More Learning

- Ensembles
- Bayes Rule
- Neural Nets
- K-means Clustering
- EM Clustering
- WEKA
Ensembles

• An ensemble is a set of classifiers whose combined results give the final decision.
Bagging

- Generate “bootstrap” replicates of training set by sampling with replacement
- Learn one model on each replicate
- Combine by uniform voting
Boosting

- Maintain vector of weights for examples
- Initialize with uniform weights
- Loop:
  - Apply learner to weighted examples (or sample)
  - Increase weights of misclassified examples
- Combine models by weighted voting
Idea of Boosting
ADABOOST

- ADABOOST boosts the accuracy of the original learning algorithm.

- If the original learning algorithm does slightly better than 50% accuracy, ADABOOST with a large enough number of classifiers is guaranteed to classify the training data perfectly.
Sample Application: Insect Recognition

Using circular regions of interest selected by an interest operator, train a classifier to recognize the different classes of insects.
Boosting Comparison

- **ADTree classifier only** (alternating decision tree)

  - Correctly Classified Instances: 268 (70.1571 %)
  - Incorrectly Classified Instances: 114 (29.8429 %)
  - Mean absolute error: 0.3855
  - Relative absolute error: 77.2229 %

<table>
<thead>
<tr>
<th>Classified as -&gt;</th>
<th>Hesperperla</th>
<th>Doroneuria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real Hesperperlas</td>
<td>167</td>
<td>28</td>
</tr>
<tr>
<td>Real Doroneuria</td>
<td>51</td>
<td>136</td>
</tr>
</tbody>
</table>
Boosting Comparison

AdaboostM1 with ADTree classifier

- Correctly Classified Instances: 303, 79.3194%
- Incorrectly Classified Instances: 79, 20.6806%
- Mean absolute error: 0.2277
- Relative absolute error: 45.6144%

<table>
<thead>
<tr>
<th>Classified as -&gt;</th>
<th>Hesperperla</th>
<th>Doroneuria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real Hesperperlas</td>
<td>167</td>
<td>28</td>
</tr>
<tr>
<td>Real Doroneuria</td>
<td>51</td>
<td>136</td>
</tr>
</tbody>
</table>
Boosting Comparison

- **RepTree classifier only** (reduced error pruning)

- Correctly Classified Instances: 294, 75.3846%
- Incorrectly Classified Instances: 96, 24.6154%
- Mean absolute error: 0.3012
- Relative absolute error: 60.606%

<table>
<thead>
<tr>
<th>Classified as -&gt;</th>
<th>Hesperperla</th>
<th>Doroneuria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real Hesperperlas</td>
<td>169</td>
<td>41</td>
</tr>
<tr>
<td>Real Doroneuria</td>
<td>55</td>
<td>125</td>
</tr>
</tbody>
</table>
Boosting Comparison

**AdaboostM1 with RepTree classifier**

- Correctly Classified Instances 324  83.0769 %
- Incorrectly Classified Instances 66  16.9231 %
- Mean absolute error 0.1978
- Relative absolute error 39.7848 %

<table>
<thead>
<tr>
<th>Classified as -&gt;</th>
<th>Hesperperla</th>
<th>Doroneuria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real Hesperperlas</td>
<td>180</td>
<td>30</td>
</tr>
<tr>
<td>Real Doroneuria</td>
<td>36</td>
<td>144</td>
</tr>
</tbody>
</table>
Bayesian Learning

• **Bayes’ Rule** provides a way to calculate probability of a hypothesis based on
  
  – its prior probability
  
  – the probability of observing the data, given that hypothesis
  
  – the observed data (feature vector)
Bayes’ Rule

\[
P(h \mid X) = \frac{P(X \mid h) \cdot P(h)}{P(X)}
\]

- \( h \) is the hypothesis (such as the class).
- \( X \) is the feature vector to be classified.
- \( P(X \mid h) \) is the prior probability that this feature vector occurs, given that \( h \) is true.
- \( P(h) \) is the prior probability of hypothesis \( h \).
- \( P(X) \) = the prior probability of the feature vector \( X \).
- These priors are usually calculated from frequencies in the training data set.
Example

• Suppose we want to know the probability of class 1 for feature vector [0,1,0].

• \[ P(1 \mid [0,1,0]) = \frac{P([0,1,0] \mid 1) \cdot P(1)}{P([0,1,0])} \]

= \( (0.25) \cdot (0.5) / (0.125) \)

= 1.0

Of course the training set would be much bigger and for real data could include multiple instances of a given feature vector.
• Suppose $H$ is a set of candidate hypotheses.

• We would like to find the **most probable** $h$ in $H$.

• $h_{\text{MAP}}$ is a MAP (maximum a posteriori) hypothesis if

\[
    h_{\text{MAP}} = \underset{h \in H}{\text{argmax}} \ P(h \mid X)
\]

• This just says to calculate $P(h \mid X)$ by Bayes’ rule for each possible class $h$ and take the one that gets the highest score.
Cancer Test Example

P(cancer) = .008
P(not cancer) = .992
P(positive | cancer) = .98
P(positive | not cancer) = .03
P(negative | cancer) = .02
P(negative | not cancer) = .97

New patient’s test comes back positive.

P(cancer | positive) = P(positive | cancer) P(cancer) = (.98) (.008) = .0078

P(not cancer | positive) = P(positive | not cancer) P(not cancer) = (.03) (.992) = .0298

h_{MAP} would say it’s not cancer. Depends strongly on priors!
Neural Net Learning

• Motivated by studies of the brain.

• A network of “artificial neurons” that learns a function.

• Doesn’t have clear decision rules like decision trees, but highly successful in many different applications. (e.g. face detection)

• Our hierarchical classifier used neural net classifiers as its components.
$10^{11}$ neurons of $>20$ types, $10^{14}$ synapses, 1ms–10ms cycle time
Signals are noisy “spike trains” of electrical potential
McCulloch-Pitts “unit”

Output is a “squashed” linear function of the inputs:

\[
a_i \leftarrow g(in_i) = g(\sum_j W_{j,i}a_j)
\]

\[a_0 = -1\]

\[
a_t = g(in_t)
\]

A gross oversimplification of real neurons, but its purpose is to develop understanding of what networks of simple units can do.
(a) is a step function or threshold function

(b) is a sigmoid function \(\frac{1}{1 + e^{-x}}\)

Changing the bias weight \(W_{0,i}\) moves the threshold location
Feed-forward network = a parameterized family of nonlinear functions:

\[ a_5 = g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4) \]
\[ = g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2)) \]

Adjusting weights changes the function: do learning this way!
Perceptron learning

Learn by adjusting weights to reduce error on training set

The squared error for an example with input $x$ and true output $y$ is

$$ E = \frac{1}{2} E_{rr}^2 = \frac{1}{2} (y - h_W(x))^2, $$

Perform optimization search by gradient descent:

$$ \frac{\partial E}{\partial W_j} = E_{rr} \times \frac{\partial E_{rr}}{\partial W_j} = E_{rr} \times \frac{\partial}{\partial W_j} (y - g(\sum_{j=0}^{n} W_j x_j)) $$

$$ = -E_{rr} \times g'(in) \times x_j $$

Simple weight update rule:

$$ W_j \leftarrow W_j + \alpha \times E_{rr} \times g'(in) \times x_j $$

E.g., +ve error $\Rightarrow$ increase network output

$\Rightarrow$ increase weights on +ve inputs, decrease on -ve inputs
Perceptron learning contd.

Perceptron learning rule converges to a consistent function for any linearly separable data set.

- **Majority**
- **Restaurant**

Perceptron learns majority function easily, DTL is hopeless.

DTL learns restaurant function easily, perceptron cannot represent it.
Multilayer perceptrons with back-propagation learning are more powerful.
MLPs are quite good for complex pattern recognition tasks, but resulting hypotheses cannot be understood easily.

Decision tree still wins, but not by as much.
Neural nets (MLP) work great on handwritten digit recognition.

3-nearest-neighbor = 2.4% error
400–300–10 unit MLP = 1.6% error
LeNet: 768–192–30–10 unit MLP = 0.9% error

Current best (kernel machines, vision algorithms) \approx 0.6% error
<table>
<thead>
<tr>
<th>Kernel Machines</th>
</tr>
</thead>
<tbody>
<tr>
<td>• A relatively new learning methodology (1992) derived from statistical learning theory.</td>
</tr>
<tr>
<td>• Became famous when it gave accuracy comparable to neural nets in a handwriting recognition class.</td>
</tr>
<tr>
<td>• Was introduced to computer vision researchers by Tomaso Poggio at MIT who started using it for face detection and got better results than neural nets.</td>
</tr>
<tr>
<td>• Has become very popular and widely used with packages available.</td>
</tr>
</tbody>
</table>
Support Vector Machines (SVM)

- Support vector machines are learning algorithms that try to find a hyperplane that separates the different classes of data the most.

- They are a specific kind of kernel machines based on two key ideas:
  - maximum margin hyperplanes
  - a kernel ‘trick’
Maximal Margin (2 class problem)

In 2D space, a hyperplane is a line.

In 3D space, it is a plane.

Find the hyperplane with maximal margin for all the points. This originates an optimization problem which has a unique solution.
Support Vectors

• The weights $\alpha_i$ associated with data points are zero, except for those points closest to the separator.

• The points with nonzero weights are called the support vectors (because they hold up the separating plane).

• Because there are many fewer support vectors than total data points, the number of parameters defining the optimal separator is small.
A Geometric Interpretation

Class 2

\( \alpha_8 = 0.6 \)

\( \alpha_{10} = 0 \)

\( \alpha_7 = 0 \)

\( \alpha_2 = 0 \)

\( \alpha_1 = 0.8 \)

\( \alpha_6 = 1.4 \)

Class 1

\( \alpha_4 = 0 \)

\( \alpha_9 = 0 \)

\( \alpha_3 = 0 \)

\( w^T x + b = 1 \)

\( w^T x + b = 0 \)

\( w^T x + b = -1 \)
The Kernel Trick

The SVM algorithm implicitly maps the original data to a feature space of possibly infinite dimension in which data (which is not separable in the original space) becomes separable in the feature space.
True decision boundary is $x_1^2 + x_2^2 \leq 1$.

- Mapping the data to the 3D space defined by $f_1 = x_1^2$, $f_2 = x_2^2$, $f_3 = 2^{1/2} x_1 x_2$ makes it linearly separable by a plane in 3D.

- For this problem $F(x_i) \cdot F(x_j)$ is just $(x_i \cdot x_j)^2$, which is called a kernel function.
Kernel Functions

- The kernel function is designed by the developer of the SVM.

- It is applied to pairs of input data to evaluate dot products in some corresponding feature space.

- Kernels can be all sorts of functions including polynomials and exponentials.
Kernel Function used in our 3D Computer Vision Work

- \( k(A,B) = \exp(-\theta^2_{AB}/\sigma^2) \)
- \( A \) and \( B \) are shape descriptors (big vectors).
- \( \theta \) is the angle between these vectors.
- \( \sigma^2 \) is the “width” of the kernel.
Unsupervised Learning

• Find patterns in the data.
• Group the data into clusters.
• Many clustering algorithms.
  – K means clustering
  – EM clustering
  – Graph-Theoretic Clustering
  – Clustering by Graph Cuts
  – etc
Clustering by K-means Algorithm

Form K-means clusters from a set of \( n \)-dimensional feature vectors

1. Set \( ic \) (iteration count) to 1

2. Choose randomly a set of \( K \) means \( m_1(1), \ldots, m_K(1) \).

3. For each vector \( x_i \), compute \( D(x_i, m_k(ic)) \), \( k=1, \ldots, K \) and assign \( x_i \) to the cluster \( C_j \) with nearest mean.

4. Increment \( ic \) by 1, update the means to get \( m_1(ic), \ldots, m_K(ic) \).

5. Repeat steps 3 and 4 until \( C_k(ic) = C_k(ic+1) \) for all \( k \).
K-Means Classifier
(shown on RGB color data)

original data
one RGB per pixel

color clusters
K-Means → EM

The clusters are usually Gaussian distributions.

• **Boot Step:**
  – Initialize $K$ clusters: $C_1, ..., C_K$
    
    $(\mu_j, \Sigma_j)$ and $P(C_j)$ for each cluster $j$.

• **Iteration Step:**
  – Estimate the cluster of each datum
    
    $p(C_j | x_i)$
  – Re-estimate the cluster parameters
    
    $(\mu_j, \Sigma_j), p(C_j)$ For each cluster $j$

The resultant set of clusters is called a **mixture model**; if the distributions are Gaussian, it’s a Gaussian mixture.
EM Algorithm Summary

• **Boot Step:**
  – Initialize $K$ clusters: $C_1, \ldots, C_K$

  $(\mu_j, \Sigma_j)$ and $p(C_j)$ for each cluster $j$.

• **Iteration Step:**
  – **Expectation Step**

  $$p(C_j \mid x_i) = \frac{p(x_i \mid C_j) \cdot p(C_j)}{p(x_i)} = \frac{p(x_i \mid C_j) \cdot p(C_j)}{\sum_j p(x_i \mid C_j) \cdot p(C_j)}$$

  – **Maximization Step**

  $$\mu_j = \frac{\sum_i p(C_j \mid x_i) \cdot x_i}{\sum_i p(C_j \mid x_i)}$$
  $$\Sigma_j = \frac{\sum_i p(C_j \mid x_i) \cdot (x_i - \mu_j) \cdot (x_i - \mu_j)^T}{\sum_i p(C_j \mid x_i)}$$
  $$p(C_j) = \frac{\sum_i p(C_j \mid x_i)}{N}$$
EM Clustering using color and texture information at each pixel
(from Blobworld)
EM for Classification of Images in Terms of their Color Regions

Initial Model for “trees”

Initial Model for “sky”

Final Model for “trees”

Final Model for “sky”

EM
Sample Results

cheetah
Sample Results (Cont.)

grass
Sample Results (Cont.)

lion
WEKA

- WEKA is a set of data mining tools written in Java from the University of Waikato in New Zealand and is named after a flightless bird.
- WEKA is open source software provided under GNU.
- We use it heavily in our research to test out different classifiers. We may later replace a WEKA classifier with a more efficient C++ version.
Open training set file.
1. Choose classify

2. Choose Supplied Test Set

3. Choose a classifier (here MLP)

4. Click Start and wait.
Lots of Available Classifiers

Some that we have used.

- Bayes Classifiers
  - Naive Bayes
  - Bayes Nets
- Functions
  - Multilayered Perception
  - SMO (an SVM)
- Meta classifiers
  - Bagging
  - Adaboost
- Trees
  - REP-tree
  - Random Forest
What Applications Use Machine Learning?

- Computer Vision
- Speech and Natural Language Processing
- Medical Diagnosis
- Predicting Waiting Times in Emergency Rooms
- Financial Planning
- Credit Card Fraud
- Identify Spam