we assign a new weight to every single instance.

The weight of i for the new round is calculated by multiplying the weight of i for the current round by Beta for the current round.

We only update the weight if my model is getting the example correct. This has the impact of decreasing the weight on all those instances that were correctly classified by this base learner.

Put another way. The weights of the incorrect examples are untouched and the weights of the correct examples are reduced by multiplying them by Beta.

| Training Instance | w1 | Base Learner 1 Accuracy | w2 |
|-------------------|------------|-------------------------|------------|
| X^1 | 1/7 (0.14) | F | 1/7 (0.14) |
| X^2 | 1/7 (0.14) | T | 0.059 |
| X^3 | 1/7 (0.14) | T | 0.059 |
| X^4 | 1/7 (0.14) | T | 0.059 |
| X^5 | 1/7 (0.14) | F | 1/7 (0.14) |
| X^6 | 1/7 (0.14) | F | 1/7 (0.14) |
| X^7 | 1/7 (0.14) | T | 0.059 |

$$\varepsilon_1 = \frac{1}{7} * 3 = 0.42$$

$$\beta_1 = \frac{0.42}{(1-0.42)} = 0.72$$

ADABOOST($S, Learn, k$)

S : Training set $\{(x_1, y_1), \dots, (x_m, y_m)\}, y_i \in \mathcal{Y}$

$Learn$: Learner(S , weights)

k : # Rounds

For all i in S : $w_1(i) = 1/m$

For $r = 1$ to k do

For all i : $p_r(i) = w_r(i) / \sum_i w_r(i)$

$h_r = Learn(S, p_r)$

$\epsilon_r = \sum_i p_r(i) \mathbf{1}[h_r(i) \neq y_i]$

If $\epsilon_r > 1/2$ then

$k = r - 1$

Exit

$\beta_r = \epsilon_r / (1 - \epsilon_r)$

For all i : $w_{r+1}(i) = w_r(i) \beta_r^{1 - \mathbf{1}[h_r(x_i) \neq y_i]}$

We repeat the iterative process k times and this enables us produce k learners for our ensemble.

However, we mentioned that at inference time when we take an unseen instance the prediction of each base model is weighted.

This weight is directly related to the beta.

The weight for the model is round r is:

$$\log \frac{1}{\beta_r}$$

| Training Instance | w1 | Base Learner 1 Accuracy | w2 |
|-------------------|------------|-------------------------|------------|
| X^1 | 1/7 (0.14) | F | 1/7 (0.14) |
| X^2 | 1/7 (0.14) | T | 0.059 |
| X^3 | 1/7 (0.14) | T | 0.059 |
| X^4 | 1/7 (0.14) | T | 0.059 |
| X^5 | 1/7 (0.14) | F | 1/7 (0.14) |
| X^6 | 1/7 (0.14) | F | 1/7 (0.14) |
| X^7 | 1/7 (0.14) | T | 0.059 |

$$\varepsilon_1 = \frac{1}{7} * 3 = 0.42$$

$$\beta_1 = \frac{0.42}{(1-0.42)} = 0.72$$

$$Model\ Weight_1 = \log \frac{1}{\beta_1} = \log \frac{1}{0.72} = 0.14$$

Inference

- ▶ Therefore, when classifying an unseen instance we should apply the following:

$$\text{Output: } h(x) = \operatorname{argmax}_{y \in Y} \sum_{r=1}^k (\log \frac{1}{\beta_r}) \mathbf{1}[h_r(x) = y]$$

- ▶ For **each possible class we add up the weight of the models that predicted that class**. The class with the highest cumulative weight is the winner.

Ensemble Learning

- ▶ Ensemble methods utilize/combine the results from a number of learning algorithms to obtain a better predictive performance than could be obtained from any single one of the learning algorithms learning algorithms
- ▶ There are two families of ensemble methods:
- ▶ **Averaging methods**, build many estimators independently and then to average/vote based on their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced.
- ▶ **Boosting methods**, base estimators are built sequentially and one tries to address the deficiencies of the previous estimators. The motivation is to combine several weak models to produce a powerful ensemble.