CS 60050
Machine Learning

Decision Tree Classifier

Slides taken from course materials of Tan, Steinbach, Kumar
Illustrating Classification Task

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>

Test Set

<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>
Intuition behind a decision tree

- Ask a series of questions about a given record
  - Each question is about one of the attributes
  - Answer to one question decides what question to ask next (or if a next question is needed)
  - Continue asking questions until we can infer the class of the given record
Example of a Decision Tree

<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>

Training Data

Model: Decision Tree
Structure of a decision tree

- Decision tree: hierarchical structure
  - One root node: no incoming edge, zero or more outgoing edges
  - Internal nodes: exactly one incoming edge, two or more outgoing edges
  - Leaf or terminal nodes: exactly one incoming edge, no outgoing edge

- Each leaf node assigned a class label
- Each non-leaf node contains a test condition on one of the attributes
Applying a Decision Tree Classifier

### 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>

### Test Set

<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>
Apply Model to Test Data

Start from the root of tree.

Refund

Yes

NO

No

MarSt

Single, Divorced

NO

Married

TaxInc

< 80K

NO

> 80K

YES

Test Data

Refund  Marital  Taxable  Cheat
Status   Income

No  Married  80K  ?

Once a decision tree has been constructed (learned), it is easy to apply it to test data.
Apply Model to Test Data

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>

Refund

Yes

NO

No

Married

MarSt

Single, Divorced

TaxInc

< 80K

NO

> 80K

NO

YES
Apply Model to Test Data

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>

Refund
- Yes
- No
  - Yes
  - No
    - MarSt
      - Single, Divorced
      - Married
        - TaxInc
          - < 80K
            - NO
          - > 80K
            - YES
        - NO
Apply Model to Test Data

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>

Refund

Marital Status

Taxable Income

Cheat

< 80K

> 80K

Married

Single, Divorced

NO

YES

NO

NO
Apply Model to Test Data

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>

Refund

MarSt

TaxInc

< 80K

> 80K

Yes

No

Single, Divorced

Married

NO

YES
Apply Model to Test Data

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”
Learning a Decision Tree Classifier

### How to learn a decision tree?

#### Induction

- **Train Model**
  - Tree Induction algorithm

#### Deduction

- **Apply Model**
  - Decision Tree

#### 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>

#### Test Set

<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>
A Decision Tree (seen earlier)

Splitting Attributes

Training Data

Model: Decision Tree
Another Decision Tree on same dataset

There could be more than one tree that fits the same data!
Challenge in learning decision tree

- Exponentially many decision trees can be constructed from a given set of attributes
  - Some of the trees are more ‘accurate’ or better classifiers than the others
  - Finding the optimal tree is computationally infeasible

- Efficient algorithms available to learn a reasonably accurate (although potentially suboptimal) decision tree in reasonable time
  - Employs greedy strategy
  - Locally optimal choices about which attribute to use next to partition the data
Decision Tree Induction

Many Algorithms:

- Hunt’s Algorithm (one of the earliest)
- CART
- ID3, C4.5
- SLIQ, SPRINT
Let $D_t$ be the set of training records that reach a node $t$

General Procedure:

- If $D_t$ contains records that all belong the same class $y_t$, then $t$ is a leaf node labeled as $y_t$
- If $D_t$ is an empty set, then $t$ is a leaf node labeled by the default class $y_d$
- 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.
Hunt’s Algorithm

<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>

Don’t Cheat

Default class is “Don’t cheat” since it is the majority class in the dataset
Hunt’s Algorithm

For now, assume that “Refund” has been decided to be the best attribute for splitting in some way (to be discussed soon)
Hunt’s Algorithm

Refund

Don’t Cheat

Yes

No

Don’t Cheat

Refund

Don’t Cheat

Yes

No

Don’t Cheat

Marital Status

Single, Divorced

Cheat

Don’t Cheat

Married

Don’t Cheat

Refund

Don’t Cheat

Yes

No

Don’t Cheat

Tid | Refund | Marital Status | Taxable Income | Cheat
---|---|---|---|---
1 | Yes | Single | 125K | No
2 | No | Married | 100K | No
3 | No | Single | 70K | No
4 | Yes | Married | 120K | No
5 | No | Divorced | 95K | Yes
6 | No | Married | 60K | No
7 | Yes | Divorced | 220K | No
8 | No | Single | 85K | Yes
9 | No | Married | 75K | No
10 | No | Single | 90K | Yes
Hunt’s Algorithm

Don’t Cheat

Refund

→

Yes

Don’t Cheat

No

Don’t Cheat

Refund

→

Yes

Marital Status

→

Single, Divorced

Cheat

Don’t Cheat

No

Married

Refund

→

Yes

Marital Status

→

Single, Divorced

Don’t Cheat

Cheat

Married

Taxable Income

< 80K

Don’t Cheat

Cheat

=> 80K

Marital Status

Tid | Refund | Marital Status | Taxable Income | Cheat
---|--------|----------------|----------------|-------
1   | Yes    | Single         | 125K           | No    |
2   | No     | Married        | 100K           | No    |
3   | No     | Single         | 70K            | No    |
4   | Yes    | Married        | 120K           | No    |
5   | No     | Divorced       | 95K            | Yes   |
6   | No     | Married        | 60K            | No    |
7   | Yes    | Divorced       | 220K           | No    |
8   | No     | Single         | 85K            | Yes   |
9   | No     | Married        | 75K            | No    |
10  | No     | Single         | 90K            | Yes   |
Tree Induction

- **Greedy strategy**
  - Split the records based on an attribute test that optimizes certain criterion

- **Issues**
  - Determine how to split the records
    - How to specify the attribute test condition?
    - How to determine the best split?
  - Determine when to stop splitting
Tree Induction

● Greedy strategy
  – Split the records based on an attribute test that optimizes certain criterion

● Issues
  – Determine how to split the records
    ◆ How to specify the attribute test condition?
    ◆ How to determine the best split?
  – Determine when to stop splitting
How to Specify Test Condition?

- Depends on attribute types
  - **Nominal**: two or more distinct values (special case: binary) E.g., marital status: {single, divorced, married}
  - **Ordinal**: two or more distinct values that have an ordering. E.g. shirt size: {S, M, L, XL}
  - **Continuous**: continuous range of values

- 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.

```
CarType
  Family
  Sports
  Luxury
```

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

```
CarType
  {Sports, Luxury}
  {Family}
```

OR
```
CarType
  {Family, Luxury}
  {Sports}
```
Multi-way split: Use as many partitions as distinct values.

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

What about this split?
Splitting Based on Continuous Attributes

- Different ways of handling
  - **Discretization** to form an ordinal categorical attribute
    - Static – discretize once at the beginning
    - Dynamic – ranges can be found by equal interval bucketing, equal frequency bucketing (percentiles), or clustering.
  - **Binary Decision**: \((A < v)\) or \((A \geq v)\)
    - consider all possible splits and finds the best cut
    - can be more compute intensive
Splitting Based on Continuous Attributes

(i) Binary split

Taxable Income > 80K?
- Yes
- No

(ii) Multi-way split

Taxable Income?
- < 10K
- [10K, 25K)
- [25K, 50K)
- [50K, 80K)
- > 80K
Tree Induction

- Greedy strategy.
  - Split the records based on an attribute test that optimizes certain criterion.

- Issues
  - Determine how to split the records
    - How to specify the attribute test condition?
    - How to determine the best split?
  - Determine when to stop splitting
What is meant by “determine 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

- Greedy approach:
  - Nodes with **homogeneous class distribution** are preferred

- Need a measure of node impurity:

<table>
<thead>
<tr>
<th>C0</th>
<th>C1</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>

Non-homogeneous, High degree of impurity

Homogeneous, Low degree of impurity
Measures of Node Impurity

- Gini Index
- Entropy
- Misclassification error
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>N00</td>
<td>N01</td>
</tr>
<tr>
<td>N10</td>
<td>N11</td>
<td>N12</td>
</tr>
<tr>
<td>N20</td>
<td>N21</td>
<td>N22</td>
</tr>
<tr>
<td>N30</td>
<td>N31</td>
<td>N32</td>
</tr>
<tr>
<td>N40</td>
<td>N41</td>
<td>N42</td>
</tr>
</tbody>
</table>

Gain = M0 – M12 vs M0 – M34
Measures of Node Impurity

- Gini Index
- Entropy
- Misclassification error
Measure of Impurity: GINI Index

- Gini Index for a given node $t$:

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

$p(j | t)$ is the relative frequency of class $j$ at node $t$
Examples for computing GINI

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

<table>
<thead>
<tr>
<th></th>
<th>P(C1)</th>
<th>P(C2)</th>
<th>Gini</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>0/6</td>
<td>6/6</td>
<td>1 - 0^2 - 1 = 0</td>
</tr>
<tr>
<td>C2</td>
<td>6/6</td>
<td>0/6</td>
<td>1 - 6/6^2 - 0/6^2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>P(C1)</th>
<th>P(C2)</th>
<th>Gini</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>1/6</td>
<td>5/6</td>
<td>1 - (1/6)^2 - (5/6)^2 = 0.278</td>
</tr>
<tr>
<td>C2</td>
<td>5/6</td>
<td>1/6</td>
<td>1 - (5/6)^2 - (1/6)^2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>P(C1)</th>
<th>P(C2)</th>
<th>Gini</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>2/6</td>
<td>4/6</td>
<td>1 - (2/6)^2 - (4/6)^2 = 0.444</td>
</tr>
<tr>
<td>C2</td>
<td>4/6</td>
<td>2/6</td>
<td></td>
</tr>
</tbody>
</table>
Measure of Impurity: GINI Index

- Gini Index for a given node $t$:

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

- $p(j \mid t)$ 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 [$n_c$: number of classes]

- 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>Gini</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>0</td>
<td>6</td>
<td>0.000</td>
</tr>
<tr>
<td>C2</td>
<td>6</td>
<td>5</td>
<td>0.278</td>
</tr>
<tr>
<td>Gini</td>
<td>0.000</td>
<td>0.278</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
<th>Gini</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>1</td>
<td>5</td>
<td>0.444</td>
</tr>
<tr>
<td>C2</td>
<td>5</td>
<td>4</td>
<td>0.500</td>
</tr>
<tr>
<td>Gini</td>
<td>0.444</td>
<td>0.500</td>
<td></td>
</tr>
</tbody>
</table>
Splitting Based on GINI

- Used in CART, SLIQ, SPRINT.
- 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 $p$. 
Binary Attributes: Computing GINI Index

- Splits into two partitions
- Effect of Weighing partitions:
  - Larger and Purer Partitions are sought for.

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

\[
\begin{array}{c|c|c|c}
\text{Parent} & \text{N1} & \text{N2} & \text{Gini(Children)} \\
\hline
\text{C1} & 5 & 1 & 7/12 \times 0.408 + 5/12 \times 0.32 \\
\text{C2} & 2 & 4 & = 0.371
\end{array}
\]

\[\text{Gini} = 0.500\]
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! Repetition of work.

<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>
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>Yes</th>
<th>No</th>
<th>No</th>
<th>No</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></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>

<table>
<thead>
<tr>
<th>Taxable Income</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
</tr>
<tr>
<td>55</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;=</th>
<th>&gt;</th>
<th>&lt;=</th>
<th>&gt;</th>
<th>&lt;=</th>
<th>&gt;</th>
<th>&lt;=</th>
<th>&gt;</th>
<th>&lt;=</th>
<th>&gt;</th>
<th>&lt;=</th>
<th>&gt;</th>
</tr>
</thead>
<tbody>
<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>
Measures of Node Impurity

- Gini Index
- Entropy
- Misclassification error
Alternative Splitting Criteria based on INFO

- Entropy at a given node $t$:

$$\text{Entropy}(t) = -\sum_j p(j \mid t) \log_2 p(j \mid t)$$

$p(j \mid t)$ is the relative frequency of class $j$ at node $t$

- Measures homogeneity of a node
Examples for computing Entropy

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

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>P(C1) = 0/6 = 0</td>
<td>P(C2) = 6/6 = 1</td>
<td></td>
</tr>
<tr>
<td>Entropy = – 0 log 0 – 1 log 1 = – 0 – 0 = 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>P(C1) = 1/6</td>
<td>P(C2) = 5/6</td>
<td></td>
</tr>
<tr>
<td>Entropy = – (1/6) log₂ (1/6) – (5/6) log₂ (1/6) = 0.65</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>P(C1) = 2/6</td>
<td>P(C2) = 4/6</td>
<td></td>
</tr>
<tr>
<td>Entropy = – (2/6) log₂ (2/6) – (4/6) log₂ (4/6) = 0.92</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
Alternative Splitting Criteria based on INFO

- Entropy at a given node $t$:

$$\text{Entropy}(t) = -\sum_j p(j \mid t) \log_2 p(j \mid t)$$

$p(j \mid t)$ is the relative frequency of class $j$ at node $t$

- Measures homogeneity of a node
  - Maximum ($\log n_c$) when records are equally distributed among all classes implying least information
  - Minimum (0.0) when all records belong to one class, implying most information
Splitting Based on INFO...

- 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)
- Used in ID3 and C4.5
- Disadvantage: Tends to prefer splits that result in large number of partitions, each being small but pure.
**Splitting Based on INFO...**

- **Gain Ratio:**

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

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

  Parent Node, p is split into k partitions
  
  \(n_i\) is the number of records in partition i

  - Adjusts Information Gain by the entropy of the partitioning (SplitINFO). Higher entropy partitioning (large number of small partitions) is penalized!

  - Used in C4.5

  - Designed to overcome the disadvantage of Information Gain
Measures of Node Impurity

- Gini Index
- Entropy
- Misclassification error
Splitting Criteria based on Classification Error

- Classification error at a node $t$:

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

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

- Measures misclassification error made by a node
Examples for Computing Error

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

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

\(P(C1) = 1/6\), \(P(C2) = 5/6\)
Error = 1 – max \((1/6, 5/6)\) = 1 – 5/6 = 1/6

\(P(C1) = 2/6\), \(P(C2) = 4/6\)
Error = 1 – max \((2/6, 4/6)\) = 1 – 4/6 = 1/3
Splitting Criteria based on 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:
- 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.
Comparison among Splitting Criteria

For a 2-class problem:
Tree Induction

- Greedy strategy.
  - Split the records based on an attribute test that optimizes certain criterion.

- Issues
  - Determine how to split the records
    ◆ How to specify the attribute test condition?
    ◆ How to determine the best split?
  - Determine when to stop splitting
Stopping Criteria for Tree Induction

- Stop expanding a node when all the records belong to the same class.

- Stop expanding a node when all the records have similar attribute values (if different class values, then usually assign the majority class).

- Early termination, usually to prevent overfitting (to be discussed later).
DT classification: points to note

- Finding an optimal DT is NPC, but efficient and fast heuristic methods available

- Advantages:
  - Extremely fast at classifying unknown records
  - Easy to interpret, especially for small-sized trees
  - Accuracy is comparable to other classification techniques for many simple data sets
DT classification: points to note

- In what we discussed till now, the test condition always involved a single attribute
  - Decision boundaries are ‘rectilinear’ i.e., parallel to ‘coordinate axes’ of the feature space
  - Limits the expressiveness of DTs
- Oblique DTs – allows test conditions that involve more than one attribute (e.g., \( x + y < 1 \))
  - Better expressiveness
  - But finding a good tree is computationally more expensive
• 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
Oblique Decision Trees

- Test condition may involve multiple attributes
- More expressive representation
- Finding optimal test condition is computationally expensive

\[ x + y < 1 \]
Example: C4.5

- Simple depth-first construction.
- Uses Information Gain
- Sorts Continuous Attributes at each node.
- Needs entire data to fit in memory.
- Unsuitable for Large Datasets.
  - Needs out-of-core sorting.

You can download the software from:
http://www.cse.unsw.edu.au/~quinlan/c4.5r8.tar.gz
Practical issues of Decision Tree classifier
Underfitting and Overfitting (Example)

500 circular and 500 triangular data points.

Circular points:
\[ 0.5 \leq \sqrt{x_1^2 + x_2^2} \leq 1 \]

Triangular points:
\[ \sqrt{x_1^2 + x_2^2} > 0.5 \text{ or } \sqrt{x_1^2 + x_2^2} < 1 \]
**Underfitting and Overfitting**

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

**Overfitting**: DT has grown too large, and is now fitting the noise in the dataset
Overfitting

- Overfitting results in decision trees that are more complex than necessary

- Training error no longer provides a good estimate of how well the tree will perform on previously unseen records
Overfitting due to Noise

Decision boundary is distorted by noise point
Overfitting due to Insufficient Examples

Lack of data points in the lower half of the diagram makes it difficult to predict correctly the class labels of that region.

- Insufficient number of training records in the region causes the decision tree to predict the test examples using other training records that are irrelevant to the classification task.
Occam’s Razor

- Given two models of similar generalization errors, one should prefer the simpler model over the more complex model.

- For complex models, there is a greater chance that it was fitted accidentally by errors in data.

- Therefore, one should include model complexity when evaluating a model.
Minimum Description Length (MDL)

- \( \text{Cost}(\text{Model,Data}) = \text{Cost}(\text{Data|Model}) + \text{Cost}(\text{Model}) \)
  - Cost is the number of bits needed for encoding.
  - Search for the least costly model.
- \( \text{Cost}(\text{Data|Model}) \) encodes the misclassification errors.
- \( \text{Cost}(\text{Model}) \) uses node encoding (number of children) plus splitting condition encoding.
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
  - More restrictive conditions:
    - Stop if number of instances is less than some user-specified threshold
    - Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain)
How to Address Overfitting...

- **Post-pruning**
  - Grow decision tree to its entirety
  - Trim the nodes of the decision tree in a bottom-up fashion
  - 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
  - Can use MDL for post-pruning
Other Issues

- Data Fragmentation
- Search Strategy
- Expressiveness
- Tree Replication
Data Fragmentation

- Number of instances gets smaller as you traverse down the tree
- Number of instances at the leaf nodes could be too small to make any statistically significant decision
Search Strategy

- Finding an optimal decision tree is NP-hard
- The algorithm presented so far uses a greedy, top-down, recursive partitioning strategy to induce a reasonable solution
- Other strategies?
  - Bottom-up
  - Bi-directional
Expressiveness

- Decision tree provides expressive representation for learning discrete-valued function
  - But they do not generalize well to certain types of Boolean functions
    - Example: parity function:
      - Class = 1 if there is an even number of Boolean attributes with truth value = True
      - Class = 0 if there is an odd number of Boolean attributes with truth value = True
    - For accurate modeling, must have a complete tree

- Not expressive enough for modeling continuous variables
  - Particularly when test condition involves only a single attribute at-a-time
Tree Replication

- Same subtree appears in multiple branches