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.

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

Weighted data

- Training data $(\tilde{\mathbf{x}}_1, \tilde{y}_1), \dots, (\tilde{\mathbf{x}}_n, \tilde{y}_n)$.
- 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:

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

Input

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

Training algorithm

1. Initialize the observation weights $w_i = \frac{1}{n}$ for $i = 1, 2, \ldots, n$.
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 $i = 1, 2, \ldots, n$.
3. Output
   $$f(x) := \text{sign}\left(\sum_{m=1}^{M} \alpha_m g_m(x)\right)$$

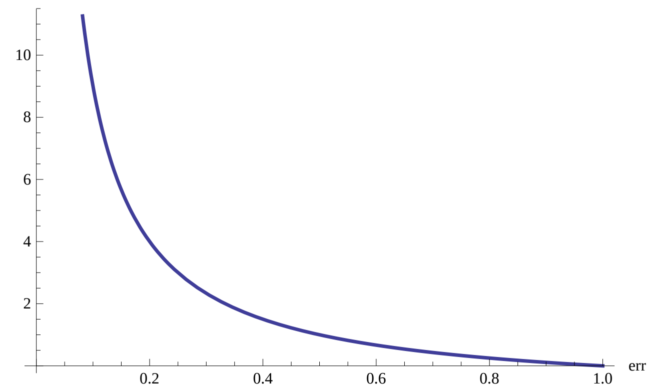
Weight updates

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

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

Hence:

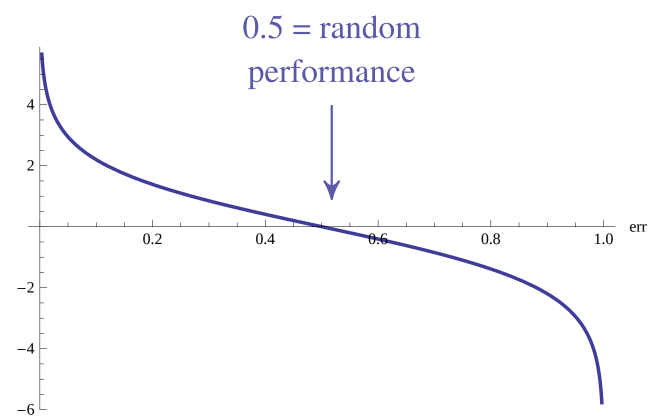
$$w_i^{(\mathrm{m})} = \begin{cases} w_i^{(\mathrm{m-1})} & \text{if } g_m \text{ classifies } x_i \text{ correctly} \\ w_i^{(\mathrm{m-1})} \cdot \frac{1-\mathrm{err}_m}{\mathrm{err}_m} & \text{if } g_m \text{ misclassifies } x_i \end{cases}$$



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

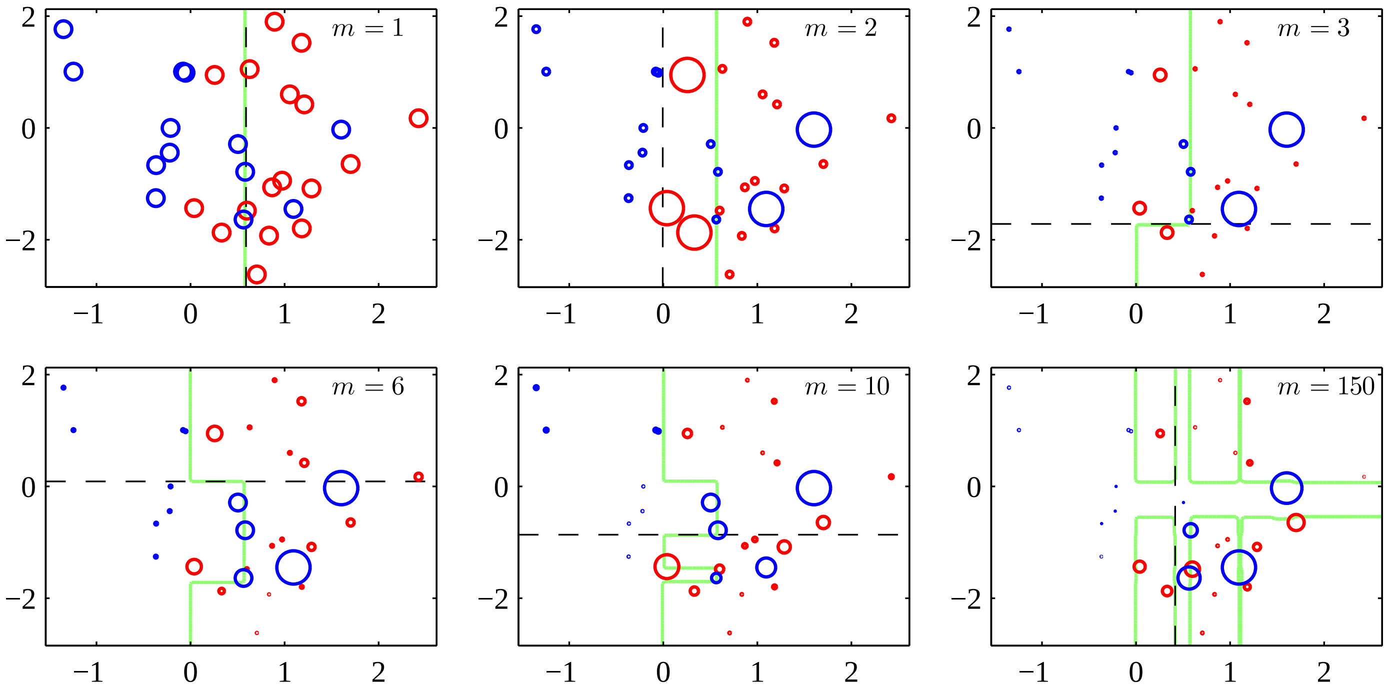
Weighted classifier

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



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

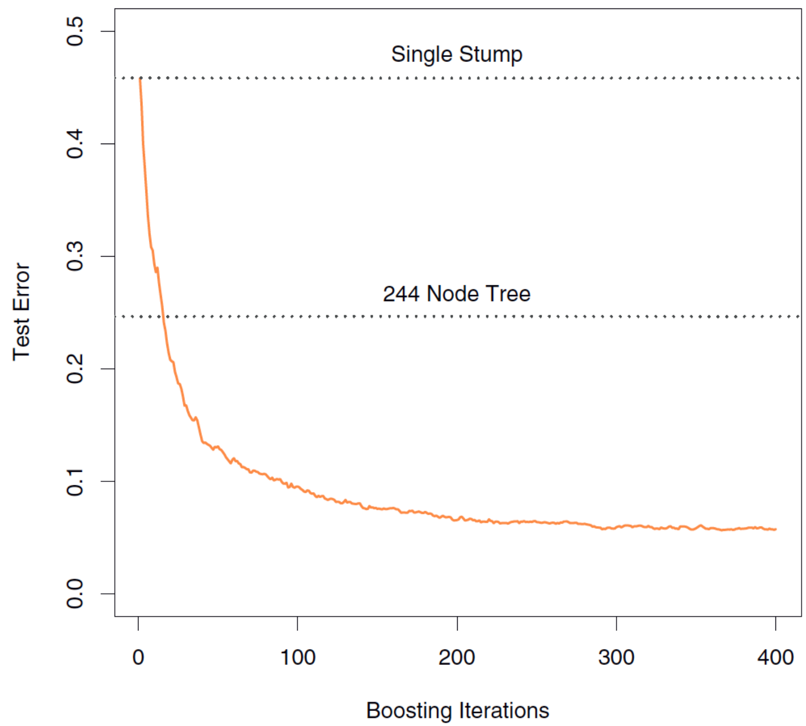
ILLUSTRATION



Circle = data points, circle size = weight.

Dashed line: Current weak learner. Green line: Aggregate decision boundary.

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 training error does not imply an optimal solution.

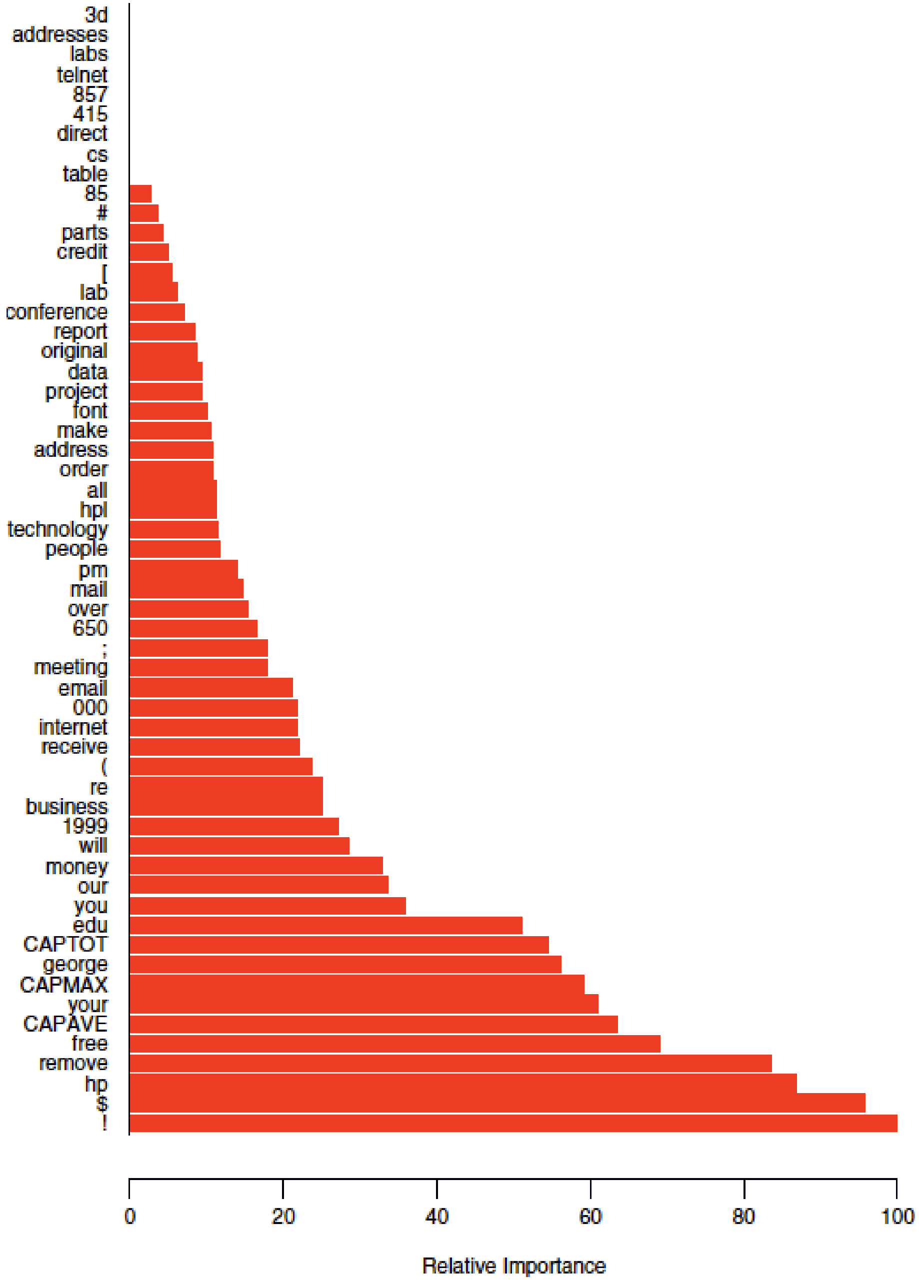
AdaBoost with Decision Stumps

- 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

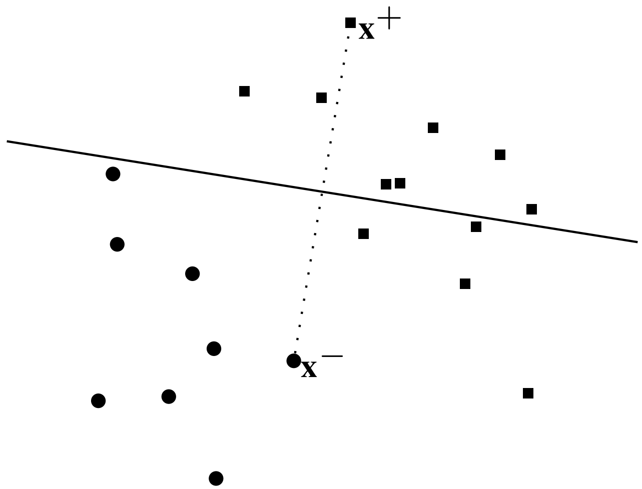
The dimensions of $\mathbb{R}^d$ (= the measurements) are often called the **features** of the data. The process of selecting features which contain important information for the problem is called **feature selection**. Thus, AdaBoost with Decision Stumps can be used to perform feature selection.

Spam Data



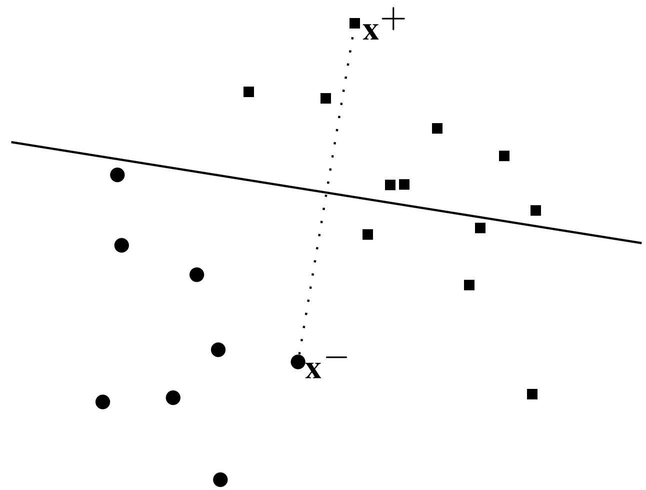
- Tree classifier: 9.3% overall error rate
- Boosting with decision stumps: 4.5%
- Figure shows feature selection results of Boosting.

HOMEWORK: A PRIMITIVE ENSEMBLE



Idea

- Try to implement the “randomly throwing out hyperplanes” idea directly.
- Strategy: Build a “weak lerner” by selecting two points at random and let them determine a hyperplane.



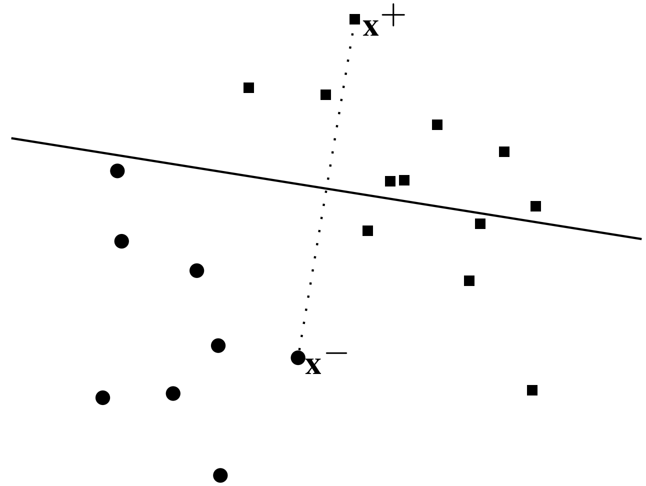
Weak classifier

- Choose two training data points $\mathbf{x}^-$ and $\mathbf{x}^+$, one in each class.
- Place an affine plane "in the middle" between the two:

$$\mathbf{w} := \frac{\mathbf{x}^+ - \mathbf{x}^-}{\|\mathbf{x}^+ - \mathbf{x}^-\|} \quad \text{and} \quad c := \left\langle \mathbf{w}, \mathbf{x}^- + \frac{1}{2}(\mathbf{x}^+ - \mathbf{x}^-) \right\rangle$$

- Choose the orientation with smaller training error: Define weak classifier as

$$f(\,.\,) = \operatorname{sgn}(\langle\,.\,,\mathbf{v}\rangle - c) \qquad \text{where either } \mathbf{v} := \mathbf{w} \text{ or } \mathbf{v} := -\mathbf{w}\ .$$



Ensemble training

- Split the available data into to equally sized parts (training and test).
- Select m pairs of points $(\mathbf{x}_1^-, \mathbf{x}_1^+), \dots, (\mathbf{x}_m^-, \mathbf{x}_m^+)$ uniformly (with replacement).
- For each such pair $(\mathbf{x}_i^-, \mathbf{x}_i^+)$, compute the classifer f_i given by $(\mathbf{v}_i, c_i)$ as described above.
- The overall classifier g_m is defined as the majority vote

$$g_m(\mathbf{x}) = \operatorname{sgn}\Big(\sum_{j=1}^{m} f_i(\mathbf{x})\Big) = \operatorname{sgn}\Big(\sum_{j=1}^{m} \operatorname{sgn}(\langle \mathbf{v}_i, \mathbf{x}\rangle - c_i)\Big)$$

Application: 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

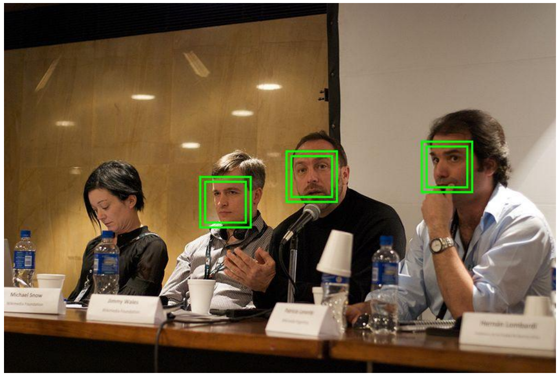
- Divide image into patches.
- Classify each patch as "face" or "not face"

Reference: Viola & Jones, "Robust real-time face detection", Int. Journal of Computer Vision, 2004.

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.

Image source: Wikipedia

Addressing class imbalance

- We have to change cost function: False negatives (= classify face as background) are 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.

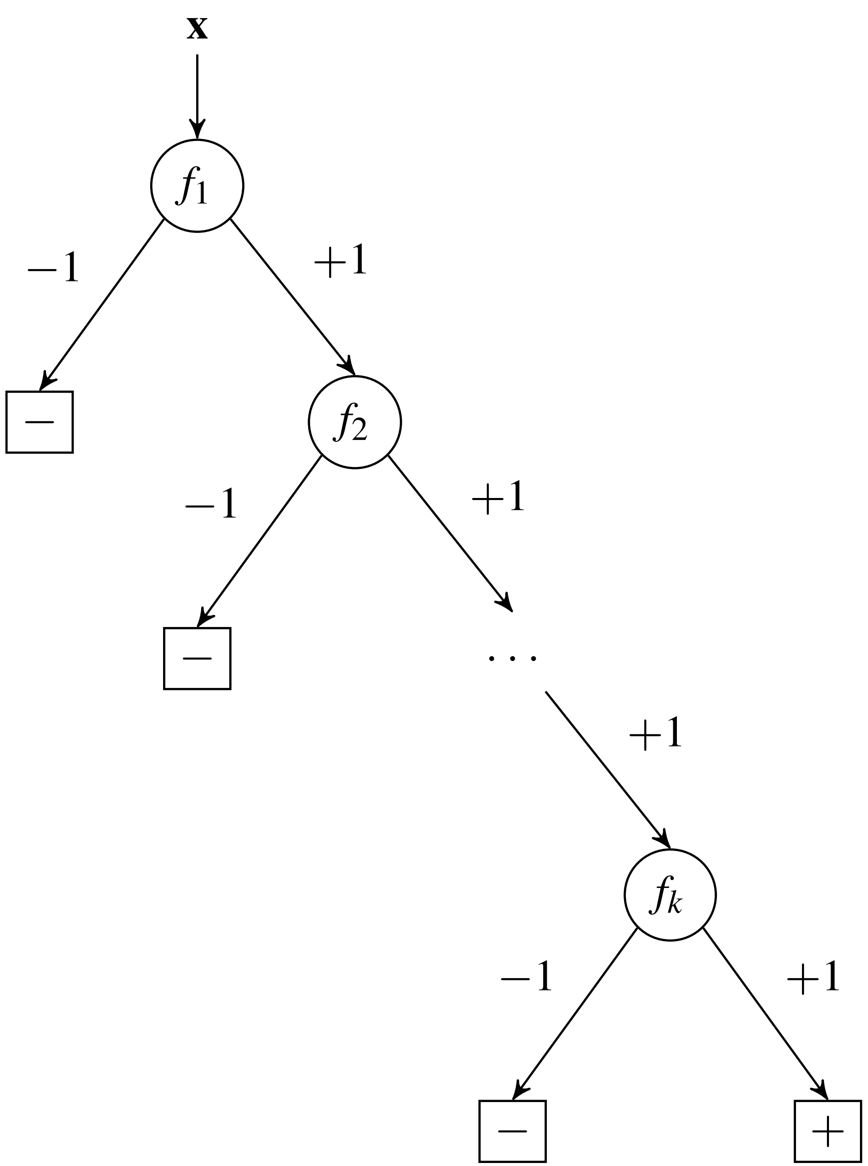
Training a cascade

Use imbalanced loss, with 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, $f(\mathbf{x}) = +1$.



We have to consider two rates

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

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

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

Class imbalance

In face detection example:

- Number of faces classified as background is (size of face class) $\times$ $(1 - \mathrm{Recall}(f))$
- 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.

Cascade recall

The rates of the overall cascade classifier f are

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

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

Objectives

- Classification step should be computationally efficient.
- Expensive training affordable.

Strategy

- Extract very large set of measurements (features), i.e. d in $\mathbb{R}^d$ large.
- Use Boosting with decision stumps.
- From Boosting weights, select small number of important features.
- Class imbalance: Use Cascade.

Classification step

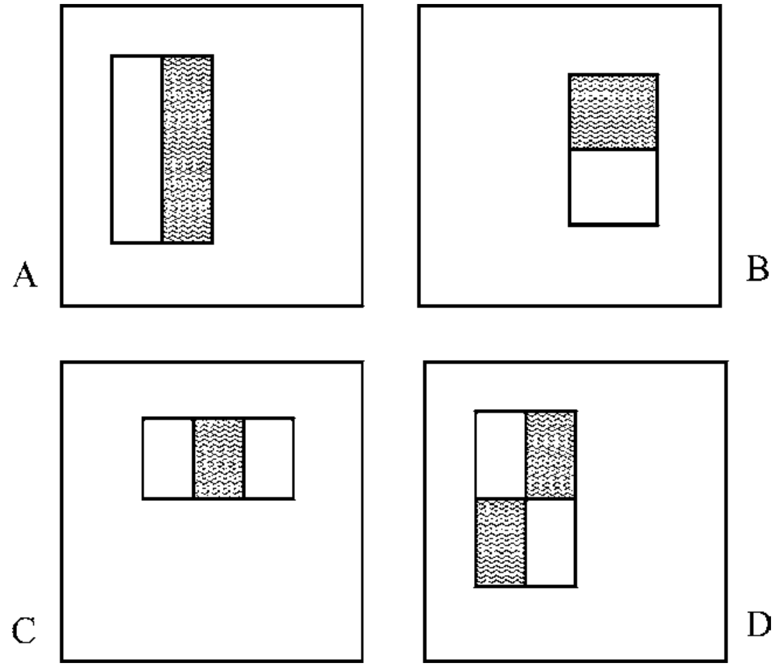
Compute only the selected features from input image.

Extraction method

1. Enumerate possible windows (different shapes and locations) by $j = 1, \ldots, d$.
2. For training image i and each window j, compute

$$x_{ij} := \text{average of pixel values in gray block(s)} - \text{average of pixel values in white block(s)}$$

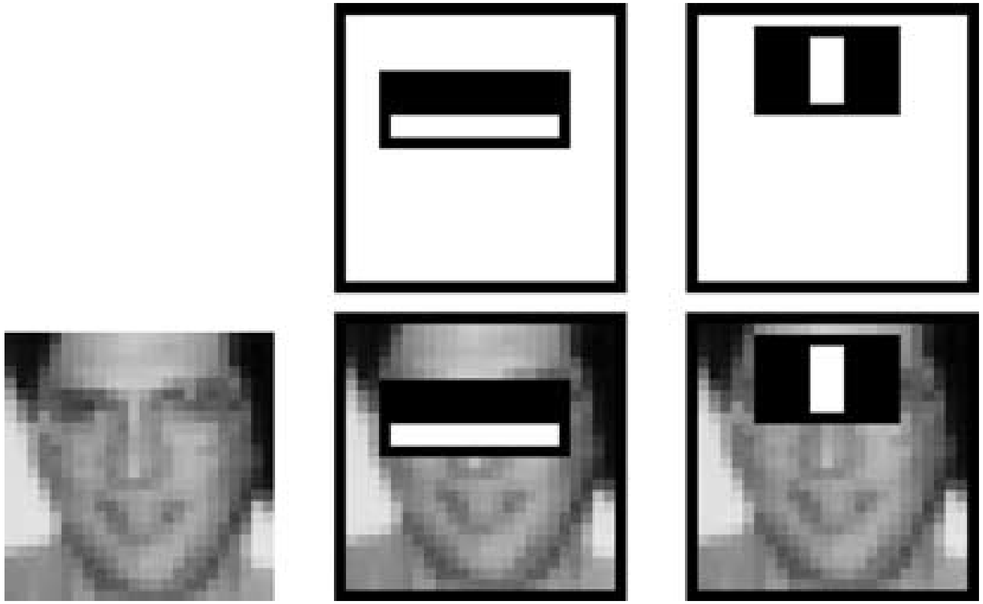
3. Collect values for all j in a vector $\mathbf{x}_i := (x_{i1}, \ldots, x_{id}) \in \mathbb{R}^d$.



The dimension is huge

- One entry for (almost) every possible location of a rectangle in image.
- Start with small rectangles and increase edge length repeatedly by 1.5.
- In Viola-Jones paper: Images are 384×288 pixels, $d \approx 160000$.

First two selected features



200 features are selected in total.

Training procedure

1. User selects acceptable rates (FPR and Recall) for each level of the cascade.
2. At each level of the cascade:
   - Train a boosting classifier.
   - Gradually increase the number of selected features until required rates are achieved.

Use of training data

Each training step uses:

- All positive examples (= faces).
- Negative examples (= non-faces) misclassified at previous cascade layer.

Example Results

Table 3. Detection rates for various numbers of false positives on the MIT + CMU test set containing 130 images and 507 faces.

	False detections							
Detector	10	31	50	65	78	95	167	422
Viola-Jones	76.1%	88.4%	91.4%	92.0%	92.1%	92.9%	93.9%	94.1%
Viola-Jones (voting)	81.1%	89.7%	92.1%	93.1%	93.1%	93.2%	93.7%	–
Rowley-Baluja-Kanade	83.2%	86.0%	–	–	–	89.2%	90.1%	89.9%
Schneiderman-Kanade	–	–	–	94.4%	–	–	–	–
Roth-Yang-Ahuja	–	–	–	–	(94.8%)	–	–	–