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 a powerful 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

Problem statement

Mathematical description of a cost function

Mathematical description of how to minimize/maximize the cost function

Implementation
Generative vs. Discriminative Models

- **Generative question:**
  - “How can we model the joint distribution of the classes and the features?”
  - 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 \( c = \vec{w} \cdot \vec{x} \)

where \( \vec{w} \) is the normal to the hyperplane

Or more simply (renormalizing) by \( 0 = \vec{w} \cdot \vec{x} \)

Consider a two-dimensional 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:

+ - blue
- - red

(Vectors normalized to length 1, and we assume that the hyperplane passes through 0,0)

---

**Computing the sign…**

One definition of dot product: \( W \cdot X = ||W|| ||X|| \cos \theta \)

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

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

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

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

Perceptron Learning Algorithm

Input: A list \( T \) of training examples \( \langle \mathbf{x}_0, y_0 \rangle \ldots \langle \mathbf{x}_n, y_n \rangle \) where

\[ y : y_i \in \{+1,-1\} \]

Output: A classifying hyperplane \( \mathbf{w} \)

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

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

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

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

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

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

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 created in training 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 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 large 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

Margin Width

Margin Width

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:

\[
\text{max. } \sum_{i=1}^{2} \left( y_i (w \cdot x_i + b) \right) \text{ s.t. } y_i (w \cdot x_i + b) \geq 1, \forall \{i \}
\]

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

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

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!

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

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 \( \Phi(x) \) 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 8bit grayscale values
- Learning methods
  - 3-nearest neighbors
  - Hidden layer 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>Kernel SVM</th>
<th>Shape Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error %</td>
<td>2.4</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>300</td>
<td></td>
</tr>
<tr>
<td>Memory (MB)</td>
<td>12</td>
<td>49</td>
<td>612</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>

Beaten in 2010 (.35% error) by a very complex deep 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)
Tricks to take home

- 'kernel trick': If our data isn't linearly separable, we can define a projection to map it into a much higher dimensional feature space where it is.

- If you somehow figure out ....
  - Map the problem to THE feature space

Margin-Infused Relaxed Algorithm (MIRA)

- Multiclass; each class has a prototype vector
- Classify an instance by choosing the class whose prototype vector has the greatest dot product with the instance
- During training, when updating make the 'smallest' (in a sense) change to the prototype vectors which guarantees correct classification by a minimum margin
- Pays attention to the margin directly