Classification and Prediction Overview

- Introduction
- Decision Trees
- Statistical Decision Theory
- Nearest Neighbor
- Bayesian Classification
- Artificial Neural Networks
- Support Vector Machines (SVMs)
- Prediction
- Accuracy and Error Measures
- Ensemble Methods

Classification vs. Prediction

- Assumption: after data preparation, have single data set where each record has attributes $X_1,...,X_n$ and $Y$.
- Goal: learn a function $f:(X_1,...,X_n)\rightarrow Y$, then use this function to predict $y$ for a given input record $(x_1,...,x_n)$.
  - Classification: $Y$ is a discrete attribute, called the class label
  - Prediction: $Y$ is a continuous attribute
- Called supervised learning, because true labels ($Y$-values) are known for the initially provided data
- Typical applications: credit approval, target marketing, medical diagnosis, fraud detection

Induction: Model Construction

```
NAME    RANK     YEARS TENURED
Mike    Assistant Prof 3 no
Mary    Assistant Prof 7 yes
Bill    Professor    2 yes
Jim     Associate Prof 7 yes
Dave    Assistant Prof 6 no
Anne    Associate Prof 3 no
```

```
IF rank = 'professor' OR years > 6
THEN tenured = 'yes'
```

Deduction: Using the Model

```
NAME    RANK     YEARS TENURED
Tom     Assistant Prof 2 no
Merlisa Associate Prof 7 no
George  Professor    5 yes
Joseph  Assistant Prof 7 yes
```

```
Tenured?
Yes
```
Example of a Decision Tree

Training Data

Marital Status
- Single
- Married
- Divorced

Taxable Income

Cheat
- No
- Yes

Another Example of Decision Tree

Model: Decision Tree

There could be more than one tree that fits the same data!
Apply Model to Test Data

Test Data

<table>
<thead>
<tr>
<th>Refund</th>
<th>MarSt</th>
<th>TaxInc</th>
<th>Cheat</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>Single, Div</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>NO</td>
<td>Married</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Decision Tree Induction

- Basic greedy algorithm
  - Top-down, recursive divide-and-conquer
  - At start, all the training records are at the root
  - Training records partitioned recursively based on split attributes
  - Split attributes selected based on a heuristic or statistical measure (e.g., information gain)
- Conditions for stopping partitioning
  - Pure node (all records belong to same class)
  - No remaining attributes for further partitioning
    - Majority voting for classifying the leaf
  - No cases left

How to Specify Split Condition?

- Depends on attribute types
  - Nominal
  - Ordinal
  - Numeric (continuous)
- Depends on number of ways to split
  - 2-way split
  - Multi-way split

Splitting Nominal Attributes

- Multi-way split: use as many partitions as distinct values.
- Binary split: divides values into two subsets; need to find optimal partitioning.

Decision Boundary

Decision boundary = border between two neighboring regions of different classes.
For trees that split on a single attribute at a time, the decision boundary is parallel to the axes.
Splitting Ordinal Attributes

- Multi-way split:

- Binary split:

- What about this split?

Splitting Continuous Attributes

- Different options
  - Discretization to form an ordinal categorical attribute
    - Static – discretize once at the beginning
    - Dynamic – ranges found by equal interval bucketing, equal frequency bucketing (percentiles), or clustering.
  - Binary Decision: \((A < v)\) or \((A \geq v)\)
    - Consider all possible splits, choose best one

How to Determine Best Split

- Greedy approach:
  - Nodes with homogeneous class distribution are preferred
- Need a measure of node impurity:

  \[
  \text{Gain}_A(D) = \text{Info}(D) - \sum_{j=1}^{v} \frac{|D_j|}{|D|} \text{Info}(D_j)
  \]

  Information gained by splitting on attribute \(A\):

Attribute Selection Measure: Information Gain

- Select attribute with highest information gain
- \(p_i\) = probability that an arbitrary record in \(D\) belongs to class \(C_i, i = 1, \ldots, m\)
- Expected information (entropy) needed to classify a record in \(D\):

  \[
  \text{Info}(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)
  \]
Example

- Predict if somebody will buy a computer
- Given data set:

<table>
<thead>
<tr>
<th>Age</th>
<th>Credit_rating</th>
<th>Income</th>
<th>Buy_computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>≤ 30 High No No Bad No</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>≤ 30 High No Good No</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt; 40 Medium No Bad Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt; 40 Low Yes Bad Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>≥ 40 Low Yes Good Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>≥ 40 Medium No Good Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt; 40 Medium No Bad No</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Information Gain Example

- Class P: buys_computer = “yes”
- Class N: buys_computer = “no”

\[ \text{GainRatio}_{\text{attribute}} = \frac{\text{Gain}_{\text{attribute}}}{\text{SplitInfo}_{\text{attribute}}} \]

\[ \text{SplitInfo}_{\text{attribute}} = \sum_{j=1}^{v} \left| \frac{D_j}{|D|} \right| \log_2 \left( \frac{|D_j|}{|D|} \right) \]

E.g., \[ \text{SplitInfo}_{\text{age}} = \frac{4}{14} \log_2 \left( \frac{4}{14} \right) + \frac{6}{14} \log_2 \left( \frac{6}{14} \right) + \frac{4}{14} \log_2 \left( \frac{4}{14} \right) = 0.926 \]

\[ \text{GainRatio}_{\text{age}} = 0.029/0.926 = 0.031 \]

- Attribute with maximum gain ratio is selected as splitting attribute

Gain Ratio for Attribute Selection

- Information gain is biased towards attributes with a large number of values
- Use gain ratio to normalize information gain:
  - \( \text{GainRatio}_{\text{attribute}} = \frac{\text{Gain}_{\text{attribute}}}{\text{SplitInfo}_{\text{attribute}}} \)
  - \( \text{Gain}_{\text{attribute}} \) is defined as \( \sum_{i=1}^{c} p_i \log \left( p_i \right) \)

Gain Index

- Gini index, \( \text{gini}(D) \), is defined as \( \text{gini}(D) = 1 - \sum_{i=1}^{c} p_i^2 \)
- If data set \( D \) is split on \( A \) into \( v \) subsets \( D_1, \ldots, D_v \), the gini index \( \text{gini}_A(D) \) is defined as

\[ \text{gini}_A(D) = \sum_{j=1}^{v} \left| \frac{D_j}{|D|} \right| \text{gini}(D_j) \]

- Reduction in impurity:

\[ \Delta \text{gini}_A(D) = \text{gini}(D) - \text{gini}_A(D) \]

- Attribute that provides smallest \( \Delta \text{gini}_A(D) \) (i.e., largest reduction in impurity) is chosen to split the node

Comparing Attribute Selection Measures

- No clear winner (and there are many more)
  - Information gain:
    - Biased towards multivalued attributes
  - Gain ratio:
    - Tends to prefer unbalanced splits where one partition is much smaller than the others
  - Gini index:
    - Biased towards multivalued attributes
    - Tends to favor tests that result in equal-sized partitions and purity in both partitions

Practical Issues of Classification

- Underfitting and overfitting
- Missing values
- Computational cost
- Expressiveness
How Good is the Model?

- **Training set error**: compare prediction of training record with true value
  - Not a good measure for the error on unseen data. (Discussed soon.)
- **Test set error**: for records that were not used for training, compare model prediction and true value
  - Use holdout data from available data set

### Training versus Test Set Error

- We’ll create a training dataset
  - Five inputs, all bits, are generated in all 32 possible combinations
  - Output $y = \text{copy of e, except a random } 25\% \text{ of the records have } y \text{ set to the opposite of e}$

### Test Data

- Generate test data using the same method: copy of $e$, but 25% inverted.
- Some $y$’s that were corrupted in the training set will be uncorrupted in the testing set.
- Some $y$’s that were uncorrupted in the training set will be corrupted in the test set.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>y (training data)</th>
<th>y (test data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Testing The Tree with The Test Set

| 1/4 of the tree nodes are corrupted | 3/4 are fine |
| 1/4 of the test set records are corrupted | 1/16 of the test set will be correctly predicted for the wrong reasons |
| 3/16 of the test set will be wrongly predicted because the test record is corrupted |
| 3/16 of the test predictions will be wrong because the tree node is corrupted |

- In total, we expect to be wrong on 3/8 of the test set predictions

### What’s This Example Shown Us?

- Discrepancy between training and test set error
- But more importantly
  - It indicates that there is something we should do about it if we want to predict well on future data.
Suppose We Had Less Data

Output $y$ = copy of $e$, except a random 25% of the records have $y$ set to the opposite of $e$

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

These bits are hidden

Tree Learned Without Access to The Irrelevant Bits

These nodes will be unexpandable

In about 12 of the 16 records in this node the output will be 0
So this will almost certainly predict 0

In about 12 of the 16 records in this node the output will be 1
So this will almost certainly predict 1

Tree Learned Without Access to The Irrelevant Bits

<table>
<thead>
<tr>
<th>e=0</th>
<th>e=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>almost certainly none of the tree nodes are corrupted</th>
<th>almost certainly all are fine</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/4 of the test set records are corrupted IVa</td>
<td>1/4 of the test set will be wrongly predicted because the test record is corrupted</td>
</tr>
<tr>
<td>3/4 are fine IVa</td>
<td>3/4 of the test predictions will be fine</td>
</tr>
</tbody>
</table>

In total, we expect to be wrong on only 1/4 of the test set predictions

Typical Observation

<table>
<thead>
<tr>
<th>Number of instances</th>
<th>Overfitting</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
</tbody>
</table>

Underfitting: when model is too simple, both training and test errors are large

Reasons for Overfitting

- **Noise**
  - Too closely fitting the training data means the model's predictions reflect the noise as well
- **Insufficient training data**
  - Not enough data to enable the model to generalize beyond idiosyncrasies of the training records
- **Data fragmentation (special problem for trees)**
  - Number of instances gets smaller as you traverse down the tree
  - Number of instances at a leaf node could be too small to make any confident decision about class
Avoiding Overfitting

- General idea: make the tree smaller
  - Addresses all three reasons for overfitting
- Prepruning: Halt tree construction early
  - Do not split a node if this would result in the goodness measure falling below a threshold
  - Difficult to choose an appropriate threshold, e.g., tree for XOR
- Postpruning: Remove branches from a “fully grown” tree
  - Use a set of data different from the training data to decide when to stop pruning
    - Validation data: train tree on training data, prune on validation data, then test on test data

Minimum Description Length (MDL)

- Alternative to using validation data
  - Motivation: data mining is about finding regular patterns in data; regularity can be used to compress the data; method that achieves greatest compression found most regularity and hence is best
  - Minimize \( \text{Cost(Model, Data)} = \text{Cost(Model)} + \text{Cost(Data|Model)} \)
  - \( \text{Cost} \) is the number of bits needed for encoding.
  - \( \text{Cost(Data|Model)} \) encodes the misclassification errors.
  - \( \text{Cost(Model)} \) uses node encoding plus splitting condition encoding.

MDL-Based Pruning Intuition

Handling Missing Attribute Values

- Missing values affect decision tree construction in three different ways:
  - How impurity measures are computed
  - How to distribute instance with missing value to child nodes
  - How a test instance with missing value is classified

Distribute Instances

Computing Impurity Measure

Split on Refund: assume records with missing values are distributed as discussed before
- 3/9 of record 10 go to Refund=Yes
- 6/9 of record 10 go to Refund=No

\[ \text{Entropy(Refund=Yes)} = -\left( \frac{3}{10} \log \frac{3}{10} + \frac{7}{10} \log \frac{7}{10} \right) \approx 0.690 \]

\[ \text{Entropy(Refund=No)} = -\left( \frac{8}{10} \log \frac{8}{10} + \frac{2}{10} \log \frac{2}{10} \right) \approx 0.971 \]

\[ \text{Gain} = 0.690 - 0.971 = -0.281 \]

Before Splitting: \[ \text{Entropy(Parent)} = -0.3 \log(0.3) - 0.7 \log(0.7) = 0.881 \]
Classification in Large Databases

- **Scalability:** Classify data sets with millions of examples and hundreds of attributes with reasonable speed
- **Why use decision trees for data mining?**
  - Relatively fast learning speed
  - Can handle all attribute types
  - Convertible to simple and easy to understand classification rules
  - Good classification accuracy, but not as good as newer methods (but tree ensembles are top!)

Tree Cost Analysis

- Finding an optimal decision tree is NP-complete
  - Optimization goal: minimize expected number of binary tests to uniquely identify any record from a given finite set
- **Greedy algorithm**
  - O(#attributes * #training_instances * log(#training_instances))
  - At each tree depth, all instances considered
  - Assume tree depth is logarithmic (fairly balanced splits)
  - Need to test each attribute at each node
  - What about binary splits?
    - Sort data once on each attribute, use to avoid re-sorting subsets
    - Incrementally maintain counts for class distribution as different split points are explored
  - In practice, trees are considered to be fast both for training (when using the greedy algorithm) and making predictions

Rule Extraction from a Decision Tree

- One rule is created for each path from the root to a leaf
  - Precondition: conjunction of all split predicates of nodes on path
  - Consequent: class prediction from leaf
- **Rules are mutually exclusive and exhaustive**
- Example: Rule extraction from buys_computer decision-tree
  - IF age = young AND student = no THEN buys_computer = no
  - IF age = young AND student = yes THEN buys_computer = yes
  - IF age = mid-age THEN buys_computer = yes
  - IF age = old AND credit_rating = excellent THEN buys_computer = yes
  - IF age = young AND credit_rating = fair THEN buys_computer = no

Scalable Tree Induction

- High cost when the training data at a node does not fit in memory
- **Solution 1:** special I/O-aware algorithm
  - Keep only class list in memory, access attribute values on disk
  - Maintain separate list for each attribute
  - Use count matrix for each attribute
- **Solution 2:** Sampling
  - Common solution: train tree on a sample that fits in memory
  - More sophisticated versions of this idea exist, e.g., Rainforest
  - Build tree on sample, but do this for many bootstrap samples
  - Combine all into a single new tree that is guaranteed to be almost identical to the one trained from entire data set
  - Can be computed with two data scans
Tree Conclusions

• Very popular data mining tool
  – Easy to understand
  – Easy to implement
  – Easy to use
    • Little tuning, handles all attribute types and missing values
  – Computationally cheap
• Overfitting problem
• Focused on classification, but easy to extend to prediction (future lecture)

Classification and Prediction Overview

• Introduction
• Decision Trees
• Statistical Decision Theory
• Nearest Neighbor
• Bayesian Classification
• Artificial Neural Networks
• Support Vector Machines (SVMs)
• Prediction
• Accuracy and Error Measures
• Ensemble Methods

Theoretical Results

• Trees make sense intuitively, but can we get some hard evidence and deeper understanding about their properties?
• Statistical decision theory can give some answers
• Need some probability concepts first

Random Variables

• Intuitive version of the definition:
  – Can take on one of possibly many values, each with a certain probability (discrete versus continuous)
  – These probabilities define the probability distribution of the random variable
  – E.g., let X be the outcome of a coin toss, then Pr(X=’heads’)=0.5 and Pr(X=’tails’)=0.5; distribution is uniform
• Consider a discrete random variable X with numeric values $x_1, x_2, \ldots, x_k$
  – Expectation: $E[X] = \sum x_i \cdot \text{Pr}(X=x_i)$
  – Variance: $\text{Var}(X) = E[(X - E[X])^2] = E[X^2] - (E[X])^2$

Working with Random Variables

• $E[X + Y] = E[X] + E[Y]$
• $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y) + 2 \text{Cov}(X,Y)$
• For constants $a, b$
  – $E[aX + b] = a \cdot E[X] + b$
  – $\text{Var}(aX + b) = a^2 \cdot \text{Var}(X)$
• Iterated expectation:
  – $E[X] = E[ E[Y | X] ]$, where $E_Y[Y | X] = \sum y \cdot \text{Pr}(Y=y | X=x)$ is the expectation of $Y$ for a given value of $X$, i.e., is a function of $X$
  – In general for any function $f(X,Y)$:
    $E_{Y|X}[f(X,Y)] = E_X[E_Y[f(X,Y) | X]]$

What is the Optimal Model $f(X)$?

Let $X$ denote a real-valued random input variable and $Y$ a real-valued random output variable.

The squared error of trained model $f(X)$ is $E_{Y|X}[(Y - f(X))^2]$

Which function $f(X)$ will minimize the squared error?

Consider the error for a specific value of $X$ and let $\mathcal{E} = E_{Y|X}[Y | X]$

$$E_{Y|X}[Y - f(X)]^2 = E_{Y|X}[(Y - \mathcal{E} + \mathcal{E} - f(X))^2]$$

$$= E_{Y|X}[(Y - \mathcal{E} + \mathcal{E} - f(X))^2] + 2 \mathcal{E} (\mathcal{E} - f(X)) + \mathcal{E}^2 + (f(X))^2$$

$$= E_{Y|X}[(Y - \mathcal{E} + \mathcal{E} - f(X))^2] + 2 \mathcal{E} (\mathcal{E} - f(X)) + \mathcal{E}^2 + (f(X))^2$$

$$= E_{Y|X}[(Y - \mathcal{E} + \mathcal{E} - f(X))^2] + 2 \mathcal{E} (\mathcal{E} - f(X)) + \mathcal{E}^2 + (f(X))^2$$

$$(\text{Notice: } E_{Y|X}[Y] = E_X[E_{Y|X}[Y | X]] - E_X[E_{Y|X}[Y | X]] - \mathcal{E} = 0)$$
Optimal Model $f(X)$ (cont.)

The choice of $f(X)$ does not affect $E_{x}[Y - f(x)]^2$, but $E_{x}[f(x) - f(X)]^2$ is minimized for $f(X) = f(x)$. Hence

$$E_{x}[f(x) - f(X)]^2 = E_{x}[E_{Y|X}[Y - f(x)]^2]$$

Hence the squared error is minimized by choosing $f(x) = E_{Y|X}[Y]$. For each $X$, then, the best model is $f(x) = \text{median}(X)$. Note that for minimizing absolute error $E_{x}||f(x) - f(X)||$ one can show that the best model is $f(X) = X$.

Bias-Variance Tradeoff

- Let's take this one step further and see if we can understand overfitting through statistical decision theory.
- As before, consider two random variables $X$ and $Y$.
- From a training set $D$ with $n$ records, we want to construct a function $f(X)$ that returns good approximations of $Y$ for future inputs $X$.
- Make dependence of $f$ on $D$ explicit by writing $f(X; D)$.
- Goal: minimize mean squared error over all $X$, $Y$, and $D$, i.e., $E_{X,D}[ (Y - f(X; D))^2 ]$.

Variance Tradeoff Derivation

$E_{x}[f(x) - f(X)]^2 = E_{x}[E_{Y|X}[Y - f(x)]^2]$

(First term does not depend on $D$, hence $E_{x}[f(x) - f(X)]^2 = E_{x}[E_{Y|X,D}[Y - f(x)]^2]$)

Consider the second term:

$$E_{x}[E_{Y|X,D}[Y - f(x)]^2] = E_{x}[E_{Y|X,D}[E_{X|Y,D}[f(x) - f(X)]|Y]^2]$$

$$= E_{x}[E_{Y|X,D}[E_{X|Y,D}[f(x) - f(X)]|Y]E_{X|Y,D}[f(x) - f(X)]|Y]$$

$$= E_{x}[E_{Y|X,D}E_{X|Y,D}[f(x) - f(X)]|Y]$$

Consider the third term:

$$E_{x}[E_{Y|X,D}E_{X|Y,D}[f(x) - f(X)]|Y]$$

(Third term is zero, because $E_{x}[E_{Y|X,D}E_{X|Y,D}[f(x) - f(X)]|Y] = E_{x}[E_{Y|X,D}-E_{x}[E_{Y|X,D}|Y] = 0$)

Overall we therefore obtain:

$$E_{x}[f(x) - f(X)]^2 = E_{x}[E_{Y|X,D}[Y - f(x)]^2]$$

Implications for Trees

- Best prediction for input $X = x$ is the mean of the $Y$-values for all records $(x, y)$ with $x = x$.
- What about classification?
  - Two classes: encode as 0 and 1, use squared error as before.
  - $k$ classes: can show that for $0 < 1 - 0$ if correct class, $\text{error} = 1$ if wrong class predicted the optimal choice is to return the majority class for a given input $X = x$.
- Called the Bayes classifier.
- Problem: How can we estimate $E[Y|X = x]$ or the majority class for $X$ from the training data?
  - Often there is just one or no training record for a given $X = x$.
- Solution: approximate it.
  - Use $Y$-values from training records in neighborhood around $X = x$.
  - Tree: leaf defines neighborhood in the data space; make sure there are enough records in the leaf to obtain reliable estimate of correct answer.

Bias-Decreasing Property

- Bias decreases as tree becomes larger.
  - Larger tree can fit training data better.
- Variance increases as tree becomes larger.
  - Sample variance affects predictions of larger tree.
- Find right tradeoff as discussed earlier.
  - Validation data to find best pruned tree.
  - MDL principle.
Classification and Prediction Overview

- Introduction
- Decision Trees
- Statistical Decision Theory
- Nearest Neighbor
- Bayesian Classification
- Artificial Neural Networks
- Support Vector Machines (SVMs)
- Prediction
- Accuracy and Error Measures
- Ensemble Methods

Lazy vs. Eager Learning

- **Lazy** learning: Simply stores training data (or only minor processing) and waits until it is given a test record
- **Eager** learning: Given a training set, constructs a classification model before receiving new (test) data to classify
- General trend: Lazy = faster training, slower predictions
- **Accuracy:** not clear which one is better!
  - Lazy method: typically driven by local decisions
  - Eager method: driven by global and local decisions

Nearest-Neighbor

- Recall our statistical decision theory analysis: Best prediction for input \( X=x \) is the mean of the \( Y \)-values of all records \((x(i),y(i))\) with \( x(i)=x \) (majority class for classification)
- Problem was to estimate \( E[Y|X=x] \) or majority class for \( X=x \) from the training data
- Solution was to approximate it
  - Use \( Y \)-values from training records in neighborhood around \( X=x \)

Nearest-Neighbor Classifiers

- Requires:
  - Set of stored records
  - Distance metric for pairs of records
    - Common choice: Euclidean
    \[ d(p,q) = \sqrt{\sum (p_i - q_i)^2} \]
  - Parameter \( k \)
    - Number of nearest neighbors to retrieve
- To classify a record:
  - Find its \( k \) nearest neighbors
  - Determine output based on (distance-weighted) average of neighbors' output

Definition of Nearest Neighbor

- **K-nearest neighbors of a record** \( x \) are data points that have the \( k \) smallest distance to \( x \)

1-Nearest Neighbor

Voronoi Diagram
Nearest Neighbor Classification

• Choosing the value of \( k \):
  – \( k \) too small: sensitive to noise points
  – \( k \) too large: neighborhood may include points from other classes

Effect of Changing \( k \)

Explaining the Effect of \( k \)

• Recall the bias-variance tradeoff
• Small \( k \), i.e., predictions based on few neighbors
  – High variance, low bias
• Large \( k \), e.g., average over entire data set
  – Low variance, but high bias
• Need to find \( k \) that achieves best tradeoff
• Can do that using validation data

Scaling Issues

• Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes
• Example:
  – Height of a person may vary from 1.5m to 1.8m
  – Weight of a person may vary from 90lb to 300lb
  – Income of a person may vary from $10K to $1M
  – Income difference would dominate record distance

Other Problems

• Problem with Euclidean measure:
  – High dimensional data: curse of dimensionality
  – Can produce counter-intuitive results

\[
\begin{array}{c|c}
11111111110 & 100000000000 \\
01111111111 & 000000000001 \\
\end{array}
\]

\[d = 1.4142 \quad d = 1.4142\]

  – Solution: Normalize the vectors to unit length
  – Irrelevant attributes might dominate distance
    – Solution: eliminate them

Computational Cost

• Brute force: \( O(\#\text{trainingRecords}) \)
  – For each training record, compute distance to test record, keep if among top-\( k \)
• Pre-compute Voronoi diagram (expensive), then search spatial index of Voronoi cells: if lucky \( O(\log(\#\text{trainingRecords})) \)
• Store training records in multi-dimensional search tree, e.g., R-tree: if lucky \( O(\log(\#\text{trainingRecords})) \)
• Bulk-compute predictions for many test records using spatial join between training and test set
  – Same worst-case cost as one-by-one predictions, but usually much faster in practice
Classification and Prediction Overview

- Introduction
- Decision Trees
- Statistical Decision Theory
- Nearest Neighbor
- Bayesian Classification
- Artificial Neural Networks
- Support Vector Machines (SVMs)
- Prediction
- Accuracy and Error Measures
- Ensemble Methods

Bayesian Classification

- Performs probabilistic prediction, i.e., predicts class membership probabilities
- Based on Bayes’ Theorem
- Incremental training
  - Update probabilities as new training records arrive
  - Can combine prior knowledge with observed data
- Even when Bayesian methods are computationally intractable, they can provide a standard of optimal decision making against which other methods can be measured

Bayesian Theorem: Basics

- $X$ = random variable for data records ("evidence")
- $H$ = hypothesis that specific record $X=x$ belongs to class $C$
- Goal: determine $P(H|X=x)$
  - Probability that hypothesis holds given a record $x$
- $P(H)$ = prior probability
  - The initial probability of the hypothesis
  - E.g., person $x$ will buy computer, regardless of age, income etc.
- $P(X=x)$ = probability that data record $x$ is observed
- $P(X=x|H)$ = probability of observing record $x$, given that the hypothesis holds
  - E.g., given that $x$ will buy a computer, what is the probability that $x$ is in age group 31...40, has medium income, etc.?

Bayes’ Theorem

- Given data record $x$, the posterior probability of a hypothesis $H$, $P(H|X=x)$, follows from Bayes theorem:
  \[
P(H|X=x) = \frac{P(X=x|H)P(H)}{P(X=x)}
\]
  - Informally: posterior = likelihood * prior / evidence
  - Among all candidate hypotheses $H$, find the maximally probably one, called maximum a posteriori (MAP) hypothesis
  - Note: $P(X=x)$ is the same for all hypotheses
  - If all hypotheses are equally probable a priori, we only need to compare $P(X=x|H)$
  - Winning hypothesis is called the maximum likelihood (ML) hypothesis
  - Practical difficulties: requires initial knowledge of many probabilities and has high computational cost

Towards Naïve Bayes Classifier

- Suppose there are $m$ classes $C_1, C_2, ..., C_m$
- Classification goal: for record $x$, find class $C_i$ that has the maximum posterior probability $P(C_i|X=x)$
- Bayes’ theorem:
  \[
P(C_i|X=x) = \frac{P(X=x|C_i)P(C_i)}{P(X=x)}
\]
  - Since $P(X=x)$ is the same for all classes, only need to find maximum of $P(X=x|C_i)P(C_i)$

Computing $P(X=x|C_i)$ and $P(C_i)$

- Estimate $P(C_i)$ by counting the frequency of class $C_i$ in the training data
- Can we do the same for $P(X=x|C_i)$?
  - Need very large set of training data
  - Have $|X_1|*|X_2|*...*|X_m|$ different combinations of possible values for $X$ and $C_i$
  - Need to see every instance $x$ many times to obtain reliable estimates
- Solution: decompose into lower-dimensional problems
Example: Computing \( P(X=x_i | C_i) \) and \( P(C_i) \)

- \( P(\text{buys\_computer} = \text{yes}) = 9/14 \)
- \( P(\text{buys\_computer} = \text{no}) = 5/14 \)
- \( \text{age} = 30, \text{income} = \text{low}, \text{student} = \text{no}, \text{credit\_rating} = \text{bad} \) \( \text{buys\_computer} = \text{yes} \) = 0.7

<table>
<thead>
<tr>
<th>Age</th>
<th>Income</th>
<th>Student</th>
<th>Credit_rating</th>
<th>Buys_computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>≤ 30</td>
<td>High</td>
<td>No</td>
<td>Bad</td>
<td>No</td>
</tr>
<tr>
<td>≤ 30</td>
<td>High</td>
<td>No</td>
<td>Good</td>
<td>No</td>
</tr>
<tr>
<td>31–40</td>
<td>High</td>
<td>No</td>
<td>Bad</td>
<td>Yes</td>
</tr>
<tr>
<td>&gt; 40</td>
<td>Medium</td>
<td>No</td>
<td>Bad</td>
<td>Yes</td>
</tr>
<tr>
<td>&gt; 40</td>
<td>Low</td>
<td>Yes</td>
<td>Bad</td>
<td>Yes</td>
</tr>
<tr>
<td>&gt; 40</td>
<td>Low</td>
<td>Yes</td>
<td>Good</td>
<td>No</td>
</tr>
<tr>
<td>≤ 30</td>
<td>Medium</td>
<td>No</td>
<td>Bad</td>
<td>No</td>
</tr>
<tr>
<td>≤ 30</td>
<td>Medium</td>
<td>No</td>
<td>Good</td>
<td>No</td>
</tr>
<tr>
<td>31–40</td>
<td>Medium</td>
<td>No</td>
<td>Bad</td>
<td>Yes</td>
</tr>
<tr>
<td>≤ 30</td>
<td>Medium</td>
<td>Yes</td>
<td>Bad</td>
<td>Yes</td>
</tr>
<tr>
<td>≤ 30</td>
<td>Medium</td>
<td>Yes</td>
<td>Good</td>
<td>Yes</td>
</tr>
<tr>
<td>31–40</td>
<td>Medium</td>
<td>Yes</td>
<td>Good</td>
<td>Yes</td>
</tr>
<tr>
<td>&gt; 40</td>
<td>Medium</td>
<td>Yes</td>
<td>Bad</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Conditional Independence**

- \( X, Y, Z \) random variables
- \( X \) is conditionally independent of \( Y \), given \( Z \), if \( P(X|Y,Z) = P(X|Z) \)
- Equivalent to: \( P(X,Y|Z) = P(X|Z) * P(Y|Z) \)
- Example: people with longer arms read better
  - Confounding factor: age
    - Young child has shorter arms and lacks reading skills of adult
    - If age is fixed, observed relationship between arm length and reading skills disappears

**Derivation of Naïve Bayes Classifier**

- Simplifying assumption: all input attributes conditionally independent, given class
  - \( P(X_1, ..., X_k | C_i) = \prod_{m=1}^{k} P(X_m | C_i) \)
- Each \( P(X_k = x_k | C_i) \) can be estimated robustly
  - If \( X_k \) is categorical attribute
    - \( P(X_k = x_k | C_i) = #\text{records in } C_i \text{ that have value } x_k \text{ for } X_k \text{ divided by } #\text{records of class } C_i \text{ in training data set} \)
  - If \( X_k \) is continuous, we could discretize it
  - Problem: interval selection
    - Too many intervals: too few training cases per interval
    - Too few intervals: limited choices for decision boundary

**Estimating \( P(X_k=x_k | C_i) \) for Continuous Attributes without Discretization**

- \( P(X_k=x_k | C_i) \) computed based on Gaussian distribution with mean \( \mu \) and standard deviation \( \sigma \):
  - \( g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi \sigma}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \)
  - \( P(X_k = x_k | C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i}) \)
  - Estimate \( \mu_{C_i} \) from sample mean of attribute \( X_k \) for all training records of class \( C_i \)
  - Estimate \( \sigma_{C_i} \) similarly from sample

**Naïve Bayes Example**

- Classes:
  - \( C_1: \text{buys\_computer} = \text{yes} \)
  - \( C_2: \text{buys\_computer} = \text{no} \)
- Data sample \( x \)
  - \( \text{age} \leq 30 \)
  - \( \text{income} = \text{medium} \)
  - \( \text{student} = \text{yes} \)
  - \( \text{credit\_rating} = \text{fair} \)

**Naïve Bayesian Computation**

- Compute \( P(C_i) \) for each class
  - \( P(\text{buys\_computer} = \text{yes}) = 9/14 \approx 0.643 \)
  - \( P(\text{buys\_computer} = \text{no}) = 5/14 \approx 0.357 \)
- Compute \( P(X_k=x_k | C_i) \) for each class
  - \( \text{age} \leq 30 \) \( \text{income} = \text{low}, \text{student} = \text{no}, \text{credit\_rating} = \text{bad} \) \( \text{buys\_computer} = \text{yes} \)
  - \( \text{age} \geq 30 \) \( \text{income} = \text{low}, \text{student} = \text{no}, \text{credit\_rating} = \text{bad} \) \( \text{buys\_computer} = \text{yes} \)
  - Compute \( P(X|C) \) using the Naïve Bayes assumption
    - \( X = \text{age}, \text{income}, \text{student}, \text{credit\_rating} \)
    - \( P(\text{age} = 30, \text{income} = \text{low}, \text{student} = \text{no}) | \text{buys\_computer} = \text{yes} \) = 0.222
    - \( P(\text{income} = \text{medium}) \) \( \text{buys\_computer} = \text{yes} \) = 0.444
    - \( P(\text{student} = \text{yes}) \) \( \text{buys\_computer} = \text{yes} \) = 0.667
    - \( P(\text{credit\_rating} = \text{fair}) \) \( \text{buys\_computer} = \text{yes} \) = 0.667
- Compute final result \( P(C_i | x) \)\( P(C_1 | x) = 0.222 \cdot 0.444 \cdot 0.667 \cdot 0.667 \approx 0.094 \)
  - \( P(\text{age} = 30, \text{income} = \text{medium}, \text{student} = \text{yes}, \text{credit\_rating} = \text{fair}) = 0.094 \approx 0.094 \)

Therefore we predict \( \text{buys\_computer} = \text{yes} \) for input \( x = \{ \text{age} = 30, \text{income} = \text{medium}, \text{student} = \text{yes}, \text{credit\_rating} = \text{fair} \} \)
Zero-Probability Problem

- Naïve Bayesian prediction requires each conditional probability to be non-zero (why?)

\[ P(X_1, \ldots, X_n | C_i) = \prod P(X_i = x_i | C_i) \]

- Example: 1000 records for \( \text{buys\_computer}=\text{yes} \) with \( \text{income}=\text{low} \) (0), \( \text{income}=\text{medium} \) (990), and \( \text{income}=\text{high} \) (10)
  - For input with \( \text{income}=\text{low} \), conditional probability is zero
- Use Laplacian correction (or Laplace estimator) by adding 1 dummy record to each income level
  - \( \text{Prob}(\text{income}=\text{low}) = 1/1003 \)
  - \( \text{Prob}(\text{income}=\text{medium}) = 991/1003 \)
  - \( \text{Prob}(\text{income}=\text{high}) = 11/1003 \)
  - "Corrected" probability estimates close to their "uncorrected" counterparts, but none is zero

Naïve Bayesian Classifier: Comments

- Easy to implement
- Good results obtained in many cases
  - Robust to isolated noise points
  - Handles missing values by ignoring the instance during probability estimate calculations
  - Robust to irrelevant attributes
- Disadvantages
  - Assumption: class conditional independence, therefore loss of accuracy
  - Practically, dependencies exist among variables
- How to deal with these dependencies?

Probabilities

- Summary of elementary probability facts we have used already and/or will need soon
- Let \( X \) be a random variable as usual
- Let \( A \) be some predicate over its possible values
  - \( A \) is true for some values of \( X \), false for others
  - E.g., \( X \) is outcome of throw of a die, \( A \) could be “value is greater than 4”
- \( P(A) \) is the fraction of possible worlds in which \( A \) is true
  - \( P(\text{die value is greater than 4}) = 2/6 = 1/3 \)

Axioms

- \( 0 \leq P(A) \leq 1 \)
- \( P(\text{True}) = 1 \)
- \( P(\text{False}) = 0 \)
- \( P(A \lor B) = P(A) + P(B) - P(A \land B) \)

Theorems from the Axioms

- \( 0 \leq P(A) \leq 1, P(\text{True}) = 1, P(\text{False}) = 0 \)
- \( P(A \lor B) = P(A) + P(B) - P(A \land B) \)
- From these we can prove:
  - \( P(\neg A) = P(\neg A) = 1 - P(A) \)
  - \( P(A) = P(A \land B) + P(A \land \neg B) \)

Conditional Probability

- \( P(A|B) = \text{Fraction of worlds in which } B \text{ is true that also have } A \text{ true} \)

\( H \) = “Have a headache”
\( F \) = “Coming down with Flu”

\[
\begin{align*}
P(H) &= 1/10 \\
P(F) &= 1/40 \\
P(H|F) &= 1/2
\end{align*}
\]

“Headaches are rare and flu is rarer, but if you’re coming down with flu there’s a 50-50 chance you’ll have a headache.”
### Definition of Conditional Probability

\[ P(A \cap B) = \frac{P(A \mid B) \cdot P(B)}{P(B)} \]

Corollary: the Chain Rule

\[ P(A \cap B) = P(A \mid B) \cdot P(B) \]

### Multivalued Random Variables

- Suppose X can take on more than 2 values
- X is a random variable with *arity* k if it can take on exactly one value out of \( \{v_1, v_2, ..., v_k\} \)
- Thus
  \[ P(X = v_i \wedge X = v_j) = 0 \text{ if } i \neq j \]
  \[ P(X = v_1 \lor X = v_2 \lor ... \lor X = v_k) = 1 \]

### Easy Fact about Multivalued Random Variables

- Using the axioms of probability
  - \( 0 \leq P(A) \leq 1 \), \( P(\text{True}) = 1 \), \( P(\text{False}) = 0 \)
  - \( P(A \lor B) = P(A) + P(B) - P(A \land B) \)
- And assuming that X obeys
  \[ P(X = v_i \wedge X = v_j) = 0 \text{ if } i \neq j \]
  \[ P(X = v_i \lor X = v_j \lor ... \lor X = v_j) = 1 \]
- We can prove that
  \[ P(X = v_i \lor X = v_j \lor ... \lor X = v_j) = \sum_{j=1}^{k} P(X = v_j) \]
- And therefore:
  \[ \sum_{j=1}^{k} P(X = v_j) = 1 \]

### Useful Easy-to-Prove Facts

\[ P(A \mid B) + P(\sim A \mid B) = 1 \]

### The Joint Distribution

**Example: Boolean variables A, B, C**

Recipe for making a joint distribution of \( d \) variables:

1. Make a truth table listing all combinations of values of your variables (has \( 2^d \) rows for \( d \) Boolean variables).
The Joint Distribution

Recipe for making a joint distribution of \(d\) variables:

1. Make a truth table listing all combinations of values of your variables (has \(2^d\) rows for \(d\) Boolean variables).
2. For each combination of values, say how probable it is.

Example: Boolean variables A, B, C

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.30</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.05</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.10</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.05</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.05</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.10</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.25</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.10</td>
</tr>
</tbody>
</table>

Using the Joint Dist.

Once you have the JD you can ask for the probability of any logical expression involving your attribute

\[
P(E) = \sum_{\text{rows matching } E} P(\text{row})
\]

Example: P(Poor \& Male) = 0.4654

P(Poor) = 0.7604

Inference with the Joint Dist.

\[
P(E_1 | E_2) = \frac{P(E_1 \land E_2)}{P(E_2)} = \sum_{\text{rows matching } E_1 \land E_2} P(\text{row}) / \sum_{\text{rows matching } E_2} P(\text{row})
\]
Inference with the Joint Dist.

\[
P(E_1 | E_2) = \frac{P(E_1 \land E_2)}{P(E_2)} = \sum_{\text{rows matching } E_1 \land E_2} \frac{P(\text{row})}{\text{rows matching } E_2}
\]

\[
P(\text{Male | Poor}) = \frac{0.4654}{0.7604} = 0.612
\]

Joint Distributions

- Good news: Once you have a joint distribution, you can answer important questions that involve uncertainty.
- Bad news: Impossible to create joint distribution for more than about ten attributes because there are so many numbers needed when you build it.

What Would Help?

- Full independence
  - \(P(\text{gender=g \land hours\_worked=h \land wealth=w}) = P(\text{gender=g}) \times P(\text{hours\_worked=h}) \times P(\text{wealth=w})\)
  - Can reconstruct full joint distribution from a few marginals
- Full conditional independence given class value
  - Naïve Bayes
- What about something between Naïve Bayes and general joint distribution?

Bayesian Belief Networks

- Subset of the variables conditionally independent
- Graphical model of causal relationships
  - Represents dependency among the variables
  - Gives a specification of joint probability distribution

Bayesian Network Properties

- Each variable is conditionally independent of its non-descendents in the graph, given its parents
- Naïve Bayes as a Bayesian network:
Creating a Bayes Network

T: The lecture started
L: The lecturer arrives late
R: The lecture concerns data mining
M: The lecturer is Mike
S: It is snowing

Computing with Bayes Net

\[
P(T \land \neg R \land L \land \neg M \land S) = P(T \mid \neg R \land L \land \neg M \land S) \cdot P(\neg R \land L \land \neg M \land S)
\]

\[
P(T \mid \neg R \land L) \cdot P(\neg R \land L \land \neg M \land S)
\]

\[
P(T \mid \neg R \land \neg M) \cdot P(L \land \neg M \land S) \cdot P(\neg M \land S)
\]

\[
P(T \mid \neg R \land \neg M) \cdot P(L \land \neg M \land S) \cdot P(\neg M) \cdot P(S)
\]

\[
P(S) = 0.3
\]

\[
P(M) = 0.6
\]

\[
P(R \mid M) = 0.3
\]

\[
P(R \mid \neg M) = 0.6
\]

\[
P(T \mid L) = 0.3
\]

\[
P(T \mid \neg L) = 0.8
\]

\[
P(L \mid M \land S) = 0.05
\]

\[
P(L \mid M \land \neg S) = 0.1
\]

\[
P(L \mid \neg M \land S) = 0.1
\]

\[
P(L \mid \neg M \land \neg S) = 0.2
\]

Inference with Bayesian Networks

- Want to compute \( P(C_i \mid X=x) \)
  - Assume the output attribute \( Y \) node's parents are all input attribute nodes and all these input values are given
  - Then we have \( P(C_i \mid X=x) = P(C_i \mid \text{parents}(Y)) \), i.e., we can read it directly from CPT

- What if values are given only for a subset of attributes?
  - Can still compute it from the Bayesian network
  - But: exact inference of probabilities in general for an arbitrary Bayesian network is \( NP \)-hard
  - Solutions: probabilistic inference, trade precision for efficiency

Training Bayesian Networks

- Several scenarios:
  - Given both the network structure and all variables are observable: learn only the CPTs
  - Network structure known, some hidden variables: gradient descent (greedy hill-climbing) method, analogous to neural network learning
  - Network structure unknown, all variables observable: search through the model space to reconstruct network topology
  - Unknown structure, all hidden variables: No good algorithms known for this purpose

  * Ref.: D. Heckerman: Bayesian networks for data mining

Classification and Prediction Overview

- Introduction
- Decision Trees
- Statistical Decision Theory
- Nearest Neighbor
- Bayesian Classification
- Artificial Neural Networks
- Support Vector Machines (SVMs)
- Prediction
- Accuracy and Error Measures
- Ensemble Methods
Basic Building Block: Perceptron

For Example:
\[ f(x) = \text{sign}(b + \sum_{i} w_i x_i) \]

Representing Boolean Functions

- AND with two-input perceptron
  - \( b = 0.8, w_1 = w_2 = 0.5 \)
- OR with two-input perceptron
  - \( b = 0.3, w_1 = w_2 = 0.5 \)
- m-of-n function: true if at least \( m \) out of \( n \) inputs are true
  - All input weights 0.5, threshold weight \( b \) is set according to \( m, n \)
- Can also represent NAND, NOR
- What about XOR?

Gradient Descent

- If training records are not linearly separable, find best fit approximation.
  - Gradient descent to search the space of possible weight vectors
  - Basis for Backpropagation algorithm
- Consider un-thresholded perceptron (no sign function applied), i.e., \( u(x) = b + w \cdot x \)
- Measure training error by squared error
  \[ E(b, w) = \frac{1}{2} \sum_{(x, y) \in D} (y - u(x))^2 \]
  - \( D \) = training data
Gradient Descent Summary

- Epoch updating (aka batch mode)
  - Do until satisfied with model
    - Compute gradient over entire training set
    - Update all weights based on gradient
- Case updating (aka incremental mode, stochastic gradient descent)
  - Do until satisfied with model
    - For each training record
      - Compute gradient for this single training record
      - Update all weights based on gradient
- Case updating can approximate epoch updating arbitrarily close if $\eta$ is small enough
- Perceptron training rule and case updating might seem identical
  - Difference: error computation on thresholded vs. unthresholded output

Multilayer Feedforward Networks

- Use another perceptron to combine output of lower layer
  - What about linear units only?
    - Can only construct linear functions!
    - Need nonlinear component
      - $\text{sign}$ function: not differentiable (gradient descent!)
      - Use $\sigma(x) = \frac{1}{1 + e^{-x}}$

1-Hidden Layer Net Example

- Inputs: all input data attributes
  - Record fed simultaneously into the units of the input layer
  - Then weighted and fed simultaneously to a hidden layer
- Number of hidden layers is arbitrary, although usually only one
- Weighted outputs of the last hidden layer are the input to the units in the output layer, which emits the network’s prediction
  - Statistical point of view: neural networks perform nonlinear regression

Making Predictions

- We discussed gradient descent to find the best weights for a single perceptron using simple un-thresholded function
  - If sigmoid (or other differentiable) function is applied to weighted sum, use complete function for gradient descent
- Multiple perceptrons: optimize over all weights of all perceptrons
  - Problems: huge search space, local minima
- Backpropagation
  - Initialize all weights with small random values
  - Iterate many times
    - Compute gradient, starting at output and working back
    - Error of hidden units: how do we get the true output value? Use weighted sum of errors of each unit influenced by h.
    - Update all weights in the network

Overfitting

- When do we stop updating the weights?
  - Might overfit to training data
- Overfitting tends to happen in later iterations
  - Weights initially small random values
  - Weights all similar $\Rightarrow$ smooth decision surface
  - Surface complexity increases as weights diverge
- Preventing overfitting
  - Weight decay: decrease each weight by small factor during each iteration, or
  - Use validation data to decide when to stop iterating
Backpropagation Remarks

- Computational cost
  - Each iteration costs $O(|D| \cdot |w|)$, with $|D|$ training records and $|w|$ weights
  - Number of iterations can be exponential in $n$, the number of inputs (in practice often tens of thousands)
- Local minima can trap the gradient descent algorithm
  - Convergence guaranteed to local minimum, not global
- Backpropagation highly effective in practice
  - Many variants to deal with local minima issue
  - E.g., case updating might avoid local minimum

Defining a Network

1. Decide network topology
   - # input units, # hidden layers, # units in each hidden layer, # output units
2. Normalize input values for each attribute to $[0.0, 1.0]$:
   - Transform nominal and ordinal attributes: one input unit per domain value, each initialized to 0
   - Why not map the attribute to a single input with domain $[0.0, 1.0]$?
3. Output for classification task with $\geq 2$ classes: one output unit per class
4. Choose learning rate $\eta$
   - Too small: can take days instead of minutes to converge
   - Too large: diverges (MSE gets larger while the weights increase and usually oscillate)
   - Heuristic: set it to $1 / (\text{#training iterations})$
5. If model accuracy is unacceptable, re-train with different network topology, different set of initial weights, or different learning rate
   - Might need a lot of trial-and-error

Representational Power

- Boolean functions
  - Each can be represented by a 2-layer network
  - Number of hidden units can grow exponentially with number of inputs
    - Create hidden unit for each input record
    - Set its weights to activate only for that input
    - Implement output unit as OR gate that only activates for desired output patterns
- Continuous functions
  - Every bounded continuous function can be approximated arbitrarily close by a 2-layer network
  - Any function can be approximated arbitrarily close by a 3-layer network

Neural Network as a Classifier

- Weaknesses
  - Long training time
  - Many non-trivial parameters, e.g., network topology
  - Poor interpretability: What is the meaning behind learned weights and hidden units?
    - Note: hidden units are alternative representation of input values, capturing their relevant features
- Strengths
  - High tolerance to noisy data
  - Well-suited for continuous-valued inputs and outputs
  - Successful on a wide array of real-world data
  - Techniques exist for extraction of rules from neural networks

Classification and Prediction Overview

- Introduction
- Decision Trees
- Statistical Decision Theory
- Nearest Neighbor
- Bayesian Classification
- Artificial Neural Networks
- Support Vector Machines (SVMs)
- Prediction
- Accuracy and Error Measures
- Ensemble Methods
SVM—Support Vector Machines

- Newer and very popular classification method
- Uses a nonlinear mapping to transform the original training data into a higher dimension
- Searches for the optimal separating hyperplane (i.e., “decision boundary”) in the new dimension
- SVM finds this hyperplane using support vectors (“essential” training records) and margins (defined by the support vectors)

SVM—History and Applications

- Vapnik and colleagues (1992)
  - Groundwork from Vapnik & Chervonenkis’ statistical learning theory in 1960s
- Training can be slow but accuracy is high
  - Ability to model complex nonlinear decision boundaries (margin maximization)
- Used both for classification and prediction
- Applications: handwritten digit recognition, object recognition, speaker identification, benchmarking time-series prediction tests

Linear Classifiers

\[ f(x, w, b) = \text{sign}(w \cdot x + b) \]

How would you classify this data?

\[ f(x, w, b) = \text{sign}(w \cdot x + b) \]

How would you classify this data?
### Linear Classifiers

- If we made a small error in the location of the boundary, this gives us the least chance of causing a misclassification.
- Model is immune to removal of any non-support-vector data records.
- There is some theory (using VC dimension) that is related to (but not the same as) the proposition that this is a good thing.
- Empirically it works very well.

### Why Maximum Margin?

1. **Plus-plane** = \( \{ x : w \cdot x + b = +1 \} \)
2. **Minus-plane** = \( \{ x : w \cdot x + b = -1 \} \)
3. Classify as:
   - \( +1 \) if \( w \cdot x + b \geq 1 \)
   - \( -1 \) if \( w \cdot x + b \leq -1 \)
   - **what** if \( -1 < w \cdot x + b < 1 \)?
What Are the SVM Constraints?

- Consider \( n \) training records \((x(k), y(k))\), where \( y(k) = +/- 1 \)
- How many constraints will we have?
- What should they be?

Finding the Maximum Margin

- How do we find \( w \) and \( b \) such that the margin is maximized and all training records are in the correct zone for their class?
- Solution: Quadratic Programming (QP)
- QP is a well-studied class of optimization algorithms to maximize a quadratic function of some real-valued variables subject to linear constraints.
  - There exist algorithms for finding such constrained quadratic optima efficiently and reliably.

Putting It All Together

- We have so far:
  - \( w \cdot x' + b = +1 \)
  - \( x' = x + \lambda w \)
  - \( |x' - x| = M \)
- Derivation:
  - \( w (x' + \lambda w) + b = +1 \), hence \( w \cdot x' + b + \lambda w \cdot w = 1 \)
  - This implies \( \lambda w \cdot w = 2 \), i.e., \( \lambda = 2 / w \cdot w \)
  - Since \( M = |x' - x| = |\lambda w| = \lambda |w| = \lambda (w \cdot w)^{0.5} \)
  - We obtain \( M = 2 (w \cdot w)^{0.5} / w \cdot w = 2 / (w \cdot w)^{0.5} \)

Quadratic Programming

Find \( \arg \max_w c + d^T u + \frac{u' Ru}{2} \) \( \text{Quadratic criterion} \)

Subject to \( a_1 u_1 + a_2 u_2 + \ldots + a_n u_n \leq b_1 \)
\( : \)
\( a_i u_1 + a_2 u_2 + \ldots + a_n u_n \leq b_i \)
\( n \) additional linear inequality constraints

And subject to \( a_{1+n} u_1 + a_{2+n} u_2 + \ldots + a_{n+n} u_n = b_{(n+1)} \)
\( : \)
\( a_{1+n+2} u_1 + a_{2+n+2} u_2 + \ldots + a_{n+n+2} u_n = b_{(n+2)} \)
\( e \) additional linear equality constraints

Computing Margin Width

- Choose arbitrary point \( x' \) on minus-plane
- Let \( x' \) be the point in plus-plane closest to \( x \)
- Since vector \( w \) is perpendicular to these planes, it holds that \( x' = x + \lambda w \), for some value of \( \lambda \).
What Are the SVM Constraints?

- What is the quadratic optimization criterion?
  - Minimize \( w \cdot w \)
- Consider \( n \) training records \((x(k), y(k))\), where \( y(k) = +/- 1 \)
- How many constraints will we have? \( n \).
- What should they be?
  For each \( 1 \leq k \leq n \):
    \[
    w \cdot x(k) + b \geq 1, \quad \text{if } y(k) = 1
    \]
    \[
    w \cdot x(k) + b \leq -1, \quad \text{if } y(k) = -1
    \]
- Inequalities for training records are not satisfiable by any \( w \) and \( b \)

Problem: Classes Not Linearly Separable

- Inequalities for training records are not satisfiable by any \( w \) and \( b \)

Solution 1?

- Find minimum \( w \cdot w \), while also minimizing number of training set errors
  - Not a well-defined optimization problem (cannot optimize two things at the same time)

Solution 2?

- Minimize \( w \cdot w + C \cdot \text{#trainSetErrors} \)
  - \( C \) is a tradeoff parameter
- Problems:
  - Cannot be expressed as QP, hence finding solution might be slow
  - Does not distinguish between disastrous errors and near misses

Solution 3

- Minimize \( w \cdot w + C \cdot \text{distance of error records to their correct place} \)
  - This works!
- But still need to do something about the unsatisfiable set of inequalities

What Are the SVM Constraints?

- Consider \( n \) training records \((x(k), y(k))\), where \( y(k) = +/- 1 \)
- How many constraints will we have? \( n \).
- What should they be?
  For each \( 1 \leq k \leq n \):
    \[
    w \cdot x(k) + b \geq 1 - \varepsilon_k, \quad \text{if } y(k) = 1 \]
    \[
    w \cdot x(k) + b \leq -1 + \varepsilon_k, \quad \text{if } y(k) = -1 \]
  - \( \varepsilon_k \geq 0 \)

Facts About the New Problem Formulation

- Original QP formulation had \( d+1 \) variables
  - \( w_1, w_2, \ldots, w_d \) and \( b \)
- New QP formulation has \( d+1+n \) variables
  - \( w_1, w_2, \ldots, w_d \) and \( b \)
  - \( \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n \)
- \( C \) is a new parameter that needs to be set for the SVM
  - Controls tradeoff between paying attention to margin size versus misclassifications

Effect of Parameter \( C \)

An Equivalent QP (The “Dual”)

Maximize
\[
\sum_{k=1}^{n} \alpha_k - \frac{1}{2} \sum_{k=1}^{n} \sum_{l \neq k} \alpha_k \alpha_l y(k) \cdot y(l) \cdot x(k) \cdot x(l)
\]
Subject to these constraints:
\[
\forall k: 0 \leq \alpha_k \leq C \quad \sum_{k=1}^{n} \alpha_k y(k) = 0
\]

Then define:
\[
w = \sum_{k=1}^{n} \alpha_k y(k) \cdot x(k)
\]
\[
b = \text{AVG} \left\{ \frac{1}{y(k)} - x(k) \cdot w \right\}
\]

Then classify with:
\[
f(x, w, b) = \text{sign}(w \cdot x + b)
\]

Important Facts

- Dual formulation of QP can be optimized more quickly, but result is equivalent
- Data records with \( \alpha_k > 0 \) are the support vectors
  - Those with \( 0 < \alpha_k < C \) lie on the plus- or minus-plane
  - Those with \( \alpha_k = C \) are on the wrong side of the classifier boundary (have \( \varepsilon_k > 0 \))
- Computation for \( w \) and \( b \) only depends on those records with \( \alpha_k > 0 \), i.e., the support vectors
- Alternative QP has another major advantage, as we will see now...

Easy To Separate

What would SVMs do with this data?

Easy To Separate

Not a big surprise
Harder To Separate

What can be done about this?

Now Separation Is Easy Again

Corresponding “Planes” in Original Space

Common SVM Basis Functions

- Polynomial of attributes $X_1, \ldots, X_d$ of certain max degree, e.g., $X_2 + X_3 X_4 + X_4^2$
- Radial basis function
  - Symmetric around center, i.e., $\text{KernelFunction}(|X - c|) / \text{kernelWidth}$
- Sigmoid function of $X$, e.g., hyperbolic tangent
- Let $\Phi(x)$ be the transformed input record
  - Previous example: $\Phi(x) = (x, x^2)$

Quadratic Basis Functions

- Number of terms (assuming $d$ input attributes):
  - $(d+2)\text{-choose-2}$
  - $= \frac{(d+2)(d+1)}{2}$
  - $= \frac{d^2}{2}$
- Why did we choose this specific transformation?
Dual QP With Basis Functions

Maximize \[ \sum_{k=1}^{n} a_k - \frac{1}{2} \sum_{k=1}^{n} a_k \cdot y(k) \cdot y(l) \cdot \Phi(x(k)) \cdot \Phi(x(l)) \]

Subject to these constraints:
\[ \forall k : 0 \leq a_k \leq C \]
\[ \sum_{k=1}^{n} a_k y(k) = 0 \]

Then define:
\[ w = \sum_{k=1}^{n} a_k \cdot y(k) \cdot \Phi(x(k)) \]

Then classify with:
\[ f(x, w, b) = \text{sign}(w \cdot \Phi(x) + b) \]

Computation Challenge

- Input vector \( x \) has \( d \) components (its \( d \) attribute values)
- The transformed input vector \( \Phi(x) \) has \( d^2/2 \) components
- Hence computing \( \Phi(x(k)) \cdot \Phi(x(l)) \) now costs order \( d^2/2 \) instead of order \( d \) operations (additions, multiplications)
- ... or is there a better way to do this?
  - Take advantage of properties of certain transformations

Computational Challenges

- Input vector \( x \) has \( d \) components (its \( d \) attribute values)
- The transformed input vector \( \Phi(x) \) has \( d^2/2 \) components
- Hence computing \( \Phi(x(k)) \cdot \Phi(x(l)) \) now costs order \( d^2/2 \) instead of order \( d \) operations (additions, multiplications)
- ... or is there a better way to do this?
  - Take advantage of properties of certain transformations

Any Other Computation Problems?

\[ w = \sum_{k=1}^{n} a_k \cdot y(k) \cdot \Phi(x(k)) \]
\[ b = \text{AVG} \begin{cases} 
\frac{1}{y(k)} - \Phi(x(k)) \cdot w 
\end{cases} 
\]

- What about computing \( w \)?
  - Finally need \( f(x, w, b) = \text{sign}(w \cdot \Phi(x) + b) \):
    \[ w \cdot \Phi(x) = \sum_{k=1}^{n} a_k \cdot y(k) \cdot \Phi(x(k)) \cdot \Phi(x) \]
  - Can be computed using the same trick as before
  - Can apply the same trick again to \( b \), because
    \[ \Phi(x(k)) \cdot w = \sum_{j=1}^{n} a_j \cdot y(j) \cdot \Phi(x(k)) \cdot \Phi(x(j)) \]
SVM Kernel Functions

- For which transformations, called kernels, does the same trick work?
- Polynomial: \( K(a, b) = (a \cdot b + 1)^q \)
- Radial-Basis-style (RBF):
  \[ K(a, b) = \exp\left(-\frac{(a - b)^2}{2\sigma^2}\right) \]
  \( \sigma, \kappa \) and \( \delta \) are magic parameters that must be chosen by a model selection method.
- Neural-net-style sigmoidal:
  \[ K(a, b) = \tanh(\kappa \cdot a \cdot b - \delta) \]

Overfitting

- With the right kernel function, computation in high dimensional transformed space is no problem
- But what about overfitting? There are so many parameters...
- Usually not a problem, due to maximum margin approach
  - Only the support vectors determine the model, hence SVM complexity depends on number of support vectors, not dimensions (still, in higher dimensions there might be more support vectors)
  - Minimizing \( w \) discourages extremely large weights, which smoothes the function (recall weight decay for neural networks!)

Multi-Class Classification

- SVMs can only handle two-class outputs (i.e. a categorical output variable with arity 2).
- What can be done?
- Answer: with output arity \( N \), learn \( N \) SVM’s
  - SVM 1 learns “Output==1” vs “Output != 1”
  - SVM 2 learns “Output==2” vs “Output != 2”
  - ...
  - SVM \( N \) learns “Output==N” vs “Output != N”
- To predict the output for a new input, just predict with each SVM and find out which one puts the prediction the furthest into the positive region.

SVM vs. Neural Network

- SVM
  - Relatively new concept
  - Deterministic algorithm
  - Nice Generalization properties
  - Hard to train – learned in batch mode using quadratic programming techniques
  - Using kernels can learn very complex functions
- Neural Network
  - Relatively old
  - Nondeterministic algorithm
  - Generalizes well but doesn’t have strong mathematical foundation
  - Can easily be learned in incremental fashion
  - To learn complex functions—use multilayer perceptron (not that trivial)
Classification and Prediction Overview

- Introduction
- Decision Trees
- Statistical Decision Theory
- Nearest Neighbor
- Bayesian Classification
- Artificial Neural Networks
- Support Vector Machines (SVMs)
- Prediction
- Accuracy and Error Measures
- Ensemble Methods

What Is Prediction?

- Essentially the same as classification, but output is continuous, not discrete
  - Construct a model
  - Use model to predict continuous output value for a given input
- Major method for prediction: regression
  - Many variants of regression analysis in statistics literature; not covered in this class
- Neural network and k-NN can do regression “out-of-the-box”
- SVMs for regression exist
- What about trees?

Regression Trees and Model Trees

- Regression tree: proposed in CART system (Breiman et al. 1984)
  - CART: Classification And Regression Trees
  - Each leaf stores a continuous-valued prediction
    - Average output value for the training records that reach the leaf
- Model tree: proposed by Quinlan (1992)
  - Each leaf holds a regression model—a multivariate linear equation
- Training: like for classification trees, but uses variance instead of purity measure for selecting split predicates

Classifier Accuracy Measures

<table>
<thead>
<tr>
<th>True class</th>
<th>Predicted class</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>buy_computer = yes</td>
<td>buy_computer = no</td>
</tr>
<tr>
<td>buy_computer = yes</td>
<td>6954</td>
<td>46</td>
</tr>
<tr>
<td>buy_computer = no</td>
<td>412</td>
<td>2588</td>
</tr>
<tr>
<td>total</td>
<td>7366</td>
<td>2634</td>
</tr>
</tbody>
</table>

- Accuracy of a classifier M, acc(M): percentage of test records that are correctly classified by M
  - Error rate (misclassification rate) of M = 1 – acc(M)
  - Given m classes, CM[i][j], an entry in a confusion matrix, indicates # of records in class i that are labeled by the classifier as class j

<table>
<thead>
<tr>
<th>C_i</th>
<th>C_j</th>
</tr>
</thead>
<tbody>
<tr>
<td>True positive</td>
<td>False negative</td>
</tr>
<tr>
<td>False positive</td>
<td>True negative</td>
</tr>
</tbody>
</table>

Precision and Recall

- Precision: measure of exactness
  - t-pos / (t-pos + f-pos)
- Recall: measure of completeness
  - t-pos / (t-pos + f-neg)
- F-measure: combination of precision and recall
  - 2 * precision * recall / (precision + recall)
- Note: Accuracy = (t-pos + t-neg) / (t-pos + t-neg + f-pos + f-neg)
Limitation of Accuracy

- Consider a 2-class problem
  - Number of Class 0 examples = 9990
  - Number of Class 1 examples = 10
  - If model predicts everything to be class 0, accuracy is 9990/10000 = 99.9 %
  - Accuracy is misleading because model does not detect any class 1 example
- Always predicting the majority class defines the baseline
  - A good classifier should do better than baseline

Cost-Sensitive Measures: Cost Matrix

<table>
<thead>
<tr>
<th>ACTUAL CLASS</th>
<th>PREDICTED CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class=Yes</td>
<td>C(Yes</td>
</tr>
<tr>
<td>Class=No</td>
<td>C(Yes</td>
</tr>
</tbody>
</table>

C(i | j): Cost of misclassifying class j example as class i

Computing Cost of Classification

<table>
<thead>
<tr>
<th>ACTUAL CLASS</th>
<th>PREDICTED CLASS</th>
<th>Cost Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class=Yes</td>
<td>+ + 100 - -</td>
<td></td>
</tr>
<tr>
<td>Class=No</td>
<td>+ - 100 - -</td>
<td></td>
</tr>
</tbody>
</table>

Accuracy = 80%
Cost = 3910

Model M₁

<table>
<thead>
<tr>
<th>ACTUAL CLASS</th>
<th>PREDICTED CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ + 150 40</td>
<td></td>
</tr>
<tr>
<td>- - 60 250</td>
<td></td>
</tr>
</tbody>
</table>

Accuracy = 90%
Cost = 4255

Model M₂

<table>
<thead>
<tr>
<th>ACTUAL CLASS</th>
<th>PREDICTED CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ + 250 45</td>
<td></td>
</tr>
<tr>
<td>- - 5 200</td>
<td></td>
</tr>
</tbody>
</table>

Prediction Error Measures

- Continuous output: it matters how far off the prediction is from the true value
- Loss function: distance between y and predicted value y'
  - Absolute error: |y - y’|
  - Squared error: (y - y’)^2
- Test error (generalization error): average loss over the test set
  - Mean absolute error: Mean squared error:
  - Relative absolute error: Relative squared error:
  - Squared-error exaggerates the presence of outliers

Evaluating a Classifier or Predictor

- Holdout method
  - The given data set is randomly partitioned into two sets
    - Training set (e.g., 2/3) for model construction
    - Test set (e.g., 1/3) for accuracy estimation
  - Can repeat holdout multiple times
    - Accuracy = avg. of the accuracies obtained
- Cross-validation (k-fold, where k = 10 is most popular)
  - Randomly partition data into k mutually exclusive subsets, each approximately equal size
  - In i-th iteration, use Dᵢ as test set and others as training set
  - Leave-one-out: k folds where k = # of records
    - Expensive, often results in high variance of performance metric

Learning Curve

- Accuracy versus sample size
- Effect of small sample size:
  - Bias in estimate
  - Variance of estimate
- Helps determine how much training data is needed
  - Still need to have enough test and validation data to be representative of distribution
ROC (Receiver Operating Characteristic)

- Developed in 1950s for signal detection theory to analyze noisy signals
  - Characterizes trade-off between positive hits and false alarms
- ROC curve plots T-Pos rate (y-axis) against F-Pos rate (x-axis)
- Performance of each classifier is represented as a point on the ROC curve
  - Changing the threshold of the algorithm, sample distribution or cost matrix changes the location of the point

ROC Curve

- 1-dimensional data set containing 2 classes (positive and negative)
  - Any point located at x > t is classified as positive

Diagonal Line for Random Guessing

- Classify a record as positive with fixed probability p, irrespective of attribute values
- Consider test set with a positive and b negative records
- True positives: p*a, hence true positive rate = (p*a)/a = p
  - False positives: p*b, hence false positive rate = (p*b)/b = p
- For every value 0 ≤ p ≤ 1, we get point (p,p) on ROC curve

How to Construct an ROC curve

- Use classifier that produces posterior probability \( P(+|x) \) for each test record x
- Sort records according to \( P(+|x) \) in decreasing order
- Apply threshold at each unique value of \( P(+|x) \)
  - Count number of TP, FP, TN, FN at each threshold
    - TP rate, TPR = TP/(TP+FN)
    - FP rate, FPR = FP/(FP+TN)

Using ROC for Model Comparison

- Neither model consistently outperforms the other
  - M1 better for small FPR
  - M2 better for large FPR
- Area under the ROC curve
  - Ideal: area = 1
  - Random guess: area = 0.5
How To Construct An ROC Curve

Confidence Interval for Accuracy

• Classification can be regarded as a Bernoulli trial
  – A Bernoulli trial has 2 possible outcomes, “correct” or “wrong” for classification
  – Collection of Bernoulli trials has a Binomial distribution
    • Probability of getting c correct predictions if model accuracy is p (=probability to get a single prediction right):
      \[
      \binom{n}{c} p^c (1-p)^{n-c}
      \]
    • Given c, or equivalently, ACC = c / n and n (#test records), can we predict p, the true accuracy of the model?

Confidence Interval for Accuracy

• Consider a model that produces an accuracy of 80% when evaluated on 100 test instances
  – n = 100, ACC = 0.8
  – Let 1-\alpha = 0.95 (95% confidence)
  – From probability table, \( Z_{0.025} = 1.96 \)

\[
\begin{array}{c|cccccc}
  N & 50 & 100 & 500 & 1000 & 5000 \\
  \hline
  p(\text{lower}) & 0.670 & 0.711 & 0.763 & 0.774 & 0.789 \\
  p(\text{upper}) & 0.888 & 0.866 & 0.833 & 0.824 & 0.811 \\
  p & 2n \cdot \text{ACC} + Z^2_{1-\alpha} & \sqrt{Z^2_{1-\alpha} + 4n \cdot \text{ACC} - 4n \cdot \text{ACC}^2} & \text{2(n + Z^2_{1-\alpha})} \\
  \hline
  1-\alpha & 0.99 & 2.58 & 0.98 & 2.33 & 0.95 & 1.96 & 0.90 & 1.65 \\
\end{array}
\]

Test of Significance

• Given two models:
  – Model M1: accuracy = 85%, tested on 30 instances
  – Model M2: accuracy = 75%, tested on 5000 instances
• Can we say M1 is better than M2?
  – How much confidence can we place on accuracy of M1 and M2?
  – Can the difference in accuracy be explained as a result of random fluctuations in the test set?

Confidence Interval for Accuracy

• Binomial distribution for X = “number of correctly classified test records out of n”
  – \( E(X) = n \cdot \text{ACC} \), \( Var(X) = n \cdot \text{ACC} \cdot (1 - \text{ACC}) / n \n  
  If n is large, Binomial distribution is closely approximated by normal distribution with mean and variance:
  – ACC has a normal distribution with mean p, variance \( p(1-p)/n \)
  
  \[
  P \left( \frac{Z_{\alpha/2}}{\sqrt{1/n \cdot (p - \text{ACC})}} < Z < \frac{Z_{1-\alpha/2}}{\sqrt{1/n \cdot (p - \text{ACC})}} \right) = 1-\alpha
  \]

• Confidence Interval for p:

\[
\begin{aligned}
  p & = \frac{2n \cdot \text{ACC} + Z_{1-\alpha}^2 \pm \sqrt{Z_{1-\alpha}^2 + 4n \cdot \text{ACC} - 4n \cdot \text{ACC}^2}}{2(n + Z_{1-\alpha}^2)} \\
  1-\alpha & = 0.99, Z = 2.58 \\
  1-\alpha & = 0.95, Z = 1.96 \\
  1-\alpha & = 0.90, Z = 1.65
\end{aligned}
\]

Comparing Performance of Two Models

• Given two models M1 and M2, which is better?
  – M1 is tested on \( D_1 \) (size = \( n_1 \)), found error rate = \( e_1 \)
  – M2 is tested on \( D_2 \) (size = \( n_2 \)), found error rate = \( e_2 \)
  – Assume \( D_1 \) and \( D_2 \) are independent
  – If \( n_1 \) and \( n_2 \) are sufficiently large, then
  \[
  \text{err}_1 \sim N(\mu_1, \sigma_1) \\
  \text{err}_2 \sim N(\mu_2, \sigma_2)
  \]
  – Estimate: \( \hat{\mu}_i = e_i \) and \( \hat{\sigma}^2_i = e_i(1-e_i) / n_i \)
Testing Significance of Accuracy Difference

• Consider random variable \( d = \text{err}_1 - \text{err}_2 \)
  – Since \( \text{err}_1, \text{err}_2 \) are normally distributed, so is their difference
  – Hence \( d \sim N(\mu_d, \sigma_d) \) where \( \mu_d \) is the true difference

• Estimator for \( \mu_d \):
  – \( \hat{E}[d] = \hat{E}[\text{err}_1 - \text{err}_2] = \hat{E}[\text{err}_1] - \hat{E}[\text{err}_2] \approx e_1 - e_2 \)
  – Since \( D_1 \) and \( D_2 \) are independent, variance adds up:
  – At \( (1-\alpha) \) confidence level, \( d_j = \hat{E}[d] \pm Z_{\alpha/2} \hat{\sigma}_d \)

An Illustrative Example

• Given: \( M1: n_1 = 30, e_1 = 0.15 \)
  \( M2: n_2 = 5000, e_2 = 0.25 \)
  \( \hat{E}[d] = |e_1 - e_2| = 0.1 \)
  \( 2\)-sided test: \( d_j = 0 \) versus \( d_j \neq 0 \)
  – At 95% confidence level, \( Z_{\alpha/2} = 1.96 \)
  – Interval contains zero, hence difference may not be statistically significant
  – But: may reject null hypothesis \( (d_j \neq 0) \) at lower confidence level

Significance Test for K-Fold Cross-Validation

• Each learning algorithm produces \( K \) models:
  – \( L1 \) produces \( M11, M12, \ldots, M1K \)
  – \( L2 \) produces \( M21, M22, \ldots, M2K \)
• Both models are tested on the same test sets \( D_1, D_2, \ldots, D_K \)
  – For each test set, compute \( d_j = e_{1j} - e_{2j} \)
  – For large enough \( K \), \( d_j \) is normally distributed with mean \( \mu_d \) and variance \( \sigma_d \)
  – Estimate:
    \[ \hat{\sigma}_d^2 = \frac{1}{K(K-1)} \sum (d_j - \bar{d})^2 \]
    \( \bar{d} = \frac{1}{K} \sum d_j \)
    \( d_j = \bar{d} \pm t_{\alpha/2, K-1} \hat{\sigma}_d \)

Classification and Prediction Overview

• Introduction
  • Decision Trees
  • Statistical Decision Theory
  • Nearest Neighbor
  • Bayesian Classification
  • Artificial Neural Networks
  • Support Vector Machines (SVMs)
• Prediction
  • Accuracy and Error Measures
  • Ensemble Methods

Ensemble Methods

• Construct a set of classifiers from the training data
• Predict class label of previously unseen records by aggregating predictions made by multiple classifiers

General Idea
Why Does It Work?

- Consider 2-class problem
- Suppose there are 25 base classifiers
  - Each classifier has error rate $\varepsilon = 0.35$
  - Assume the classifiers are independent
- Return majority vote of the 25 classifiers
  - Probability that the ensemble classifier makes a wrong prediction:
    \[ \sum_{i=1}^{25} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06 \]

Model Averaging and Bias-Variance Tradeoff

- Single model: lowering bias will usually increase variance
  - “Smother” model has lower variance but might not model function well enough
- Ensembles can overcome this problem
  1. Let models overfit
     * Low bias, high variance
  2. Take care of the variance problem by averaging many of these models
- This is the basic idea behind bagging

Bagging: Bootstrap Aggregation

- Given training set with $n$ records, sample $n$ records randomly with replacement
- Train classifier for each bootstrap sample
- Note: each training record has probability $1 - (1 - 1/n)^n$ of being selected at least once in a sample of size $n$

Bagged Trees

- Create $k$ trees from training data
  - Bootstrap sample, grow large trees
- Design goal: independent models, high variability between models
- Ensemble prediction = average of individual tree predictions (or majority vote)
- Works the same way for other classifiers

Typical Result
Bagging Challenges

- Ideal case: all models independent of each other
- Train on independent data samples
  - Problem: limited amount of training data
  - Training set needs to be representative of data distribution
  - Bootstrap sampling allows creation of many "almost" independent training sets
- Diversify models, because similar sample might result in similar tree
  - Random Forest: limit choice of split attributes to small random subset of attributes (new selection of subset for each node) when training tree
  - Use different model types in same ensemble: tree, ANN, SVM, regression models

Additive Grove

- Ensemble technique for predicting continuous output
- Instead of individual trees, train additive models
  - Prediction of single Grove model = sum of tree predictions
  - Prediction of ensemble = average of individual Grove predictions
- Combines large trees and additive models
  - Challenge: how to train the additive models without having the first trees fit the training data too well
    - Next tree is trained on residuals of previously trained trees in same Grove model
    - If previously trained trees capture training data too well, next tree is mostly trained on noise

Training Groves

Typical Grove Performance

- Root mean squared error
  - Lower is better
- Horizontal axis: tree size
  - Fraction of training data when to stop splitting
- Vertical axis: number of trees in each single Grove model
- 100 bagging iterations
Boosting

- Iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records
- Initially, all \( n \) records are assigned equal weights
- Record weights may change at the end of each boosting round

Boosting Details

- Weight update: 
  \[ w_{j}^{(i+1)} = \frac{w_{j}^{(i)} \cdot \frac{1}{1 - \epsilon_{j}}}{Z_{i}} \]
  if \( C(x_{j}) = y_{j} \)
  \[ w_{j}^{(i+1)} = \frac{w_{j}^{(i)} \cdot 1}{Z_{i}} \]
  if \( C(x_{j}) \neq y_{j} \)

  where \( Z_{i} \) is the normalization factor

- Weights initialized to \( \frac{1}{n} \)
- If any intermediate rounds produce error rate higher than 50%, the weights are reverted back to \( \frac{1}{n} \) and the resampling procedure is repeated
- Final classification:
  \[ C^{*}(x) = \arg \max_{y} \sum_{j=1}^{n} \alpha_{j} \delta(C_{j}(x) = y) \]

Example: AdaBoost

- Base classifiers: \( C_{1}, C_{2}, \ldots, C_{T} \)
- Error rate (\( n \) training records, \( w_{i} \) are weights that sum to 1):
  \[ \epsilon_{i} = \frac{1}{n} \sum_{j=1}^{n} w_{j} \delta(C_{j}(x_{j}) \neq y_{j}) \]
- Importance of a classifier:
  \[ \alpha_{i} = \ln \left( \frac{1 - \epsilon_{i}}{\epsilon_{i}} \right) \]

Illustrating AdaBoost

Note: The numbers appear to be wrong, but they convey the right idea...
### Bagging vs. Boosting

- **Analogy**
  - Bagging: diagnosis based on multiple doctors’ majority vote
  - Boosting: weighted vote, based on doctors’ previous diagnosis accuracy

- **Sampling procedure**
  - Bagging: records have same weight; easy to train in parallel
  - Boosting: weights record higher if model predicts it wrong; inherently sequential process

- **Overfitting**
  - Bagging robust against overfitting
  - Boosting susceptible to overfitting: make sure individual models do not overfit

- **Accuracy**
  - Usually significantly better than a single classifier
  - Best boosted model often better than best bagged model

- **Additive Grove**
  - Combines strengths of bagging and boosting (additive models)
  - Shown empirically to make better predictions on many data sets
  - Training more tricky, especially when data is very noisy

### Classification/Prediction Summary

- **Forms of data analysis** that can be used to train models from data and then make predictions for new records
- Effective and scalable methods have been developed for decision tree induction, Naive Bayesian classification, Bayesian networks, rule-based classifiers, Backpropagation, Support Vector Machines (SVM), nearest neighbor classifiers, and many other classification methods
- Regression models are popular for prediction. Regression trees, model trees, and ANNs are also used for prediction.

- **K-fold cross-validation** is a popular method for accuracy estimation, but determining accuracy on large test set is equally accepted
  - If test sets are large enough, a significance test for finding the best model is not necessary

- **Area under ROC curve** and many other common performance measures exist

- **Ensemble methods** like bagging and boosting can be used to increase overall accuracy by learning and combining a series of individual models
  - Often state-of-the-art in prediction quality, but expensive to train, store, use

- No single method is superior over all others for all data sets
  - Issues such as accuracy, training and prediction time, robustness, interpretability, and scalability must be considered and can involve trade-offs