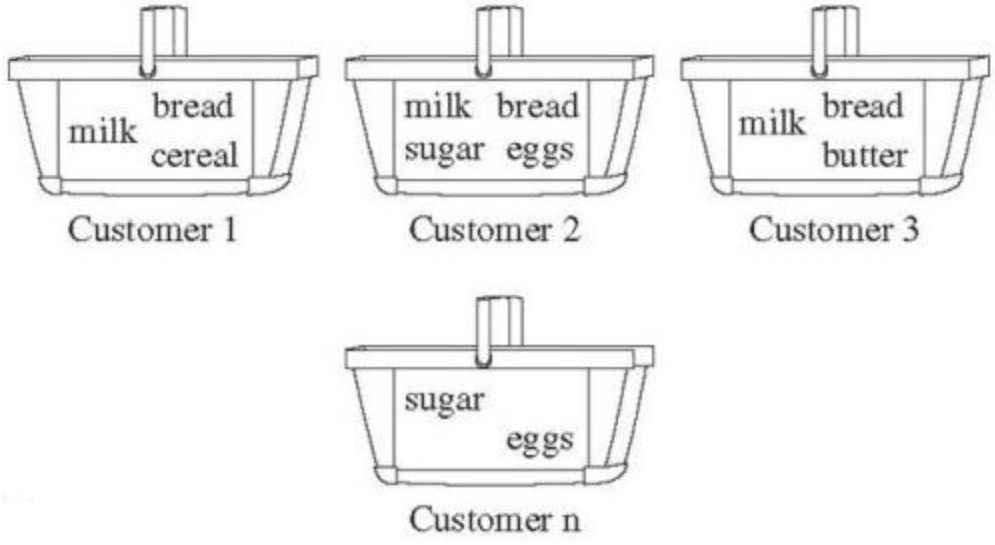
Mining Frequent Pattern

Taking a set of data and applying statistical methods to find interesting and previously-unknown patterns within said set of data. We aren't looking to classify instances or perform instance clustering; we simply want to learn patterns of subsets which emerge within a dataset and across instances, which ones emerge frequently, which items are associated, and which items correlate with others. It's easy to see why the above terms become conflated.

So, let's have a look at this essential aspect of data mining. Foregoing the Apriori algorithm for now, I will simply use the term *frequent pattern mining* to refer to the big tent of concepts outlined above, even if somewhat flawed (and even if I personally prefer the less often used term *association mining*).



Market Basket Analysis

Frequent patterns are patterns which appear frequently within a dataset (surprised?). A *frequent itemset* is one which is made up of one of these patterns, which is why frequent pattern mining is often alternately referred to as frequent itemset mining.

Frequent pattern mining is most easily explained by introducing *market basket analysis* (or affinity analysis), a typical usage for which it is well-known. Market basket analysis attempts to identify associations, or patterns, between the various items that have been chosen by a particular shopper and placed in their market basket, be it real or virtual, and assigns support and confidence measures for comparison. The value of this lies in cross-marketing and customer behavior analysis.

The generalization of market basket analysis is frequent pattern mining, and is actually quite similar to classification except that any attribute, or combination of attributes (and not just the *class*), can be predicted in association. As association does not require the pre-labeling of classes, it is a form of unsupervised learning.

Confidence, Support, and Association Rules

If we think of the total set of items available in our set (sold at a physical store, at an online retailer, or something else altogether, such as transactions for fraud detection analysis), then each item can be represented by a Boolean variable, representing whether or not the item is present within a given "basket." Each basket is then simply a Boolean vector, possibly quite lengthy dependent on the number of available items. A dataset would then be the resulting matrix of all possible basket vectors.

This collection of Boolean basket vectors are then analyzed for associations, patterns, correlations, or whatever it is you would like to call these relationships. One of the most common ways to represent these patterns is via association rules, a single example of which is given below:

milk => bread [support = 25%, confidence = 60%]

How do we know how interesting or insightful a given rule may be? That's where support and confidence come in.

Support is a measure of *absolute frequency*. In the above example, the support of 25% indicates that, in our finite dataset, milk and bread are purchased together in 25% of all transactions.

Confidence is a measure of *correlative frequency*. In the above example, the confidence of 60% indicates that 60% of those who purchased milk also purchased bread.

In a given application, association rules are generally generated within the bounds of some predefined minimum threshold for both confidence and support, and rules are only considered interesting and insightful if they meet these minimum thresholds.

Apriori Algorithm:

The Apriori algorithm uses frequent itemsets to generate association rules, and it is designed to work on the databases that contain transactions. With the help of these association rule, it determines how strongly or how weakly two objects are connected. This algorithm uses a **breadth-first search** and **Hash Tree** to calculate the itemset associations efficiently. It is the iterative process for finding the frequent itemsets from the large dataset.

This algorithm was given by the **R. Agrawal** and **Srikant** in the year **1994**. It is mainly used for *market basket analysis* and helps to find those products that can be bought together. It can also be used in the healthcare field to find drug reactions for patients.

What is Frequent Itemset?

Frequent itemsets are those items whose support is greater than the threshold value or user-specified minimum support. It means if A & B are the frequent itemsets together, then individually A and B should also be the frequent itemset.

Suppose there are the two transactions: A= {1,2,3,4,5}, and B= {2,3,7}, in these two transactions, 2 and 3 are the frequent itemsets.

Note: To better understand the apriori algorithm, and related term such as support and confidence, it is recommended to understand the association rule learning.

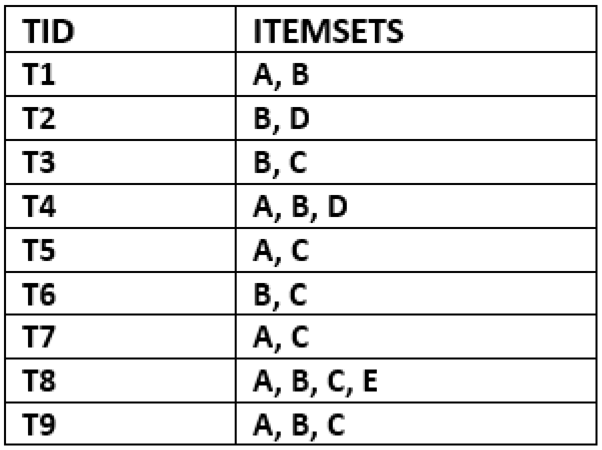
Steps for Apriori Algorithm

Below are the steps for the apriori algorithm:
Step-1: Determine the support of itemsets in the transactional database, and select the minimum support and confidence.
Step-2: Take all supports in the transaction with higher support value than the minimum or selected support value.
Step-3: Find all the rules of these subsets that have higher confidence value than the threshold or minimum confidence.
Step-4: Sort the rules as the decreasing order of lift.

Apriori Algorithm Working

We will understand the apriori algorithm using an example and mathematical calculation:

Example: Suppose we have the following dataset that has various transactions, and from this dataset, we need to find the frequent itemsets and generate the association rules using the Apriori algorithm:



TID	ITEMSETS
T1	A, B
T2	B, D
T3	B, C
T4	A, B, D
T5	A, C
T6	B, C
T7	A, C
T8	A, B, C, E
T9	A, B, C

Given: Minimum Support= 2, Minimum Confidence= 50%

Solution:

Step-1: Calculating C1 and L1:

- In the first step, we will create a table that contains support count (The frequency of each itemset individually in the dataset) of each itemset in the given dataset. This table is called the **Candidate set or C1.**

Itemset	Support_Count
A	6
B	7
C	5
D	2
E	1

Now, we will take out all the itemsets that have the greater support count that the Minimum Support (2). It will give us the table for the **frequent itemset L1.** Since all the itemsets have greater or equal support count than the minimum support, except the E, so E itemset will be removed.

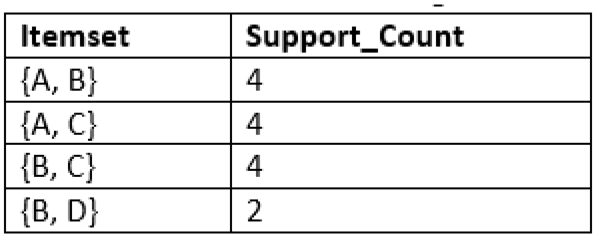
Itemset	Support_Count
A	6
B	7
C	5
D	2

Step-2: Candidate Generation C2, and L2:

- In this step, we will generate C2 with the help of L1. In C2, we will create the pair of the itemsets of L1 in the form of subsets.
- After creating the subsets, we will again find the support count from the main transaction table of datasets, i.e., how many times these pairs have occurred together in the given dataset. So, we will get the below table for C2:

Itemset	Support_Count
{A, B}	4
{A,C}	4
{A, D}	1
{B, C}	4
{B, D}	2
{C, D}	0

Again, we need to compare the C2 Support count with the minimum support count, and after comparing, the itemset with less support count will be eliminated from the table C2. It will give us the below table for L2

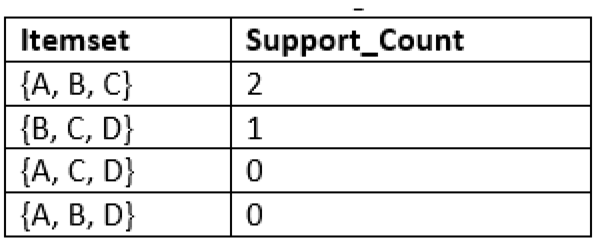


Itemset	Support_Count
{A, B}	4
{A, C}	4
{B, C}	4
{B, D}	2

A, B, C, D

Step-3: Candidate generation C3, and L3:

- For C3, we will repeat the same two processes, but now we will form the C3 table with subsets of three itemsets together, and will calculate the support count from the dataset. It will give the below table:



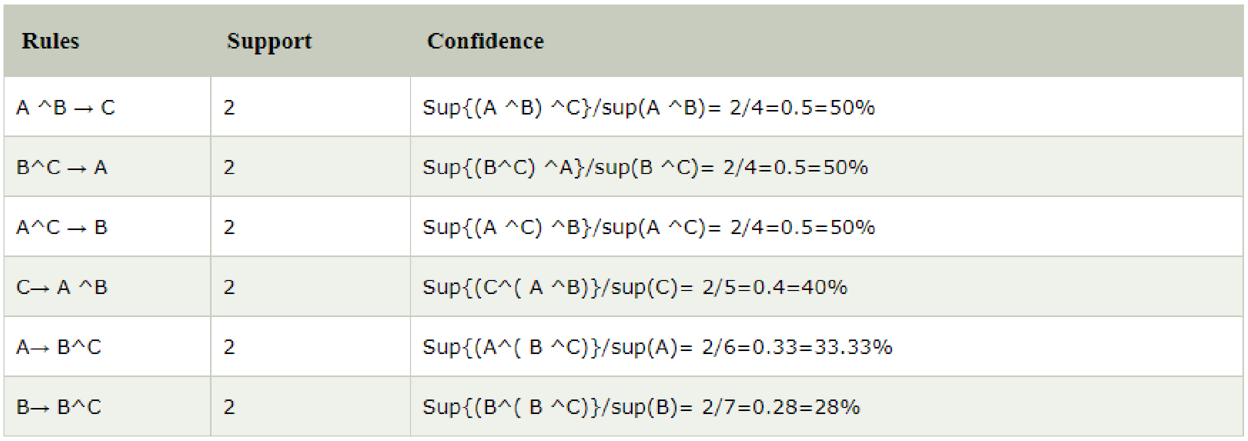
Itemset	Support_Count
{A, B, C}	2
{B, C, D}	1
{A, C, D}	0
{A, B, D}	0

- Now we will create the L3 table. As we can see from the above C3 table, there is only one combination of itemset that has support count equal to the minimum support count. So, the L3 will have only one combination, i.e., **{A, B, C}.**

Step-4: Finding the association rules for the subsets:

To generate the association rules, first, we will create a new table with the possible rules from the occurred combination {A, B.C}. For all the rules, we will calculate the Confidence using formula **sup(A ^B)/A.** After calculating the confidence value for all rules, we will exclude the rules that have less confidence than the minimum threshold (50%).

Consider the below table:



Rules	Support	Confidence
A ^B → C	2	Sup{(A ^B) ^C}/sup(A ^B)= 2/4=0.5=50%
B^C → A	2	Sup{(B^C) ^A}/sup(B ^C)= 2/4=0.5=50%
A^C → B	2	Sup{(A ^C) ^B}/sup(A ^C)= 2/4=0.5=50%
C→ A ^B	2	Sup{(C^(A ^B)}/sup(C)= 2/5=0.4=40%
A→ B^C	2	Sup{(A^(B ^C)}/sup(A)= 2/6=0.33=33.33%
B→ B^C	2	Sup{(B^(B ^C)}/sup(B)= 2/7=0.28=28%

As the given threshold or minimum confidence is 50%, so the first three rules **A ^B → C, B^C → A, and A^C → B** can be considered as the strong association rules for the given problem.

Advantages of Apriori Algorithm

- This is easy to understand algorithm
- The join and prune steps of the algorithm can be easily implemented on large datasets.

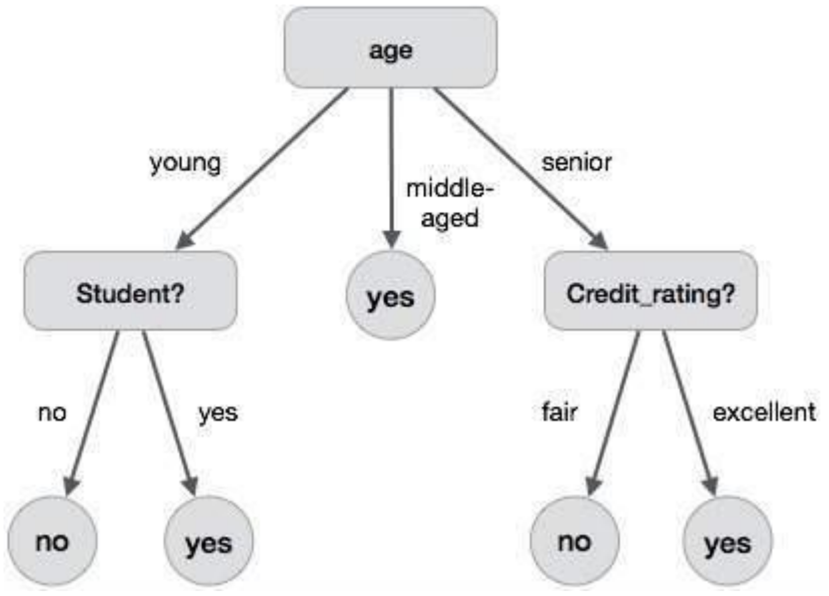
Disadvantages of Apriori Algorithm

- The apriori algorithm works slow compared to other algorithms.
- The overall performance can be reduced as it scans the database for multiple times.
- The time complexity and space complexity of the apriori algorithm is O(2D), which is very high. Here D represents the horizontal width present in the database.

Decision Tree Induction

A decision tree is a structure that includes a root node, branches, and leaf nodes. Each internal node denotes a test on an attribute, each branch denotes the outcome of a test, and each leaf node holds a class label. The topmost node in the tree is the root node.

The following decision tree is for the concept buy_computer that indicates whether a customer at a company is likely to buy a computer or not. Each internal node represents a test on an attribute. Each leaf node represents a class.



The benefits of having a decision tree are as follows

- It does not require any domain knowledge.
- It is easy to comprehend.
- The learning and classification steps of a decision tree are simple and fast.

Decision Tree Induction Algorithm

In this algorithm, there is no backtracking; the trees are constructed in a top-down recursive divide-and-conquer manner.

Generating a decision tree form training tuples of data partition D

Algorithm : Generate_decision_tree

Input:
Data partition, D, which is a set of training tuples and their associated class labels. attribute_list, the set of candidate attributes. Attribute selection method, a procedure to determine the splitting criterion that best partitions that the data tuples into individual classes. This criterion includes a

splitting_attribute and either a splitting point or splitting subset.

Output:
A Decision Tree
Method:

```
create a node N;

if tuples in D are all of the same class, C then
   return N as leaf node labeled with class C;

if attribute_list is empty then
   return N as leaf node with labeled
   with majority class in D;|| majority voting

apply attribute_selection_method(D, attribute_list)
to find the best splitting_criterion;
label node N with splitting_criterion;

if splitting_attribute is discrete-valued and
   multiway splits allowed then  // no restricted to binary trees

attribute_list = splitting attribute; // remove splitting attribute
for each outcome j of splitting criterion

   // partition the tuples and grow subtrees for each partition
   let Dj be the set of data tuples in D satisfying outcome j; // a partition

   if Dj is empty then
      attach a leaf labeled with the majority
      class in D to node N;
   else
      attach the node returned by Generate
      decision tree(Dj, attribute list) to node N;
   end for
return N;
```

Tree Pruning

Tree pruning is performed in order to remove anomalies in the training data due to noise or outliers. The pruned trees are smaller and less complex.

Tree Pruning Approaches

There are two approaches to prune a tree −

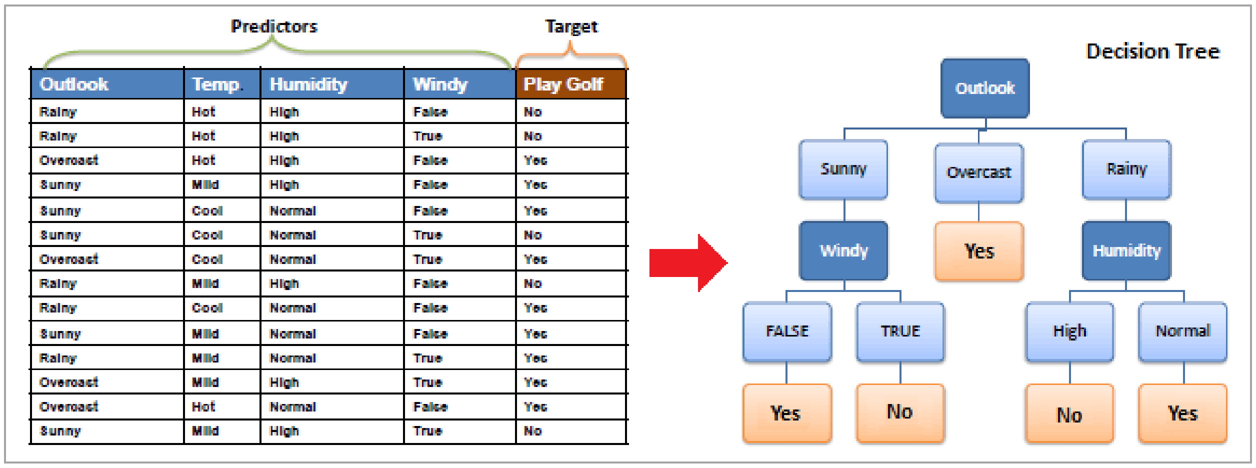
- Pre-pruning − The tree is pruned by halting its construction early.

- Post-pruning - This approach removes a sub-tree from a fully grown tree.

Cost Complexity

The cost complexity is measured by the following two parameters –

- Number of leaves in the tree, and
- Error rate of the tree.



Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

Bayesian classification

Bayesian classification is based on Bayes' Theorem. Bayesian classifiers are the statistical classifiers. Bayesian classifiers can predict class membership probabilities such as the probability that a given tuple belongs to a particular class.

Baye's Theorem

Bayes' Theorem is named after Thomas Bayes. There are two types of probabilities −

- Posterior Probability [P(H/X)]
- Prior Probability [P(H)]

where X is data tuple and H is some hypothesis.

According to Bayes' Theorem,

$$P(H/X)= P(X/H)P(H) / P(X)$$

Bayesian Belief Network

Bayesian Belief Networks specify joint conditional probability distributions. They are also known as Belief Networks, Bayesian Networks, or Probabilistic Networks.

- A Belief Network allows class conditional independencies to be defined between subsets of variables.
- It provides a graphical model of causal relationship on which learning can be performed.
- We can use a trained Bayesian Network for classification.

There are two components that define a Bayesian Belief Network −

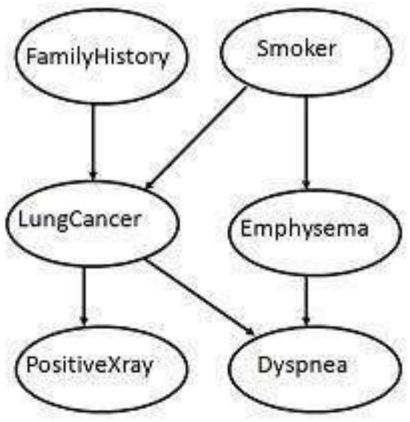
- Directed acyclic graph
- A set of conditional probability tables

Directed Acyclic Graph

- Each node in a directed acyclic graph represents a random variable.
- These variable may be discrete or continuous valued.
- These variables may correspond to the actual attribute given in the data.

Directed Acyclic Graph Representation

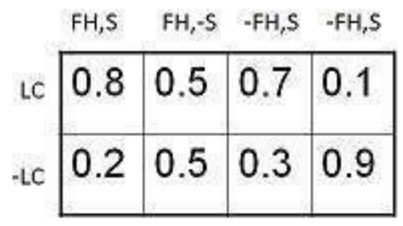
The following diagram shows a directed acyclic graph for six Boolean variables.



The arc in the diagram allows representation of causal knowledge. For example, lung cancer is influenced by a person's family history of lung cancer, as well as whether or not the person is a smoker. It is worth noting that the variable Positive Xray is independent of whether the patient has a family history of lung cancer or that the patient is a smoker, given that we know the patient has lung cancer.

Conditional Probability Table

The conditional probability table for the values of the variable Lung Cancer (LC) showing each possible combination of the values of its parent nodes, Family History (FH), and Smoker (S) is as follows −



	FH,S	FH,-S	-FH,S	-FH,S
LC	0.8	0.5	0.7	0.1
-LC	0.2	0.5	0.3	0.9

We have a dataset of weather conditions and corresponding target variable "Play". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

Problem: If the weather is sunny, then the Player should play or not?

Solution: To solve this, first consider the below dataset:

	Outlook	Play
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No
12	Overcast	Yes
13	Overcast	Yes

Frequency table for the Weather Conditions:

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	5

Likelihood table weather condition:

Weather	No	Yes	
Overcast	0	5	5/14= 0.35
Rainy	2	2	4/14=0.29
Sunny	2	3	5/14=0.35
All	4/14=0.29	10/14=0.71	

Applying Bayes'theorem:

P(Yes|Sunny)= P(Sunny|Yes)*P(Yes)/P(Sunny)

P(Sunny|Yes)= 3/10= 0.3

P(Sunny)= 0.35

P(Yes)=0.71

So P(Yes|Sunny) = 0.3*0.71/0.35= **0.60**

P(No|Sunny)= P(Sunny|No)*P(No)/P(Sunny)

P(Sunny|NO)= 2/4=0.5

P(No)= 0.29

P(Sunny)= 0.35

So P(No|Sunny)= 0.5*0.29/0.35 = **0.41**

So as we can see from the above calculation that **P(Yes|Sunny)>P(No|Sunny)**

Hence on a Sunny day, Player can play the game.

Advantages of Naïve Bayes Classifier:

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for **text classification problems**.

Disadvantages of Naïve Bayes Classifier:

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

Applications of Naïve Bayes Classifier:

- It is used for **Credit Scoring**.
- It is used in **medical data classification**.
- It can be used in **real-time predictions** because Naïve Bayes Classifier is an eager learner.
- It is used in Text classification such as **Spam filtering** and **Sentiment analysis**.

Rule based Classification

IF-THEN Rules

Rule-based classifier makes use of a set of IF-THEN rules for classification. We can express a rule in the following from −

IF condition THEN conclusion

Let us consider a rule R1,

```
R1: IF age = youth AND student = yes
  THEN buy_computer = yes
```

Points to remember −

- The IF part of the rule is called rule antecedent or precondition.
- The THEN part of the rule is called rule consequent.
- The antecedent part the condition consist of one or more attribute tests and these tests are logically ANDed.
- The consequent part consists of class prediction.

Note − We can also write rule R1 as follows −

R1: (age = youth) ^ (student = yes))(buys computer = yes)

If the condition holds true for a given tuple, then the antecedent is satisfied.

Rule Extraction

Here we will learn how to build a rule-based classifier by extracting IF-THEN rules from a decision tree.

Points to remember −

To extract a rule from a decision tree −

- One rule is created for each path from the root to the leaf node.
- To form a rule antecedent, each splitting criterion is logically ANDed.
- The leaf node holds the class prediction, forming the rule consequent.

Rule Induction Using Sequential Covering Algorithm

Sequential Covering Algorithm can be used to extract IF-THEN rules form the training data. We do not require to generate a decision tree first. In this algorithm, each rule for a given class covers many of the tuples of that class.

Some of the sequential Covering Algorithms are AQ, CN2, and RIPPER. As per the general strategy the rules are learned one at a time. For each time rules are learned, a tuple covered by the rule is removed and the process continues for the rest of the tuples. This is because the path to each leaf in a decision tree corresponds to a rule.

Note − The Decision tree induction can be considered as learning a set of rules simultaneously.

The Following is the sequential learning Algorithm where rules are learned for one class at a time. When learning a rule from a class Ci, we want the rule to cover all the tuples from class C only and no tuple form any other class.

Algorithm: Sequential Covering

Input:
D, a data set class-labeled tuples,
Att_vals, the set of all attributes and their possible values.

Output: A Set of IF-THEN rules.

Method:

```
Rule_set={ }; // initial set of rules learned is empty

for each class c do

  repeat
    Rule = Learn_One_Rule(D, Att_valls, c);
    remove tuples covered by Rule form D;
  until termination condition;

  Rule_set=Rule_set+Rule; // add a new rule to rule-set
end for
return Rule_Set;
```

Rule Pruning

The rule is pruned is due to the following reason −

- The Assessment of quality is made on the original set of training data. The rule may perform well on training data but less well on subsequent data. That's why the rule pruning is required.

- The rule is pruned by removing conjunct. The rule R is pruned, if pruned version of R has greater quality than what was assessed on an independent set of tuples.

FOIL is one of the simple and effective method for rule pruning. For a given rule R,

FOIL_Prune = pos - neg / pos + neg

where pos and neg is the number of positive tuples covered by R, respectively.

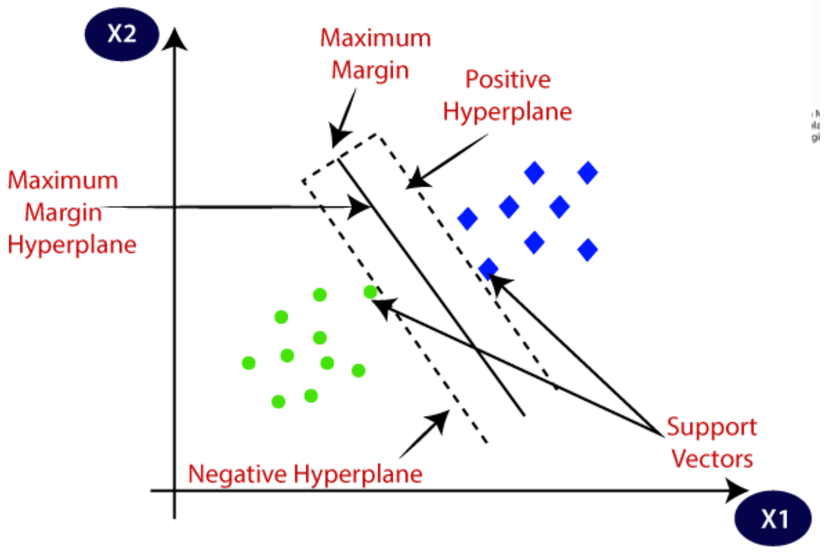
Note − This value will increase with the accuracy of R on the pruning set. Hence, if the FOIL_Prune value is higher for the pruned version of R, then we prune R.

Support Vector Machine Algorithm

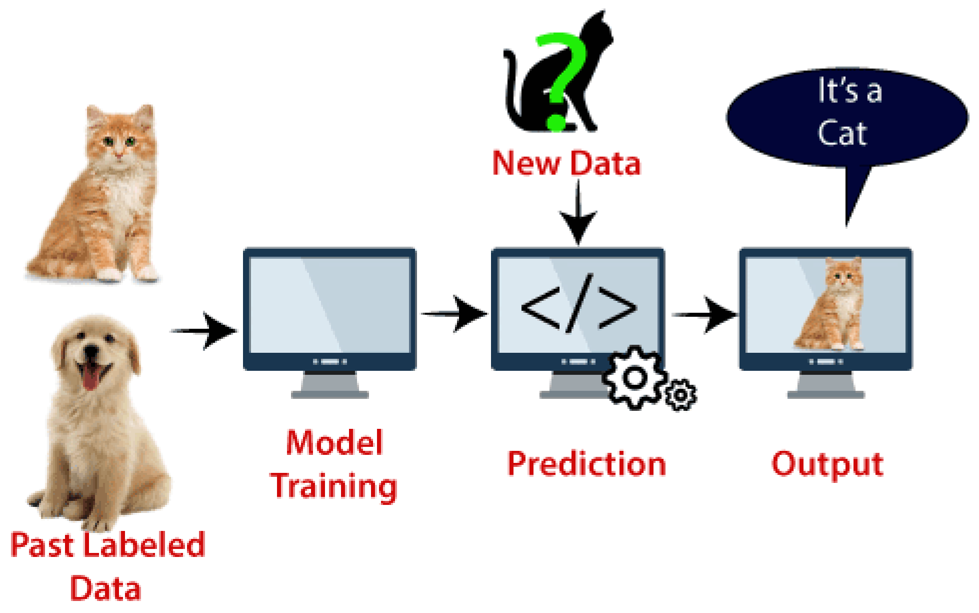
Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



Example: SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:



SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

Types of SVM

SVM can be of two types:

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

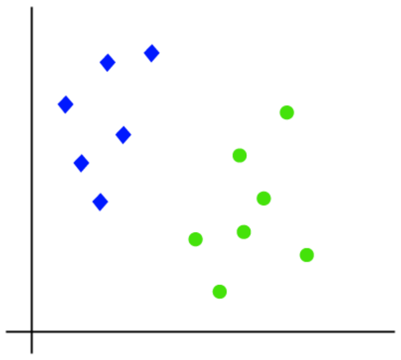
Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

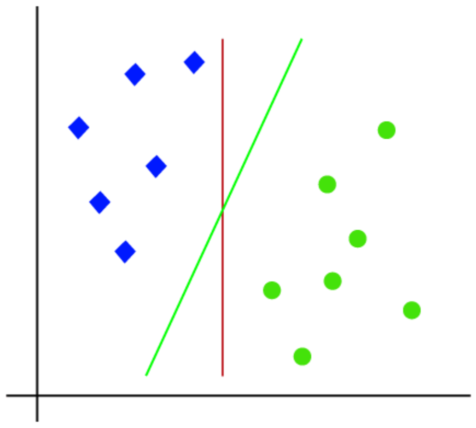
How does SVM works?

Linear SVM:

The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:

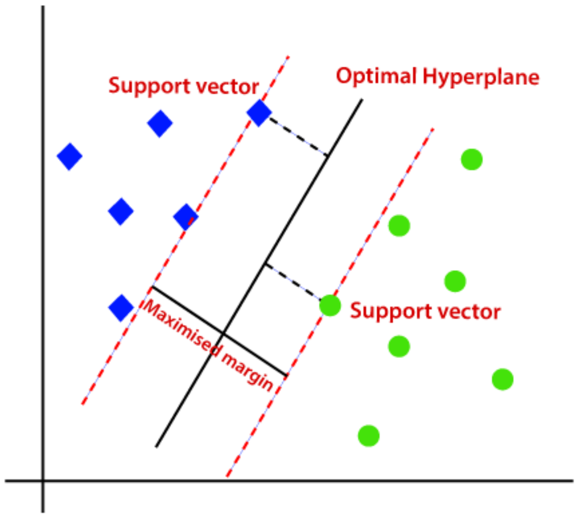


So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:



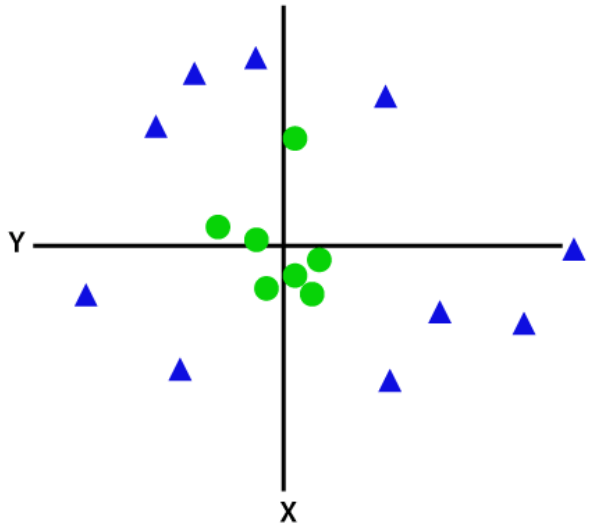
Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both

the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin.
The **hyperplane** with maximum margin is called the **optimal hyperplane**.



Non-Linear SVM:

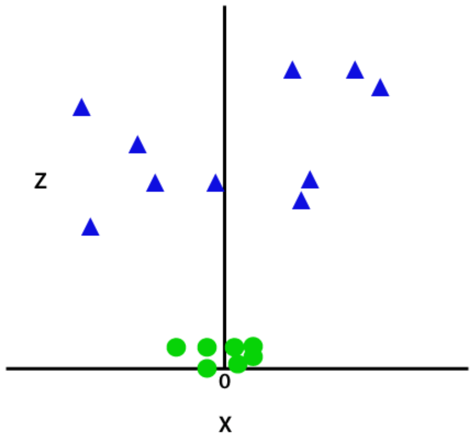
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



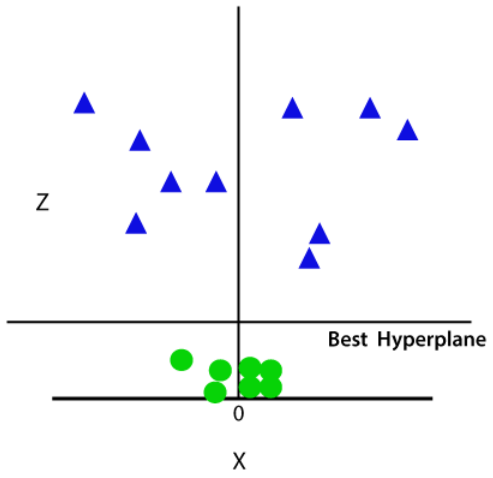
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

z=x2 +y2

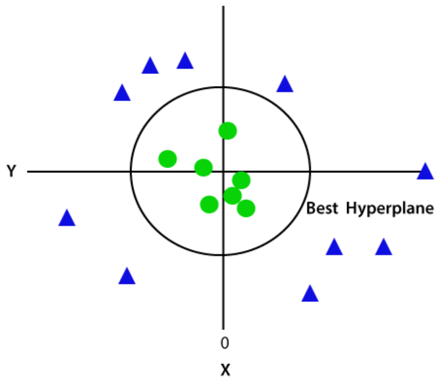
By adding the third dimension, the sample space will become as below image:



So now, SVM will divide the datasets into classes in the following way. Consider the below image:



Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as:



Hence we get a circumference of radius 1 in case of non-linear data.

Pros of SVM classifiers

SVM classifiers offers great accuracy and work well with high dimensional space. SVM classifiers basically use a subset of training points hence in result uses very less memory.

Cons of SVM classifiers

They have high training time hence in practice not suitable for large datasets. Another disadvantage is that SVM classifiers do not work well with overlapping classes.