Data Mining

Classification

- Part 1 -
1. What is Classification?

- Goal: Previously unseen records should be assigned a class from a given set of classes as accurately as possible.

- Approach: Given a collection of records (training set)
  - each record contains a set of attributes
  - one of the attributes is the class (label) that should be predicted.

- Find a model for the class attribute as a function of the values of other attributes.

- Variants
  - single-class problems (class labels e.g. true/false or fraud/no fraud)
  - multi-class problems (class labels e.g. low, medium, high)
Introduction to Classification

A Couple of Questions:

– What is this?
– Why do you know?
– How have you come to that knowledge?
Introduction to Classification

- Goal: Learn a model for recognizing a concept, e.g. trees
- Training data:

  "tree"  "tree"  "tree"

  "not a tree"  "not a tree"  "not a tree"

"not a tree"  "not a tree"  "not a tree"
Introduction to Classification

- We (or the learning algorithm) look at positive and negative examples (training data)
- ... and derive a model
e.g., "Trees are big, green plants that have a trunk."
- Goal: Classification of unseen instances

Warning:
Models are only approximating examples! Not guaranteed to be correct or complete!
Model Learning and Model Application Process

Class/Label Attribute

Training Set

<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>60K</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>

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>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>80K</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>67K</td>
<td>?</td>
</tr>
</tbody>
</table>

Learning algorithm

Induction

Learn Model

Apply Model

Deduction
Classification Examples

- **Credit Risk Assessment**
  - Attributes: your age, income, debts, …
  - Class: Are you getting credit by your bank?

- **Marketing**
  - Attributes: previously bought products, browsing behaviour
  - Class: Are you a target customer for a new product?

- **Tax Fraud**
  - Attributes: the values in your tax declaration
  - Class: Are you trying to cheat?

- **SPAM Detection**
  - Attributes: words and header fields of an e-mail
  - Class: Is it a spam e-mail?
Classification Techniques

1. K-Nearest-Neighbors
2. Decision Trees
3. Rule Learning
4. Naïve Bayes
5. Support Vector Machines
6. Artificial Neural Networks
7. Deep Neural Networks
8. Many others …
Example Problem

- Predict what the current weather is in a certain place
- where there is no weather station.
- How could you do that?
Basic Idea

- Use the average of the nearest stations

- Example:
  - 3x sunny
  - 2x cloudy
  - result = sunny

- This approach is called K-Nearest-Neighbors
  - where k is the number of neighbors to consider
  - in the example: k=5
  - in the example: “near” denotes geographical proximity
K-Nearest-Neighbors Classifiers

- Require three things
  - The set of stored records
  - A distance measure to compute distance between records
  - The value of $k$, the number of nearest neighbors to consider
- To classify an unknown record:
  1. Compute distance to each training record
  2. Identify $k$-nearest neighbors
  3. Use class labels of nearest neighbors to determine the class label of unknown record
     - by taking majority vote or
     - by weighing the vote according to distance
The k-nearest neighbors of a record $x$ are data points that have the k smallest distances to $x$.

(a) 1-nearest neighbor  
(b) 2-nearest neighbor  
(c) 3-nearest neighbor
Choosing a Good Value for K

- If \( k \) is too small, the result is sensitive to noise points.
- If \( k \) is too large, the neighborhood may include points from other classes.

- Rule of thumb: Test \( k \) values between 1 and 10.
Discussion of K-Nearest-Neighbor Classification

- Often very accurate
  - for instance for optical character recognition (OCR)

- ... but slow
  - as training data needs to be searched

- Assumes that all attributes are equally important
  - remedy: attribute selection or attribute weights

- Can handle decision boundaries which are not parallel to the axes (unlike decision trees)
Decision Boundaries of a 1-NN Classifier
KNN in RapidMiner
Applying the Model
### Resulting Dataset

<table>
<thead>
<tr>
<th>Row No.</th>
<th>Play</th>
<th>Prediction (Play)</th>
<th>confidence (no)</th>
<th>confidence (yes)</th>
<th>Outlook</th>
<th>Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>yes</td>
<td>no</td>
<td>0.667</td>
<td>0.333</td>
<td>sunny</td>
<td>85</td>
</tr>
<tr>
<td>2</td>
<td>no</td>
<td>no</td>
<td>0.667</td>
<td>0.333</td>
<td>overcast</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>yes</td>
<td>yes</td>
<td>0.333</td>
<td>0.667</td>
<td>overcast</td>
<td>83</td>
</tr>
<tr>
<td>4</td>
<td>yes</td>
<td>yes</td>
<td>0.333</td>
<td>0.667</td>
<td>rain</td>
<td>70</td>
</tr>
<tr>
<td>5</td>
<td>yes</td>
<td>yes</td>
<td>0.333</td>
<td>0.667</td>
<td>rain</td>
<td>68</td>
</tr>
<tr>
<td>6</td>
<td>no</td>
<td>yes</td>
<td>0.333</td>
<td>0.667</td>
<td>rain</td>
<td>65</td>
</tr>
<tr>
<td>7</td>
<td>yes</td>
<td>yes</td>
<td>0.333</td>
<td>0.667</td>
<td>overcast</td>
<td>64</td>
</tr>
<tr>
<td>8</td>
<td>no</td>
<td>yes</td>
<td>0.333</td>
<td>0.667</td>
<td>sunny</td>
<td>72</td>
</tr>
<tr>
<td>9</td>
<td>yes</td>
<td>yes</td>
<td>0.333</td>
<td>0.667</td>
<td>sunny</td>
<td>69</td>
</tr>
<tr>
<td>10</td>
<td>no</td>
<td>yes</td>
<td>0.333</td>
<td>0.667</td>
<td>sunny</td>
<td>75</td>
</tr>
<tr>
<td>11</td>
<td>yes</td>
<td>yes</td>
<td>0.333</td>
<td>0.667</td>
<td>sunny</td>
<td>68</td>
</tr>
<tr>
<td>12</td>
<td>yes</td>
<td>yes</td>
<td>0.333</td>
<td>0.667</td>
<td>overcast</td>
<td>72</td>
</tr>
<tr>
<td>13</td>
<td>no</td>
<td>yes</td>
<td>0</td>
<td>1</td>
<td>overcast</td>
<td>81</td>
</tr>
<tr>
<td>14</td>
<td>yes</td>
<td>yes</td>
<td>0.333</td>
<td>0.667</td>
<td>rain</td>
<td>71</td>
</tr>
</tbody>
</table>
Lazy versus Eager Learning

- **Lazy Learning**
  - Instance-based learning approaches, like KNN, are also called lazy learning as no explicit knowledge (model) is learned
  - **Single goal**: Classify unseen records as accurately as possible

- **Eager Learning**
  - but actually, we might have **two goals**
    1. classify unseen instances
    2. understand the application domain as a human
  - Eager learning approaches generate models that are (might be) interpretable by humans
  - Examples of eager techniques: Decision Tree Learning, Rule Learning
3. Decision Tree Classifiers

Decision trees encode a procedure for taking a classification decision.
Applying a Decision Tree to Unseen Data

Start from the root of tree.

Unseen Record

<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”
The decision boundaries are parallel to the axes because the test condition involves a single attribute at-a-time.
### Decision Tree Induction

#### 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>

#### Decision Tree

- **Refund**: 
  - Yes → NO
  - No → MarSt
    - Single, Divorced: TaxInc < 80K → NO
      - > 80K → YES
    - Married: TaxInc > 80K → NO

**Model**: Decision Tree

### Learning Algorithm
Decision Tree Induction

- How to learn a decision tree from training data?
- Finding an optimal decision tree is NP-hard.
- Tree building algorithms thus use a greedy, top-down, recursive partitioning strategy to induce a reasonable solution.
- Many different algorithms have been proposed:
  - Hunt’s Algorithm
  - ID3
  - CHAID
  - C4.5
Hunt’s Algorithm

- Let $D_t$ be the set of training records that reach a node $t$

- General procedure:
  - If $D_t$ only contains 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 subsets having a higher purity.
  - Recursively apply the procedure to each subset.
Hunt’s Algorithm – Step 1

1. We test all possible splits and measure the purity of the resulting subsets
2. We find the split on Refund to produce the purest subsets

<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>
Hunt’s Algorithm – Step 2

1. We further examine the Refund=No records
2. Again, we test all possible splits
3. We find the split on Marital Status to produce the purest subsets
Hunt’s Algorithm – Step 3

1. We further examine the Marital Status=Single or =Divorced records
2. We find a split on Taxable Income to produce pure subsets

<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>3</td>
<td>No</td>
<td>Single</td>
<td>70K</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>No</td>
<td>Divorced</td>
<td>95K</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>No</td>
<td>Single</td>
<td>85K</td>
<td>Yes</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
3.1 How to Specify the Attribute Test Condition?

1. Depends on attribute types
   - Nominal
   - Ordinal
   - Continuous

2. 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 example]

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

  ![Binary split example]
Splitting Based on Ordinal Attributes

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

  ![Diagram of Multi-way split]

  - Small
  - Medium
  - Large

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

  ![Diagram of Binary split]

  - {Small, Medium} -> {Large}
  - {Medium, Large} -> {Small}
  - OR
  - {Small} -> {Medium, Large}

  ![Diagram of Binary split with alternative]

  - {Small} -> {Medium, Large}
  - {Medium, Large} -> {Small}
Splitting Based on Continuous Attributes

(i) Binary split

Taxable Income $> 80K$?

- Yes
- No

(ii) Multi-way split

- Taxable Income $< 10K$
- Taxable Income $[10K,25K]$
- Taxable Income $[25K,50K]$
- Taxable Income $[50K,80K]$
- Taxable Income $> 80K$
Splitting Based on Continuous Attributes

Different ways of handling continuous attributes

- **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
  - find the best splitting border $v$
    - based on a purity measure
    - (see below)
  - can be compute intensive
Discretization Example

- **Values of the attribute, e.g., age of a person:**
  - 0, 4, 12, 16, 16, 18, 24, 26, 28

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

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

Before splitting the dataset contains:
- 10 records of class 0 and
- 10 records of class 1

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

- Nodes with **homogeneous** class distribution are preferred.
- Need a measure of **node impurity**:

<table>
<thead>
<tr>
<th></th>
<th>C0: 5</th>
<th>C1: 5</th>
<th>C0: 9</th>
<th>C1: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-homogeneous, High degree of impurity</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Homogeneous, Low degree of impurity</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Common measures of node impurity:
  - GINI Index
  - Entropy
  - Misclassification error
Comparing different Splits

Before Splitting:

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

A?

Yes

Node N1

<table>
<thead>
<tr>
<th></th>
<th>C0</th>
<th>N10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C1</td>
<td>N11</td>
</tr>
</tbody>
</table>

No

Node N2

<table>
<thead>
<tr>
<th></th>
<th>C0</th>
<th>N20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C1</td>
<td>N21</td>
</tr>
</tbody>
</table>

B?

Yes

Node N3

<table>
<thead>
<tr>
<th></th>
<th>C0</th>
<th>N30</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C1</td>
<td>N31</td>
</tr>
</tbody>
</table>

No

Node N4

<table>
<thead>
<tr>
<th></th>
<th>C0</th>
<th>N40</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C1</td>
<td>N41</td>
</tr>
</tbody>
</table>

M1

M2

M3

M4

M12

M34

Purity Gain = M0 – M12 vs M0 – M34
3.2.1 Measure of Impurity: GINI Index

- GINI Index for a given node $t$:

$$\text{GINI} (t) = 1 - \sum_{j} [p(j | t)]^2$$

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

- Minimum (0.0) when all records belong to one class
- Maximum $(1 - 1/n_c)$ when records are equally distributed among all classes. $n_c = \text{number of classes}$

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C1</th>
<th>C1</th>
<th>C1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>C2</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Gini=0.000   \hspace{1cm} \text{Gini}=0.278   \hspace{1cm} \text{Gini}=0.444   \hspace{1cm} \text{Gini}=0.500
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>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[P(C1) = 0/6 = 0 \quad P(C2) = 6/6 = 1\]

Gini \[= 1 - P(C1)^2 - P(C2)^2 = 1 - 0 - 1 = 0\]

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

\[P(C1) = 1/6 \quad P(C2) = 5/6\]

Gini \[= 1 - (1/6)^2 - (5/6)^2 = 0.278\]

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

\[P(C1) = 2/6 \quad P(C2) = 4/6\]

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 GINI index of each partition is weighted according to the partition's size.

- The quality of the overall split is computed as:

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

where: $n_i = \text{number of records at child } i$

$n = \text{number of records at node } p$
Computing the Purity Gain of a Possible Split

Split into two partitions

\[
\begin{align*}
\text{GINI}(N1) &= 1 - \left(\frac{5}{7}\right)^2 - \left(\frac{2}{7}\right)^2 \\
&= 0.408 \\
\text{GINI}(N2) &= 1 - \left(\frac{1}{5}\right)^2 - \left(\frac{4}{5}\right)^2 \\
&= 0.32
\end{align*}
\]

\[
\begin{array}{c|c|c}
 & N1 & N2 \\
\hline
C1 & 5 & 1 \\
C2 & 2 & 4 \\
\hline
\end{array}
\]

\[\text{GINI}_{\text{Split}} = \frac{7}{12} \times 0.408 + \frac{5}{12} \times 0.32 = 0.371\]

Purity Gain = 0.5 – 0.371 = 0.129
Categorical Attributes: Computing Gini Index

For each distinct attribute value, gather counts for each class.

### 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

- How to find the best binary split for a continuous attribute?
- Efficient computation:
  1. Sort the attribute on values
  2. Linearly scan these values, each time updating the count matrix and computing the gini index
  3. Choose the split position that has the smallest gini index

#### Example

<table>
<thead>
<tr>
<th>Taxable Income</th>
<th>Yes</th>
<th>No</th>
<th>Gini</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= 60</td>
<td>0</td>
<td>0</td>
<td>0.420</td>
</tr>
<tr>
<td>&gt; 60 &lt;= 70</td>
<td>3</td>
<td>7</td>
<td>0.400</td>
</tr>
<tr>
<td>&gt; 70 &lt;= 75</td>
<td>0</td>
<td>1</td>
<td>0.375</td>
</tr>
<tr>
<td>&gt; 75 &lt;= 80</td>
<td>3</td>
<td>5</td>
<td>0.343</td>
</tr>
<tr>
<td>&gt; 80 &lt;= 85</td>
<td>0</td>
<td>3</td>
<td>0.417</td>
</tr>
<tr>
<td>&gt; 85 &lt;= 90</td>
<td>3</td>
<td>1</td>
<td>0.400</td>
</tr>
<tr>
<td>&gt; 90 &lt;= 95</td>
<td>2</td>
<td>2</td>
<td><strong>0.300</strong></td>
</tr>
<tr>
<td>&gt; 95 &lt;= 100</td>
<td>1</td>
<td>3</td>
<td>0.343</td>
</tr>
<tr>
<td>&gt; 100 &lt;= 110</td>
<td>0</td>
<td>3</td>
<td>0.375</td>
</tr>
<tr>
<td>&gt; 110 &lt;= 120</td>
<td>0</td>
<td>3</td>
<td>0.400</td>
</tr>
<tr>
<td>&gt; 120 &lt;= 125</td>
<td>0</td>
<td>0</td>
<td>0.420</td>
</tr>
<tr>
<td>&gt; 125 &lt;= 172</td>
<td>0</td>
<td>0</td>
<td>0.343</td>
</tr>
<tr>
<td>&gt; 172 &lt;= 220</td>
<td>0</td>
<td>0</td>
<td>0.375</td>
</tr>
<tr>
<td>&gt; 220</td>
<td>0</td>
<td>0</td>
<td>0.400</td>
</tr>
</tbody>
</table>
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
3.2.2 Alternative Splitting Criterion: Information Gain

- Calculating the information gain relies on the entropy of each node.

- Entropy of a given node $t$:
  \[
  \text{Entropy} (t) = - \sum_j p(j | t) \log_2 p(j | t)
  \]
  
  $p(j | t)$ is the relative frequency of class $j$ at node $t$

- Entropy measures homogeneity of a node
  
  - Minimum (0.0) when all records belong to one class.
  
  - Maximum ($\log n_c$) when records are equally distributed among all classes.
**Examples for Computing Entropy**

\[
\text{Entropy} (t) = - \sum_{j} p(j \mid t) \log_2 p(j \mid t)
\]

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>P(C1) = 0/6 = 0</td>
<td>P(C2) = 6/6 = 1</td>
</tr>
<tr>
<td>6</td>
<td>Entropy = (- 0 \log_2 0 - 1 \log_2 1 = - 0 - 0 = 0)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>P(C1) = 1/6</td>
<td>P(C2) = 5/6</td>
</tr>
<tr>
<td>5</td>
<td>Entropy = (- (1/6) \log_2 (1/6) - (5/6) \log_2 (5/6) = 0.65)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>P(C1) = 2/6</td>
<td>P(C2) = 4/6</td>
</tr>
<tr>
<td>4</td>
<td>Entropy = (- (2/6) \log_2 (2/6) - (4/6) \log_2 (4/6) = 0.92)</td>
<td></td>
</tr>
</tbody>
</table>
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

- Information gain measures the entropy reduction of a split.
- We choose the split with the largest reduction (maximal GAIN)
- Disadvantage: Tends to prefer splits that result in large number of partitions, each being small but pure (split by ID attribute?)
3.2.3 Alternative Splitting Criterion: GainRATIO

- GainRATIO is designed to overcome the tendency to generate a large number of small partitions.

- GainRATIO adjusts information gain by the entropy of the partitioning (SplitINFO).

- Higher entropy of the partitioning (large number of small partitions) is penalized!

\[
\text{GainRATIO}_{\text{split}} = \frac{\text{GAIN}_{\text{split}}}{\text{SplitINFO}}
\]

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

Parent Node $p$ is split into $k$ partitions

$n_i$ is the number of records in partition $i$
3.2.4 Alternative Splitting Criterion: Classification Error

- Measures the misclassification error made by a node.
- Assumption: The node classifies every example to belong to the majority class.
- Classification error at a node $t$:

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

- Minimum (0.0) when all records belong to one class
- Maximum (1 - 1/\(n_c\)) when records are equally distributed among all classes
Examples for Computing the Classification Error

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

<table>
<thead>
<tr>
<th>C1</th>
<th>0</th>
<th>P(C1) = 0/6 = 0</th>
<th>P(C2) = 6/6 = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2</td>
<td>6</td>
<td>Error = 1 - max (0, 1) = 1 - 1 = 0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C1</th>
<th>1</th>
<th>P(C1) = 1/6</th>
<th>P(C2) = 5/6</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2</td>
<td>5</td>
<td>Error = 1 - max (1/6, 5/6) = 1 - 5/6 = 1/6</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C1</th>
<th>2</th>
<th>P(C1) = 2/6</th>
<th>P(C2) = 4/6</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2</td>
<td>4</td>
<td>Error = 1 - max (2/6, 4/6) = 1 - 4/6 = 1/3</td>
<td></td>
</tr>
</tbody>
</table>
Comparison of the different Impurity Measures

For a 2-class problem:
3.3 Overfitting

- **Problem:** Learned models can fit the training data too closely and thus work poorly on unseen data.

- Possible model fitting the training data:
  
  "Trees are **big**, **green plants** that have a **trunk** and **no wheels**."

- Deviating unseen instance:

- **Goal:** Find good compromise between specificity and generality of a model.
Overfitting: Second Example

- Example: Predict credit rating
  - possible decision tree:

```
Debts >5000

Yes  No

-  +
```

<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>
Overfitting: Second Example

- 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>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>
Overfitting: Second Example

- Both trees seem equally good
  - as they classify all instances in the training set correctly
  - Which one do you prefer?

```
Debts >5000
- Yes
+ No

Name = "John"
- Yes
+ No

Name = "Alice"
- Yes
+ No
```

Which tree 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
Overfitting: Symptoms and Causes

- Symptoms:
  1. tree too deep and
  2. too many branches

- Typical causes of overfitting
  1. noise / outliers
  2. too little training data
  3. poor learning algorithm
Example of an Outlier causing Overfitting

(A) A partition of the data space

(B) The decision tree

Likely to overfit the data
How to Address Overfitting?

- Pre-Pruning (Early Stopping)
  - Stop the algorithm before tree becomes fully-grown
  - Normal stopping conditions for a node (no pruning):
    - Stop if all instances belong to the same class
    - Stop if all the attribute values are the same
  - More restrictive conditions for Pre-Pruning:
    - 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 (unseen data not used for training)
     - or an *estimate* of the generalization error (see next slide)

  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
Estimating the Generalization Error

- Re-substitution error: error on training ($\sum e(t)$)
- Generalization error: error on testing ($\sum e'(t)$)

- Methods for estimating the generalization error:
  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%
  3. Reduced Error Pruning (REP):
      - use validation data set to estimate generalization error
3.4 Discussion of Decision Trees

- **Advantages**
  - Inexpensive to construct
  - Extremely fast at classifying unknown records
  - Easy to interpret by humans for small-sized trees (eager learning)
  - Accuracy is comparable to other classification techniques for many low dimensional data sets

- **Disadvantages**
  - Decisions are based only on a single attribute at a time
  - Can only represent decision boundaries parallel to the axes
3.5 Decision Trees in RapidMiner

The **ID3 operator** learns an decision tree from nominal attributes only.
The **Decision Tree operator** provides a more flexible algorithm that includes discretization and post-pruning.
Learned Decision Tree