LECTURE 22

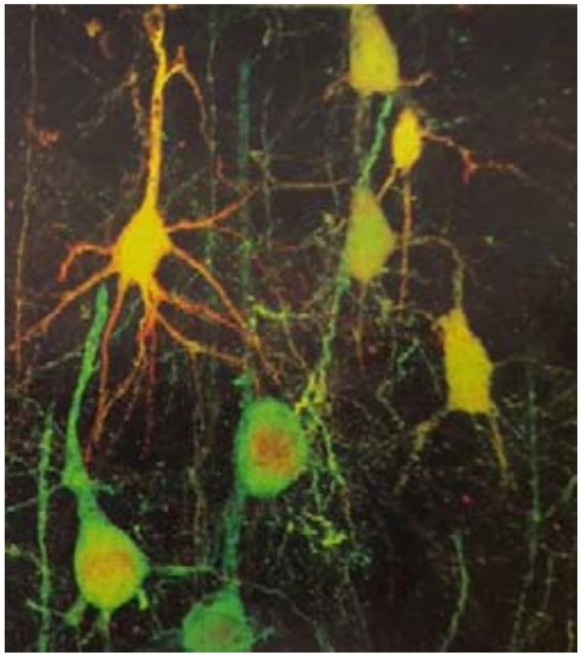
Boosting – Part I

A technique for combining a number of “weak” classifiers to make a “strong” classifier

Agenda

- Basic algorithm: introducing the boosting procedure
- Ensemble Methods: Boosting and Bagging
- Different versions:
    - Adaboost, RealBoost, and LogitBoost optimizing different design of loss functions.
- A statistical framework: discussing its relation to Maximum likelihood estimation (MLE learning).

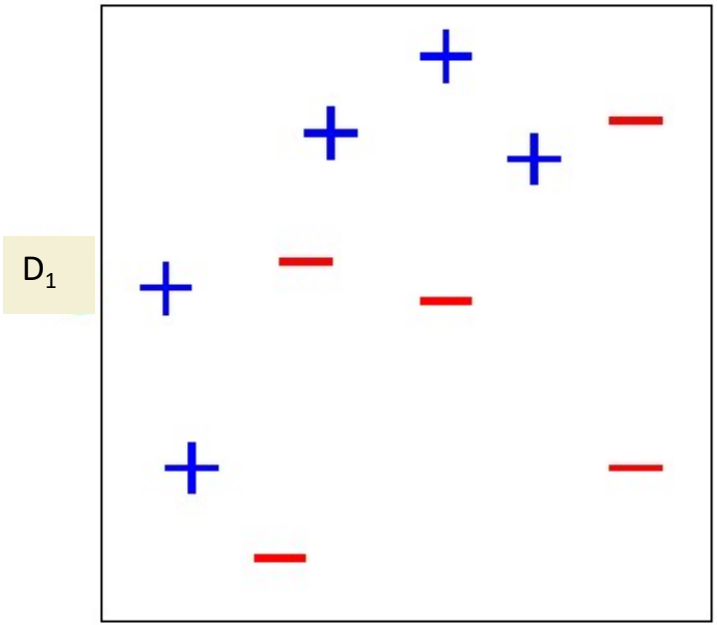
Background



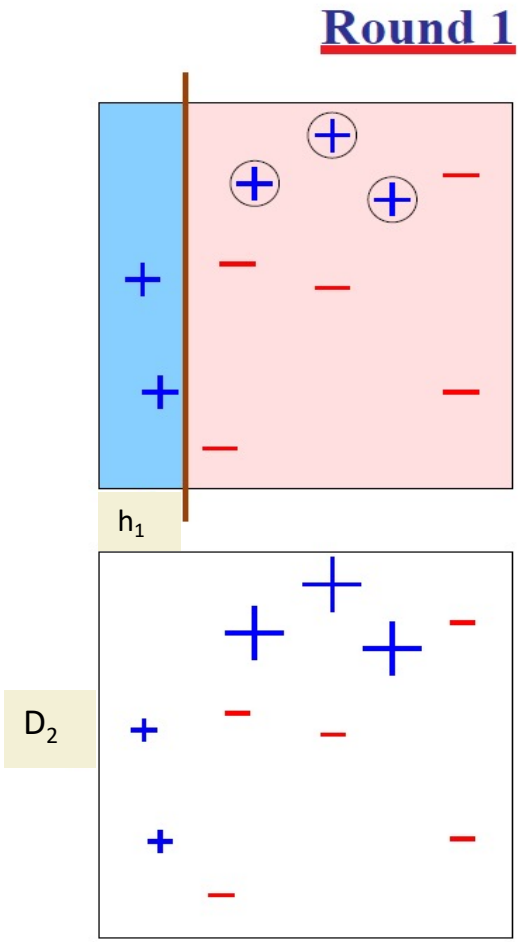
- Boosting is related to the early ideas in neural network, more specifically the perceptron proposed by Frank Rosenblatt 1962 as a model of neurons.
- Each neuron is modeled by a linear product of the input feature vector and a weight vector, which is then followed by a threshold.

A toy example

A Toy Example

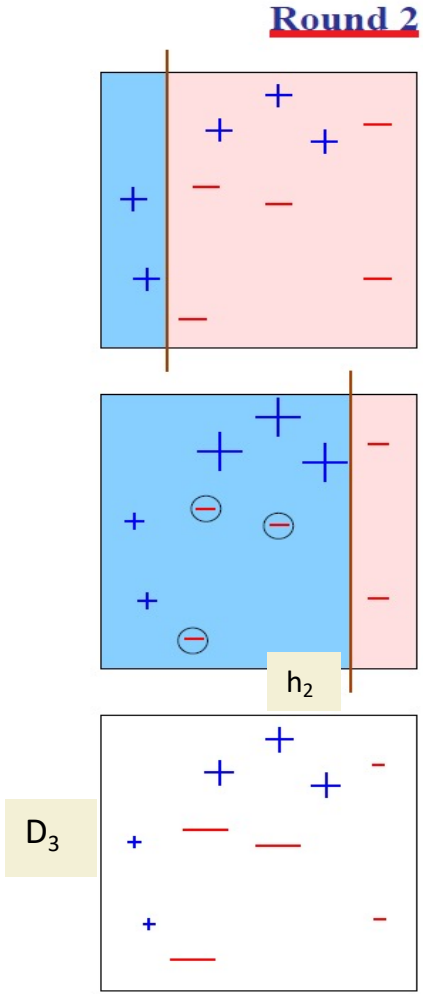


A Toy Example



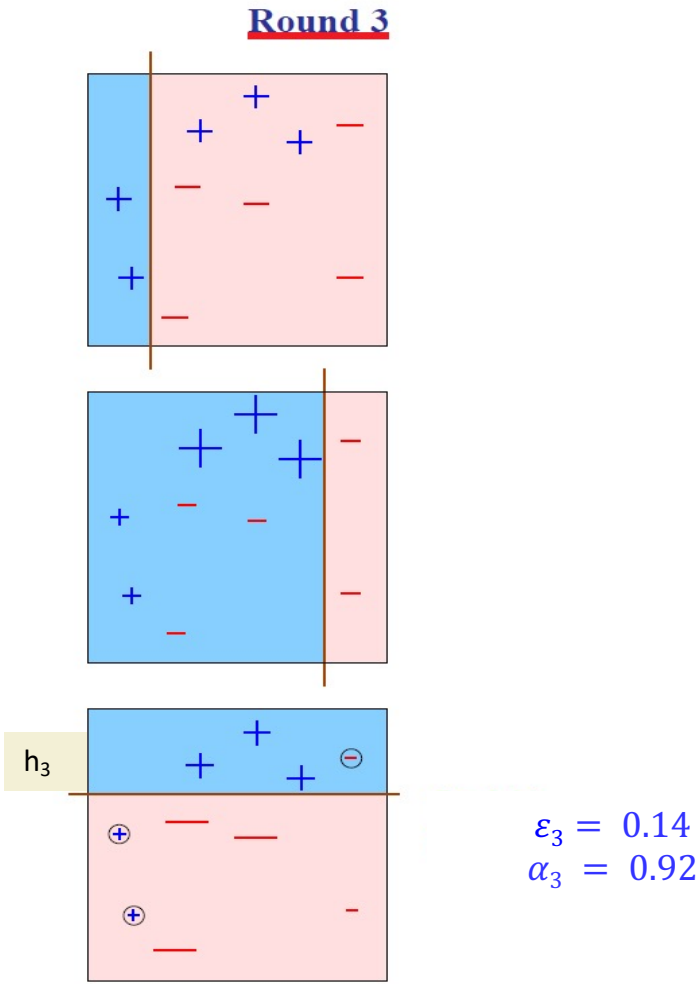
$\varepsilon_1 = 0.3$
$\alpha_1 = 0.42$

A Toy Example



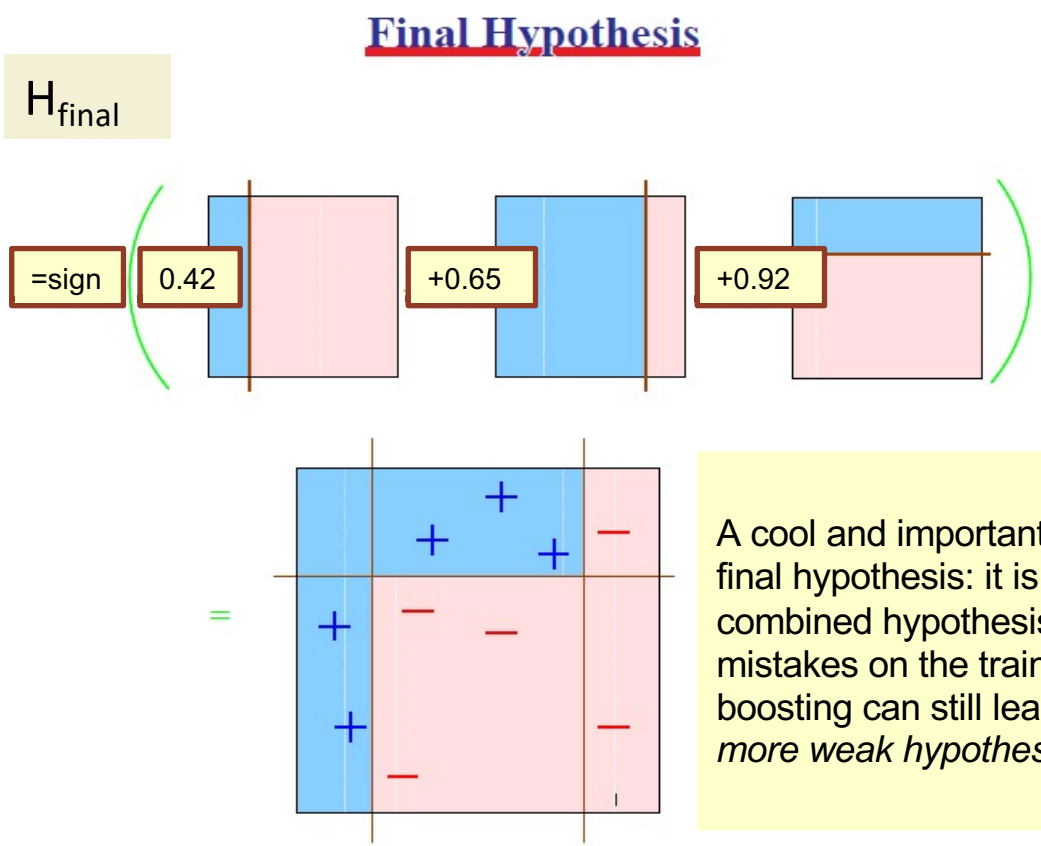
$\varepsilon_2 = 0.21$
$\alpha_2 = 0.65$

A Toy Example



$\varepsilon_3 = 0.14$
$\alpha_3 = 0.92$

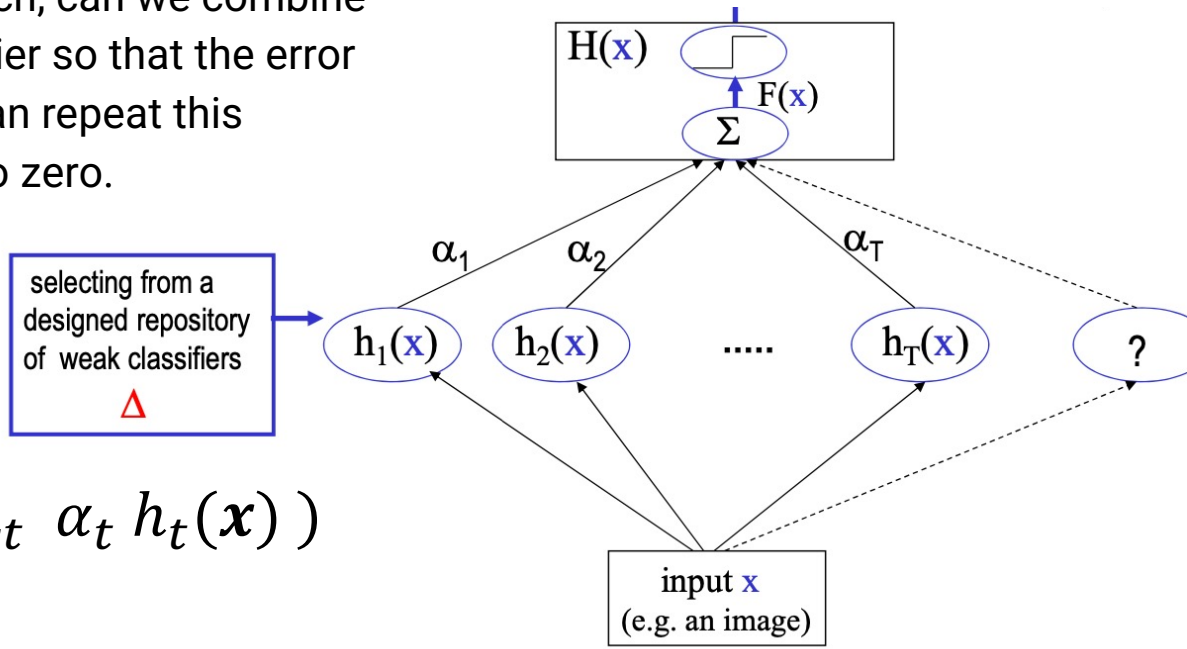
A Toy Example



A cool and important note about the final hypothesis: it is possible that the combined hypothesis makes no mistakes on the training data, but boosting can still learn, *by adding more weak hypotheses.*

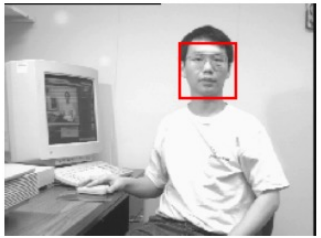
Intuition

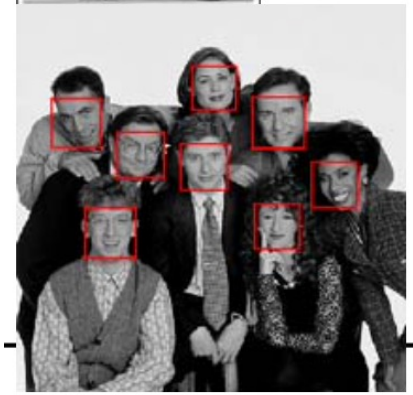
- Suppose we have two weak classifiers h1 and h2 which have 49% error each, can we combine them to make a new classifier so that the error becomes lower? If so, we can repeat this process to make the error to zero.

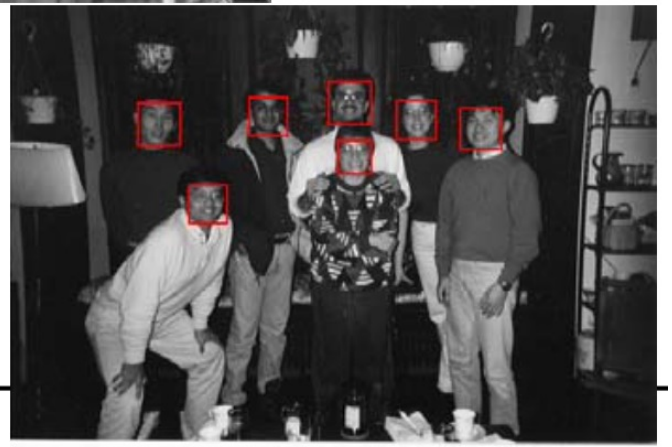


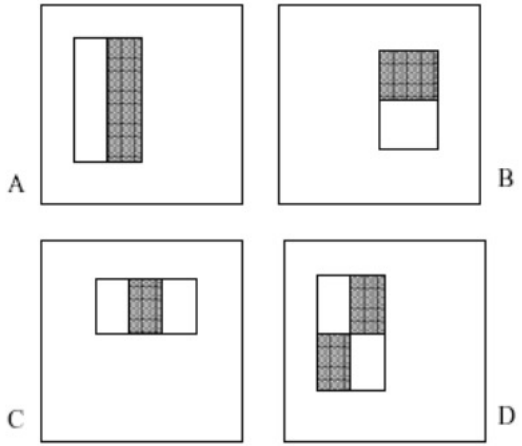
$$H_{final}(\boldsymbol{x}) = sign\left(\sum_t \alpha_t h_t(\boldsymbol{x})\right)$$

Example of weak classifier









Weak classifiers used for face detection in (Viola and Jones 01) are windowed features A,B,C,D on a 24x24 pixel image patches. The features are designed for easy computation using the integral image.

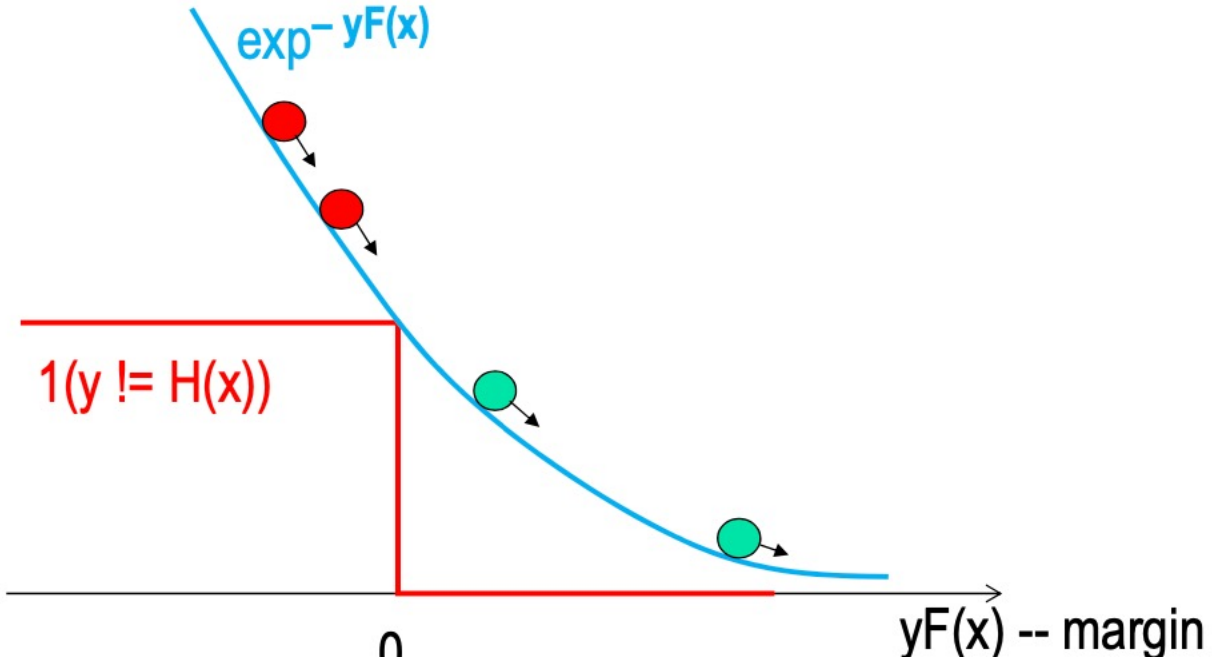
Basic Boost

- A strong classifier is a combination of a number of **weaker classifiers**:
  - $H(\boldsymbol{x}) = sign\left(\sum_t \alpha_t h_t(\boldsymbol{x})\right)$
- We denote by
  - $h = (h1, \ldots, h_T)$
  - $\alpha = (\alpha_1, \ldots, \alpha_T)$
  - F(x) = $\sum_t \alpha_t h_t(\boldsymbol{x}) = < \alpha, h >$
- So our objective is to choose h and parameters α to minimize the empirical error of the strong classifier
  - $Err(H) = \frac{1}{n}\sum_{i=1}^{n} 1(H(x_i) \neq y_i)$
  - $(\hat{\alpha}, h) = argmin\ Err(H)$

Boosting

- Initialization:
  - Weigh all training samples equally
- Iteration Step:
  - Train model on (weighted) train set
    - Choose your favorite hypothesis space & learning algorithm
  - Compute error of model on train set
  - Update the distribution:
    - Increase/decrease weights on training cases model gets wrong/correct.
- Typically requires $100's$ to $1000's$ of iterations
- Return final model:
  - Carefully weighted prediction of each model

Adaboost



Intuitively, a margin measures how far away
a data point is away from the decision boundary.

Adaboost Loss function

- It is difficult to derive such a loss function, so the following function is used instead.
- $Err(H) = \frac{1}{n}\sum_{i=1}^{n} 1(H(x_i) \neq y_i) \leq \frac{1}{n}\sum_{i=1}^{n} e^{-y_i F(x_i)}$

  (minimize the exponential loss) , an upper bound on 0/1 loss
- $Loss(F) = \frac{1}{n}\sum_{i=1}^{n} e^{-y_i F(x_i)}$
- $(h_{t+1}, \alpha_{t+1}) = \text{argmin}_{h,\alpha}\left(Loss(F_t + \alpha h)\right) = \text{argmin}_{h,\alpha}\frac{1}{n}\sum_{i=1}^{n} e^{-y_i[F(x_i)+\alpha h]}$

  $= \text{argmin}_{h,\alpha}\frac{1}{n}\sum_{i=1}^{n} \omega_i e^{-y_i \alpha h(x_i)}$

Basic AdaBoost algorithm

1. Initialize the data with uniform weight

$$D_0(x_i) = \frac{1}{n} \text{ so, } \sum_{i=1}^{n} D_0(x_i) = 1$$

2. At step t, compute the weighted error for each weak classifier

$$\varepsilon_t(h) = \sum_{i=1}^{n} D_t(x_i)\, 1(h(x_i) \neq y_i)$$

3. Choose a new weak classifier which has the least weighted error

$$h_t = argmin\ \varepsilon_t(h)$$

4. Assign weight for the new classifier

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

5. Update the weights of the data points

$$D_t(x_i) = \frac{1}{Z_t} D_{t-1}(x_i) e^{-y_i \alpha_t h_t(x_i)} \quad Z_t \text{ is a normalization function}$$

Set t+1→t, repeat 2-5 until stopping conditions

The algorithm stops under three possible conditions:

- The training error of the strong classifier H(x) is below a threshold, or become zero.
  - In fact, people can continue to boost after the training error becomes zero, such that the positive and negative examples are separated by a bigger margin
- All the remaining weak classifiers have error close to 0.5 and thus redundant.
- A maximum number of weak classifier T is reach.

Summary of Ensemble Methods

- Boosting
- Bagging (Random Forests)

Boosting: Different Perspectives

- Boosting is a maximum-margin method (Schapire et al. 1998, Rosset et al. 2004)
  - Trades lower margin on easy cases for higher margin on harder cases
- Boosting is an additive logistic regression model (Friedman, Hastie and Tibshirani 2000)
  - Tries to fit the logit of the true conditional probabilities
- Boosting is a linear classifier, over an incrementally acquired “feature space”.

Bagging

- Bagging predictors is a method for generating multiple versions of a predictor and using these to get an aggregated predictor.
  - The aggregation averages over the versions when predicting a numerical outcome and does a plurality vote when predicting a class.
- The multiple versions are formed by making bootstrap replicates of the learning set and using these as new learning sets.
  - That is, use samples of the data, with repetition

- Tests on real and simulated data sets using classification and regression trees and subset selection in linear regression show that bagging can give substantial gains in accuracy.
- The vital element is the instability of the prediction method. If perturbing the learning set can cause significant changes in the predictor constructed, then bagging can improve accuracy.

Boosting Vs Bagging

Boosting	**Bagging**
Reweights data points (modifies their distribution)	Resamples data points
Weight is dependent on classifier's accuracy	Weight of each classifier is the same
Both bias and variance reduced – learning rule becomes more complex with each iteration	Only variance reduction

Facts about the weights

- The weight for the new classifier is always positive
  - The smaller classification error, and bigger weight, and stronger “voice” in the strong classifier.

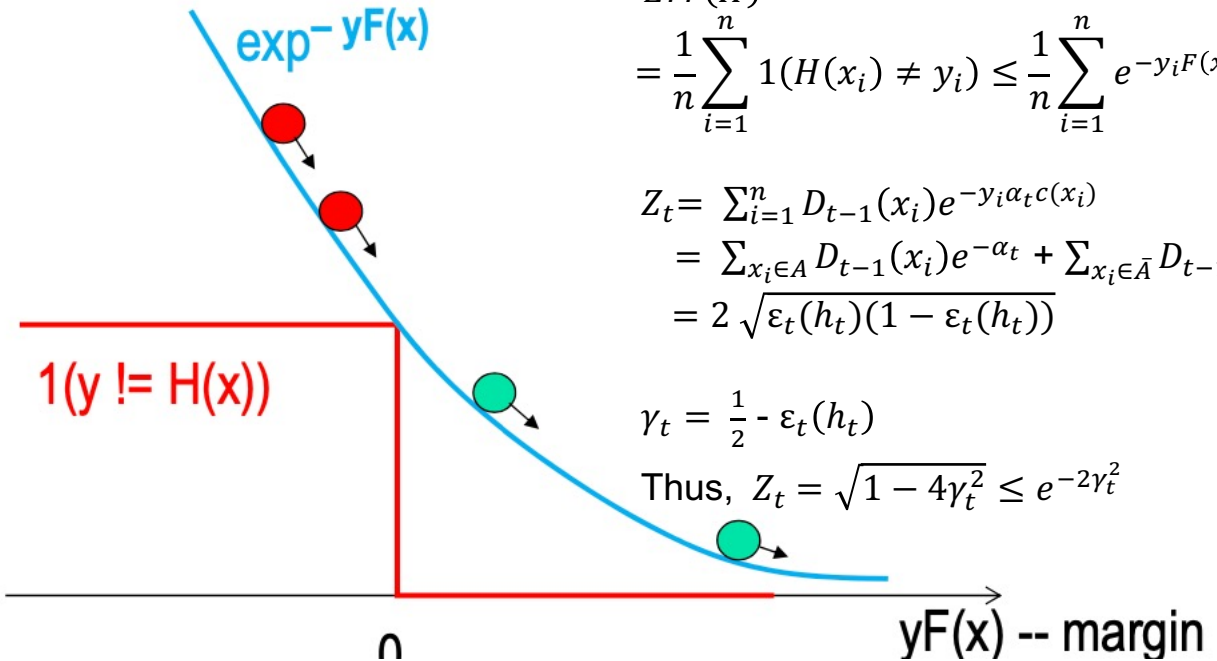
    $\varepsilon_t(h_t) = \sum_{i=1}^{n} D_t(x_i)\ 1(h(x_i) \neq y_i) < ½$

    $\xrightarrow{\text{yields}} \alpha_t = \frac{1}{2}\log(\frac{1-\varepsilon(h_t)}{\varepsilon(h_t)}) > 0$
- The weights of the data points are multiplied by
  - $e^{-y_i \alpha_t h_t(x_i)} = \begin{cases} e^{-\alpha_t} < 1 & if\ y_i = h_t(x_i) \\ e^{\alpha_t} > 1 & if\ y_i \neq h_t(x_i) \end{cases}$
  - The weights of the incorrectly classified data points increase, and the weights of the correctly classified data points decreases. So the incorrectly points receive more “attention” in the next run.

Facts about the normalizing functions

- At each step, the weights of the data points are normalized by
  - $Z_t = \sum_{i=1}^{n} D_{t-1}(x_i) e^{-y_i \alpha_t c(x_i)}$
  - $= \sum_{x_i \in A} D_{t-1}(x_i) e^{-\alpha_t} + \sum_{x_i \in \bar{A}} D_{t-1}(x_i) e^{\alpha_t}$
- *A* is the correctly classified data points by h_t. $Z_t = Z_t(\alpha_t)$ is a function of α_t. The data weights can be computed recursively,
  - $D_t(x_i) = \frac{1}{Z_t} D_{t-1}(x_i) e^{-y_i \alpha_t h_t(x_i)} = \frac{1}{Z_t Z_{t-1} \ldots Z_1} \frac{1}{n} e^{-y_i F(x_i)}$
  - Therefore $Z = Z_t Z_{t-1} \ldots Z_1 = \frac{1}{n} \sum_{i=1}^{n} e^{-y_i F(x_i)}$

AdaBoost Convergence Analysis



$$Err(H) = \frac{1}{n}\sum_{i=1}^{n} 1(H(x_i) \neq y_i) \leq \frac{1}{n}\sum_{i=1}^{n} e^{-y_i F(x_i)} = Z_t Z_{t-1} \ldots Z_1$$

$$Z_t = \sum_{i=1}^{n} D_{t-1}(x_i) e^{-y_i \alpha_t c(x_i)}$$
$$= \sum_{x_i \in A} D_{t-1}(x_i) e^{-\alpha_t} + \sum_{x_i \in \bar{A}} D_{t-1}(x_i) e^{\alpha_t}$$
$$= 2\sqrt{\varepsilon_t(h_t)(1-\varepsilon_t(h_t))}$$

$$\gamma_t = \frac{1}{2} - \varepsilon_t(h_t)$$

Thus, $Z_t = \sqrt{1-4\gamma_t^2} \leq e^{-2\gamma_t^2}$

It is clear that each step the upper bound of the error decrease exponentially. A weak classifier with small error rate will lead to faster descent.

Adaboost with {0,1} output

- Suppose the weak classifier output in {0,1} rather than {-1,1}
  - $\Delta' = \{g_t(x)\colon \Omega^d \rightarrow \{0,1\}\}$
- Then we get weak classifier as
  - $\Delta \ = \{h_t(x)\colon h_t(x) = 2g_t(x) - 1 \quad \rightarrow \{-1,1\}\}$
- The updating functions will be
  - $\varepsilon_t(h) = \sum_{i=1}^{n} D_t(x_i)\ 1(g(x_i) \neq y_i)$
  - $\alpha_t = \frac{1}{2}\log(\frac{1-\varepsilon(h_t)}{\varepsilon(h_t)})$
  - $D_t(x_i) = \frac{1}{Z_t} D_{t-1}(x_i) e^{-\alpha_t 1(g_t(x_i)=y_i)}$
  - $H(\boldsymbol{x}) \ = \ sign\ (\sum_t \ \alpha_t \ g_t(\boldsymbol{x}) - \mathbf{\frac{1}{2}} \sum_t \ \alpha_t)$

Multi-class Adaboost

- Suppose we have a set of weak classifiers for K-classification
  - $\Delta = \{h_t(x)\colon \Omega^d \rightarrow \{1,2,\dots K\}\}$
- The algorithm is the same as the binary Adaboost in calculating the weighted error of the weak classifiers and the data weights, except the last step becomes.
  - $H(x) = argmax_{y\in\{1,\dots,K\}} \sum_{t=1}^{T} \alpha_t 1(h_t(x) = y)$
- Intuitively, the class that receives the highest weighted votes from the weak classifiers will win. The classification error is also upper bounded in a similar way as the binary case.

Variations of Boost: Logitboost and RealBoost

- So far, we didn't derive the algorithm using probabilistic models as we did in Bayesian Decision Theory. The algorithms are derived by design:
- 1, Selecting a formula for the classifier
  - $y = H(x;\theta)$ or the form of F$(x;\theta)$
    θ includes both the parameters and structures of classifiers.
- 2, Optimizing some loss function
  - $\theta^* = argmax\ L(\theta)$

Once we understand the Adaboost, we can see that there are many variations by different designs of F() and L().

Adaboost vs Logistic Regression

- For Logistic regression $y = \{1,0\}$

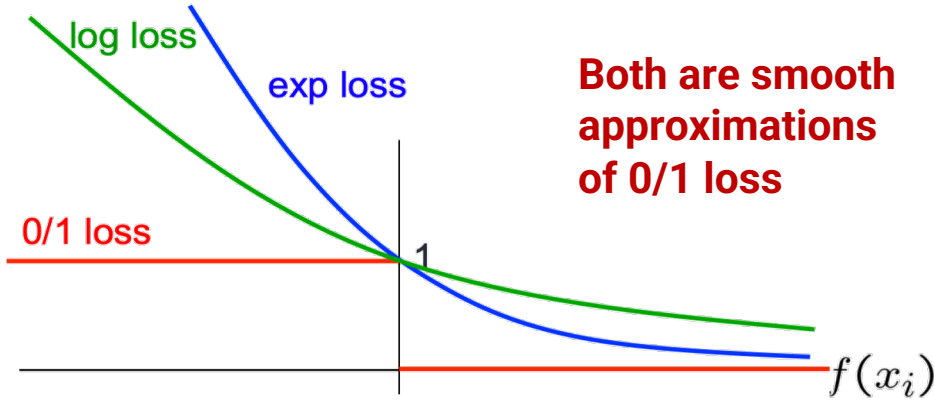
$$loss = \frac{1}{n}\sum_{i=1}^{n} y_i \ln\left(1 + \exp\left(-f(x_i)\right)\right) + (1 - y_i)\ln\left(1 + \exp\left(+f(x_i)\right)\right)$$

- If $y = \{1, -1\}$

$$loss = \frac{1}{n}\sum_{i=1}^{n} \ln(1 + \exp(-y_i f(x_i)))$$

Boosting minimizes similar loss function

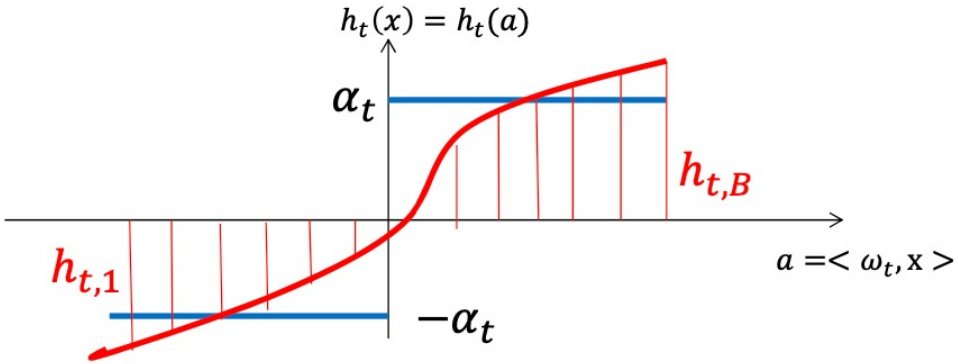
$$Loss(F) = \frac{1}{n}\sum_{i=1}^{n} \exp(-y_i F(x_i))$$



Both are smooth approximations of 0/1 loss

RealBoost

- We may extend $h_t(x)$ to a more general form $h_t(x, \theta_t)$ with $\theta_t = (h_{t,1}, \ldots, h_{t,B})$ is a vector of B parameters,
  With B being the number of bins that we choose to approximate an arbitrary 1D function with



$h_t(x) = h_{t,b}$ if $< \omega, x > = a$ falls in the b-th bin.

XGBoost (Extreme Gradient Boosting)

- Gradient Tree Boosting with Regularization
- Parallelization construction on CPU cores Distributed
- Handles missing data