Classification:
Basic Concepts and Techniques
Classification problem

- **What we have**
  - A set of objects, each of them described by some features
    - people described by age, gender, height, etc.
    - bank transactions described by type, amount, time, etc.

- **What we want to do**
  - Associate the objects of a set to a class, taken from a predefined list
    - “good customer” vs. “churner”
    - “normal transaction” vs. “fraudulent”
    - “low risk patient” vs. “risky”
Classification problem

- What we know
  - No domain knowledge or theory
  - Only examples: Training Set
    🚚 Subset of labelled objects

- What we can do
  - Learn from examples
  - Make inferences about the other objects
The most stupid classifier

- **Rote learner**
  - To classify object X, check if there is a labelled example in the training set identical to X
  - Yes □ X has the same label
  - No □ I don’t know
Classify by similarity

- K-Nearest Neighbors
  - Decide label based on K most similar examples
Build a model

- Example 1: linear separation line
Build a model

- Example 2: Support Vector Machine (linear)
Build a model

- Example 3: Non-linear separation line
Build a model

- Decision Trees

```
<table>
<thead>
<tr>
<th>Income &gt; 15k€?</th>
<th>Age &gt; 50y?</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
```

**Diagram:**
- Income > 15k€?
  - Yes: Age > 50y
    - Yes: [Leaf or Condition]
    - No: [Leaf or Condition]
  - No: [Leaf or Condition]
Classification: Definition

- Given a collection of records (training set)
  - Each record is characterized by a tuple \((x,y)\), where \(x\) is the attribute set and \(y\) is the class label
    - \(x\): attribute, predictor, independent variable, input
    - \(y\): class, response, dependent variable, output
- **Task**: Learn a model that maps each attribute set \(x\) into one of the predefined class labels \(y\)
- **Goal**: previously unseen records should be assigned a class as accurately as possible.
  - A test set is used to determine the accuracy of the model. Usually, the given data set is divided into training and test sets, with training set used to build the model and test set used to validate it.
Supervised learning refers to problems where the value of a target attribute should be predicted based on the values of other attributes.

Unsupervised learning (e.g. Cluster analysis) is not concerned with a specific target attribute.

Problems with a categorical target attribute are called classification, problems with a numerical target attribute are called regression.
General Approach for Building Classification Model

**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>87K</td>
<td>?</td>
</tr>
</tbody>
</table>

**Learning Algorithm**

**Induction**

**Apply Model**

**Deduction**
## Examples of Classification Task

<table>
<thead>
<tr>
<th>Task</th>
<th>Attribute set, ( x )</th>
<th>Class label, ( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Categorizing email messages</td>
<td>Features extracted from email message header and content</td>
<td>spam or non-spam</td>
</tr>
<tr>
<td>Identifying tumor cells</td>
<td>Features extracted from MRI scans</td>
<td>malignant or benign cells</td>
</tr>
<tr>
<td>Cataloging galaxies</td>
<td>Features extracted from telescope images</td>
<td>Elliptical, spiral, or irregular-shaped galaxies</td>
</tr>
</tbody>
</table>
Classification Techniques

- Base Classifiers
  - Decision Tree based Methods
  - Rule-based Methods
  - Nearest-neighbor
  - Neural Networks
  - Deep Learning
  - Naïve Bayes and Bayesian Belief Networks
  - Support Vector Machines

- Ensemble Classifiers
  - Boosting, Bagging, Random Forests
Consider the problem of predicting whether a loan borrower will repay the loan or default on the loan payments.

<table>
<thead>
<tr>
<th>ID</th>
<th>Home Owner</th>
<th>Marital Status</th>
<th>Annual Income</th>
<th>Defaulted Borrower</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
Another Example of Decision Tree

<table>
<thead>
<tr>
<th>ID</th>
<th>Home Owner</th>
<th>Marital Status</th>
<th>Annual Income</th>
<th>Defaulted Borrower</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>

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

Start from the root of tree.

- **Home Owner**
  - Yes
    - **NO**
  - No
    - **MarSt**
      - Single, Divorced
        - **NO**
      - Married
        - **Income**
          - < 80K
            - **NO**
          - > 80K
            - **YES**

**Test Data**

<table>
<thead>
<tr>
<th>Home Owner</th>
<th>Marital Status</th>
<th>Annual Income</th>
<th>Defaulted Borrower</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Married</td>
<td>80K</td>
<td>?</td>
</tr>
</tbody>
</table>

*
Apply Model to Test Data

Test Data

<table>
<thead>
<tr>
<th>Home Owner</th>
<th>Marital Status</th>
<th>Annual Income</th>
<th>Defaulted Borrower</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Married</td>
<td>80K</td>
<td>?</td>
</tr>
</tbody>
</table>

*
Apply Model to Test Data

Test Data

- **Home Owner**: No
- **Marital Status**: Married
- **Annual Income**: 80K
- **Defaulted Borrower**: ?

**Decision Tree**

- **Home Owner**: No
- **Marital Status**: Married
- **Income**: < 80K

**Outcome**: No
Apply Model to Test Data

Test Data

<table>
<thead>
<tr>
<th>Home Owner</th>
<th>Marital Status</th>
<th>Annual Income</th>
<th>Defaulted Borrower</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Married</td>
<td>80K</td>
<td>?</td>
</tr>
</tbody>
</table>

No

Home Owner

Married

MarSt

< 80K

> 80K

Income

NO

Single, Divorced

NO

NO

NO

YES
Apply Model to Test Data

Test Data

<table>
<thead>
<tr>
<th>Home Owner</th>
<th>Marital Status</th>
<th>Annual Income</th>
<th>Defaulted Borrower</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Married</td>
<td>80K</td>
<td>?</td>
</tr>
</tbody>
</table>

Diagram:
- Yes Home Owner -> No
- No Home Owner
  - Marital Status: Single, Divorced
    - Income: < 80K -> No
    - Income: > 80K
      - Married
        - Yes
        - No
Apply Model to Test Data

Test Data

<table>
<thead>
<tr>
<th>Home Owner</th>
<th>Marital Status</th>
<th>Annual Income</th>
<th>Defaulted Borrower</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Married</td>
<td>80K</td>
<td>?</td>
</tr>
</tbody>
</table>

Assign Defaulted to “No”
Decision Tree Classification Task

Training Set

<table>
<thead>
<tr>
<th>Tid</th>
<th>Attrb1</th>
<th>Attrb2</th>
<th>Attrb3</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>Attrb1</th>
<th>Attrb2</th>
<th>Attrb3</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>
Build a model

- Decision Trees

- Income > 15k€?
- Age > 50y?
Decision Tree Induction

- Many Algorithms:
  - Hunt’s Algorithm (one of the earliest)
  - CART
  - ID3, C4.5
  - SLIQ, SPRINT
General Structure of Hunt’s Algorithm

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

- General Procedure:
  - If $D_t$ contains records that belong 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.

<table>
<thead>
<tr>
<th>ID</th>
<th>Home Owner</th>
<th>Marital Status</th>
<th>Annual Income</th>
<th>Defaulted Borrower</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

Defaultered = No

(7,3)

(a)

<table>
<thead>
<tr>
<th>ID</th>
<th>Home Owner</th>
<th>Marital Status</th>
<th>Annual Income</th>
<th>Defaulted Borrower</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

(a)

(b)

<table>
<thead>
<tr>
<th>ID</th>
<th>Home Owner</th>
<th>Marital Status</th>
<th>Annual Income</th>
<th>Defaulted Borrower</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

(a) Defaulted = No
(3,0)

(b) Home Owner
Yes
(3,0)
No
(4,3)

(c) Home Owner
Yes
Defaulted = No
(3,0)
Marital Status
Single, Divorced
Defaulted = Yes
(1,3)
Married
Defaulted = No
(3,0)

ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower
---|------------|----------------|---------------|-------------------
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

(a) Defaulted = No

(b) Home Owner
   Yes
   No

   (3,0)
   Defaulted = No
   (4,3)
   Defaulted = No

(c) Home Owner
   Yes
   No
       Defaulted = Yes
       Defaulted = No

   (3,0)
   Marital Status
   Single, Divorced
       Defaulted = Yes
       Defaulted = No

   (1,3)
   Married

(d) Marital Status
   Yes
   No

   (3,0)
   Single, Divorced

   (3,0)
   Annual Income
   < 80K
   >= 80K

   (1,0)
   Defaulted = No
   (0,3)
   Defaulted = Yes

Table:

<table>
<thead>
<tr>
<th>ID</th>
<th>Home Owner</th>
<th>Marital Status</th>
<th>Annual Income</th>
<th>Defaulted Borrower</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>
Design Issues of Decision Tree Induction

- **Greedy strategy:**
  - the number of possible decision trees can be very large, many decision tree algorithms employ a heuristic-based approach to guide their search in the vast hypothesis space.
  
  - **Split the records based on an attribute test that optimizes certain criterion.**
Tree Induction

- How should training records be split?
  - Method for specifying test condition
    - depending on attribute types
  - Measure for evaluating the goodness of a test condition

- How should the splitting procedure stop?
  - Stop splitting if all the records belong to the same class or have identical attribute values
  - Early termination
How to specify the attribute test condition?
Methods for Expressing Test Conditions

- Depends on attribute types
  - Binary
  - Nominal
  - Ordinal
  - Continuous

- Depends on number of ways to split
  - 2-way split
  - Multi-way split
Test Condition for Nominal Attributes

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

- **Binary split:**
  - Divides values into two subsets
Test Condition for Ordinal Attributes

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

- **Binary split:**
  - Divides values into two subsets
  - Preserve order property among attribute values

This grouping violates order property
Test Condition for Continuous Attributes

(i) Binary split

Annual Income > 80K?

Yes
No

(ii) Multi-way split

Annual 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
    - Ranges can be found by equal interval bucketing, equal frequency bucketing (percentiles), or clustering.
    - Static – discretize once at the beginning
    - Dynamic – repeat at each node

  - **Binary Decision**: $(A < v)$ or $(A \geq v)$
    - consider all possible splits and finds the best cut
    - can be more compute intensive
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?
How to determine the Best Split

- Greedy approach:
  - Nodes with **purer / 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>

  - High degree of impurity, Non-homogeneous
  - Low degree of impurity, Homogeneous
Measures of Node Impurity

- Gini Index

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

- Entropy

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

- Misclassification error

\[ Error(t) = 1 - \max_i P(i \mid t) \]
Finding the Best Split

1. Compute impurity measure (P) before splitting
2. Compute impurity measure (M) after splitting
   - Compute impurity measure of each child node
   - M is the weighted sum of impurity of children
3. Choose the attribute test condition that produces the highest gain
   \[
   \text{Gain} = P - M
   \]
   or equivalently, lowest impurity measure after splitting (M)
Finding the Best Split

Before Splitting:

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

A?

Yes

Node N1

C0  N10
C1  N11

No

Node N2

C0  N20
C1  N21

B?

Yes

Node N3

C0  N30
C1  N31

No

Node N4

C0  N40
C1  N41

Gain = P – M1 vs P – M2

*
Measure of Impurity: GINI

- Gini Index for a given node $t$:

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

(NOTE: $p(j \mid t)$ 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**

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

  For 2-class problem ($p, 1 - p$):
  - GINI = $1 - p^2 - (1 - p)^2 = 2p(1-p)$

<table>
<thead>
<tr>
<th>$C1$</th>
<th>0</th>
<th>$C1$</th>
<th>1</th>
<th>$C1$</th>
<th>2</th>
<th>$C1$</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C2$</td>
<td>6</td>
<td>$C2$</td>
<td>5</td>
<td>$C2$</td>
<td>4</td>
<td>$C2$</td>
<td>3</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>
</tr>
</tbody>
</table>
Computing Gini Index of a Single Node

\[
GINI(t) = 1 - \sum_{j} \left[ p(j \mid t) \right]^2
\]

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

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>P(C1)</th>
<th>P(C2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>1</td>
<td>1/6</td>
<td>5/6</td>
</tr>
<tr>
<td>C2</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Gini = 1 – (1/6)^2 – (5/6)^2 = 0.278

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>P(C1)</th>
<th>P(C2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>2</td>
<td>2/6</td>
<td>4/6</td>
</tr>
<tr>
<td>C2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Gini = 1 – (2/6)^2 – (4/6)^2 = 0.444
Gini Index for a Collection of Nodes

- When a node $p$ is split into $k$ partitions (children)

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

where, $n_i = \text{number of records at child } i$, $n = \text{number of records at parent node } p$.

- Choose the attribute that minimizes weighted average Gini index of the children

- Gini index is used in decision tree algorithms such as CART, SLIQ, SPRINT
Binary Attributes: Computing GINI Index

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

\[
\text{Gini(N1)} = 1 - \left(\frac{5}{6}\right)^2 - \left(\frac{1}{6}\right)^2 = 0.278
\]

\[
\text{Gini(N2)} = 1 - \left(\frac{2}{6}\right)^2 - \left(\frac{4}{6}\right)^2 = 0.444
\]

Weighted Gini of N1 N2
\[
= \frac{6}{12} \times 0.278 + \frac{6}{12} \times 0.444
= 0.361
\]

Gain = 0.486 – 0.361 = 0.125
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>8</td>
<td>1</td>
</tr>
<tr>
<td>C2</td>
<td>3</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Gini</td>
<td>0.163</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Two-way split

<table>
<thead>
<tr>
<th>CarType</th>
<th>{Sports, Luxury}</th>
<th>{Family}</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>C2</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>Gini</td>
<td>0.468</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>8</td>
<td>2</td>
</tr>
<tr>
<td>C2</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Gini</td>
<td>0.167</td>
<td></td>
</tr>
</tbody>
</table>

Which of these is the best?
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! ($O(N^2)$)

Repetition of work.

<table>
<thead>
<tr>
<th>ID</th>
<th>Home Owner</th>
<th>Marital Status</th>
<th>Annual Income</th>
<th>Defaulted</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>

Annual Income?

<table>
<thead>
<tr>
<th></th>
<th>$\leq 80$</th>
<th>$&gt; 80$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defaulted Yes</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Defaulted No</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

$*$
For efficient computation $O(N\log N)$: 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
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>Split Positions</th>
<th>Sorted Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cheat</td>
<td>No</td>
</tr>
<tr>
<td>Annual Income</td>
<td>60 70 75 85 90 95 100 120 125 220</td>
</tr>
<tr>
<td>&lt;= &gt;</td>
<td>&lt;= &gt; &lt;= &gt; &lt;= &gt; &lt;= &gt; &lt;= &gt; &lt;= &gt; &lt;= &gt; &lt;= &gt; &lt;= &gt;</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>No</th>
<th>No</th>
<th>No</th>
<th>No</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted Values</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>Annual Income</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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Split Positions</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>
<th>&lt;=</th>
<th>&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>0</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gini</td>
<td>0.343</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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
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>Sorted Values</td>
<td>Annual Income</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorted Values</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>220</td>
<td></td>
</tr>
<tr>
<td>Split Positions</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>&lt;= &gt;</td>
<td>&lt;= &gt;</td>
<td>&lt;= &gt;</td>
<td>&lt;= &gt;</td>
<td>&lt;= &gt;</td>
<td>&lt;= &gt;</td>
<td>&lt;= &gt;</td>
<td>&lt;= &gt;</td>
<td>&lt;= &gt;</td>
<td>&lt;= &gt;</td>
<td>&lt;= &gt;</td>
<td></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>
</tbody>
</table>
Measure of Impurity: Entropy

- Entropy at a given node $t$:

$$Entropy(t) = -\sum_j p(j \mid t) \log p(j \mid t)$$

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

- 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.

- Entropy based computations are quite similar to the GINI index computations.
## Computing Entropy of a Single Node

The entropy of a node can be calculated using the formula:

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

### Example Calculations

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

1. **P(C1) = 0/6 = 0, P(C2) = 6/6 = 1**
   - Entropy: $- 0 \log_2 0 - 1 \log_2 1 = - 0 - 0 = 0$

2. **P(C1) = 1/6, P(C2) = 5/6**
   - Entropy: $- (1/6) \log_2 (1/6) - (5/6) \log_2 (1/6) = 0.65$

3. **P(C1) = 2/6, P(C2) = 4/6**
   - Entropy: $- (2/6) \log_2 (2/6) - (4/6) \log_2 (4/6) = 0.92$
Computing Information Gain After Splitting

- Information Gain:

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

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 the ID3 and C4.5 decision tree algorithms
- **Disadvantage**: Tends to prefer splits that result in large number of partitions, each being small but pure.
Problem with large number of partitions

- Node impurity measures tend to prefer splits that result in large number of partitions, each being small but pure

  - Customer ID has highest information gain because entropy for all the children is zero
  
  - Can we use such a test condition on new test instances?
Solution

- A **low impurity value alone is insufficient** to find a good attribute test condition for a node
- **Solution**: Consider the **number of children** produced by the splitting attribute in the identification of the best split
- High number of child nodes implies more complexity

- **Method 1**: Generate only binary decision trees
  - This strategy is employed by decision tree classifiers such as CART
- **Method 2**: Modify the splitting criterion to take into account the number of partitions produced by the attribute
Gain Ratio

- Gain Ratio:

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

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

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 algorithm
- Designed to overcome the disadvantage of Information Gain
Gain Ratio

- Gain Ratio:

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

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

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

<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>8</td>
<td>1</td>
</tr>
<tr>
<td>C2</td>
<td>3</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Gini</td>
<td>0.163</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\text{SplitINFO} = 1.52

<table>
<thead>
<tr>
<th>CarType</th>
<th>{Sports, Luxury}</th>
<th>{Family}</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>C2</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>Gini</td>
<td>0.468</td>
<td></td>
</tr>
</tbody>
</table>

\text{SplitINFO} = 0.72

<table>
<thead>
<tr>
<th>CarType</th>
<th>{Sports}</th>
<th>{Family, Luxury}</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>C2</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Gini</td>
<td>0.167</td>
<td></td>
</tr>
</tbody>
</table>

\text{SplitINFO} = 0.97
Measure of Impurity: Classification Error

- Classification error at a node $t$:

$$\text{Error}(t) = 1 - \max_{i} P(i \mid t)$$

- Maximum $(1 - 1/n_c)$ when records are equally distributed among all classes, implying least interesting information
- Minimum $(0)$ when all records belong to one class, implying most interesting information
Computing Error of a Single Node

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

<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>
Weighted Misclassification Error

- Easy to compute

\[
M_1 = \text{Error(N1)} = \frac{\#\text{minority(N1)}}{\#N1}
\]

\[
M_2 = \text{Error(N2)} = \frac{\#\text{minority(N2)}}{\#N2}
\]

\[
M = \text{weighted avg (M1, M2)} = \frac{\#\text{minority(N1)}}{N} + \frac{\#\text{minority(N2)}}{N} = \frac{\#\text{minority(N1 and N2)}}{N}
\]

- The aggregate error M is equivalent to the fraction of errors over both nodes after the split
Comparison among Impurity Measures

For a 2-class problem:

Consistency among the impurity measures
• if a node N1 has lower entropy than node N2, then the Gini index and error rate of N1 will also be lower than that of N2

The attribute chosen as splitting criterion by the impurity measures can still be different!
Misclassification Error vs Gini Index

Gini(N1) = 1 – (3/3)^2 – (0/3)^2 = 0

Gini(N2) = 1 – (4/7)^2 – (3/7)^2 = 0.489

Gini(Children) = 3/10 * 0 + 7/10 * 0.489 = 0.342

Gini improves but error remains the same!!
Misclassification Error vs Gini Index

<table>
<thead>
<tr>
<th>Parent</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>7</td>
</tr>
<tr>
<td>C2</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ \text{Gini} = 0.42 \]

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>3</td>
</tr>
<tr>
<td>C2</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ \text{Gini} = 0.342 \]

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>3</td>
</tr>
<tr>
<td>C2</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ \text{Gini} = 0.416 \]

Misclassification error for all three cases = 0.3 !
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
- Early termination (to be discussed later)
Algorithms: ID3, C4.5, C5.0, CART

- **ID3** uses the Hunt’s algorithm with information gain criterion and gain ratio.

- **C4.5** improves **ID3**
  - Needs entire data to fit in memory
  - Handles missing attributes and continuous attributes
  - Performs tree post-pruning
  - **C5.0** is the current commercial successor of **C4.5**
  - Unsuitable for Large Datasets

- **CART** builds multivariate decision (binary) trees
Advantages of Decision Tree

- Easy to interpret for small-sized trees
- Accuracy is comparable to other classification techniques for many simple data sets
- Robust to noise (especially when methods to avoid overfitting are employed)
- Can easily handle redundant or irrelevant attributes
- Inexpensive to construct
- Extremely fast at classifying unknown record
- Handle Missing Values
Irrelevant Attributes

- **Irrelevant** attributes are poorly associated with the target class labels, so they have little or no gain in purity.

- In case of a large number of irrelevant attributes, some of them may be accidentally chosen during the tree-growing process.

- Feature selection techniques can help to eliminate the irrelevant attributes during preprocessing.
Redundant Attributes

- Decision trees can handle the presence of redundant attributes.

- An attribute is **redundant** if it is strongly **correlated** with another attribute in the data.

- Since redundant attributes show **similar gains** in purity if they are selected for splitting, **only one** of them will be selected as an attribute test condition in the decision tree algorithm.
Advantages of Decision Tree

- Easy to interpret for small-sized trees
- Accuracy is comparable to other classification techniques for many simple data sets
- Robust to noise (especially when methods to avoid overfitting are employed)
- Can easily handle redundant or irrelevant attributes
- Inexpensive to construct
- Extremely fast at classifying unknown record
- Handle Missing Values
Computational Complexity

- Finding an optimal decision tree is NP-hard.
- Hunt’s Algorithm uses a greedy, top-down, recursive partitioning strategy for growing a decision tree.
- Such techniques quickly construct a reasonably good decision tree even when the training set size is very large.
- **Construction DT Complexity**: $O(M \times N \log N)$ where $M=n$. attributes, $N=n$. instances.
- Once a decision tree has been built, classifying a test record is extremely fast, with a worst-case complexity of $O(w)$, where $w$ is the maximum depth of the tree.
Advantages of Decision Tree

- Easy to interpret for small-sized trees
- Accuracy is comparable to other classification techniques for many simple data sets
- Robust to noise (especially when methods to avoid overfitting are employed)
- Can easily handle redundant or irrelevant attributes
- Inexpensive to construct
- Extremely fast at classifying unknown record
- Handle Missing Values
Handling Missing Attribute Values

- Missing values affect decision tree construction in three different ways:
  - Affects how impurity measures are computed
  - Affects how to distribute instance with missing value to child nodes
  - Affects how a test instance with missing value is classified
Computing Impurity Measure
(DT construction)

Before Splitting:
Entropy(Parent) = \(-0.3 \log(0.3) - (0.7)\log(0.7) = 0.8813\)

<table>
<thead>
<tr>
<th>Tid</th>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Class</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>?</td>
<td>Single</td>
<td>90K</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Split on Refund:

Entropy(Refund=Yes) = 0

Entropy(Refund=No) = -(2/6)\log(2/6) - (4/6)\log(4/6) = 0.9183

Entropy(Children) = 0.3 \times 0 + 0.6 \times 0.9183 = 0.551

Gain = 0.9 \times (0.8813 - 0.551) = 0.3303
Distribute Instances
(DT construction)

<table>
<thead>
<tr>
<th>Tid</th>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Class</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>
</tbody>
</table>

Probability that Refund=Yes is 3/9
Probability that Refund=No is 6/9
Assign record to the left child with weight = 3/9 and to the right child with weight = 6/9

Refund

<table>
<thead>
<tr>
<th>Tid</th>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>?</td>
<td>Single</td>
<td>90K</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Class=Yes 0 + 3/9
Class=No 3

Probability that Refund=Yes is 3/9
Probability that Refund=No is 6/9
Assign record to the left child with weight = 3/9 and to the right child with weight = 6/9
New record:

<table>
<thead>
<tr>
<th>Tid</th>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>No</td>
<td>?</td>
<td>85K</td>
<td>?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Married</th>
<th>Single</th>
<th>Divorced</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class=No</td>
<td>3</td>
<td>1</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Class=Yes</td>
<td>1 + 6/9</td>
<td>1</td>
<td></td>
<td>2.67</td>
</tr>
<tr>
<td>Total</td>
<td>3</td>
<td>2.66</td>
<td>1</td>
<td>6.67</td>
</tr>
</tbody>
</table>

Probabilistic split method (C4.5)

Probability that Marital Status = Married is $\frac{3}{6.67} = 0.45$

Probability that Marital Status =\{Single, Divorced\} is $\frac{3.67}{6.67} = 0.55$

[ NOTICE: numbers contained errors in previous versions of this slide ]
Disadvantages

- Space of possible decision trees is exponentially large. Greedy approaches are often unable to find the best tree.
- Does not take into account **interactions** between attributes
- Each decision boundary involves **only a single attribute**
Handling interactions

**Interacting attributes:** able to distinguish between classes when used together, but individually they provide little or no information.

+ : 1000 instances

o : 1000 instances

**Test Condition:**

- $X \leq 10$ and $Y \leq 10$

- Entropy ($X$) : 0.99
- Entropy ($Y$) : 0.99

No reduction in the impurity measure when used individually.
Handling interactions

+ : 1000 instances

\(Y\)

\(Z\)

\(X\)

\(o : 1000\) instances

Entropy (\(X\)) : 0.99

Entropy (\(Y\)) : 0.99

Entropy (\(Z\)) : 0.98

Adding \(Z\) as a noisy attribute generated from a uniform distribution

Attribute \(Z\) will be chosen for splitting!
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.
Limitations of single attribute-based decision boundaries

Both positive (+) and negative (o) classes generated from skewed Gaussians with centers at (8,8) and (12,12) respectively.

Test Condition $x + y < 20$
Oblique Decision Trees

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

\[ x + y < 1 \]
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
- Solution: constrain the splittable nodes by
  - min. number of objects in each leaf
  - number nodes / leaves in the tree
Practical Issues of Classification

- Underfitting and Overfitting
- Costs of Classification
Classification Errors

- Training errors (apparent errors)
  - Errors committed on the training set

- Test errors
  - Errors committed on the test set

- Generalization errors
  - Expected error of a model over random selection of records from same distribution
Underfitting and Overfitting

**Underfitting**: when model is too simple, both training and test errors are large.
Example Data Set

Two class problem:

+ : 5200 instances
  • 5000 instances generated from a Gaussian centered at (10,10)
  • 200 noisy instances added

o : 5200 instances
  • Generated from a uniform distribution

10 % of the data used for training and 90% of the data used for testing
Increasing number of nodes in Decision Trees
Decision Tree with 4 nodes

Decision Tree

Decision boundaries on Training data

Error

Number of Nodes
Decision Tree with 50 nodes

Decision Tree

Decision boundaries on Training data

Number of Nodes

Train Error
Which tree is better?

Decision Tree with 4 nodes

Decision Tree with 50 nodes

Which tree is better?
Model Overfitting

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

**Overfitting**: when model is too complex, training error is small but test error is large
Model Overfitting

Using twice the number of data instances

- If training data is under-representative, testing errors increase and training errors decrease on increasing number of nodes.
- Increasing the size of training data reduces the difference between training and testing errors at a given number of nodes.
Model Overfitting

- If training data is under-representative, testing errors increase and training errors decrease on increasing number of nodes.
- Increasing the size of training data reduces the difference between training and testing errors at a given number of nodes.
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.
Overfitting due to Noise

Decision boundary is distorted by noise point
Notes on 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

- Need new ways for estimating errors
Model Selection

- Performed during model building
- Purpose is to ensure that model is not overly complex (to avoid overfitting)
- Need to estimate generalization error
  - Using Validation Set
  - Incorporating Model Complexity
  - Estimating Statistical Bounds
Model Selection Using Validation Set

- Divide training data into two parts:
  - Training set:
    - use for model building
  - Validation set:
    - use for estimating generalization error
    - Note: validation set is not the same as test set

- Drawback:
  - Less data available for training
Rationale: Occam’s Razor

- Given two models of similar generalization errors, one should prefer the simpler model over the more complex model
- A complex model has a greater chance of being fitted accidentally by errors in data
- Therefore, one should include model complexity when evaluating a model

\[
\text{Gen. Error(Model)} = \text{Train. Error(Model, Train. Data)} + \alpha \times \text{Complexity(Model)}
\]
Estimating Generalization Errors

- Re-substitution errors: error on training ($\Sigma \text{err}(t)$)

- Generalization errors: error on testing ($\Sigma \text{err}'(t)$)

- Methods for estimating generalization errors:
  - Pessimistic approach
  - Optimistic approach
  - Reduced error pruning (REP):
    - uses validation data set to estimate generalization error
Estimating the Complexity of Decision Trees

- **Pessimistic Error Estimate** of decision tree $T$ with $k$ leaf nodes:

  $$err_{gen}(T) = err(T) + \Omega \times \frac{k}{N_{train}}$$

  - $err(T)$: error rate on all training records
  - $\Omega$: Relative cost of adding a leaf node
  - $k$: number of leaf nodes
  - $N_{train}$: total number of training records
Estimating the Complexity of Decision Trees: Example

Decision Tree, $T_L$

$$e_{gen}(T_L) = \frac{4}{24} + \frac{1*7}{24} = \frac{11}{24} = 0.458$$

Decision Tree, $T_R$

$$e_{gen}(T_R) = \frac{6}{24} + \frac{1*4}{24} = \frac{10}{24} = 0.417$$

$e(T_L) = \frac{4}{24}$

$e(T_R) = \frac{6}{24}$

$\Omega = 1$
Estimating the Complexity of Decision Trees

- Re-substitution Estimate:
  - Using training error as an **optimistic** estimate of generalization error
  - Referred to as optimistic error estimate

Decision Tree, $T_L$

- $e(T_L) = \frac{4}{24}$

Decision Tree, $T_R$

- $e(T_R) = \frac{6}{24}$
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)

- Cost(Model, Data) = Cost(Data|Model) + Cost(Model)
  - Cost is the number of bits needed for encoding.
  - Search for the least costly model.
- Cost(Data|Model) encodes the misclassification errors.
- Cost(Model) uses node encoding (number of children) plus splitting condition encoding.
Estimating Statistical Bounds

Apply a **statistical correction** to the training error rate of the model that is indicative of its model complexity.

• Need probability distribution of training error: available or assumed.
• The number of errors committed by a leaf node in a decision tree can be assumed to follow a **binomial distribution**.

\[
e'(N, e, \alpha) = \frac{e + \frac{z_{\alpha/2}^2}{2N} + z_{\alpha/2}}{1 + \frac{z_{\alpha/2}^2}{N}} \sqrt{\frac{e(1-e)}{N} + \frac{z_{\alpha/2}^2}{4N^2}}
\]

Before splitting: \( e = 2/7 \), \( e'(7, 2/7, 0.25) = 0.503 \)
\( e'(T) = 7 \times 0.503 = 3.521 \)

After splitting:

- \( e(T_L) = 1/4 \), \( e'(4, 1/4, 0.25) = 0.537 \)
- \( e(T_R) = 1/3 \), \( e'(3, 1/3, 0.25) = 0.650 \)

\( e'(T) = 4 \times 0.537 + 3 \times 0.650 = 4.098 \)

Therefore, do not split
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 class distribution of instances are independent of the available features (e.g., using $\chi^2$ test)
    - Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain).
    - Stop if estimated generalization error falls below certain threshold
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
Example of Post-Pruning

| Class = Yes | 20 |
| Class = No | 10 |
| Error = 10/30 |

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 × 0.5)/30 = 11/30

PRUNE!
Model Evaluation
Model Evaluation

- Metrics for Performance Evaluation
  - How to evaluate the performance of a model?

- Methods for Performance Evaluation
  - How to obtain reliable estimates?

- Methods for Model Comparison
  - How to compare the relative performance among competing models?
Model Evaluation

● Metrics for Performance Evaluation
  – How to evaluate the performance of a model?

● Methods for Performance Evaluation
  – How to obtain reliable estimates?

● Methods for Model Comparison
  – How to compare the relative performance among competing models?
Metrics for Performance Evaluation

- Focus on the predictive capability of a model
  - Rather than how fast it takes to classify or build models, scalability, etc.
- Confusion Matrix:

<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>a</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>Class=No</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

a: TP (true positive)
b: FN (false negative)
c: FP (false positive)
d: TN (true negative)
Metrics for Performance Evaluation...

<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>a \ (TP)</td>
<td>b \ (FN)</td>
<td></td>
</tr>
<tr>
<td>Class=No</td>
<td>c \ (FP)</td>
<td>d \ (TN)</td>
<td></td>
</tr>
</tbody>
</table>

- Most widely-used metric:

\[
\text{Accuracy} = \frac{a + d}{a + b + c + d} = \frac{TP + TN}{TP + TN + FP + FN}
\]
Limitation of Accuracy

- Consider a 2-class problem
  - Number of Class 0 examples = 9990
  - Number of Class 1 examples = 10

- If model predicts everything to be class 0, accuracy is $9990/10000 = 99.9\%$
  - Accuracy is misleading because model does not detect any class 1 example
## Cost Matrix

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

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

## Cost Matrix

<table>
<thead>
<tr>
<th>ACTUAL CLASS</th>
<th>PREDICTED CLASS</th>
<th>$C(ij)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>+</td>
<td>-1</td>
<td>100</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

## Model $M_1$

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

Accuracy = 80%
Cost = 3910

## Model $M_2$

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

Accuracy = 90%
Cost = 4255
## Cost vs Accuracy

### Count

<table>
<thead>
<tr>
<th>ACTUAL CLASS</th>
<th>PREDICTED CLASS</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Class=Yes</td>
<td>Class=No</td>
<td></td>
</tr>
<tr>
<td>Class=Yes</td>
<td>a</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>Class=No</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

Accuracy is proportional to cost if

1. \( C(\text{Yes}|\text{No})=C(\text{No}|\text{Yes}) = q \)
2. \( C(\text{Yes}|\text{Yes})=C(\text{No}|\text{No}) = p \)

\[
N = a + b + c + d
\]

\[
\text{Accuracy} = \frac{a + d}{N}
\]

### Cost

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

\[
\text{Cost} = p (a + d) + q (b + c)
\]

\[
= p (a + d) + q (N - a - d)
\]

\[
= q N - (q - p)(a + d)
\]

\[
= N [q - (q-p) \times \text{Accuracy}]
\]
Cost-Sensitive Measures

Precision (p) = \frac{TP}{TP + FP}

Recall (r) = \frac{TP}{TP + FN}

F-measure (F) = \frac{2rp}{r + p} = \frac{2TP}{2TP + FN + FP}

- Precision is biased towards C(Yes|Yes) & C(Yes|No)
- Recall is biased towards C(Yes|Yes) & C(No|Yes)
- F-measure is biased towards all except C(No|No)

Weighted Accuracy = \frac{w_1a + w_4d}{w_1a + w_2b + w_3c + w_4d}
Model Evaluation

- **Metrics for Performance Evaluation**
  - How to evaluate the performance of a model?

- **Methods for Performance Evaluation**
  - How to obtain reliable estimates?

- **Methods for Model Comparison**
  - How to compare the relative performance among competing models?
Methods for evaluation
Parameter Tuning

- It is important that the test data is not used in any way to create the classifier.

- Some learning schemes operate in two stages:
  - **Stage 1**: builds the basic structure
  - **Stage 2**: optimizes parameter settings
  - **The test data can’t be used for parameter tuning!**
  - Proper procedure uses three sets:
    - training data,
    - validation data,
    - test data
  - **Validation data is used to optimize parameters**

- Once evaluation is complete, all the data can be used to build the final classifier.

- Generally, the larger the training data the better the classifier.

- The larger the test data the more accurate the error estimate.
Evaluation: training, validation, test
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:
  - Class distribution
  - Cost of misclassification
  - Size of training and test sets
Learning Curve

- Learning curve shows how accuracy changes with varying sample size.
- Requires a **sampling** schedule for creating learning curve:

  **Effect of small sample size:**
  - Bias in the estimate
  - Variance of estimate

1. How much a classification model benefits from adding more training data?
2. Does the model suffer from a variance error or a bias error?
Methods of Estimation

- **Holdout**
  - Reserve 2/3 for training and 1/3 for testing
- **Random subsampling**
  - Repeated holdout
- **Cross validation**
  - Partition data into k disjoint subsets
  - k-fold: train on k-1 partitions, test on the remaining one
  - Leave-one-out: k=n
- **Stratified sampling**
  - oversampling vs undersampling
- **Bootstrap**
  - Sampling with replacement
Small & Unbalanced Data

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

- Usually, **one third for testing**, the rest for training

- For small or “unbalanced” datasets, **samples might not be representative**
  - For instance, few or none instances of some classes

- Stratified sample
  - **Balancing the data**
  - Make sure that each class is represented with approximately equal proportions in both subsets
Repeated holdout method

- Holdout estimate can be made more reliable by repeating the process with different subsamples
  - In each iteration, a certain proportion is randomly selected for training (possibly with stratification)
  - The error rates on the different iterations are averaged to yield an overall error rate

- This is called the repeated holdout method
- Still not optimum: the different test sets overlap
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**

- Often the subsets are stratified before cross-validation is performed

- The **error estimates** are **averaged** to yield an overall error estimate

- **Even better**: repeated stratified cross-validation
  E.g. ten-fold cross-validation is repeated ten times and results are averaged (reduces the variance)
Model Evaluation

- Metrics for Performance Evaluation
  - How to evaluate the performance of a model?

- Methods for Performance Evaluation
  - How to obtain reliable estimates?

- Methods for Model Comparison
  - How to compare the relative performance among competing models?
ROC (Receiver Operating Characteristic)

- Developed in 1950s for signal detection theory to analyze noisy signals
  - Characterize the trade-off between positive hits and false alarms

- ROC curve plots TP (on the y-axis) against FP (on the x-axis)

- Performance of each classifier represented as a point on the ROC curve
  - changing the threshold of algorithm, sample distribution or cost matrix changes the location of the point
ROC Curve

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

At threshold $t$:
$TP=0.5$, $FN=0.5$, $FP=0.12$, $FN=0.88$
ROC Curve

(TP,FP):
- (0,0): declare everything to be negative class
- (1,1): declare everything to be positive class
- (1,0): ideal

Diagonal line:
- Random guessing
- Below diagonal line:
  - prediction is opposite of true class
Using ROC for Model Comparison

- No model consistently outperform the other
  - $M_1$ is better for small FPR
  - $M_2$ is better for large FPR

- Area Under the ROC curve
  - Ideal:
    - Area = 1
  - Random guess:
    - Area = 0.5
How to Construct an ROC curve

• Use classifier that produces posterior probability for each test instance $P(+|A)$

• Sort the instances according to $P(+|A)$ in decreasing order

• Apply threshold at each unique value of $P(+|A)$

• Count the number of TP, FP, TN, FN at each threshold

• TP rate, TPR = TP/(TP+FN)

• FP rate, FPR = FP/(FP + TN)

| Instance | $P(+|A)$ | True Class |
|----------|----------|------------|
| 1        | 0.95     | +          |
| 2        | 0.93     | +          |
| 3        | 0.87     | -          |
| 4        | 0.85     | -          |
| 5        | 0.85     | -          |
| 6        | 0.85     | +          |
| 7        | 0.76     | -          |
| 8        | 0.53     | +          |
| 9        | 0.43     | -          |
| 10       | 0.25     | +          |
How to construct an ROC curve

<table>
<thead>
<tr>
<th>Threshold &gt;=</th>
<th>0.25</th>
<th>0.43</th>
<th>0.53</th>
<th>0.76</th>
<th>0.85</th>
<th>0.85</th>
<th>0.85</th>
<th>0.87</th>
<th>0.93</th>
<th>0.95</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TP</strong></td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>FP</strong></td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>TN</strong></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td><strong>FN</strong></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td><strong>TPR</strong></td>
<td>1</td>
<td>0.8</td>
<td>0.8</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.4</td>
<td>0.4</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td><strong>FPR</strong></td>
<td>1</td>
<td>1</td>
<td>0.8</td>
<td>0.8</td>
<td>0.6</td>
<td>0.4</td>
<td>0.2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

ROC Curve:
Test of Significance

- Given two models:
  - Model M1: accuracy = 85%, tested on 30 instances
  - Model M2: accuracy = 75%, tested on 5000 instances

- Can we say M1 is better than M2?
  - How much confidence can we place on accuracy of M1 and M2?
  - Can the difference in performance measure be explained as a result of random fluctuations in the test set?
Confidence Interval for Accuracy

- Prediction can be regarded as a Bernoulli trial (binomial random experiment)
  - A Bernoulli trial has 2 possible outcomes
  - Possible outcomes for prediction: correct or wrong
  - Probability of success is constant
  - Collection of Bernoulli trials has a Binomial distribution:
    - \( x \sim \text{Bin}(N, p) \) \( x \): # of correct predictions, \( N \) trials, \( p \) constant prob.
    - e.g: Toss a fair coin 50 times, how many heads would turn up?
      Expected number of heads = \( N \times p = 50 \times 0.5 = 25 \)

Given \( x \) (# of correct predictions) or equivalently, \( \text{acc} = x/N \), and \( N \) (# of test instances)

Can we predict \( p \) (true accuracy of model)?
Confidence Interval for Accuracy

- For large test sets \((N > 30)\),
  - acc has a normal distribution with mean \(p\) and variance \(p(1-p)/N\)

\[
P\left( Z_{\alpha/2} < \frac{\text{acc} - p}{\sqrt{p(1-p)/N}} < Z_{1-\alpha/2} \right) = 1 - \alpha
\]

- Confidence Interval for \(p\):

\[
p = \frac{2 \times N \times \text{acc} + Z_{\alpha/2}^2 \pm \sqrt{Z_{\alpha/2}^2 + 4 \times N \times \text{acc} - 4 \times N \times \text{acc}^2}}{2(N + Z_{\alpha/2}^2)}
\]
Confidence Interval for Accuracy

- Consider a model that produces an accuracy of 80% when evaluated on 100 test instances:
  - $N=100$, $\text{acc} = 0.8$
  - Let $1-\alpha = 0.95$ (95% confidence)
  - Which is the confidence interval?
  - From probability table, $Z_{\alpha/2}=1.96$

<table>
<thead>
<tr>
<th>N</th>
<th>50</th>
<th>100</th>
<th>500</th>
<th>1000</th>
<th>5000</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(lower)</td>
<td>0.670</td>
<td>0.711</td>
<td>0.763</td>
<td>0.774</td>
<td>0.789</td>
</tr>
<tr>
<td>p(upper)</td>
<td>0.888</td>
<td>0.866</td>
<td>0.833</td>
<td>0.824</td>
<td>0.811</td>
</tr>
</tbody>
</table>

$1-\alpha$ | $Z$
---|---
0.99 | 2.58
0.98 | 2.33
0.95 | 1.96
0.90 | 1.65
Comparing Performance of 2 Models

- Given two models, say M1 and M2, which is better?
  - M1 is tested on D1 (size=n1), found error rate = $e_1$
  - M2 is tested on D2 (size=n2), found error rate = $e_2$
  - Assume D1 and D2 are independent
  - If n1 and n2 are sufficiently large, then

\[
e_1 \sim N(\mu_1, \sigma_1)
\]

\[
e_2 \sim N(\mu_2, \sigma_2)
\]

- Approximate variance of error rates: $\hat{\sigma}_i = \frac{e_i(1-e_i)}{n_i}$
Comparing Performance of 2 Models

- To test if performance difference is statistically significant:  \( d = e_1 - e_2 \)
  - \( d \sim N(d_t, \sigma_t) \) where \( d_t \) is the true difference
  - Since D1 and D2 are independent, their variance adds up:
    \[
    \sigma_t^2 = \sigma_1^2 + \sigma_2^2 \approx \hat{\sigma}_1^2 + \hat{\sigma}_2^2
    \]
    \[
    = \frac{e_1(1-e_1)}{n_1} + \frac{e_2(1-e_2)}{n_2}
    \]
  - It can be shown at \((1-\alpha)\) confidence level,
    \[
    d_t = d \pm Z_{\alpha/2} \hat{\sigma}_t
    \]
An Illustrative Example

- Given: M1: n1 = 30, e1 = 0.15
  M2: n2 = 5000, e2 = 0.25

- $d = |e_2 - e_1| = 0.1$ (2-sided test to check: $d_t = 0$ or $d_t <> 0$)

$$\hat{d}^2 = \frac{0.15(1-0.15)}{30} + \frac{0.25(1-0.25)}{5000} = 0.0043$$

- At 95% confidence level, $Z_{\alpha/2} = 1.96$

$$d_t = 0.100 \pm 1.96 \times \sqrt{0.0043} = 0.100 \pm 0.128$$

=> Interval contains 0 => difference may not be statistically significant
Comparing Performance of 2 Algorithms

- Each learning algorithm may produce k models:
  - L1 may produce M11, M12, …, M1k
  - L2 may produce M21, M22, …, M2k

- If models are generated on the same test sets D1, D2, …, Dk (e.g., via cross-validation)
  - For each set: compute \( d_j = e_{1j} - e_{2j} \)
  - \( d_j \) has mean \( d_t \) and variance \( \sigma_t^2 \)
  - Estimate:
    \[
    \hat{\sigma}_t^2 = \frac{\sum_{j=1}^{k} (d_j - \bar{d})^2}{k(k-1)}
    \]
    \[
    d_t = \bar{d} \pm t_{1-\alpha, k-1} \hat{\sigma}_t
    \]