Recap

- We have been discussing ensemble classifiers.
- The idea is to combine many classifiers together to achieve high accuracy even though individual classifiers may not be highly accurate.
- we need to generate multiple training data and ensure that there is good variability among the ensemble.
- Finally, we also need to decide how to combine all classifier decisions.
- Bagging and Boosting provide two popular approaches.

Recap

- Bagging exploits randomness to generate good ensembles.
- Random Forests is a popular algorithm of this kind.
- The base classifiers here are decision trees
- We briefly discussed algorithms for learning decision trees.
- In Random Forests we generate multiple (random) bootstrap samples to train individual (decision tree) classifiers.
- We employ randomness in classifier learning too.
- We use majority voting to combine all classifier decisions.

Recap

- AdaBoost is a popular boosting algorithm.
- In AdaBoost we assign (non-negative) weights to points in the data set.
- In each iteration, we learn a classifier to minimize **weighted error** on training data.
- After learning the current classifier, we increase the (relative) weights of data points that are misclassified by the current classifier.
- We learn another classifier with the modified weights.
- The variability in the classifier ensemble comes from the modification of weights.
- The final classifier is a weighted majority voting by all the classifiers.

Notation

- $\{(X_1, y_1), \cdots, (X_n, y_n)\}$ is the data. We take $y_i \in \{-1, +1\}$.
- $w_i(k)$ denotes the weight for the i^{th} data point at k^{th} iteration.
- h_k denotes the classifier learned at k^{th} iteration; we take $h_k(X) \in \{-1, +1\}$.

AdaBoost Algorithm

1. Initialize: $w_i(1) = \frac{1}{n},\ \forall i$.
2. For m=1 to M do
   a. Learn classifier h_m to minimize $\sum_{i=1}^{n} w_i(m) I_{[y_i \neq h_m(X_i)]}$ where I_A is an indicator of A.
   b. Let

$$\xi_m = \sum_{i=1}^{n} w_i(m) I_{[y_i \neq h_m(X_i)]}$$

   (We assume $\xi_m < 0.5$).
   c. Set $\alpha_m = \frac{1}{2} \ln \left(\frac{1-\xi_m}{\xi_m} \right)$. (We have $\alpha_m > 0$).

Algorithm contd.

d. Update the weights by

$$\begin{aligned} w_i'(m+1) &= w_i(m)\ \exp\left(-\alpha_m\ y_i\ h_m(X_i)\right) \\ w_i(m+1) &= \frac{w_i'(m+1)}{\sum_i w_i'(m+1)} \end{aligned}$$

3. Output the final classifier:

$$h(X) = \text{sgn}\left(\sum_{m=1}^{M} \alpha_m\ h_m(X)\right)$$

- In each iteration we update weights as

$$\begin{aligned} w_i'(m+1) &= w_i(m)\ \exp\left(-\alpha_m\ y_i\ h_m(X_i)\right) \\ w_i(m+1) &= \frac{w_i'(m+1)}{\sum_i w_i'(m+1)} \end{aligned}$$

- If $h_m(X_i) \neq y_i$, then $w_i(m+1) > w_i(m)$.
- If the current classifier misclassifies a pattern, its weight for the next iteration is increased.

- We have shown that

$$\sum_{i\,:\,h_m(X_i)\neq y_i} w_i(m+1) = \frac{1}{2}$$

- The weight update is such that at the next iteration, half the total weight is for patterns misclassified by the current classifier and the remaining half is for patterns correctly classified by the current classifier.
- This gives correct level of variability in successive classifiers that are learnt.

- The final classification decision on a new X by this classifier ensemble is

$$h(X) = \text{sgn}\left(\sum_{m=1}^{M} \alpha_m \, h_m(X)\right)$$

- If α_m are same for all m, this is a simple majority decision. (Recall $h_m(X) \in \{-1, +1\}$).
- Here each component classifier has a different weight for its vote.
- The weight is based on the accuracy of that classifier.

- This algorithm can generate a classifier ensemble with any number of component classifiers.
- The component classifiers can be of any type.
- For each classifier we only require that $\xi_m < 0.5$.
- The strength of this boosting techniques is that we can combine many such weak classifiers to finally come up with a classifier with high accuracy.

Base Learning Algorithm

- The base learning algorithm can be anything.
- However, base classifier should minimize weighted error.
- Most learning algorithms minimize errors in training set.
- Here training samples have weights.
- How does one design/learn classifiers to minimize weighted errors?

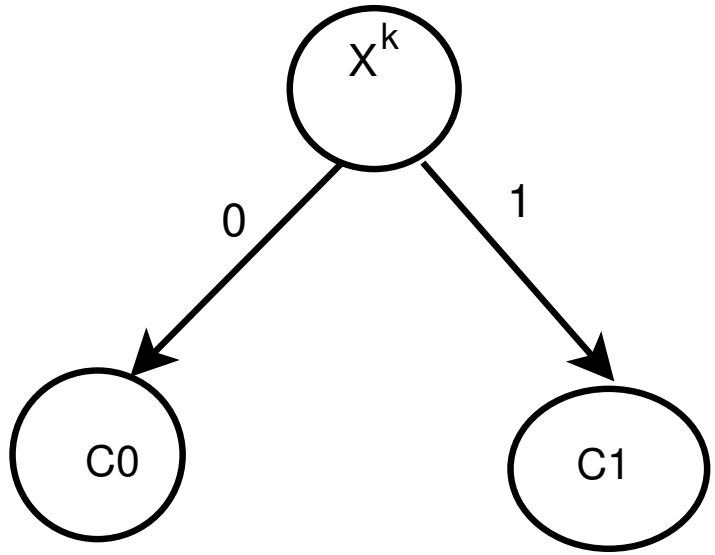
- One way is to generate different random training sets for each iteration.
- At m^{th} iteration, we generate a random set of examples by sampling with replacement from the original training set using the distribution $\{w_i(m)\}$.
- Now an algorithm minimizing errors on this random training set (approximately) minimizes the weighted error as needed.
- This is one of the standard ways of using AdaBoost.
- Called boosting by resampling.

- We can also design an algorithm to generate a classifier to minimize weighted errors on training set.
- We can use a very simple structure for the classifier to make minimizing weighted error easy.
- Note that the individual classifiers need not achieve high accuracy.
- We consider an example of such a base classifier.

A Simple Base learning Algorithm

- A simple method to minimize weighted training error is to learn the best **decision stump**.
- A decision stump is a two-level decision tree.
- At the root, we test a chosen feature and based on its value take one of the branches.
- All second level nodes are leaves with class labels.

- Let us fix a k and consider learning the best decision stump with respect to k^{th} feature, X^k, which is binary:



- Every decision stump is the classifier:

$$\begin{aligned} h(X) &= c_0 \text{ if } X^k = 0 \\ &= c_1 \text{ if } X^k = 1 \end{aligned}$$

  Note that $c_0, c_1 \in \{+1, -1\}$.
- Our task is to find optimal values for c_0 and c_1

- Suppose we calculate for $j \in \{0, 1\}$ and $b \in \{+1, -1\}$,

$$W_b^j = \sum_{i: X_i^k = j \& y_i = b} w_i$$

- W_{+1}^0 would be the sum of weights of all examples in class $+1$ whose k^{th} component is 0.
- Suppose a classifier assigns '$+1$' when $X^k = 0$. Then, sum total weight of examples correctly classified by t is W_{+1}^0 and weight of examples wrongly classified by it is W_{-1}^0
- We can compute W_b^j in linear time over number of examples.

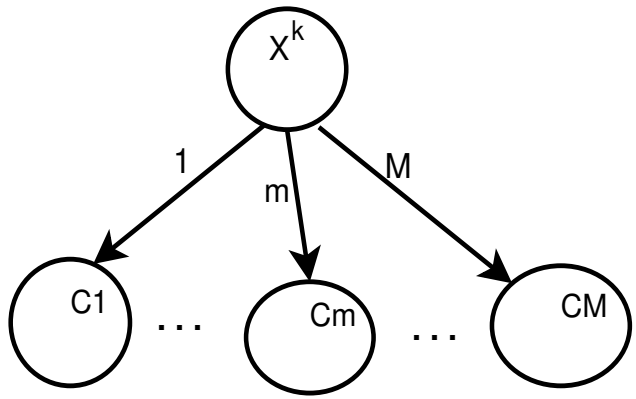
- For our h we need to find optimal values of c_0 and c_1 to minimize weighted error.
- It is easy to see that the best values are

$$\begin{aligned} c_0 &= +1 \text{ if } W^0_{+1} \geq W^0_{-1} \\ &= -1 \text{ if } W^0_{-1} > W^0_{+1} \end{aligned}$$

- That is, if X^k takes value 0, then we are better off putting X in class $+1$ if this feature is 0 "more often" in class $+1$.
- Similarly we can decide best value for c_1.
- This gives us the best decision stump we can get by splitting on k^{th} feature.
- Note that $W^0_{-c_0} + W^1_{-c_1}$ is the weighted error of our h.

- Now suppose X^k takes values in $\{1, \cdots, M\}$.
- The structure of decision stump classifier is

$$h(X) = c_m \text{ if } X^k = m, \ m = 1, \cdots, M.$$



- We now compute W_b^j for $j \in \{1, \cdots, M\}$ and $b \in \{+1, -1\}$.
- Specifying h is fixing c_m (as either $+1$ or -1) for each m.
- The best h is given by the same equation as earlier:

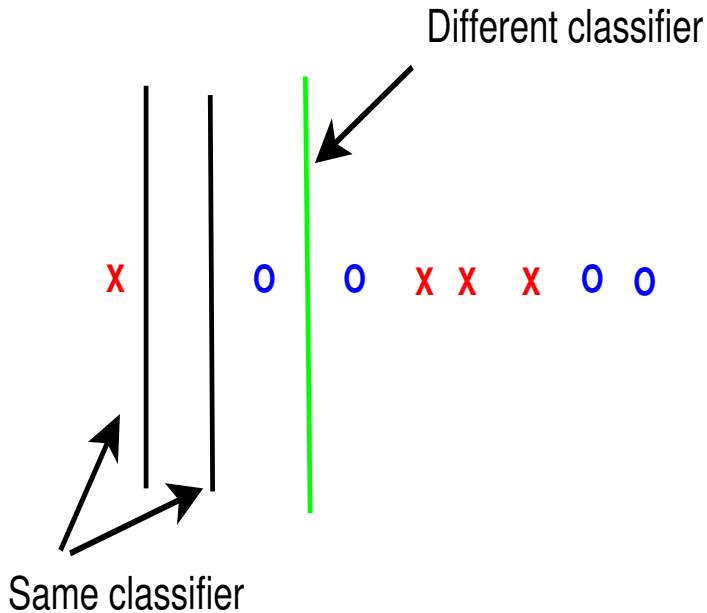
$$\begin{aligned} c_m &= +1 \text{ if } W_{+1}^m \geq W_{-1}^m \\ &= -1 \text{ if } W_{-1}^m > W_{+1}^m \end{aligned}$$

- Now suppose X^k is real valued.
- Then the structure of decision stump classifier is

$$\begin{aligned} h(X) &= c_0 \text{ if } X^k \geq \tau \\ &= c_1 \text{ if } X^k < \tau \end{aligned}$$

- For a given τ this is like a binary feature.
- So, if we fix τ we know how to find the best decision stump.
- How many different τ do we need to consider?

- We need to consider only finitely many τ.
- Given n examples, there are only $n + 1$ distinguishable values for τ.



- When X^k is real valued, the structure of decision stump classifier is

$$\begin{aligned} h(X) &= c_0 \text{ if } X^k \geq \tau \\ &= c_1 \text{ if } X^k < \tau \end{aligned}$$

- For a given τ this is like a binary feature.
- We need to consider only $n+1$ values for τ
- So, we can find the best decision stump for k^{th} feature when it is real-valued also.

- Thus, we can find best decision stump using any one of the features.
- We know the weighted error rate of each of them.
- Hence we can find the best decision stump classifier to minimize weighted error considering all the features.
- So, if we take decision stump as the structure for base classifier, we have a method to find a classifier that minimizes weighted error and guarantees that the weighted error is less than 0.5

Weak Learnability Assumption

- Whatever base learner we use, we want to learn a classifier that minimizes weighted training error.
- We want the weighted error rate to be less than 50%.
- The assumption in Adaboost is that given any distribution $\{w_i\}$, there exists a classifier with weighted error rate less than 0.5.
- This is called the weak learnability assumption.
- In many situations, it holds.

- As long as each component classifier has an error rate better than 0.5, the ensemble does very well.
- One can show that the training error of the ensemble classifier decreases exponentially with increasing number of classifiers in the ensemble.
- While this does not necessarily ensure low test error, in practice the test error also decreases.

Training Error of AdaBoost

- The final classifier is $H(X) = \text{sign}(F(X))$ where

$$F(X) = \sum_{m=1}^{M} \alpha_m h_m(X)$$

- Suppose $\forall m, \xi_m = 0.5 - \gamma_m$, for some $\gamma_m > 0$.
- Then we can show that

$$\sum_i w_i(1)\ I_{[H(X_i) \neq y_i]} \leq \prod_{m=1}^{M} \sqrt{1 - 4\gamma_m^2} \leq \exp\left(-2 \sum_{m=1}^{M} \gamma_m^2\right)$$

Training error decreases exponentially

- Let $Z_m = \sum_i w_i'(m+1)$. Then, for any m

$$\begin{aligned} w_i(m+1) &= \frac{w_i(m)\ \exp(-\alpha_m y_i h_m(X_i))}{Z_m} \\ &= \frac{w_i(m-1)\ \exp(-\alpha_{m-1} y_i h_{m-1}(X_i))}{Z_{m-1}} \frac{\exp(-\alpha_m y_i h_m(X_i))}{Z_m} \end{aligned}$$

and so on. Thus we get

$$\begin{aligned} w_i(M+1) &= \frac{w_i(1) \prod_{m=1}^{M} \exp(-\alpha_m y_i h_m(X_i))}{\prod_{m=1}^{M} Z_m} \\ &= \frac{w_i(1) \exp\left(-y_i \sum_{m=1}^{M} \alpha_m h_m(X_i)\right)}{\prod_{m=1}^{M} Z_m} \\ &= \frac{w_i(1) \exp\left(-y_i\ F(X_i)\right)}{\prod_{m=1}^{M} Z_m} \end{aligned}$$

- Hence

$$w_i(1)\exp(-y_iF(X_i)) = w_i(M+1)\prod_{m=1}^{M} Z_m$$

- Recall the final classification is $H(X_i) = \text{sign}(F(X_i))$.
- If $H(X_i) \neq y_i$ then $\exp(-y_iF(X_i)) \geq 1$.
- This gives us

$$I_{[H(X_i)\neq y_i]} \leq \exp(-y_iF(X_i))$$

Hence we get

$$\begin{aligned}
\sum_{i=1}^{n} w_i(1)\ I_{[H(X_i)\neq y_i]} &\leq \sum_{i=1}^{n} w_i(1)\ \exp(-y_i F(X_i)) \\
&= \sum_{i=1}^{n} w_i(M+1) \prod_{m=1}^{M} Z_m \\
&= \prod_{m=1}^{M} Z_m
\end{aligned}$$

- We have already shown that $Z_m = 2\sqrt{\xi_m(1-\xi_m)}$.
- Since $\xi_m = 0.5 - \gamma_m$, we have

$$\xi_m(1-\xi_m) = \left(\frac{1}{2} - \gamma_m\right)\left(\frac{1}{2} + \gamma_m\right) = \frac{1-4\gamma_m^2}{4}$$

- This gives us

$$\sum_{i=1}^{n} w_i(1) I_{[H(X_i)\neq y_i]} \leq \prod_{m=1}^{M} Z_m = \prod_{m=1}^{M} \sqrt{1-4\gamma_m^2}$$

- Thus the training error of AdaBoost (with M component classifiers) is bounded as

$$\sum_{i=1}^{n} w_i(1) I_{[H(X_i) \neq y_i]} \leq \prod_{m=1}^{M} \sqrt{1 - 4\gamma_m^2} \leq \exp\left(-2 \sum_{m=1}^{M} \gamma_m^2\right)$$

  The last inequality uses $1 - x \leq e^{-x}$

- If $\gamma_m = \gamma$, then the training error is bounded by $\exp(-2\gamma^2 M)$.
- It decreases exponentially with the number of component classifiers.

- Low training error does not necessarily imply low generalization error.
- However, AdaBoost is seen to perform very well in practice (e.g., Viola-Jones face finder).
- There are different ways to understand the performance of AdaBoost.
- Next we look at a simple risk minimization viewpoint.

Risk minimization view of Adaboost

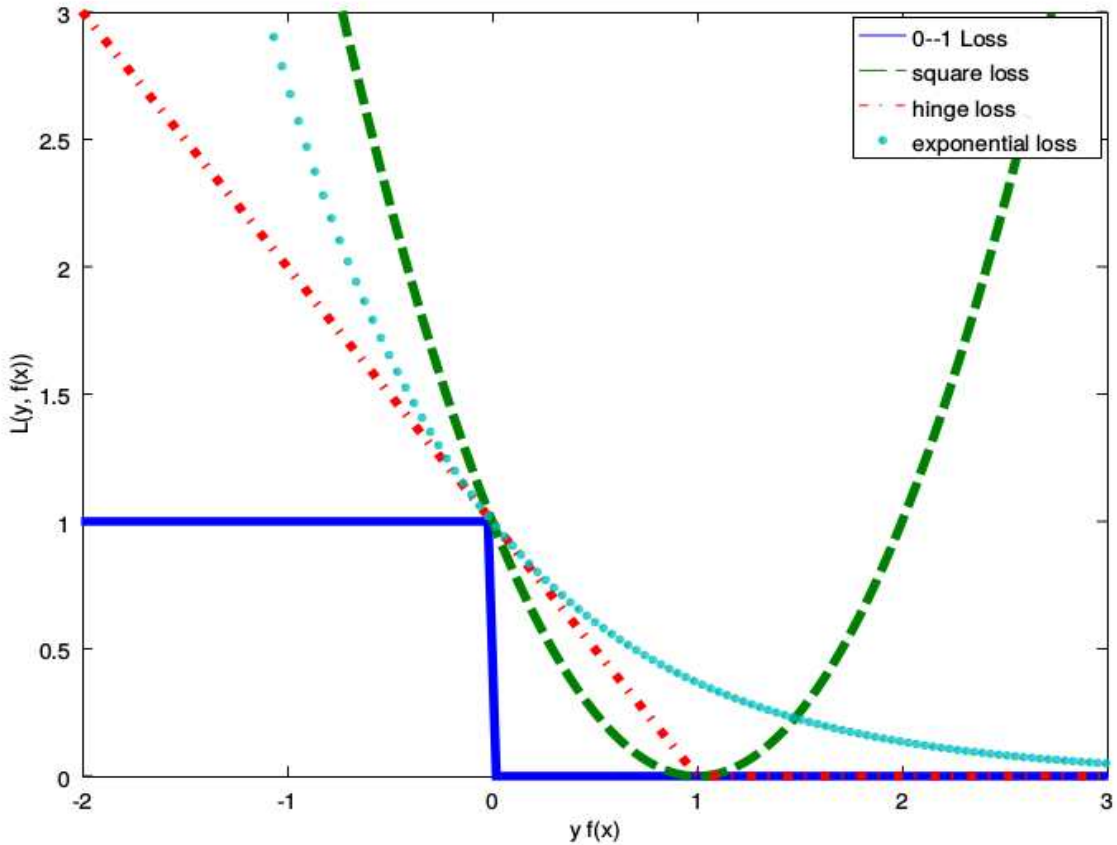
- We consider a loss function called exponential loss which is what AdaBoost minimizes.
- We introduce a class of models called additive models which capture the ensemble classifiers we are considering.
- We show that AdaBoost is essentially finding an additive model by minimizing empirical risk with respect to the exponential loss function.

Exponential Loss

- The exponential loss function is defined by

$$L(h(X), y) = \exp\left(-yh(X)\right)$$

- Consider the case where $h(X), y \in \{+1, -1\}$.
- Then, if $y = h(X)$, the loss is low; otherwise it is high.
- It can be seen as another convex approximation to the 0–1 loss.


0--1 Loss
square loss
hinge loss
exponential loss
L(y, f(x))
y f(x)
3
2.5
2
1.5
1
0.5
0
-2
-1
0
1
2
3

- Next, we can ask why exponential loss function?
- It can be shown that the minimizer of risk under exponential loss over all real-valued functions h is

$$h^*(X) = \frac{1}{2} \log \left(\frac{P[Y = 1|X]}{P[Y = -1|X]} \right)$$

- Then, using sign of $h^*(X)$ gives us the optimal classifier.
- This is one reason why exponential loss is attractive.

Additive Models

- Consider a classifier expressed as

$$f_M(X) = \text{sign}\left(\sum_{m=1}^{M} \beta_m h_m(X)\right)$$

  where each h_m is a classifier and $\beta_m \in \Re$.
- Since we are considering 2-class classifiers, we used the sign function.
- In general, f_M is a weighted sum of the outputs of h_m.
- Such models are called additive models.

- When learning a classifier ensemble, we are learning an additive model.
- If we are doing empirical risk minimization for learning an additive model (with fixed M), then we need to solve the optimization problem

$$\min_{\beta_m,\, h_m} \sum_{i=1}^{n} L\left(\sum_{m=1}^{M} \beta_m h_m(X_i),\ y_i\right)$$

where $L(\cdot, \cdot)$ is the loss function.

- We need to simultaneously optimize over $\beta_1, \cdots, \beta_M$ and $h_1, \cdots, h_M$.
- Optimizing with respect to h_m means searching over a chosen (parameterized) family of base classifiers.
- This is a difficult optimization problem.
- A good heuristic in such situations is to employ a greedy approach.
- That is, at each stage, we optimize on one more classifier to be added to the model.
- It is called stage-wise learning of additive models.

- AdaBoost is essentially doing empirical risk minimization with respect to exponential loss for stage-wise learning of additive models.
- Recall that we want to learn a model

$$f_M(X) = \left(\sum_{m=1}^{M} \beta_m h_m(X) \right)$$

- For any m, let f_m denote the weighted sum of the first m classifiers in the above summation:

$$f_m(X) = \sum_{j=1}^{m} \beta_j h_j(X) = f_{m-1}(X) \; + \; \beta_m h_m(X)$$

- At stage m, we have already learnt $m-1$ component classifiers and are trying to add one more.
- So, we need to find 'best' β_m and h_m to minimize empirical risk of the m-component additive model.
- In the greedy approach, we leave all the previously learned ones unchanged.
- Hence, the solution for the next component of the model can be written as

$$(\beta_m,\ h_m) = \arg\min_{\beta,\, h} \sum_{i=1}^{n} \exp\left(-y_i[f_{m-1}(X_i) + \beta h(X_i)]\right)$$

- Define

$$w_i(m) = \exp(-y_i f_{m-1}(X_i)), \quad i = 1, \cdots, n.$$

- We think of $w_i(m)$ as weight for X_i at the m^{th} stage.
- Since this depends only on f_{m-1}, it is a constant for the optimization problem at the m^{th} stage.
- We would finally show that this is same as the weight used in AdaBoost.
- So, the optimization problem now is

$$(\beta_m,\ h_m) = \arg\min_{\beta,\, h} \sum_{i=1}^{n} w_i(m)\ \exp\left(-y_i \beta h(X_i)\right)$$

- We first show that for any fixed $\beta > 0$, if we optimize over h, the solution is

$$h_m = \arg\min_h \sum_{i=1}^{n} w_i(m)\, I_{[y_i \neq h(X_i)]}$$

- Thus, h_m is essentially the one that has least 'weighted error'.
- This is the error that we have considered in the AdaBoost algorithm.

- Recall that here h is a 2-class classifier and we have $y_i, h(X_i) \in \{-1, +1\}$.
- Hence, $y_i h(X_i) \in \{-1, +1\}, \forall i$.
- Recall

$$(\beta_m, h_m) = \arg\min_{\beta, h} \sum_{i=1}^{n} w_i(m) \exp\left(-y_i \beta h(X_i)\right)$$

- In this expression, the exponential term would be either e^{β} or $e^{-\beta}$ because $y_i h(X_i) \in \{-1, +1\}$.

Denote by A the expression being optimized. Then

$$
\begin{aligned}
A &= \sum_{i=1}^{n} w_i(m)\, \exp\left(-y_i \beta h(X_i)\right) \\
&= e^{\beta} \sum_i w_i(m) I_{[y_i \neq h(X_i)]} \; + \; e^{-\beta} \sum_i w_i(m) I_{[y_i = h(X_i)]} \\
&= e^{\beta} \sum_i w_i(m) I_{[y_i \neq h(X_i)]} \; + \; e^{-\beta} \sum_i w_i(m) (1 - I_{[y_i \neq h(X_i)]}) \\
&= (e^{\beta} - e^{-\beta}) \sum_i w_i(m) I_{[y_i \neq h(X_i)]} \; + \; e^{-\beta} \sum_i w_i(m)
\end{aligned}
$$

- The second term in the above expression is a constant.
- Hence for a fixed $\beta > 0$, optimizing the above over h gives us

$$h_m = \arg\min_h \sum_{i=1}^{n} w_i(m)\, I_{[y_i \neq h(X_i)]}$$

- Thus, the next classifier that we should add is the one which minimizes this weighted misclassification error.

- This is the classifier that we are finding in each iteration in AdaBoost.
- Please note (again) that we are yet to show that our $w_i(m)$ would be the same weights as in AdaBoost algorithm.
- We would do that a little later.
- First, given this solution h_m, we find the β_m so that we can complete finding the solution that minimizes empirical risk.

- Now, with h_m fixed, we need to find β_m by optimizing (over β),

$$(e^{\beta} - e^{-\beta}) \sum_i w_i(m) I_{[y_i \neq h_m(X_i)]} \; + \; e^{-\beta} \sum_i w_i(m)$$

- The optimization problem will be same if we divide the above by $\sum_i w_i(m)$.
- Define

$$\xi_m = \frac{\sum_i w_i(m) \; I_{[y_i \neq h_m(X_i)]}}{\sum_i w_i(m)}$$

- Hence, we can now write β_m as

$$\beta_m = \arg\min_{\beta} \left(e^{\beta} - e^{-\beta}\right) \xi_m \; + \; e^{-\beta}$$

- Differentiating w.r.t β and equating to zero we get

$$(e^{\beta} + e^{-\beta})\xi_m \; - \; e^{-\beta} = 0$$

which gives us

$$\beta_m = \frac{1}{2} \ln \left(\frac{1 - \xi_m}{\xi_m} \right)$$

- Thus, our final solution to the m^{th} stage component is

$$\begin{aligned} h_m &= \arg\min_h \sum_{i=1}^{n} w_i(m)\, I_{[y_i \neq h(X_i)]} \\ \beta_m &= \frac{1}{2} \ln\left(\frac{1-\xi_m}{\xi_m}\right) \\ \xi_m &= \frac{\sum_i w_i(m)\, I_{[y_i \neq h_m(X_i)]}}{\sum_i w_i(m)} \end{aligned}$$

- All these exactly correspond to our AdaBoost algorithm.
- We are trying to find h_m to minimize the weighted error.
- If the weights are normalized at each stage, the ξ_m is the same.
- The α_m in the algorithm corresponds to β_m here.

- To complete our analysis to show that AdaBoost is essentially doing a stage-wise empirical risk minimization on exponential loss, we have to show that the $w_i(m)$ used in the algorithm are same as what we defined here.
- Specifically, we need to show that the method of obtaining $w_i(m+1)$ from $w_i(m)$ is justified.
- That is what we do next.

- Recall that under our notation

$$f_m(X) = \sum_{j=1}^{m} \beta_j h_j(X)$$

- Hence, we have

$$f_m(X) = f_{m-1}(X) + \beta_m h_m(X)$$

- Recall that we have defined $w_i(m)$ by

$$w_i(m) = \exp(-y_i f_{m-1}(X_i))$$

- Hence we get

$$\begin{aligned} w_i(m+1) &= \exp(-y_i f_m(X_i)) \\ &= \exp(-y_i f_{m-1}(X_i) - y_i \beta_m h_m(X_i)) \\ &= w_i(m)\ \exp(-y_i \beta_m h_m(X_i)) \end{aligned}$$

- Except for the normalization, this is the weight update in AdaBoost.

- Thus, AdaBoost can be understood as learning an additive model in a greedy fashion by minimizing empirical risk under exponential loss.
- This provides some justification why AdaBoost is so successful in practice.
- There are many variants of this boosting algorithm and now there are also many more theoretical results on the behaviour of classifier ensembles.

Multiclass case

- The AdaBoost algorithm that we presented is for 2-class case.
- The next question is: how do we handle multi-class case
- One general method is 'one Vs rest'.
- It is also possible to thik of a straight-forward extension of AdaBoost to multiclass case.

AdaBoost M1

1. Initialize: $w_i(1) = \frac{1}{n},\ \forall i$.
2. For m=1 to M do
   a. Learn classifier h_m to minimize $\sum_{i=1}^{n} w_i(m) I_{[y_i \neq h_m(X_i)]}$ where I_A is an indicator of A.
   b. Let
   $$\xi_m = \sum_{i=1}^{n} w_i(m) I_{[y_i \neq h_m(X_i)]}$$
   (We assume $\xi_m < 0.5$).
   c. Set $\alpha_m = \frac{1}{2} \ln \left(\frac{1-\xi_m}{\xi_m} \right)$. (We have $\alpha_m > 0$).

Algorithm contd.

d. Update the weights by

$$\begin{aligned} w_i'(m+1) &= w_i(m)\ \exp\left(-\alpha_m\ z_{mi}\right) \\ w_i(m+1) &= \frac{w_i'(m+1)}{\sum_i w_i'(m+1)} \end{aligned}$$

where $z_{mi} = +1$ if $h_m(X_i) = y_i$ and is -1 otherwise.

3. Output the final classifier:

$$H(X) = \arg\max_y \sum_{m=1}^{M} \alpha_m I_{[h_m(X)=y]}$$

- The extension is very straight-forward!
- All the same properties can be established for this too.
- However, assuming that you always get a classifier with weighted error less than 0.5 is now a restrictive assumption.
- AdaBoost is a very competitive method for classifier ensembles.
- Boosting is also an active research area.