Introduction to Machine Learning and Data Mining Lecture-11: Ensemble Methods

Prof. Eugene Chang

Today

- Ensemble methods
  - Bagging
  - AdaBoost
  - Random Forest
- Midterm discussion
- Homework-4 assigned
  - Due 08/02 6pm
- Project proposals

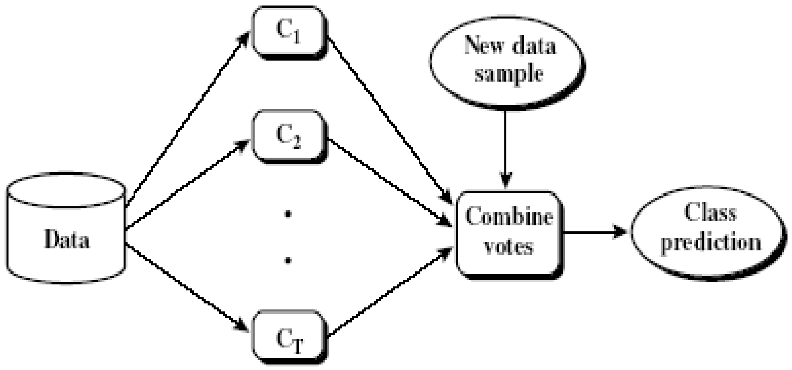
From the Past: Cross-Validation

- Cross validation separates the train data into k-folds
- Trains k classifiers, one for each fold.
- The evaluation measure is constructed from an average of the k evaluations
- No ensemble (combination) is used in the classification scheme.
- The ensemble is used only for evaluation

Ensemble Methods: Increasing the Accuracy

- Ensemble methods
    - Use a combination of models to increase accuracy
    - Combine a series of k learned models, M_1, M_2, …, M_k, with the aim of creating an improved model M*
- Popular ensemble methods
    - Bagging: averaging the prediction over a collection of classifiers
    - Boosting: weighted vote with a collection of classifiers
    - Ensemble: combining a set of heterogeneous classifiers

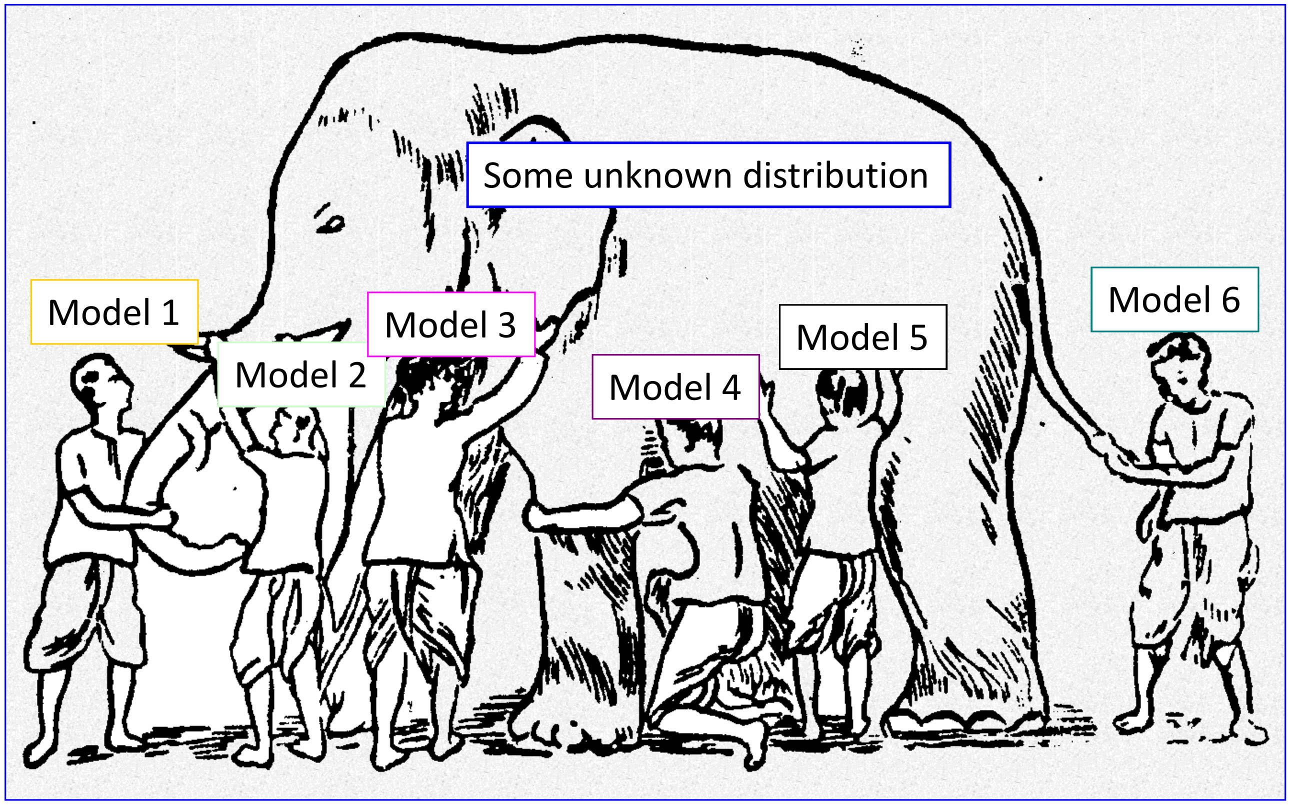
not important.



Why Ensemble Works?

- Intuition
  - combining diverse, independent opinions in human decision-making as a protective mechanism (e.g. stock portfolio)
- Uncorrelated error reduction
  - Suppose we have 5 completely independent classifiers for majority voting
  - If accuracy is 70% for each
    - 10 (.7^3)(.3^2)+5(.7^4)(.3)+(.7^5)
    - **83.7% majority vote accuracy** $\nearrow$ accuracy improves after combining.
  - 101 such classifiers
    - **99.9% majority vote accuracy**

Why Ensemble Works?



Ensemble gives the global picture!

Bagging: Boostrap Aggregation

- Analogy: Diagnosis based on multiple doctors' majority vote
- Training
  - Given a set D of *d* tuples, at each iteration *i*, a training set D_i of *d* tuples is sampled with replacement from D (i.e., bootstrap)
  - A classifier model M_i is learned for each training set D_i
- Classification: classify an unknown sample **X**
  - Each classifier M_i returns its class prediction ← combine all classifyers & use voting to decide.
  - The bagged classifier M* counts the votes and assigns the class with the most votes to **X**
- Prediction: can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple
- Accuracy
  - Often significantly better than a single classifier derived from D
  - For noise data: not considerably worse, more robust
  - Proved improved accuracy in prediction

Bagging

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

- Sampling uniformly with replacement
- Build classifier on each bootstrap sample
- 0.632 bootstrap
- Each bootstrap sample Di contains approx. 63.2% of the original training data
- Remaining (36.8%) are used as test set

Bagging

- Accuracy of bagging:

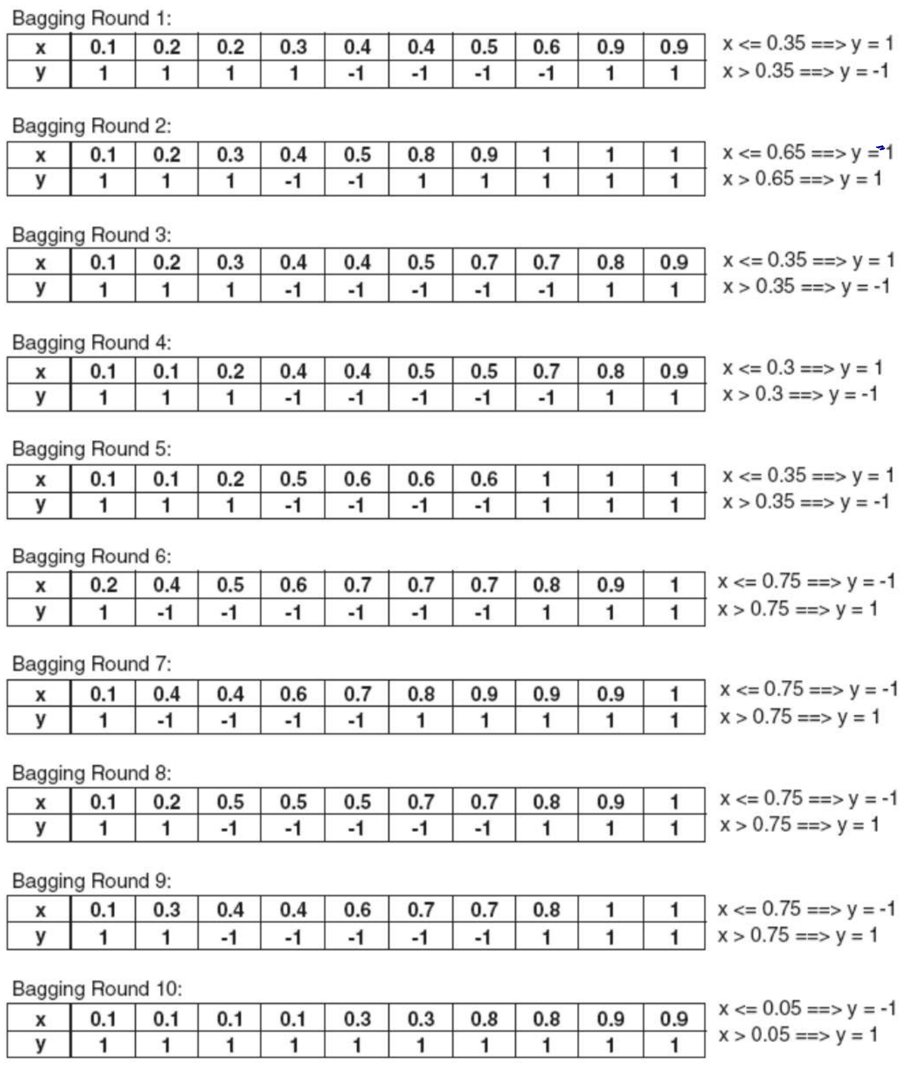
$$Acc(M) = \sum_{i=1}^{k} (0.632 * Acc(M_i)_{test\_set} + 0.368 * Acc(M_i)_{train\_set})$$

- Works well for small data sets
- Example:

X	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
y	1	1	1	-1	-1	-1	-1	1	1	1

Bagging

- Decision Stump ← root of tree, mean.
  1 level tree
- Single level decision binary tree
- Entropy – x<=0.35 or x<=0.75
- Accuracy at most 70%



Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.8	0.9	1	1	1
y	1	1	1	-1	-1	1	1	1	1	1

x <= 0.65 ==> y = -1
x > 0.65 ==> y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ==> y = 1
x > 0.3 ==> y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 10:

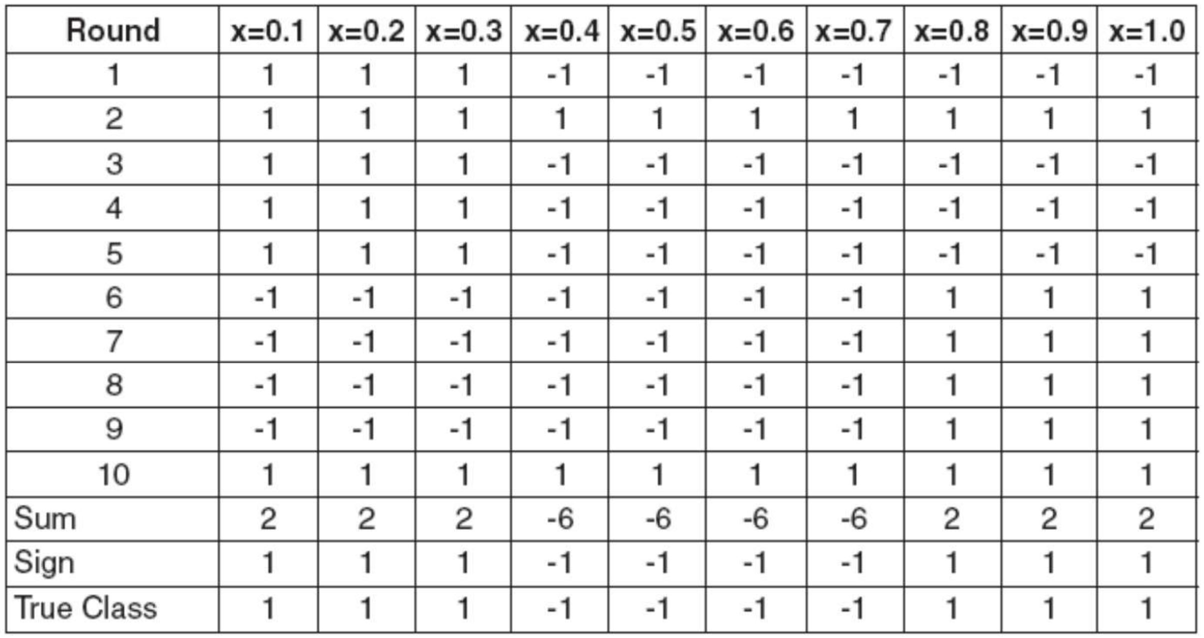
x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ==> y = -1
x > 0.05 ==> y = 1

Figure 5.35. Example of bagging.

Bagging

→ better 10-20% performance than the normal tree decision classifier



Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1
True Class	1	1	1	-1	-1	-1	-1	1	1	1

Figure 5.36. Example of combining classifiers constructed using the bagging approach.

Accuracy of ensemble classifier: 100% ☺

Bagging- Final Points

- Works well if the base classifiers are unstable
- Increased accuracy because it reduces the variance of the individual classifier
- Does not focus on any particular instance of the training data
- Therefore, less susceptible to model over-fitting when applied to noisy data
- What if we want to focus on a particular instances of training data?

Boosting

- Analogy: Consult several doctors, based on a combination of weighted diagnoses—weight assigned based on the previous diagnosis accuracy
- Boosting algorithm can be extended for numeric prediction
- Comparing with bagging: Boosting tends to have greater accuracy, but it also risks overfitting the model to misclassified data

Boosting

- Equal weights are assigned to each training tuple (1/d for round 1)
- After a classifier M_i is learned, the weights are adjusted to allow the subsequent classifier M_{i+1} to "pay more attention" to tuples that were misclassified by M_i.
- Final boosted classifier M* combines the votes of each individual classifier
- Weight of each classifier's vote is a function of its accuracy
- Adaboost – popular boosting algorithm

Boosting

- Records that are wrongly classified will have their weights increased
- Records that are classified correctly will have their weights decreased

Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

- Example 4 is hard to classify
- Its weight is increased, therefore it is more likely to be chosen again in subsequent rounds

Adaboost

- Given a set of *d* class-labeled tuples, ($\mathbf{X_1}$, y_1), …, ($\mathbf{X_d}$, y_d)
- Initially, all the weights of tuples are set the same (1/d)
- Generate k classifiers in k rounds. At round i,
  - Tuples from D are sampled (with replacement) to form a training set D_i of the same size
  - Each tuple's chance of being selected is based on its weight
  - A classification model M_i is derived from D_i
  - Its error rate is calculated using D_i as a test set
  - If a tuple is misclassified, its weight is increased, o.w. it is decreased
- Error rate: err($\mathbf{X_j}$) is the misclassification error of tuple $\mathbf{X_j}$. Classifier M_i error rate is the sum of the weights of the misclassified tuples:

$$error(M_i) = \sum_{j}^{d} w_j \times err(\mathbf{X_j})$$

- The weight of classifier M_i's vote is $\log \frac{1 - error(M_i)}{error(M_i)}$

AdaBoost

- Adaptive Boosting is an approach that constructs an ensemble of simple “weak” classifiers.
- Each classifier is trained on a single feature.
  - Often single split decision trees.
- The task of the classification training is to identify an ensemble of classifiers and their weights for combination

AdaBoost Classification

weight

$$C(x) = \alpha_1 C_1(x) + \alpha_2 C_2(x) + \ldots + \alpha_k C_k(x)$$

- AdaBoost generates a prediction from a weighted sum of predictions of each classifier.
- The AdaBoost algorithm determines the weights.
- Similar to systems that use a second tier classifier to learn a combination function.

AdaBoost Algorithm

- Repeat
  - Identify the best unused classifier Ci.
  - Assign it a weight based on its performance
  - Update the weights of each **data point** based on whether or not it is classified correctly
- Until performance converges or all classifiers are included.

Identify the best classifier

- Evaluate the performance of each unused classifier.
- Calculate weighted accuracy using the current data point weights.

$$W_e = \sum_{y_i \neq k_m(x_i)} w_i^{(m)}$$

Assign Weight for the Current Classifier

$$\alpha_m = \frac{1}{2}\ln\left(\frac{1-e_m}{e_m}\right)$$

$$e_m = \frac{W_m}{W}$$

- The larger the reduction in error, the larger the classifier weight

Update Data Point Weights for Next Iteration

- If i is a miss:

$$w_i^{(m+1)} = w_i^{(m)} e^{-\alpha_m} = w_i^{(m)} \sqrt{\frac{e_m}{1-e_m}}$$

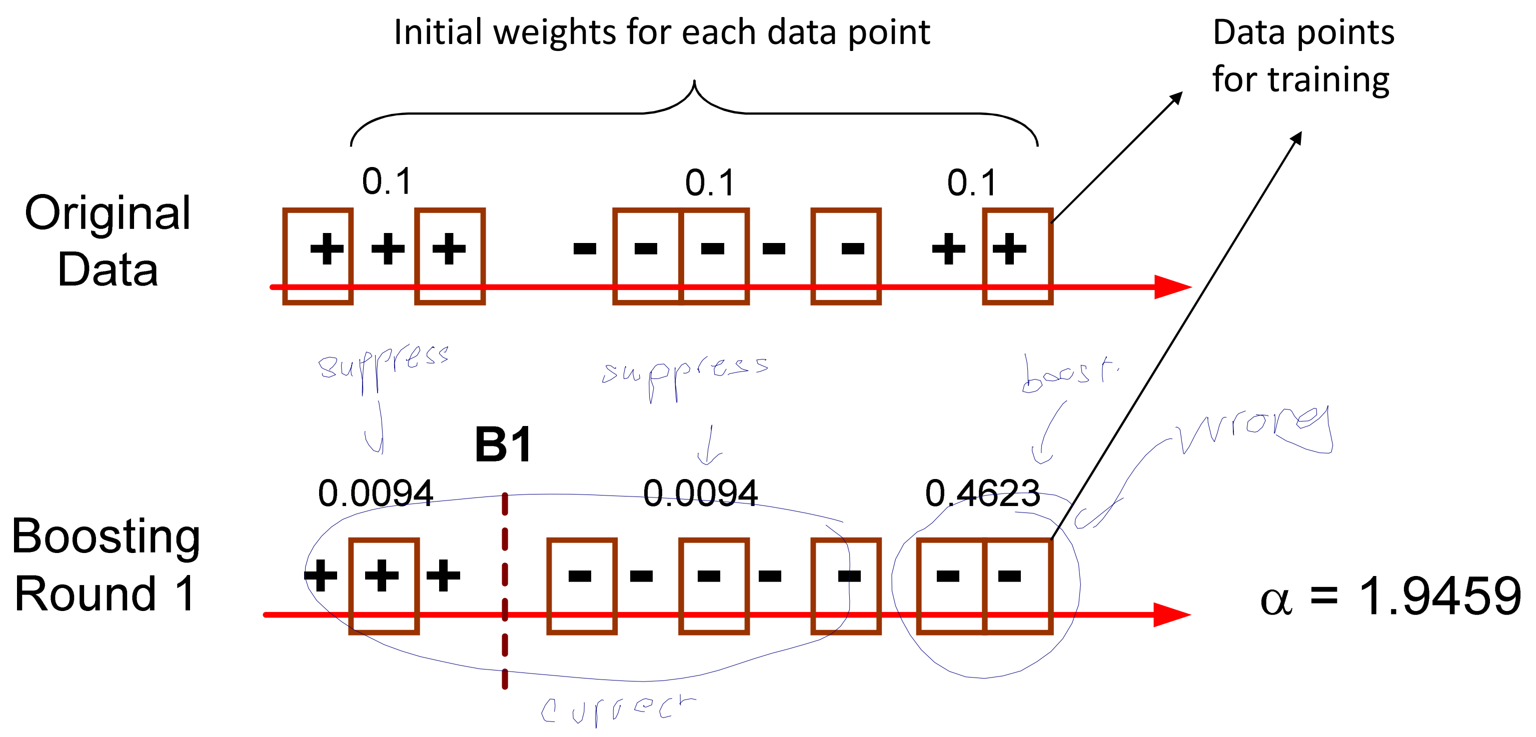
- If i is a hit:

$$w_i^{(m+1)} = w_i^{(m)} e^{\alpha_m} = w_i^{(m)} \sqrt{\frac{1-e_m}{e_m}}$$

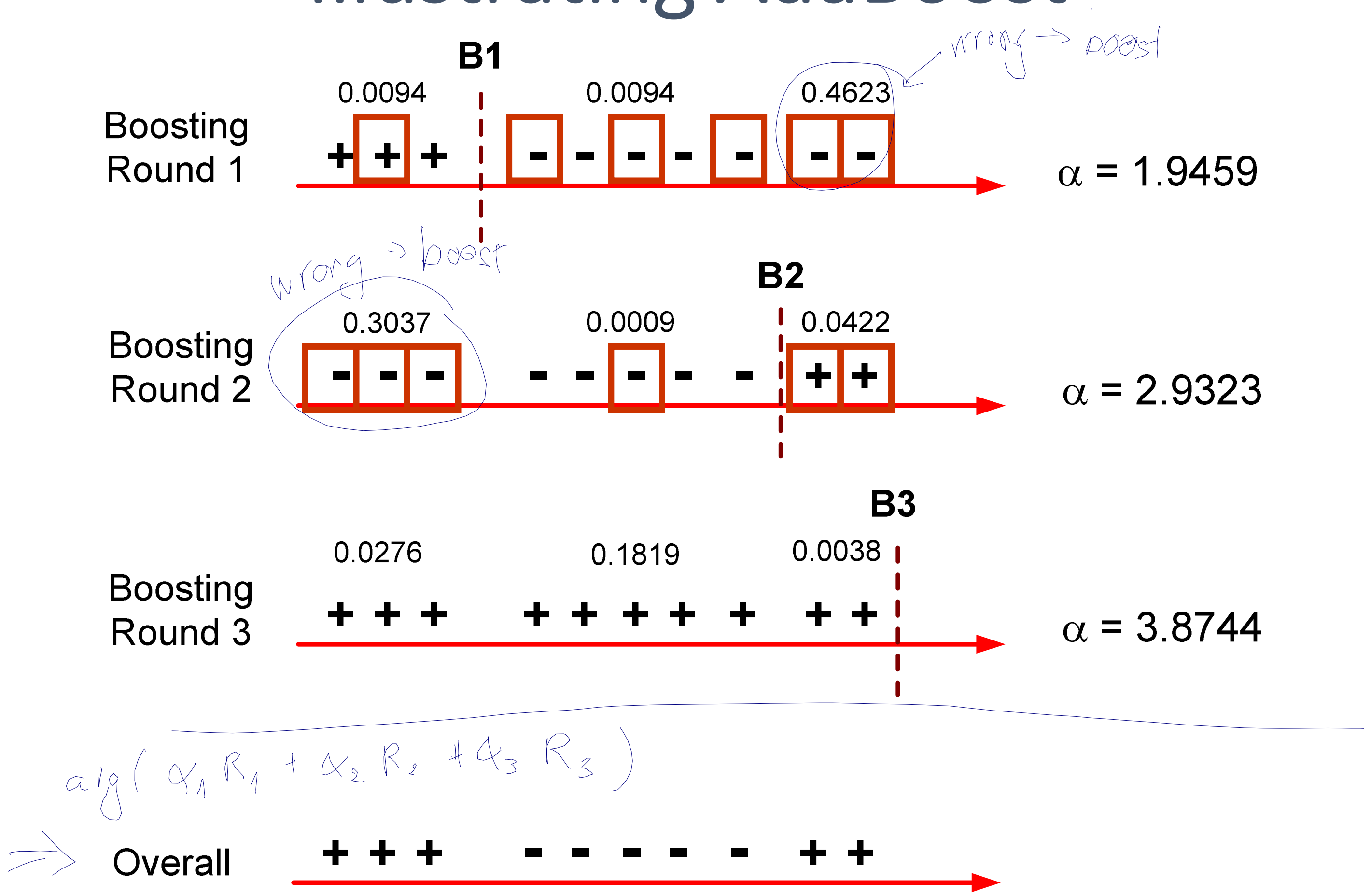
AdaBoost Algorithm

- Repeat
  - Identify the best unused classifier Ci.
  - Assign it a weight based on its performance
  - Update the weights of each **data point** based on whether or not it is classified correctly
- Until performance converges or all classifiers are included.

Illustrating AdaBoost



Illustrating AdaBoost



3rd.

Random Forests

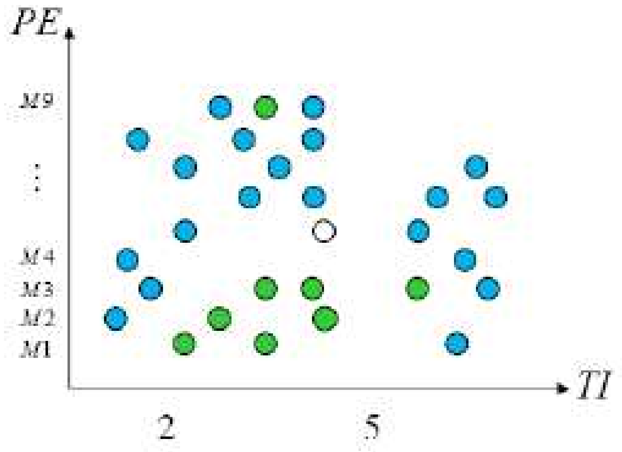
- Random Forests are similar to AdaBoost decision trees.
- **Random Forests** is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the class's output by individual trees.
- An ensemble of classifiers is trained each on a different random subset of features.
  - Random subspace projection
- The method combines Breiman's "bagging" idea and the random selection of features.

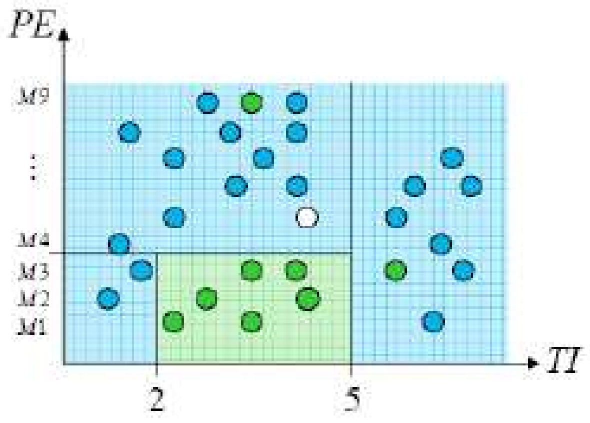
Decision Trees

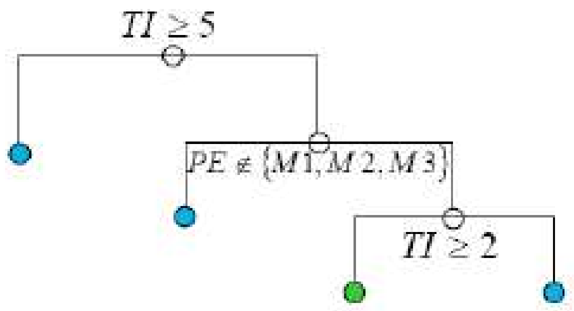
- Based on if-then-else rules normally with thresholds
- During training, information gain estimate is used to select feature at each node to split the branch
- Decision trees are individual learners that are combined. They are one of the most popular learning methods commonly used for data exploration.

Decision Trees

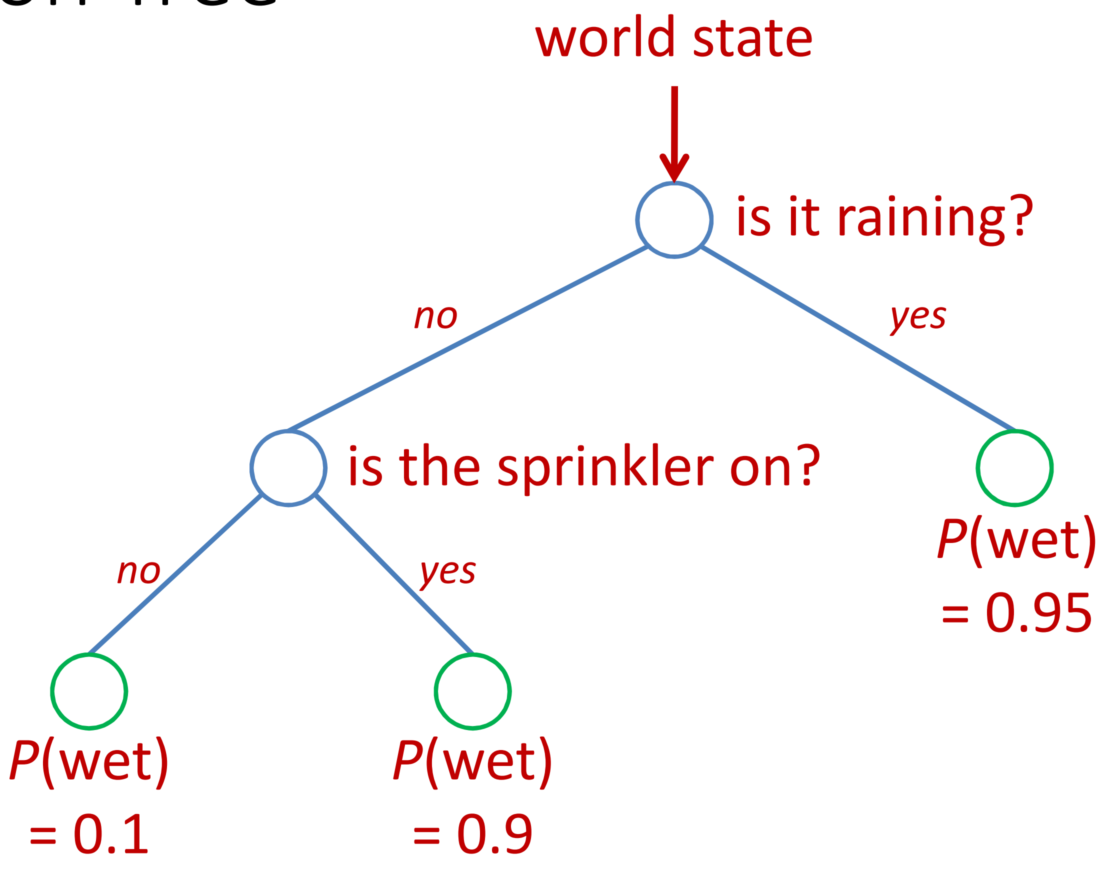
TI	PE	Response
1.0	M2	good
2.0	M1	bad
...	...	...
4.5	M5	?



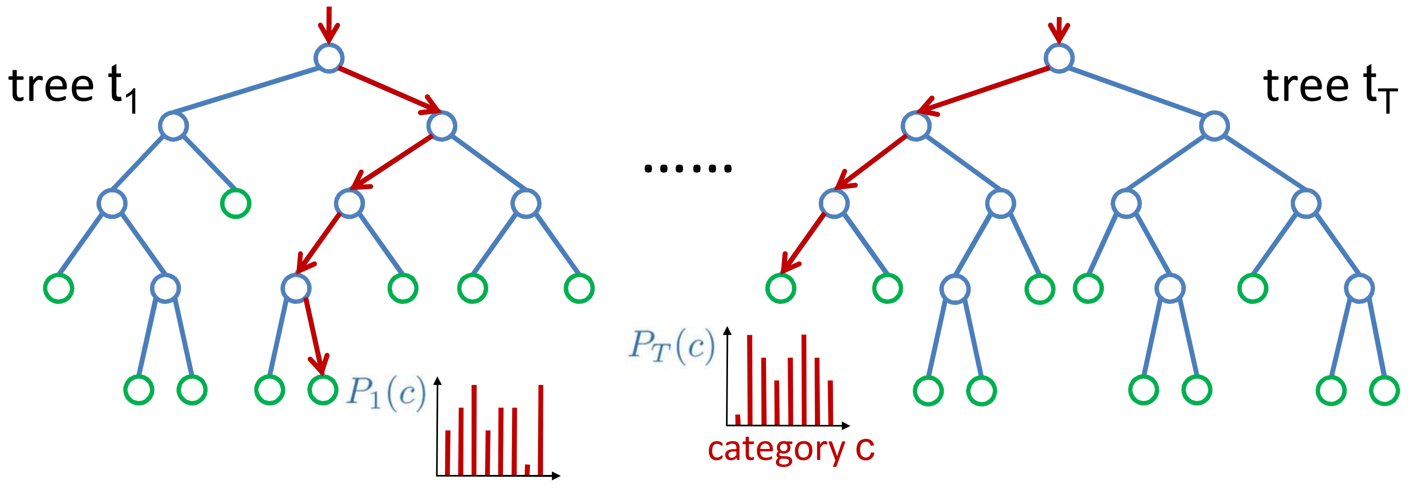




Decision Tree



Construct a forest of trees



$$P(c|\mathbf{v}) = \frac{1}{T}\sum_{t=1}^{T} P_t(c|\mathbf{v})$$

Random Forest

- Random Forest:
  - Each classifier in the ensemble is a *decision tree* classifier and is generated using a random selection of attributes at each node to determine the split
  - During classification, each tree votes and the most popular class is returned
- Two Methods to construct Random Forest:
  - Forest-RI (*random input selection*): Randomly select, at each node, F attributes as candidates for the split at the node. The CART methodology is used to grow the trees to maximum size
  - Forest-RC (*random linear combinations*): Creates new attributes (or features) that are a linear combination of the existing attributes (reduces the correlation between individual classifiers)
- Comparable in accuracy to Adaboost, but more robust to errors and outliers
- Insensitive to the number of attributes selected for consideration at each split, and faster than bagging or boosting

Features and Advantages

- It is one of the most accurate learning algorithms available. For many data sets, it produces a highly accurate classifier.
- It runs efficiently on large databases.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.
- It generates an internal unbiased estimate of the generalization error as the forest building progresses.
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.

Disadvantages

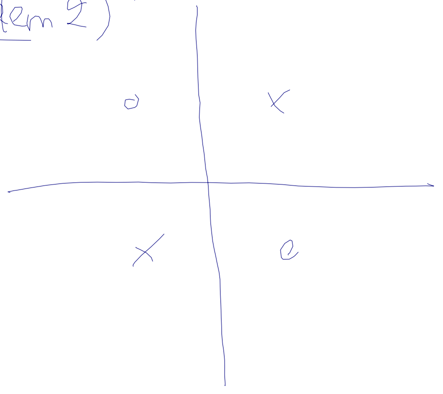
- Random forests have been observed to overfit for some datasets with noisy classification/regression tasks.
- For data including categorical variables with different number of levels, random forests are biased in favor of those attributes with more levels. Therefore, the variable importance scores from random forest are not reliable for this type of data.

Examples

- plot_classifier_comparison.py
  - Compare various classifiers on artificial data set
- adaboost_demo.py
  - Illustrate the step by step computation of adaboost algorithm
- plot_forest_iris.py
  - Illustrate and compare various decision tree and random forest on the iris data set

Homework:

Problem 2) XOR



Prob3) can do manually or using python.

Prob4)

bagging
Single level decision tree → 1 threshold
→ can use sk-learn

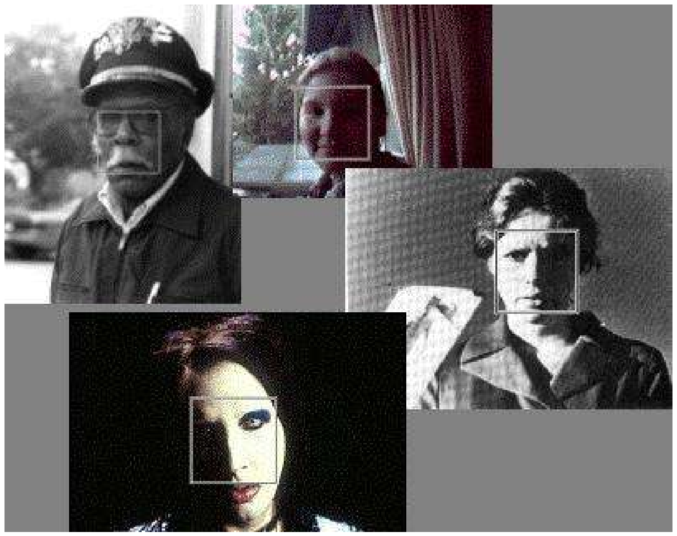
4a) table result of training data.

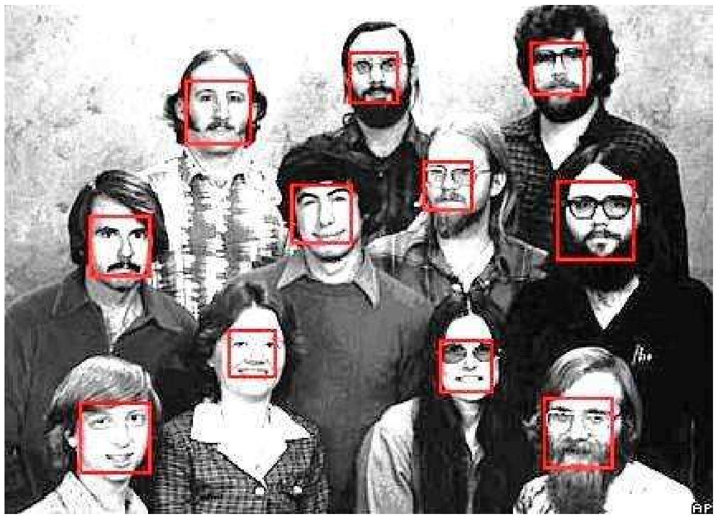
Prob5) Needs sk-learn random forest python.

Adaboost for Face Detection

Slides adapted from P. Viola and Tai-Wan Yue

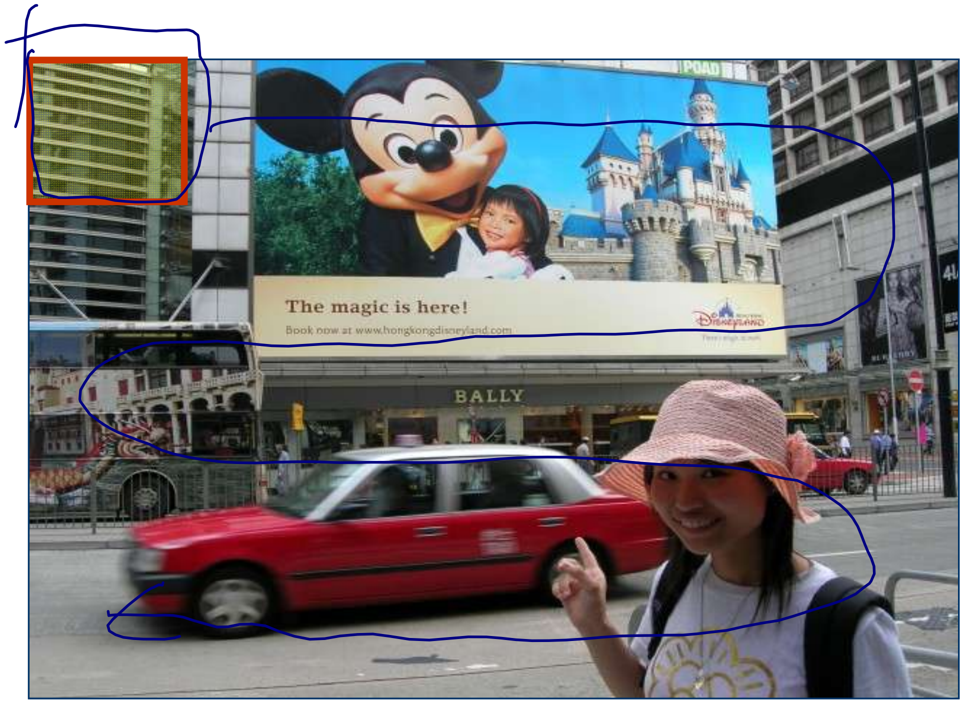
The Task of Face Detection





Basic Idea

Slide a window across image and evaluate a face model at every location; no. of locations where face is present is very very small



Challenges

- Slide a window across image and evaluate a face model at every location
- Sliding window detector must evaluate tens of thousands of location/scale combinations
- Faces are rare: 0–10 per image
  - For computational efficiency, we should try to spend as little time as possible on the non-face windows
  - A megapixel image has **~10^6** pixels and a comparable number of candidate face locations
  - To avoid having a false positive in every image, the false positive rate has to be less than **10^{-6}**

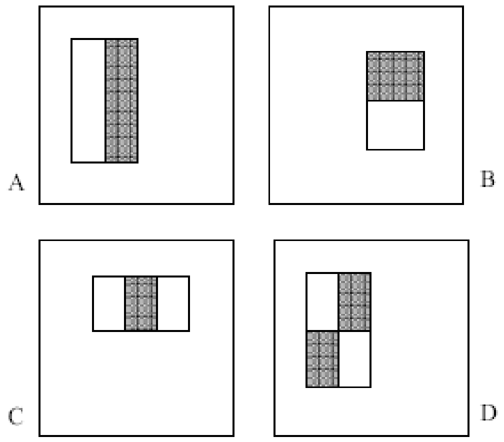
The Viola/Jones Face Detector

- A seminal approach to real-time object detection
- Key ideas
  - *Integral images* for fast feature evaluation
  - *Boosting* for feature selection
  - *Attentional cascade* for fast rejection of non-face windows

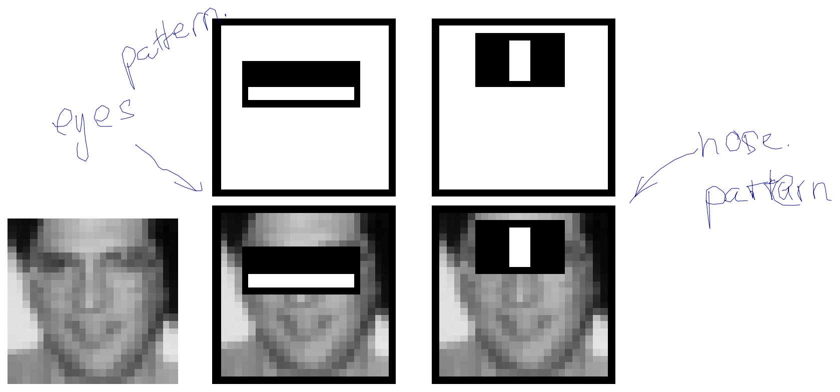
P. Viola and M. Jones. *Rapid object detection using a boosted cascade of simple features.* CVPR 2001.

P. Viola and M. Jones. *Robust real-time face detection.* IJCV 57(2), 2004.

Image Features



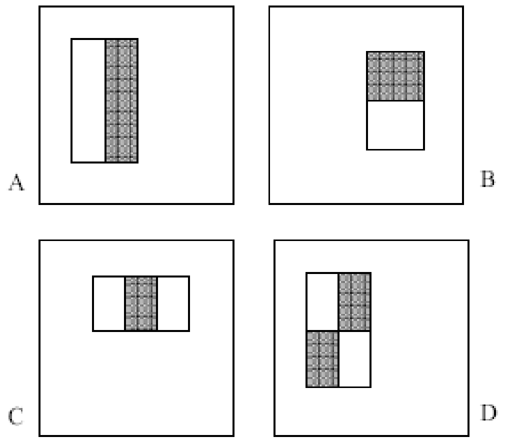
Rectangular filters



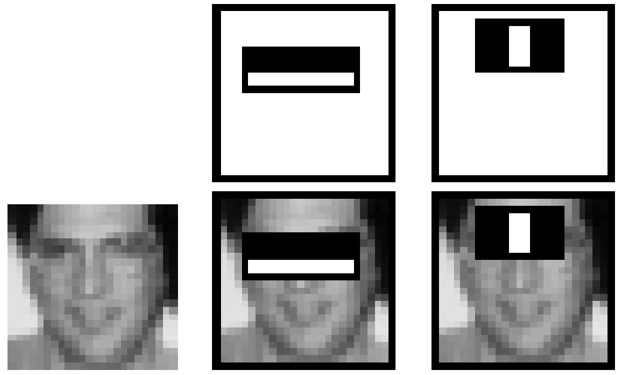
$$f(x,y) = \sum_i p_b(i) - \sum_i p_w(i)$$

Local features: Subtract sum of pixels in white area from the sum of pixels in black area; 2-rectangle features (A and B), 3-rectangle feature (C) and 4-rectangle feature (D)

Image Features



Rectangular filters



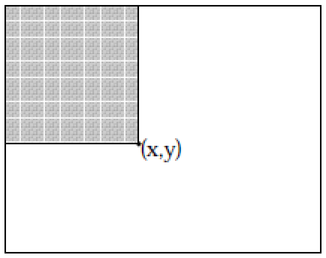
$f(x, y) = \sum_i p_b(i) - \sum_i p_w(i)$

Local features: S… …rea from the sum of pixels in black a… …d B), 3-rectangle feature (C) and 4-… …D)

In a 24 x 24 patch with 4x4 detector, there are over 160,000 locations for rectangles

Image Features

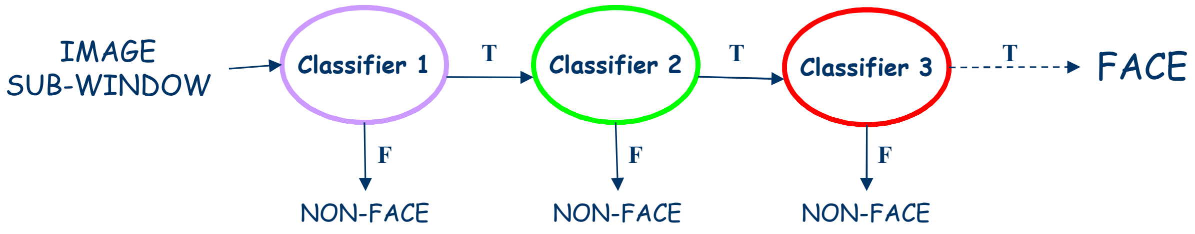
- Integral Image: An intermediate representation of the image for rapid calculation of rectangle features
- s(x,y) is the cumulative row sum, s(x,-1) =0 and ii(-1, y) = 0; the integral image can be computed in one pass over the original image



$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x',y')$$

$$\begin{aligned} s(x,y) &= s(x,y-1) + i(x,y) \\ ii(x,y) &= ii(x-1,y) + s(x,y) \end{aligned}$$

- Reject non-face windows through a cascade of classifiers and boosting



Rectangle Features from Integral Image

- Rectangle features are somewhat primitive and coarse, they are sensitive to presence of edges, bars, and other simple structure
- Generating a large no. of these features and their computational efficiency compensates for this.

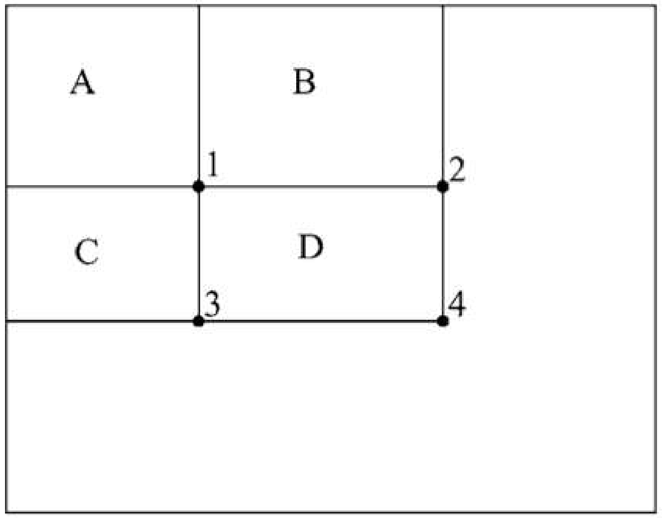


Figure 3. The sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is $A + B$, at location 3 is $A + C$, and at location 4 is $A + B + C + D$. The sum within D can be computed as $4 + 1 - (2 + 3)$.

Face Detector

- Scan the input at many scales
- Starting at the base scale in which faces are detected at 24x24 pixels, a 384x288 pixel image is scanned at 12 scales each a factor 1.25 larger than the last
- Any rectangle feature can be evaluated at any scale and location in a few operations
- Face detection @15 fps for the entire image

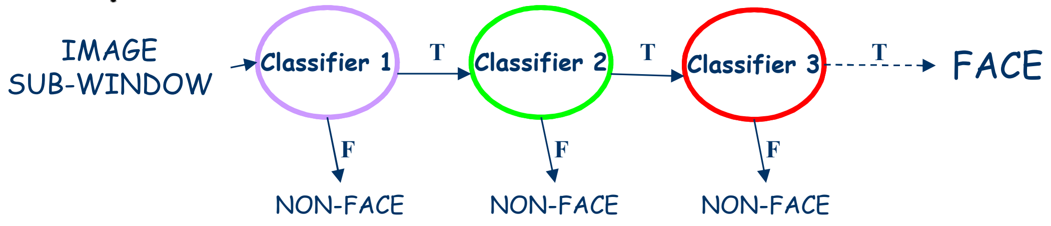
Weak Learners for Face Detection

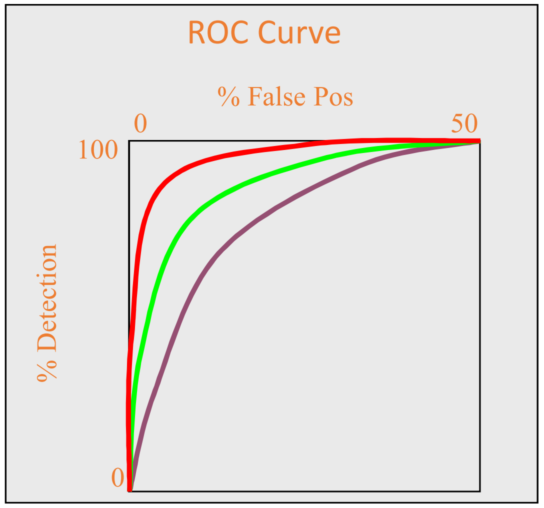
- Hypothesis: A very small no. of 160K (x4) features for each image sub-window can be formed to find an effective classifier (face vs. non-face)
- AdaBoost is used both to select the features and train the classifier
- Weak learner: a single rectangle feature that best separates positive and negative examples; so weak classifier is a thresholded single feature (can be viewed as a single node decision tree)

$$h_t(\underset{\text{window}}{x}) = \begin{cases} 1 & \text{if } p_t \overset{\text{value of rectangle feature}}{f_t(x)} > \overset{\text{parity}}{p_t}\overset{\text{threshold}}{\theta_t} \\ 0 & \text{otherwise} \end{cases}$$

Cascade of classifiers

- Train each classifier with increasing number of features until the target false positive and detection rates are achieved
- Use false positives from current stage as the negative training examples for the next stage
- Classifiers are progressively more complex and have lower false positive rates





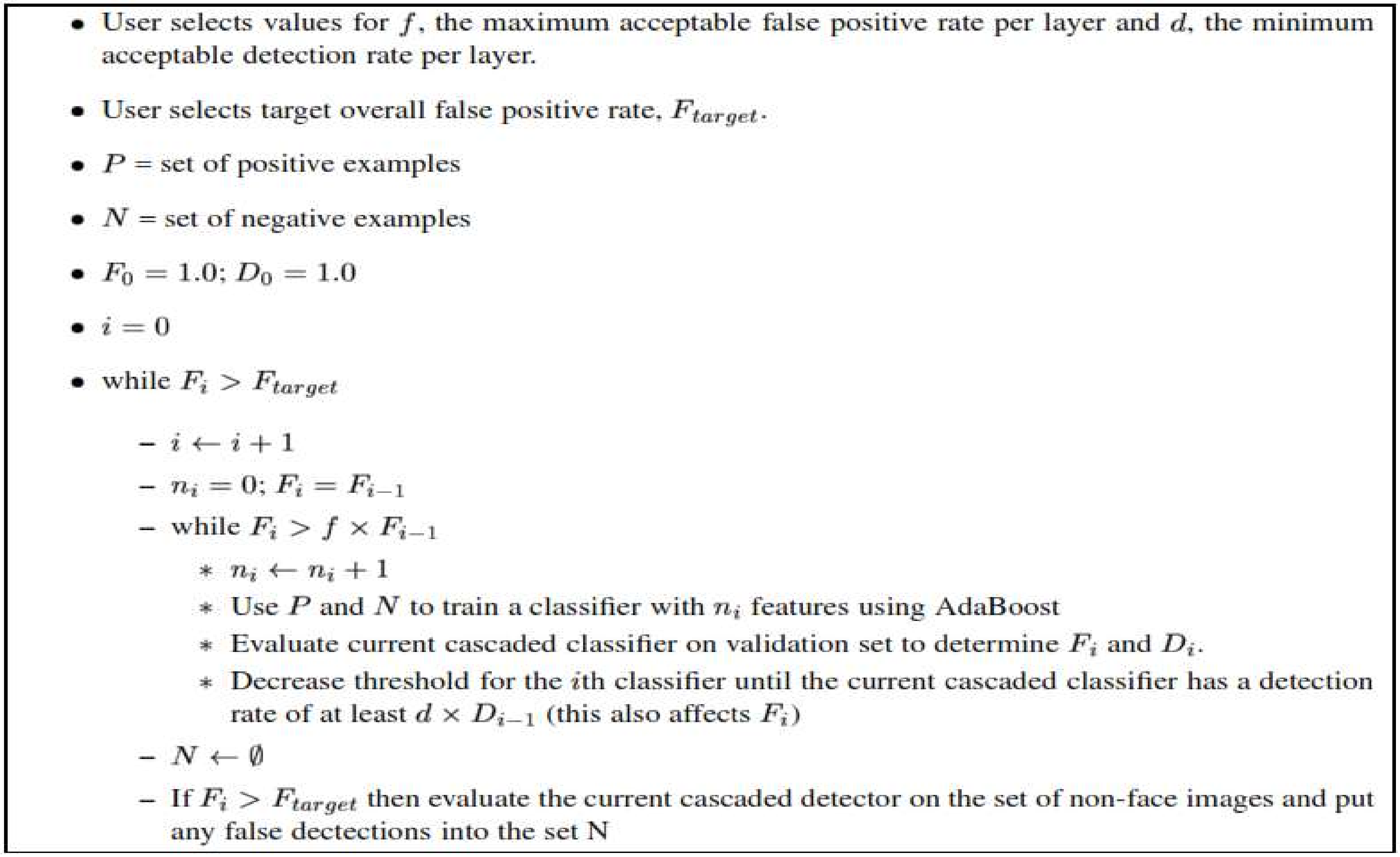
False positive rate of cascade $F = \prod_{i=1}^{K} f_i$

Detection rate of cascade $D = \prod_{i=1}^{K} d_i$

Boosting

- Training set contains face and nonface examples
  - Initially, with equal weight
- For each round of boosting:
  - Evaluate each rectangle filter on each example
  - Select best threshold for each filter
  - Select best filter/threshold combination
  - Reweight examples
- Computational complexity of learning: $O(MNK)$
  - M rounds, N examples, K features

Face Detection using Adaboost

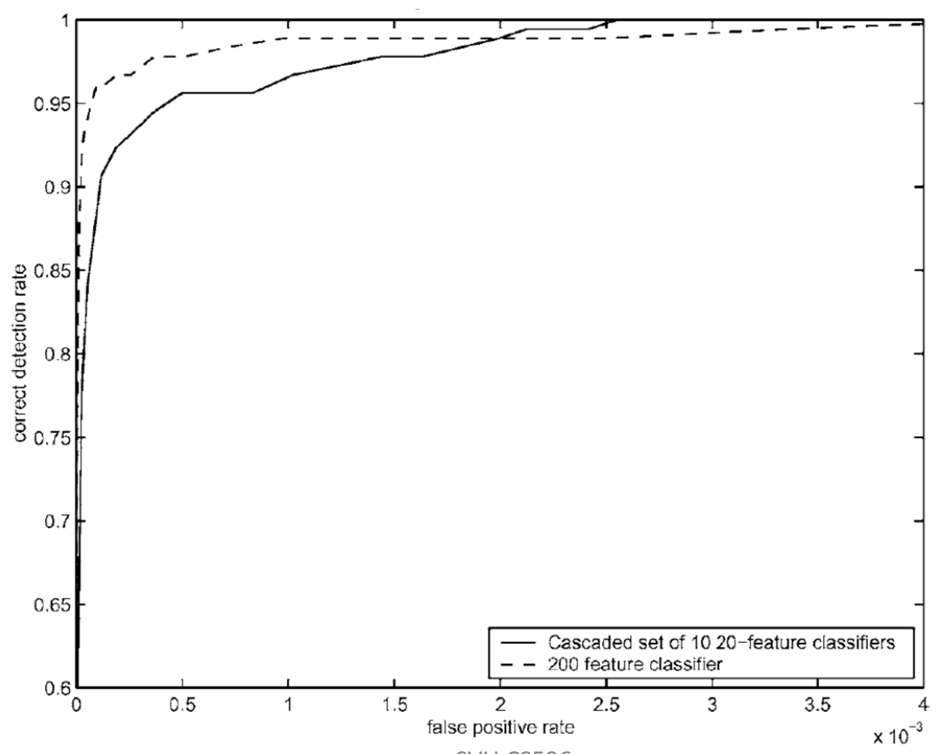


```
• User selects values for $f$, the maximum acceptable false positive rate per layer and $d$, the minimum
  acceptable detection rate per layer.
• User selects target overall false positive rate, $F_{target}$.
• $P$ = set of positive examples
• $N$ = set of negative examples
• $F_0 = 1.0$; $D_0 = 1.0$
• $i = 0$
• while $F_i > F_{target}$
    – $i \leftarrow i + 1$
    – $n_i = 0$; $F_i = F_{i-1}$
    – while $F_i > f \times F_{i-1}$
        * $n_i \leftarrow n_i + 1$
        * Use $P$ and $N$ to train a classifier with $n_i$ features using AdaBoost
        * Evaluate current cascaded classifier on validation set to determine $F_i$ and $D_i$.
        * Decrease threshold for the $i$th classifier until the current cascaded classifier has a detection
          rate of at least $d \times D_{i-1}$ (this also affects $F_i$)
    – $N \leftarrow \emptyset$
    – If $F_i > F_{target}$ then evaluate the current cascaded detector on the set of non-face images and put
      any false dectections into the set N
```

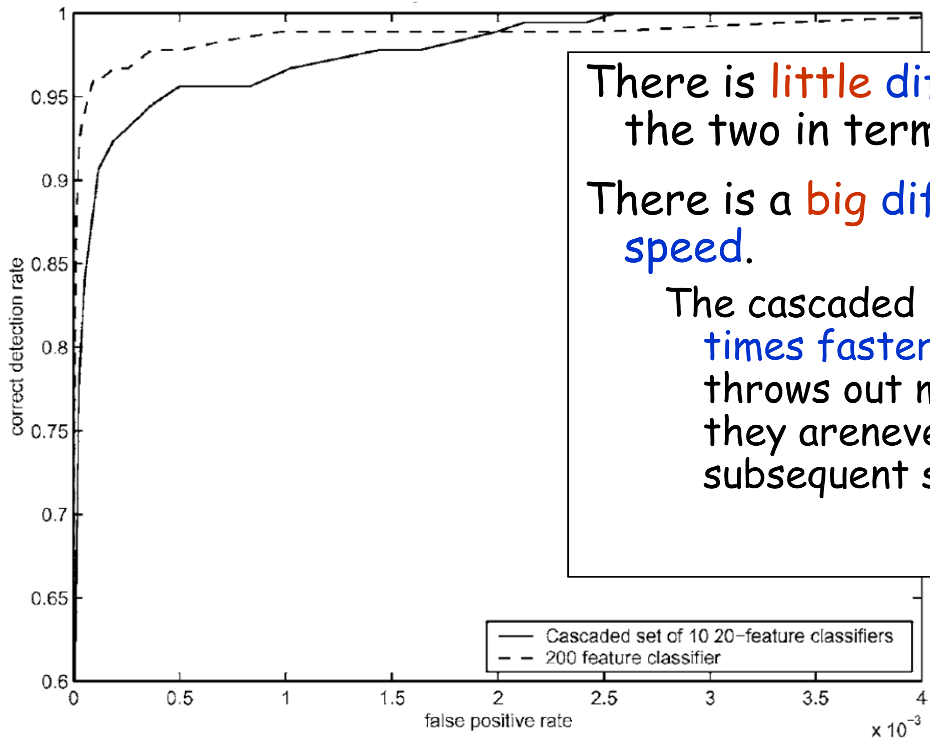
Table 2: The training algorithm for building a cascaded detector.

ROC Curves Cascaded Classifier to Monolithic Classifier

Speed of cascade classifier is 10 times faster



ROC Curves Cascaded Classifier to Monolithic Classifier



- There is little difference between the two in terms of accuracy.
- There is a big difference in terms of speed.
  - The cascaded classifier is nearly 10 times faster since its first stage throws out most non-faces so that they arenever evaluated by subsequent stages.

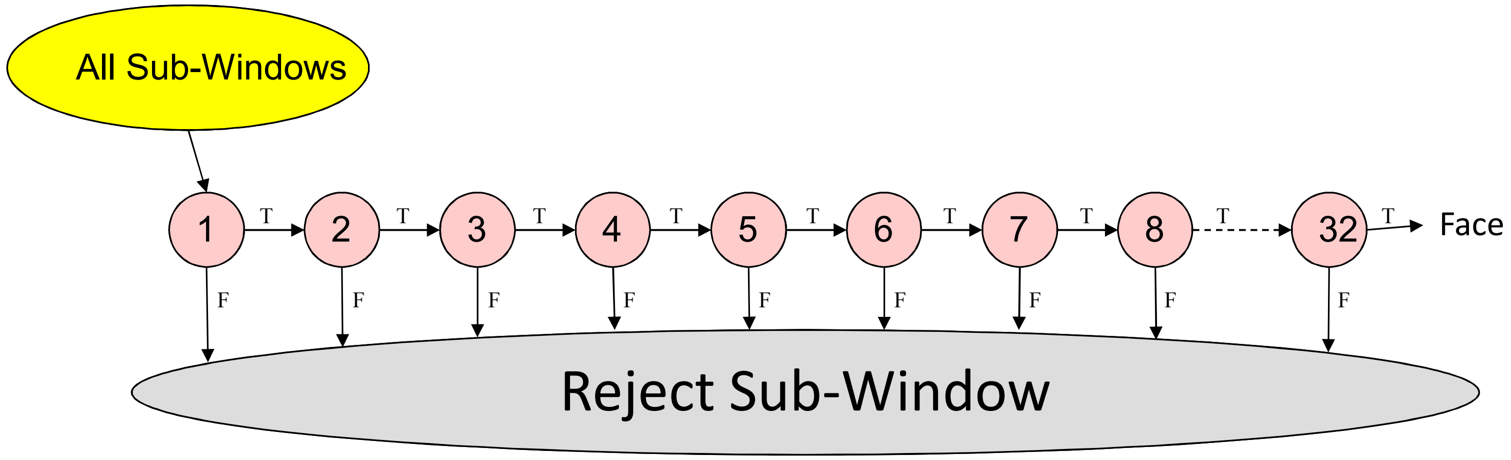
Face Detection System

- Training Data
  - 5000 faces
    - All frontal, rescaled to 24x24 pixels
  - 9500 million non-faces
  - Faces are normalized
    - Scale, translation
- Many variations
  - Across individuals
  - Illumination
  - Pose

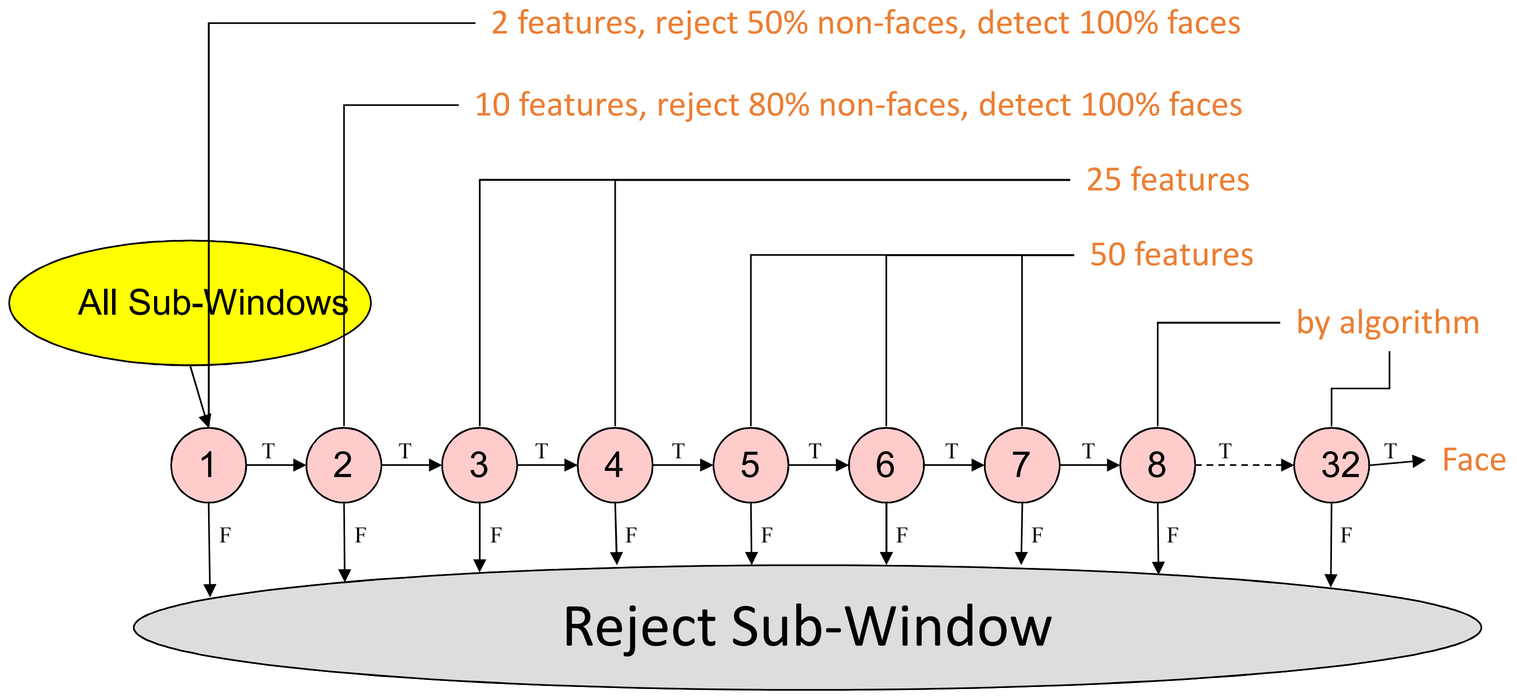
Structure of the Detector Cascade

Combining successively more complex classifiers in cascade

- 32 stages
- included a total of 4297 features



Structure of the Detector Cascade


2 features, reject 50% non-faces, detect 100% faces
10 features, reject 80% non-faces, detect 100% faces
25 features
50 features
by algorithm
All Sub-Windows
1
2
3
4
5
6
7
8
32
T
F
Face
Reject Sub-Window

Feature Selection

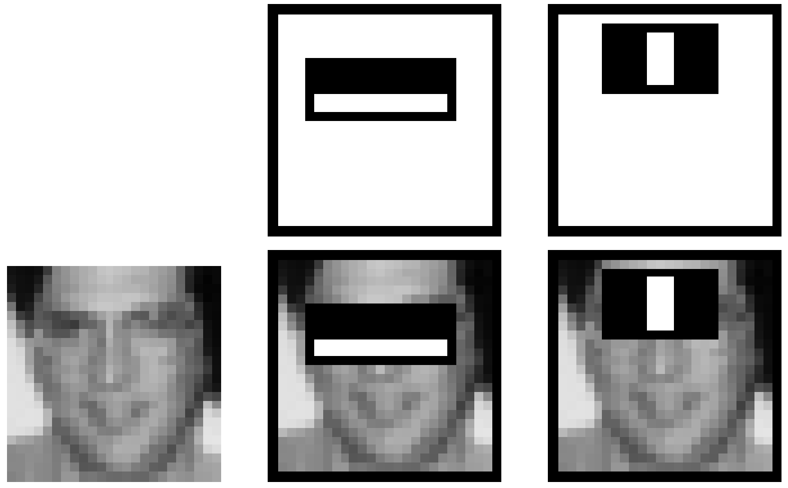


Figure 5: The first and second features selected by AdaBoost. The two features are shown in the top row and then overlayed on a typical training face in the bottom row. The first feature measures the difference in intensity between the region of the eyes and a region across the upper cheeks. The feature capitalizes on the observation that the eye region is often darker than the cheeks. The second feature compares the intensities in the eye regions to the intensity across the bridge of the nose.

Speed of the Final Detector

- On a 700 Mhz Pentium III processor, the face detector can process a 384×288 pixel image in about .067 seconds"
  - 15 times faster than previous detector of comparable accuracy (Rowley et al., 1998)
- Average of 8 features evaluated per window on test set

Output of Face Detector on Test Images