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

Given training vectors $x_1, \ldots, x_N$ and targets $t_1, \ldots, t_N$, find...

Universal Machine Learning Diagram

Today: Perceptron, SVM and Friends
Generative vs. Discriminative Models

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

  \[ C_{\text{MLE}} = \arg \max_{\theta \in \mathcal{C}} P(D \mid c) \]

- 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...

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

\[ 0 = x - y \]

\[ y = x \]

Linear Classification: Slightly more formal

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

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

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

\( \text{sign}(y) \) tell us the class:

\[ + \rightarrow \text{blue} \]

\[ - \rightarrow \text{red} \]

(All vectors normalized to length 1, for simplicity)

Computing the sign...

One definition of dot product:

\[ W \cdot X = \|W\| \|X\| \cos \theta \]

So \( \text{sign}(W \cdot X) = \text{sign}(\cos \theta) \)

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

If \( \mathbf{w} \) is supposed to be on the other side….

\[
\mathbf{w} = \mathbf{w} + y_i x_i
\]

Perceptron Learning Algorithm

Input: A list \( T \) of training examples \((x_0, y_0), \ldots, (x_n, y_n)\) where \( y_i \in \{+1,-1\} \)
Output: A classifying hyperplane \( \mathbf{w} \)

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

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

for \((x_i, y_i) \) in \( T \) do

Let \( \hat{y} = \text{sign}(\mathbf{w} \cdot x_i) \);

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

\[
\mathbf{w} = \mathbf{w} + y_i x_i
\]

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

\[
 f(x) = \begin{cases} 
 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \\
 0 & \text{else} 
\end{cases}
\]

Voted & Averaged Perceptron

--Works just like a regular perceptron, except keeping track of all the intermediate models created

--Much better generalization performance than regular perceptron (almost as good as SVMs)

- 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
  - 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
Support Vectors

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

Support Vector Machines

- If class 1 corresponds to 1 and class 2 corresponds to -1, we can rewrite
  \((w \cdot x + b) \geq 1, \forall x_i\) with \(y_i = 1\)
  \((w \cdot x + b) \leq -1, \forall x_i\) with \(y_i = -1\)
  as
  \(y_i(w \cdot x + b) \geq 1, \forall x_i\)

- So the problem becomes:
  \(\max \frac{1}{2} \|w\|^2 \quad \text{or} \quad \min \frac{1}{2} \|w\|^2\)
  \(s.t. \ y_i(w \cdot x + b) \geq 1, \forall x_i\)

Setting Up the Optimization Problem

- 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
  \(\min \frac{1}{2} \|w\|^2\)
  \(s.t. \ y_i(w \cdot x_i + b) \geq 1, \forall x_i\)
- 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:
  \[ \Phi: x \rightarrow \phi(x) \]

Kernel Trick

- If our data isn’t linearly separable, we can define a projection \( \Phi(x) \) 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></th>
<th>3-NN</th>
<th>Hidden Layer NN</th>
<th>LeNet</th>
<th>Boosted LeNet</th>
<th>SVM</th>
<th>SVM with kernels</th>
<th>Shape Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error %</td>
<td>3.4</td>
<td>1.6</td>
<td>0.9</td>
<td>0.7</td>
<td>1.1</td>
<td>0.56</td>
<td>0.63</td>
</tr>
<tr>
<td>Run time (milliseconds/digit)</td>
<td>1000</td>
<td>10</td>
<td>30</td>
<td>50</td>
<td>2000</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>Memory (MB)</td>
<td>12</td>
<td>.49</td>
<td>.013</td>
<td>.21</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Training time (days)</td>
<td>0</td>
<td>7</td>
<td>14</td>
<td>30</td>
<td>10</td>
<td></td>
<td></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)