Face Detection via AdaBoost

Linda Shapiro

CSE 455
What’s Coming

1. Review of the basic AdaBoost algorithm (next)
2. The Viola Jones face detector features
3. The modified AdaBoost algorithm that is used in Viola-Jones face detection
4. HW 4
Learning from weighted data

Consider a weighted dataset

- $W_i$ – weight of $i$th training example $(x^i, y^i)$
- Interpretations:
  - $i$th training example counts as if it occurred $W_i$ % times
  - If I were to “resample” data, I would get more samples of “heavier” data points

Now, always do weighted calculations:
Basic AdaBoost Overview

- **Input** is a set of training examples \((X_i, y_i)\) \(i = 1\) to \(m\).
- We train a **sequence of weak classifiers**, such as decision trees, neural nets or SVMs. Weak because not as strong as the final classifier.
- The training examples will have **weights**, initially all equal.
- At each step, we use the current weights, train a new classifier, and use its performance on the training data to produce new weights for the next step (normalized).
- But we **keep ALL** the weak classifiers.
- When it’s time for testing on a new feature vector, we will combine the results from all of the weak classifiers.
Idea of Boosting
(from AI text)
Face detection

State-of-the-art face detection demo
(Courtesy Boris Babenko)
Face detection and recognition

Detection

Recognition

“Sally”
Face detection

Where are the faces?
Face Detection

What kind of features?

- Rowley: 32 x 32 subimages at different levels of a pyramid
- Viola/Jones: rectangular features

What kind of classifiers?

- Rowley: neural nets
- Viola/Jones: Adaboost over simple one-node decision trees (stumps)
Image Features

“Rectangle filters”

People call them Haar-like features, since similar to 2D Haar wavelets.

Value =

\[ \sum \text{(pixels in white area)} - \sum \text{(pixels in black area)} \]
Feature extraction

“Rectangular” filters

Feature output is difference between adjacent regions

Efficiently computable with integral image: any sum can be computed in constant time

Avoid scaling images scale features directly for same cost

Viola & Jones, CVPR 2001
Recall: Sums of rectangular regions

How do we compute the sum of the pixels in the red box?

After some pre-computation, this can be done in constant time for any box.

This “trick” is commonly used for computing Haar wavelets (a fundamental building block of many object recognition approaches.)
Sums of rectangular regions

The trick is to compute an “integral image.” Every pixel is the sum of its neighbors to the upper left.

Sequentially compute using:

\[ I(x, y) = I(x, y) + I(x - 1, y) + I(x, y - 1) - I(x - 1, y - 1) \]
Sums of rectangular regions

Solution is found using:

$$A + D - B - C$$

What if the position of the box lies between pixels?

Use bilinear interpolation.
Large library of filters

Considering all possible filter parameters: position, scale, and type:

160,000+ possible features associated with each 24 x 24 window

Use AdaBoost both to select the informative features and to form the classifier

Viola & Jones, CVPR 2001
Feature selection

• For a 24x24 detection region, the number of possible rectangle features is ~160,000!

• At test time, it is impractical to evaluate the entire feature set

• Can we create a good classifier using just a small subset of all possible features?

• How to select such a subset?
AdaBoost for feature+classifier selection

Want to select the single rectangle feature and threshold that best separates positive (faces) and negative (non-faces) training examples, in terms of weighted error.

\[ \theta_t \text{ is a threshold for classifier } h_t \]

Resulting weak classifier:

\[
h_t(x) = \begin{cases} 
+1 & \text{if } f_t(x) > \theta_t \\
0 & \text{otherwise}
\end{cases}
\]

For next round, reweight the examples according to errors, choose another filter/threshold combo.

Viola & Jones, CVPR 2001
Weak Classifiers

• Each weak classifier works on exactly one rectangle feature.
• Each weak classifier has 3 associated variables
  1. its threshold $\theta$
  2. its polarity $p$
  3. its weight $\alpha$
• The polarity can be 0 or 1 (in our code)
• The weak classifier computes its one feature $f$
  • When the polarity is 1, we want $f > \theta$ for face
  • When the polarity is 0, we want $f < \theta$ for face
• The weight will be used in the final classification by AdaBoost.
AdaBoost: Intuition

Consider a 2-d feature space with positive and negative examples.

Each weak classifier splits the training examples with at least 50% accuracy.

Examples misclassified by a previous weak learner are given more emphasis at future rounds.

Figure adapted from Freund and Schapire
AdaBoost: Intuition

Weighted Example Set

Weak Classifier 1

Weights Increased

Weak Classifier 2

K. Grauman, B. Leibe
AdaBoost: Intuition

Final classifier is combination of the weak classifiers
The final strong classifier is:

\[ h(x) = \begin{cases} 
1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\
0 & \text{otherwise}
\end{cases} \]

where \( \alpha_t = \log \frac{1}{\beta_t} \)

\[ \beta_t = \frac{\epsilon_t}{1 - \epsilon_t} \]: the training error of the classifier \( h_t \)

- Final classifier is combination of the weak ones, weighted according to error they had.
- The code computes a SCORE based on the difference of the two above summations.
- I’m going to go over these things again in the assignment slides.
AdaBoost Algorithm modified by Viola Jones

- Given example images \((x_1, y_1), \ldots, (x_n, y_n)\) where \(y_i = 0, 1\) for negative and positive examples respectively.
- Initialize weights \(w_{1,i} = \frac{1}{2m}, \frac{1}{2l}\) for \(y_i = 0, 1\) respectively, where \(m\) and \(l\) are the number of negatives and positives respectively.
- For \(t = 1, \ldots, T\):
  1. Normalize the weights,
     \[
     w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}
     \]
     so that \(w_t\) is a probability distribution.
  2. For each feature, \(j\), train a classifier \(h_j\) which is restricted to using a single feature. The error is evaluated with respect to \(w_t\), \(e_j = \sum_i w_i |h_j(x_i) - y_i|\). sum over training samples
  3. Choose the classifier, \(h_t\), with the lowest error \(\epsilon_t\).
  4. Update the weights:
     \[
     w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}
     \]
     where \(e_i = 0\) if example \(x_i\) is classified correctly, \(e_i = 1\) otherwise, and \(\beta_t = \frac{\epsilon_t}{1-\epsilon_t}\).

\[\{x_1, \ldots, x_n\}\]

For \(T\) rounds: meaning we will construct \(T\) weak classifiers

NOTE: Our code uses equal weights for all samples

Normalize weights

Find the best threshold and polarity for each feature, and return error \(\epsilon\).

Re-weight the examples:
Incorrectly classified -> more weight
Correctly classified -> less weight
Recall

- Classification
  - Decision Trees and Forests
  - Neural Nets
  - SVMs
  - Boosting
  - ....

- Face Detection
  - Simple Features
  - Integral Images
  - Boosting
Picking the threshold for the best classifier: the idea

Efficient single pass approach:

At each sample compute:

\[
\epsilon = \min \left( S + (T - S), \ S + (T - S) \right)
\]

Find the minimum value of \( \epsilon \), and use the value of the corresponding sample as the threshold.

\( S = \text{sum of weights of samples with feature value below the current sample} \)
\( T = \text{total sum of all samples} \)
\( S \) and \( T \) are for faces; \( S \) and \( T \) are for background.

The features are actually sorted in the code according to numeric value!
Picking the threshold for the best classifier: the details for coding

The features for the training samples are actually sorted in the code according to numeric value!

Algorithm:
1. find $AFS$, the sum of the weights of all the face samples
2. find $ABG$, the sum of the weights of all the background samples
3. set to zero $FS$, the sum of the weights of face samples so far
4. set to zero $BG$, the sum of the weights of background samples so far
5. go through each sample $s$ in a loop IN THE SORTED ORDER

At each sample, add weight to $FS$ or $BG$ and compute:

$$e = \min (BG + (AFS - FS), FS + (ABG - BG))$$

Find the minimum value of $e$, and use the feature value of the corresponding sample as the threshold.
What’s going on?

\[
\text{error} = \min \left( \frac{BG + (AFS - FS)}{\text{left}}, \frac{FS + (ABG - BG)}{\text{right}} \right)
\]

- Let’s pretend the weights on the samples are all 1’s.
- The samples are arranged in ascending order by feature value and we know which ones are faces (f) and background (b).
- **Left** is the number of background patches so far plus the number of faces yet to be encountered.
- **Right** is the number of faces so far plus the number of background patches yet to be encountered.

\[
\begin{align*}
1+5-0 &\quad b & 0+5-1 &\quad b \\
(6,4) &\quad (7,3) &\quad (8,2) &\quad (7,3) \\
&\quad (8,2) &\quad (7,3) &\quad (4,4) \\
&\quad (7,3) &\quad (6,4) &\quad (5,5) \\
\text{min} &\quad 4 &\quad 3 &\quad 2 &\quad 3 &\quad 4 &\quad 3 &\quad 4 &\quad 5 &\quad 28
\end{align*}
\]
Measuring classification performance

- Confusion matrix

- Accuracy
  - \( \frac{(TP+TN)}{(TP+TN+FP+FN)} \)

- True Positive Rate = Recall
  - \( \frac{TP}{(TP+FN)} \)

- False Positive Rate
  - \( \frac{FP}{(FP+TN)} \)

- Precision
  - \( \frac{TP}{(TP+FP)} \)

- F1 Score
  - \( \frac{2 \times \text{Recall} \times \text{Precision}}{(\text{Recall} + \text{Precision})} \)

<table>
<thead>
<tr>
<th>Actual class</th>
<th>Class1</th>
<th>Class2</th>
<th>Class3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class1</td>
<td>40</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Class2</td>
<td>3</td>
<td>25</td>
<td>7</td>
</tr>
<tr>
<td>Class3</td>
<td>4</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual</td>
<td>True Positive</td>
<td>False Negative</td>
</tr>
<tr>
<td>Positive</td>
<td>True Positive</td>
<td>True Negative</td>
</tr>
<tr>
<td>Negative</td>
<td>False Positive</td>
<td>True Negative</td>
</tr>
</tbody>
</table>
Boosting for face detection

- First two features selected by boosting:

This feature combination can yield 100% detection rate and 50% false positive rate.
Boosting for face detection

• A 200-feature classifier can yield 95% detection rate and a false positive rate of 1 in 14084

Is this good enough?

Receiver operating characteristic (ROC) curve
Attentional cascade (from Viola-Jones)

This part will be extra credit for HW4

- We start with simple classifiers which reject many of the negative sub-windows while detecting almost all positive sub-windows.
- Positive response from the first classifier triggers the evaluation of a second (more complex) classifier, and so on.
- A negative outcome at any point leads to the immediate rejection of the sub-window.
Attentional cascade

• Chain of classifiers that are progressively more complex and have lower false positive rates:

![Diagram of a cascade of classifiers with decision trees for face detection.]
Attentional cascade

• The detection rate and the false positive rate of the cascade are found by multiplying the respective rates of the individual stages

• A detection rate of 0.9 and a false positive rate on the order of $10^{-6}$ can be achieved by a 10-stage cascade if each stage has a detection rate of 0.99 ($0.99^{10} \approx 0.9$) and a false positive rate of about 0.30 ($0.3^{10} \approx 6 \times 10^{-6}$)
Training the cascade

• Set target detection and false positive rates for each stage
• Keep adding features to the current stage until its target rates have been met
  • Need to lower AdaBoost threshold to maximize detection (as opposed to minimizing total classification error)
  • Test on a validation set
• If the overall false positive rate is not low enough, then add another stage
• Use false positives from current stage as the negative training examples for the next stage
Viola-Jones Face Detector: Summary

Train with 5K positives, 350M negatives
Real-time detector using 38 layer cascade
6061 features in final layer

[Implementation available in OpenCV: http://www.intel.com/technology/computing/opencv/]
The implemented system

• Training Data
  • 5000 faces
    – All frontal, rescaled to 24x24 pixels
  • 300 million non-faces
    – 9500 non-face images
  • Faces are normalized
    – Scale, translation

• Many variations
  • Across individuals
  • Illumination
  • Pose
System performance

• Training time: “weeks” on 466 MHz Sun workstation
• 38 layers, total of 6061 features
• Average of 10 features evaluated per window on test set
• “On a 700 Mhz Pentium III processor, the face detector can process a 384 by 288 pixel image in about .067 seconds”
  • 15 Hz
  • 15 times faster than previous detector of comparable accuracy (Rowley et al., 1998)
Non-maximal suppression (NMS)

Many detections above threshold.
Non-maximal suppression (NMS)
Similar accuracy, but 10x faster

Is this good?
Viola-Jones Face Detector: Results
Viola-Jones Face Detector: Results
Viola-Jones Face Detector: Results
Detecting profile faces?

Detecting profile faces requires training separate detector with profile examples.
Viola-Jones Face Detector: Results
Summary: Viola/Jones detector

- Rectangle features
- Integral images for fast computation
- Boosting for feature selection
- Attentional cascade for fast rejection of negative windows