Ensemble Learning

Ensembles, Bagging, Boosting, AdaBoost Technique

CS 6316 – Machine Learning
Fall 2017

OUTLINE

- Methods for Constructing Ensembles
- Combination Strategies
- Mixtures of Experts (ME)
- Bagging
- Boosting
- AdaBoost
- AdaBoost Steps
- AdaBoost Example

Methods for Constructing Ensembles

- **Subsampling the training examples**
  - Multiple hypotheses are generated by training individual classifiers on different datasets obtained by resampling a common training set (Bagging, Boosting)

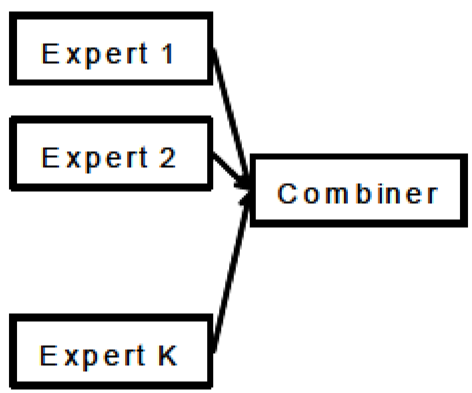
- **Manipulating the input features**
  - Multiple hypotheses are generated by training individual classifiers on different representations, or different subsets of a common feature vector

Methods for Constructing Ensembles

- **Manipulating the output targets**
  - The output targets for C classes are encoded with an L-bit codeword, and an individual classifier is built to predict each one of the bits in the codeword
  - Additional "auxiliary" targets may be used to differentiate classifiers
- **Modifying the learning parameters of the classifier**
  - A number of classifiers are built with different learning parameters, such as number of neighbors in a k Nearest Neighbor rule, initial weights in an MLP, etc.

Structure of Ensemble Classifiers (Parallel)

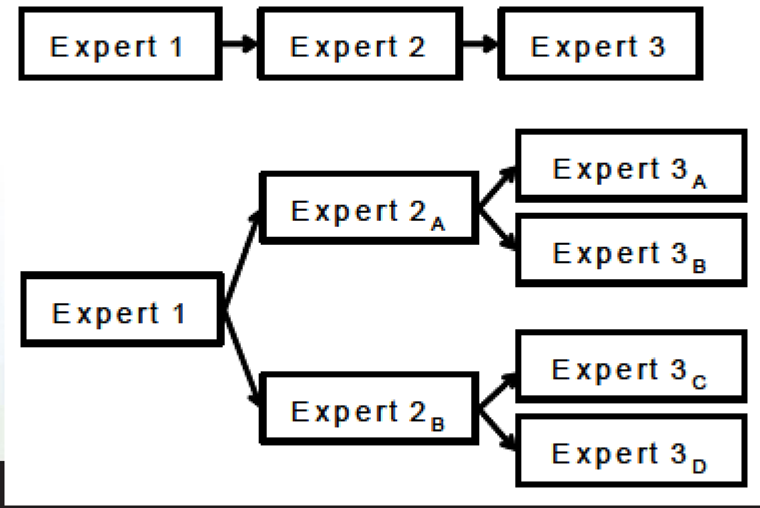
- All the individual classifiers are invoked independently, and their results are fused with a combination rule (e.g., average, weighted voting) or a meta-classifier (e.g., stacked generalization)
- The majority of ensemble approaches in the literature fall under this category



Jain, 2000

Structure of Ensemble Classifiers (Cascading or Hierarchical)

- Classifiers are invoked in a sequential or tree-structured fashion
- For the purpose of efficiency, inaccurate but fast methods are invoked first (maybe using a small subset of the features), and computationally more intensive but accurate methods are left for the latter stages



Jain, 2000

Combination Strategies (Static Combiners)

- The combiner decision rule is independent of the feature vector. Static approaches can be broadly divided into non-trainable and trainable:

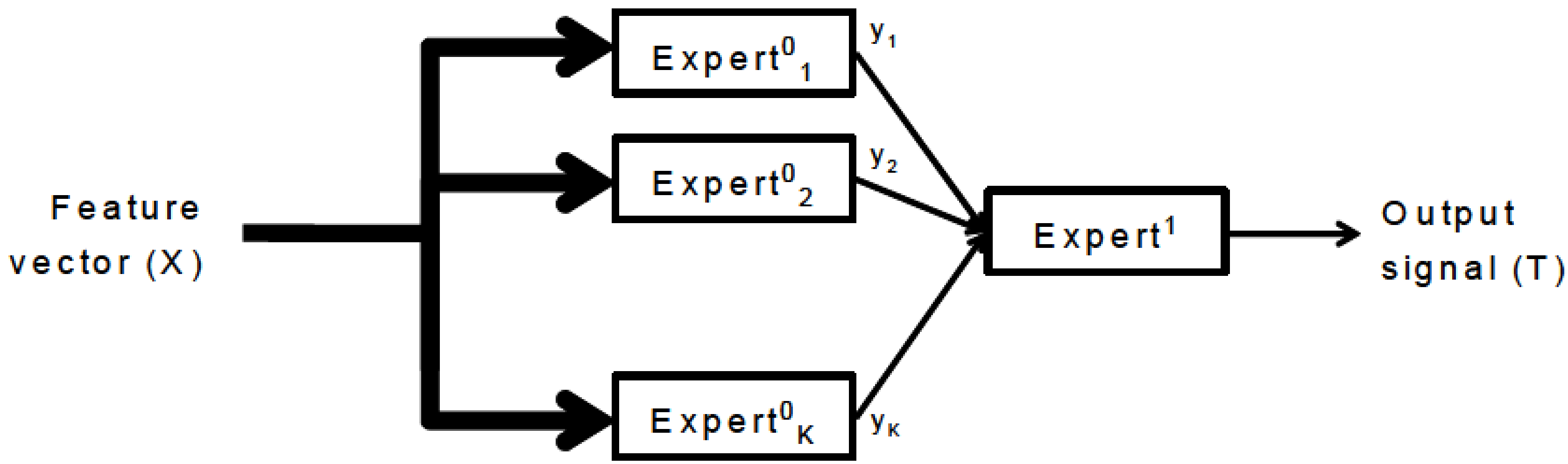
- Non-trainable: The voting is performed independently of the performance of each individual classifier
- Trainable: The combiner undergoes a separate training phase to improve the performance of the ensemble

Non-trainable

- Various combiners may be used, depending on the type of output produced by the classifier, including:
  - **VOTING**: used when each classifier produces a single class label. In this case, each classifier "votes" for a particular class, and the class with the majority vote on the ensemble wins
  - **AVERAGING**: used when each classifier produces a confidence estimate (e.g., a posterior). In this case, the winner is the class with the highest average posterior across the ensemble
  - **BORDA COUNTS**: used when each classifier produces a rank. The Borda count of a class is the number of classes ranked below it [Ho et al., 1994]

Trainable

- **Weighted averaging**: the output of each classifier is weighted by a measure of its own performance, e.g., prediction accuracy on a separate validation set
- **Stacked generalization**: the output of the ensemble serves as a feature vector (input) to a meta-classifier (second-level expert)

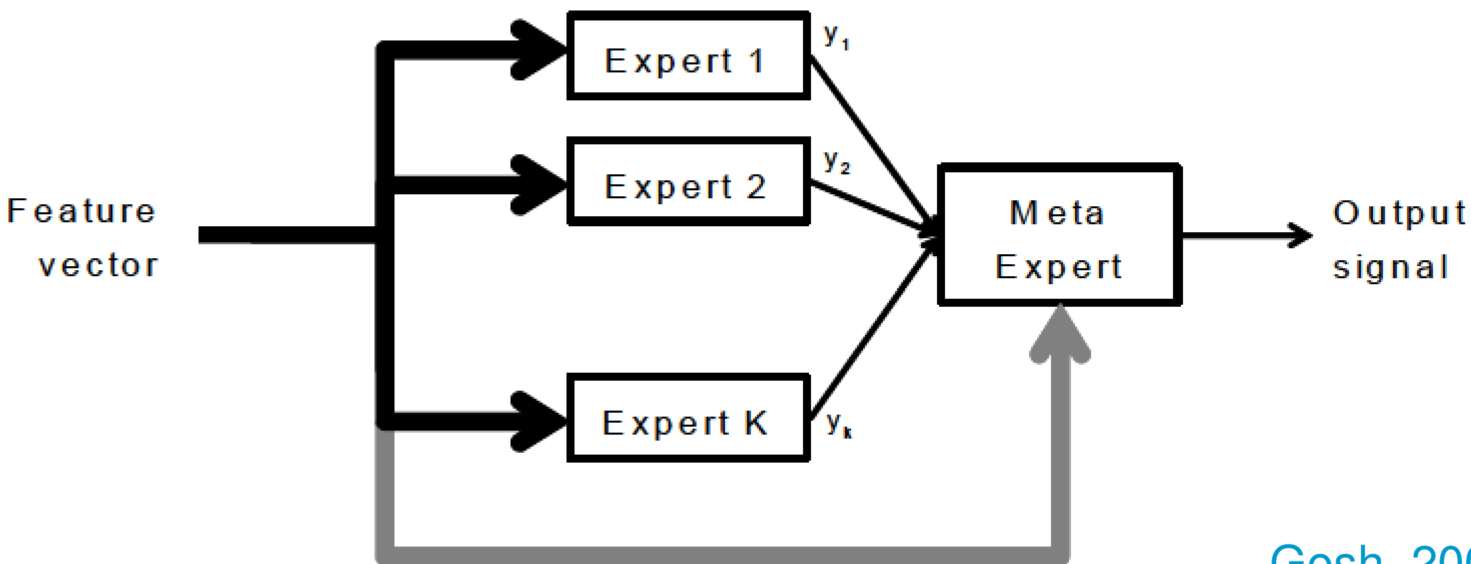


Adaptive Combiners

- The combiner is a function that depends on the input feature vector
  - Thus, the ensemble implements a function that is local to each region in feature space
  - This divide-and-conquer approach leads to modular ensembles where relatively simple classifiers specialize in different parts of I/O space
  - In contrast with static-combiner ensembles, the individual experts here do not need to perform well for all inputs, only in their region of expertise

Adaptive Combiners

- Representative examples of this approach are Mixture of Experts (ME) and Hierarchical ME [Jacobs et al., 1991; Jordan and Jacobs, 1994]

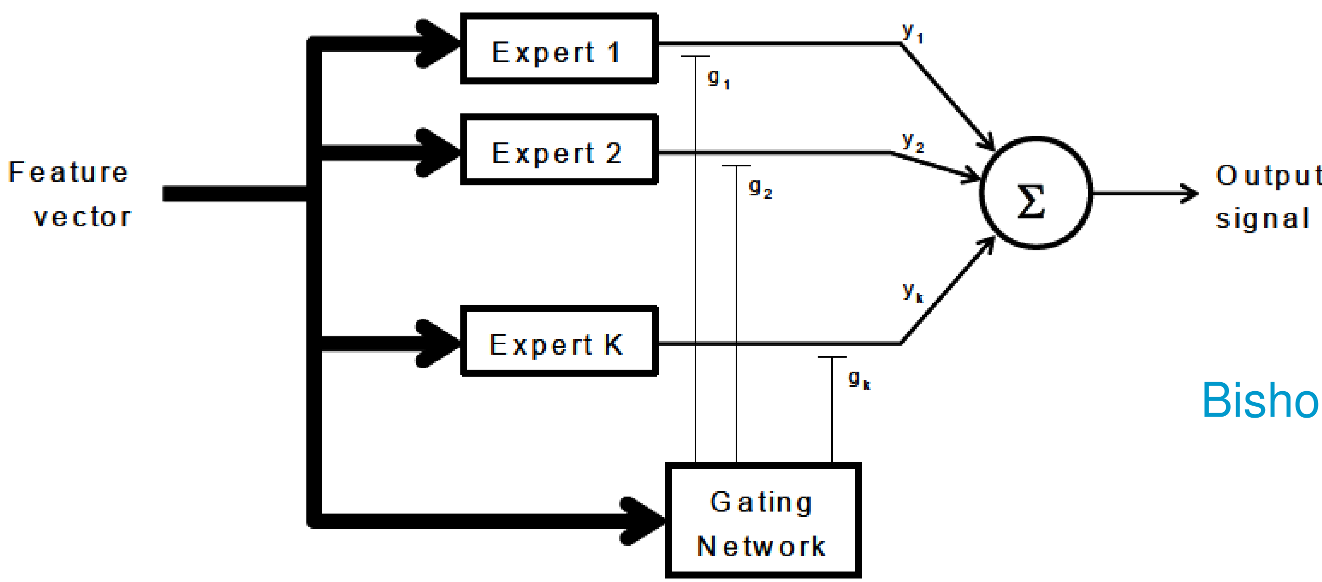


Gosh, 2002

Mixture of Experts (ME)

ME is the classical adaptive ensemble method

- A gating network is used to partition feature space into different regions, with one expert in the ensemble being responsible for generating the correct output within that region [Jacobs et al., 1991]
- The experts in the ensemble and the gating network are trained simultaneously



Bishop, 1995

Bagging

- Subsampling the training set → Bagging [Breiman, 1996]
- Bagging (for bootstrap aggregation) creates an ensemble by training individual classifiers on bootstrap samples of the training set
- As a result of the sampling-with-replacement procedure, each classifier is trained on the average of 63.2% of the training examples
- Bagging traditionally uses component classifiers of the same type (e.g., decision trees), and a simple combiner consisting of a majority vote across the ensemble

Boosting

- [Schapire, 1990; Freund and Schapire, 1996]
- Boosting takes a different resampling approach than bagging, which maintains a constant probability of 1/N for selecting each example
- In boosting, this probability is adapted over time based on performance
  - The component classifiers are built sequentially, and examples that are mislabeled by previous components are chosen more often than those that are correctly classified

Boosting

- Boosting is based on the concept of a "*weak learner*", an algorithm that performs slightly better than chance (e.g., 50% classification rate on binary tasks)
  - Schapire has shown that a weak learner can be converted into a strong learner by changing the distribution of training examples
  - *Small* benefits achieved by using *highly accurate* classifiers
  - There are a number of variants of boosting available in literature
    - A popular one is **AdaBoost** (Adaptive Boosting) – which allows the designer to continue adding components until an arbitrarily small error rate is obtained on the training set

AdaBoost

- AdaBoost (Adaptive Boosting) is a popular boosting technique which helps combine multiple "weak classifiers" into a single "strong classifier"
- A weak classifier is simply a classifier that performs poorly, but performs better than random guessing

- *A simple example might be classifying a person as male or female based on their **height**. You could say anyone over 5' 9" is a **male** and anyone under that is a **female**. You'll misclassify a lot of people that way, but your accuracy will still be greater than 50%*

AdaBoost

- AdaBoost can be applied to any classification algorithm, so it's really a technique that builds on top of other classifiers as opposed to being a classifier itself

AdaBoost: What Do We Get?

- You could just train a bunch of weak classifiers on your own and combine the results, so what does AdaBoost do for you? There's really two things it figures out for you:

1. It helps you choose the training set for each new classifier that you train based on the results of the previous classifier
2. It determines how much weight should be given to each classifier's proposed answer when combining the results

AdaBoost: Training Set Selection

- Each weak classifier should be trained on a random subset of the total training set
- The subsets can overlap–it's not the same as, for example, dividing the training set into ten portions
- AdaBoost assigns a "weight" to each training example, which determines the probability that each example should appear in the training set
- Examples with higher weights are more likely to be included in the training set, and vice versa

AdaBoost: Training Set Selection

- *Examples with higher weights are more likely to be included in the training set, and vice versa*

- After training a classifier, AdaBoost increases the weight on the misclassified examples so that these examples will make up a larger part of the next classifiers training set, and hopefully the next classifier trained will perform better on them

AdaBoost

- Assume 2-class problem, with labels +1 and -1
  - y^i in {-1, 1}
- Discriminant function:

$$g(x) = \sum_{t=1}^{T} \alpha_t h_t(x) = \alpha_1 h_1(x) + \alpha_2 h_2(x) + \ldots \alpha_T h_T(x)$$

- Where $h_t(x)$ is a weak classifier, for example:

$$h_t(x) = \begin{cases} -1 & e.g.\, if\ email\ has\ word\ \text{"money"} \Rightarrow Spam \\ 1 & e.g.\, if\ email\ doesn't\ have\ word\ \text{"money"} \Rightarrow Not\ Spam \end{cases}$$

- The final classifier is the sign of the discriminant function

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

Idea Behind AdaBoost

- Algorithm is iterative
- Maintains distribution of weights over the training examples
- Initially weights are equal
- Main Idea: at successive iterations, the weight of misclassified examples is increased
- This forces the algorithm to concentrate on examples that have not been classified correctly so far

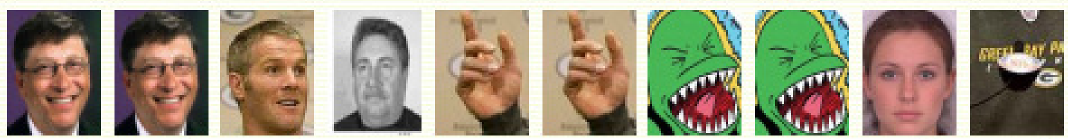
Idea Behind AdaBoost

- Examples of high weight are shown more often at later rounds
- Face/nonface classification problem:

Round 1

	1/7	1/7	1/7	1/7	1/7	1/7	1/7
best weak classifier:	✓	✗	✓	✓	✗	✓	✗
change weights:	1/16	1/4	1/16	1/16	1/4	1/16	1/4

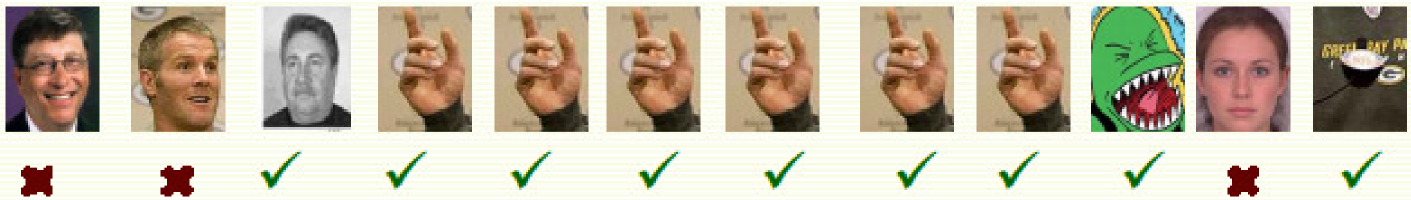
Round 2

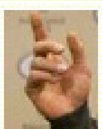


best weak classifier:	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓
change weights:		1/8	1/32	11/32		1/2		1/8	1/32	1/32

Idea Behind AdaBoost

Round 3



- out of all available weak classifiers, we choose the one that works best on the data we have at round 3
- we assume there is always a weak classifier better than random (better than 50% error)
-  image is half of the data given to the classifier
- chosen weak classifier **has to** classify this image correctly

Additional Comments

- Ada boost is very simple to implement, provided you have an implementation of a "weak learner"
- Will work as long as the "basic" classifier $h_t(x)$ is at least slightly better than random
  - will work if the error rate of $h_t(x)$ is less than 0.5
  - 0.5 is the error rate of a ***random guessing*** for a 2-class problem

AdaBoost for 2 Classes

- Initialization step: for each example x, set weight

$$D_1(x) = \frac{1}{\boldsymbol{N}}, \text{ where } \mathbf{N} \text{ is the number of examples}$$

- Iteration step (for **t** = 1 … T):

1. Find best weak classifier $h_t(x)$ using weights D(x) of ex's
2. Compute the error rate ε_t as

$$\boxed{\varepsilon_t = \sum_{i=1}^{N} D(x^i) \cdot I\underbrace{\left[y^i \neq h_t(x^i)\right]}_{= \begin{cases} 1 & if\ y^i \neq h_t(x^i) \\ 0 & otherwise \end{cases}}}$$

*Will only accumulate weights of examples that were **misclassified** (i.e. $y^i \neq h_t(x^i)$)*

AdaBoost

- Iteration step (for t = 1 … T) (CONTINUED)

3. Compute weight α_t of classifier h_t

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) > 0$$

Note: natural log

4. For each example x^i, update the weights (t+1 iteration):

$$D_{t+1}(x^i) = D_t(x^i) \cdot \exp\left(\alpha_t \cdot I\left[y^i \neq h_t(x^i)\right]\right)$$

Recall: $exp(x) = e^x$

Normalize $D_{t+1}(x^i)$ so that $\sum_{i=1}^{N} D(x^i) = 1$

$$D_{t+1}(x^i) = \frac{D_t(x^i) \cdot \exp\left(\alpha_t \cdot I\left[y^i \neq h_t(x^i)\right]\right)}{Z_t}$$

← *Normalization constant*

AdaBoost

- The final combined classifier that will classify example "x"

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

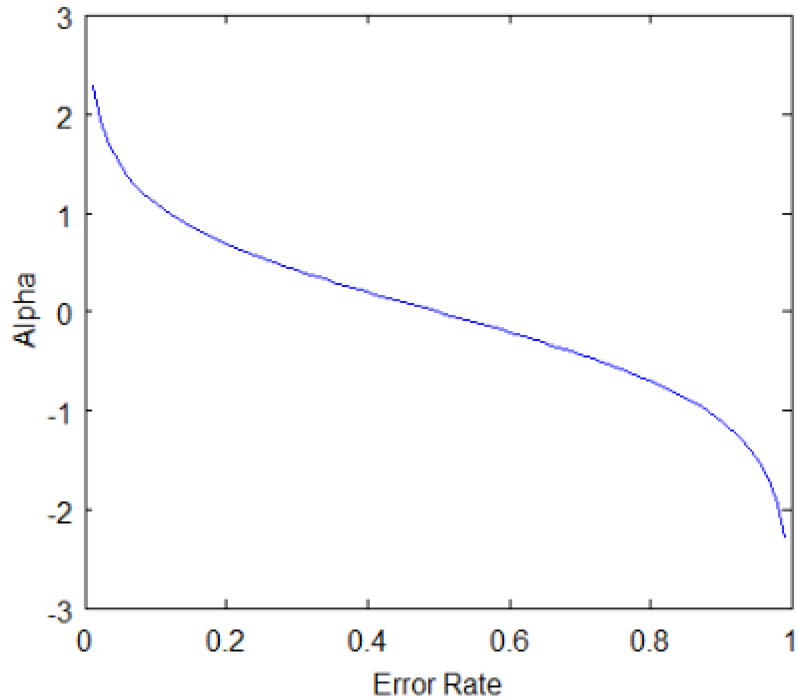
AdaBoost

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

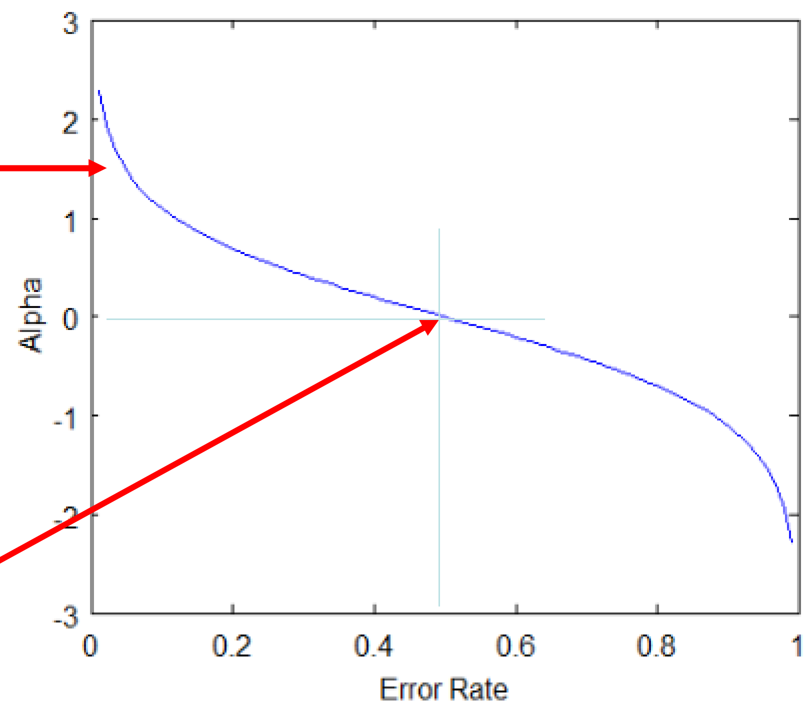
- The final classifier consists of "T" weak classifiers
- $h_t(x)$ is the output of weak classifier **t** (-1/+1 in this ex.)
- α_t is the weight applied to classifier **t** as determined by AdaBoost
- So the final output is just a linear combination of all of the weak classifiers. Final decision: LOOKING AT SIGN OF THE SUM

A little about… α_t

- Plot of what α_t will look like for classifiers with different error rates:

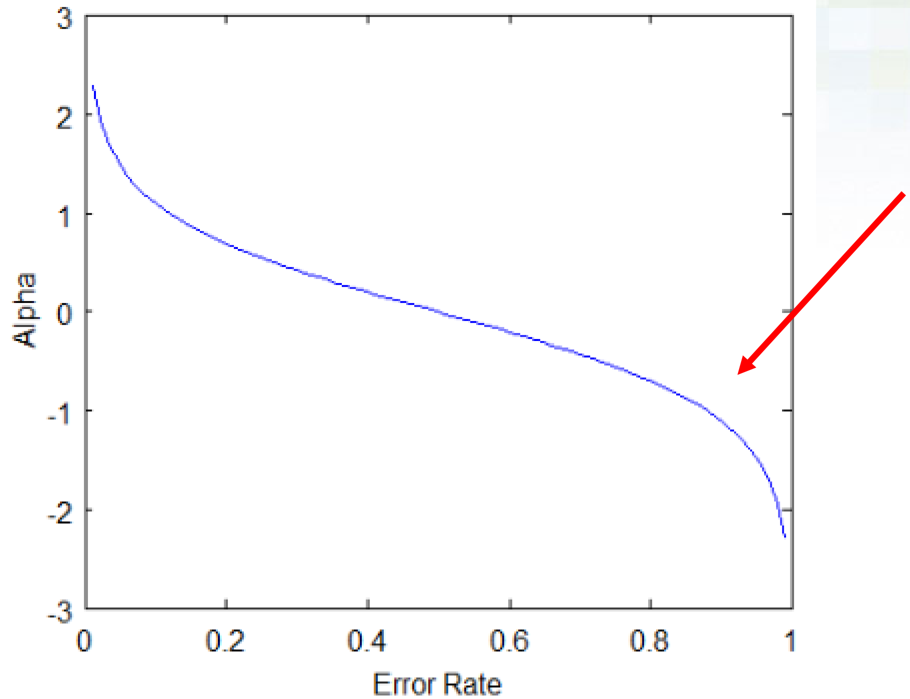


A little about… α_t



1. The classifier weight grows exponentially as the error approaches 0 (better classifiers are given exponentially more weight)
2. The classifier weight is zero if the error rate is 0.5. A classifier with 50% accuracy is no better than random guessing, so we ignore it!
3. The classifier weight grows exponentially negative as the error approaches 1.

A little about… α_t



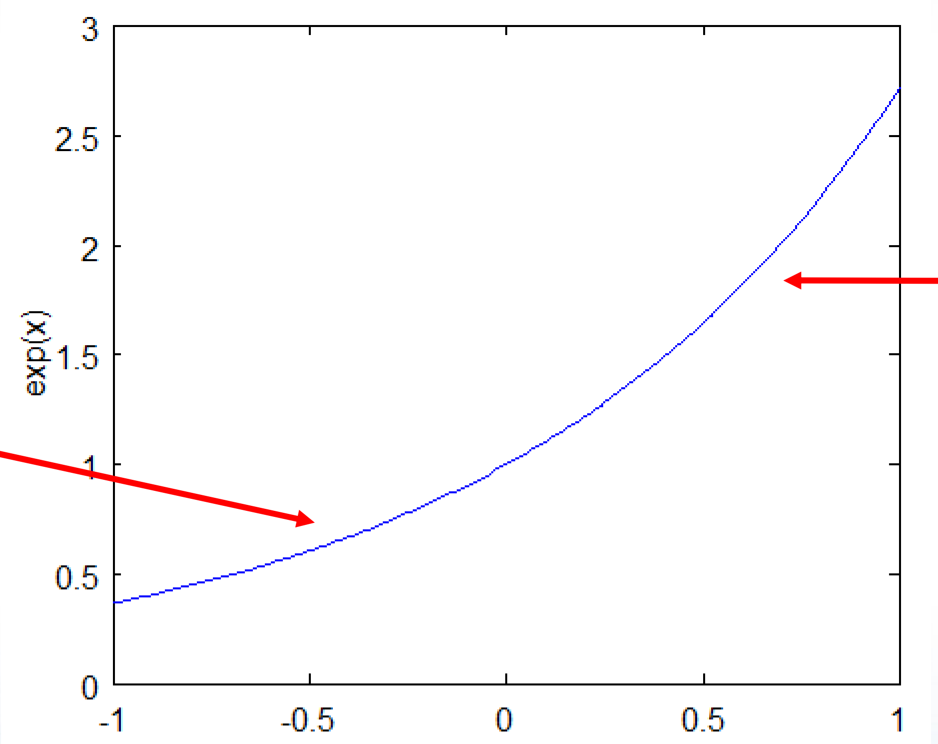
3. The classifier weight grows exponentially negative as the error approaches 1. We give negative weight to classifiers with worse than 50% accuracy
→ *"Whatever the classifier says, do the opposite!"*

Recall: $exp(x) = e^x$ $e = Euler\ constant$

Exp() term when updating weights

- In the equation to **update the weights** (**D**) there exists the **exp** term. Exp(x) behaves as follows:

*Exp(x) will return a fraction for **NEGATIVE** values of x*



*Exp(x) will return a value greater than one for **POSITIVE** values of x*

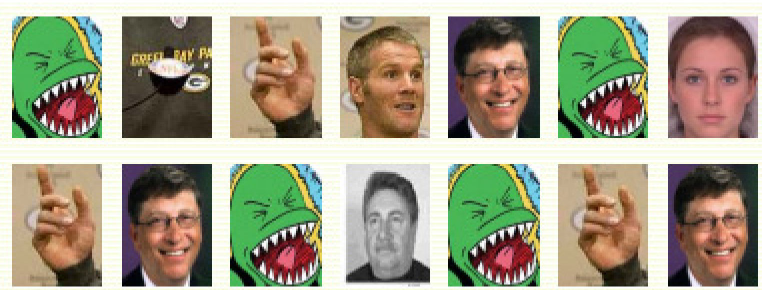
So the weight for training sample i will be either increased or decreased depending on the final sign of the term

AdaBoost: Step 1 [**important**]

[1] Find best weak classifier $h_t(x)$ using weights D(x)

- Some classifiers accept weighted samples, but many don't
- If classifier does not take weighted samples, sample from the training samples according to the distribution D(x)

- Draw k samples, each x with probability equal to D(x):

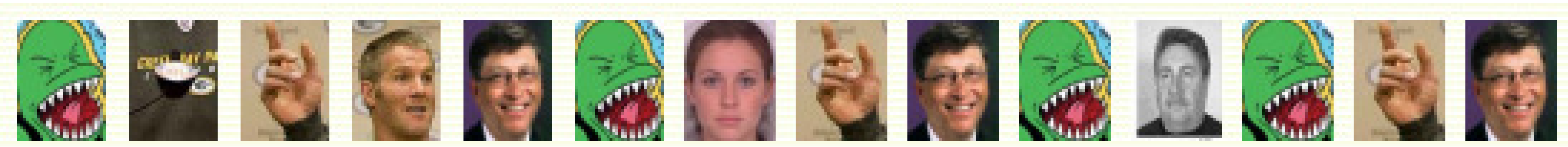


re-sampled examples

AdaBoost: Step 1

[1] Find best weak classifier $h_t(x)$ using weights D(x)

- Give to the classifier the re-sampled examples:



- To find the best weak classifier, go through **all** weak classifiers, and find the one that gives the smallest error on the re-sampled examples

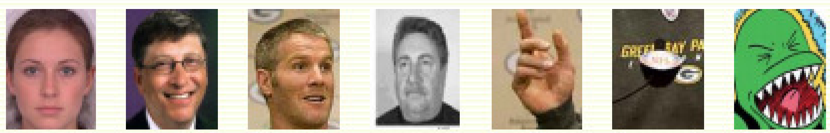
Weak Classifiers	$h_1(x)$	$h_2(x)$	$h_3(x)$	…	$h_m(x)$
Errors:	0.46	0.36	0.16	…	0.43

The best classifier to choose at iteration t

AdaBoost: Step 2

[2] Compute ε_t the error rate

$$\varepsilon_t = \sum_{i=1}^{N} D(x^i)\cdot \overbrace{I[y^i \neq h_t(x^i)]} = \begin{cases} 1 & \text{if } y^i \neq h_t(x^i) \\ 0 & \text{otherwise} \end{cases}$$



1/16	1/4	1/16	1/16	1/4	1/16	1/4
✓	✓	✓	✗	✗	✓	✓

$$\varepsilon_t = \frac{1}{4} + \frac{1}{16} = \frac{5}{16}$$

If classified correctly the weight of that ex will be multiplied by 0 and thus not included

- ε_t is the weight of all misclassified examples added
  - The error rate is computed over original examples, not the re-sampled examples
- If a weak classifier is better than random, then $\varepsilon_t < ½$

AdaBoost: Step 3

[3] Compute weight α_t of classifier h_t

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

- In example from previous slide error was 5/16

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-5/16}{5/16}\right) = \frac{1}{2}\ln\left(\frac{0.6875}{0.3125}\right) = 0.394 \approx 0.4$$

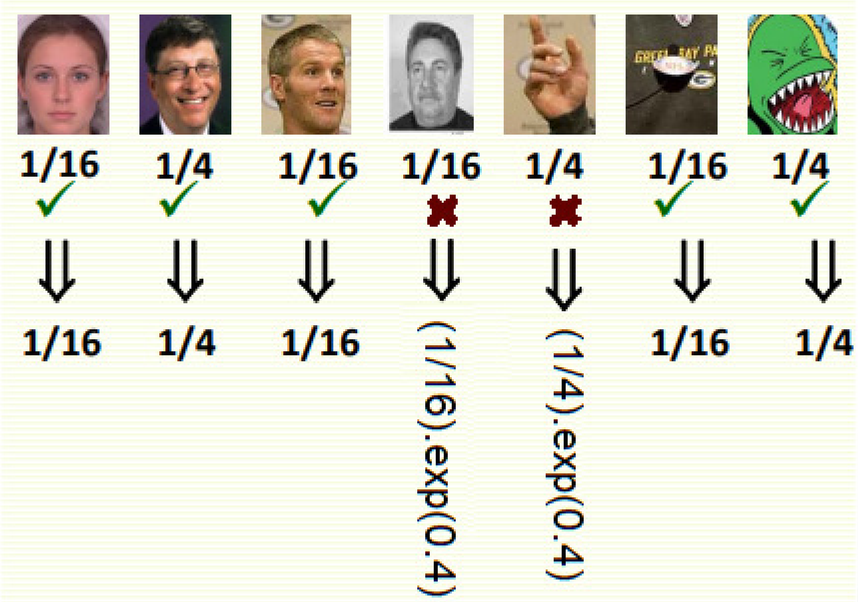
- Recall that $\varepsilon_t < ½$ if weak classifier is better than random
- The smaller is ε_t, the larger is α_t, and thus the more importance (weight) classifier $h_t(x)$

AdaBoost: Step 4

[4] For each x^i, update weights for next iteration

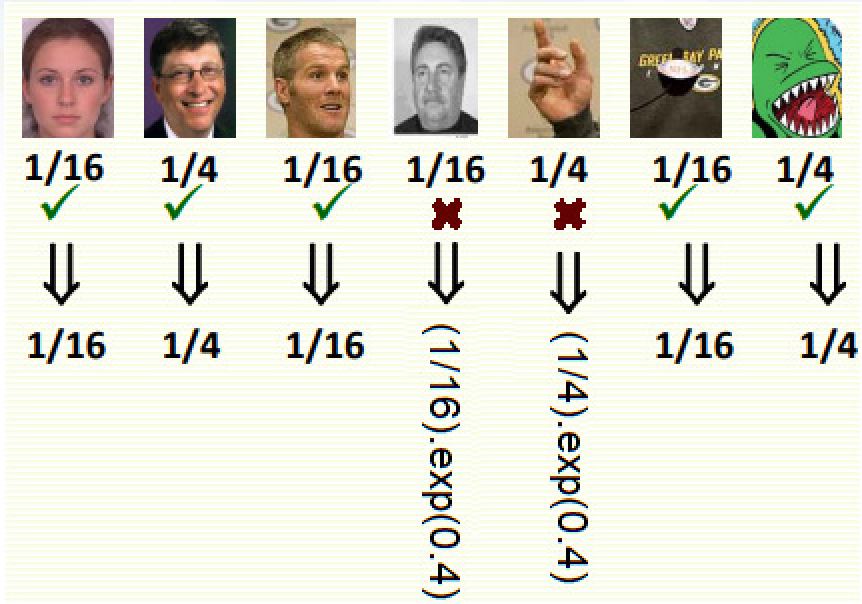
$$D_{t+1}(x^i) = D_t(x^i) \cdot \exp(\alpha_t \cdot I[y^i \neq h_t(x^i)])$$

- From previous slide $\alpha_t = 0.4$



- Weight of misclassified examples is increased

AdaBoost: Step 4



- First example (face) increased to 0.093 from 0.062 (which was $\frac{1}{16}$)
- Second example (face) increased to 0.373 from 0.25

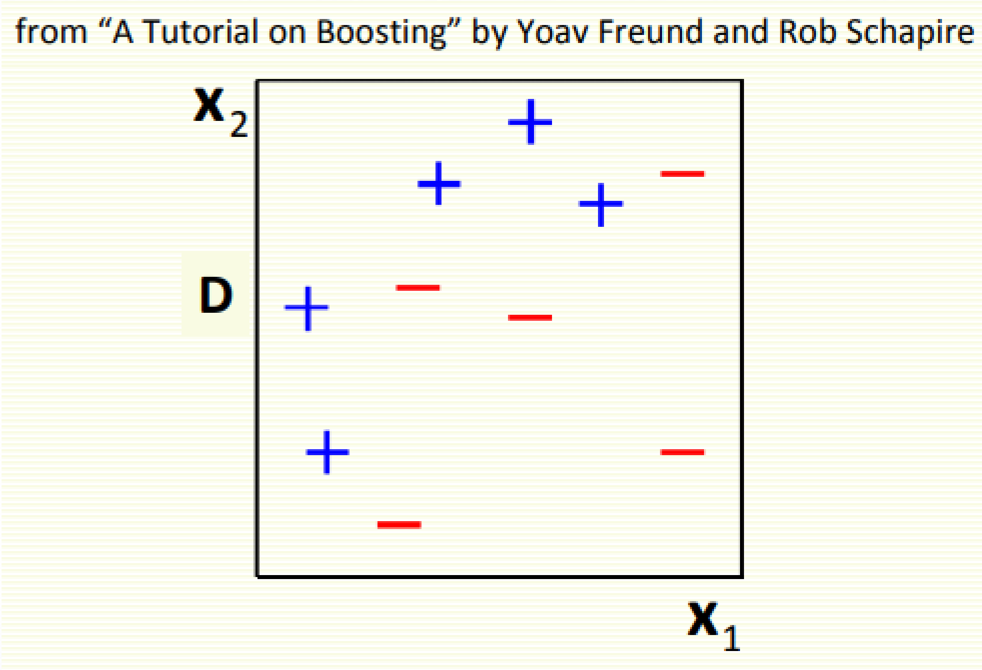
AdaBoost: Step 5

[5] Normalize $D\left(x^{i}\right)$ so that $\sum D\left(x^{i}\right) = 1$

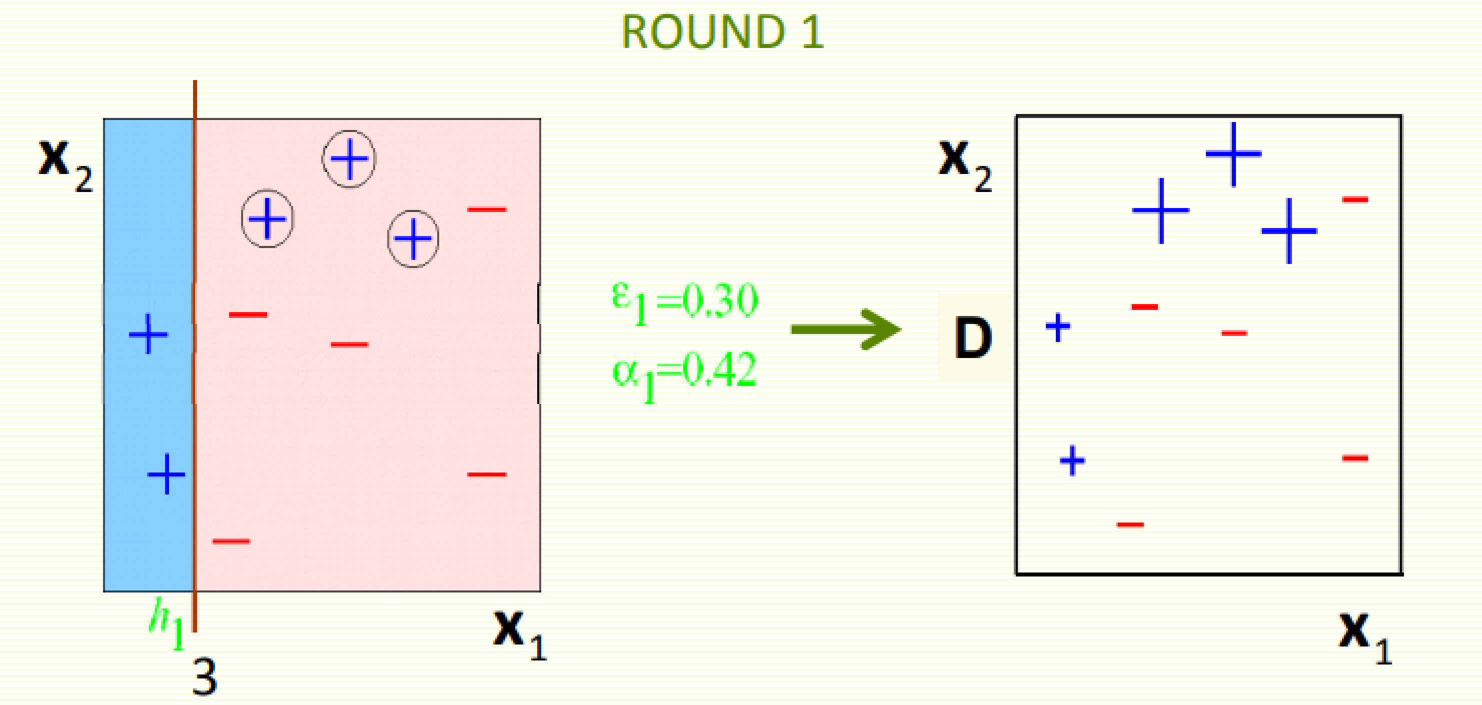
- *Then start over!*

AdaBoost Example

- Initialization: all examples have equal weights



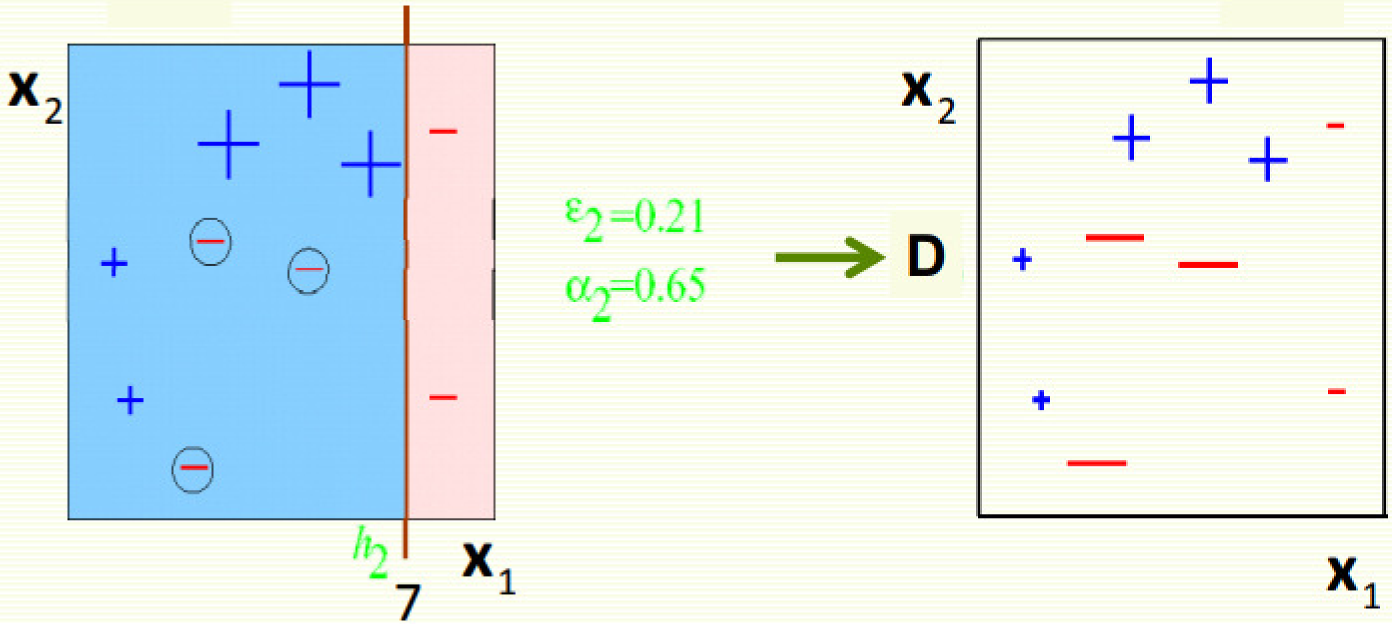
AdaBoost Example



$$h_1(x) = \text{sign}(3 - x_1)$$

AdaBoost Example

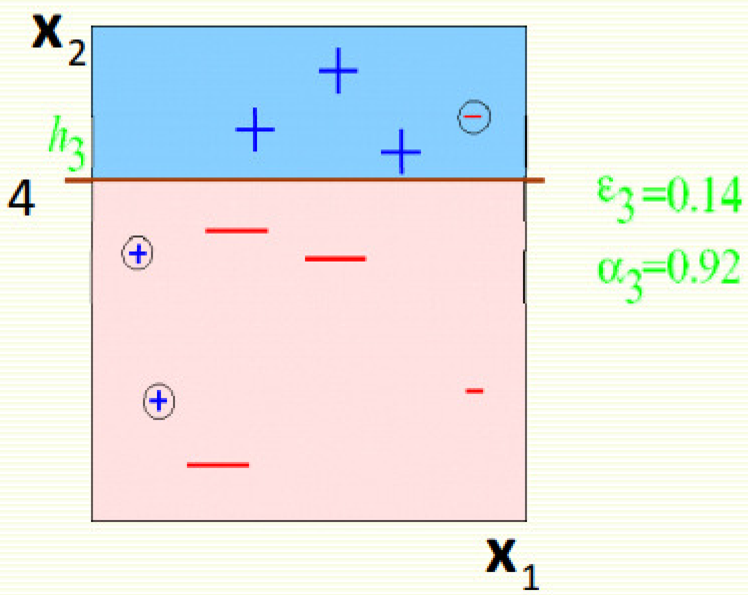
ROUND 2



$$h_2(x) = \mathrm{sign}(7 - x_1)$$

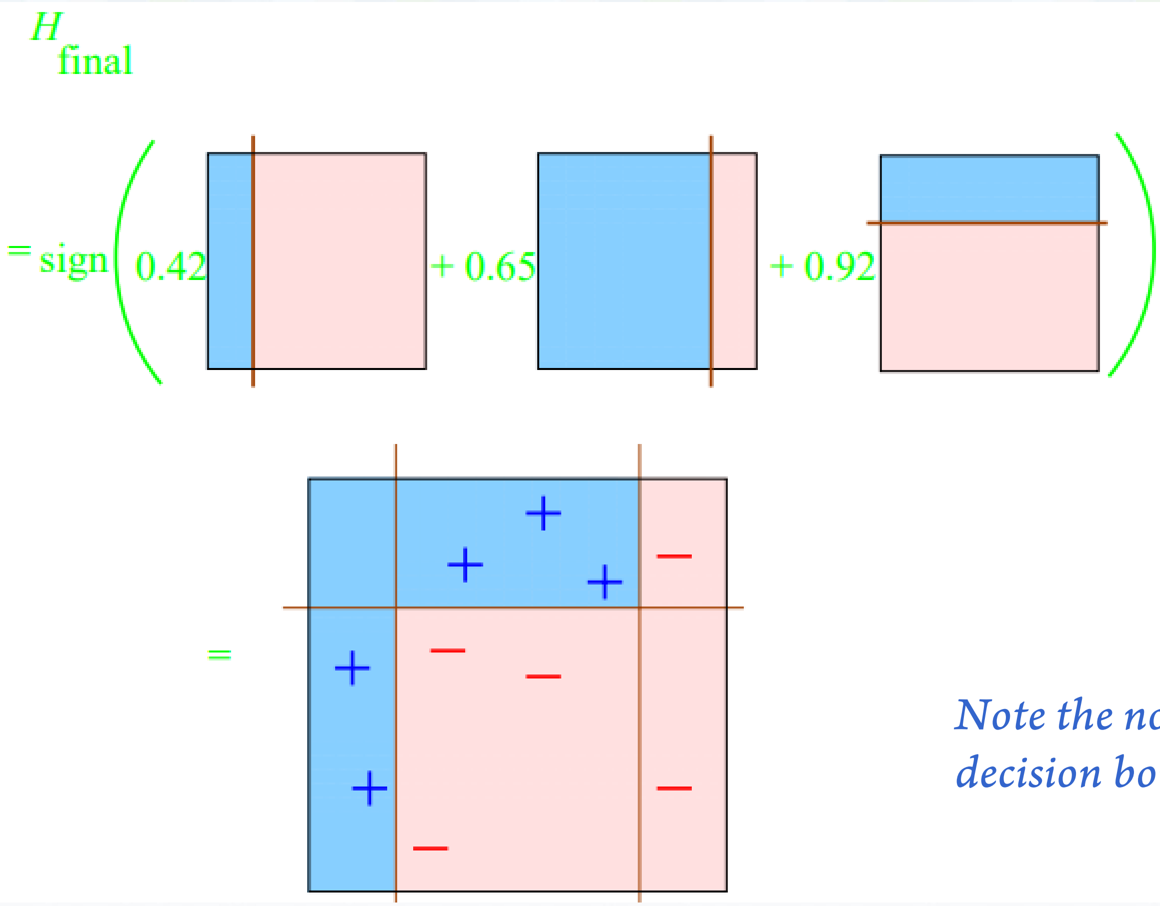
AdaBoost Example

ROUND 3



$$h_3(x) = \text{sign}(x_2 - 4)$$

AdaBoost Example



Note the non-linear decision boundary

$$\mathrm{sign}\left(0.42\,\mathrm{sign}(3-x_1)+0.65\,\mathrm{sign}(7-x_1)+0.92\,\mathrm{sign}(x_2-4)\right)$$

AdaBoost Advantages

- Can construct arbitrarily complex decision regions
- Fast
- Simple
- Has only one parameter to tune ,T
- Flexible: can be combined with any classifier
- Provably effective (assuming weak learner)
  - Shift in mind set: goal now is merely to find hypotheses that are better than random guessing!
- While overfitting could occur, it would need a lot of iterations (which isn't always necessary for final classifier)

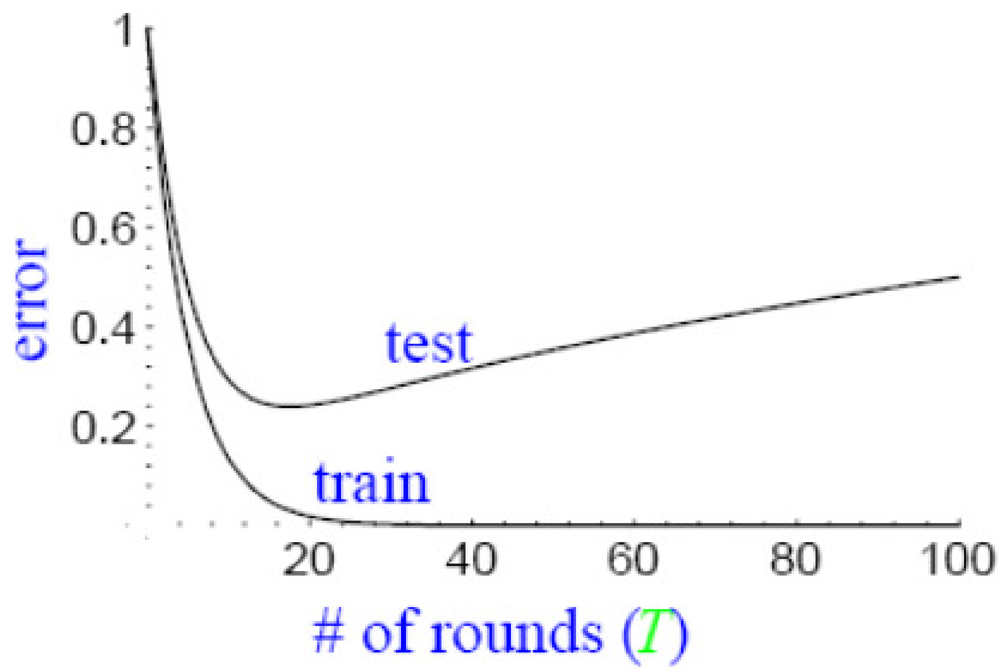
Caveats

- AdaBoost can <u>fail</u> if
  - If weak hypothesis (weak learners) are too complex
    - Overfitting
  - If weak hypothesis (weak learners) are too weak
    - Underfitting

- Empirically, AdaBoost seems especially susceptible to noise
  - Data with wrong labels

Test Error // AdaBoost

- Think about overfitting/underfitting
- Occam's Razor



Sample Snippets of Code

```
from sklearn.ensemble import AdaBoostClassifier #For Classification
from sklearn.ensemble import AdaBoostRegressor #For Regression
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()
clf = AdaBoostClassifier(n_estimators=100, base_estimator=dt,learning_rate=1)
# Decision tree was used as a base estimator, you can use any
# ML learner as base estimator if it accepts sample weight
clf.fit(x_train,y_train)
```

You can tune the parameters to optimize the performance of algorithms Some key parameters for tuning mentioned below:

n_estimators: It controls the number of weak learners.

learning_rate: Controls the contribution of weak learners in the final combination. There is a trade-off between learning_rate and n_estimators.

base_estimators: It helps to specify different ML algorithm.

You can also tune the parameters of base learners to optimize its performance.

Additional Material

More on Bagging

- The perturbation in the training set due to the bootstrap resampling causes different hypotheses to be built, particularly if the classifier is unstable
  - A classifier is said to be unstable if a small change in the training data (e.g., order of presentation of examples) can lead to a radically different hypothesis. This is the case of decision trees and, arguably, neural networks
- Bagging can be expected to improve accuracy if the induced classifiers are uncorrelated
  - In some cases, such as k Nearest Neighbors, bagging has been shown to degrade performance as compared to individual classifiers as a result of an effectively smaller training set
- A related approach to bagging is "cross-validated committees", in which the component classifiers are built on different partitions of the training set obtained through k-fold cross-validation

Another Description of AdaBoost

AdaBoost operates as follows

- At iteration n, boosting provides the weak learner with a distribution D_n over the training set, where $D_n(i)$ represents the probability of selecting the i-th example
  - The initial distribution is uniform: $D_1(i) = 1/N$. Thus, all examples are equally likely to be selected for the first component
- The weak learner subsamples the training set according to D_n and generates a trained model or hypothesis H_n
- The error rate of H_n is measured with respect to the distribution D_n
- A new distribution D_{n+1} is produced by decreasing the probability of those examples that were correctly classified, and increasing the probability of the misclassified examples
- The process is repeated T times, and a final hypothesis is obtained by weighting the votes of individual hypotheses $\{h_1, h_2, \ldots, h_T\}$ according to their performance

Note

- The strength of AdaBoost derives from the adaptive re-sampling of examples, not from the final weighted combination
  - To prove this point Breiman developed a variant of AdaBoost, known as 'arc-x4', in which the ensemble voting is unweighted [Breiman, 1996]: his results show that AdaBoost (referred to as 'arc-fs') and 'arc-x4' have similar performance [Bauer and Kohavi, 1999]

The Bias and Variance Decomposition

The effectiveness of Bagging and Boosting has been explained in terms of the bias-variance decomposition of classification error

- The expected error of a learning algorithm can be decomposed into
  - A **bias** term that measures how closely the average classifier produced by the learning algorithm matches the target function
  - A **variance** term that measures how much the learning algorithm's predictions fluctuate for different training sets (of the same size)
  - An **intrinsic target noise**, which is the minimum error that can be achieved: that of the Bayes optimal classifier
- Following this line of reasoning, Breiman has suggested that both Bagging and Boosting reduce errors by reducing the variance term
- Along the same lines, Freund and Schapire have argued that Boosting also attempts to reduce the bias term since it focuses on misclassified samples
  - Work by Bauer and Kohavi, however, seems to indicate that Bagging can also reduce the bias term

References:

~Ensemble Learning ~ R. Gutierrez-Osuna ~ Pattern Analysis ~ TAMU

~Analytics Vidhya ~ Quick Introduction to Boosting Algorithms in Machine Learning ~ S.Ray

~AdaBoost Tutorial ~ C.McCormick

~Boosting ~ Machine Learning ~ O.Veksler

~Boosting (AdaBoost Algorithm) ~ E.Emer

~Excellent paper on AdaBoost written by one of the original authors of the algorithm, *Robert Schapire*: http://rob.schapire.net/papers/explaining-adaboost.pdf

~A Tutorial on Boosting ~ Y.Freund and R.Schapire ~ di.unipi.it