Perceptrons, SVMs, and Friends: Some Discriminative Models for Classification

Parallel to AIMA 18.1, 18.2, 18.6.3, 18.9

The Automatic Classification Problem

- Assign object/event or sequence of objects/events to one of a given finite set of categories.
  - Fraud detection for credit card transactions, telephone calls, etc.
  - Worm detection in network packets
  - Spam filtering in email
  - Recommending articles, books, movies, music
  - Medical diagnosis
  - Speech recognition
  - OCR of handwritten letters
  - Recognition of specific astronomical images
  - Recognition of specific DNA sequences
  - Financial investment

- Machine Learning methods provide one set of approaches to this problem

Universal Machine Learning Diagram

Example: handwritten digit recognition

Machine learning algorithms that
- Automatically cluster these images
- Use a training set of labeled images to learn to classify new images
- Discover how to account for variability in writing style

A machine learning algorithm development pipeline: minimization

Universal Machine Learning Diagram

Today: Perceptron, SVM and Friends

Naive Bayes Classifiers are one example
Generative vs. Discriminative Models

- **Generative question:**
  - “How can we model the joint distribution of the classes and the features?”
  
  \[ c_{ML} = \arg \max_{c \in C} P(c \mid D) \]
  
  Bayes’ rule + Assumption that all hypotheses are a priori equally likely

  \[ c_{ML} = \arg \max_{c \in C} P(D \mid c) \]

- Naïve Bayes, Markov Models, HMMs all generative

- **Discriminative question:**
  - “What features distinguish the classes from one another?”

Example

Modeling what sort of bizarre distribution produced these training points is hard, but distinguishing the classes is a piece of cake!

Chart from MIT tech report #507, Tony Jebara

Linear Classification: Informal...

Find a (line, plane, hyperplane) that divides the red points from the blue points....

Hyperplane

A **hyperplane** can be defined by

\[ c = \vec{w} \cdot \vec{x} \]

Or more simply (renormalizing) by

\[ 0 = \vec{w} \cdot \vec{x} \]

Consider a two-dimension example...

\[ 0 = [1, -1] \begin{bmatrix} x \\ y \end{bmatrix} \]

\[ y = y \]

\[ y = x \]

Linear Classification: Slightly more formal

Input encoded as feature vector \( \vec{x} \)

Model encoded as \( \vec{w} \)

Just return \( y = \vec{w} \cdot \vec{x} \)

**sign**\( (y) \) tell us the class:

+ red

- blue

(All vectors normalized to length 1, for simplicity)

Computing the sign...

One definition of dot product:

\[ \vec{w} \cdot \vec{x} = \| \vec{w} \| \| \vec{x} \| \cos \theta \]

So \( \text{sign}(\vec{w} \cdot \vec{x}) = \text{sign}(\cos \theta) \)

Let \( y = \text{sign}(\cos \theta) \)
### Perceptron Update Example

If \( \vec{x}_i \) is supposed to be on the other side:

\[
\vec{W} = \vec{W} + y_i \vec{x}_i
\]

### Perceptron Learning Algorithm

**Input:** A list \( T \) of training examples \( \langle \vec{x}_0, y_0 \rangle \ldots \langle \vec{x}_n, y_n \rangle \) where \( y_i \in \{+1,-1\} \)

**Output:** A classifying hyperplane \( \vec{w} \)

Randomly initialize \( \vec{w} \);

while model \( \vec{w} \) makes errors on the training data do

for \( \langle \vec{x}_i, y_i \rangle \) in \( T \) do

Let \( \hat{y} = \text{sign}(\vec{w} \cdot \vec{x}_i) \);

if \( \hat{y} \neq y_i \) then

\[
\vec{w} = \vec{w} + y_i \vec{x}_i
\]

end

end

Converges if the training set is linearly separable

May not converge if the training set is not linearly separable

### Compared to the biological neuron

- **Input**
  - A neuron’s dendritic tree is connected to a thousand neighboring neurons. When one of those neurons fires, a positive or negative charge is received
  - The strengths of all the received charges are added together ...

- **Output**
  - If the aggregate input is greater than the axon hillock’s threshold value, then the neuron fires
  - The physical and neurochemical characteristics of each synapse determines the strength and polarity of the new signal

### Voted & Averaged Perceptron

- **Voted Perceptron** (Freund & Schapire 1999)
  - Let each of the (many, many) models vote on the answer and take the majority
  - As fast to train but slower in run-time

- **Averaged Perceptron** (Collins 2002)
  - Return as your final model the average of all intermediate models created
  - Nearly as fast to train and exactly as fast to run as regular perceptron

### Properties of the Simple Perceptron

- You can prove that
  - If it’s possible to separate the data with a hyperplane (i.e. if it’s linearly separable),
  - Then the algorithm will converge to that hyperplane.

- But what if it isn’t? Then perceptron is very unstable and oscillates back and forth.

### Support vector machines
What's wrong with these hyperplanes?

They're unjustifiably biased!

A less biased choice

Margin
- the distance to closest point in the training data
- We tend to get better generalization to **unseen** data if we choose the separating hyperplane which **maximizes the margin**

Support Vector Machines
- A learning method which *explicitly calculates the maximum margin hyperplane* by solving a gigantic quadratic programming minimization problem.
- Among the very highest-performing current machine learning techniques.
- But it’s relatively slow and quite complicated.

Maximizing the Margin

Select the separating hyperplane that maximizes the margin
Support Vectors

Support Vector Machines

- A learning method which explicitly calculates the maximum margin hyperplane.

Setting Up the Optimization Problem

- The maximum margin can be characterized as a solution to an optimization problem:

\[
\begin{align*}
\text{max. } & \frac{1}{2} ||w||^2 \\
\text{s.t. } & y_i(w \cdot x_i + b) \geq 1, \forall x_i
\end{align*}
\]

Define the margin (what ever it turns out to be) to be one unit of width.

Linear, (Hard-Margin) SVM Formulation

- Find \(w, b\) that solves

\[
\begin{align*}
\min. & \frac{1}{2} ||w||^2 \\
\text{s.t. } & y_i(w \cdot x_i + b) \geq 1, \forall x_i
\end{align*}
\]

- Problem is convex, so there is a unique global minimum value (when feasible)
- There is also a unique minimizer, i.e. weight and \(b\) value that provides the minimum
- Quadratic Programming
  - very efficient computationally with procedures that take advantage of the special structure

What if it isn't separable?
Project it to someplace where it is!

\[ \phi(x, y) = x^2 + y^2 \]

Non-linear SVMs: Feature spaces

- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is linearly separable:

Kernel Trick

- If our data isn’t linearly separable, we can define a projection \( \Phi(x_i) \) to map it into a much higher dimensional feature space where it is.
- For SVM where everything can be expressed as the dot products of instances this can be done efficiently using the ‘kernel trick’:
  - A kernel \( K \) is a function such that: \( K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j) \)
  - Then, we never need to explicitly map the data into the high-dimensional space to solve the optimization problem – magic!!

Gaussian Kernel: Example

The appropriate \( K \) maps this into a hyperplane in some space!!

SVMs vs. other ML methods

- Examples from the NIST database of handwritten digits
  - 60K labeled digits 20x20 pixels 8-bit grayscale values
- Learning methods
  - 3-nearest neighbors
  - Hidden layer neural net
  - Specialized neural net (LeNet)
  - Boosted neural net
  - SVM
  - SVM with kernels on pairs of nearby pixels + specialized transforms
  - Shape matching (vision technique)
- Human error: on similar US Post Office database 2.5%

Performance on the NIST digit set (2003)

<table>
<thead>
<tr>
<th>Method</th>
<th>Error %</th>
<th>Run time (ms/digit)</th>
<th>Memory (MB)</th>
<th>Training time (days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-NN</td>
<td>2.4</td>
<td>1000</td>
<td>0.49</td>
<td>0</td>
</tr>
<tr>
<td>Hidden Layer NN</td>
<td>1.6</td>
<td>10</td>
<td>0.012</td>
<td>14</td>
</tr>
<tr>
<td>LeNet</td>
<td>0.9</td>
<td>30</td>
<td>0.21</td>
<td>21</td>
</tr>
<tr>
<td>Boosted LeNet</td>
<td>0.7</td>
<td>50</td>
<td>0.21</td>
<td>11</td>
</tr>
<tr>
<td>SVM</td>
<td>1.1</td>
<td>2000</td>
<td>1.0</td>
<td>30</td>
</tr>
<tr>
<td>SVM + kernels</td>
<td>0.7</td>
<td>2000</td>
<td>1.0</td>
<td>10</td>
</tr>
<tr>
<td>Shape Match</td>
<td>0.56</td>
<td>2000</td>
<td>0.59</td>
<td>10</td>
</tr>
<tr>
<td>Shape Match</td>
<td>0.63</td>
<td>2000</td>
<td>0.63</td>
<td>10</td>
</tr>
</tbody>
</table>

Recently beaten (2010) (.35% error) by a very complex neural network (if you want details: a 6 layer NN with 784-2500-2000-1500-1000-500-10 topology with elastic distortions running on modern GPU)