Classification and Regression

- We want to learn functions of the form:
  - $y = f(x)$

- $Y$ is discrete valued:
  - Classification

- $Y$ is continuous
  - Regression

- $X$ can be one or more continuous or discrete values.
Classification

- Estimate a discrete-valued quantity in terms of a number of features
  - Example: Car or Motorcycle?
    - Features:
      - Size in pixels
      - Aspect ratio
      - Average color
      - ...

Regression

- Estimate a continuous-valued quantity in terms of a number of features
- Example: APPL stock price
  - Features:
    - Number of news articles about upcoming products
    - Last quarter’s revenue
    - Cash on hand
    - Whether Steve Jobs is CEO

- Example: Movie rating predictions
  - Features:
    - How much did the user like other movies?
    - How much did other users like this movie?
Basics

- Training dataset
  - Data used to learn our model

- Test dataset
  - Data used to see how well we’ve learned f(x)
  - Why is this separate from training data?

Classification: roadmap

- kNN
- Decision Trees
- Boosting
- SVM
- Neural networks
kNN

**Nearest Neighbor**

- Given feature vector $\mathbf{x}$, estimate $y$ based on previously seen examples close to $\mathbf{x}$
- K-Nearest Neighbors
  - Find $k$ closest examples
    - Majority vote
  - Special data structures make nearest-neighbor lookups relatively fast. (How would you do it?)
- Very simple, effective, little parameter tuning
  - A good “first try” method
Nearest Neighbor

- Example: Predict MPG given:
  - # of cylinders
  - car mass

  Distance = \((c_i - c_j)^2 + (m_i - m_j)^2\)

- What happens?
  - # of cylinders doesn’t matter much at all!
  - Scaling matters!
    - Normalization

Decision Trees
Decision Trees

- Classify attribute vectors into two or more classes
- Boolean case: learn goal predicate

Which boolean functions can we learn?

Mushroom Decision Tree

(Large ∧ ¬Yellow) ∨ (¬Large ∧ Spotted ∧ OnPizza)

from Ginsberg, Essentials of AI
Building Decision Trees

- Given set of examples, derive consistent decision tree
- Idea: just include path for each positive example
  - What’s wrong with this?
  - How can we do better?

Ockham’s Razor

“Pluralitas non est ponenda sine neccesitate” —William of Ockham, 14th century

- Plurality should not be posited without necessity
- Prefer the simplest consistent hypothesis
- Allows for generalization
Building Decision Tree

- **Bad news**
  - Finding smallest possible tree intractable

- **Greedy approach**
  - Starting from root (containing all examples)
  - Until stuck:
    - Pick a node in which not all examples are the same
      - (And at least one attribute is left)
    - Pick attribute most effective in distinguishing among examples
    - Split node using attribute.

### Mushroom Instances

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Size</th>
<th>Color</th>
<th>OnPizza</th>
<th>Edible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spotted</td>
<td>Large</td>
<td>Yellow</td>
<td>Yes</td>
<td>NO</td>
</tr>
<tr>
<td>Spotted</td>
<td>Large</td>
<td>Yellow</td>
<td>No</td>
<td>NO</td>
</tr>
<tr>
<td>Spotted</td>
<td>Large</td>
<td>NY</td>
<td>Yes</td>
<td>YES</td>
</tr>
<tr>
<td>Spotted</td>
<td>Large</td>
<td>NY</td>
<td>No</td>
<td>YES</td>
</tr>
<tr>
<td>Spotted</td>
<td>Small</td>
<td>Yellow</td>
<td>Yes</td>
<td>YES</td>
</tr>
<tr>
<td>Spotted</td>
<td>Small</td>
<td>Yellow</td>
<td>No</td>
<td>NO</td>
</tr>
<tr>
<td>Spotted</td>
<td>Small</td>
<td>NY</td>
<td>Yes</td>
<td>YES</td>
</tr>
<tr>
<td>Spotted</td>
<td>Small</td>
<td>NY</td>
<td>No</td>
<td>NO</td>
</tr>
<tr>
<td>No Spots</td>
<td>Large</td>
<td>Yellow</td>
<td>Yes</td>
<td>NO</td>
</tr>
<tr>
<td>No Spots</td>
<td>Large</td>
<td>Yellow</td>
<td>No</td>
<td>NO</td>
</tr>
<tr>
<td>No Spots</td>
<td>Large</td>
<td>NY</td>
<td>Yes</td>
<td>YES</td>
</tr>
<tr>
<td>No Spots</td>
<td>Large</td>
<td>NY</td>
<td>No</td>
<td>YES</td>
</tr>
<tr>
<td>No Spots</td>
<td>Small</td>
<td>Yellow</td>
<td>Yes</td>
<td>NO</td>
</tr>
<tr>
<td>No Spots</td>
<td>Small</td>
<td>Yellow</td>
<td>No</td>
<td>NO</td>
</tr>
<tr>
<td>No Spots</td>
<td>Small</td>
<td>NY</td>
<td>Yes</td>
<td>NO</td>
</tr>
<tr>
<td>No Spots</td>
<td>Small</td>
<td>NY</td>
<td>No</td>
<td>NO</td>
</tr>
</tbody>
</table>
Choose-Attribute in DTL

Measuring Information Value

- Consider binary event with probability $p$.
- Finding out event resolves uncertainty.
  - $p = 1$ or $0$. Already knew it, no new information.
  - $p = 1/2$. Maximal information from event: 1 bit.
- General formula:
  $$I(q) = -q \log_2 q - (1-q) \log_2 (1-q)$$
Information Gain

- Set of $p$ positive examples, $n$ negative.
- Information value, $I(p/[p+n])$.
- After observing binary attribute, average information is:

$$\frac{p_i + n_i}{p+n} I(\frac{p_i}{[p_i + n_i]}) + \frac{p_f + n_f}{p+n} I(\frac{p_f}{[p_f + n_f]})$$

Information gain is difference between information before and after observing attribute.

Choose-Attribute in DTL

- Size? (Large, Large)
- Pattern? (Spotted, Spotted)
- Color? (Yellow, Yellow)
- Pizza? (OnPizza, OnPizza)
Calculating Initial Information

Initially:

\[ I(5/12) = -\frac{5}{12} \log_2 (5/12) - (7/12) \log_2 (7/12) \]
\[ = -\frac{5}{12}(-1.263) - \frac{7}{12}(-0.778) \]
\[ = .980 \]

Fair amount of uncertainty!

Attribute Information Calculations

After observing “Large” (remainder):

\[ (6/12) I(4/6) + (6/12) I(1/6) = .784 \]
So Gain(Large) = .980 – .784 = .196

After observing “Spotted” (remainder):

\[ (6/12) I(3/6) + (6/12) I(2/6) = .959 \]
So Gain(Spotted) = .980 – .959 = .021

After observing “Yellow” (remainder):

\[ (4/12) I(0) + (8/12) I(5/8) = .636 \]
So Gain(Yellow) = .980 - .636 = .354

After observing “OnPizza” (remainder):

Same as Spotted.

So, split on Yellow: positive = NO, negative is 8 cases.
### Remaining Mushroom Instances

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Size</th>
<th>Color</th>
<th>OnPizza</th>
<th>Edible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spotted</td>
<td>Large</td>
<td>Yellow</td>
<td>Yes</td>
<td>NO</td>
</tr>
<tr>
<td>Spotted</td>
<td>Large</td>
<td>Yellow</td>
<td>No</td>
<td>NO</td>
</tr>
<tr>
<td>Spotted</td>
<td>Large</td>
<td>NY</td>
<td>Yes</td>
<td>YES</td>
</tr>
<tr>
<td>Spotted</td>
<td>Large</td>
<td>NY</td>
<td>No</td>
<td>YES</td>
</tr>
<tr>
<td>Spotted</td>
<td>Small</td>
<td>Yellow</td>
<td>Yes</td>
<td>YES</td>
</tr>
<tr>
<td>Spotted</td>
<td>Small</td>
<td>Yellow</td>
<td>No</td>
<td>NO</td>
</tr>
<tr>
<td>Spotted</td>
<td>Small</td>
<td>NY</td>
<td>Yes</td>
<td>YES</td>
</tr>
<tr>
<td>Spotted</td>
<td>Small</td>
<td>NY</td>
<td>No</td>
<td>NO</td>
</tr>
<tr>
<td>No Spots</td>
<td>Large</td>
<td>Yellow</td>
<td>Yes</td>
<td>NO</td>
</tr>
<tr>
<td>No Spots</td>
<td>Large</td>
<td>Yellow</td>
<td>No</td>
<td>NO</td>
</tr>
<tr>
<td>No Spots</td>
<td>Large</td>
<td>NY</td>
<td>Yes</td>
<td>YES</td>
</tr>
<tr>
<td>No Spots</td>
<td>Large</td>
<td>NY</td>
<td>No</td>
<td>YES</td>
</tr>
<tr>
<td>No Spots</td>
<td>Small</td>
<td>Yellow</td>
<td>Yes</td>
<td>NO</td>
</tr>
<tr>
<td>No Spots</td>
<td>Small</td>
<td>Yellow</td>
<td>No</td>
<td>NO</td>
</tr>
<tr>
<td>No Spots</td>
<td>Small</td>
<td>NY</td>
<td>Yes</td>
<td>NO</td>
</tr>
<tr>
<td>No Spots</td>
<td>Small</td>
<td>NY</td>
<td>No</td>
<td>NO</td>
</tr>
</tbody>
</table>

### Induced Tree

\[
(\neg \text{Yellow} \land \text{Large}) \lor (\neg \text{Yellow} \land \neg \text{Large} \land \text{Spotted} \land \text{OnPizza})
\]
Boosting

**Ensemble Learning**

- Combine predictions from multiple hypotheses
  - May be produced by different learning algorithms
  - Or variations of same algorithm
- To the extent errors are independent, hypotheses are complementary
- Combination more likely to be right than any individual hypothesis
Ensemble Learning

- Combine predictions from multiple hypotheses
  - May be produced by different learning algorithms
  - Or variations of same algorithm
- To the extent errors are independent, hypotheses are complementary
- Combination more likely to be right than any individual hypothesis

Simple Majority Voting

- Build M simple classifiers (e.g. M=5)
  - Suppose (optimistically) that each has an error rate P.
  - Ensemble is wrong only when three or more classifiers are wrong:
    \[ P_M = \binom{5}{3} P^3 (1-P)^2 + \binom{5}{4} P^4 (1-P) + P^5 \]
  - Suppose P = 0.1. Estimate \( P_M \).
- Why is independence assumption optimistic?
Boosting

- Requires: learning method operating over weighted training set.
  - Method attempts to minimize weighted error
  - E.g., decision stumps: decision trees with only one attribute test

- Approach
  - Modify weights over time to reward good performance over “difficult” instances
  - Combine hypotheses derived in each iteration

Boosting Algorithm

- $W(x)$ is the distribution of weights over the $N$ training instances $\sum W(x_i) = 1$
- Initially assign uniform weights $W_0(x) = 1/N$ for all $x$, step $k=0$
- At each iteration $k$:
  - Find hypothesis $H_k(x)$ with minimum error $\varepsilon_k$ using weights $W_k(x)$
  - Compute $a_k = \frac{1}{2} \log \frac{1 - e_k}{e_k}$ — What does $a_k$ look like?
  - Update weights of every training example
    - Correctly labeled points: $W_{k+1} = W_k \cdot \exp(-a_k)$
    - Incorrectly labeled points: $W_{k+1} = W_k \cdot \exp(a_k)$
  - $H_{FINAL}(x) = \text{sign} \left[ \sum a_i H_i(x) \right]$
AdaBoost (Example)

Original Training set: Equal weights for all training samples

Can you find a reasonable decision stump?

Taken from "A Tutorial on Boosting" by Yoav Freund and Rob Schapire

AdaBoost (Example)

ROUND 1

\[ \varepsilon_1 = 0.30 \]
\[ \alpha_1 = 0.42 \]

\[ \varepsilon_1 = \sum_{n=\text{incorrect}} w_n \]
\[ \alpha_1 = \frac{1}{2} \ln \frac{1 - \varepsilon_1}{\varepsilon_1} = \frac{1}{2} \ln \frac{3}{7} = .42 \]

\[ w'_n = C \cdot w_{\text{normalizer}} \]
\[ w_{\text{normalizer}} = (1.091) \cdot (1.654653) = .0714 \]
\[ w_{\text{normalizer}} = (1.091) \cdot (1.527525) = .1667 \]
\[ C_{\text{normalizer}} = \frac{1}{\sum w'_n} = 1.091 \]
AdaBoost (Example)  ROUND 2

\[\epsilon_2 = \sum_{\text{incorrect}} w(\text{incorrect}) = 0.21\]
\[\alpha_2 = \frac{1}{2} \ln \left( \frac{1 - \epsilon_2}{\epsilon_2} \right) = \frac{1}{2} \ln \left( \frac{0.79}{0.21} \right) = 0.65\]

AdaBoost (Example)  ROUND 3

\[\epsilon_3 = 0.14\]
\[\alpha_3 = 0.92\]
**AdaBoost (Example)**

$$H_{\text{final}} = \text{sign}(0.42 + 0.65 + 0.92)$$

**Mushroom Instances**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Size</th>
<th>Color</th>
<th>OnPizza</th>
<th>Edible</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>L</td>
<td>Y</td>
<td>Y</td>
<td>No</td>
</tr>
<tr>
<td>S</td>
<td>L</td>
<td>N</td>
<td>Y</td>
<td>Yes</td>
</tr>
<tr>
<td>S</td>
<td>L</td>
<td>N</td>
<td>N</td>
<td>Yes</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>Y</td>
<td>N</td>
<td>No</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>N</td>
<td>Y</td>
<td>Yes</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>N</td>
<td>N</td>
<td>No</td>
</tr>
<tr>
<td>N</td>
<td>L</td>
<td>Y</td>
<td>N</td>
<td>No</td>
</tr>
<tr>
<td>N</td>
<td>L</td>
<td>N</td>
<td>Y</td>
<td>Yes</td>
</tr>
<tr>
<td>N</td>
<td>L</td>
<td>N</td>
<td>N</td>
<td>Yes</td>
</tr>
<tr>
<td>N</td>
<td>S</td>
<td>Y</td>
<td>Y</td>
<td>No</td>
</tr>
<tr>
<td>N</td>
<td>S</td>
<td>N</td>
<td>Y</td>
<td>No</td>
</tr>
<tr>
<td>N</td>
<td>S</td>
<td>N</td>
<td>N</td>
<td>No</td>
</tr>
</tbody>
</table>
Boosting on Features

Computing Weighting

Hypothesis is: Yellow=Not edible, ~Yellow=Edible

\[ \varepsilon_1 = \sum w(\text{incorrect}) = \frac{1}{12} + \frac{1}{12} + \frac{1}{12} = \frac{1}{4} \]

\[ \alpha_1 = \frac{1}{2} \ln(3/1) = 0.55 \]

\[ w'(\text{correct}) = Cn(1/12)(e^{-0.55}) = 0.048Cn \]

\[ w'(\text{incorrect}) = Cn(1/12)(e^{0.55}) = 0.144Cn \]

Cn normalizes so it is 1.1574

\[ w'(\text{correct}) = 0.0555 \]

\[ w'(\text{incorrect}) = 0.1666 \]
Mushroom Instances

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Size</th>
<th>Color</th>
<th>OnPizza</th>
<th>Edible</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>L</td>
<td>Y</td>
<td>Y</td>
<td>No .0555</td>
</tr>
<tr>
<td>S</td>
<td>L</td>
<td>N</td>
<td>Y</td>
<td>Yes .0555</td>
</tr>
<tr>
<td>S</td>
<td>L</td>
<td>N</td>
<td>N</td>
<td>Yes .0555</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>Y</td>
<td>N</td>
<td>No .0555</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>N</td>
<td>Y</td>
<td>Yes .0555</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>N</td>
<td>N</td>
<td>No .1666</td>
</tr>
<tr>
<td>N</td>
<td>L</td>
<td>Y</td>
<td>N</td>
<td>No .0555</td>
</tr>
<tr>
<td>N</td>
<td>L</td>
<td>N</td>
<td>Y</td>
<td>Yes .0555</td>
</tr>
<tr>
<td>N</td>
<td>L</td>
<td>N</td>
<td>N</td>
<td>Yes .0555</td>
</tr>
<tr>
<td>N</td>
<td>S</td>
<td>Y</td>
<td>Y</td>
<td>No .0555</td>
</tr>
<tr>
<td>N</td>
<td>S</td>
<td>N</td>
<td>Y</td>
<td>No .1666</td>
</tr>
<tr>
<td>N</td>
<td>S</td>
<td>N</td>
<td>N</td>
<td>No .1666</td>
</tr>
</tbody>
</table>

Boosting on Features (step 2)
Computing Weighting

Hypothesis is: Large=Yes, ~Large=No

\[ \varepsilon^2 = \sum w(\text{incorrect}) = (2 \cdot 0.0555) + (1 \cdot 0.0555) = 0.1665 \]

\[ \alpha^2 = \frac{1}{2} \ln(0.8335/0.1665) = 0.80 \]
An Alternative

- A good world model often has several interacting processes
  - Bayes nets, for example
    - Inputs = Earthquake, Burglary
    - Outputs = John/Mary calls
  - Hidden nodes moderate influence between other nodes
    - Alarm

- Conceptual idea: perhaps hidden nodes are there, even if we don’t know what they are
  - Can we assume the presence of hidden nodes and learn their behavior

Brain Inspiration

- It is hard to make a machine behave intelligently
- Approach: reverse engineering!
- Problem: we don’t really know all about how brains work, either
Neurons

- Brains are made out of **neurons**.
- Lots of them (~$10^{11}$)
  - Highly connected
  - Really slow (~1ms)

- Cartoon version
  - Neuron “fires” along axon given sufficient signal from dendrites

McCulloch-Pitts Model

- (1943) Neuron as threshold unit
- Output is one iff weighted sum of inputs exceeds threshold
Representing Logical Fns

with AND and NOT, can represent any combinational circuit (any boolean function).

Slightly Generalized Model

\[ in_i = \sum_j W_{j,i} a_j \]  
input fn

\[ g \]  
activation fn

\[ a_i = g(in_i) \]  
output
Activation Functions

- **Step function**
  
  \[ g(x) = \begin{cases} 
  1 & \text{iff } x > 0, \\
  0 & \text{else}. 
  \end{cases} \]

- **Sigmoid**
  
  \[ g(x) = \frac{1}{1 + e^{-x}} \]

Neural Networks

- **Collection of units, connected together**
  
  Recurrent: cycles allowed
  Feedforward: no cycles
  Layered: can partition into strata
Perceptrons

- (Rosenblatt, 1950s)
- Set of units in a single feedforward layer
  - (inputs connected directly to outputs)

\[
\text{out} = \text{Step}_0 \left( \sum_j W_j x_j \right) = \text{Step}_0 (\mathbf{W} \cdot \mathbf{x})
\]

Output is 1 iff: \( \mathbf{W} \cdot \mathbf{x} \geq 0 \)

For two inputs: \( W_1 x_1 + W_2 x_2 \geq W_0 \)

\[
x_2 \geq \frac{W_0}{W_2} - \frac{W_1}{W_2} x_1
\]

Perceptron Boundaries

\[
x_2 \geq \frac{W_0}{W_2} - \frac{W_1}{W_2} x_1
\]
Linear Separability

- $x_1 \text{ AND } x_2$
- $x_1 \text{ OR } x_2$
- $x_1 \text{ IFF } x_2$

Perceptron Limitations

- Can’t learn functions that aren’t linearly separable
- But, we can learn some “hard” functions easily!
Perceptron Learning

- Suppose we have weights $\mathbf{w}$
- Observe $\mathbf{x}_i, y_i$

- What is the error?
  \[ e = y_i - g(\mathbf{w} \cdot \mathbf{x}_i) \]

- Squared error:
  \[ e^2 = (y_i - g(\mathbf{w} \cdot \mathbf{x}_i))^2 \]

Perceptron Learning

- Squared error: \[ e^2 = (y_i - g(\mathbf{w} \cdot \mathbf{x}_i))^2 \]

- How do we minimize the squared error?
  - We can adjust $\mathbf{w}$'s:
  - \[ \frac{de^2}{dw_i} = \]

  - Adjusting $w_j$ in the opposite direction will reduce $e^2$
    \[ w_j' = w_j - \frac{de^2}{\delta w_j} \quad (????) \]

- How big a step should we take?
Perceptron Learning

- How big a step should we take?
  - Could we compute how big a step would reduce the error to zero?
  - Do we really want to fit this training example?

- Learning rate: \( \alpha \)
  \[ w_j' = w_j + \alpha \frac{\delta e^2}{\delta w_j} \]

- Yes, but what should \( \alpha \) be?

Learning Rate

- What should \( \alpha \) be?
  - Hard to pick... must tune.

- Stochastic Gradient Descent
  - Learning rate schedule
  - Fancier strategies, e.g. search then converge
Perceptron Learning Wrap-Up

- Repeat
  - Pick an example $x_i, y_i$
  - Compute error: $e = y_i - g(w \cdot x_i)$
  - For each input $j$:
    $$w_j' = w_j + \alpha \frac{\delta e^2}{\delta w_j}$$

- Hill Climbing – iterative improvement
  - Given small enough $\alpha$, it will converge.

- A bit of terminology:
  - Epoch: do an update for every example

Limitations

- Many (most?) interesting functions not linearly separable
  - From late 1960s, interest in perceptrons waned
- Can get around expressive limitations with multilayer networks
Multilayer Networks

With enough hidden units, can represent any continuous function, not just linearly separable ones.

Learning Multilayer Networks

- More difficult, because we do not know what hidden units should represent.
- Multiple weights between every input and output.
- Credit (blame) assignment problem.
- (Re)discovery of backpropagation in 1980s led to resurgent interest in neural networks.
Back-propagation

- Basic idea:
  - Compute effect of every weight on output.
    - Work backwards from output to input
    - Similar to chain rule.
  - If output is wrong value, move weights in \(-\)gradient direction.

Neural Networks

- Appealing due to brain analogy
- Other advantages
  - Simplicity, expressiveness,
  - Ability to handle noise
- Disadvantages
  - Opaque: cannot be used in some applications due to regulatory constraints!
  - Black art of designing structures and tuning parameters
- Ultimately, one of many forms of nonlinear regression
Support Vector Machines

- All about separability:
  - Given a bunch of features, find the (linear) separator that maximizes the *margin*.

- This can be formulated as a quadratic programming problem.
SVMs: Features

- The key is to find features that make the data linearly separable

- When viewed from the original space, these features can be complex looking.

SVMs: Kernel Trick

- Where do we get the “right” features?
  - In higher dimensions, data tends to become linearly separable, even if the features aren’t particularly clever.

- Idea: generate features from our data
  - E.g., compute the dot product of every point \( x_i \) with respect to \( x_1 \)
  - In fact, let’s make every point its own feature

- Linear separators can be efficiently computed for features of this form
  - “Kernel Trick”
  - We won’t worry about mechanics
Scoreboard

X

O

Regression

78