Learning and Vision: Discriminative Models

Paul Viola
Microsoft Research
viola@microsoft.com

Overview

- Perceptrons
- Support Vector Machines
  - Face and pedestrian detection
- AdaBoost
  - Faces
- Building Fast Classifiers
  - Trading off speed for accuracy...
  - Face and object detection
- Memory Based Learning
  - Simard
  - Moghaddam

Historical Lesson

- 1950’s: Perceptrons are cool
  - Very simple learning rule, can learn “complex” concepts
  - Generalized perceptrons are better – too many weights
- 1960’s: Perceptron’s stink (M+P)
  - Some simple concepts require exponential # of features
    - Can’t possibly learn that, right?
- 1980’s: MLP’s are cool (R+M / PDP)
  - Sort of simple learning rule, can learn anything (?)
  - Create just the features you need
- 1990: MLP’s stink
  - Hard to train: Slow / Local Minima
- 1996: Perceptron’s are cool

Why did we need multi-layer perceptrons?

- Problems like this seem to require very complex non-linearities.
- Minsky and Papert showed that an exponential number of features is necessary to solve generic problems.

Why an exponential number of features?

\[
\Phi(x) = \begin{cases} 
1, \\
 x_1, x_2, \\
 x_1^2, x_1 x_2, x_2^2, \\
 x_1^3, x_1^2 x_2, x_1 x_2^2, x_2^3, \\
\vdots 
\end{cases}
\]

- 14th Order???
- 120 Features

\[
\begin{align*}
\text{polyorder} : & n \text{ variables} \\
\text{order poly} : & k \text{ monomials} \\
\frac{(n+k)!}{N!} \frac{(n+k)!}{k!} \Rightarrow 65,000 \text{ features}
\end{align*}
\]

MLP’s vs. Perceptron

- MLP’s are hard to train…
  - Takes a long time (unpredictably long)
  - Can converge to poor minima
- MLP are hard to understand
  - What are they really doing?
- Perceptrons are easy to train…
  - Type of linear programming. Polynomial time.
  - One minimum which is global.
- Generalized perceptrons are easier to understand.
  - Polynomial functions.
Perceptron Training is Linear Programming

\[ y_i (w^T x_i) > 0 \]

Polynomial time in the number of variables and in the number of constraints.

What about linearly inseparable?

\[ y_i (w^T x_i) + s_i > 0 \]

\[ s_i > 0 \quad \forall i \]

Rebirth of Perceptrons

- How to train effectively
  - Linear Programming (… later quadratic programming)
  - Though on-line works great too.
- How to get so many features inexpensively?!!
  - Kernel Trick
- How to generalize with so many features?
  - VC dimension. (Or is it regularization?)

Support Vector Machines

Lemma 1: Weight vectors are simple

\[ w_0 = 0 \quad \Delta w_i = \eta_i x_i \]

\[ w_i = \sum_{i \in \ell} \eta_i x_i = \sum_{i} b_i x_i \quad w_i = \sum_{i} b_i \Phi(x_i) \]

- The weight vector lives in a sub-space spanned by the examples…
  - Dimensionality is determined by the number of examples not the complexity of the space.

Lemma 2: Only need to compare examples

\[ w_i = \sum_{i} b_i \Phi(x_i) \]

\[ y(x) = w^T \Phi(x) \]

\[ = \left( \sum_{i} b_i \Phi(x_i) \right)^T \Phi(x) \]

\[ = \sum_{i} b_i K(x_i, x) \]

Simple Kernels yield Complex Features

\[ K(x, x') = (1 + x^T x')^2 \]

\[ = (1 + x_1 x'_1 + x_2 x'_2)^2 \]

\[ = 1 + x_1^2 x'_1^2 + x_2^2 x'_2^2 + 2 x_1 x'_1 x_2 x'_2 \]

\[ \Phi(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1 x_2 \end{bmatrix} \]

But Kernel Perceptrons Can Generalize Poorly

\[ K(x, x') = (1 + x^T x')^{14} \]
**Perceptron Rebirth: Generalization**

- Too many features … Occam is unhappy
  - Perhaps we should encourage smoothness?

\[ \forall i: \ y_i \sum_j b_j K(x_j, x_i) + s_i > 0 \quad \min \sum_i s_i \]

\[ \forall i: \ s_i > 0 \quad \min \sum_j b_j^2 \]

**Linear Program is not unique**

The linear program can return any multiple of the correct weight vector...

\[ \forall i: \ y_i (w^T x_i) > 0 \Rightarrow \forall i: \ y_i (\lambda w)^T x_i > 0 \]

Slack variables & Weight prior
- Force the solution toward zero

**Definition of the Margin**

- Geometric Margin: Gap between negatives and positives measured perpendicular to a hyperplane

- Classifier Margin

\[ \min_{i \in \text{POS}} (w^T x_i) - \max_{i \in \text{NEG}} (w^T x_i) \]

**Require non-zero margin**

- Allows solutions with zero margin

- Enforces a non-zero margin between examples and the decision boundary.

\[ w^T x_i + s_i > 0 \quad \forall i \]

\[ w^T x_i + s_i > 1 \quad \forall i \]

**Constrained Optimization**

\[ y_i \sum_j b_j K(x_j, x_i) + s_i > 1 \quad \min \sum_i s_i \]

\[ s_i > 0 \quad \min \sum_j b_j^2 \]

- Find the smoothest function that separates data
  - Quadratic Programming (similar to Linear Programming)
    - Single Minima
    - Polynomial Time algorithm

**Constrained Optimization 2**

\[ w^T x_i = 1 \]

\[ w = \alpha x_i \beta i \]

\[ x^3 \text{ is inactive} \]
**SVM: Key Ideas**

- Augment inputs with a very large feature set
  - Polynomials, etc.
- Use Kernel Trick(TM) to do this efficiently
- Enforce/Encourage Smoothness with weight penalty
- Introduce Margin
- Find best solution using Quadratic Programming

---

**SVM: Zip Code recognition**

- Data dimension: 256
- Feature Space: 4th order
  - roughly 100,000,000 dims

---

**The Classical Face Detection Process**

- Classifier is Learned from Labeled Data
  - Training Data
    - 5000 faces
      - All frontal
    - $10^6$ non faces
    - Faces are normalized
      - Scale, translation
  - Many variations
    - Across individuals
    - Illumination
    - Pose (rotation both in plane and out)

**Key Properties of Face Detection**

- Each image contains 10-50 thousand locs/scales
- Faces are rare 0-50 per image
  - 1000 times as many non-faces as faces
- Extremely small # of false positives: $10^{-6}$
On to AdaBoost

- Given a set of weak classifiers
  - Originally: \( h_j(x) \in \{+1, -1\} \)
  - Also: \( h_j(x) \in \{\alpha, \beta\} \) "confidence rated"
  - None much better than random
- Iteratively combine classifiers
  - Form a linear combination
  \[
  C(x) = \theta \left( \sum h_j(x) + b \right)
  \]
  - Training error converges to 0 quickly
  - Test error is related to training margin
**AdaBoost**

\[ h_t = \min_h \sum_i D_i(i) e^{-y_i h(x_i)} / Z_i \]

\[ \{\alpha, \beta\} = \frac{1}{2} \log \left( \frac{W_0}{W_0} \right) \]

\[ D_{i+1}(i) = \frac{D_i(i) e^{-y_i h(i)} / Z_i}{Z_i} \]

*Weights Increased*

Weak classifier 3

Final classifier is linear combination of weak classifiers

**AdaBoost Properties**

\[ D_{i+1}(i) = \frac{D_i(i) e^{-y_i h(i)} / Z_i}{Z_i} = \prod_t e^{-y_i h(i)} / Z_i = \prod Z_i \]

\[ \prod Z_i = \sum e^{-\gamma \sum h(i)} e^{-\beta \sum h(i)} \geq \text{Loss}(y_i, C(x_i)) \]

**Boosted Face Detection: Image Features**

```
“Rectangle filters”
Similar to Haar wavelets

\( h(x) = \begin{cases} 
\alpha & \text{if } f_i(x) > \theta_i \\
\beta & \text{otherwise} 
\end{cases} \)

\( C(x) = \theta \sum h(x) + b \)
```

60,000 x 100 = 6,000,000

Unique Binary Features

**AdaBoost: Super Efficient Feature Selector**

- Features = Weak Classifiers
- Each round selects the optimal feature given:
  - Previous selected features
  - Exponential Loss
Feature Selection

- For each round of boosting:
  - Evaluate each rectangle filter on each example
  - Sort examples by filter values
  - Select best threshold for each filter (min $Z$)
  - Select best filter/threshold (= Feature)
  - Reweight examples
- $M$ filters, $T$ thresholds, $N$ examples, $L$ learning time
- $O(MT L(MTN))$ Naive Wrapper Method
- $O(MN)$ AdaBoost feature selector

Example Classifier for Face Detection

A classifier with 200 rectangle features was learned using AdaBoost. 95% correct detection on test set with 1 in 14084 false positives. Not quite competitive...

Building Fast Classifiers

- Given a nested set of classifier hypothesis classes
- Computational Risk Minimization

Other Fast Classification Work

- Simard
- Rowley (Faces)
- Fleuret & Geman (Faces)

Cascaded Classifier

- A 1 feature classifier achieves 100% detection rate and about 50% false positive rate.
- A 5 feature classifier achieves 100% detection rate and 40% false positive rate (20% cumulative) -- using data from previous stage.
- A 20 feature classifier achieve 100% detection rate with 10% false positive rate (2% cumulative)

Comparison to Other Systems

<table>
<thead>
<tr>
<th>Detector</th>
<th>False Detections</th>
<th>False Detections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viola-Jones</td>
<td>76.3</td>
<td>85.2</td>
</tr>
<tr>
<td>Rowley-Baluja-Kanade</td>
<td>85.2</td>
<td>86.0</td>
</tr>
<tr>
<td>Schneiderman-Kanade</td>
<td>94.0</td>
<td>94.0</td>
</tr>
<tr>
<td>Roth-Yang-Ahuja</td>
<td>(04.8)</td>
<td></td>
</tr>
</tbody>
</table>
**Output of Face Detector on Test Images**

**Solving other “Face” Tasks**

- Facial Feature Localization
- Profile Detection
- Demographic Analysis

**Feature Localization**

- Surprising properties of our framework
  - The cost of detection is not a function of image size
  - Just the number of features
  - Learning automatically focuses attention on key regions
- Conclusion: the “feature” detector can include a large contextual region around the feature

**Feature Localization Features**

- Learned features reflect the task

**Profile Detection**

**More Results**
Profile Features

Features, Features, Features

- In almost every case:
  
  Good Features beat Good Learning
  Learning beats No Learning

- Critical classifier ratio: \( \frac{\text{quality}}{\text{complexity}} \)

- AdaBoost \( \gg \) SVM