AdaBoost
Lecture 8

Jason Corso

SUNY at Buffalo

March 30 2010

Introduction

- In the last lecture, we looked at
  1. Lack of inherent superiority of any one particular classifier;
  2. Some systematic ways for selecting a particular method over another for a given scenario;
- And, we talked briefly about the bagging method for integrating component classifiers.
- Now, we turn to boosting and the AdaBoost method for integrating component classifiers into one strong classifier.

Rationale

- Imagine the situation where you want to build an email filter that can distinguish spam from non-spam.

Rationale

- Imagine the situation where you want to build an email filter that can distinguish spam from non-spam.
- The general way we would approach this problem in ML/PR follows the same scheme we have for the other topics:

Rationale

- Imagine the situation where you want to build an email filter that can distinguish spam from non-spam.
- The general way we would approach this problem in ML/PR follows the same scheme we have for the other topics:
  1. Gathering as many examples as possible of both spam and non-spam emails.
  2. Train a classifier using these examples and their labels.
  3. Take the learned classifier, or prediction rule, and use it to filter your mail.
  4. **The goal is to train a classifier that makes the most accurate predictions possible on new test examples.**
     And, we've covered related topics on how to measure this like bias and variance.

Rationale

- Imagine the situation where you want to build an email filter that can distinguish spam from non-spam.
- The general way we would approach this problem in ML/PR follows the same scheme we have for the other topics:
  1. Gathering as many examples as possible of both spam and non-spam emails.
  2. Train a classifier using these examples and their labels.
  3. Take the learned classifier, or prediction rule, and use it to filter your mail.
  4. **The goal is to train a classifier that makes the most accurate predictions possible on new test examples.**
     And, we've covered related topics on how to measure this like bias and variance.
- But, building a highly accurate classifier is a difficult task. (You still get spam, right?!)

- We could probably come up with many quick **rules of thumb**. These could be only moderately accurate. Can you think of an example for this situation?

- We could probably come up with many quick **rules of thumb**. These could be only moderately accurate. Can you think of an example for this situation?
- An example could be "if the subject line contains 'buy now' then classify as spam."

- We could probably come up with many quick **rules of thumb**. These could be only moderately accurate. Can you think of an example for this situation?
- An example could be "if the subject line contains 'buy now' then classify as spam."
- This certainly doesn't cover all spams, but **it will be significantly better than random guessing.**

Basic Idea of Boosting

- **Boosting** refers to a general and provably effective method of producing a very accurate classifier by combining rough and moderately inaccurate rules of thumb.

Basic Idea of Boosting

- **Boosting** refers to a general and provably effective method of producing a very accurate classifier by combining rough and moderately inaccurate rules of thumb.
- It is based on the observation that **finding many rough rules of thumb can be a lot easier than finding a single, highly accurate classifier**.

Basic Idea of Boosting

- **Boosting** refers to a general and provably effective method of producing a very accurate classifier by combining rough and moderately inaccurate rules of thumb.
- It is based on the observation that **finding many rough rules of thumb can be a lot easier than finding a single, highly accurate classifier**.
- To begin, we define an algorithm for finding the rules of thumb, which we call a **weak learner**.

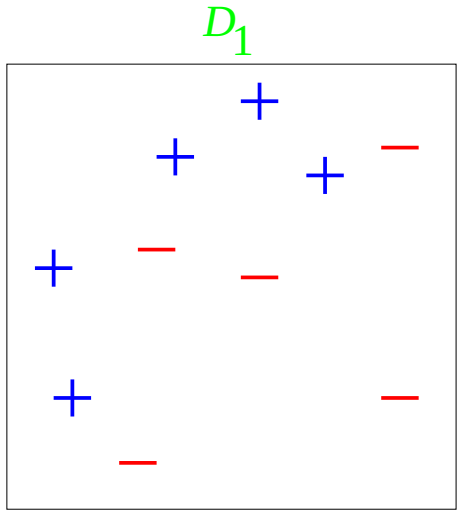
Basic Idea of Boosting

- **Boosting** refers to a general and provably effective method of producing a very accurate classifier by combining rough and moderately inaccurate rules of thumb.
- It is based on the observation that **finding many rough rules of thumb can be a lot easier than finding a single, highly accurate classifier**.
- To begin, we define an algorithm for finding the rules of thumb, which we call a **weak learner**.
- The boosting algorithm repeatedly calls this weak learner, each time feeding it a different distribution over the training data (in Adaboost).

Basic Idea of Boosting

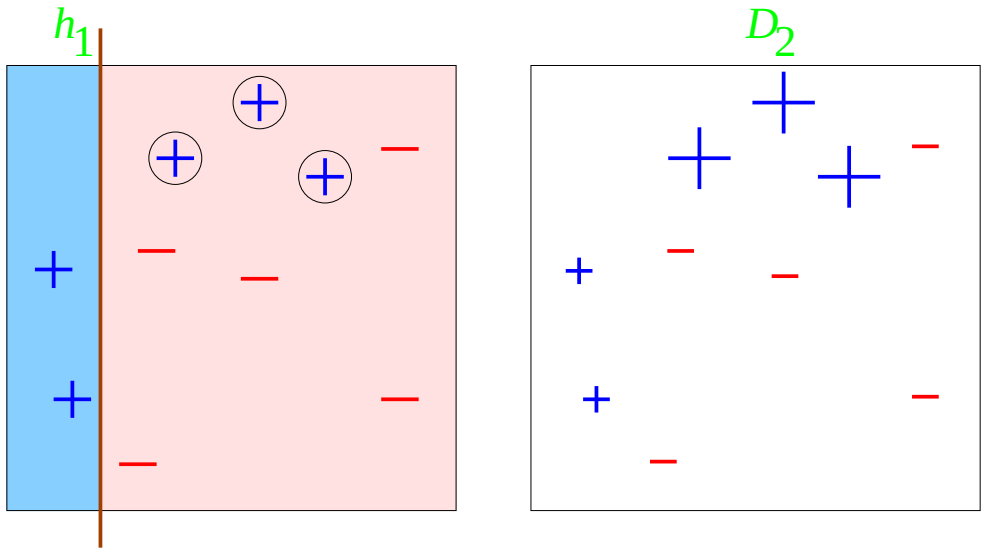
- **Boosting** refers to a general and provably effective method of producing a very accurate classifier by combining rough and moderately inaccurate rules of thumb.
- It is based on the observation that **finding many rough rules of thumb can be a lot easier than finding a single, highly accurate classifier**.
- To begin, we define an algorithm for finding the rules of thumb, which we call a **weak learner**.
- The boosting algorithm repeatedly calls this weak learner, each time feeding it a different distribution over the training data (in Adaboost).
- Each call generates a **weak classifier** and we must combine all of these into a single classifier that, hopefully, is much more accurate than any one of the rules.

Toy Example



weak classifiers = vertical or horizontal half-planes

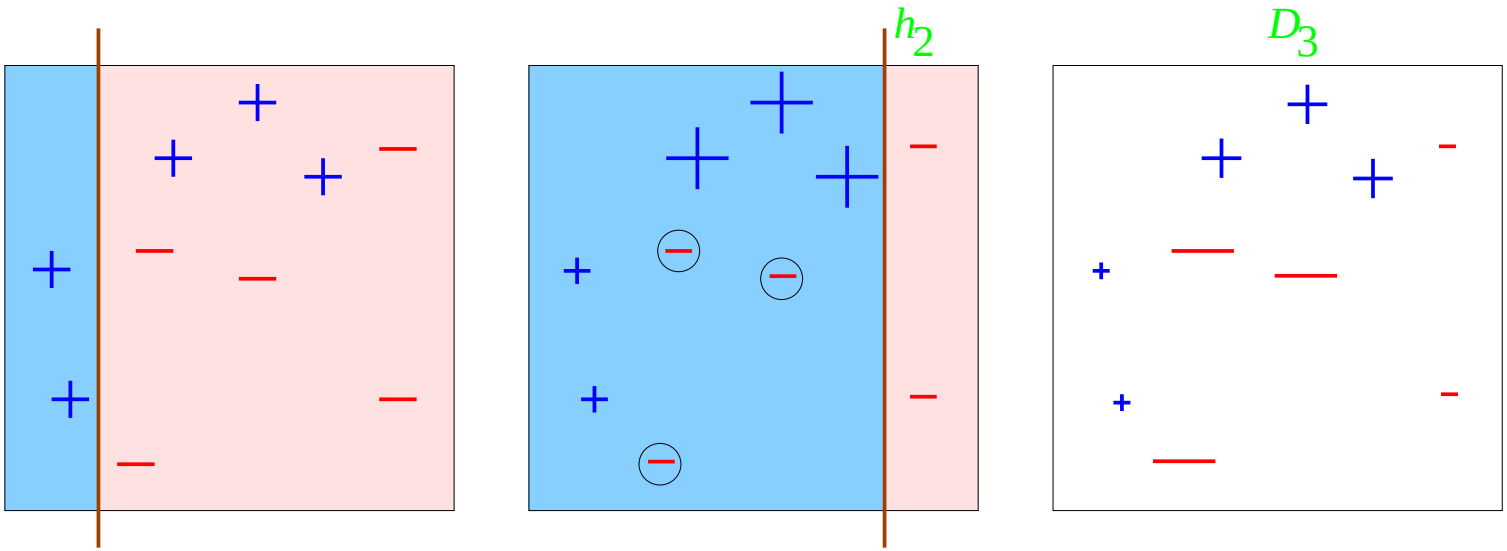
Round 1



$\varepsilon_1 = 0.30$

$\alpha_1 = 0.42$

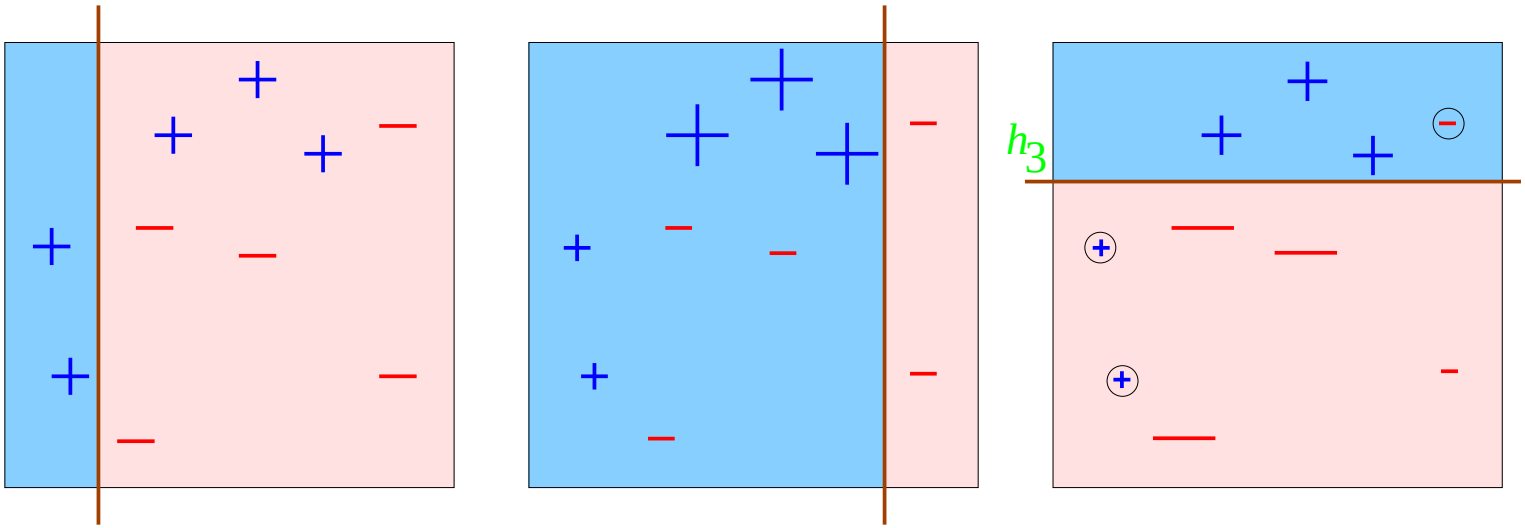
Round 2



$\varepsilon_2 = 0.21$

$\alpha_2 = 0.65$

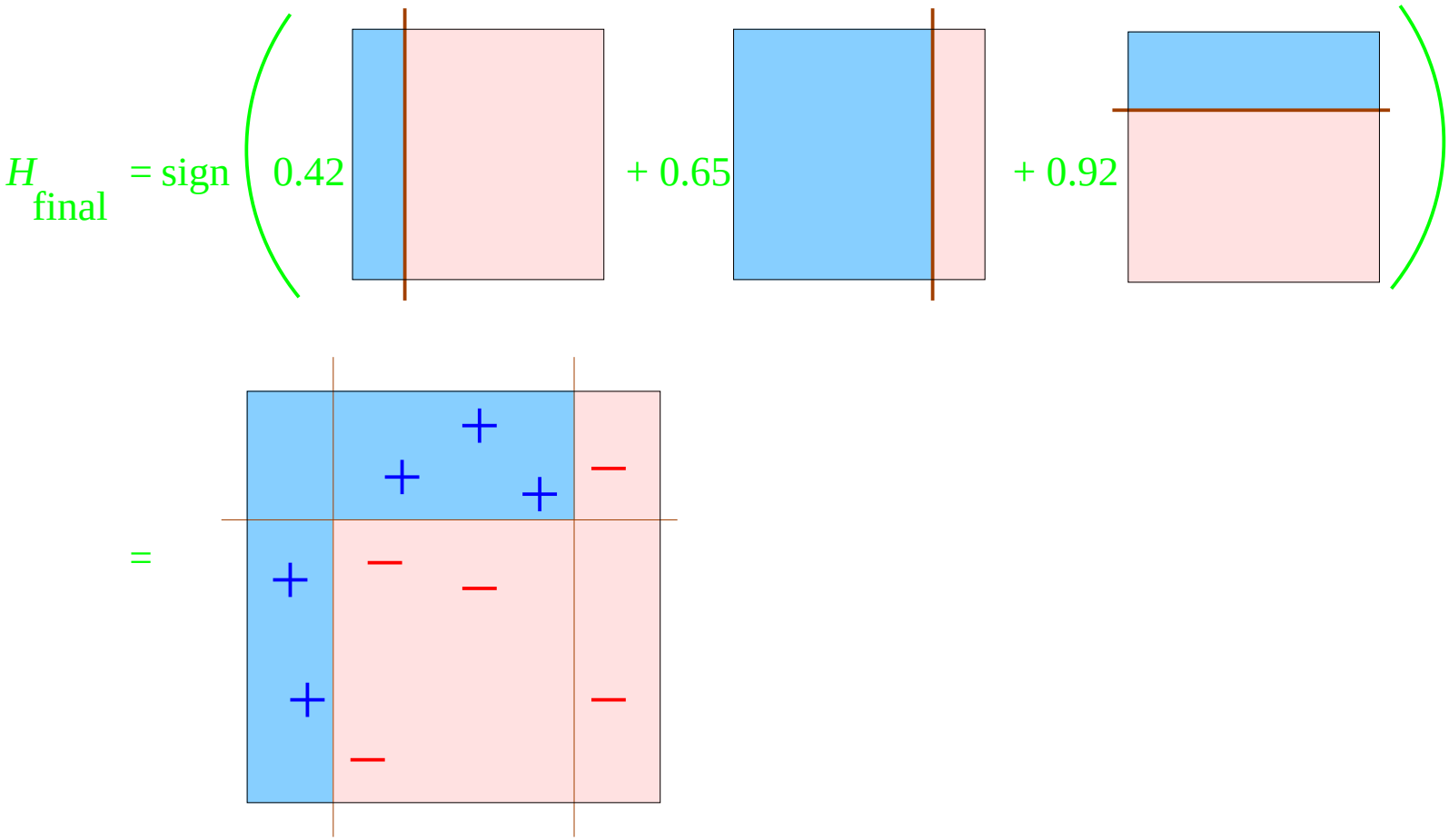
Round 3



$\varepsilon_3 = 0.14$

$\alpha_3 = 0.92$

Final Classifier



STOP!

Key Questions Defining and Analyzing Boosting

1. How should the distribution be chosen each round?

Key Questions Defining and Analyzing Boosting

1. How should the distribution be chosen each round?
2. How should the weak rules be combined into a single rule?

Key Questions Defining and Analyzing Boosting

1. How should the distribution be chosen each round?
2. How should the weak rules be combined into a single rule?
3. How should the weak learner be defined?

Key Questions Defining and Analyzing Boosting

1. How should the distribution be chosen each round?
2. How should the weak rules be combined into a single rule?
3. How should the weak learner be defined?
4. How many weak classifiers should we learn?

Getting Started

- We are given a training set

$$\mathcal{D} = \{(\mathbf{x}_i, y_i) : \mathbf{x}_i \in \mathbb{R}^d, y_i \in \{-1, +1\}, i = 1, \ldots, m\}. \quad (1)$$

Getting Started

- We are given a training set

$$\mathcal{D} = \{(\mathbf{x}_i, y_i) : \mathbf{x}_i \in \mathbb{R}^d, y_i \in \{-1, +1\}, i = 1, \ldots, m\}. \quad (1)$$

- For example, $\mathbf{x}_i$ could represent some encoding of an email message (say in the vector-space text model), and y_i is whether or not this message is spam.

Getting Started

- We are given a training set

$$\mathcal{D} = \{(\mathbf{x}_i, y_i) : \mathbf{x}_i \in \mathbb{R}^d, y_i \in \{-1, +1\}, i = 1, \ldots, m\}. \quad (1)$$

- For example, $\mathbf{x}_i$ could represent some encoding of an email message (say in the vector-space text model), and y_i is whether or not this message is spam.
- Note that we are working in a two-class setting, and this will be the case for the majority of our discussion. Some extensions to multi-class scenarios will be presented.

Getting Started

- We are given a training set

$$\mathcal{D} = \{(\mathbf{x}_i, y_i) : \mathbf{x}_i \in \mathbb{R}^d, y_i \in \{-1, +1\}, i = 1, \ldots, m\}. \quad (1)$$

- For example, $\mathbf{x}_i$ could represent some encoding of an email message (say in the vector-space text model), and y_i is whether or not this message is spam.
- Note that we are working in a two-class setting, and this will be the case for the majority of our discussion. Some extensions to multi-class scenarios will be presented.
- We need to define a distribution D over the dataset $\mathcal{D}$ such that $\sum_i D(i) = 1$.

Weak Learners and Weak Classifiers

- First, we concretely define a **weak classifier**:

$$h_t : \mathbb{R}^d \rightarrow \{-1, +1\} \tag{2}$$

Weak Learners and Weak Classifiers

- First, we concretely define a **weak classifier**:

$$h_t : \mathbb{R}^d \rightarrow \{-1, +1\} \tag{2}$$

- A weak classifier must work better than chance. In the two-class setting this means it has less than 50% error and this is easy; if it would have higher than 50% error, just flip the sign. So, we want only a classifier that does not have exactly 50% error (since these classifiers would add no information).

Weak Learners and Weak Classifiers

- First, we concretely define a **weak classifier**:

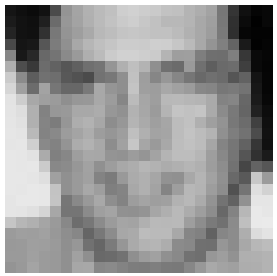
$$h_t : \mathbb{R}^d \rightarrow \{-1, +1\} \tag{2}$$

- A weak classifier must work better than chance. In the two-class setting this means it has less than 50% error and this is easy; if it would have higher than 50% error, just flip the sign. So, we want only a classifier that does not have exactly 50% error (since these classifiers would add no information).
- The error rate of a weak classifier $h_t(\mathbf{x})$ is calculated empirically over the training data:

$$\epsilon(h_t) = \frac{1}{m} \sum_{i=1}^{m} \delta(h_t(x_i) \neq y_i) < \frac{1}{2} \ . \tag{3}$$

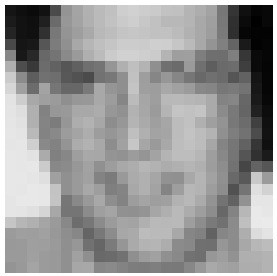
A WL/WC Example for Images

- Consider the case that our input data $\mathbf{x}_i$ are rectangular image patches.
  - And consider this case closely as it is part of project I.

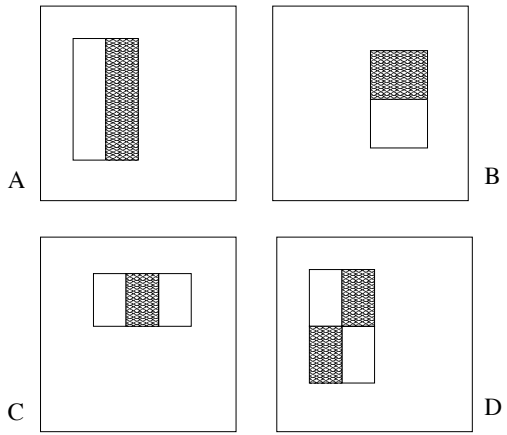


A WL/WC Example for Images

- Consider the case that our input data $\mathbf{x}_i$ are rectangular image patches.
  - And consider this case closely as it is part of project I.



- Define a collection of Haar-like rectangle features.
- The feature value extracted is the difference of the pixel sum in the white sub-regions and the black sub-regions.
- With a base patch size of 24x24, there are over 180,000 possible such rectangle features.



- Although these features are somewhat primitive in comparison to things like steerable filters, SIFT keys, etc., they do provide a rich set on which boosting can learn.

- Although these features are somewhat primitive in comparison to things like steerable filters, SIFT keys, etc., they do provide a rich set on which boosting can learn.
- And, they are quite efficiently computed when using the **integral image** representation.

- Although these features are somewhat primitive in comparison to things like steerable filters, SIFT keys, etc., they do provide a rich set on which boosting can learn.
- And, they are quite efficiently computed when using the **integral image** representation.
- Define the integral image as the image whose pixel value at a particular pixel x, y is the sum of the pixel values to the left and above x, y in the original image:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x, y) \tag{4}$$

where ii is the integral image and i is the original image.

- Although these features are somewhat primitive in comparison to things like steerable filters, SIFT keys, etc., they do provide a rich set on which boosting can learn.
- And, they are quite efficiently computed when using the **integral image** representation.
- Define the integral image as the image whose pixel value at a particular pixel x, y is the sum of the pixel values to the left and above x, y in the original image:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x, y) \tag{4}$$

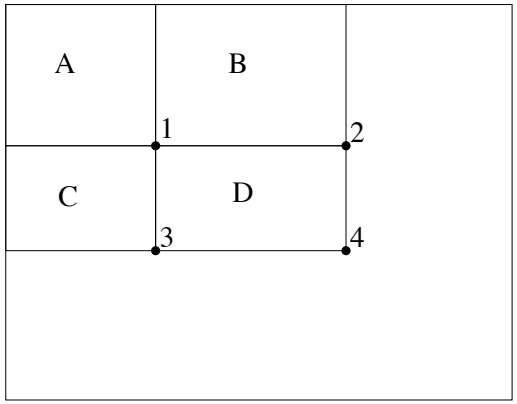
  where ii is the integral image and i is the original image.
- Use the following pair of recurrences to compute the integral image in just one pass.

$$s(x, y) = s(x, y - 1) + i(x, y) \tag{5}$$

$$ii(x, y) = ii(x - 1, y) + s(x, y) \tag{6}$$

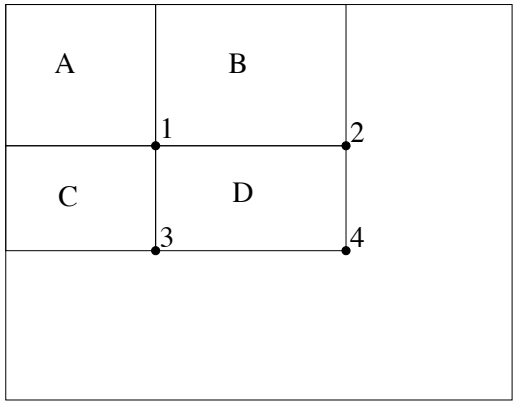
  where we define $s(x, -1) = 0$ and $ii(-1, y) = 0$.

- The sum of a particular rectangle can be computed in just 4 references using the integral image.
  - The value at point 1 is the sum of the pixels in rectangle A.
  - Point 2 is A+B.
  - Point 3 is A+C.
  - Point 4 is A+B+C+D.

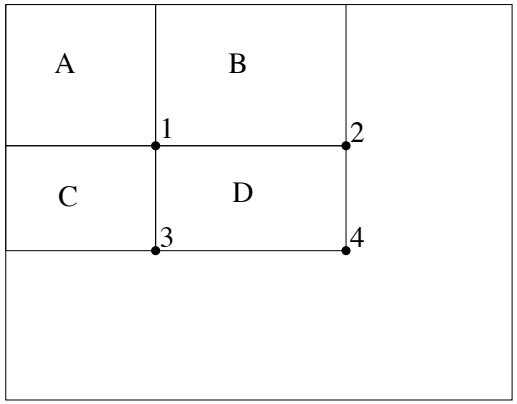


- The sum of a particular rectangle can be computed in just 4 references using the integral image.

  - The value at point 1 is the sum of the pixels in rectangle A.
  - Point 2 is A+B.
  - Point 3 is A+C.
  - Point 4 is A+B+C+D.
  - So, the sum within D alone is 4+1-2-3.



- The sum of a particular rectangle can be computed in just 4 references using the integral image.
  - The value at point 1 is the sum of the pixels in rectangle A.
  - Point 2 is A+B.
  - Point 3 is A+C.
  - Point 4 is A+B+C+D.
  - So, the sum within D alone is 4+1-2-3.



- We have a bunch of features. We certainly can't use them all. So, we let the boosting procedure select the best. But before we can do this, we need to pair these features with a simple weak learner.

- Each run, the weak learner is designed to select the single rectangle feature which best separates the positive and negative examples.
- The weak learner searches for the optimal threshold classification function, such that the minimum number of examples are misclassified.
- The weak classifier $h_t(\mathbf{x})$ hence consists of the feature $f_t(\mathbf{x})$, a threshold θ_t, and a parity p_t indicating the direction of the inequality sign:

$$h_t(\mathbf{x}) = \begin{cases} +1 & \text{if } p_t f_t(\mathbf{x}) < p_t \theta_t \\ -1 & \text{otherwise.} \end{cases} \tag{7}$$

The Strong AdaBoost Classifier

- Let's assume we have selected T weak classifiers and a scalar constant α_t associated with each:

$$h = \{h_t : t = 1, \ldots, T\} \tag{8}$$

$$\alpha = \{\alpha_t : t = 1, \ldots, T\} \tag{9}$$

The Strong AdaBoost Classifier

- Let's assume we have selected T weak classifiers and a scalar constant α_t associated with each:

$$h = \{h_t : t = 1, \ldots, T\} \tag{8}$$

$$\alpha = \{\alpha_t : t = 1, \ldots, T\} \tag{9}$$

- Denote the inner product over all weak classifiers as F:

$$F(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) = \langle \alpha, h(\mathbf{x}) \rangle \tag{10}$$

The Strong AdaBoost Classifier

- Let's assume we have selected T weak classifiers and a scalar constant α_t associated with each:

$$h = \{h_t : t = 1, \ldots, T\} \tag{8}$$

$$\alpha = \{\alpha_t : t = 1, \ldots, T\} \tag{9}$$

- Denote the inner product over all weak classifiers as F:

$$F(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) = \langle \alpha, h(\mathbf{x}) \rangle \tag{10}$$

- Define the **strong classifier** as the sign of this inner product:

$$H(\mathbf{x}) = \text{sign}\left[F(\mathbf{x})\right] = \text{sign}\left[\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right] \tag{11}$$

- Our objective is to choose h and α to minimize the empirical classification error of the strong classifier.

$$(h, \alpha)^* = \text{argmin}\, \text{Err}(H; \mathcal{D}) \tag{12}$$

$$= \text{argmin}\, \frac{1}{m} \sum_{i=1}^{m} \delta(H(\mathbf{x}_i) \neq y_i) \tag{13}$$

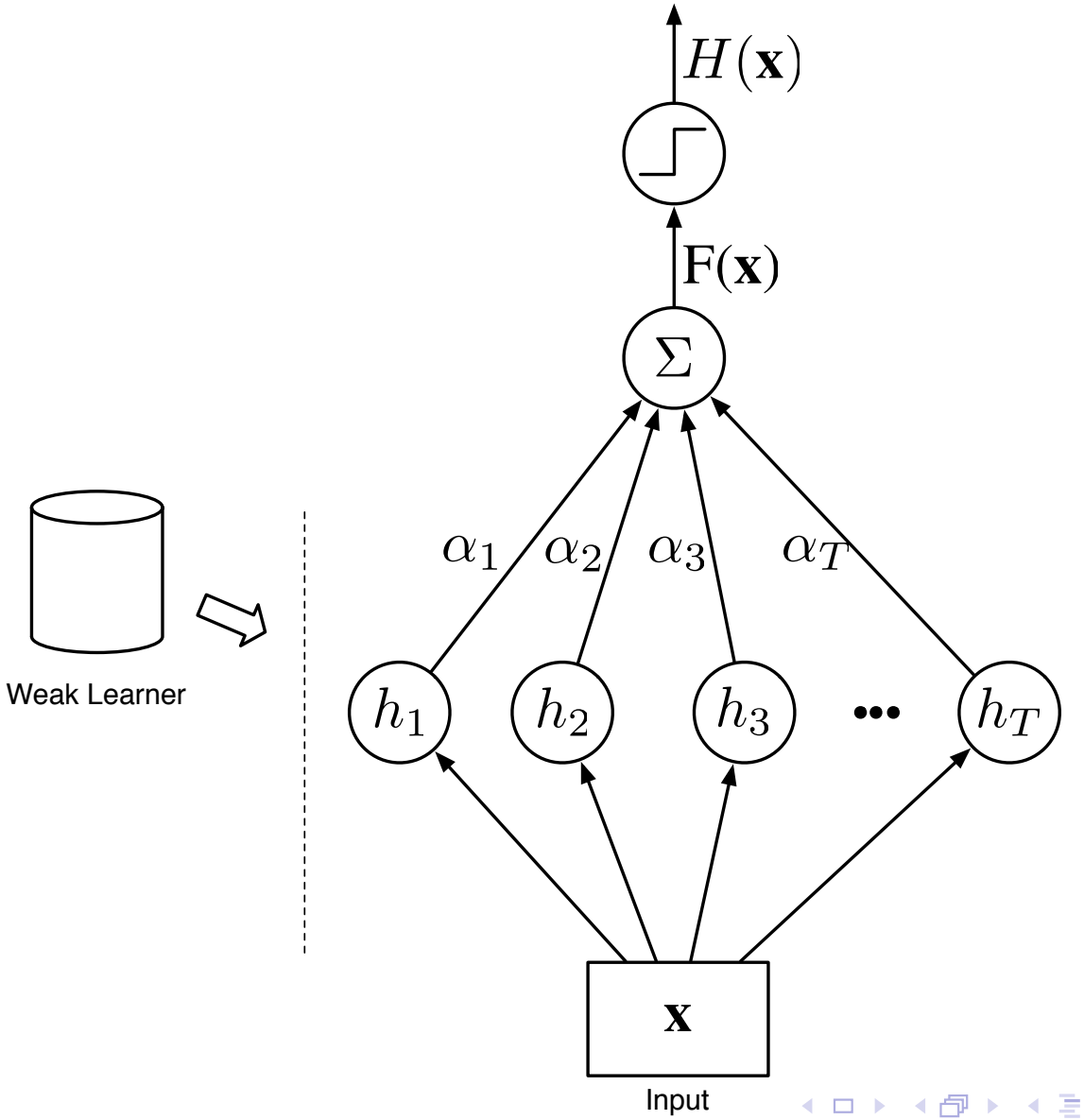
- Our objective is to choose h and α to minimize the empirical classification error of the strong classifier.

$$(h, \alpha)^* = \text{argmin}\, \text{Err}(H; \mathcal{D}) \tag{12}$$

$$= \text{argmin}\, \frac{1}{m} \sum_{i=1}^{m} \delta(H(\mathbf{x}_i) \neq y_i) \tag{13}$$

- Adaboost doesn't directly minimize this error but rather minimizes an upper bound on it.

Illustration of AdaBoost Classifier



The Basic AdaBoost Algorithm

Given $\mathcal{D} = (\mathbf{x}_i, y_i), \ldots, (\mathbf{x}_m, y_m)$ as before.
Initialize the distribution D_1 to be uniform: $D_1(i) = \frac{1}{m}$.
Repeat for $t = 1, \ldots, T$:

1. Learn weak classifier h_t using distribution D_t.
   - For the example given, this requires you to learn the threshold and the parity at each iteration given the current distribution D_t for the weak classifier h over each feature:
   - Note, there are other ways of doing this step...

The Basic AdaBoost Algorithm

Given $\mathcal{D} = (\mathbf{x}_i, y_i), \ldots, (\mathbf{x}_m, y_m)$ as before.
Initialize the distribution D_1 to be uniform: $D_1(i) = \frac{1}{m}$.
Repeat for $t = 1, \ldots, T$:

1. Learn weak classifier h_t using distribution D_t.
   - For the example given, this requires you to learn the threshold and the parity at each iteration given the current distribution D_t for the weak classifier h over each feature:
   1. Compute the weighted error for each weak classifier.

$$\epsilon_t(h) = \sum_{i=1}^{m} D_t(i)\delta(h(\mathbf{x}_i) \neq y_i), \quad \forall h \tag{14}$$

   - Note, there are other ways of doing this step...

The Basic AdaBoost Algorithm

Given $\mathcal{D} = (\mathbf{x}_i, y_i), \ldots, (\mathbf{x}_m, y_m)$ as before.
Initialize the distribution D_1 to be uniform: $D_1(i) = \frac{1}{m}$.
Repeat for $t = 1, \ldots, T$:

1. Learn weak classifier h_t using distribution D_t.
   - For the example given, this requires you to learn the threshold and the parity at each iteration given the current distribution D_t for the weak classifier h over each feature:

   1. Compute the weighted error for each weak classifier.

$$\epsilon_t(h) = \sum_{i=1}^{m} D_t(i)\delta(h(\mathbf{x}_i) \neq y_i), \quad \forall h \tag{14}$$

   2. Select the weak classifier with minimum error.

$$h_t = \text{argmin}_h \, \epsilon_t(h) \tag{15}$$

   - Note, there are other ways of doing this step...

2 Set weight α_t based on the error:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t(h_t)}{\epsilon_t(h_t)} \right) \tag{16}$$

2 Set weight α_t based on the error:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t(h_t)}{\epsilon_t(h_t)} \right) \tag{16}$$

3 Update the distribution based on the performance so far:

$$D_{t+1}(i) = \frac{1}{Z_t} D_t(i) \exp \left[-\alpha_t y_i h_t(x_i) \right] \tag{17}$$

where Z_t is a normalization factor to keep D_{t+1} a distribution. Note the careful evaluation of the term inside of the exp based on the possible $\{-1, +1\}$ values of the label.

2 Set weight α_t based on the error:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t(h_t)}{\epsilon_t(h_t)} \right) \tag{16}$$

3 Update the distribution based on the performance so far:

$$D_{t+1}(i) = \frac{1}{Z_t} D_t(i) \exp \left[-\alpha_t y_i h_t(x_i) \right] \tag{17}$$

where Z_t is a normalization factor to keep D_{t+1} a distribution. Note the careful evaluation of the term inside of the exp based on the possible $\{-1, +1\}$ values of the label.

One chooses T based on some established error criterion or some fixed number.

Contents for AdaBoost Analysis

- Facts about the weights and normalizing functions.
- AdaBoost Convergence (why and how fast).
- Why do we calculate the weight of each weak classifier to be

$$\alpha_t = \frac{1}{2} \ln \frac{1 - \epsilon_t(h_t)}{\epsilon_t(h_t)} \quad ?$$

- Why do we choose the weak classifier that has the minimum weighted error?
- Testing Error Analysis.

Facts About the Weights

Weak Classifier Weights

- The selected weight for each new weak classifier is always positive.

$$\epsilon_t(h_t) < \frac{1}{2} \Rightarrow \alpha_t = \frac{1}{2} \ln \frac{1 - \epsilon_t(h_t)}{\epsilon_t(h_t)} > 0 \tag{18}$$

Facts About the Weights

Weak Classifier Weights

- The selected weight for each new weak classifier is always positive.

$$\epsilon_t(h_t) < \frac{1}{2} \Rightarrow \alpha_t = \frac{1}{2} \ln \frac{1 - \epsilon_t(h_t)}{\epsilon_t(h_t)} > 0 \tag{18}$$

- The smaller the classification error, the bigger the weight and the more this particular weak classifier will impact the final strong classifier.

$$\epsilon(h_A) < \epsilon(h_B) \Rightarrow \alpha_A > \alpha_B \tag{19}$$

Facts About the Weights

Data Sample Weights

- The weights of the data points are multiplied by $\exp\left[-y_i \alpha_t h_t(\mathbf{x}_i)\right]$.

Facts About the Weights

Data Sample Weights

- The weights of the data points are multiplied by $\exp\left[-y_i \alpha_t h_t(\mathbf{x}_i)\right]$.

$$\exp\left[-y_i \alpha_t h_t(\mathbf{x}_i)\right] = \begin{cases} \exp\left[-\alpha_t\right] < 1 & \text{if } h_t(\mathbf{x}_i) = y_i \\ \exp\left[\alpha_t\right] > 1 & \text{if } h_t(\mathbf{x}_i) \neq y_i \end{cases} \tag{20}$$

- The weights of correctly classified points are reduced and the weights of incorrectly classified points are increased. Hence, the incorrectly classified points will receive more attention in the next run.

- The weight distribution can be computed recursively:

$$
\begin{aligned}
D_{t+1}(i) &= \frac{1}{Z_t} D_t(i) \exp\left[-\alpha_t y_i h_t(x_i)\right] \qquad (21)\\
&= \frac{1}{Z_{t-1} Z_t} D_{t-1}(i) \exp\left[-y_i \left(\alpha_t h_t(x_i) + \alpha_{t-1} h_{t-1}(x_i)\right)\right]\\
&= \ldots\\
&= \frac{1}{Z_1 \ldots Z_t} D_1(i) \exp\left[-y_i \left(\alpha_t h_t(x_i) + \cdots + \alpha_1 h_1(x_i)\right)\right]
\end{aligned}
$$

Facts About the Normalizing Functions

- At each iteration, the weights on the data points are normalized by

$$Z_t = \sum_{\mathbf{x}_i} D_t(\mathbf{x}_i) \exp\left[-y_i \alpha_i h_t(\mathbf{x}_i)\right] \tag{22}$$

$$= \sum_{\mathbf{x}_i \in \mathcal{A}} D_t(\mathbf{x}_i) \exp\left[-\alpha_t\right] + \sum_{\mathbf{x}_i \in \overline{\mathcal{A}}} D_t(\mathbf{x}_i) \exp\left[\alpha_t\right] \tag{23}$$

where $\mathcal{A}$ is the set of correctly classified points: $\{\mathbf{x}_i : y_i = h_t(\mathbf{x}_i)\}$.

Facts About the Normalizing Functions

- At each iteration, the weights on the data points are normalized by

$$Z_t = \sum_{\mathbf{x}_i} D_t(\mathbf{x}_i) \exp\left[-y_i \alpha_i h_t(\mathbf{x}_i)\right] \tag{22}$$

$$= \sum_{\mathbf{x}_i \in \mathcal{A}} D_t(\mathbf{x}_i) \exp\left[-\alpha_t\right] + \sum_{\mathbf{x}_i \in \overline{\mathcal{A}}} D_t(\mathbf{x}_i) \exp\left[\alpha_t\right] \tag{23}$$

  where $\mathcal{A}$ is the set of correctly classified points: $\{\mathbf{x}_i : y_i = h_t(\mathbf{x}_i)\}$.
- We can write these normalization factors as functions of α_t, then:

$$Z_t = Z_t(\alpha_t) \tag{24}$$

- Recall the data weights can be computed recursively:

$$D_{t+1}(i) = \frac{1}{Z_1 \dots Z_t} \frac{1}{m} \exp\left[-y_i F(\mathbf{x}_i)\right] \quad . \tag{25}$$

- Recall the data weights can be computed recursively:

$$D_{t+1}(i) = \frac{1}{Z_1 \dots Z_t} \frac{1}{m} \exp\left[-y_i F(\mathbf{x}_i)\right] \quad . \tag{25}$$

- And, since we know the data weights must sum to one, we have

$$\sum_{i=1}^{m} D_t(\mathbf{x}_i) = \frac{1}{Z_1 \dots Z_t} \frac{1}{m} \sum_{i=1}^{m} \exp\left[-y_i F(\mathbf{x}_i)\right] = 1 \tag{26}$$

- Recall the data weights can be computed recursively:

$$D_{t+1}(i) = \frac{1}{Z_1 \dots Z_t} \frac{1}{m} \exp\left[-y_i F(\mathbf{x}_i)\right] \quad . \tag{25}$$

- And, since we know the data weights must sum to one, we have

$$\sum_{i=1}^{m} D_t(\mathbf{x}_i) = \frac{1}{Z_1 \dots Z_t} \frac{1}{m} \sum_{i=1}^{m} \exp\left[-y_i F(\mathbf{x}_i)\right] = 1 \tag{26}$$

- Therefore, we can summarize this with a new normalizing function:

$$Z = Z_1 \dots Z_t = \frac{1}{m} \sum_{i=1}^{m} \exp\left[-y_i F(\mathbf{x}_i)\right] \quad . \tag{27}$$

AdaBoost Convergence

- Key Idea: **AdaBoost minimizes an upper bound on the classification error**.

AdaBoost Convergence

- Key Idea: **AdaBoost minimizes an upper bound on the classification error**.
- Claim: After t steps, the error of the strong classifier is bounded above by quantity Z, as we just defined it (the product of the data weight normalization factors):

$$\text{Err}(H) \leq Z = Z(\alpha, h) = Z_t(\alpha_t, h_t) \dots Z_1(\alpha_1, h_1) \tag{28}$$

AdaBoost Convergence

- Key Idea: **AdaBoost minimizes an upper bound on the classification error**.
- Claim: After t steps, the error of the strong classifier is bounded above by quantity Z, as we just defined it (the product of the data weight normalization factors):

$$\text{Err}(H) \leq Z = Z(\alpha, h) = Z_t(\alpha_t, h_t) \dots Z_1(\alpha_1, h_1) \tag{28}$$

- AdaBoost is a greedy algorithm that minimizes this upper bound on the classification error by choosing the optimal h_t and α_t to minimize Z_t at each step.

$$(h, \alpha)^* = \text{argmin}\, Z(\alpha, h) \tag{29}$$

$$(h_t, \alpha_t)^* = \text{argmin}\, Z_t(\alpha_t, h_t) \tag{30}$$

AdaBoost Convergence

- Key Idea: **AdaBoost minimizes an upper bound on the classification error**.
- Claim: After t steps, the error of the strong classifier is bounded above by quantity Z, as we just defined it (the product of the data weight normalization factors):

$$\text{Err}(H) \leq Z = Z(\alpha, h) = Z_t(\alpha_t, h_t) \dots Z_1(\alpha_1, h_1) \tag{28}$$

- AdaBoost is a greedy algorithm that minimizes this upper bound on the classification error by choosing the optimal h_t and α_t to minimize Z_t at each step.

$$(h, \alpha)^* = \text{argmin}\, Z(\alpha, h) \tag{29}$$

$$(h_t, \alpha_t)^* = \text{argmin}\, Z_t(\alpha_t, h_t) \tag{30}$$

- As Z goes to zero, the classification error goes to zero. Hence, it converges. (But, we need to account for the case when no new weak classifier has an error rate better than 0.5, upon which time we should stop.)

- We need to show the claim on the error bound is true:

$$\text{Err}(H) = \frac{1}{m}\sum_{i=1}^{m}\delta(H(\mathbf{x}_i) \neq y_i) \leq Z = \frac{1}{m}\sum_{i=1}^{m}\exp\left[-y_i F(\mathbf{x}_i)\right] \quad (31)$$

- We need to show the claim on the error bound is true:

$$\text{Err}(H) = \frac{1}{m}\sum_{i=1}^{m}\delta(H(\mathbf{x}_i) \neq y_i) \leq Z = \frac{1}{m}\sum_{i=1}^{m}\exp\left[-y_i F(\mathbf{x}_i)\right] \quad (31)$$

- Proof:

$$F(\mathbf{x}_i) = \text{sign}(F(\mathbf{x}_i))|F(\mathbf{x}_i)| \quad (32)$$

$$= H(\mathbf{x}_i)|F(\mathbf{x}_i)| \quad (33)$$

- We need to show the claim on the error bound is true:

$$\text{Err}(H) = \frac{1}{m}\sum_{i=1}^{m}\delta(H(\mathbf{x}_i) \neq y_i) \leq Z = \frac{1}{m}\sum_{i=1}^{m}\exp\left[-y_i F(\mathbf{x}_i)\right] \quad (31)$$

- Proof:

$$F(\mathbf{x}_i) = \text{sign}(F(\mathbf{x}_i))|F(\mathbf{x}_i)| \quad (32)$$
$$= H(\mathbf{x}_i)|F(\mathbf{x}_i)| \quad (33)$$

- The two cases are:

$$\text{If } H(\mathbf{x}_i) \neq y_i \text{ then the LHS} = 1 \leq \text{RHS} = e^{+|F(\mathbf{x}_i)|}.$$
$$\text{If } H(\mathbf{x}_i) = y_i \text{ then the LHS} = 0 \leq \text{RHS} = e^{-|F(\mathbf{x}_i)|}.$$

- We need to show the claim on the error bound is true:

$$\text{Err}(H) = \frac{1}{m}\sum_{i=1}^{m}\delta(H(\mathbf{x}_i) \neq y_i) \leq Z = \frac{1}{m}\sum_{i=1}^{m}\exp\left[-y_i F(\mathbf{x}_i)\right] \quad (31)$$

- Proof:

$$F(\mathbf{x}_i) = \text{sign}(F(\mathbf{x}_i))|F(\mathbf{x}_i)| \quad (32)$$
$$= H(\mathbf{x}_i)|F(\mathbf{x}_i)| \quad (33)$$

- The two cases are:

$$\text{If } H(\mathbf{x}_i) \neq y_i \text{ then the LHS} = 1 \leq \text{RHS} = e^{+|F(\mathbf{x}_i)|}.$$
$$\text{If } H(\mathbf{x}_i) = y_i \text{ then the LHS} = 0 \leq \text{RHS} = e^{-|F(\mathbf{x}_i)|}.$$

- So, the inequality holds for each term

$$\delta(H(\mathbf{x}_i) \neq y_i) \leq \exp\left[-y_i F(\mathbf{x}_i)\right] \quad (34)$$

and hence, the inequality is true.

Weak Classifier Pursuit

- Now, we want to explore how we are solving the step-wise minimization problem:

$$(h_t, \alpha_t)^* = \text{argmin}\, Z(\alpha_t, h_t) \tag{35}$$

- Recall, we can separate Z into two parts:

$$Z_t(\alpha_t, h_t) = \sum_{\mathbf{x}_i \in \mathcal{A}} D_t(\mathbf{x}_i) \exp[-\alpha_t] + \sum_{\mathbf{x}_i \in \overline{\mathcal{A}}} D_t(\mathbf{x}_i) \exp[\alpha_t] \tag{36}$$

where $\mathcal{A}$ is the set of correctly classified points: $\{\mathbf{x}_i : y_i = h_t(\mathbf{x}_i)\}$.

Weak Classifier Pursuit

- Now, we want to explore how we are solving the step-wise minimization problem:

$$(h_t, \alpha_t)^* = \text{argmin}\, Z(\alpha_t, h_t) \tag{35}$$

- Recall, we can separate Z into two parts:

$$Z_t(\alpha_t, h_t) = \sum_{\mathbf{x}_i \in \mathcal{A}} D_t(\mathbf{x}_i) \exp[-\alpha_t] + \sum_{\mathbf{x}_i \in \overline{\mathcal{A}}} D_t(\mathbf{x}_i) \exp[\alpha_t] \tag{36}$$

where $\mathcal{A}$ is the set of correctly classified points: $\{\mathbf{x}_i : y_i = h_t(\mathbf{x}_i)\}$.

- Take the derivative w.r.t. α_t and set it to zero:

$$\frac{dZ_t(\alpha_t, h_t)}{d\alpha_t} = \sum_{\mathbf{x}_i \in \mathcal{A}} -D_t(\mathbf{x}_i) \exp[-\alpha_t] + \sum_{\mathbf{x}_i \in \overline{\mathcal{A}}} D_t(\mathbf{x}_i) \exp[\alpha_t] = 0 \tag{37}$$

$$\frac{dZ_t(\alpha_t, h_t)}{d\alpha_t} = \sum_{\mathbf{x}_i \in \mathcal{A}} -D_t(\mathbf{x}_i) \exp\left[-\alpha_t\right] + \sum_{\mathbf{x}_i \in \overline{\mathcal{A}}} D_t(\mathbf{x}_i) \exp\left[\alpha_t\right] = 0 \quad (38)$$

$$\sum_{\mathbf{x}_i \in \mathcal{A}} D_t(\mathbf{x}_i) = \sum_{\mathbf{x}_i \in \overline{\mathcal{A}}} D_t(\mathbf{x}_i) \exp\left[2\alpha_t\right] \quad (39)$$

$$\frac{dZ_t(\alpha_t, h_t)}{d\alpha_t} = \sum_{\mathbf{x}_i \in \mathcal{A}} -D_t(\mathbf{x}_i) \exp\left[-\alpha_t\right] + \sum_{\mathbf{x}_i \in \overline{\mathcal{A}}} D_t(\mathbf{x}_i) \exp\left[\alpha_t\right] = 0 \quad (38)$$

$$\sum_{\mathbf{x}_i \in \mathcal{A}} D_t(\mathbf{x}_i) = \sum_{\mathbf{x}_i \in \overline{\mathcal{A}}} D_t(\mathbf{x}_i) \exp\left[2\alpha_t\right] \quad (39)$$

- And, by definition, we can write the error as

$$\epsilon_t(h) = \sum_{i=1}^{m} D_t(\mathbf{x}_i)\delta(h(\mathbf{x}_i) \neq y_i) = \sum_{\mathbf{x}_i \in \overline{\mathcal{A}}} D_t(\mathbf{x}_i), \quad \forall h \quad (40)$$

$$\frac{dZ_t(\alpha_t, h_t)}{d\alpha_t} = \sum_{\mathbf{x}_i \in \mathcal{A}} -D_t(\mathbf{x}_i) \exp[-\alpha_t] + \sum_{\mathbf{x}_i \in \overline{\mathcal{A}}} D_t(\mathbf{x}_i) \exp[\alpha_t] = 0 \quad (38)$$

$$\sum_{\mathbf{x}_i \in \mathcal{A}} D_t(\mathbf{x}_i) = \sum_{\mathbf{x}_i \in \overline{\mathcal{A}}} D_t(\mathbf{x}_i) \exp[2\alpha_t] \quad (39)$$

- And, by definition, we can write the error as

$$\epsilon_t(h) = \sum_{i=1}^{m} D_t(\mathbf{x}_i)\delta(h(\mathbf{x}_i) \neq y_i) = \sum_{\mathbf{x}_i \in \overline{\mathcal{A}}} D_t(\mathbf{x}_i), \quad \forall h \quad (40)$$

- Rewriting (39) and solving for α_t yields

$$\alpha_t = \frac{1}{2} \ln \frac{1 - \epsilon_t(h_t)}{\epsilon_t(h_t)} \quad (41)$$

- We can plug it back into the normalization term to get the minimum:

$$Z_t(\alpha_t, h_t) = \sum_{\mathbf{x}_i \in \mathcal{A}} D_t(\mathbf{x}_i) \exp[-\alpha_t] + \sum_{\mathbf{x}_i \in \overline{\mathcal{A}}} D_t(\mathbf{x}_i) \exp[\alpha_t] \tag{42}$$

$$= (1 - \epsilon_t(h_t)) \sqrt{\frac{\epsilon_t(h_t)}{1 - \epsilon_t(h_t)}} + \epsilon_t(h_t) \sqrt{\frac{1 - \epsilon_t(h_t)}{\epsilon_t(h_t)}} \tag{43}$$

$$= 2\sqrt{\epsilon_t(h_t)(1 - \epsilon_t(h_t))} \tag{44}$$

- We can plug it back into the normalization term to get the minimum:

$$Z_t(\alpha_t, h_t) = \sum_{\mathbf{x}_i \in \mathcal{A}} D_t(\mathbf{x}_i) \exp[-\alpha_t] + \sum_{\mathbf{x}_i \in \overline{\mathcal{A}}} D_t(\mathbf{x}_i) \exp[\alpha_t] \tag{42}$$

$$= (1 - \epsilon_t(h_t)) \sqrt{\frac{\epsilon_t(h_t)}{1 - \epsilon_t(h_t)}} + \epsilon_t(h_t) \sqrt{\frac{1 - \epsilon_t(h_t)}{\epsilon_t(h_t)}} \tag{43}$$

$$= 2\sqrt{\epsilon_t(h_t)(1 - \epsilon_t(h_t))} \tag{44}$$

- Change a variable, $\gamma_t = \frac{1}{2} - \epsilon_t(h_t)$, $\gamma_t \in (0, \frac{1}{2}]$.

- We can plug it back into the normalization term to get the minimum:

$$Z_t(\alpha_t, h_t) = \sum_{\mathbf{x}_i \in \mathcal{A}} D_t(\mathbf{x}_i) \exp\left[-\alpha_t\right] + \sum_{\mathbf{x}_i \in \overline{\mathcal{A}}} D_t(\mathbf{x}_i) \exp\left[\alpha_t\right] \tag{42}$$

$$= (1 - \epsilon_t(h_t))\sqrt{\frac{\epsilon_t(h_t)}{1 - \epsilon_t(h_t)}} + \epsilon_t(h_t)\sqrt{\frac{1 - \epsilon_t(h_t)}{\epsilon_t(h_t)}} \tag{43}$$

$$= 2\sqrt{\epsilon_t(h_t)(1 - \epsilon_t(h_t))} \tag{44}$$

- Change a variable, $\gamma_t = \frac{1}{2} - \epsilon_t(h_t)$, $\gamma_t \in (0, \frac{1}{2}]$.
- **Then, we have the minimum to be**

$$Z_t(\alpha_t, h_t) = 2\sqrt{\epsilon_t(h_t)(1 - \epsilon_t(h_t))} \tag{45}$$

$$= \sqrt{1 - 4\gamma_t^2} \tag{46}$$

$$\leq \exp\left[-2\gamma_t^2\right] \tag{47}$$

- **Therefore, after t steps, the error rate of the strong classifier is bounded on top by**

$$\text{Err(H)} \leq Z \leq \exp\left[-2\sum_{t=1}^{T}\gamma_t^2\right] \tag{48}$$

- **Therefore, after t steps, the error rate of the strong classifier is bounded on top by**

$$\text{Err(H)} \leq Z \leq \exp\left[-2\sum_{t=1}^{T}\gamma_t^2\right] \tag{48}$$

- Hence, each step decreases the upper bound exponentially.

- **Therefore, after t steps, the error rate of the strong classifier is bounded on top by**

$$\text{Err(H)} \leq Z \leq \exp\left[-2\sum_{t=1}^{T}\gamma_t^2\right] \tag{48}$$

- Hence, each step decreases the upper bound exponentially.
- And, a weak classifier with small error rate will lead to faster descent.

Summary of AdaBoost Convergence

- The objective of AdaBoost is to minimize an upper bound on the classification error:

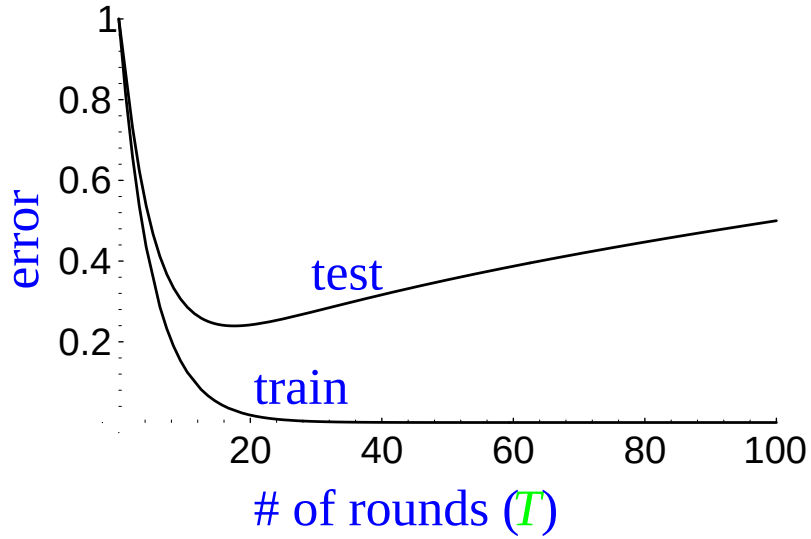
$$(\alpha, h)^* = \text{argmin}\, Z(\alpha, h) \tag{49}$$

$$= \text{argmin}\, Z_t(\alpha_t, h_t) \dots Z_1(\alpha_1, h_1) \tag{50}$$

$$= \text{argmin} \sum_{i=1}^{m} \exp\left[-y_i \langle \alpha, h(\mathbf{x}_i) \rangle\right] \tag{51}$$

- AdaBoost takes a stepwise minimization scheme, which may not be optimal (it is greedy). When we calculate the parameter for the t^{th} weak classifier, the others remain set.
- We should stop AdaBoost if all of the weak classifiers have an error rate of $\frac{1}{2}$.

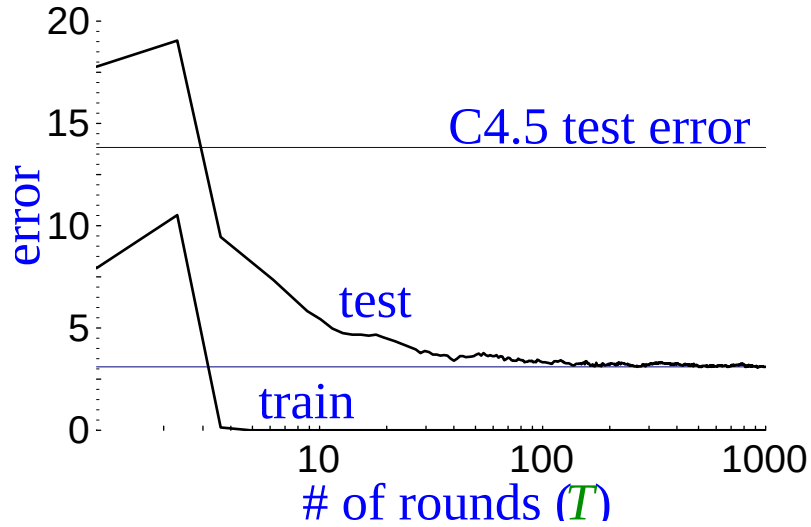
How Will Test Error Behave? (A First Guess)



expect:

- training error to continue to drop (or reach zero)
- test error to increase when H_{final} becomes "too complex"
  - "Occam's razor"
  - overfitting
    - hard to know when to stop training

Actual Typical Run



(boosting C4.5 on "letter" dataset)

- test error does not increase, even after 1000 rounds
  - (total size $>$ 2,000,000 nodes)
- test error continues to drop even after training error is zero!

	# rounds		
	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1

- Occam's razor wrongly predicts "simpler" rule is better

A Better Story: The Margins Explanation

[with Freund, Bartlett & Lee]

- key idea:
  - training error only measures whether classifications are right or wrong
  - should also consider confidence of classifications

A Better Story: The Margins Explanation

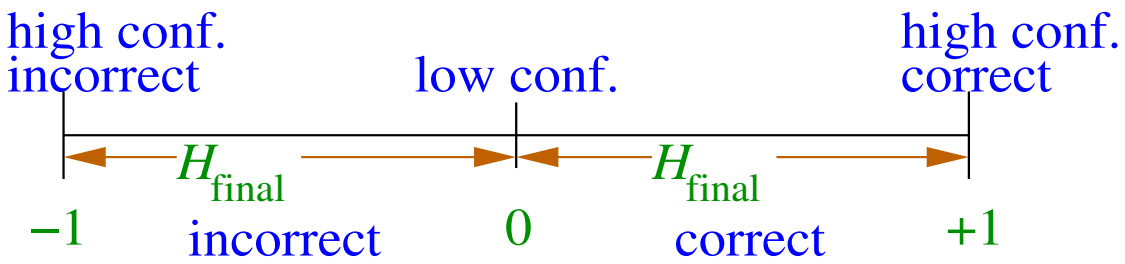
[with Freund, Bartlett & Lee]

- key idea:
  - training error only measures whether classifications are right or wrong
  - should also consider confidence of classifications
- recall: H_{final} is weighted majority vote of weak classifiers

A Better Story: The Margins Explanation

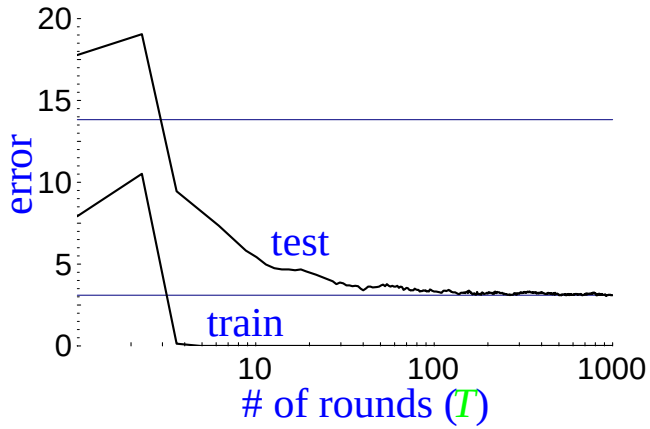
[with Freund, Bartlett & Lee]

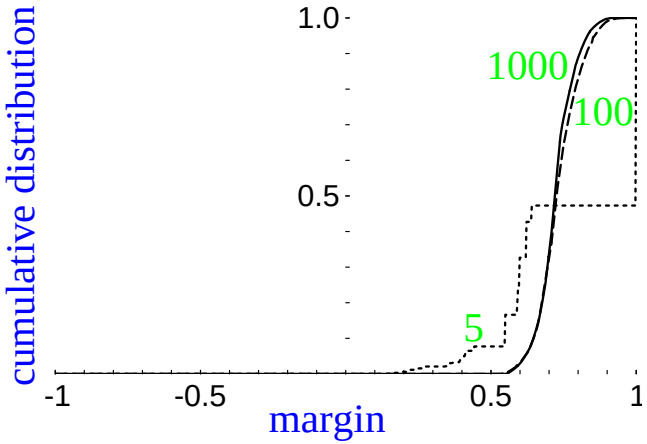
- key idea:
  - training error only measures whether classifications are right or wrong
  - should also consider confidence of classifications
- recall: H_{final} is weighted majority vote of weak classifiers
- measure confidence by margin = strength of the vote
  = (fraction voting correctly) − (fraction voting incorrectly)



Empirical Evidence: The Margin Distribution

- margin distribution
  = cumulative distribution of margins of training examples





	# rounds		
	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1
% margins ≤ 0.5	7.7	0.0	0.0
minimum margin	0.14	0.52	0.55

Theoretical Evidence: Analyzing Boosting Using Margins

- Theorem: large margins $\Rightarrow$ better bound on generalization error (independent of number of rounds)

Theoretical Evidence: Analyzing Boosting Using Margins

- Theorem: large margins $\Rightarrow$ better bound on generalization error (independent of number of rounds)
  - proof idea: if all margins are large, then can approximate final classifier by a much smaller classifier (just as polls can predict not-too-close election)

Theoretical Evidence: Analyzing Boosting Using Margins

- Theorem: large margins $\Rightarrow$ better bound on generalization error (independent of number of rounds)
  - proof idea: if all margins are large, then can approximate final classifier by a much smaller classifier (just as polls can predict not-too-close election)
- Theorem: boosting tends to increase margins of training examples (given weak learning assumption)

Theoretical Evidence: Analyzing Boosting Using Margins

- Theorem: large margins $\Rightarrow$ better bound on generalization error (independent of number of rounds)
  - proof idea: if all margins are large, then can approximate final classifier by a much smaller classifier (just as polls can predict not-too-close election)
- Theorem: boosting tends to increase margins of training examples (given weak learning assumption)
  - proof idea: similar to training error proof

Theoretical Evidence: Analyzing Boosting Using Margins

- Theorem: large margins $\Rightarrow$ better bound on generalization error (independent of number of rounds)
  - proof idea: if all margins are large, then can approximate final classifier by a much smaller classifier (just as polls can predict not-too-close election)
- Theorem: boosting tends to increase margins of training examples (given weak learning assumption)
  - proof idea: similar to training error proof
- so:
  although final classifier is getting larger,
  margins are likely to be increasing,
  so final classifier actually getting close to a simpler classifier,
  driving down the test error

More Technically...

- with high probability, $\forall \theta > 0$:

$$\text{generalization error} \leq \hat{\Pr}[\text{margin} \leq \theta] + \tilde{O}\left(\frac{\sqrt{d/m}}{\theta}\right)$$

($\hat{\Pr}[\,] =$ empirical probability)

  - bound depends on
    - $m = \#$ training examples
    - $d =$ "complexity" of weak classifiers
    - entire distribution of margins of training examples
- $\hat{\Pr}[\text{margin} \leq \theta] \rightarrow 0$ exponentially fast (in T) if (error of h_t on D_t) $< 1/2 - \theta$ $(\forall t)$
  - so: if weak learning assumption holds, then all examples will quickly have "large" margins

Summary of Basic AdaBoost

- AdaBoost is a sequential algorithm that minimizes an upper bound of the empirical classification error by selecting the weak classifiers and their weights. These are "pursued" one-by-one with each one being selected to maximally reduce the upper bound of error.

Summary of Basic AdaBoost

- AdaBoost is a sequential algorithm that minimizes an upper bound of the empirical classification error by selecting the weak classifiers and their weights. These are "pursued" one-by-one with each one being selected to maximally reduce the upper bound of error.
- AdaBoost defines a distribution of weights over the data samples. These weights are updated each time a new weak classifier is added such that samples misclassified by this new weak classifiers are given more weight. In this manner, currently misclassified samples are emphasized more during the selection of the subsequent weak classifier.

Summary of Basic AdaBoost

- AdaBoost is a sequential algorithm that minimizes an upper bound of the empirical classification error by selecting the weak classifiers and their weights. These are "pursued" one-by-one with each one being selected to maximally reduce the upper bound of error.
- AdaBoost defines a distribution of weights over the data samples. These weights are updated each time a new weak classifier is added such that samples misclassified by this new weak classifiers are given more weight. In this manner, currently misclassified samples are emphasized more during the selection of the subsequent weak classifier.
- The empirical error will converge to zero at an exponential rate.

Practical AdaBoost Advantages

- It is fast to evaluate (linear-additive) and can be fast to train (depending on weak learner).
- T is the only parameter to tune.
- It is flexible and can be combined with any weak learner.
- It is provably effective if it can consistently find the weak classifiers (that do better than random).
- Since it can work with any weak learner, it can handle the gamut of data.

AdaBoost Caveats

- Performance depends on the data and the weak learner.
- It can fail if
  - The weak classifiers are too complex and overfit.
  - The weak classifiers are too weak, essentially underfitting.
- AdaBoost seems, empirically, to be especially susceptible to uniform noise.

Coordinate Descent

[Breiman]

- $\{g_1, \ldots, g_N\}$ = space of all weak classifiers
- want to find $\lambda_1, \ldots, \lambda_N$ to minimize

$$L(\lambda_1, \ldots, \lambda_N) = \sum_i \exp\left(-y_i \sum_j \lambda_j g_j(x_i)\right)$$

Coordinate Descent

[Breiman]

- $\{g_1, \ldots, g_N\}$ = space of all weak classifiers
- want to find $\lambda_1, \ldots, \lambda_N$ to minimize

$$L(\lambda_1, \ldots, \lambda_N) = \sum_i \exp\left(-y_i \sum_j \lambda_j g_j(x_i)\right)$$

- AdaBoost is actually doing coordinate descent on this optimization problem:
  - initially, all $\lambda_j = 0$
  - each round: choose one coordinate λ_j (corresponding to h_t) and update (increment by α_t)
  - choose update causing biggest decrease in loss
- powerful technique for minimizing over huge space of functions

Estimating Conditional Probabilities

[Friedman, Hastie & Tibshirani]

- often want to estimate probability that $y = +1$ given x
- AdaBoost minimizes (empirical version of):

$$\mathrm{E}_{x,y}\left[e^{-yf(x)}\right] = \mathrm{E}_x\left[\mathrm{P}\left[y=+1|x\right]e^{-f(x)} + \mathrm{P}\left[y=-1|x\right]e^{f(x)}\right]$$

where x, y random from true distribution

Estimating Conditional Probabilities

[Friedman, Hastie & Tibshirani]

- often want to estimate probability that $y = +1$ given x
- AdaBoost minimizes (empirical version of):

$$\mathrm{E}_{x,y}\left[e^{-yf(x)}\right] = \mathrm{E}_x\left[\mathrm{P}\left[y=+1|x\right]e^{-f(x)} + \mathrm{P}\left[y=-1|x\right]e^{f(x)}\right]$$

  where x, y random from true distribution

- over all f, minimized when

$$f(x) = \frac{1}{2}\cdot \ln\left(\frac{\mathrm{P}\left[y=+1|x\right]}{\mathrm{P}\left[y=-1|x\right]}\right)$$

  or

$$\mathrm{P}\left[y=+1|x\right] = \frac{1}{1+e^{-2f(x)}}$$

- so, to convert f output by AdaBoost to probability estimate, use same formula

Multiclass Problems

[with Freund]

- say $y \in Y = \{1, \ldots, k\}$
- direct approach (AdaBoost.M1):

$$h_t : X \to Y$$

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$$

$$H_{\text{final}}(x) = \arg\max_{y \in Y} \sum_{t : h_t(x) = y} \alpha_t$$

Multiclass Problems

[with Freund]

- say $y \in Y = \{1, \ldots, k\}$
- direct approach (AdaBoost.M1):

$$h_t : X \to Y$$

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$$

$$H_{\text{final}}(x) = \arg\max_{y \in Y} \sum_{t : h_t(x) = y} \alpha_t$$

- can prove same bound on error if $\forall t : \epsilon_t \leq 1/2$
  - in practice, not usually a problem for "strong" weak learners (e.g., C4.5)
  - significant problem for "weak" weak learners (e.g., decision stumps)
- instead, reduce to binary

Reducing Multiclass to Binary

[with Singer]

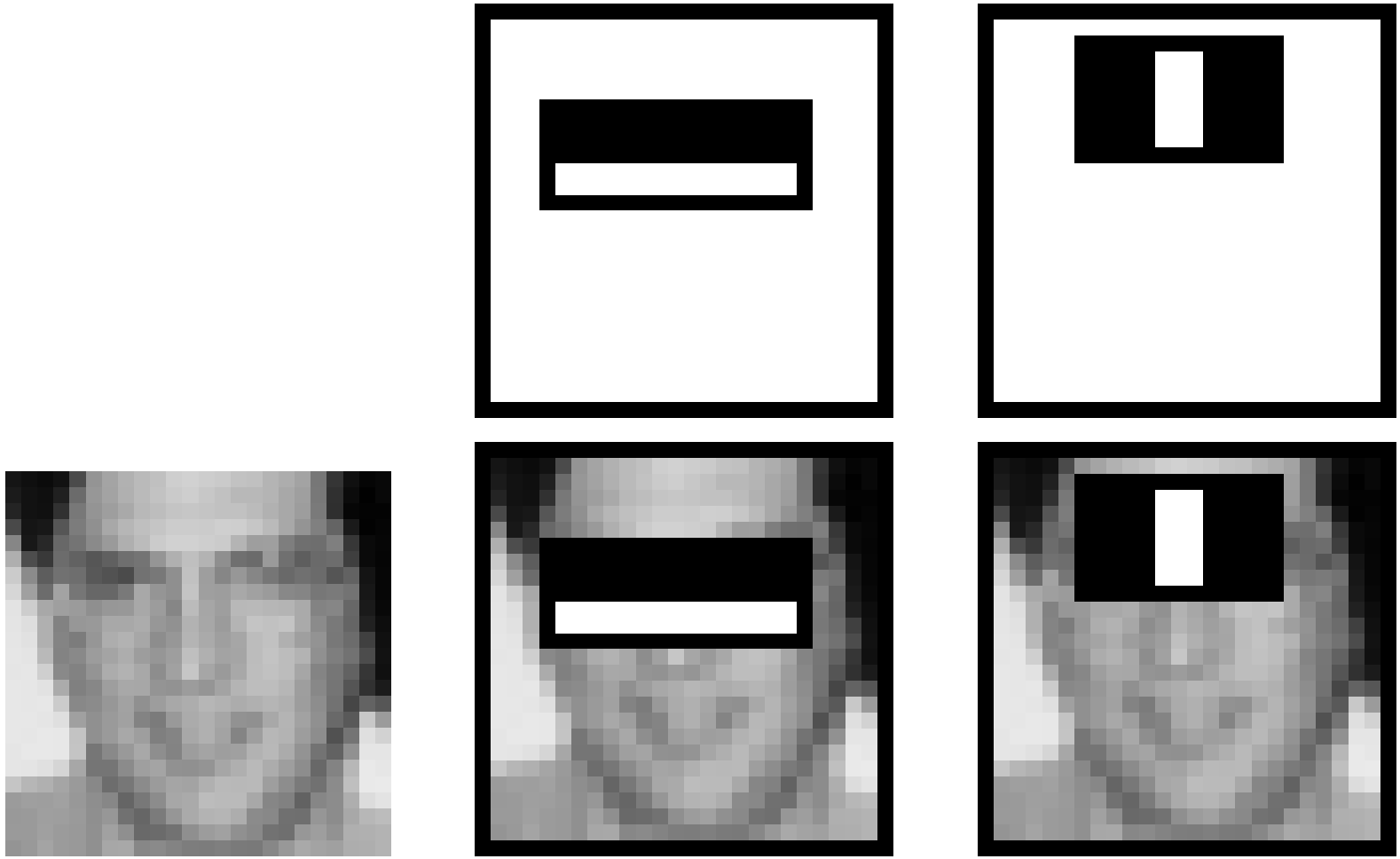
- say possible labels are $\{a, b, c, d, e\}$
- each training example replaced by five $\{-1, +1\}$-labeled examples:

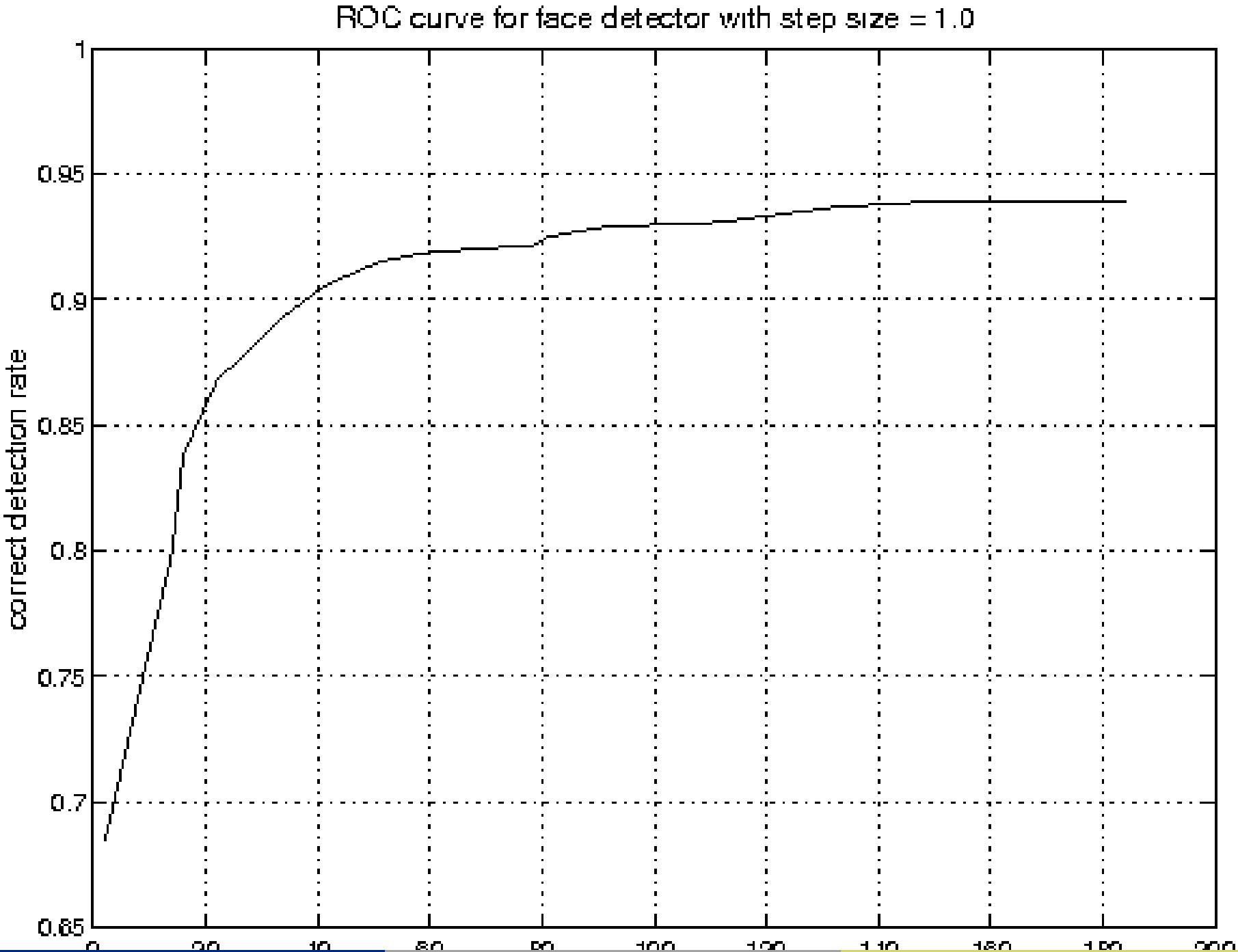
$$x \quad , \quad c \quad \rightarrow \begin{cases} (x, a) & , \quad -1 \\ (x, b) & , \quad -1 \\ (x, c) & , \quad +1 \\ (x, d) & , \quad -1 \\ (x, e) & , \quad -1 \end{cases}$$

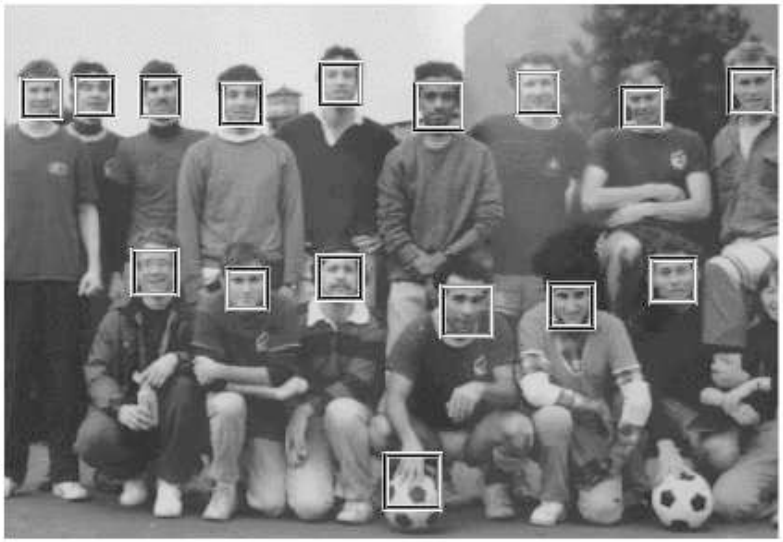
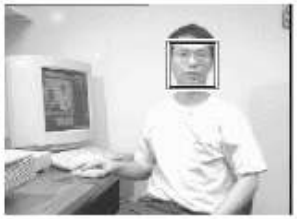
- predict with label receiving most (weighted) votes

AdaBoost for Face Detection

- Viola and Jones 2000-2002.




ROC curve for face detector with step size = 1.0
correct detection rate
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65

AdaBoost for Car and Pedestrian Detection

- F. Moutarde, B. Stanciulescu, and A. Breheret. *Real-time visual detection of vehicles and pedestrians with new efficient AdaBoost features*. 2008.
- They define a different pixel level "connected control point" feature.

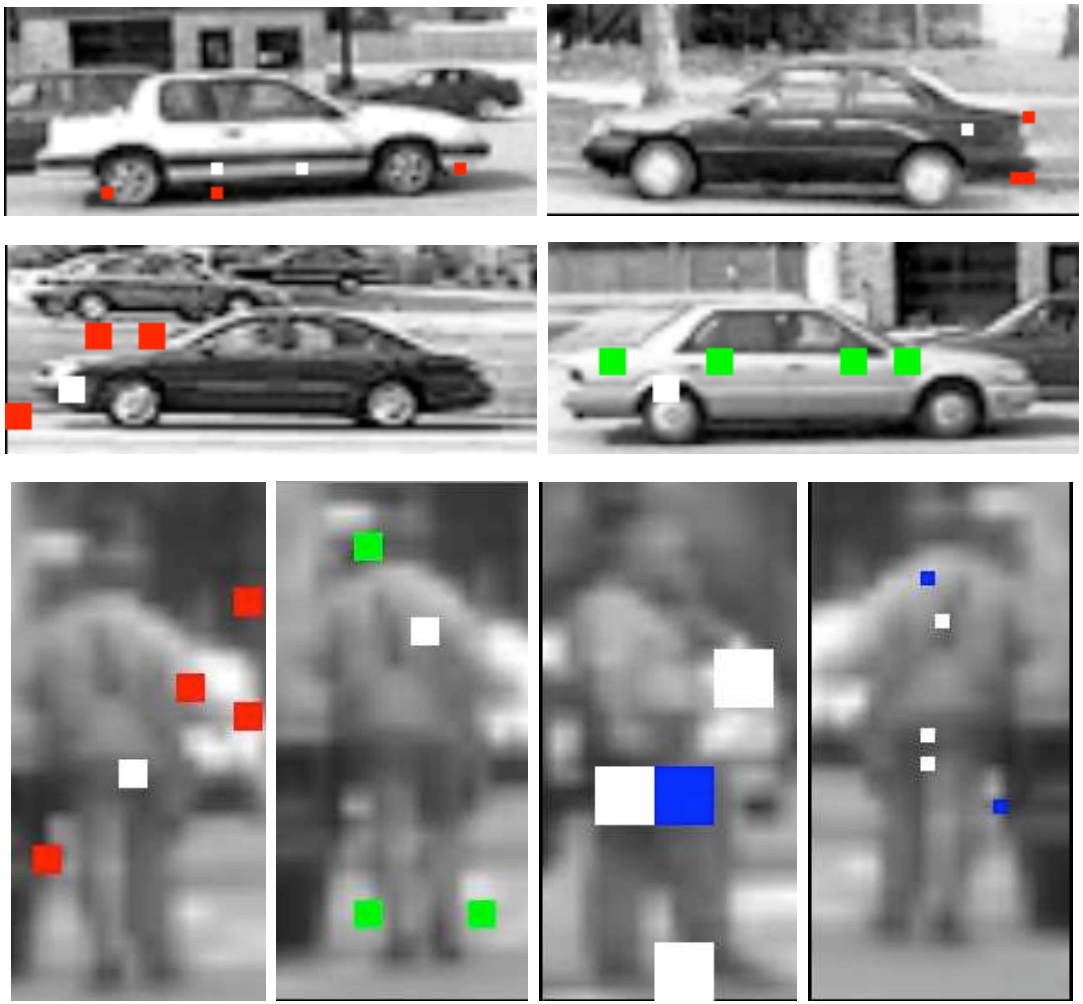
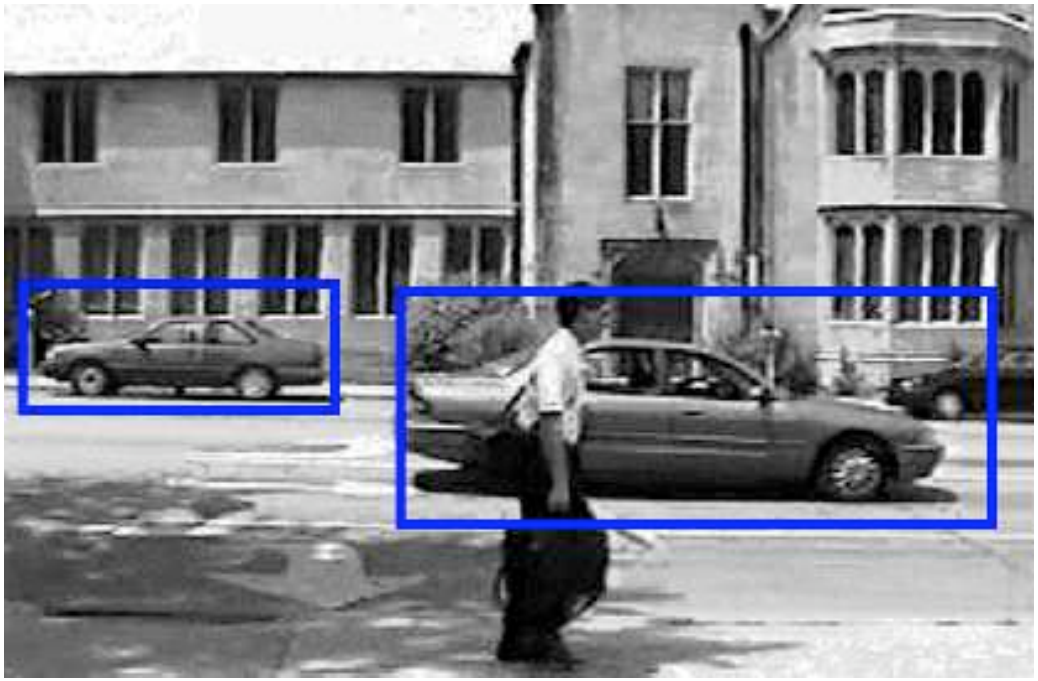
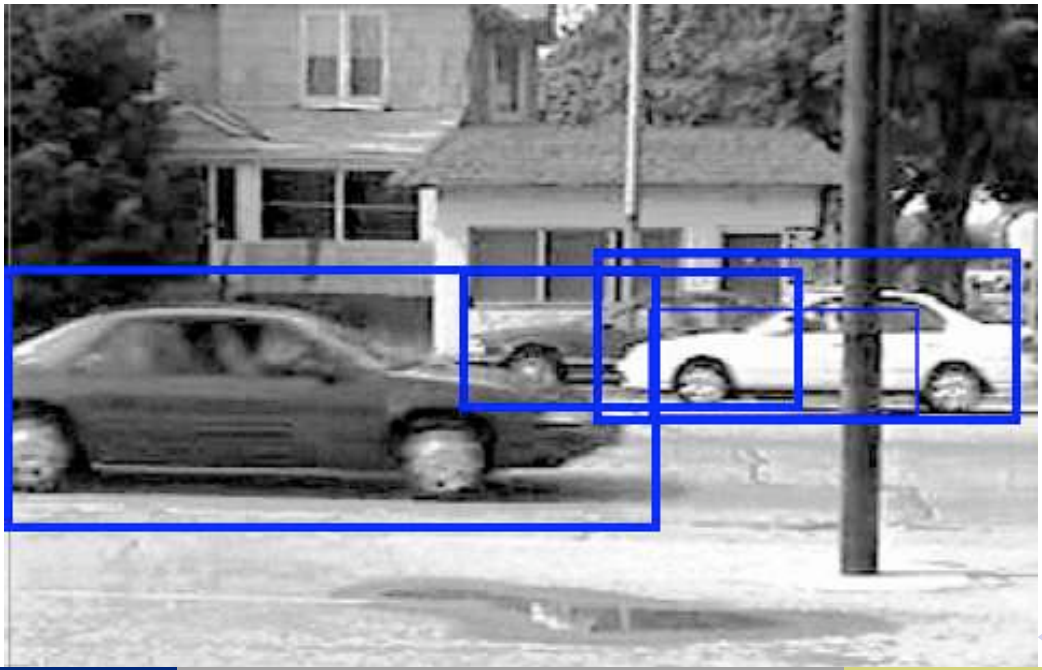


Fig. 4: Some examples of adaBoost-selected Control-Points features for car detection (top) and pedestrian detection (bottom line). Some features operate at full resolution of detection window (eg rightmost bottom), while others work on half-resolution (eg leftmost bottom), or even at quarter-resolution (third on bottom line).





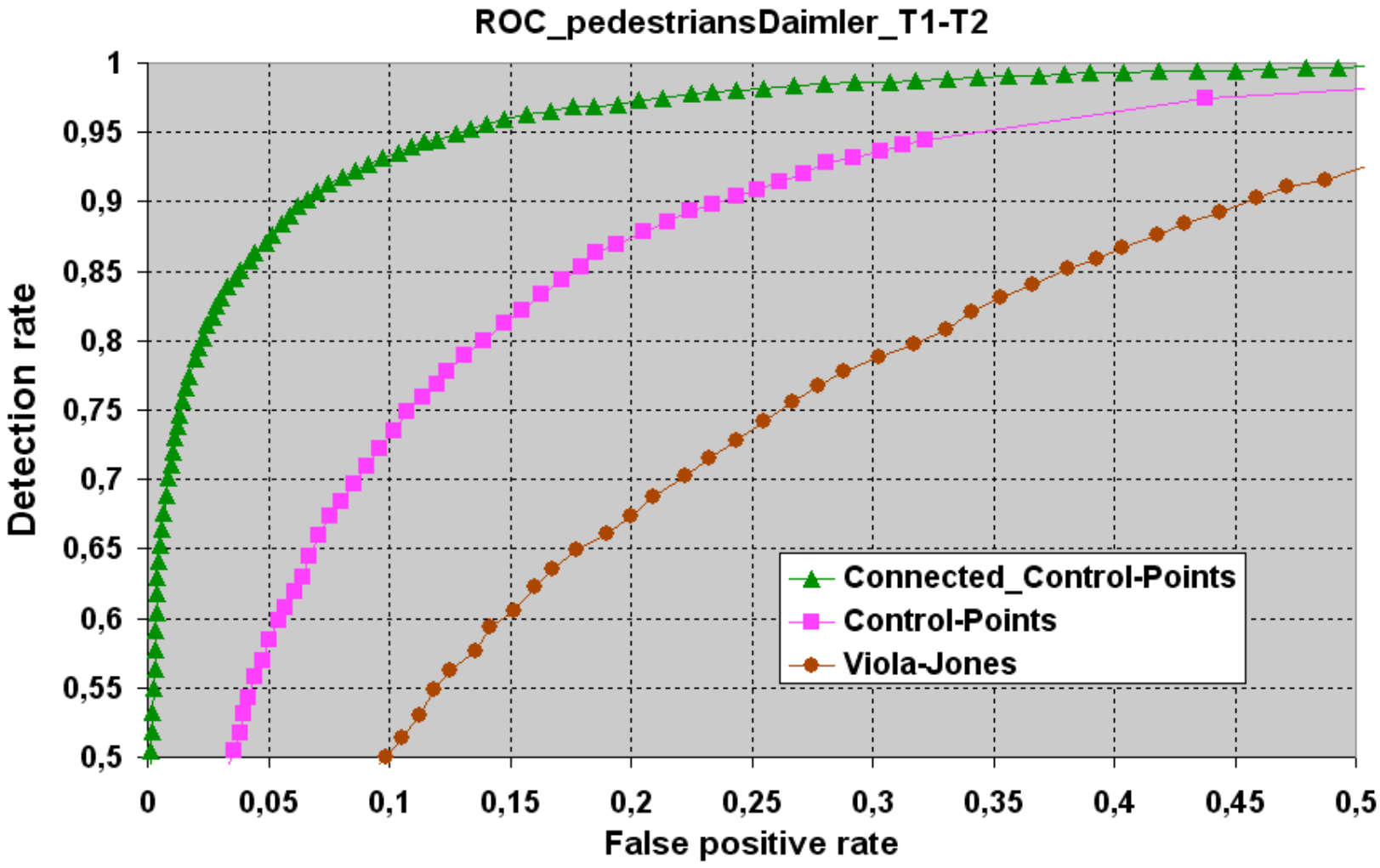


Fig. 9: Averaged ROC curves for adaBoost pedestrian classifiers obtained with various feature families

Sources

These slides have made extensive use of the following sources.

- Y.Freund and R.E. Schapire. A Decision-Theoretic Generalization of On-line Learning and an Application to Boosting. Journal of Computer and System Science, 55(1):119139, 1997.
- R.E. Schapire. The boosting approach to machine learning: an overview. In MSRI Workshop on Nonlinear Estimation and Classification, 2002.
- Schapire's NIPS Tutorial `http://nips.cc/Conferences/2007/Program/schedule.php?Session=Tutorials`
- P.Viola and M.Jones. Rapid object detection using a boosted cascade of simple features. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2001.
- P.Viola and M.Jones. Fast and Robust Classification Using Asymmetric AdaBoost and a Detector Cascade. In Proceedings of Neural Information Processing Systems (NIPS), 2002.
- SC Zhu's slides for AdaBoost (UCLA).