# CS256 – Midterm Exam Study Guide

**By: Zayd Hammoudeh**

## Chapter #04 – Classification: Basic Concepts, Decision Trees, and Model Evaluation

| **Classification** | **Training Set** | **Model** | **Test Set** | **Example Classification Techniques** |
|---|---|---|---|---|
| Task of assigning objects to one of several predefined categories. | A collection of records. Each **record** contains a set of attributes one of which is the **class**. | A function from the value of record attributes to the class attribute. | A collection of records used to determine the accuracy of the classification model. | 1. Neural Networks<br>2. Decision Tree<br>3. Rule Based Classifier<br>4. Memory Based Reasoning<br>5. Support Vector Machines<br>6. Naïve Bayes and Bayesian Belief Networks |

**Induction**
Using a training set to generate a model.

**Deduction**
Process of applying a model to a training set.

**Decision Tree Induction**
- **Greedy Strategy**
- **Key Decision #1:** Attribute to expand next
- **Key Decision #2:** When to stop expanding

**Hunt's Decision Tree Induction Algorithm:**
- Let $D_t$ be the set of training records that reach a node $t$.

1. If $D_t$ contains records that **all belong to the same class $y_t$**, then $t$ is a leaf node with class value $y_t$.
2. If $D_t$ is an **empty set**, then $t$ is a leaf node with default value $y_d$.
3. If $D_t$ contains **records that belong to more than one class and there are no attributes left**, then $t$ is a leaf node with default value is a leaf node with default value $y_d$.
4. If $D_t$ contains **records that belong to more than one class**, then use an attribute test to split the data into smaller subsets. Recursively apply the same procedure above.

**Attribute Types**
- **Binary** – Attribute with exactly two possible values.
- **Nominal** – Two or more class values with no intrinsic Order
- **Ordinal** – Two or more class values that can be ordered or ranked
- **Continuous** – Quantitative attribute that can be measured along a continuum.

**Splitting Nominal and Ordinal Attributes**
- **Binary** – Divides attribute values into two subsets. **This requires the additional step of finding optimal partitioning.**
- **Multi-way** – Use as many partitions as distinct values.

**Splitting Based on Continuous Attributes**
- **Discretization** – Form an ordinal categorical attribute.
  - **Static** – Discretize once at the beginning
  - **Dynamic** – Ranges can be found by equal interval bucketing, equal frequency bucketing, or clustering.
- **Binary Decision** (A < v or A >v) – Consider all possible splits and find the best cut.
  - **Binary Decision Procedure:** Go between each training set record value and calculate the GINI index if the splitting point was at that value. **Select the splitting point with the lowest $GINI_{SPLIT}$ value.**
    - **Computationally inefficient** $O(n)$ – where $n$ is the number of records.

**Homogeneity/Low Impurity** – Extent to which nodes in the decision tree have the same class value/distribution.

**Nodes with high levels of homogeneity (i.e. low levels of impurity) are preferred.**

### Impurity Measures

**For all of these metrics, a lower score is generally preferable.**

#### GINI Index

$$GINI(t) = 1 - \sum_{j=1}^{n_c} \big(p(j|t)\big)^2$$

- $t$ – Node in the decision tree
- $j$ – Class value
- $n_c$ – Number of class values
- $p(j|t)$ – Probability (i.e. relative frequency) of class value $j$ in node $t$

**Minimum Value:** 0 when:

$$\exists j(p(j|t) = 1)$$

**Maximum Value:** $1 - \frac{1}{n_c}$ when:

$$\forall j\left(p(j|t) = \frac{1}{n_c}\right)$$

#### $GINI_{SPLIT}$

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

- $i$ – Child node
- $n$ – Number of records in parent node. Note:

$$n = \sum_{i=1}^{k} n_i$$

- $n_i$ – Number of child nodes (i.e. attribute partitions)
- $GINI(i)$ – GINI index value of node $i$.

**Minimum Value:** 0 when:

$$\forall i(GINI(i) = 0)$$

**Maximum Value:** $1 - \frac{1}{n_c}$ when:

$$\forall i\left(GINI(i) = 1 - \frac{1}{n_c}\right)$$

#### Entropy

$$Entropy(t) = -\sum_{j=1}^{n_c} p(j|t) \cdot log_2\big(p(j|t)\big)$$

- $t$ – Node in the decision tree
- $j$ – Class value
- $n_c$ – Number of class values
- $p(j|t)$ – Probability (i.e. relative frequency) of class value $j$ in node $t$

**Minimum Value:** 0 when:

$$\exists j(p(j|t) = 1)$$

**Maximum Value:** $\log_2(n_c)$ when:

$$\forall j\left(p(j|t) = \frac{1}{n_c}\right)$$

## Information Gain

$$GAIN_{SPLIT}(t) = Entropy(p) - \left(\sum_{i=1}^{k} \frac{n_i}{n} \cdot Entropy(i)\right)$$

- $p$ – Parent node in the decision tree
- $i$ – Child node in the decision tree
- $k$ – Number of child nodes
- $n_i$ – Number of records in child node $i$
- $n$ – Number of records in parent node $p$

$$n = \sum_{i=1}^{k} n_i$$

**Key Note:** A higher $GAIN_{SPLIT}$ is preferable unlike with the other metrics where a lower value was better.

**Disadvantage of Information Gain:** Tends to prefer splits that result in a large number of partitions, each being small but pure (i.e. overfitting)

### Normalizing for Split Size

$$GainRATIO_{Split} = \frac{Gain_{SPLIT}(t)}{SplitINFO}$$

$$SplitINFO = -\sum_{i=1}^{k} \frac{n_i}{n} \cdot log_2\left(\frac{n_i}{n}\right)$$

**$Split_{INFO}$ penalizes a large split by reducing the gain.**

## Classification Error

$$Error(t) = 1 - max_j(p(j|t))$$

- $t$ – Node in the decision tree
- $j$ – Class value
- $p(j|t)$ – Probability (i.e. relative frequency) of class value $j$ in node $t$

**Minimum Value:** 0 when:
$\exists j(p(j|t) = 1)$

**Maximum Value:** $1 - \frac{1}{n_c}$ when:
$\forall j\left(p(j|t) = \frac{1}{n_c}\right)$

## Stopping Criteria for Decision Tree Induction

### Three Stopping Criteria for Decision Tree Induction

- When all records in a node have the same class value
- When all records in a node have similar attribute values.
- Early Termination

**Underfitting** – When a model is too simple, both training and test errors are large.

**Overfitting** – When a model becomes too complex (e.g. too large a tree), the test error begins to increase even though the training error decreases.

- **Result:** Training error is **NOT** representative for generalization error.

### Causes of Overfitting

- Noise
- Insufficient training records (i.e. lack of representative samples)

### Resubstitution Error

Error on the **training** set.

Single Leaf Node Error: $e(t)$
Total Resubstitution Error: $e(T)$

$$e(T) = \sum e(t)$$

### Generalization Error

Error on the **testing** data.

Single Leaf Node Error: $e'(t)$
Total Generalization Error: $e'(T)$

$$e'(T) = \sum e'(t)$$

## Generalization Error Estimation

### Optimistic Estimation

Training error is equal to the testing error.

$$\sum e(t) = \sum e'(t)$$

### Pessimistic Estimation

Assign a penalty term to ea.

$$e'(t) = e(t) + 0.5$$

**Total Pessimistic Error**

$$e'^{(T)} = \sum(e(t)) + N \cdot 0.5$$

$N$ – Number of leaf nodes.

### Reduced Error Pruning

Use a validation set to estimate the generalization error.

### Occam's Razor

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

**This is because more complex model has a greater chance of fitting accidentally by errors in the data.**

### Pre-pruning (Early Stopping Rule)

- **Stop the induction algorithm before it becomes a full tree.**
- **Typical Stopping Rules:**
  - All remaining records have the same class value
  - All attribute values are the same.
- **More restrictive conditions:**
  - Number of instances is below a user-specified threshold.
  - Expanding the current node does not improve impurity measures (e.g. GINI Index, Information Gain)
  - Class distribution of instances are independent of available features.

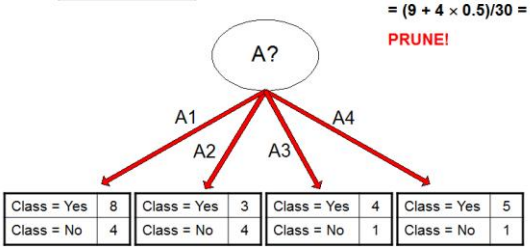
### Post-pruning (Early Stopping Rule)

- **Grow the decision tree to its entirety**.
- Trim nodes in the tree in a **bottom-up fashion**.
- Only **trim nodes if by trimming the estimate of the generalization error improves**.
- New leaf node's **class label is determined from the majority class of instances in the merged node**.

## Example of Post-Pruning

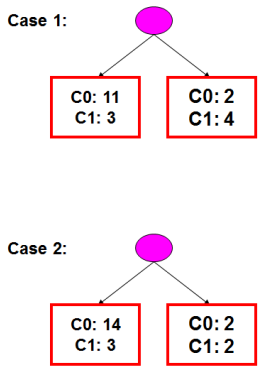
| | |
|---|---|
| Class = Yes | 20 |
| Class = No | 10 |
| Error = 10/30 | |

**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!**




## Examples of Post-pruning

- Optimistic error?
  - **Don't prune for both cases**
- Pessimistic error?
  - **Don't prune case 1, prune case 2**
- Reduced error pruning?
  - **Depends on validation set**



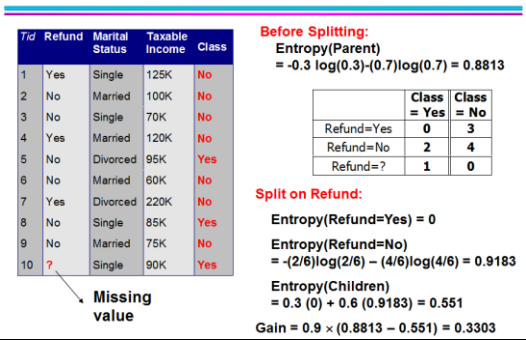
# Handling Missing Attribute Values

### Issues Associated with Missing Attribute Values

- Affects how impurity measures are computed
- Affects how to distribute instances with missing value to child nodes.
- Affects how to test instance with missing value is classified.

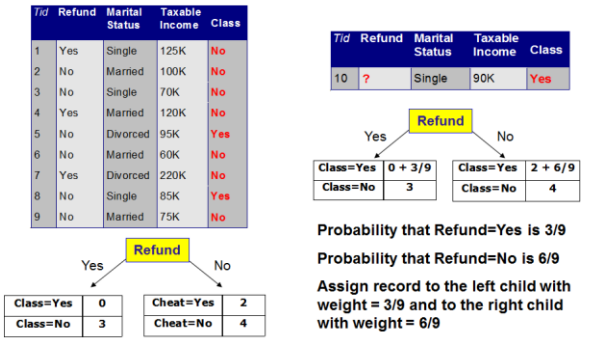
### Computing Impurity Measure

- Calculate entropies (i.e. information gain) with element with missing value EXCLUDED.
- Multiply by scalar of elements included over total number of elements (in below example 9 elements included over 10 total elements hence 0.9):



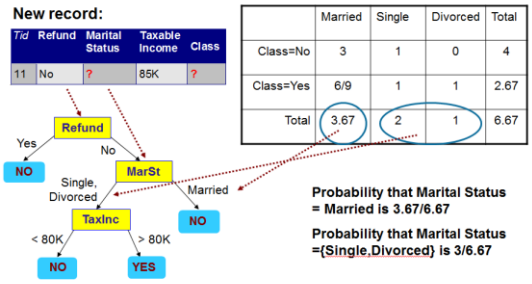

### Distribute Instances

- Split the missing record between the two child nodes
- Percentage of child node that goes to each child is portion to the relative frequency of that attribute value.




### Classifying New/Unseen Records with Missing Data

- Pick the most likely of child nodes and use continue down that portion of the tree.




**Data Fragmentation** – At each level of the tree, the number of instances gets smaller. At leaf nodes, the number of instances could be too small to be statistically significant.

**Tree Induction: NP Hard**

**Alternate Strategies**

- Bottom Up Tree Generation
- Bidirectional Tree Generation
  - Inside-out Bidirectional
  - Outside-in Bidirectional

**Decision Boundary** – Borderline between two neighboring regions of different classes. In non-oblique decision trees, this is parallel to access since it involves a single attribute at a time.

**Oblique Decision Tree** – Test condition in a node may involve multiple attributes.

- Advantage – Most expressive decision tree
- Disadvantage – Finding optimal test condition is computationally expensive.

**Expressiveness** – Decision trees do not generalize well to certain types of functions including a parity function which would require a complete tree.

**Tree Replication** – In a decision tree, a subtree may appear in multiple branches. This leads to unnecessary memory usage.

# Performance Evaluation

- Focus on the predictive capability of a model.

**Confusion Matrix**

| | | Predicted Class | |
|---|---|---|---|
| | | Class = Yes | Class=No |
| Actual Class | Class = Yes | a | b |
| | Class=No | c | d |

a – True Positive (TP)
b – False Negative (FN)
c – False Positive (FP)
d – True Negative (TN)

**Accuracy**

$$Accuracy = \frac{A + D}{A + B + C + D}$$

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

- Accuracy only tells part of the story.
  - Example: Two Class Problem
    - Number of Class 0 Examples: 9990
    - Number of Class 1 Examples: 10
    - If the model predicts everything as class 0, its accuracy is 99.9% but it cannot detect any class 1.

**Cost Matrix**

| | | Predicted Class | |
|---|---|---|---|
| | | Class = Yes | Class=No |
| Actual Class | Class = Yes | C(Y\|Y) | C(N\|Y) |
| | Class=No | C(Y\|N) | C(N\|N) |

- $C(j \mid k)$ – Cost of predicting class "j" given the actual class is "k".

$$TotalCost = a \cdot C(Y|Y) + b \cdot C(N|Y) + c \cdot C(Y|N) + d \cdot C(N|N)$$

- Cost matrix can be a better performance evaluation as it accounts for different costs of depending on the type of error.

$$Precision\ (p) = \frac{a}{a + c}$$

- Precision – Accuracy of positive predictions. Biased towards C(Y|Y) & C(Y|N).
  - a – True positive.
  - c – False positive.

$$Recall\ (r) = \frac{a}{a + b}$$

- Precision – Accuracy of records with positive class value. Biased towards C(Y|Y) & C(N|Y).
  - a – True positive.
  - b – False negative.

$$F\_Measure\ (F) = \frac{2 \cdot r \cdot p}{r + p} = \frac{a + c}{2 \cdot a + b + c}$$

- F-Measure – Biased two all except C(N|N) (i.e. true negative)
  - r – Recall
  - p – Precision
  - c – False Positive

$$WA = \frac{w_1 \cdot a + w_4 \cdot d}{w_1 \cdot a + w_2 \cdot b + w_3 \cdot c + w_4 \cdot d}$$

$n$ – Number of instances covered by rule
$n_c$ – Number of positive instances covered by rule.

**Proportionality of Cost and Accuracy**

- Cost and accuracy are proportional if:

$$C(Y|Y) = C(N|N) \text{ and } C(Y|N) = C(N|Y)$$

**Sample Size and Model Performance**

- **Learning Curve** – Shows how model accuracy changes (and varies) with sample size.
- **Effects of Small Sample Size:**
  - **Bias in the estimate**
  - **Variance in the estimate.**

## Methods for Model Comparison

**Holdout** – Reserve 2/3 of labelled examples for training and 1/3 for testing.

**Disadvantages**

- **Uses on a subset of the labelled examples** when training the model.
- Model **dependent on the composition of the training and test sets**.
- **Training and test sets are not independent** since come from same original set. If one class value is over- or under-represented in either set, it will skew the results.

**Random Subsampling** – Repeats the whole out method multiple times with replacement.

**Disadvantages:**

- Still **uses only a subset of the labelled examples** to build the model.
- **No control of how many times each record appears in the training and test sets**. If a particular record is always in the training set, it may skew the model.

**Accuracy of *k* Random Subsamplings**

$$acc_{sub} = \frac{1}{k} \cdot \sum_{i=1}^{k} acc_i$$

- $k$ – Number of iterations
- $acc_i$ – Accuracy of the $i^{th}$ iteration.

**Cross Validation** – Partition the labelled dataset into *k* disjoint subsets.

- **k-Fold** – Train on k-1 partitions and test on the remaining one.
- **Leave-One-Out** – The number of partitions equals the number of training samples.

**Accuracy of *k*-Fold Cross Validation**

$$acc_{sub} = \frac{1}{k} \cdot \sum_{i=1}^{k} acc_i$$

- $k$ – Number of iterations
- $acc_i$ – Accuracy of the $i^{th}$ iteration.

**Disadvantages:**

- **Computationally expensive** as process is repeated *k* times.
- Depending on size of partition (e.g. 1 for Leave-One-Out), **accuracy from iteration to iteration** can vary significantly.

**Bootstrap** –

**Minimum Description Length**

$$WeightedDistance = \frac{1}{d^2}$$

# Chapter #05 – Additional Classification Techniques

## Rule-Based Classifiers

**Classifies records using a collection of "if…then…" rules.**
**Form of Rule:**

$$(Condition) \rightarrow y$$

- **Condition** (**Antecedent**, **LHS**) – Conjunction of attributes.
- $y$ (**Consequent**, **RHS**) – Class value.

**Cover** – A rule $r$ covers an instance $x$ if the attributes of $x$ satisfy the condition (LHS) of the rule.

**Coverage of a Rule** – Fraction of records that satisfy the antecedent of a rule.

**Accuracy of a Rule** – For records covered by a rule, it is the fraction of records that have the matching class value.

**Mutually Exclusive Rule Set** – Rules in the set are independent of each other such that **each record is covered by at most one rule**.

**Exhaustive Rule Set** – A set of rules that covers every possible combination of attribute values. Hence, **each record is covered by at least one rule**.

**Decision Tree** – Can be used to formed a mutually exclusive, exhaustive rule set.

Rules in a decision tree can be simplified.

**Effects of rule simplification:**
- **Problem #1:** Rules become non-mutually exclusive.
- **Solution:**
  - **Ordered Rule Set** – Rules ordered from highest to lowest priority. Records classified according to highest priority rule they satisfy.
  - **Unordered Rule Set** – Voting scheme
- **Problem #2:** Rules become non-exhaustive.
- **Solution:** Use a default class.

### Rule Ordering Schemes

**Rules Based Ordering** – Individual rules are ranked based off their quality.
- **Advantage:** Ensures each record is classified by the "best rule" covering it.
- **Disadvantage:** Interpreting lower priority rules becomes more difficult as they are negations of higher priority rules.

**Class-Based Ordering** – Rules that belong to the same class appear together.
- **Advantage:** Simplifies rule ordering.
- **Disadvantage:** May allow a lower quality rule to have higher priority than a higher quality one.

**Direct Method for Rule Building** – Extract rules directly from the data.
- **Examples:** RIPPER, CN2, Holte's 1R

**Indirect Method for Rule Building** – Extract rules from other classification models (e.g. decision tree, neural network, etc.)
- **Examples:** C4.5rules

### Sequential Covering Algorithm

1. **Start with an empty rule set.**
2. **Grow a rule using the "Learn-One-Rule" function.**
3. **Remove training records covered by the rule.**
4. **Repeat steps #2 and #3 until stopping criterion is met.**

**Aspects of Sequential Covering**
1. **Rule Growing**
2. **Instance Elimination**
3. **Rule Evaluation**
4. **Stopping Criterion**
5. **Rule Pruning**

**Rule Growing Strategies**
1. **General to Specific**
   a. Example: Ripper
2. **Specific to General**

### CN2 Algorithm

1. Start from an empty rule
2. Add conjuncts that minimize the entropy measure.
3. Determine the rule consequent by taking majority class of covered instances.

### Instance Elimination

- **Reason for Eliminating Instances** – Otherwise next rule is identical to previous rule.
- **Reason for Removing Positive Instances** – To ensure future rules are different.
- **Reason for Removing Negative Instances** – Prevent underestimating accuracy of the rule.

### Stopping Criterion

Compute the information gain with the rule. If the gain is insignificant, discard the rule.

### Rule Pruning

- **Similar to post-pruning of decision trees.**
- Uses reduced error pruning.
  - **Remove one of the conjuncts of the rule.**
  - **Compare error rate on validation set before and after pruning.**
  - **If error improves, remove the conjunct.**

**Rule Simplification** – Used to reduce the likelihood of overfitting.

## Rule Evaluation Metrics

$$Accuracy = \frac{n_c}{n}$$

$n$ – Number of instances covered by rule
$n_c$ – Number of positive instances covered by rule.

$$Laplace = \frac{n_c + 1}{n + k}$$

$n$ – Number of instances covered by rule
$n_c$ – Number of positive instances covered by rule.
$k$ – Number of classes
**Used to ensure greater coverage for a rule.**

$$m\_estimate = \frac{n_c + p \cdot k}{n + k}$$

$n$ – Number of instances covered by rule
$n_c$ – Number of positive instances covered by rule.
$k$ – Number of classes
$p$ – Prior probability of positive class.

### FOIL Information Gain

$$Gain(R0, R1) = t \cdot \left( \log_2 \left( \frac{p_1}{p_1 + n_1} \right) + \log_2 \left( \frac{p_0}{p_0 + n_0} \right) \right)$$

$R0$ – Initial Rule
$R1$ – Modified version of $R0$ with added conjunct
$t$ – Number of positive instances covered by both $R0$ and $R1$
$p_0$ – Positive instances covered by $R0$
$n_0$ – Negative instances covered by $R0$
$p_1$ – Positive instances covered by $R1$
$n_1$ – Negative instances covered by $R1$

### RIPPER Algorithm

1. For two classes, define one class as positive class and other as negative class.
   a. In two class problem, **negative class is the default class**.
2. **In multi-class problem, create list of classes ordered by increasing prevalence**.
   a. Select smallest as first as positive class and rest are negative class.
   b. **Learn rules for the smallest class first.**
   c. **Repeat with next smallest class**.

### RIPPER Algorithm – Growing a Rule

1. Start from an empty rule set.
2. Add conjuncts as long as they improve **FOIL Information Gain** (i.e. **General-to-Specific**).
3. Stop adding conjuncts when the rule starts covering negative examples.
4. Begin pruning the rule immediately (i.e. before generating new rules) using Reduced Error Pruning.
5. Delete conjuncts to maximize $v$ as defined by:

$$v = \frac{p - n}{p + n}$$

$p$ – Number of positive instances covered by the rule.
$n$ – Number of negative instances covered by the rule.

**RIPPER Algorithm – Building the Rule Set**

1. **Use Sequential Covering**
   a. Find the rule that best covers the current set of positive examples.
   b. Eliminate both positive and negative examples covered by the rule.
   c. **Uses ordered rule set with class based ordering.**

2. Each time a rule is added to the rule set, compute the new description length. **Example Stopping Conditions:**
   a. Stopping growing the rule set if the new **rule increases the description length of the rule set by more than $d$** (e.g. 64) **bits**.
   b. Stop if the error rate of the rule **on the validation set** is more than 50%.

**C4.5rules – Indirect Method**

1. Start from an **unpruned** decision tree.
2. For each rule $r: A \rightarrow y$,
   a. Consider an alternative rule $r': A' \rightarrow y$ where $A'$ is obtained by **removing one of the conjuncts** of $A$
   b. **Keep the rule with the lowest pessimistic error rate** (assuming it is less than the original).
   c. Repeat until it is no longer possible to improve the generalization error.
3. Use class-based ordering for the rule set (i.e. group by the rule consequent).
4. Compute the description length of each class and order the rules by increasing description length.

$$DescriptionLength = L(error) + g \cdot L(model)$$

- $L(error)$ – Number of bits required to encode misclassified examples.
- $L(model)$ – Number of bits required to encode the model.
- $g$ – Tuning parameter whose default is 0.5 and takes into account the presence of redundant attributes in the rule set.

## Nearest Neighbor Classifiers

**Instance-Based Classifier** – Stores all training records and uses the training records directly to predict the class label of unseen records.

**Rote-Learner** – Memorizes the entire training set and performs classification only if attributes of a record match one the training examples exactly.

**Nearest Neighbor** – Uses $k$ "closest" training records (i.e. nearest neighbors) for performing classification.

$$WeightedDistance = \frac{1}{d^2}$$

**ROC Curve**

- Used to illustrate the performance of a binary classifier.
- Two Dimensional
  - **X-Axis** – False Positive Rate
  - **Y-Axis** – True Positive Rate

# Miscellaneous

### Decision Tree Algorithm

**Advantages**

- **Inexpensive to construct**
- **Extremely fast at classifying unknown records.**
- **Easy to interpret for small sized trees.**
- **Accuracy is comparable to other classification techniques for many simple datasets.** (Since everything comes right from the data)

**Disadvantages**

- **May not generalize well for certain types of functions** (e.g. Parity function requires a complete tree)
- **May be insufficient for modelling continuous variables** that do not allow oblique nodes.

### Rule Based Classifiers

**Advantages**

- **As highly expressive as decision trees**
  - A decision tree can be expressed via rules based classifier).
  - Allows for more complex models than a decision tree by allowing multiple rules to trigger on a single rule.
- **Easy to interpret.**
- **Easy to generate.**
- **Can classify new records quickly.**
- **Performance comparable to decision trees**
- **Well suited for handling data sets with imbalanced class distributions.**