Outline

1. What is Classification? ✓
2. k Nearest Neighbors ✓
3. Naïve Bayes ✓
4. Decision Trees
5. Evaluating Classification
6. The Overfitting Problem
7. Rule Learning
8. Other Classification Approaches
9. Parameter Tunining
Lazy vs. Eager Learning

- Both k-NN and Naïve Bayes are “lazy” methods
- They do not build an explicit model!
  - “learning” is only performed on demand for unseen records

<table>
<thead>
<tr>
<th>Tid</th>
<th>Attrib1</th>
<th>Attrib2</th>
<th>Attrib3</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Large</td>
<td>125K</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>Medium</td>
<td>100K</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>Small</td>
<td>70K</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Medium</td>
<td>120K</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>No</td>
<td>Large</td>
<td>95K</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>Medium</td>
<td>80K</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Large</td>
<td>220K</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>No</td>
<td>Small</td>
<td>85K</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>No</td>
<td>Medium</td>
<td>75K</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>No</td>
<td>Small</td>
<td>90K</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tid</th>
<th>Attrib1</th>
<th>Attrib2</th>
<th>Attrib3</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>No</td>
<td>Small</td>
<td>55K</td>
<td>?</td>
</tr>
<tr>
<td>12</td>
<td>Yes</td>
<td>Medium</td>
<td>50K</td>
<td>?</td>
</tr>
<tr>
<td>13</td>
<td>Yes</td>
<td>Large</td>
<td>110K</td>
<td>?</td>
</tr>
<tr>
<td>14</td>
<td>No</td>
<td>Small</td>
<td>95K</td>
<td>?</td>
</tr>
<tr>
<td>15</td>
<td>No</td>
<td>Large</td>
<td>87K</td>
<td>?</td>
</tr>
</tbody>
</table>
Today: Eager Learning

• Actually, we have two goals
  – classify unseen instances
  – learn a model

• Model
  – explains how to classify unseen instances
  – sometimes: interpretable by humans
Decision Tree Classifiers

Training Data

<table>
<thead>
<tr>
<th>Tid</th>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Cheat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Single</td>
<td>125K</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>Married</td>
<td>100K</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>Single</td>
<td>70K</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Married</td>
<td>120K</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>No</td>
<td>Divorced</td>
<td>95K</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>Married</td>
<td>60K</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Divorced</td>
<td>220K</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>No</td>
<td>Single</td>
<td>85K</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>No</td>
<td>Married</td>
<td>75K</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>No</td>
<td>Single</td>
<td>90K</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Model: Decision Tree

Splitting Attributes

Terminal node = decision
Another Example of a Possible Decision Tree

There can be more than one tree that fits the same data!
Decision Boundary

• Border line between two neighboring regions of different classes is known as decision boundary

• Decision boundary is parallel to axes because test condition involves a single attribute at-a-time
Applying a Decision Tree to Test Data

Start from the root of the tree.

Refund

Test Data

<table>
<thead>
<tr>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Cheat</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Married</td>
<td>80K</td>
<td>?</td>
</tr>
</tbody>
</table>

Assign Cheat to “No”
Decision Tree Induction

• How to learn a decision Tree from test data?
• Finding an optimal decision tree is NP-hard
• Tree building algorithms use a greedy, top-down, recursive partitioning strategy to induce a reasonable solution
• The algorithms split the records based on an attribute test that optimizes a certain criterion
• Many different algorithms have been proposed:
  – Hunt’s Algorithm
  – ID3
  – CHAID
  – C4.5
Let $D_t$ be the set of training records that reach a node $t$.

General Procedure:
- If $D_t$ contains only records that belong to the same class $y_t$, then $t$ is a leaf node labeled as $y_t$.
- If $D_t$ contains records that belong to more than one class, use an attribute test to split the data into smaller subsets.
- Recursively apply the procedure to each subset.
- If $D_t$ is an empty set, then $t$ is a leaf node labeled by the default class, $y_d$. 

<table>
<thead>
<tr>
<th>Tid</th>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Cheat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Single</td>
<td>125K</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>Married</td>
<td>100K</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>Single</td>
<td>70K</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Married</td>
<td>120K</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>No</td>
<td>Divorced</td>
<td>95K</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>Married</td>
<td>60K</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Divorced</td>
<td>220K</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>No</td>
<td>Single</td>
<td>85K</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>No</td>
<td>Married</td>
<td>75K</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>No</td>
<td>Single</td>
<td>90K</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Tree Induction Issues

1. Determine how to split the records
   - How to specify the attribute test condition?
   - How to determine the best split?

2. Determine when to stop splitting
How to Specify the Attribute Test Condition?

• Depends on attribute types
  – Nominal
  – Ordinal
  – Continuous

• Depends on number of ways to split
  – 2-way split
  – Multi-way split
Splitting Based on Nominal Attributes

- **Multi-way split**: Use as many partitions as distinct values

  ![Multi-way split diagram]

- **Binary split**: Divides values into two subsets. Need to find optimal partitioning

  ![Binary split diagram]
Splitting Based on Ordinal Attributes

- **Multi-way split**: Use as many partitions as distinct values.

  ![Multi-way split diagram]

- **Binary split**: Divides values into two subsets, while keeping the order. Need to find optimal partitioning.

  ![Binary split diagram]
Splitting Based on Continuous Attributes

(i) Binary split

(ii) Multi-way split

Taxable Income > 80K?

Yes

No

Taxable Income?

< 10K

[10K, 25K)

[25K, 50K)

[50K, 80K)

> 80K
Splitting Based on Continuous Attributes

- Different ways of handling
  - **Discretization** to form an ordinal categorical attribute
    - equal-interval binning
    - equal-frequency binning
    - binning based on user-provided boundaries
  - **Binary Decision**: $(A < v)$ or $(A \geq v)$
    - usually sufficient in practice
    - consider all possible splits
    - find the best cut based on a purity measure (see below)
    - can be computationally expensive
Discretization Example

- Attribute values (for one attribute e.g., age):
  - 0, 4, 12, 16, 16, 18, 24, 26, 28

- Equal-width binning – for bin width of e.g., 10:
  - Bin 2: 12, 16, 16, 18 \([10,20)\) bin
  - Bin 3: 24, 26, 28 \([20, +\infty)\) bin
  - Bin 1: 0, 4 \([-\infty, 10)\) bin
    - \(-\infty\) denotes negative infinity, \(+\infty\) positive infinity

- Equal-frequency binning – for bin density of e.g., 3:
  - Bin 1: 0, 4, 12 \([-\infty, 14)\) bin
  - Bin 2: 16, 16, 18 \([14, 21)\) bin
  - Bin 3: 24, 26, 28 \([21, +\infty]\) bin
How to determine the Best Split?

Before Splitting: 10 records of class 0, 10 records of class 1

Which test condition is the best?
How to determine the Best Split?

- Nodes with **homogeneous** class distribution are preferred.
- Need a measure of node impurity:
  
<table>
<thead>
<tr>
<th>C0: 5</th>
<th>C0: 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1: 5</td>
<td>C1: 1</td>
</tr>
</tbody>
</table>

  - Non-homogeneous, High degree of impurity
  - Homogeneous, Low degree of impurity

- Common measures of node impurity:
  - Gini Index
  - Entropy
  - Misclassification error
Gini Index

- Named after Corrado Gini (1885-1965)
- Used to measure the distribution of income
  - 1: somebody gets everything
  - 0: everybody gets an equal share
Measure of Impurity: GINI

- Gini Index for a given node $t$:

$$GINI(t) = 1 - \sum_j [p(j | t)]^2$$

(NOTE: $p(j | t)$ is the relative frequency of class $j$ at node $t$).

- Maximum ($1 - 1/n_c$) when records are equally distributed among all classes, implying least interesting information

- Minimum (0.0) when all records belong to one class, implying most interesting information

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
<th></th>
<th>C1</th>
<th>C2</th>
<th></th>
<th>C1</th>
<th>C2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>6</td>
<td></td>
<td>1</td>
<td>5</td>
<td></td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Gini=0.000</td>
<td>Gini=0.278</td>
<td>Gini=0.444</td>
<td>Gini=0.500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Examples for computing GINI

\[
GINI(t) = 1 - \sum_j [p(j \mid t)]^2
\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

P(C1) = 0/6 = 0    P(C2) = 6/6 = 1
Gini = 1 – P(C1)^2 – P(C2)^2 = 1 – 0 – 1 = 0

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

P(C1) = 1/6    P(C2) = 5/6
Gini = 1 – (1/6)^2 – (5/6)^2 = 0.278

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

P(C1) = 2/6    P(C2) = 4/6
Gini = 1 – (2/6)^2 – (4/6)^2 = 0.444
Splitting Based on GINI

- When a node $p$ is split into $k$ partitions (children), the quality of split is computed as

$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

- where $n_i$ = number of records at child $i$,
- $n$ = number of records at node $p$.

- Intuition:
  - The GINI index of each partition is weighted
  - according to the partition's size
Binary Attributes: Computing GINI Index

- Splits into two partitions

Gini(N1)
= 1 – (5/7)^2 – (2/7)^2
= 0.408

Gini(N2)
= 1 – (1/5)^2 – (4/5)^2
= 0.320

Gini(Children)
= 7/12 * 0.408 + 5/12 * 0.320
= 0.371
Categorical Attributes: Computing Gini Index

- For each distinct value, gather counts for each class in the dataset
- Use the count matrix to make decisions

Multi-way split

<table>
<thead>
<tr>
<th>CarType</th>
<th>Family</th>
<th>Sports</th>
<th>Luxury</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>C2</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Gini</td>
<td>0.393</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Two-way split

(find best partition of values)

<table>
<thead>
<tr>
<th>CarType</th>
<th>{Sports, Luxury}</th>
<th>{Family}</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>C2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Gini</td>
<td>0.400</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CarType</th>
<th>{Sports}</th>
<th>{Family, Luxury}</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>C2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Gini</td>
<td>0.419</td>
<td></td>
</tr>
</tbody>
</table>
Continuous Attributes: Computing Gini Index

- Use Binary Decisions based on one value
- Several Choices for the splitting value
  - Number of possible splitting values = Number of distinct values
- Each splitting value has a count matrix associated with it
  - Class counts in each of the partitions, $A < v$ and $A \geq v$
- Simple method to choose best $v$
  - For each $v$, scan the database to gather count matrix and compute its Gini index
  - Computationally Inefficient!

<table>
<thead>
<tr>
<th>Tid</th>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Cheat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Single</td>
<td>125K</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>Married</td>
<td>100K</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>Single</td>
<td>70K</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Married</td>
<td>120K</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>No</td>
<td>Divorced</td>
<td>95K</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>Married</td>
<td>60K</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Divorced</td>
<td>220K</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>No</td>
<td>Single</td>
<td>85K</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>No</td>
<td>Married</td>
<td>75K</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>No</td>
<td>Single</td>
<td>90K</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Taxable Income > 80K?
Continuous Attributes: Computing Gini Index

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

<table>
<thead>
<tr>
<th>Cheat</th>
<th>No</th>
<th>No</th>
<th>No</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>No</th>
<th>No</th>
<th>No</th>
<th>No</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Taxable Income</strong></td>
<td>60</td>
<td>70</td>
<td>75</td>
<td>85</td>
<td>90</td>
<td>95</td>
<td>100</td>
<td>120</td>
<td>125</td>
<td>172</td>
<td>220</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
</tr>
<tr>
<td><strong>Sorted Values</strong></td>
<td>55</td>
<td>65</td>
<td>72</td>
<td>80</td>
<td>87</td>
<td>92</td>
<td>97</td>
<td>110</td>
<td>122</td>
<td>172</td>
<td>230</td>
</tr>
<tr>
<td><strong>Split Positions</strong></td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
</tr>
<tr>
<td>Yes</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>No</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Gini</td>
<td>0.420</td>
<td>0.400</td>
<td>0.375</td>
<td>0.343</td>
<td>0.417</td>
<td>0.400</td>
<td>0.300</td>
<td>0.343</td>
<td>0.375</td>
<td>0.400</td>
<td>0.420</td>
</tr>
</tbody>
</table>
## Continuous Attributes: Computing Gini Index

- Note: it is enough to compute the GINI for those positions where the label changes!

<table>
<thead>
<tr>
<th>Cheat</th>
<th>No</th>
<th>No</th>
<th>No</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>No</th>
<th>No</th>
<th>No</th>
<th>No</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taxable Income</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>70</td>
<td>75</td>
<td>85</td>
<td>90</td>
<td>95</td>
<td>100</td>
<td>120</td>
<td>125</td>
<td>150</td>
<td>172</td>
<td>220</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
</tr>
<tr>
<td>55</td>
<td>65</td>
<td>72</td>
<td>80</td>
<td>87</td>
<td>92</td>
<td>97</td>
<td>110</td>
<td>122</td>
<td>172</td>
<td>230</td>
<td></td>
</tr>
<tr>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
</tr>
<tr>
<td>Yes</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>No</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Gini</td>
<td>0.420</td>
<td>0.400</td>
<td>0.375</td>
<td>0.343</td>
<td>0.417</td>
<td>0.400</td>
<td>0.300</td>
<td>0.343</td>
<td>0.375</td>
<td>0.400</td>
<td>0.420</td>
</tr>
</tbody>
</table>
Alternative Splitting Criteria: Information Gain

- Entropy at a given node \( t \):

\[
Entropy(t) = - \sum_j p(j \mid t) \log_2 p(j \mid t)
\]

(NOTE: \( p(j \mid t) \) is the relative frequency of class \( j \) at node \( t \)).

- Measures homogeneity of a node
  - Maximum (\( \log nc \)) when records are equally distributed among all classes implying least information
  - Minimum (0.0) when all records belong to one class, implying most information

- Entropy based computations are similar to the GINI index computations
Splitting Based on Information Gain

- Information Gain:

\[ GAIN_{\text{split}} = \text{Entropy}(p) - \left( \sum_{i=1}^{k} \frac{n_i}{n} \text{Entropy}(i) \right) \]

- Parent Node, \( p \) is split into \( k \) partitions;
- \( n_i \) is number of records in partition \( i \)
  - Measures reduction in entropy achieved because of the split
    - Choose the split that achieves most reduction (maximizes GAIN)
  - Disadvantage: Tends to prefer splits that result in large number of partitions, each being small but pure
    - e.g., split by ID attribute
How to Find the Best Split

Before Splitting:

<table>
<thead>
<tr>
<th></th>
<th>C0</th>
<th>C1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N00</td>
<td>N01</td>
<td></td>
</tr>
</tbody>
</table>

A?

Yes: Node N1

No: Node N2

Gain = M0 – M12 vs M0 – M34

B?

Yes: Node N3

No: Node N4
Alternative Splitting Criteria: GainRATIO

• Gain Ratio:

\[
GainRATIO_{\text{split}} = \frac{GAIN_{\text{split}}}{SplitINFO}
\]

\[
SplitINFO = -\sum_{i=1}^{k} \frac{n_i}{n} \log \frac{n_i}{n}
\]

• Parent Node, p is split into k partitions
• \(n_i\) is the number of records in partition I
  – Adjusts Information Gain by the entropy of the partitioning (SplitINFO)
    • Higher entropy partitioning (large number of small partitions) is penalized!
  – Designed to overcome the tendency to generate a large number of small partitions
Alternative Splitting Criteria: Classification Error

- Classification error at a node \( t \) :

\[
\text{Error}(t) = 1 - \max_i P(i \mid t)
\]

- Measures misclassification error made by a node.
  - Assumption: The node classifies every example to belong to the majority class
  - Maximum \( (1 - 1/n_c) \) when records are equally distributed among all classes, implying least interesting information
  - Minimum \( (0.0) \) when all records belong to one class, implying most interesting information
Decision Trees in RapidMiner (ID3)

Learns an un-pruned decision tree from nominal attributes only.

- **criterion**: gain_ratio
- **minimal size for split**: 4
- **minimal leaf size**: 2
- **minimal gain**: 0.1
Decision Trees in RapidMiner

More flexible algorithm that includes pruning and discretization
Model Evaluation

- Metrics
  - how to measure performance?
- Evaluation methods
  - how to obtain meaningful estimates?
Model Evaluation

• Models are evaluated by looking at
  • correctly and incorrectly classified instances

• For a two-class problems, four cases can occur:
  • true positives: positive class correctly predicted
  • false positives: positive class incorrectly predicted
  • true negatives: negative class correctly predicted
  • false negatives: negative class incorrectly predicted
Metrics for Performance Evaluation

- Focus on the predictive capability of a model
- Rather than how fast it takes to classify or build models

Confusion Matrix:

<table>
<thead>
<tr>
<th>ACTUAL CLASS</th>
<th>PREDICTED CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Class=Yes</td>
</tr>
<tr>
<td>Class=Yes</td>
<td>TP</td>
</tr>
<tr>
<td>Class=No</td>
<td>FP</td>
</tr>
</tbody>
</table>
Metrics for Performance Evaluation

- Most frequently used metrics:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Error Rate = $1 - \text{Accuracy}$

<table>
<thead>
<tr>
<th>ACTUAL CLASS</th>
<th>PREDICTED CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Class=Yes</td>
</tr>
<tr>
<td>Class=Yes</td>
<td>TP</td>
</tr>
<tr>
<td>Class=No</td>
<td>FP</td>
</tr>
</tbody>
</table>
What is a Good Accuracy?

• i.e., when are you done?
  – at 75% accuracy?
  – at 90% accuracy?
  – at 95% accuracy?

• Depends on difficulty of the problem!

• Baseline: naive guessing
  – always predict majority class

• Compare
  – Predicting coin tosses with accuracy of 50%
  – Predicting dice roll with accuracy of 50%
Limitation of Accuracy: Unbalanced Data

- Sometimes, classes have very unequal frequency
  - Fraud detection: 98% transactions OK, 2% fraud
  - eCommerce: 99% don’t buy, 1% buy
  - Intruder detection: 99.99% of the users are no intruders
  - Security: >99.99% of Americans are not terrorists

- The class of interest is commonly called the **positive class**, and the rest **negative classes**.

- 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
Precision and Recall

**Alternative:** Use measures from information retrieval which are biased towards the positive class.

<table>
<thead>
<tr>
<th></th>
<th>Classified Positive</th>
<th>Classified Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual Positive</td>
<td>TP</td>
<td>FN</td>
</tr>
<tr>
<td>Actual Negative</td>
<td>FP</td>
<td>TN</td>
</tr>
</tbody>
</table>

Precision $p$ is the number of correctly classified positive examples divided by the total number of examples that are classified as positive.

Recall $r$ is the number of correctly classified positive examples divided by the total number of actual positive examples in the test set.

\[
p = \frac{TP}{TP + FP}.
\]

\[
r = \frac{TP}{TP + FN}.
\]
Precision and Recall Example

<table>
<thead>
<tr>
<th></th>
<th>Classified Positive</th>
<th>Classified Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual Positive</td>
<td>1</td>
<td>99</td>
</tr>
<tr>
<td>Actual Negative</td>
<td>0</td>
<td>1000</td>
</tr>
</tbody>
</table>

- This confusion matrix gives us
  - precision $p = 100\%$ and
  - recall $r = 1\%$
- because we only classified one positive example correctly and no negative examples wrongly
- We want a measure that combines precision and recall
F$_1$-Measure

- It is hard to compare two classifiers using two measures
- F$_1$-Score combines precision and recall into one measure

\[ F_1 = \frac{2pr}{p + r} \]

F$_1$-score is the harmonic mean of precision and recall.

\[ F_1 = \frac{2}{\frac{1}{p} + \frac{1}{r}} \]

- The harmonic mean of two numbers tends to be closer to the smaller of the two.
- For F$_1$-value to be large, both $p$ and $r$ must be large
$F_1$-Measure
## Alternative for Unbalanced Data: Cost Matrix

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

<table>
<thead>
<tr>
<th>ACTUAL CLASS</th>
<th>PREDICTED CLASS</th>
<th>Class=Yes</th>
<th>Class=No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class=Yes</td>
<td>C(Yes</td>
<td>Yes)</td>
<td>C(No</td>
</tr>
<tr>
<td>Class=No</td>
<td>C(Yes</td>
<td>No)</td>
<td>C(No</td>
</tr>
</tbody>
</table>
Computing Cost of Classification

<table>
<thead>
<tr>
<th>COST MATRIX</th>
<th>PREDICTED CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTUAL CLASS</td>
<td>C(i</td>
</tr>
<tr>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MODEL M₁</th>
<th>PREDICTED CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTUAL CLASS</td>
<td>+</td>
</tr>
<tr>
<td>+</td>
<td>150</td>
</tr>
<tr>
<td>-</td>
<td>60</td>
</tr>
</tbody>
</table>

Accuracy = 80%
Cost = 4060

<table>
<thead>
<tr>
<th>MODEL M₂</th>
<th>PREDICTED CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTUAL CLASS</td>
<td>+</td>
</tr>
<tr>
<td>+</td>
<td>250</td>
</tr>
<tr>
<td>-</td>
<td>5</td>
</tr>
</tbody>
</table>

Accuracy = 90%
Cost = 4505

Heiko Paulheim
ROC Curves

• Some classification algorithms provide confidence scores
  – how sure the algorithms is with its prediction
  – e.g., Naive Bayes: the probability
  – e.g., Decision Trees: the purity of the respective leaf node

• Drawing a ROC Curve
  – Sort classifications according to confidence scores
  – Evaluate
    • right prediction: draw one step up
    • wrong prediction: draw one step to the right
ROC Curves

- Drawing ROC Curves in RapidMiner
Example ROC Curve of Naive Bayes
Example ROC Curve of Decision Tree Learner
Interpreting ROC Curves

- Best possible result:
  - all correct predictions have higher confidence than all incorrect ones
- The steeper, the better
  - random guessing results in the diagonal
  - so a decent algorithm should result in a curve significantly above the diagonal
- Comparing algorithms:
  - Curve A above curve B means algorithm A better than algorithm B
- Frequently used criterion
  - Area under curve
  - normalized to 1
Methods for Performance Evaluation

- How to obtain a reliable estimate of performance?
- Performance of a model may depend on other factors besides the learning algorithm:
  - Size of training and test sets (it often expensive to get labeled data)
  - Class distribution (balanced, skewed)
  - Cost of misclassification (your goal)
- Methods for estimating the performance
  - Holdout
  - Random Subsampling
  - Cross Validation
Learning Curve

- Learning curve shows how accuracy changes with varying sample size
- Conclusion: Use as much data as possible for training
Holdout Method

• The **holdout method** reserves a certain amount for testing and uses the remainder for training

• Usually: one third for testing, the rest for training

• applied when **lots of sample data** is available

• For “unbalanced” datasets, samples might not be representative
  - Few or none instances of some classes

• **Stratified sample**: *balances the data*
  - Make sure that each class is represented with approximately equal proportions in both subsets
Leave One Out

- Iterate over all examples
  - train a model on all examples but the current one
  - evaluate on the current one

- Yields a very accurate estimate

- Uses as much data for training as possible
  - but is computationally infeasible in most cases

- Imagine: a dataset with a million instances
  - one minute to train a single model
  - Leave one out would take almost two years
Cross-Validation

- Compromise of Leave One Out and decent runtime
- Cross-validation avoids overlapping test sets
  - First step: data is split into $k$ subsets of equal size
  - Second step: each subset in turn is used for testing and the remainder for training
- This is called $k$-fold cross-validation
- The error estimates are averaged to yield an overall error estimate
- Frequently used value for $k$: 10
  - Why ten? Extensive experiments have shown that this is the good choice to get an accurate estimate
- Often the subsets are stratified before the cross-validation is performed
Cross-Validation in RapidMiner
Practical Issue: Overfitting

- Example: predict credit rating
  - possible decision tree:

<table>
<thead>
<tr>
<th>Name</th>
<th>Net Income</th>
<th>Job status</th>
<th>Debts</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>40000</td>
<td>employed</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>Mary</td>
<td>38000</td>
<td>employed</td>
<td>10000</td>
<td>-</td>
</tr>
<tr>
<td>Stephen</td>
<td>21000</td>
<td>self-employed</td>
<td>20000</td>
<td>-</td>
</tr>
<tr>
<td>Eric</td>
<td>2000</td>
<td>student</td>
<td>10000</td>
<td>-</td>
</tr>
<tr>
<td>Alice</td>
<td>35000</td>
<td>employed</td>
<td>4000</td>
<td>+</td>
</tr>
</tbody>
</table>
Practical Issue: Overfitting

- Example: predict credit rating
  - alternative decision tree:

<table>
<thead>
<tr>
<th>Name</th>
<th>Net Income</th>
<th>Job status</th>
<th>Debts</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>40000</td>
<td>employed</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>Mary</td>
<td>38000</td>
<td>employed</td>
<td>10000</td>
<td>-</td>
</tr>
<tr>
<td>Stephen</td>
<td>21000</td>
<td>self-employed</td>
<td>20000</td>
<td>-</td>
</tr>
<tr>
<td>Eric</td>
<td>20000</td>
<td>student</td>
<td>10000</td>
<td>-</td>
</tr>
<tr>
<td>Alice</td>
<td>35000</td>
<td>employed</td>
<td>40000</td>
<td>+</td>
</tr>
</tbody>
</table>
Practical Issue: Overfitting

• Both trees seem equally good
  – Classify all instances in the training set correctly
  – Which one do you prefer?
Occam's Razor

• Named after William of Ockham (1287-1347)
• A fundamental principle of science
  – if you have two theories
  – that explain a phenomenon equally well
  – choose the simpler one

• Example:
  – phenomenon: the street is wet
  – theory 1: *it has rained*
  – theory 2: *a beer truck has had an accident, and beer has spilled. The truck has been towed, and magpies picked the glass pieces, so only the beer remains*
Practical Issue: Overfitting

- **Overfitting**: Good accuracy on training data, but poor on test data.
- **Symptoms**: Tree too deep and too many branches
- **Typical causes of overfitting**
  - too little training data
  - noise
  - poor learning algorithm
Overfitting - Illustration

Polynomial degree 1
(linear function)

Polynomial degree 4
(n-1 degrees can always fit n points)

Prediction for this value of $x$?

or here?

here?
Overfitting and Noise

Likely to overfit the data
How to Address Overfitting?

• Pre-Pruning (Early Stopping Rule)
  – Stop the algorithm before it becomes a fully-grown tree
  – Typical stopping conditions for a node:
    • Stop if all instances belong to the same class
    • Stop if all the attribute values are the same
  – Less restrictive conditions:
    • Stop if number of instances within a node is less than some user-specified threshold
    • Stop if expanding the current node only slightly improves the impurity measure (user-specified threshold)
How to Address Overfitting?

- **Post-pruning**
  1. Grow decision tree to its entire size
  2. Trim the nodes of the decision tree in a bottom-up fashion
     - using a validation data set
     - or an estimate of the generalization error
  3. If generalization error improves after trimming
     - replace sub-tree by a leaf node
     - Class label of leaf node is determined from majority class of instances in the sub-tree
Training vs. Generalization Errors

• Training error
  – also: resubstitution error, apparent error
  – errors made in training
  – evidence: misclassified training instances

• Generalization error
  – errors made on unseen data
  – evidence: no apparent evidence

• Training error can be computed
• Generalization error must be estimated
Estimating the Generalization Error

- **Training errors:** error on training ($\sum e(t)$)
- **Generalization errors:** error on testing ($\sum e'(t)$)

**Methods for estimating generalization errors:**

1. **(Too) Optimistic approach:** $e'(t) = e(t)$
2. **Pessimistic approach:**
   - For each leaf node: $e'(t) = (e(t)+0.5)$ (user-defined 0.5 penalty for large trees)
   - Total errors: $e'(T) = e(T) + N \times 0.5$ ($N$: number of leaf nodes)
   - For a tree with 30 leaf nodes and 10 errors on training (out of 1000 instances):
     - Training error = $10/1000 = 1\%$
     - Generalization error = $(10 + 30 \times 0.5)/1000 = 2.5\%$

**Reduced Error Pruning (REP):**
- use validation data set to estimate generalization error
Example of Post-Pruning

<table>
<thead>
<tr>
<th>Class = Yes</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class = No</td>
<td>10</td>
</tr>
<tr>
<td>Error</td>
<td>10/30</td>
</tr>
</tbody>
</table>

Training Error (Before splitting) = 10/30
Pessimistic error = (10 + 0.5)/30 = 10.5/30
Training Error (After splitting) = 9/30
Pessimistic error (After splitting) = (9 + 4 \times 0.5)/30 = 11/30

PRUNE!

Heiko Paulheim
Discussion of Decision Trees

• Advantages:
  – Inexpensive to construct
  – Fast at classifying unknown records
  – Easy to interpret by humans for small-sized trees
  – Accuracy is comparable to other classification techniques for many simple data sets

• Disadvantages:
  – Decisions are based only on a single attribute at a time
  – Can only represent decision boundaries that are parallel to the axes
  – Often not appropriate for continuous attributes
Comparing Decision Trees and k-NN

• Decision boundaries
  – k-NN: arbitrary
  – Decision trees: rectangular

• Sensitivity to scales
  – k-NN: needs normalization
  – Decision tree: does not require normalization (why?)

• Runtime & memory
  – k-NN is cheap to train, but expensive for classification
  – Decision tree is expensive to train, but cheap for classification
Questions?