Classification and Prediction:

- Classification and prediction are two forms of data analysis that can be used to extract modelsdescribing important data classes or to predict future data trends.
- Classification predicts categorical (discrete, unordered) labels, *prediction* models continuous valued functions.
- For example, we can build a classification model to categorize bank loan applications as either safe or risky, or a prediction model to predict the expenditures of potential customerson computer equipment given their income and occupation.
- A predictor is constructed that predicts a continuous-valued function, or ordered value, as opposed to a categorical label.
- Regression analysis is a statistical methodology that is most often used for numeric prediction.
- Many classification and prediction methods have been proposed by researchers in machine learning, pattern recognition, and statistics.
- Most algorithms are memory resident, typically assuming a small data size. Recent data mining research has built on such work, developing scalable classification and prediction techniques capable of handling large disk-resident data.

Preparing the Data for Classification and Prediction:

The following preprocessing steps may be applied to the data to help improve the accuracy,efficiency, and scalability of the classification or prediction process.

(i) Data cleaning:

- This refers to the preprocessing of data in order to remove or reduce *noise* (by applying smoothing techniques) and the treatment of *missing values* (e.g., by replacing a missing value with the most commonly occurring value for that attribute, or with the most probable value based on statistics).
- Although most classification algorithms have some mechanisms for handling noisy or missing data, this step can help reduce confusion during learning.

(ii) Relevance analysis:

- Many of the attributes in the data may be *redundant*.
- Correlation analysis can be used to identify whether any two given attributes are statistically related.
-

For example, a strong correlation between attributes *A*1 and *A*2 would suggest that one ofthe two could be removed from further analysis.

- A database may also contain *irrelevant* attributes. Attribute subset selection can be usedin these cases to find a reduced set of attributes such that the resulting probability distribution of the data classes is as close as possible to the original distribution obtained using all attributes.
- Hence, relevance analysis, in the form of correlation analysis and attribute subset selection, can be used to detect attributes that do not contribute to the classification or prediction task.
- Such analysis can help improve classification efficiency and scalability.

(iii) Data Transformation And Reduction

- The data may be transformed by normalization, particularly when neural networks ormethods involving distance measurements are used in the learning
- step.

  Normalization involves scaling all values for a given attribute so that they fall within asmall specified range, such as -1 to +1 or 0 to 1.
- The data can also be transformed by *generalizing* it to higher-level concepts. Concept hierarchies may be used for this purpose. This is particularly useful for continuousvalued attributes.
- For example, numeric values for the attribute *income* can be generalized to discrete ranges, such as *low*, *medium*, and *high*. Similarly, categorical attributes, like *street*, can be generalized to higher-level concepts, like *city*.
- Data can also be reduced by applying many other methods, ranging from wavelet transformation and principle components analysis to discretization techniques, suchas binning, histogram analysis, and clustering.

Comparing Classification and Prediction Methods:

- **Accuracy:**

  - The accuracy of a classifier refers to the ability of a given classifier to correctly predict the class label of new or previously unseen data (i.e., tuples without class label information).
  - The accuracy of a predictor refers to how well a given predictor can guess the value

of the predicted attribute for new or previously unseen data.

- **Speed:**

  This refers to the computational costs involved in generating and using the given classifier or predictor.

- **Robustness:**

  This is the ability of the classifier or predictor to make correct predictions given noisy data or data with missing values.

- **Scalability:**

  This refers to the ability to construct the classifier or predictor efficiently given large amounts of data.

- **Interpretability:**

  - This refers to the level of understanding and insight that is provided by the classifier or predictor.
  - Interpretability is subjective and therefore more difficult to assess.

General Approach to Classification

"How does classification work?" Data classification is a two-step process, consisting of a learning step (where a classification model is constructed) and a classification step (where the model is used to predict class labels for given data).

In the first step, a classifier is built describing a predetermined set of data classes or concepts. This is the learning step (or training phase), where a classification algorithm builds the classifier by analyzing or "learning from" a training set made up of database tuples and their associated class labels. A tuple, X, is represented by an n-dimensional attribute vector, X =(x1, x2, … , xn), depicting n measurements made on the tuple from n database attributes, respectively, A1, A2, …., An.1 Each tuple, X, is assumed to belong to a predefined class as determined by another database attribute called the class label attribute. The class label attribute is discrete-valued and unordered. It is categorical (or nominal) in that each value serves as a category or class. The individual tuples making up the training set are referred to as training tuples.

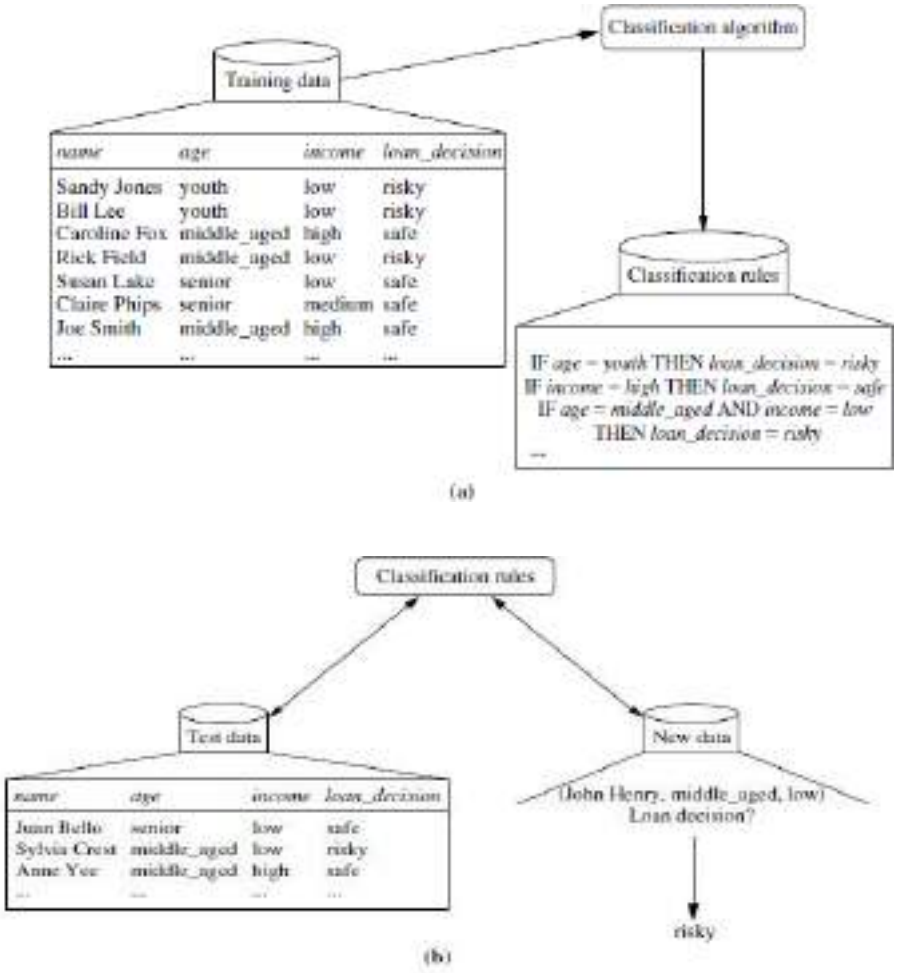


Figure 8.1 The data classification process: (a) *Learning*: Training data are analyzed by a classification algorithm. Here, the class label attribute is *loan_decision*, and the learned model or classifier is represented in the form of classification rules. (b) *Classification*: Test data are used to estimate the accuracy of the classification rules. If the accuracy is considered acceptable, the rules can be applied to the classification of new data tuples.

Because the class label of each training tuple is provided, this step is also known as supervised learning (i.e., the learning of the classifier is "supervised" in that it is told to which class each training tuple belongs). It contrasts with unsupervised learning (or clustering), in which the class label of eachtraining tuple is not known, and the number or set of classes to be learned may not be known in advance. For example, if we did not have the loan decision data available for the training set, we could use clustering to try to determine "groups of like tuples," which may correspond to risk groups within the loan application data.

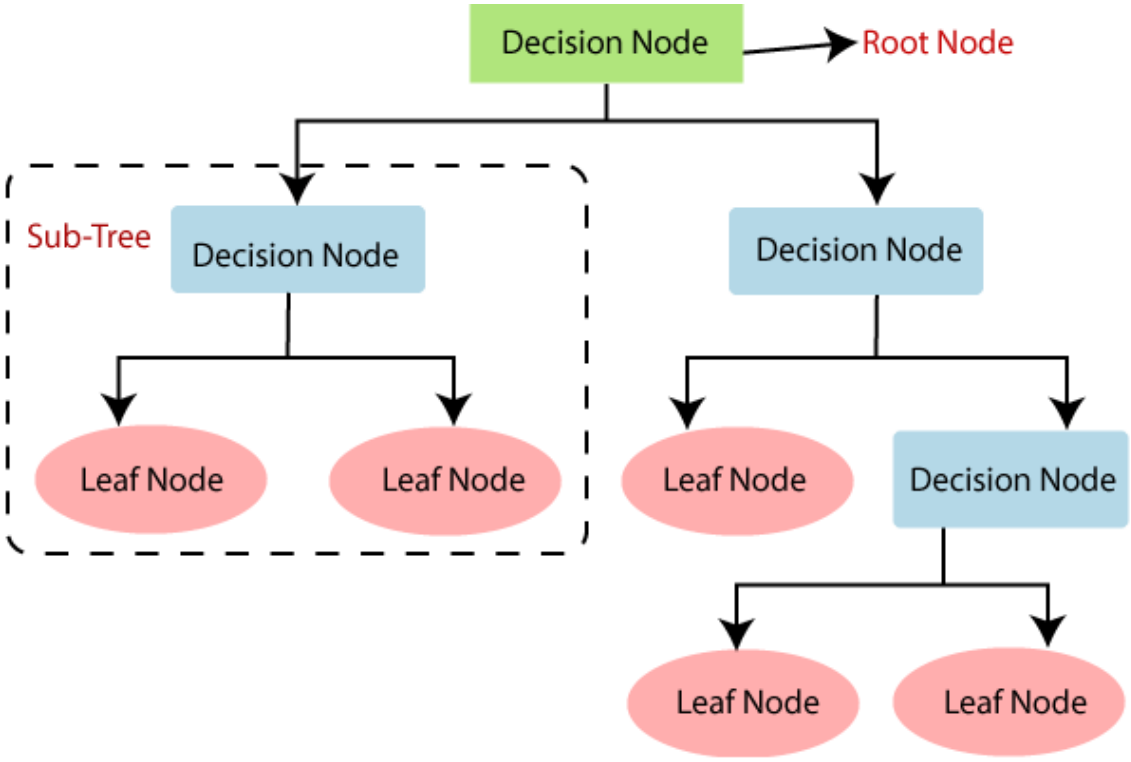
"What about classification accuracy?" In the second step (Figure 8.1b), the model is used forclassification. First, the predictive accuracy of the classifier is estimated. If we were to use the training set to measure the classifier's accuracy, this estimate would likely be optimistic, because the classifier tends to overfit the data (i.e., during learning it may

incorporate some particular anomalies of the training data that are not present in the general data set overall). Therefore, a test set is used, made up of test tuples and their associated class labels. They are independent of the training tuples, meaning that they were not used to construct the classifier.

The accuracy of a classifier on a given test set is the percentage of test set tuples that are correctly classified by the classifier.

Decision Tree Algorithm

- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems.
- It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**
- In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- ***It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.***
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the **CART algorithm,** which stands for **Classification and Regression Tree algorithm.**
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.
- The general structure of a decision tree



- **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- **Branch/Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

Why use Decision Trees?

- There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:
- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

How does the Decision Tree algorithm Work?

- In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.
- For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:
- **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.
- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.

- **Step-4:** Generate the decision tree node, which contains the best attribute.
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.
- **ID3 (Entropy, Information Gain)**
- While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.** By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:
- **Information Gain**

  Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:
- Information Gain= Entropy(S)-[(Weighted Avg) *Entropy(each feature)
- **Entropy:** Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:
- Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)

  Advantages of the Decision Tree
- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

  Disadvantages of the Decision Tree
- The decision tree contains lots of layers, which makes it complex.

- It may have an overfitting issue, which can be resolved using the **Random Forest algorithm.**
- For more class labels, the computational complexity of the decision tree may increase

Issues In Decision Tree Learning

1. Overfitting the Data
2. Incorporating Continuous valued attributes
3. Handling training examples with missing attribute values
4. Handling attributes with different costs
5. Alternative measures for selecting attributes

ID3 ALGORITHM

- ID3 stands for Iterative Dichotomiser 3 and is named such because the algorithm iteratively (repeatedly) dichotomizes(divides) features into two or more groups at each step.
- Invented by Ross Quinlan, ID3 uses a **top-down greedy** approach to build a decision tree. In simple words, the **top-down** approach means that we start building the tree from the top and the **greedy** approach means that at each iteration we select the best feature at the present moment to create a node.
- Most generally ID3 is only used for classification problems with nominal features only.
- ID3 algorithm selects the best feature at each step while building a Decision tree.
  How does ID3 select the best feature?
- ID3 uses **Information Gain** or just **Gain** to find the best feature.
- Information Gain calculates the reduction in the entropy and measures how well a given feature separates or classifies the target classes. The feature with the **highest Information Gain** is selected as the **best** one
- In simple words, **Entropy** is the measure of disorder and the Entropy of a dataset is the measure of disorder in the target feature of the dataset.
  In the case of binary classification (where the target column has only two types of classes) entropy is **0** if all values in the target column are homogenous(similar) and will be **1** if the target column has equal number values for both the classes.

EXAMPLE:

PlayTennis: training examples

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

1. COMPUTE THE **ENTROPY** FOR DATA-SET **ENTROPY(S)**

2. FOR EVERY ATTRIBUTE/FEATURE:

    1. CALCULATE ENTROPY FOR ALL OTHER VALUES **ENTROPY(A)**

    2. TAKE **AVERAGE INFORMATION ENTROPY** FOR THE CURRENT ATTRIBUTE

    3. CALCULATE **GAIN** FOR THE CURRENT ATTRIBUTE

3. PICK THE **HIGHEST GAIN ATTRIBUTE**.

4. **REPEAT** UNTIL WE GET THE TREE WE DESIRED.

STEP 1: CREATE A ROOT NODE

- HOW TO CHOOSE THE ROOT NODE?

  The attribute that best classifies the training data, use this attribute at the root of the tree.

- HOW TO CHOOSE THE BEST ATTRIBUTE?

  So from here, *ID3 algorithm* begins

- Calculate **Entropy** (Amount of uncertainity in dataset):

$$Entropy = \frac{-p}{p+n} log_2(\frac{p}{p+n}) - \frac{n}{p+n} log_2(\frac{n}{p+n})$$

- Calculate **Average Information**:

$$I(Attribute) = \sum \frac{p_i + n_i}{p+n} Entropy(A)$$

- Calculate **Information Gain**: (Difference in Entropy before and after splitting dataset on attribute A)

$$Gain = Entropy(S) - I(Attribute)$$

S. No.	Outlook	Temperature	Humidity	Windy	PlayTennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rainy	Mild	High	Weak	Yes
5	Rainy	Cool	Normal	Weak	Yes
6	Rainy	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rainy	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rainy	Mild	High	Strong	No

P = 9

N = 5

Total = 14

- Calculate **Entropy(S)**:

$$Entropy = \frac{-p}{p+n} log_2(\frac{p}{p+n}) - \frac{n}{p+n} log_2(\frac{n}{p+n})$$

$$Entropy(S) = \frac{-9}{9+5} log_2(\frac{9}{9+5}) - \frac{5}{9+5} log_2(\frac{5}{9+5})$$

$$Entropy(S) = \frac{-9}{14} log_2(\frac{9}{14}) - \frac{5}{14} log_2(\frac{5}{14}) \quad = 0.940$$

- For each Attribute: (let say **Outlook**)
  - Calculate Entropy for each Values, i.e for 'Sunny', 'Rainy','Overcast'

Outlook	PlayTennis
Sunny	No
Sunny	No
Sunny	No
Sunny	Yes
Sunny	Yes

Outlook	PlayTennis
Rainy	Yes
Rainy	Yes
Rainy	No
Rainy	Yes
Rainy	No

Outlook	PlayTennis
Overcast	Yes
Overcast	Yes
Overcast	Yes
Overcast	Yes

Outlook	p	n	Entropy
Sunny	2	3	0.971
Rainy	3	2	0.971
Overcast	4	0	0

- Calculate **Average Information Entropy**:

$$I(Outlook) = \frac{p_{sunny} + n_{sunny}}{p+n} Entropy(Outlook = Sunny) + \frac{p_{rainy} + n_{rainy}}{p+n} Entropy(Outlook = Rainy) + \frac{p_{Overcast} + n_{Overcast}}{p+n} Entropy(Outlook = Overcast)$$

$$I(Outlook) = \frac{3+2}{9+5} * 0.971 + \frac{2+3}{9+5} * 0.971 + \frac{4+0}{9+5} * 0 = 0.693$$

- Calculate **Gain**: attribute is Outlook

$$Gain = Entropy(S) - I(Attribute)$$

$$Entropy(S) = 0.940$$

$$Gain(Outlook) = 0.940 - 0.693 = 0.247$$

- For each Attribute: (let say **Temperature**)
  - Calculate Entropy for each Temp, i.e for 'Hot', 'Mild' and 'Cool'

Temperature	PlayTennis
Hot	No
Hot	No
Hot	Yes
Hot	Yes

Temperature	PlayTennis
Mild	Yes
Mild	No
Mild	Yes
Mild	Yes
Mild	Yes
Mild	No

Temperature	PlayTennis
Cool	Yes
Cool	No
Cool	Yes
Cool	Yes

Temperature	p	n	Entropy
Hot	2	2	1
Mild	4	2	0.918
Cool	3	1	0.811

- Calculate **Average Information Entropy**:

$$I(Temperature) = \frac{p_{hot} + n_{hot}}{p+n} Entropy(Temperature = Hot) +$$

$$\frac{p_{mild} + n_{mild}}{p+n} Entropy(Temperature = Mild) +$$

$$\frac{p_{Cool} + n_{Cool}}{p+n} Entropy(Temperature = Cool)$$

$$I(Temperature) = \frac{2+2}{9+5} * 1 + \frac{4+2}{9+5} * 0.918 + \frac{3+1}{9+5} * 0.811 => 0.911$$

- Calculate **Gain**: attribute is Temperature

$$Gain = Entropy(S) - I(Attribute)$$

$$Entropy(S) = 0.940$$

$$Gain(Temperature) = 0.940 - 0.911 = 0.029$$

- For each Attribute: (let say **Humidity**)
  - Calculate Entropy for each Humidity, i.e for 'High', 'Normal'

Humidity	PlayTennis
Normal	Yes
Normal	No
Normal	Yes
Normal	Yes
Normal	Yes
Normal	Yes
Normal	Yes

Humidity	PlayTennis
High	No
High	No
High	Yes
High	Yes
High	No
High	Yes
High	No

Humidity	p	n	Entropy
High	3	4	0.985
Normal	6	1	0.591

- Calculate **Average Information Entropy**:

$$I(Humidity) = \frac{p_{High} + n_{High}}{p+n} Entropy(Humidity = High) +$$

$$\frac{p_{Normal} + n_{Normal}}{p+n} Entropy(Humidity = Normal)$$

$$I(Humidity) = \frac{3+4}{9+5} * 0.985 + \frac{6+1}{9+5} * 0.591 => 0.788$$

- For each Attribute: (let say **Windy**)
  - Calculate Entropy for each Windy, i.e for 'Strong' and 'Weak'

Windy	PlayTennis
Weak	No
Weak	Yes
Weak	Yes
Weak	Yes
Weak	No
Weak	Yes
Weak	Yes
Weak	Yes

Windy	PlayTennis
Strong	No
Strong	No
Strong	Yes
Strong	Yes
Strong	Yes
Strong	No

Windy	p	n	Entropy
Strong	3	3	1
Weak	6	2	0.811

- Calculate **Average Information Entropy**:

$$I(Windy) = \frac{p_{Strong} + n_{Strong}}{p+n} Entropy(Windy = Strong) + \frac{p_{Weak} + n_{Weak}}{p+n} Entropy(Windy = Weak)$$

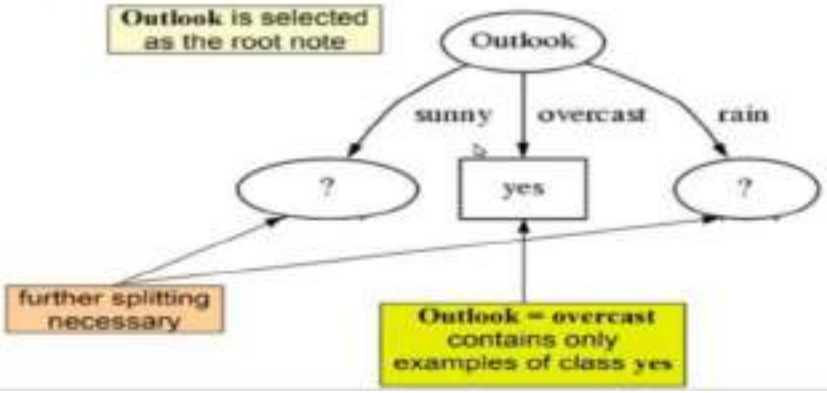
$$I(Windy) = \frac{3+3}{9+5} * 1 + \frac{6+2}{9+5} * 0.811 => 0.892$$

- PICK THE HIGHEST GAIN ATTRIBUTE.

Attributes	Gain
Outlook	0.247
Temperature	0.029
Humidity	0.152
Windy	0.048

ROOT NODE:
OUTLOOK

Outlook	Temperature	Humidity	Windy	PlayTennis
Overcast	Hot	High	Weak	Yes
Overcast	Cool	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes



- REPEAT THE SAME THING FOR SUB-TREES TILL WE GET THE TREE.

Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes

OUTLOOK = "SUNNY"

Outlook	Temperature	Humidity	Windy	PlayTennis
Rainy	Mild	High	Weak	Yes
Rainy	Cool	Normal	Weak	Yes
Rainy	Cool	Normal	Strong	No
Rainy	Mild	Normal	Weak	Yes
Rainy	Mild	High	Strong	No

OUTLOOK = "RAINY"

Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes

P= 2 N= 3
Total= 5

- ENTROPY:

$$Entropy = \frac{-p}{p+n} log_2(\frac{p}{p+n}) - \frac{n}{p+n} log_2(\frac{n}{p+n})$$

$$Entropy(S_{sunny}) = \frac{-2}{2+3} log_2(\frac{2}{2+3}) - \frac{3}{2+3} log_2(\frac{3}{2+3})$$

=>0.971

- For each Attribute: (let say **Humidity**):
  - Calculate Entropy for each Humidity, i.e for 'High' and 'Normal'

Outlook	Humidity	PlayTennis
Sunny	High	No
Sunny	High	No
Sunny	High	No
Sunny	Normal	Yes
Sunny	Normal	Yes

Humidity	p	n	Entropy
high	0	3	0
normal	2	0	0

  - Calculate **Average Information Entropy**: I(Humidity) = 0
  - Calculate **Gain**: Gain = 0.971

- For each Attribute: (let say **Windy**):
  - Calculate Entropy for each Windy, i.e for 'Strong' and 'Weak'

Outlook	Windy	PlayTennis
Sunny	Strong	No
Sunny	Strong	Yes
Sunny	Weak	No
Sunny	Weak	No
Sunny	Weak	Yes

Windy	p	n	Entropy
Strong	1	1	1
Weak	1	2	0.918

  - Calculate **Average Information Entropy**: I(Windy) = 0.951
  - Calculate **Gain**: Gain = 0.020

- For each Attribute: (let say **Temperature**):
  - Calculate Entropy for each Windy, i.e for 'Cool', 'Hot' and 'Mild'

Outlook	Temperature	PlayTennis
Sunny	Cool	Yes
Sunny	Hot	No
Sunny	Hot	No
Sunny	Mild	No
Sunny	Mild	Yes

Temperature	p	n	Entropy
Cool	1	0	0
Hot	0	2	0
Mild	1	1	1

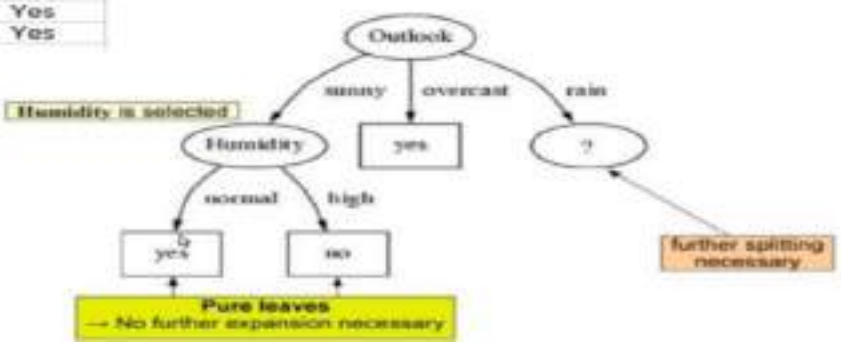
  - Calculate **Average Information Entropy**: I(Temp) = 0.4
  - Calculate **Gain:** Gain = 0.571

- PICK THE HIGHEST GAIN ATTRIBUTE.

Attributes	Gain
Temperature	0.571
Humidity	0.971
Windy	0.02

NEXT NODE IN SUNNY: HUMIDITY

Outlook	Humidity	PlayTennis
Sunny	High	No
Sunny	High	No
Sunny	High	No
Sunny	Normal	Yes
Sunny	Normal	Yes



Outlook	Temperature	Humidity	Windy	PlayTennis
Rainy	Mild	High	Weak	Yes
Rainy	Cool	Normal	Weak	Yes
Rainy	Cool	Normal	Strong	No
Rainy	Mild	Normal	Weak	Yes
Rainy	Mild	High	Strong	No

P= 3 N= 2

Total = 5

- ENTROPY:

$$Entropy = \frac{-p}{p+n} log_2(\frac{p}{p+n}) - \frac{n}{p+n} log_2(\frac{n}{p+n})$$

$$Entropy(S_{Rainy}) = \frac{-3}{3+2} log_2(\frac{3}{3+2}) - \frac{2}{3+2} log_2(\frac{2}{2+3})$$

=>0.971

- For each Attribute: (let say **Humidity**):
  - Calculate Entropy for each Humidity, i.e for 'High' and 'Normal'

Outlook	Humidity	PlayTennis
Rainy	High	Yes
Rainy	High	No
Rainy	Normal	Yes
Rainy	Normal	No
Rainy	Normal	Yes

Attribute	p	n	Entropy
High	1	1	1
Normal	2	1	0.918

  - Calculate **Average Information Entropy**: I(Humidity) = 0.951
  - Calculate **Gain**: Gain = 0.020

- For each Attribute: (let say **Windy**):
  - Calculate Entropy for each Windy, i.e for 'Strong' and 'Weak'

Outlook	Windy	PlayTennis
Rainy	Strong	No
Rainy	Strong	No
Rainy	Weak	Yes
Rainy	Weak	Yes
Rainy	Weak	Yes

Attribute	p	n	Entropy
Strong	0	2	0
Weak	3	0	0

  - Calculate **Average Information Entropy**: I(Windy) = 0
  - Calculate **Gain**: Gain = 0.971

- For each Attribute: (let say **Temperature**):
  - Calculate Entropy for each Windy, i.e for 'Cool', 'Hot' and 'Mild'

Outlook	Temperature	PlayTennis
Rainy	Mild	Yes
Rainy	Cool	Yes
Rainy	Cool	No
Rainy	Mild	Yes
Rainy	Mild	No

Attribute	p	n	Entropy
Cool	1	1	1
Mild	2	1	0.918

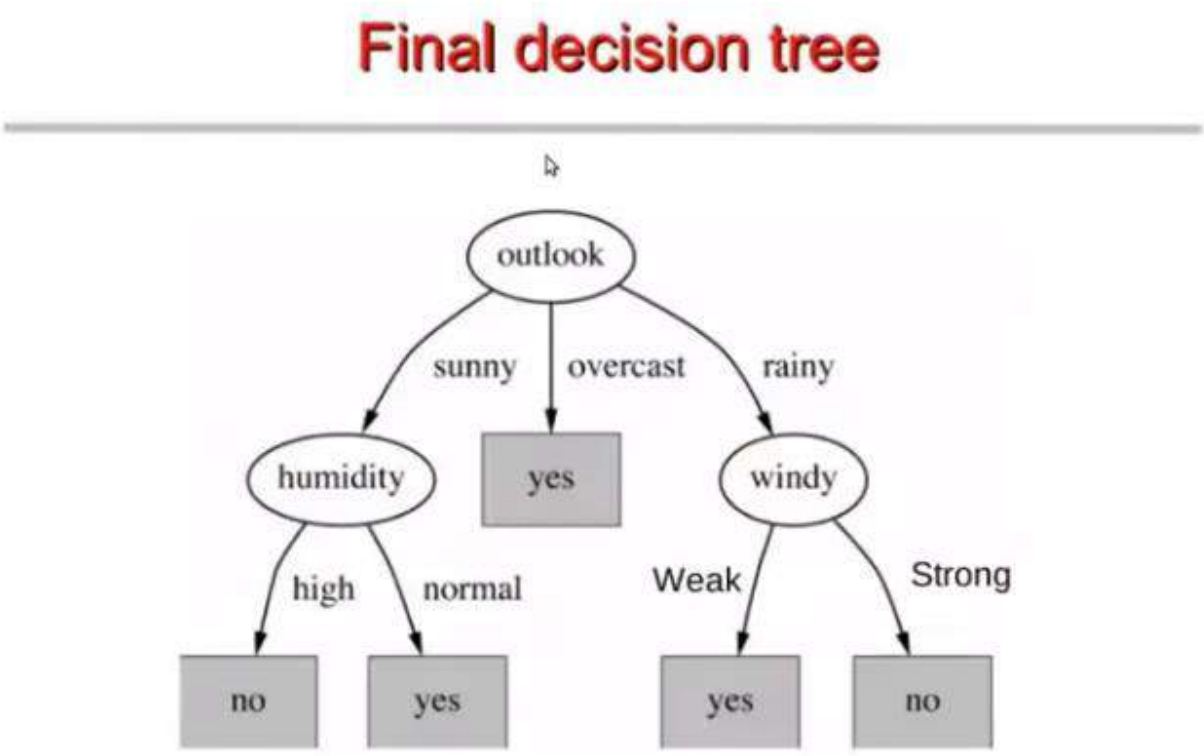
  - Calculate **Average Information Entropy**: I(Temp) = 0.951
  - Calculate **Gain:** Gain = 0.020

- PICK THE HIGHEST GAIN ATTRIBUTE.

Attributes	Gain
Humidity	0.02
Windy	0.971
Temperature	0.02

NEXT NODE IN RAINY:

WINDY



Rule-Based Classification

Using IF-THEN Rules for Classification

A rule-based classifier uses a set of IF-THEN rules for classification. An **IF-THEN** rule is an expression of the form

IF *condition* THEN *conclusion*.An example is rule *R*1,

*R*1: IF *age* =*youth* AND *student* =*yes* THEN *buys computer* = *yes*.

The "IF" part (or left side) of a rule is known as the **rule antecedent** or **precondition**. The "THEN" part (or right side) is the **rule consequent**. In the rule antecedent, the condition consists of one or more *attribute tests* (e.g., *age* = *youth* and *student* = *yes*) that are logically ANDed. The rule's consequent contains a class prediction (in this case, we are predicting whether a customer will buy a computer). *R*1 can also be written as

*R*1: (*age* = *youth*) ^ (*student* = *yes*) ⇒ (*buys_computer* = *yes*).

If the condition (i.e., all the attribute tests) in a rule antecedent holds true for a given tuple, we say that the rule antecedent is **satisfied** (or simply, that the rule is satisfied) and that the rule **covers** the tuple.

Bayesian Classification:

- Bayesian classifiers are statistical classifiers.
- They can predictclass membership probabilities, such as the probability that a given tuplebelongs toa particular class.
- Bayesian classification is based on Bayes' theorem.

Bayes' Theorem:

- Let X be a data tuple. In Bayesian terms, X is considered ‑evidence.**||**and it is described by measurements made on a set of n attributes.

Let H be some hypothesis, such as that the data tuple X belongs to a specified class C.

- For classification problems, we want to determine P(H|X), the probability that the hypothesisH holds given the ‑evidence**||** or observed data tuple X.
- P(H|X) is the posterior probability, or a posteriori probability, of H conditioned on X.
- Bayes' theorem is useful in that it providesa way of calculating the posterior probability,

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)}.$$

P(*H*|***X***), from *P*(*H*), *P*(***X***|*H*), and *P*(***X***).

Naïve Bayesian Classification:

The naïve Bayesian classifier, or simple Bayesian classifier, works as follows:

1. Let D be a training set of tuples and their associated class labels. As usual, each tuple is represented by an n-dimensional attribute vector, X = (x1, x2, …,xn), depicting n measurements made on the tuple from n attributes, respectively, A1, A2, …, An.

2. Suppose that there are m classes, C1, C2, …, Cm. Given a tuple, X, the classifier will predict that X belongs to the class having the highest posterior probability, conditioned on X.

That is, the naïve Bayesian classifier predicts that tuple X belongs to the class Ci if and onlyif

$$P(C_i|X) > P(C_j|X) \quad \text{for } 1 \leq j \leq m, j \neq i.$$

Thus we maximize P(Cij**X**). The classCifor which P(Cij**X**) is maximized is called the maximum posteriori hypothesis. By Bayes' theorem

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}.$$

3. As P(X) is constant for all classes, only P(X|Ci)P(Ci) need be maximized. If the class

prior probabilities are not known, then it is commonly assumed that the classes are equallylikely, that is, P(C1) = P(C2) = …= P(Cm), and we would therefore maximize P(X|Ci). Otherwise, we maximize P(X|Ci)P(Ci).Given data sets with many attributes, it would be extremely computationally expensiveto compute P(X|Ci). In order to reduce computation in evaluating P(X|Ci), the naive assumption of class conditional independence is made. This presumes that the values of the attributes areconditionally independent of one another, given the class label of the tuple. Thus,

$$\begin{aligned} P(X|C_i) &= \prod_{k=1}^{n} P(x_k|C_i) \\ &= P(x_1|C_i) \times P(x_2|C_i) \times \cdots \times P(x_n|C_i). \end{aligned}$$

We can easily estimate the probabilities *P*(*x*1|*Ci*), *P*(*x*2|*Ci*), : : : , *P*(*xn*|*Ci*) fromthe trainingtuples. For eachattribute, we look at whether the attribute is categorical or continuous-valued. Forinstance, to compute *P*(***X***|*Ci*), we consider the following:

- If $A_k$is categorical, then $P(x_k|Ci)$ is the number of tuples of class *Ci*in *D* havingthe value

$x_k$for A_k, divided by $|C_{i,D}|$ the number of tuples of class $C_i$in *D*.

- If $A_k$is continuous-valued, then we need to do a bit more work, but the calculationis prettystraightforward.

A continuous-valued attribute is typically assumed tohave a Gaussian distribution

with amean μ and standard deviation , defined by

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

$$P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i}).$$

4. In order to predict the class label of *X*, *P*(*X*j*Ci*)*P*(*Ci*) is evaluated for each class *Ci*. The classifier predicts that the class label of tuple *X* is the class *Ci*if and only if

$$P(X|C_i)P(C_i) > P(X|C_j)P(C_j) \quad \text{for } 1 \leq j \leq m, j \neq i.$$

Consider the given Dataset ,Apply Naive Baye's Algorithm and Predict that if a fruit has the following properties then which type of the fruit it is
Fruit = {Yellow , Sweet ,long}

Frequency Table:

Fruit	Yellow	Sweet	Long	Total
Mango	350	450	0	650
Banana	400	300	350	400
Others	50	100	50	150
Total	800	850	400	1200

Solution:

P(A|B) = (P(B|A) * P(A))/ P(B)

1. Mango:

P(X | Mango) = P(Ye | Yellow) * P(Sweet | Mango) * P(Long | Mango)

1. a)P(Yellow | Mango) = (P(Mango | Yellow) * P(Yellow))/ P (Mango)

= ((350/800) * (800/1200)) / (650/1200)

P(Yellow | Mango)= 0.53 →1

1.b)P(Sweet | Mango) = (P(Sweet | Mango) * P(Sweet))/ P (Mango)

= ((450/850) * (850/1200)) / (650/1200)

P(Sweet | Mango)= 0.69 → 2

1. c)P(Long | Mango) = (P(Long | Mango) * P(Long))/ P (Mango)

= ((0/650) * (400/1200)) / (800/1200)

P(Long | Mango)= 0 → 3

On multiplying eq 1,2,3 ==> P(X | Mango) = 0.53 * 0.69 * 0

P(X | Mango) = 0

2. Banana:

P(X | Banana) = P(Yellow | Banana) * P(Sweet | Banana) * P(Long | Banana)

2.a) P(Yellow | Banana) = (P(Banana | Yellow) * P(Yellow))/ P (Banana)

= ((400/800) * (800/1200)) / (400/1200)

P(Yellow | Banana) = 1 → 4

2.b) P(Sweet | Banana) = (P(Banana | Sweet) * P(Sweet))/ P (Banana)

= ((300/850) * (850/1200)) / (400/1200)

P(Sweet | Banana) = .75→ 5

2.c)P(Long | Banana) = (P(Banana | Yellow) * P(Long))/ P (Banana)

= ((350/400) * (400/1200)) / (400/1200)

P(Yellow | Banana) = 0.875 → 6

On multiplying eq 4,5,6 ==> P(X | Banana) = 1 * .75 * 0.875

P(X | Banana) = 0.6562

3. Others:

P(X | Others) = P(Yellow | Others) * P(Sweet | Others) * P(Long | Others)

3.a) P(Yellow | Others) = (P(Others| Yellow) * P(Yellow))/ P (Others)

= ((50/800) * (800/1200)) / (150/1200)

P(Yellow | Others) = 0.34→ 7

3.b) P(Sweet | Others) = (P(Others| Sweet) * P(Sweet))/ P (Others)

= ((100/850) * (850/1200)) / (150/1200)

P(Sweet | Others) = 0.67 → 8

3.c) P(Long | Others) = (P(Others| Long) * P(Long))/ P (Others)

= ((50/400) * (400/1200)) / (150/1200)

P(Long | Others) = 0.34 → 9

On multiplying eq 7,8,9 ==> P(X | Others) = 0.34 * 0.67* 0.34

P(X | Others) = 0.07742

So finally from P(X | Mango) == 0 , P(X | Banana) == 0.65 and P(X| Others) == 0.07742.

We can conclude **Fruit{Yellow,Sweet,Long} is Banana.**

Lazy Learners (or Learning from Your Neighbors)

Decision tree induction, Bayesian classification, rule-based classification, classification by backpropagation, support vector machines, and classification based on association rule mining—are all examples of *eager learners.* **Eager learners**, when given a set of training tuples, will construct a generalization (i.e., classification) model before receiving new (e.g., test) tuples to classify. We can think of the learned model as being ready and eager to classify previously unseen tuples. Imagine a contrasting lazy approach, in which the learner instead waits until the last minute before doing any model construction to classify a given test tuple. That is, when given a training tuple, a **lazy learner** simply stores it (or does only a little minor processing)and waits until it is given a test tuple. Only when it sees the test tuple does it perform generalization to classify the tuple based on its similarity to the stored training tuples. Unlike eager learning methods, lazy learners do less work when a training tuple is presented and more work when making a classification or numeric prediction. Because lazy learners store the training tuples or "instances," they are also referred to as **instance-based learners**, even though all learning is essentially based on instances. When making a classification or numeric prediction, lazy learners can be computationally expensive. They require efficient storage techniques and are well suited to implementation on parallel hardware. They offer little explanation or insight into the data's structure. Lazy learners, however, naturally support incremental learning. They are able to model complex decision spaces having hyper polygonal shapes that may not be as easily describable by other learning algorithms (such as hyper rectangular shapes modelled by decision trees).

k-Nearest- Neighbour Classifiers

The *k*-nearest-neighbour method was first described in the early 1950s. The method is labor intensive when given large training sets, and did not gain popularity until the 1960s when increased computing power became available. It has since been widely used in the area of pattern recognition. Nearest-neighbour classifiers are based on learning by analogy, that is, by comparing a given test tuple with training tuples that are similar to it..

Case-Based Reasoning

Case-based reasoning (CBR) classifiers use a database of problem solutions to solve new problems. Unlike nearest-neighbour classifiers, which store training tuples as points in Euclidean space, CBR stores the tuples or "cases" for problem solving as complex symbolic descriptions. Business applications of CBR include problem resolution for customer service help desks, where

cases describe product-related diagnostic problems. CBR has also been applied to areas such as engineering and law, where cases are either technical designs or legal rulings, respectively. Medical education is another area for CBR, where patient case histories and treatments are used to help diagnose and treat new patients.