Chapter 6
Classification and Prediction
(1)

Outline

- Classification and Prediction
- Decision Tree
- Naïve Bayes Classifier
- Support Vector Machines (SVM)
- K-nearest Neighbors
- Other Classification methods
- Accuracy and Error Measures
- Ensemble Methods
- Applications
- Summary

Supervised vs. Unsupervised Learning

- Supervised learning (classification)
  - Supervision: The training data (observations, measurements, etc.) are accompanied by labels indicating the class of the observations
  - New data is classified based on the training set
- Unsupervised learning (clustering)
  - The class labels of training data is unknown
  - Given a set of measurements, observations, etc. with the aim of establishing the existence of classes or clusters in the data

Classification vs. Prediction

- Classification
  - classifies data (constructs a model) based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data
  - predicts categorical class labels
- Prediction
  - models continuous-valued functions, i.e., predicts unknown or missing values
    - Regression analysis: a statistical method
      - Predict how much a given customer will spend during a sale.

Application of Classification

- Typical applications
  - Credit/loan approval:
  - Medical diagnosis: if a tumor is cancerous or benign
  - Fraud detection: if a transaction is fraudulent
  - Web page categorization: which category it is

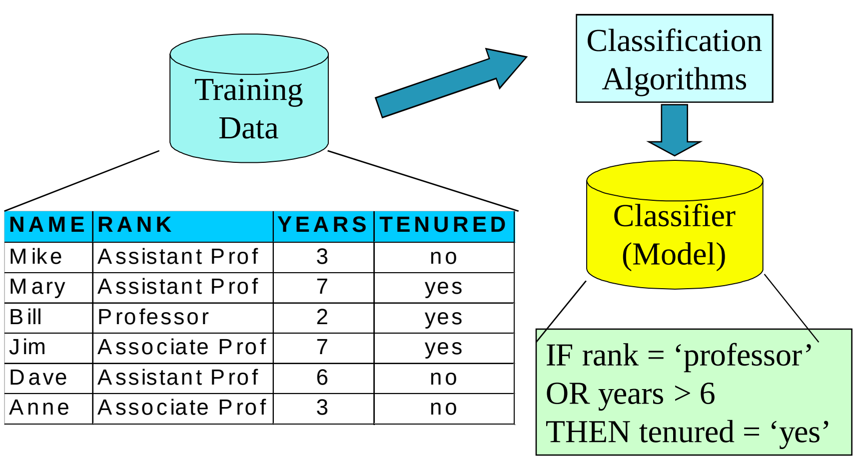
Classification—A Two-Step Process

- Model construction: describing a set of pre-determined classes
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the class label attribute
  - The set of tuples used for model construction is training set
  - The model is represented as classification rules, decision trees, or mathematical formulae

Classification—A Two-Step Process

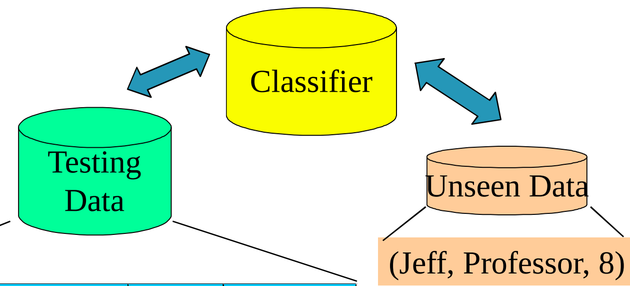
- Model usage: for classifying future or unknown objects
  - Estimate accuracy of the model
    - The known label of test sample is compared with the classified result from the model
    - Accuracy rate is the percentage of test set samples that are correctly classified by the model
    - Test set is independent of training set, otherwise over-fitting will occur
  - If the accuracy is acceptable, use the model to classify data tuples whose class labels are not known.

Process (1): Model Construction



NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

Process (2): Using the Model in Prediction



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes

Issues: Data Preparation

- Data cleaning
  - Preprocess data in order to reduce noise and handle missing values
- Relevance analysis (feature selection)
  - Remove the irrelevant or redundant attributes
- Data transformation
  - Generalize and/or normalize data

Issues: Evaluating Classification Methods

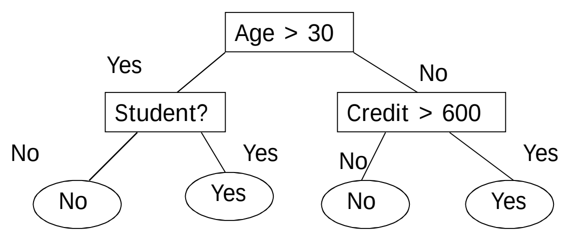
- Accuracy
  - classifier accuracy: predicting class label
- Speed
  - time to construct the model (training time)
  - time to use the model (classification/prediction time)
- Robustness: handling noise and missing values
- Scalability: efficiency in disk-resident databases
- Interpretability
  - understanding and insight provided by the model
- Other measures, e.g., goodness of rules, such as decision tree size or compactness of classification rules

Outline

- Classification and Prediction
- Decision Tree
- Naïve Bayes Classifier
- Support Vector Machines (SVM)
- K-nearest Neighbors
- Other Classification methods
- Accuracy and Error Measures
- Feature Selection Methods
- Ensemble Methods
- Applications
- Summary

Decision Tree

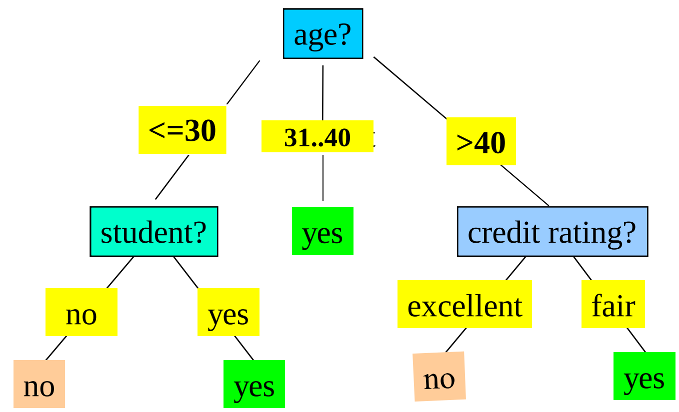
- A flowchart-like tree structure.
  - Internal (non-leaf) node denotes a test on an attribute (feature)
  - Branch represents an outcome of the test
  - Leaf node holds a class label.



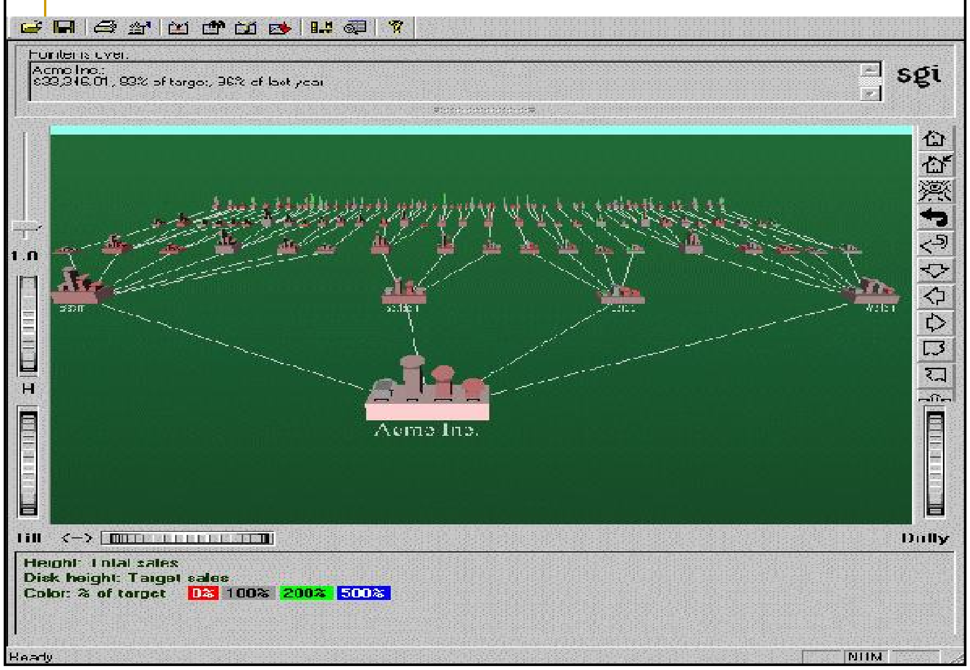
Decision Tree Induction: Training Dataset

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Output: A Decision Tree for "buys_computer"



Visualization of a Decision Tree in SGI/MineSet 3.0



Algorithm for Decision Tree Induction

- A greedy algorithm: top-down recursive divide-and-conquer manner.
  - At start, all the training examples are at the topmost node.
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a ***heuristic*** or statistical measure (e.g., information gain)

Golf Data

Outlook	Temp	Humidity	Windy	Class
Sunny	Hot	High	No	Yes
Sunny	Hot	High	Yes	Yes
O'cast	Hot	High	No	No
Rain	Mild	Normal	No	No
Rain	Cool	Normal	No	No
Rain	Cool	Normal	Yes	Yes
O'cast	Cool	Normal	Yes	No
Sunny	Mild	High	No	Yes
Sunny	Cool	Normal	No	No
Rain	Mild	Normal	No	No
Sunny	Mild	Normal	Yes	No
O'cast	Mild	High	Yes	No
O'cast	Hot	Normal	No	No
Rain	Mild	High	Yes	Yes

Steps of Decision Tree Induction Algorithm

- Starts with three parameters:
  - **D**, data partition, the set of training tuples and their class labels.
    - Example: Golf_data: 14 tuples (5 yes, 9 no)
  - **Attribute_list**, the set of candidate attributes.
    - Example: {outlook, temp, humidity, windy}
  - **Attribute_selection_method**, the procedure to determine the ***splitting critierion*** that best partitions the data tuples into individual classes.

Steps of Decision Tree Induction Algorithm

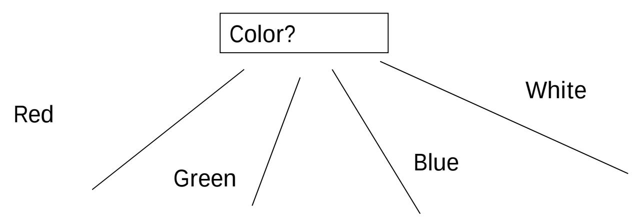
- Step 1 (Top-down)
  - The tree starts with a single node ***N***, representing the training tuples in **D**.
- Step 2
  - **IF** the tuples in **D** are all of the same class, then node ***N*** is a leaf and is labeled with the class label.
  - **ELSE** **Attribute_selection_method** determine the ***splitting criterion*** to perform Partitioning of **D** into **Dj**s. (Divide & Conquer)
- Step 3 (Recursive)
  - Form a decision tree for the tuples at each partition **Dj**.

Splitting Criterion

- Determines the *best* way to partition the tuples in D into individual classes – pureness of the partitions Dj at each branch.
  - Which attribute to test.
  - Which branches to grow from node N with respect to the outcomes of the test.
  - What is the split-point or the split-subset.

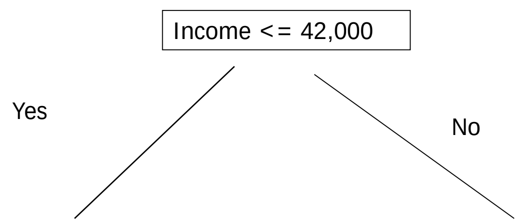
Three Partitioning Scenarios (1)

- Attribute is discrete-valued
  - A branch is created for each known value.
  - Multiple branches may be generated.



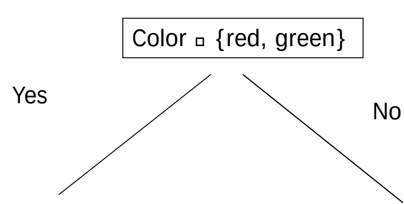
Three Partitioning Scenarios (2)

- Attribute is continuous-valued
  - Test attribute with the split-point
  - Binary tree is grown.



Three Partitioning Scenarios (3)

- Attribute is discrete-valued and binary tree is needed.
  - Test attribute with the split-subset
  - Binary tree is grown.



Attribute Selection Method: Information Gain

- Select the attribute with the highest information gain
  - This attribute minimizes the information needed to classify the tuples in the resulting partitions.
- Let p_i be the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i,D}|/|D|$
- Expected information (entropy) needed to classify a tuple in D:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

  - Entropy represents the average amount of information needed to identify the class label of a tuple in D.

Attribute Selection Method: Information Gain

- Attribute ***A*** has ***v*** distinct values.
  - ***A*** can be used to split D into ***v*** partitions, where D_j contains those tuples in D that have outcome a_j of A.
  - If A is selected, we wish each partition Dj is pure.
- Information needed (after using ***A*** to split D into ***v*** partitions) to classify D:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

  - The smaller the information needed, the greater the purity of the partitions.
- Information gained by branching on attribute A

$$Gain(A) = Info(D) - Info_A(D)$$

Algorithm for Decision Tree Induction

- Conditions for stopping partitioning
  - All tuples for a given node belong to the same class
  - Attribute_list is empty:
    - majority voting is employed for classifying the leaf
  - There are no tuples for a given branch Dj
    - A leaf is created with the majority class in D.

Let's grow one (Golf Data)

- Golf Data has two classes
  - Class 1 (Yes), Class 2 (No)
- D: 14 tuples, 5 Yes, 9 No.
  - $p_1 = 5/14$ & $p_2 = 9/14$
  - $Info(D) = - 5/14*\log_2(5/14) – 9/14*\log_2(9/14) = 0.94$

Outlook	Temp	Humidity	Windy	Class
Sunny	Hot	High	No	Yes
Sunny	Hot	High	Yes	Yes
O'cast	Hot	High	No	No
Rain	Mild	Normal	No	No
Rain	Cool	Normal	No	No
Rain	Cool	Normal	Yes	Yes
O'cast	Cool	Normal	Yes	No
Sunny	Mild	High	No	Yes
Sunny	Cool	Normal	No	No
Rain	Mild	Normal	No	No
Sunny	Mild	Normal	Yes	No
O'cast	Mild	High	Yes	No
O'cast	Hot	Normal	No	No
Rain	Mild	High	Yes	Yes

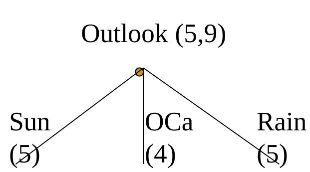
Attribute Selection

Outlook	Temp	Humidity	Windy	Class
Sunny	Hot	High	No	Yes
Sunny	Hot	High	Yes	Yes
O'cast	Hot	High	No	No
Rain	Mild	Normal	No	No
Rain	Cool	Normal	No	No
Rain	Cool	Normal	Yes	Yes
O'cast	Cool	Normal	Yes	No
Sunny	Mild	High	No	Yes
Sunny	Cool	Normal	No	No
Rain	Mild	Normal	No	No
Sunny	Mild	Normal	Yes	No
O'cast	Mild	High	Yes	No
O'cast	Hot	Normal	No	No
Rain	Mild	High	Yes	Yes

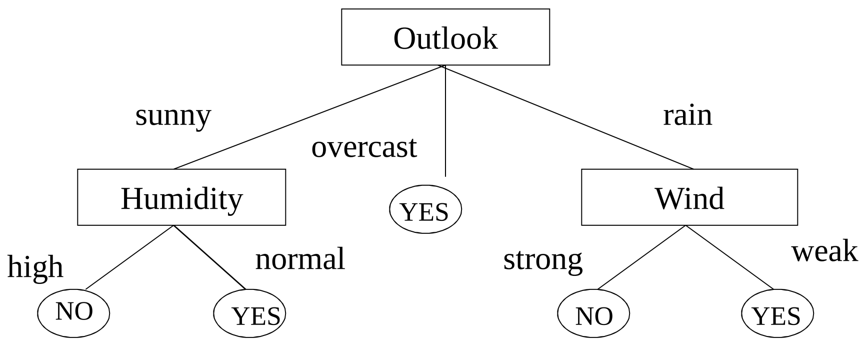
- A = outlook has 3 distinct values (sunny, overcast, rain)
  - D_{sunny} : 5 tuples, 3 Yes, 2 No, p_1= 3/5 & p_2 = 2/5
    - $Info(D_{sunny}) = -3/5*\log_2(3/5)-2/5*\log_2(2/5) = 0.97$
  - $D_{overcast}$: 4 tuples, 0 Yes, 4 No, p_1= 0 & p_2 = 1
    - $Info(D_{overcast}) = -1*\log_2(1) = 0$
  - D_{rain} : 5 tuples, 2 Yes,3 No, p_1= 2/5 & p_2 = 3/5
    - $Info(Drain) = -2/5*\log_2(2/5)-3/5*\log_2(3/5) = 0.97$
- $Info_A(D) = 5/14*0.97 + 4/14*0 + 5/14*0.97 = 0.69$

Attribute Selection

- Gain(outlook) = 0.94 – 0.69 = **0.25**
- Gain(temp) = 0.94 -0.911 = 0.029
- Gain(humidity) = 0.94 -0.704 =0.236
- Gain(windy) =0.94 -0.892 = 0.048



Decision Tree



Computing Information-Gain for Continuous-Value Attributes

- Let attribute A be a continuous-valued attribute
- Must determine the *best split point* for A
  - Sort the value A in increasing order
  - Typically, the midpoint between each pair of adjacent values is considered as a possible *split point*
    - $(a_i+a_{i+1})/2$ is the midpoint between the values of a_i and a_{i+1}
    - Given v values of attribute A, v-1 possible split points.
  - The point with the *minimum expected information requirement* for A is selected as the split-point for A
- Split:
  - D1 is the set of tuples in D satisfying A ≤ split-point, and D2 is the set of tuples in D satisfying A > split-point

Overfitting and Tree Pruning

- Overfitting: An induced tree may overfit the training data
  - Too many branches, some may reflect anomalies due to noise or outliers
  - Poor accuracy for unseen samples
- Two approaches to avoid overfitting
  - Prepruning: Halt tree construction early—do not split a node if this would result in the goodness measure falling below a threshold
    - Difficult to choose an appropriate threshold
  - *Postpruning: Remove branches from a "fully grown" tree—get a sequence of progressively pruned trees
    - Cost complexity algorithm: Use a set of data different from the training data to decide which is the "best pruned tree"

Existing Decision Tree Algorithms

- ID3
  - Use Information Gain to select attribute to split.
- C4.5
  - A successor of ID3, uses Gain Ratio to select attribute to split
  - Handling unavailable values, continuous attribute value range, and pruning the tree.
- CART
  - Use Gini Index to select attribute to split
  - Cost complexity pruning algorithm with validation set.

Classification Rules from Trees

- Easily understandable classification rules
  - Each leaf is equivalent to a classification rule.
- Example:
  - **IF** (income > 92.5) **AND** (Education < 1.5) **AND** (Family ≤ 2.5) **THEN** Class = 0

Decision Tree Induction

- Does not need any domain knowledge or parameter setting.
- Can handle high dimensional data.
- Easy to understand classification rules
- Learning and classification steps are simple and fast.
- Accuracy depends on training data.
- Can use SQL queries for accessing databases