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: Computational, statistical approaches to this problem
  - Naïve Bayes just one such approach

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
Generative vs. Discriminative Models

- **Generative question:**
  - “How can we model the joint distribution of the classes and the features?”

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

  

- **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

Hyperplane

A hyperplane can be defined by $c = \mathbf{w} \cdot \mathbf{x}$

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

Consider a two-dimension example...

$0 = \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

$0 = x - y$

$y = x$

Linear Classification: Informal...

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

Linear Classification: More formal

Input encoded as feature vector $\mathbf{x}$

Model encoded as $\mathbf{w}$

Just return $y = \mathbf{w} \cdot \mathbf{x}$

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

- + - blue
- - - red

(All vectors normalized to length 1, for simplicity)

Computing the sign...

One definition of dot product: $\mathbf{W} \cdot \mathbf{X} = \|W\| \|X\| \cos \theta$

So $\text{sign}(\mathbf{W} \cdot \mathbf{X}) = \text{sign}(\cos \theta)$
Perceptron Update Example

\[ w \] is supposed to be on the other side....

Perceptron Learning Algorithm

Input: A list \( T \) of training examples \( (x_0, y_0) \ldots (x_n, y_n) \) where \( y \in \{+1, -1\} \)
Output: A classifying hyperplane \( w \)
Randomly initialize \( w \);
while model \( w \) makes errors on the training data do
  for \( (x_i, y_i) \) in \( T \) do
    if \( y \neq y_i \) then
      \[ w = w + y_i x_i \]
  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.

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

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

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 Vector Machines

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

Support Vectors

```
CIS 391 - Intro to AI
```

Setting Up the Optimization Problem

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

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

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

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

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:

\[ \Phi : x \rightarrow \phi(x) \]

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 8bit greyscale 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>Error %</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></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 (msec/digit)</td>
<td>1000</td>
<td>10</td>
<td>30</td>
<td>50</td>
<td>200</td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td>Memory (MB)</td>
<td>12</td>
<td>4.9</td>
<td>0.12</td>
<td>0.21</td>
<td>11</td>
<td>30</td>
<td>10</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)