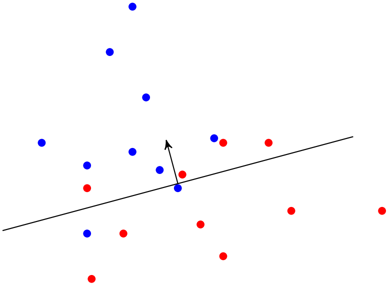
BOOSTING

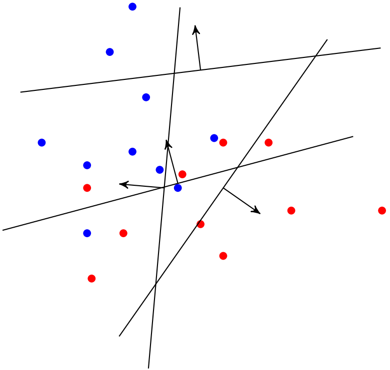
ENSEMBLES

A *randomly* chosen hyperplane classifier has an *expected* error of 0.5 (i.e. 50%).



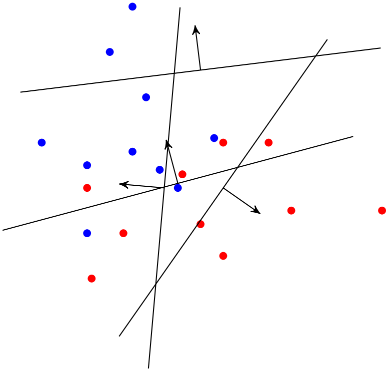
ENSEMBLES

A *randomly* chosen hyperplane classifier has an *expected* error of 0.5 (i.e. 50%).



ENSEMBLES

A *randomly* chosen hyperplane classifier has an *expected* error of 0.5 (i.e. 50%).



- Many random hyperplanes combined by majority vote: Still 0.5.
- A single classifier slightly better than random: $0.5 + \varepsilon$.
- What if we use *m* such classifiers and take a majority vote?

VOTING

Decision by majority vote

- m individuals (or classifiers) take a vote. m is an odd number.
- They decide between two choices; one is correct, one is wrong.
- After everyone has voted, a decision is made by simple majority.

Note: For two-class classifiers $f_1, \dots, f_m$ (with output ± 1):

$$\text{majority vote } = \operatorname{sgn}\Big(\sum_{j=1}^{m} f_j\Big)$$

Assumptions

Before we discuss ensembles, we try to convince ourselves that voting can be beneficial. We make some simplifying assumptions:

- Each individual makes the right choice with probability $p \in [0, 1]$.
- The votes are *independent*, i.e. stochastically independent when regarded as random outcomes.

Does the Majority Make the Right Choice?

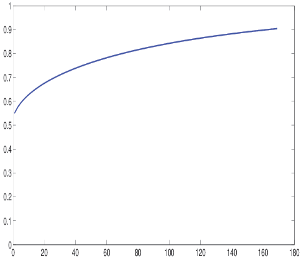
Condorcet's rule

If the individual votes are independent, the answer is

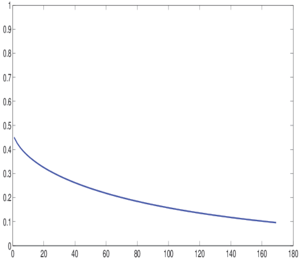
$$\Pr\{\text{ majority makes correct decision }\} = \sum_{j=\frac{m+1}{2}}^{m} \frac{m!}{j!(m-j)!} p^j (1-p)^{m-j}$$

This formula is known as **Condorcet's jury theorem**.

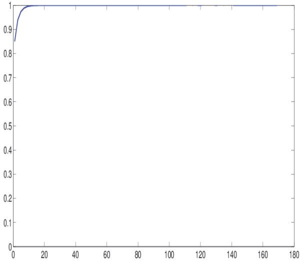
Probability as function of the number of votes



$p = 0.55$



$p = 0.45$



$p = 0.85$

Ensemble Methods

Terminology

- An **ensemble method** makes a prediction by combining the predictions of many classifiers into a single vote.
- The individual classifiers are usually required to perform only slightly better than random. For two classes, this means slightly more than 50% of the data are classified correctly. Such a classifier is called a **weak learner**.

Strategy

- We have seen above that if the weak learners are random and independent, the prediction accuracy of the majority vote will increase with the number of weak learners.
- Since the weak learners all have to be trained on the training data, producing random, independent weak learners is difficult.
- Different ensemble methods (e.g. Boosting, Bagging, etc) use different strategies to train and combine weak learners that behave relatively independently.

Methods We Will Discuss

Boosting

- After training each weak learner, data is modified using weights.
- Deterministic algorithm.

Bagging

Each weak learner is trained on a random subset of the data.

Random forests

- Bagging with tree classifiers as weak learners.
- Uses an additional step to remove dimensions in $\mathbb{R}^d$ that carry little information.

Boosting

- Arguably the most popular (and historically the first) ensemble method.
- Weak learners can be trees (decision stumps are popular), Perceptrons, etc.
- Requirement: It must be possible to train the weak learner on a *weighted* training set.

Overview

- Boosting adds weak learners one at a time.
- A weight value is assigned to each training point.
- At each step, data points which are currently classified correctly are weighted down (i.e. the weight is smaller the more of the weak learners already trained classify the point correctly).
- The next weak learner is trained on the *weighted* data set: In the training step, the error contributions of misclassified points are multiplied by the weights of the points.
- Roughly speaking, each weak learner tries to get those points right which are currently not classified correctly.

Training With Weights

Example: Decision stump

A decision stump classifier for two classes is defined by

$$f(\,\mathbf{x}\,|\,j,t\,) := \begin{cases} +1 & x^{(j)} > t \\ -1 & \text{otherwise} \end{cases}$$

where $j \in \{1,\ldots,d\}$ indexes an axis in $\mathbb{R}^d$.

Weighted data

- Training data $(\tilde{\mathbf{x}}_1, \tilde{y}_1), \ldots, (\tilde{\mathbf{x}}_n, \tilde{y}_n)$.
- With each data point $\tilde{\mathbf{x}}_i$ we associate a weight $w_i \geq 0$.

Training on weighted data

Minimize the *weighted* misclassifcation error:

$$(j^*, t^*) := \arg\min_{j,t} \frac{\sum_{i=1}^n w_i \mathbb{I}\{\tilde{y}_i \neq f(\tilde{\mathbf{x}}_i | j,t)\}}{\sum_{i=1}^n w_i}$$

AdaBoost

Input

- Training data $(\tilde{\mathbf{x}}_1, \tilde{y}_1), \dots, (\tilde{\mathbf{x}}_n, \tilde{y}_n)$
- Algorithm parameter: Number M of weak learners

Training algorithm

1. Initialize the observation weights $w_i = \frac{1}{n}$ for $i = 1, 2, ..., n$.
2. For $m = 1$ to M:

   2.1 Fit a classifier $g_m(x)$ to the training data using weights w_i.

   2.2 Compute

   $$\text{err}_m := \frac{\sum_{i=1}^{n} w_i \mathbb{I}\{y_i \neq g_m(x_i)\}}{\sum_i w_i}$$

   2.3 Compute $\alpha_m = \log(\frac{1-\text{err}_m}{\text{err}_m})$

   2.4 Set $w_i \leftarrow w_i \cdot \exp(\alpha_m \cdot \mathbb{I}(y_i \neq g_m(x_i)))$ for $i = 1, 2, ..., n$.
3. Output

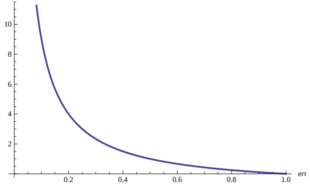
$$f(x) := \text{sign}\left(\sum_{m=1}^{M} \alpha_m g_m(x)\right)$$

AdaBoost

Weight updates

$$\alpha_m = \log\left(\frac{1-\mathrm{err}_m}{\mathrm{err}_m}\right)$$

$$w_i^{(\text{m})} = w_i^{(\text{m-1})} \cdot \exp(\alpha_m \cdot \mathbb{I}(y_i \neq g_m(x_i)))$$



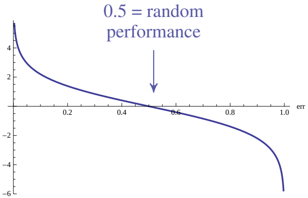
$\frac{1-\mathrm{err}_m}{\mathrm{err}_m}$

Hence:

$$w_i^{(\text{m})} = \begin{cases} w_i^{(\text{m-1})} & \text{if } g_m \text{ classifies } x_i \text{ correctly} \\ w_i^{(\text{m-1})} \cdot \frac{1-\mathrm{err}_m}{\mathrm{err}_m} & \text{if } g_m \text{ misclassifies } x_i \end{cases}$$

Weighted classifier

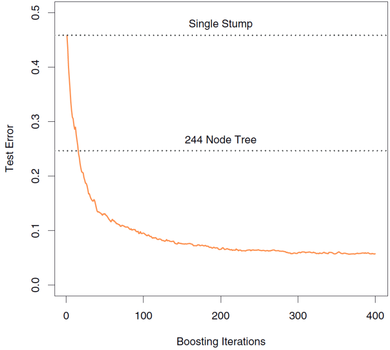
$$f(x) = \mathrm{sign}\left(\sum_{m=1}^{M} \alpha_m g_m(x)\right)$$



$\alpha_m = \log\left(\frac{1-\mathrm{err}_m}{\mathrm{err}_m}\right)$

EXAMPLE

AdaBoost test error (simulated data)



- Weak learners used are decision stumps.
- Combining many trees of depth 1 yields much better results than a single large tree.

Boosting: Properties

Properties

- AdaBoost is one of most widely used classifiers in applications.
- Decision boundary is non-linear.
- Can handle multiple classes if weak learner can do so.

Test vs training error

- Most training algorithms (e.g. Perceptron) terminate when training error reaches minimum.
- AdaBoost weights keep changing even if training error is minimal.
- Interestingly, the *test error* typically keeps decreasing even *after* training error has stabilized at minimal value.
- It can be shown that this behavior can be interpreted in terms of a margin:
  - Adding additional classifiers slowly pushes overall f towards a maximum-margin solution.
  - May not improve training error, but improves generalization properties.
- This does *not* imply that boosting magically outperforms SVMs, only that minimal test error does not imply an optimal solution.

BOOSTING AND FEATURE SELECTION

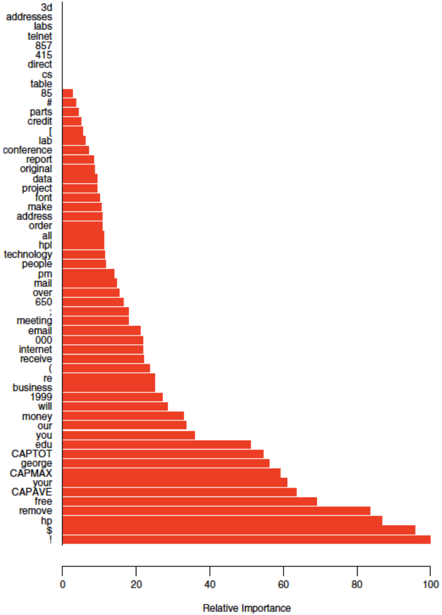
AdaBoost with Decision Stumps

- Once AdaBoost has trained a classifier, the weights α_m tell us which of the weak learners are important (i.e. classify large subsets of the data well).
- If we use Decision Stumps as weak learners, each f_m corresponds to one axis.
- From the weights α, we can read off which axis are important to separate the classes.

Terminology

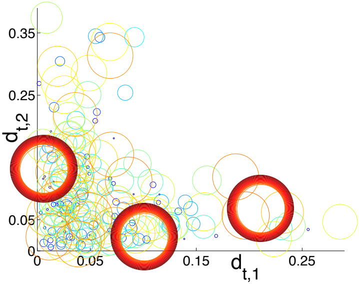
The dimensions of $\mathbb{R}^d$ (= the measurements) are often called the **features** of the data. The process of selecting features which contain important information for the problem is called **feature selection**. Thus, AdaBoost with Decision Stumps can be used to perform feature selection.

SPAM DATA



- Tree classifier: 9.3% overall error rate
- Boosting with decision stumps: 4.5%
- Figure shows feature selection results of Boosting.

CYCLES



- An odd property of AdaBoost is that it can go into a cycle, i.e. the same sequence of weight configurations occurs over and over.
- The figure shows weights (called d_t by the authors of the paper, with t=iteration number) for two weak learners.
- Circle size indicates iteration number, i.e. larger circel indicates larger t.

APPLICATION: FACE DETECTION

FACE DETECTION

Searching for faces in images

Two problems:

- **Face detection** Find locations of all faces in image. Two classes.
- **Face recognition** Identify a person depicted in an image by recognizing the face. One class per person to be identified + background class (all other people).

Face detection can be regarded as a solved problem. Face recognition is not solved.

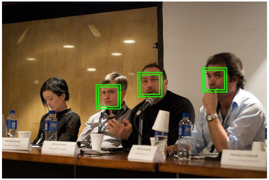
Face detection as a classification problem

- Divide image into patches.
- Classify each patch as "face" or "not face"

Classifier Cascades

Unbalanced Classes

- Our assumption so far was that both classes are roughly of the same size.
- Some problems: One class is much larger.
- Example: Face detection.
  - Image subdivided into small quadratic patches.
  - Even in pictures with several people, only small fraction of patches usually represent faces.



Standard classifier training

Suppose positive class is very small.

- Training algorithm can achieve good error rate by classifiying *all* data as negative.
- The error rate will be precisely the proportion of points in positive class.

Classifier Cascades

Addressing class imbalance

- We have to change cost function: False negatives (= classify face as background) expensive.
- Consequence: Training algorithm will focus on keeping proportion of false negatives small.
- Problem: Will result in many false positives (= background classified as face).

Cascade approach

- Use many classifiers linked in a chain structure ("cascade").
- Each classifier eliminates part of the negative class.
- With each step down the cascade, class sizes become more even.

Classifier Cascades

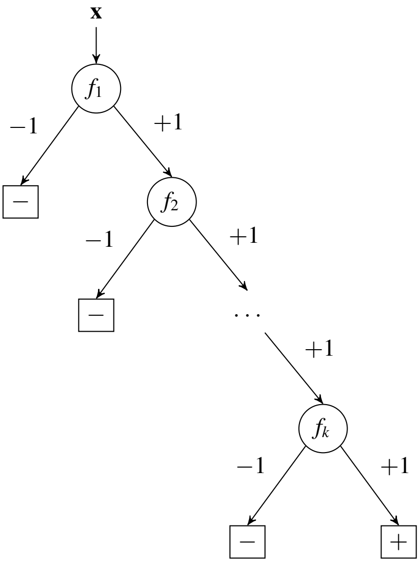
Training a cascade

Use imbalanced loss (very low false negative rate for each f_j).

1. Train classifier f_1 on entire training data set.
2. Remove all $\tilde{\mathbf{x}}_i$ in negative class which f_1 classifies correctly from training set.
3. On smaller training set, train f_2.
4. ...
5. On remaining data at final stage, train f_k.

Classifying with a cascade

- If any f_j classifies $\mathbf{x}$ as negative, $f(\mathbf{x}) = -1$.
- Only if all f_j classify $\mathbf{x}$ as positive, $f(\mathbf{x}) = +1$.



WHY DOES A CASCADE WORK?

We have to consider two rates

$$\text{false positive rate} \qquad \mathrm{FPR}(f_j) = \frac{\#\text{negative points classified as "+1"}}{\#\text{negative training points at stage } j}$$

$$\text{detection rate} \qquad \mathrm{DR}(f_j) = \frac{\#\text{correctly classified positive points}}{\#\text{positive training points at stage } j}$$

We want to achieve a low value of $\mathrm{FPR}(f)$ and a high value of $\mathrm{DR}(f)$.

Class imbalance

In face detection example:

- Number of faces classified as background is (size of face class) $\times$ $(1 - \mathrm{DR}(f))$
- We would like to see a decently high detection rate, say 90%
- Number of background patches classified as faces is (size of background class) $\times$ $(\mathrm{FPR}(f))$
- Since background class is huge, $\mathrm{FPR}(f)$ has to be *very* small to yield roughly the same amount of errors in both classes.

WHY DOES A CASCADE WORK?

Cascade detection rate

The rates of the overall cascade classifier f are

$$\mathrm{FPR}(f) = \prod_{j=1}^{k} \mathrm{FPR}(f_j) \qquad \mathrm{DR}(f) = \prod_{j=1}^{k} \mathrm{DR}(f_j)$$

- Suppose we use a 10-stage cascade ($k = 10$)
- Each $\mathrm{DR}(f_j)$ is 99% and we permit $\mathrm{FPR}(f_j)$ of 30%.
- We obtain $\mathrm{DR}(f) = 0.99^{10} \approx 0.90$ and $\mathrm{FPR}(f) = 0.3^{10} \approx 6 \times 10^{-6}$