DATA MINING

MODULE 3

Classification

There are two forms of data analysis that can be used for extracting models describing important classes or to predict future data trends. These two forms are as follows −

- Classification
- Prediction

Classification: It is a Data analysis task, i.e., the process of finding a model that describes and distinguishes data classes and concepts. Classification models predict categorical class labels; and prediction models predict continuous valued functions.Classification is the problem of identifying to which of a set of categories (subpopulations), a new observation belongs to, on the basis of a training set of data containing observations and whose category membership is known.

Following are the examples of cases where the data analysis task is Classification:

- A bank loan officer wants to analyze the data in order to know which customers (loan applicants) are risky or which are safe.
- A marketing manager at a company needs to analyze a customer with a given profile, who will buy a new computer.

In both of the above examples, a model or classifier is constructed to predict the categorical labels. These labels are risky or safe for loan application data and yes or no for marketing data.

In each of these examples, the data analysis task is classification, where a model or classifier is constructed to predict class (categorical) labels, such as “safe” or “risky” for the loan application data; “yes” or “no” for the marketing data. These categories can be represented by discrete values, where the ordering among values has no meaning.

Prediction

- Suppose that the marketing manager wants to predict how much a given customer will spend during a sale at AllElectronics.
- This data analysis task is an example of numeric prediction, where the model constructed predicts a continuous-valued function, or ordered value, as opposed to a class label.
- This model is a predictor. Regression analysis is a statistical methodology that is most often used for numeric prediction; hence the two terms tend to be used synonymously, although other methods for numeric prediction exist.
- Classification and numeric prediction are the two major types of prediction

problems.

Supervised and unsupervised learning

- Supervised learning, as the name indicates, has the presence of a supervisor as a teacher.
- Basically supervised learning is when we teach or train the machine using data that is well labeled.
- Which means some data is already tagged with the correct answer.
- After that, the machine is provided with a new set of examples(data) so that the supervised learning algorithm analyzes the training data(set of training examples) and produces a correct outcome from labeled data.

For instance, suppose you are given a basket filled with different kinds of fruits. Now the first step is to train the machine with all different fruits one by one like this:

- If the shape of the object is rounded and has a depression at the top,is red in color, then it will be labeled as –**Apple**.
- If the shape of the object is a long curving cylinder having Green-Yellow color, then it will be labeled as –**Banana**.

➔ Now suppose after training the data, you have given a new separate fruit, say Banana from the basket, and asked to identify it.
➔ Since the machine has already learned the things from previous data and this time has to use it wisely.
➔ It will first classify the fruit with its shape and color and would confirm the fruit name as BANANA and put it in the Banana category.
➔ Thus the machine learns the things from training data(basket containing fruits) and then applies the knowledge to test data(new fruit).

Supervised learning classified into two categories of algorithms:

- **Classification**: A classification problem is when the output variable is a category, such as “Red” or “blue” or “disease” and “no disease”. **Example**:

  Regression Example : Suppose from the data you come to know that your best friend likes some of the movies. You also know how many times each particular movie is seen by your friend. Now one new movie (testdata) has been released. Now you are going to find how many times this newly released movie will be watched by your friend. It could be 5 times, 6 times,10 times so here we will use Regression.
- **Regression**: A regression problem is when the output variable is a real value, such as “dollars” or “weight”.
  Example:Suppose from the train data you come to know that your best friend

likes some of the movies. Now one new movie (test data) has been released. Now you want to know whether your best friend likes it or not. Here the output which you are expecting will either be Yes or No so we will use Classification.

Supervised learning deals with or learns with “labeled” data. This implies that some data is already tagged with the correct answer.

Types :-

- **Regression**
  - Linear Regression
  - Logistic Regression
- **Classification**
  - Naive Bayes Classifiers
  - K-NN (k- nearest neighbors)
  - Decision Trees
  - Support Vector Machine

Advantages:

- Supervised learning allows collecting data and produces data output from previous experiences.
- Helps to optimize performance criteria with the help of experience.
- Supervised machine learning helps to solve various types of real-world computation problems.

Disadvantages:

- Classifying big data can be challenging.
- Training for supervised learning needs a lot of computation time. So, it requires a lot of time.

Unsupervised Learning

- Unsupervised learning is the training of a machine using information that is neither classified nor labeled and allowing the algorithm to act on that

information without guidance. Here the task of the machine is to group unsorted information according to similarities, patterns, and differences without any prior training of data.

- Unlike supervised learning, no teacher is provided, which means no training will be given to the machine. Therefore the machine is restricted to find the hidden structure in unlabeled data by itself. **For instance**, suppose it is given an image having both dogs and cats which it has never seen.

Thus the machine has no idea about the features of dogs and cats so we can't categorize it as 'dogs and cats '. But it can categorize them according to their similarities, patterns, and differences, i.e., we can easily categorize the above picture into two parts. The first may contain all pics having **dogs** in it and the second part may contain all pics having **cats** in it. Here you didn't learn anything before, which means no training data or examples.
It allows the model to work on its own to discover patterns and information that was previously undetected. It mainly deals with unlabelled data.

Unsupervised learning is classified into two categories of algorithms:

- **Clustering**: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
- **Association**: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Types of Unsupervised Learning:-

Clustering

1. Exclusive(partitioning)
2. Agglomerative
3. Overlapping
4. Probabilistic

Clustering Types:-

1. Hierarchical clustering
2. K-means clustering
3. Principal Component Analysis
4. Singular Value Decomposition
5. Independent Component Analysis

Supervised vs Unsupervised MachineLearning

Parameters	Supervised	Unsupervised
Input Data	**Algorithms are trained using labeled data.**	**Algorithms are used against data that is not labeled**
Computational Complexity	**Simpler method**	**Computationally complex**
Accuracy	**Highly accurate**	**Less accurate**

Some examples of supervised learning applications include:

Classification

- In finance and banking for credit card fraud detection(fraud,notfraud).
- Email spam detection(spam,notspam).
- In the marketing area used for text sentiment analysis (happy,not happy).
- In Medicine, for predicting whether a patient has a particular disease or not.

Regression

- Predicting house /property price
- Predicting stock market price

Classification–a two step process

Step 1: Model Construction

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

Step 2: Model usage: for classifying future or unknown objects

- Estimate accuracy of the model
- The known label of test sample is compared with the classified results 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

➔ 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 a database attribute, respectively , A1, A2,..., An.

➔ 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. Learning: mapping or function y=f(X)that can predict y of a given tuple X.

➔ The individual tuples making up the training set are referred to as **training tuples** and are randomly sampled from the database under analysis. In the context of classification, data tuples can be referred to as samples, examples, instances, data points, or objects.

➔ What about classification accuracy?" In the second step, the model is used for **classification**. 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 dataset 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. The associated class label of each test tuple is compared with the learned classifier's class prediction for that tuple.

Decision Tree Induction

- **Decision tree induction** is the learning of decision trees from class-labeled training tuples.
- A decision tree is a flowchart-like tree structure, where each **internal node**(non leaf node) denotes a test on an attribute, each **branch** represents an outcome of the test, and each leaf node**(**or terminal node) holds a class label.
- The topmost node in a tree is the root node.
- Internal nodes are denoted by rectangles, and leaf nodes are denoted by ovals.
- Some decision tree algorithms produce only binary trees (where each internal node branches to exactly two other nodes), whereas others can produce nonbinary trees.

How are decision trees used for classification?

- Given a tuple, X, for which the associated class label is unknown, the attribute values of the tuple are tested against the decision tree.
- A path is traced from the root to a leaf node, which holds the class prediction for that tuple.
- ***Decision trees can easily be converted to classification rules.***

Why are decision tree classifiers so popular?

- The construction of decision tree classifiers does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery.
- Decision trees can handle multidimensional data.
- Their representation of acquired knowledge in tree form is intuitive and generally easy to assimilate by humans.
- The learning and classification steps of decision tree induction are simple and fast.
- In general, decision tree classifiers have good accuracy. However, successful use may depend on the data at hand.
- Decision tree induction algorithms have been used for classification in many application areas such as medicine, manufacturing and production, financial analysis, astronomy, and molecular biology. Decision trees are the basis of several commercial rule induction systems.

Decision tree generation consists of two phases: Tree construction and Tree pruning

During tree construction, attribute selection measures are used to select the attribute that best partitions the tuples into distinct classes. When decision trees are built,many of the branches may reflect noise or outliers in the training data.

Tree pruning attempts to identify and remove such branches, with the goal of improving classification accuracy on unseen data.

Use of decision tree:Classifying an unknown sample

- Test the attribute values of the sample against the decision tree

Different strategies to Build decision tree

- Iterative Dichotomiser (ID3)→ (extension D3)
- C4.5→(successor of ID3)
- CART→(Classification and Regression Tree)
- CHAID → (Chi-square automatic interaction detection Performs multi-level splits when computing classification trees)
- MARS→(multivariate adaptive regression splines)

  - All of them adopt a greedy approach
  - Top-down recursive

ID3, C4.5, and CART adopt a greedy (i.e., non backtracking) approach in which decision trees are constructed in a top-down recursive divide-and-conquer manner. Most algorithms for decision tree induction also follow a top-down approach, which starts with a training set of tuples and their associated class labels. The training set is recursively partitioned into smaller subsets as the tree is being built.

Algorithm for Decision Tree Induction

- Basic algorithm (a greedy algorithm):
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - 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)

- Conditions for stopping partitioning
  - All samples for a given node belongs to the same class
  - There are no remaining attributes for further partitioning–majority voting is employed for classifying the leaf
  - There are no samples left

Attribute Selection Measure

- An attribute selection measure is a heuristic for selecting the splitting criterion that "best" separates a given data partition, D, of class-labeled training tuples into individual classes.
- If we were to split D into smaller partitions according to the outcomes of the splitting criterion, ideally each partition would be pure (i.e., all the tuples that fall into a given partition would belong to the same class).
- Conceptually, the "best" splitting criterion is the one that most closely results in such a scenario. Attribute selection measures are also known as splitting rules because they determine how the tuples at a given node are to be split.
- The attribute selection measure provides a ranking for each attribute describing the given training tuples. The attribute having the best score for the measure is chosen as the splitting attribute for the given tuples.
- If the splitting attribute is continuous- valued or if we are restricted to binary trees, then, respectively, either a split point or a splitting subset must also be determined as part of the splitting criterion.
- The tree node created for partition D is labeled with the splitting criterion, branches are grown for each outcome of the criterion, and the tuples are partitioned accordingly.
- **There are three popular attribute selection measures— Information Gain, Gain ratio, and Gini index.**

Information Gain

- ID3 uses information gain as its attribute selection measure.
- This measure is based on pioneering work by Claude Shannon on information theory, which studied the value of "information content" of messages.
- Let node N represent or hold tuples of partition D.
- **The attribute with the highest information gain is chosen as the splitting**

attribute for node N.

- This attribute minimizes the information needed to classify the tuples in the resulting partitions and reflects the least randomness or "impurity" in these partitions.
- Such an approach minimizes the expected number of tests needed to classify a given tuple and guarantees that a simple (but not necessarily the simplest) tree is found.
- The expected information needed to classify a tuple in D is given by $\mathbf{Info(D) = -\sum pi \log2 (pi)}$
- Where pi is the nonzero probability that an arbitrary tuple in D belongs to class Ci.
- information is encoded in bits.
- Info(D) is just the average amount of information needed to identify the class label of a tuple in D. Note that, at this point, the information we have is based solely on the proportions of tuples of each class. Info(D) is also known as the entropy of D.
- Information gain is defined as the difference between the original information requirement (i.e., based on just the proportion of classes) and the new requirement(i.e., obtained after partitioning on A). That is,

$$\mathbf{Gain(A)=Info(D)-InfoA(D)}$$

Information Gain: Steps

The information gain is based on the decrease in entropy after a dataset is split on an attribute. Constructing a decision tree is all about finding attributes that return the highest information gain (i.e., the most homogeneous branches).

Step1: Calculate entropy of the target.

Step 2: The dataset is then split on the different attributes. The entropy for each branch is calculated. Then it is added proportionally, to get total entropy for the split. The resulting entropy is subtracted from the entropy before the split. The result is the Information Gain, or decrease in entropy.

Step 3: Choose the attribute with the largest information gain as the decision node,divide the dataset by its branches and repeat the same process on every branch.

Step 4a: A branch with entropy of 0 is a leaf node.

Step 4b: A branch with entropy more than 0 needs further splitting.

Step 5: The ID3 algorithm is run recursively on the non-leaf branches, until all data is classified.

Gain Ratio

- The information gain measure is biased toward tests with many outcomes. That is, it prefers to select attributes having a large number of values. For example, consider an attribute that acts as a unique identifier such as productID. A split on productID would result in a large number of partitions (as many as there are values),each one containing just one tuple. Because each partition is pure, the information required to classify data set D based on this partitioning would be Info product_ID(D) = 0. Therefore, the information gained by partitioning on this attribute is maximal. Clearly, such a partitioning is useless for classification.

C4.5, a successor of ID3, uses an extension to information gain known as gain ratio, which attempts to overcome this bias. It applies a kind of normalization to information gain using a "split information" value defined analogously with Info(D)as

$$\text{SplitInfo}_A(D) = -\sum |D_j|/|D| \times \log_2(|D_j|/|D|)$$

This value represents the potential information generated by splitting the training dataset, D, into v partitions, corresponding to the v outcomes of a test on attribute A. Note that, for each outcome, it considers the number of tuples having that outcome with respect to the total number of tuples in D.

It differs from information gain, which measures the information with respect to classification that is acquired based on the same partitioning.

The gain ratio is defined as :

$$\text{Gain Ratio}(A) = \text{Gain}(A)/\text{SplitInfo}_A(D).$$

The attribute with the maximum gain ratio is selected as the splitting attribute.

Gini Index

- The Gini index is used in CART.
- The Gini index measures the impurity of D, a data partition or set of training tuples, as:

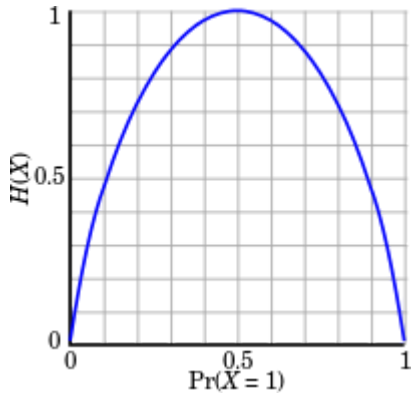
$$\textbf{Gini}(D) = 1 - \sum p^2$$

- Where p is the probability that a tuple in D belongs to class Ci and is estimated by |Ci, D|/|D|.

- The sum is computed over m classes.
- The algorithm chooses the attribute / feature with the least Gini index as the root node.

Entropy

- Shannon's Entropy
  - Entropy is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information.
  - Flipping a coin is an example of an action that provides information that is random.
  - It is quite evident that the entropy H(X) is zero when the probability is either 0 or 1. The Entropy is maximum when the probability is 0.5 because it projects perfect randomness in the data and there is no chance of perfectly determining the outcome.
- ID3 follows the rule— A branch with an entropy of zero is a leaf node and A branch with entropy more than zero needs further splitting.



Shannon's'entropy:Information Theory

- Mathematically Entropy for 1 attribute is represented as:
- Given a discrete random variable X with possible outcomes {x1,x2,….xn}, which occur with probability {p(x1,),p(x2).….p(xn)}is formally defined as:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_b P(x_i)$$

- In information theory, the entropy of a random variable is the average level of "information", "surprise", or "uncertainty" inherent in the variable's possible outcomes.

Avoid Overfitting in Classification

- Overfitting occurs when a model learns the training data too well, capturing noise and random fluctuations in the data instead of the underlying patterns. As a result, an overfitted model performs well on the training set but fails to generalize to new, unseen data.

- The generated tree may overfit the training data – Too many branches, some may reflect anomalies due to noise or outliers – Result is in poor accuracy for unseen samples.

▪ **Two approaches to avoid overfitting:**

– Pre- Pruning: 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.

– Post- Pruning: Remove branches from a "fully grown" tree—get a sequence of progressively pruned trees ▪ Use a set of data different from the training data to decide which is the "best pruned tree".

Approaches to Determine the Final Tree Size

▪ **Separate training(2/3) and testing(1/3) sets.**

▪ **Use Cross Validation,e.g.,10-fold cross validation.**

– Ink-fold cross- validation, the original sample is randomly partitioned into k equal size subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model,and the remaining k-1 subsamples are used as training data. The cross-validation process is then repeated k times(the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged (or otherwise combined) to produce a single estimation. The advantage of this method is that all observations are used for both training and validation, and each observation is used for validation exactly once.

▪ **Use all the data for training**

– but apply a statistical test to estimate whether expanding or pruning a node may improve the entire distribution

▪ **Use minimum description length(MDL) principle:**

– halting growth of the tree when the encoding is minimized.

Tree Pruning

- When a decision tree is built, many of the branches will reflect anomalies in the training data due to noise or outliers.
- Tree Pruning methods address this problem of overfitting the data. Such methods typically use statistical measures to remove the least-reliable branches.
- Pruned trees tend to be smaller and less complex and, thus, easier to comprehend. They are usually faster and better at correctly classifying independent test data (i.e., of previously unseen tuples) than unpruned trees.
- There are two common approaches to tree pruning: **pre- pruning and post-pruning.**
- In the **pre pruning** approach, a tree is "pruned" by halting its construction early(e.g.,by deciding not to further split or partition the subset of training tuples at a given node).Upon halting, the node becomes a leaf. The leaf may hold the most frequent class among the subset tuples or the probability distribution of those tuples.
- When constructing a tree, measures such as statistical significance, information gain, Gini index, and so on, can be used to assess the goodness of a split. If partitioning the tuples at a node would result in a split that falls below a prespecified threshold, then further partitioning of the given subset is halted. There are difficulties, however, in choosing an appropriate threshold. High thresholds could result in oversimplified trees, whereas low thresholds could result in very little simplification.
- The second and more common approach is **post pruning**, which removes sub trees from a "fully grown" tree. A subtree at a given node is pruned by removing its branches and replacing it with a leaf. The leaf is labeled with the most frequent class among the subtree being replaced.
- A pruning set of class-labeled tuples is used to estimate cost complexity. This set is independent of the training set used to build the unpruned tree and of any test set used for accuracy estimation. The algorithm generates a set of progressively pruned trees. In general, the smallest decision tree that minimizes the cost complexity is preferred.

ID3 Algorithm

Steps in ID3 algorithm

The ID3 algorithm builds decision trees using a top-down greedy search approach through the space of possible branches with no backtracking. A greedy algorithm,as the name suggests, always makes the choice that seems to be the best at that moment.

- It begins with the original set S as the root node.
- On each iteration of the algorithm, it iterates through the very unused attribute of the set S and calculates Entropy(H) and Information gain(IG) of this attribute.
- It then selects the attribute which has the smallest Entropy or Largest Information gain.
- The set S is then split by the selected attribute to produce a subset of the data.
- The algorithm continues to recur on each subset, considering only attributes never selected before.

ID3 Algorithm : Example

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Step 1:

Entropy of entire Dataset , H(S) = - p(yes) * log2(p(yes)) - p(no) * log2(p(no))
= -9/14 log2(9/14) - 5/14 log2(5/14)
= 0.94

Step 2:

First Attribute - Outlook

Categorical values - sunny, overcast and rain

H(Outlook=sunny) = -(2/5)*log2(2/5)-(3/5)*log2(3/5) =0.971
H(Outlook=rain) = -(3/5)*log2(3/5)-(2/5)*log2(2/5) =0.971
H(Outlook=overcast) = -(4/4)*log2(4/4)-0 = 0

Average Entropy Information for Outlook -
I(Outlook) = p(sunny) * H(Outlook=sunny) + p(rain) * H(Outlook=rain) + p(overcast) * H(Outlook=overcast)
= (5/14)*0.971 + (5/14)*0.971 + (4/14)*0
= 0.693

Information Gain = H(S) - I(Outlook)
= 0.94 - 0.693
= 0.247

Step 3:

Second Attribute - Temperature

Categorical values - hot, mild, cool

H(Temperature=hot) = -(2/4)*log2(2/4)-(2/4)*log2(2/4) = 1
H(Temperature=cool) = -(3/4)*log2(3/4)-(1/4)*log2(1/4) = 0.811
H(Temperature=mild) = -(4/6)*log2(4/6)-(2/6)*log2(2/6) = 0.9179

Average Entropy Information for Temperature -
I(Temperature) = p(hot)*H(Temperature=hot) + p(mild)*H(Temperature=mild) + p(cool)*H(Temperature=cool)
= (4/14)*1 + (6/14)*0.9179 + (4/14)*0.811
= 0.9108

Information Gain = H(S) - I(Temperature)
= 0.94 - 0.9108

= 0.0292

Step 4:

Third Attribute - Humidity
Categorical values - high, normal

H(Humidity=high) = -(3/7)*log2(3/7)-(4/7)*log2(4/7) = 0.983
H(Humidity=normal) = -(6/7)*log2(6/7)-(1/7)*log2(1/7) = 0.591

Average Entropy Information for Humidity -
I(Humidity) = p(high)*H(Humidity=high) + p(normal)*H(Humidity=normal)
= (7/14)*0.983 + (7/14)*0.591
= 0.787

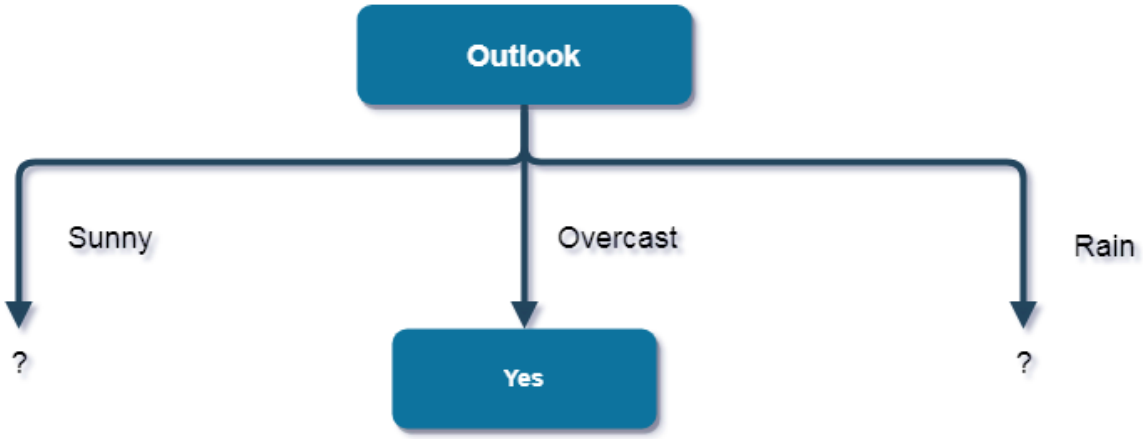
Information Gain = H(S) - I(Humidity)
= 0.94 - 0.787
= 0.153

Step 5:

Fourth Attribute - Wind
Categorical values - weak, strong
H(Wind=weak) = -(6/8)*log2(6/8)-(2/8)*log2(2/8) = 0.811
H(Wind=strong) = -(3/6)*log2(3/6)-(3/6)*log2(3/6) = 1

Average Entropy Information for Wind -
I(Wind) = p(weak)*H(Wind=weak) + p(strong)*H(Wind=strong)
= (8/14)*0.811 + (6/14)*1
= 0.892

Information Gain = H(S) - I(Wind)
= 0.94 - 0.892
= 0.048



Step 6:

Finding the best attribute for splitting the data with Outlook=Sunny values{ Dataset rows = [1, 2, 8, 9, 11]}.

Complete entropy of Sunny is -

$$H(S) = - p(yes) * \log2(p(yes)) - p(no) * \log2(p(no))$$
$$= - (2/5) * \log2(2/5) - (3/5) * \log2(3/5)$$
$$= 0.971$$

Step 7:

First Attribute - Temperature
Categorical values - hot, mild, cool

H(Sunny, Temperature=hot) = -0(2/2)*log(2/2) = 0
H(Sunny, Temperature=cool) = -(1)*log(1)- 0 = 0
H(Sunny, Temperature=mild) = -(1/2)*log(1/2)-(1/2)*log(1/2) = 1

Average Entropy Information for Temperature -
I(Sunny, Temperature) = p(Sunny, hot)*H(Sunny, Temperature=hot) + p(Sunny, mild)*H(Sunny, Temperature=mild) + p(Sunny, cool)*H(Sunny, Temperature=cool)
= (2/5)*0 + (1/5)*0 + (2/5)*1

= 0.4

Information Gain = H(Sunny) - I(Sunny, Temperature)
= 0.971 - 0.4
= 0.571

Step 8:

Second Attribute - Humidity
Categorical values - high, normal

H(Sunny, Humidity=high) = - 0 - (3/3)*log(3/3) = 0
H(Sunny, Humidity=normal) = -(2/2)*log(2/2)-0 = 0

Average Entropy Information for Humidity -
I(Sunny, Humidity) = p(Sunny, high)*H(Sunny, Humidity=high) + p(Sunny, normal)*H(Sunny, Humidity=normal)
= (3/5)*0 + (2/5)*0
= 0

Information Gain = H(Sunny) - I(Sunny, Humidity)
= 0.971 - 0
= 0.971

Step 9:

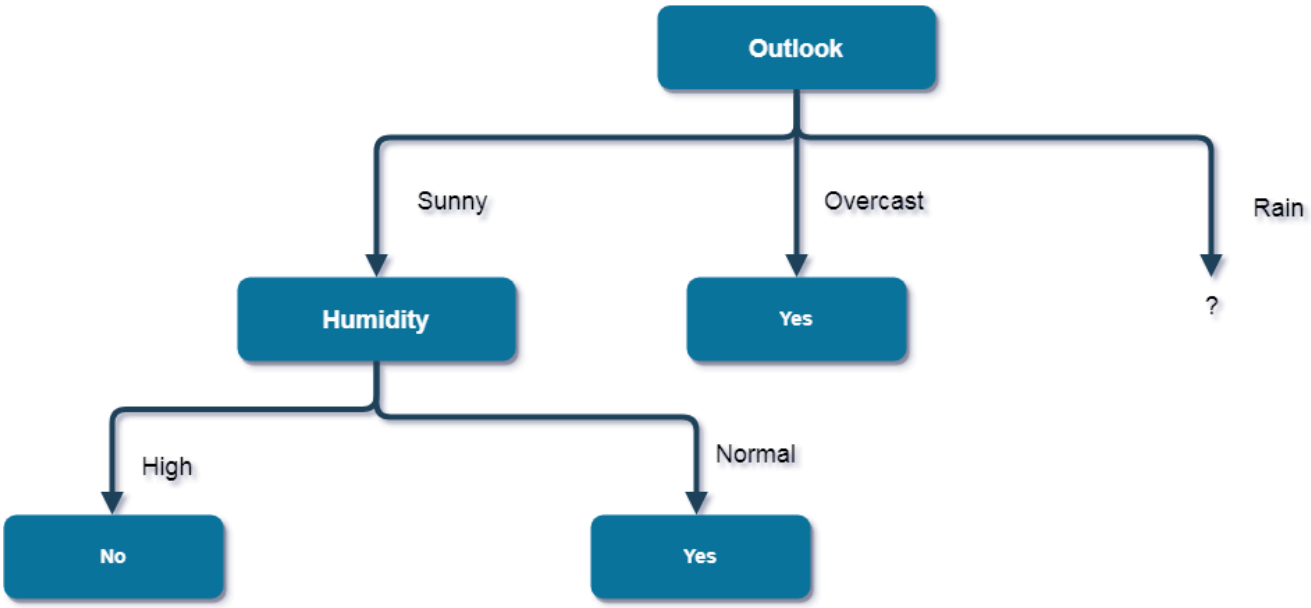
Third Attribute - Wind
Categorical values - weak, strong

H(Sunny, Wind=weak) = -(1/3)*log(1/3)-(2/3)*log(2/3) = 0.918
H(Sunny, Wind=strong) = -(1/2)*log(1/2)-(1/2)*log(1/2) = 1

Average Entropy Information for Wind -
I(Sunny, Wind) = p(Sunny, weak)*H(Sunny, Wind=weak) + p(Sunny,

strong)*H(Sunny, Wind=strong)
= (3/5)*0.918 + (2/5)*1
= 0.9508

Information Gain = H(Sunny) - I(Sunny, Wind)
= 0.971 - 0.9508
= 0.0202



Step 10:

Finding the best attribute for splitting the data with Outlook=Rainy values{ Dataset rows = [4, 5, 6, 10, 14]}.

Complete entropy of Rain is -
H(S) = - p(yes) * log2(p(yes)) - p(no) * log2(p(no))
= - (3/5) * log(3/5) - (2/5) * log(2/5)
= 0.971

First Attribute - Temperature
Categorical values - mild, cool

H(Rain, Temperature=cool) = -(1/2)*log(1/2)- (1/2)*log(1/2) = 1
H(Rain, Temperature=mild) = -(2/3)*log(2/3)-(1/3)*log(1/3) = 0.918
Average Entropy Information for Temperature -
I(Rain, Temperature) = p(Rain, mild)*H(Rain, Temperature=mild) + p(Rain, cool)*H(Rain, Temperature=cool)
= (2/5)*1 + (3/5)*0.918
= 0.9508

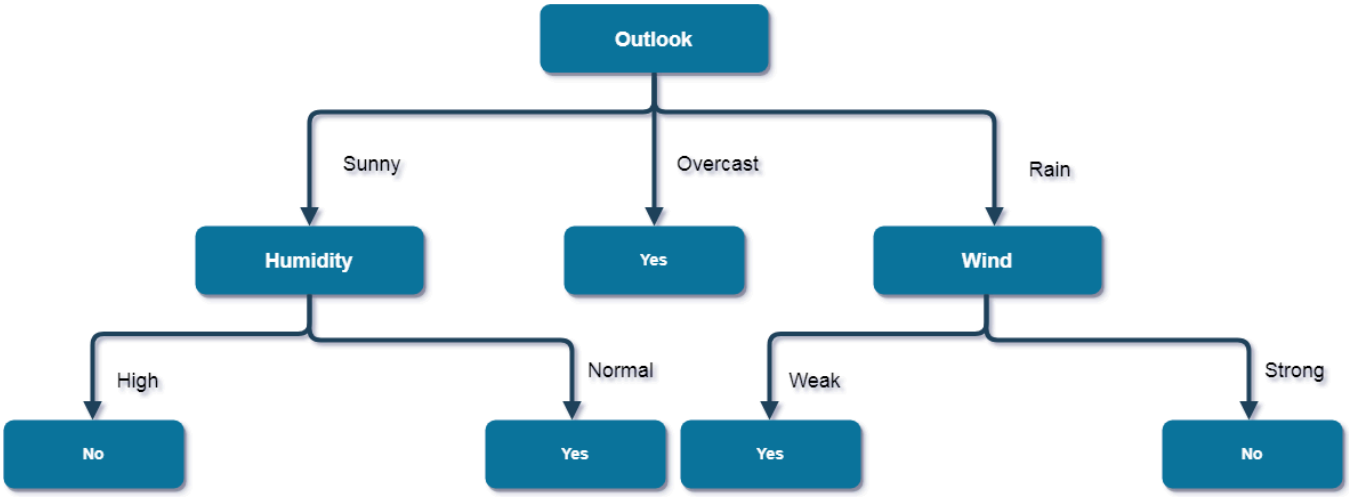
Information Gain = H(Rain) - I(Rain, Temperature)
= 0.971 - 0.9508
= 0.0202

Second Attribute - Wind
Categorical values - weak, strong
H(Wind=weak) = -(3/3)*log(3/3)-0 = 0
H(Wind=strong) = 0-(2/2)*log(2/2) = 0

Average Entropy Information for Wind -
I(Wind) = p(Rain, weak)*H(Rain, Wind=weak) + p(Rain, strong)*H(Rain, Wind=strong)
= (3/5)*0 + (2/5)*0
= 0

Information Gain = H(Rain) - I(Rain, Wind)
= 0.971 - 0
= 0.971

Here, the attribute with maximum information gain is Wind. So, the decision tree built so far -



Baye's Theorem

Bayes' theorem, named after 18th-century British mathematician Thomas Bayes, is a mathematical formula for determining conditional probability.

❖ Conditional probability is the likelihood of an outcome occurring, based on a previous outcome occurring.

❖ Bayes' theorem provides a way to revise existing predictions or theories (update probabilities) given new or additional evidence.

❖ In finance, Bayes' theorem can be used to rate the risk of lending money to potential borrowers.

Bayes' theorem relies on incorporating prior probability distributions in order to generate posterior probabilities.

❖ ***Prior probability***, in Bayesian statistical inference, is the probability of an event before new data is collected.

-This is the best rational assessment of the probability of an outcome based on the current knowledge before an experiment is performed.

❖ ***Posterior probability*** is the revised probability of an event occurring after taking into consideration new information.

-Posterior probability is calculated by updating the prior probability by using Bayes' theorem. In statistical terms, the posterior probability is the probability of event A occurring given that event B has occurred.

Let X be a data tuple. In Bayesian terms, X is considered "evidence". As usual, 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 hypothesis H holds given the "evidence" or observed data tuple X. In other words, we are looking for the probability that tuple X belongs to class C, given that we know the attribute description of X.

P(H|X) is the posterior probability, or a posteriori probability, of H conditioned on X.

P(H) is the prior probability, or a priori probability, of H.

P(X|H) is the posterior probability of X conditioned on H.

P(X) is the prior probability of X.

Bayes' theorem is useful in that it provides a way of calculating the posterior probability, P(H|X), from P(H), P(X|H), and P(X). Bayes' theorem is

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

Naive Bayes classifiers

❖ Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem.

❖ It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

Fundamental Naive Bayes assumption

❖ Is that each feature makes an:• independent •equal contribution to the outcome.

❖ We assume that no pair of features are dependent.

-For example, the temperature being 'Hot' has nothing to do with the humidity or the outlook being 'Rainy' has no effect on the winds. Hence, the features are assumed to be independent.

❖ Secondly, each feature is given the same weight(or importance).

-For example, knowing only temperature and humidity alone can't predict the outcome accurately. None of the attributes is irrelevant and assumed to be

contributing equally to the outcome.

❖ *Note: The assumptions made by Naive Bayes are not generally correct in real- world situations*

Example

❖ An internet search for "movie automatic shoelaces" brings up" Back to the future"

❖ Has the search engine watched movie? No, but it knows from lots of other searches what people are probably looking for.

❖ And it calculates that probability using Bayes' Theorem.

Naive Assumption

➢ Now, it's time to put a naive assumption to the Bayes' theorem, which is, independence among the features. So now, we split evidence into the independent parts.

➢ Now, if any two events A and B are independent, then, P(A,B) = P(A)P(B)

Hence:

Hence, we reach to the result:

$P(y|x1,...,xn)=\{P(x|y)P(x2|y)...P(xn|y)P(y)\}/\{P(x1)P(x2)...P(xn)\}$

which can be expressed as:

$P(y|x1,...,xn)=\{P(y)*\prod_{i=1ton}P(xi|y)\}/\{P(x1)P(x2)...P(xn)\}$

➢ Now, as the denominator remains constant for a given input, we can remove that term

$$P(y|x1,...,xn)\alpha P(y)\backslash\prod\{i=1\}^\{n\}P(xi|y)$$

Steps in NaïveBayes classification

❖ Convert the given dataset into frequency tables.

❖ Generate Likelihood table by finding the probability of given features.

❖ Now, use Bayes theorem to calculate the posterior probability for each class. The Class with the highest posterior probability is the outcome of prediction.

(See the Example in Notebook)

Pros:

- It is easy and fast to predict the class of a test dataset. It also performs well in multi class prediction.
- When assumption of independence holds, a Naive Bayes classifier performs better compared to other models like logistic regression and you need less training data.
- It performs well in case of categorical input variables compared to numerical variable(s). For numerical variables, normal distribution is assumed (bell curve, which is a strong assumption).

Cons:

- If a categorical variable has a category (in test dataset), which was not observed in the training data set, then the model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as "Zero Frequency". To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.
- On the other side naive Bayes is also known as a bad estimator, so the probability outputs from predict_proba are not to be taken too seriously.
- Another limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

Rule-Based Classifier

Rule-based classifiers are just another type of classifier which makes the class decision by using various "if..else" rules. These rules are easily interpretable and thus these classifiers are generally used to generate descriptive models.

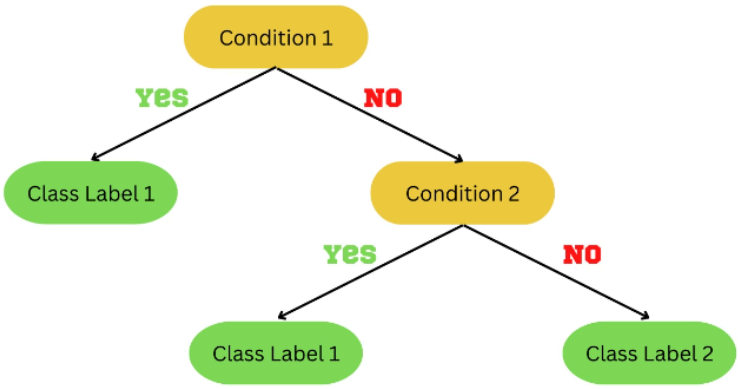
"IF condition THEN conclusion."

IF-THEN Rule

To define the IF-THEN rule, we can split it into two parts:

- **Rule Antecedent:** This is the "if condition" part of the rule. This part is present in the LHS (Left Hand Side). The antecedent can have one or more attributes as conditions, with logic AND operator.

- **Rule Consequent:** This is present in the rule's RHS (Right Hand Side). The rule consequent consists of the class prediction.



Example:

R1: IF climate= cool AND humidity = yes
THEN raining = true

Assessment of Rule

In rule-based classification in data mining, there are two factors based on which we can access the rules. These are:

- **Coverage of Rule:** **The fraction of the records which satisfy the antecedent conditions of a particular rule is called the coverage of that rule.**
  We can calculate this by dividing the number of records satisfying the rule(n1) by the total number of records(n).

  Coverage(R) = n1/n

- **Accuracy of a rule:** **The fraction of the records that satisfy the antecedent conditions and meet the consequent values of a rule is called the accuracy of that rule.**

  We can calculate this by dividing the number of records satisfying the

consequent values(n2) by the number of records satisfying the rule(n1).

Accuracy(R) = n2/n1

Example 1:

R1 : IF Status=Single THEN Class=No

Tid	Refund	Marital Status	Taxable Income	Class
1	Yes	**Single**	125K	**No**
2	No	Married	100K	**No**
3	No	**Single**	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	**Single**	85K	**Yes**
9	No	Married	75K	**No**
10	No	**Single**	90K	**Yes**

Coverage = 4/10 = 40%

Accuracy = 2/4 = 50%

Example 2 :

R1 : IF Age=Young and Income=Low THEN Purchase=No

Find the coverage and accuracy of the above rule.

Person	Age	Income	Purchase
1	Young	Low	No
2	Adult	High	Yes
3	Adult	Medium	No

Coverage = $\frac{1}{3}$ =0.333 = 33%

Accuracy =1/1= 1=100 %

Properties of Rule-Based Classifiers

There are two significant properties of rule-based classification in data mining. They are:

- **Rules may not be mutually exclusive**
- **Rules may not be exhaustive**

Rules may not be mutually exclusive in nature

- If a rule R1 is satisfied by a tuple X, then the rule is said to be **triggered.**
- If **R1 is the only rule satisfied**, then the rule **fires** by returning the class **prediction for X.**
- Triggering does not always mean firing because there may be more than one rule that is satisfied.
- Many different rules are generated for the dataset, so it is possible and likely that many of them satisfy the same data record. **This condition makes the rules not mutually exclusive.**
- Since the rules are not mutually exclusive, we cannot decide on classes that cover different parts of data on different rules.
- But this was our main objective. So, to solve this problem, we have two ways:
    - Size Ordering
    - Rule Ordering

Size Ordering

The size ordering scheme assigns the highest priority to the triggering rules that have the toughest requirements, where toughness is measured by the rule antecedent size. That is, the triggering rule with the most attribute tests is fired.

Rule Ordering

- The rule ordering scheme prioritizes the rules beforehand. The ordering may be **class-based** or **rule-based**.
- **With class-based ordering,** the classes are sorted in order of decreasing "importance" such as by decreasing order of prevalence. That is, all the rules for the most prevalent (or most frequent) class come first, the rules for the next prevalent class come next, and so on. Alternatively, they may be sorted based on

the misclassification cost per class. Within each class, the rules are not ordered—they don't have to be because they all predict the same class (and so there can be no class conflict!). **So, in this, the set of rules remains unordered.**

- **With rule-based ordering,** the rules are organized into one long priority list, according to some measure of rule quality, such as accuracy, coverage, size (number of attribute tests in the rule antecedent), or based on advice from domain experts. When rule ordering is used, the rule set is known as a **decision list**. With rule ordering, the triggering rule that appears earliest in the list has the highest priority, and so it gets to fire its class prediction. Any other rule that satisfies X is ignored. **Here it uses an ordered set of rules.**

Rules may not be exhaustive in nature

- It is not a guarantee that the rule will cover all the data entries.
- Any of the rules may leave some data entries.
- This case, on its occurrence, will make the rules non-exhaustive.
- So, we have to solve this problem too.
- So, to solve this problem, we can make use of a **default class**. Using a default class, we can assign all the data entries not covered by any rules to the default class.
- Thus using the default class will solve the problem of non-exhaustively.

Rule Extraction from a Decision Tree

- In comparison with a decision tree, the IF-THEN rules may be easier for humans to understand, particularly if the decision tree is very large.
- To extract rules from a decision tree, one rule is created for each path from the root to a leaf node.
- Each splitting criterion along a given path is logically ANDed to form the rule antecedent ("IF" part).
- The leaf node holds the class prediction, forming the rule consequent ("THEN" part).

Building a rule-based classifier by extracting IF-THEN rules from a decision tree.

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

Rules that are extracted directly from the tree can be mutually exclusive and exhaustive. By mutually exclusive, this means that we cannot have rule conflicts here because no two rules will be triggered for the same tuple. (We have one rule per leaf, and any tuple can map to only one leaf.) By exhaustive, there is one rule for each possible attribute-value combination, so that this set of rules does not require a default rule. Therefore, the order of the rules does not matter—they are unordered.

Rule Induction Using Sequential Covering Algorithm

- Sequential Covering Algorithm can be used to extract IF-THEN rules from the training data without constructing a decision tree.
- 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.
- Sequential covering algorithms are the most widely used approach to mining disjunctive sets of classification rules, and form the topic of this subsection. Note that in a newer alternative approach, classification rules can be generated using associative classification algorithms, which search for attribute-value pairs that occur frequently in the data. These pairs may form association rules, which can be analyzed and used in classification.

Algorithm: Sequential Covering

Input:

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

Output:

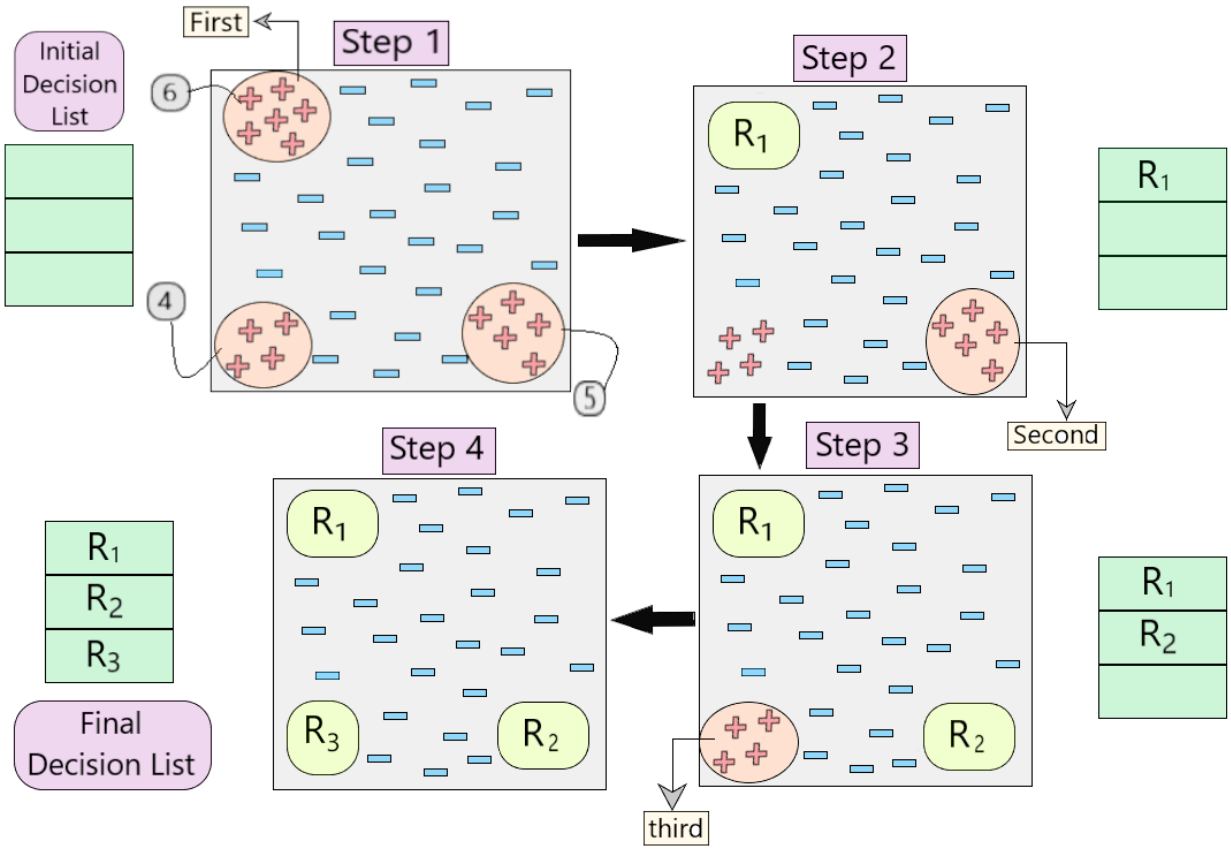
A set of IF-THEN rules.

Method:

(1) Rule_set = {}; // initial set of rules learned is empty
(2) for each class c do
(3) repeat
(4) Rule = Learn_One_Rule(D,Att_vals,c);
(5) remove tuples covered by Rule from D;
(6) Rule_set = Rule_set + Rule; // add new rule to rule set
(7) until terminating condition;
(8) endfor
(9) return Rule_Set

- **If rules are not mutually exhaustive, how would the class be decided in case different rules with different consequent cover the record.**
- **There are two solutions to the above problem:**
  - Either rules can be ordered ,i.e. the class corresponding to the highest priority rule triggered is taken as the final class.
  - Otherwise, we can assign votes for each class depending on some of their weights, i.e., the rules remain unordered.

Below, is a visual representation describing the working of the algorithm.

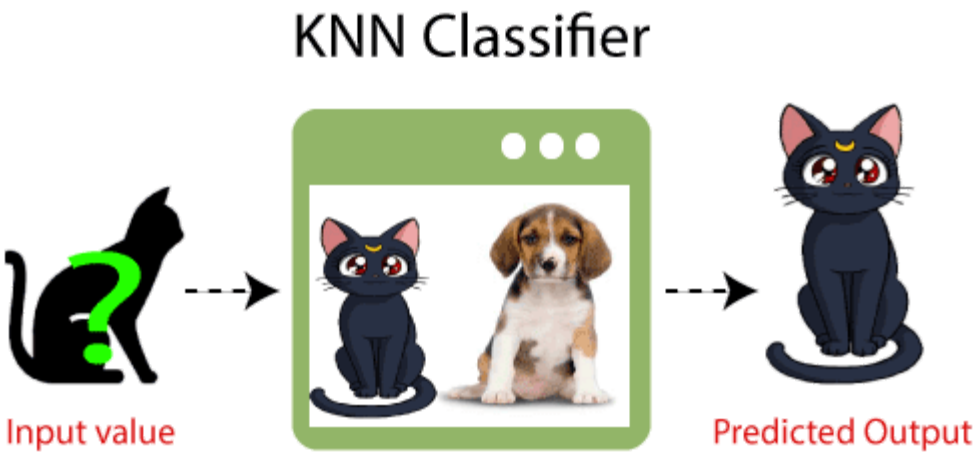


- First, we created an empty decision list. During Step 1, we see that there are three sets of positive examples present in the dataset. So, as per the algorithm, we consider the one with maximum no of positive examples. (6, as shown in Step 1).
- Once we cover these 6 positive examples, we get our first rule R1, which is then pushed into the decision list and those positive examples are removed from the dataset. (as shown in Step 2).
- Now, we take the next majority of positive examples (5, as shown in Step 2) and follow the same process until we get rule R2. (Same for R3)
- In the end, we obtain our final decision list with all the desirable rules.

K-Nearest Neighbor(KNN) Algorithm

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

- The K-NN algorithm assumes the similarity between the new case/data and available cases and puts the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suited category by using K- NN algorithm.
- The K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- The KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- Example: Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know whether it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

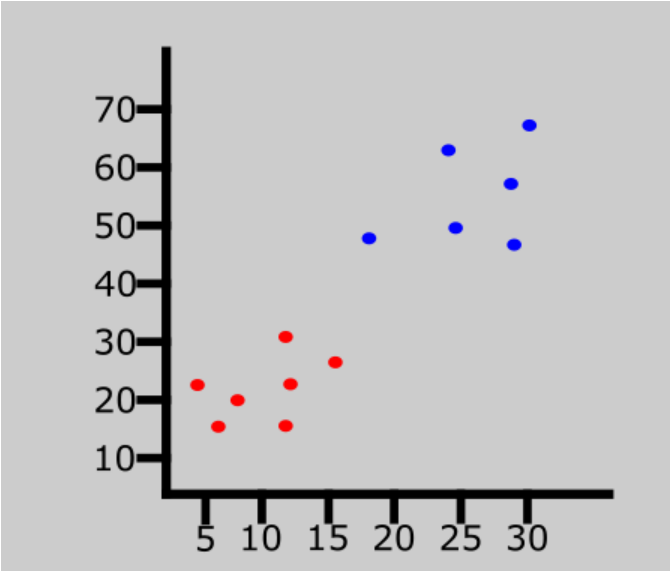


Implementing a KNN model

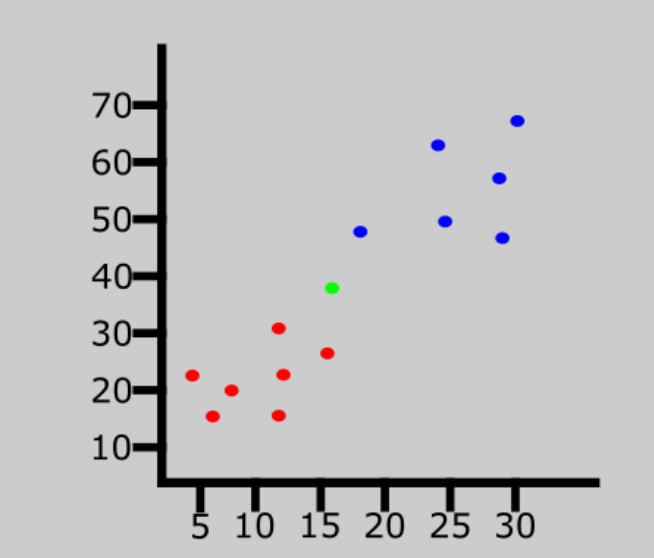
1. Load the data
2. Initialize the value of **k**
3. For getting the predicted class, iterate from 1 to total number of training data points
    - Calculate the distance between test data and each row of training dataset. Use Euclidean distance as our distance metric since it's the most popular method. The other distance functions or metrics that can be used are Manhattan distance, Minkowski distance, Chebyshev, cosine, etc. If there are categorical variables, hamming distance can be used.
    - Sort the calculated distances in ascending order based on distance values
    - Get top k rows from the sorted array
    - Get the most frequent class of these rows
    - Return the predicted class (ie, frequent class).

Example:

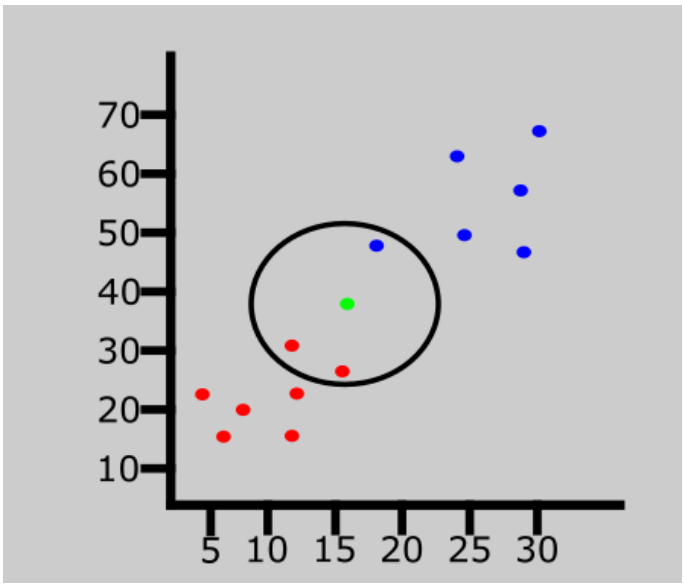
- Consider the diagram below: It is a data set consisting of two classes — red and blue.



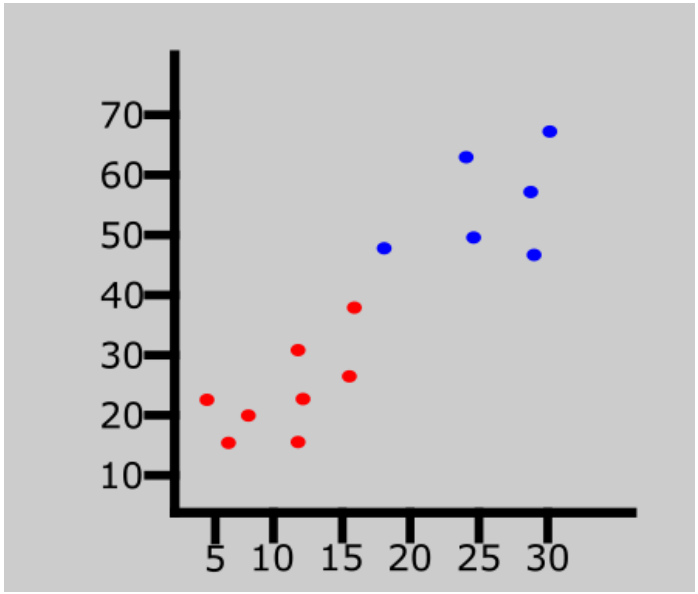
- A new data entry has been introduced to the data set. This is represented by the green point in the graph above.



- Assign a value to K which denotes the number of neighbors to consider before classifying the new data entry. Let's assume the value of K is 3.
- Since the value of K is 3, the algorithm will only consider the 3 nearest neighbors to the green point (new entry).



- Out of the 3 nearest neighbors in the diagram above, the majority class is red so the new entry will be assigned to that class.



Example:

Consider the following data set

BRIGHTNESS	SATURATION	CLASS
40	20	Red
50	50	Blue
60	90	Blue
10	25	Red
70	70	Blue
60	10	Red
25	80	Blue

Predict the class for the following

BRIGHTNESS	SATURATION	CLASS
20	35	?

Solution:

Calculate the distance from the new entry to other entries in the data set using the Euclidean distance formula.

Here's the formula: $\sqrt{(X_2-X_1)^2+(Y_2-Y_1)^2}$

Where:

- X_2 = New entry's brightness (20).
- X_1= Existing entry's brightness.
- Y_2 = New entry's saturation (35).
- Y_1 = Existing entry's saturation.

Distance #1

For the first row, d1:

BRIGHTNESS	SATURATION	CLASS
40	20	Red

$$d1 = \sqrt{(20 - 40)^2 + (35 - 20)^2}$$
$$= \sqrt{400 + 225}$$
$$= \sqrt{625}$$
$$= 25$$

We now know the distance from the new data entry to the first entry in the table. Let's update the table.

BRIGHTNESS	SATURATION	CLASS	DISTANCE
40	20	Red	25
50	50	Blue	?
60	90	Blue	?
10	25	Red	?
70	70	Blue	?
60	10	Red	?
25	80	Blue	?

BRIGHTNESS	SATURATION	CLASS	DISTANCE
40	20	Red	25
50	50	Blue	33.54
60	90	Blue	68.01
10	25	Red	10
70	70	Blue	61.03
60	10	Red	47.17
25	80	Blue	45

Let's rearrange the distances in ascending order:

BRIGHTNESS	SATURATION	CLASS	DISTANCE
10	25	Red	10
40	20	Red	25
50	50	Blue	33.54
25	80	Blue	45
60	10	Red	47.17
70	70	Blue	61.03
60	90	Blue	68.01

Since we chose 5 as the value of K, we'll only consider the first five rows. That is:

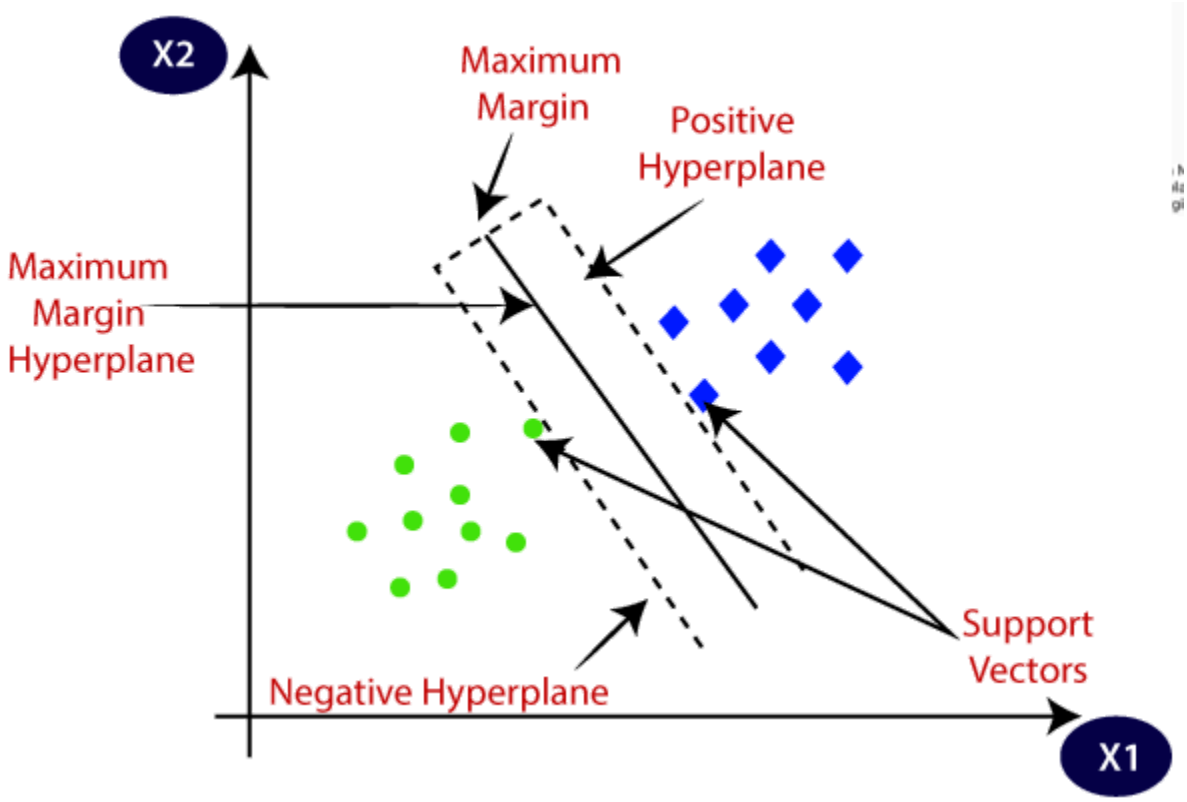
BRIGHTNESS	SATURATION	CLASS	DISTANCE
10	25	Red	10
40	20	Red	25
50	50	Blue	33.54
25	80	Blue	45
60	10	Red	47.17

As you can see above, the majority class within the 5 nearest neighbors to the new entry is **Red**. Therefore, we'll classify the new entry as **Red**.

Support Vector Machine (SVM) 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.
- SVM is particularly effective in high-dimensional spaces and is widely used for various applications, including image classification, text classification, and bioinformatics.
- **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 support vectors, and hence the 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:



Hyperplane:

In a two-dimensional space, the hyperplane is a line. In higher-dimensional spaces, it becomes a hyperplane. The decision boundary is determined by this hyperplane.

Margin:

The margin is the distance between the hyperplane and the nearest data point from each class. SVM seeks to find the hyperplane that maximizes this margin.

Support Vectors:

Support vectors are the data points that lie closest to the decision boundary. They are crucial in defining the hyperplane and calculating the margin.

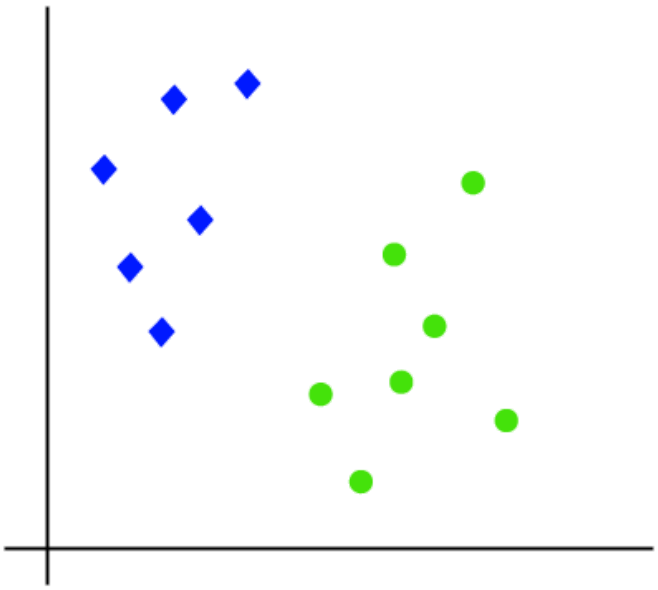
Types of SVM

- **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.

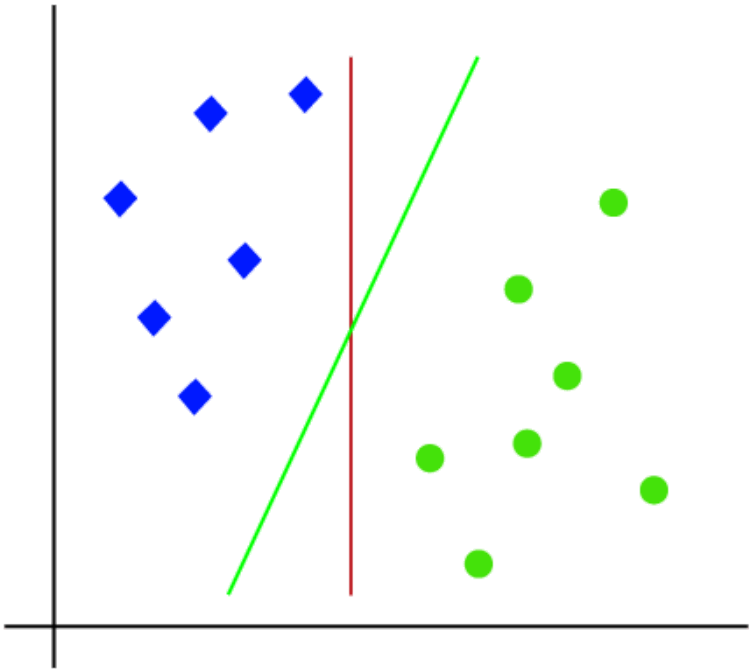
How does SVM works?

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:



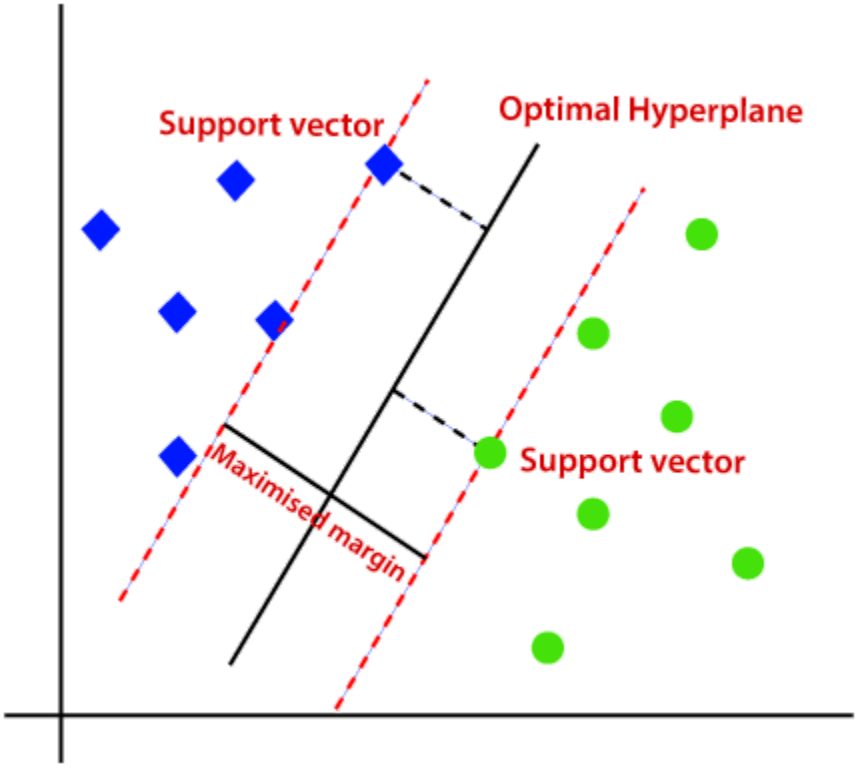
- So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:



- Hence, the SVM algorithm helps to find the best line or decision

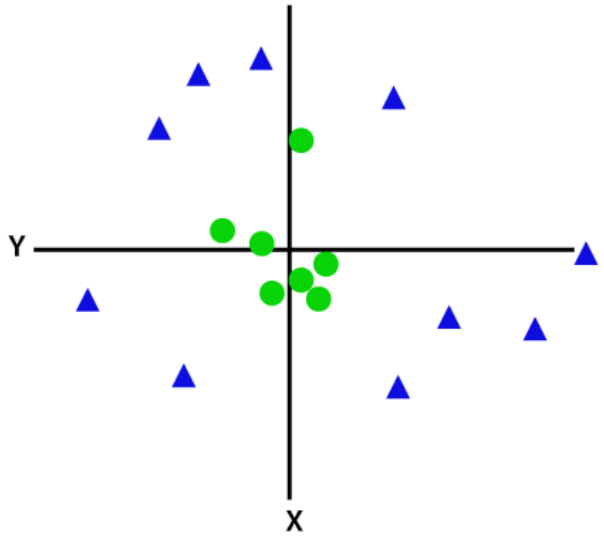
boundary; this best boundary or region is called a hyperplane.

- 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 the margin.
- And the goal of SVM is to maximize this margin. The hyperplane with maximum margin is called the optimal hyperplane.



Non-Linear SVM:

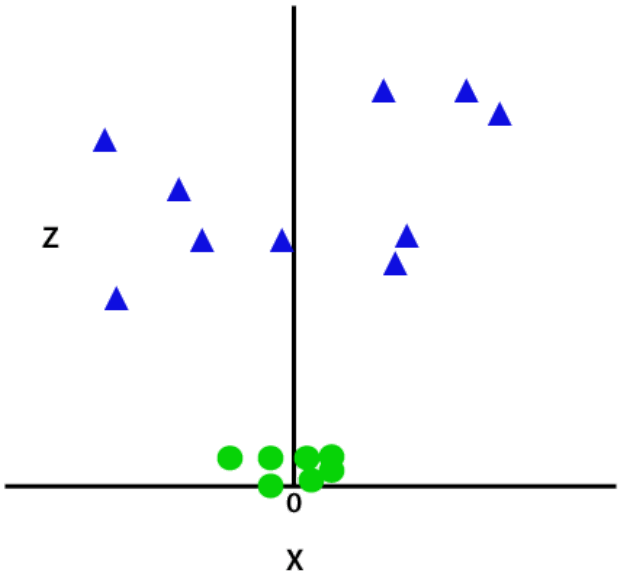
- If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



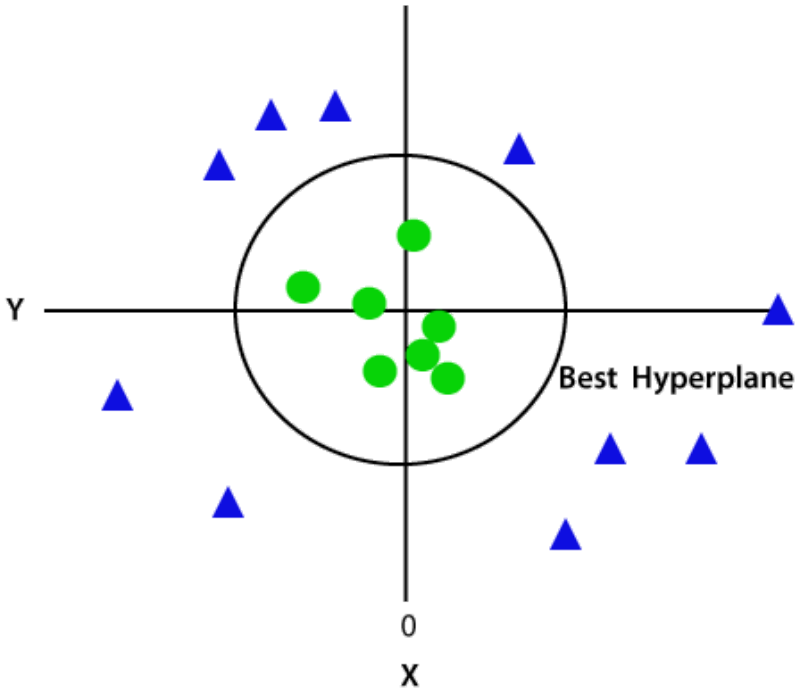
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

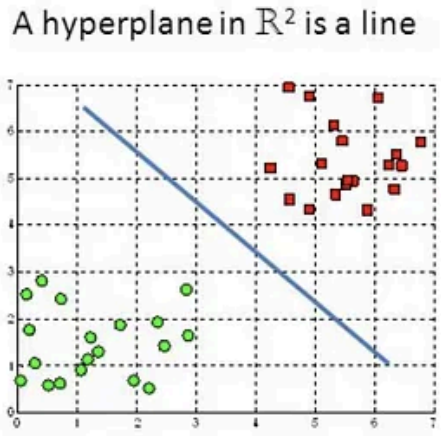
$$z=x2 +y2$$

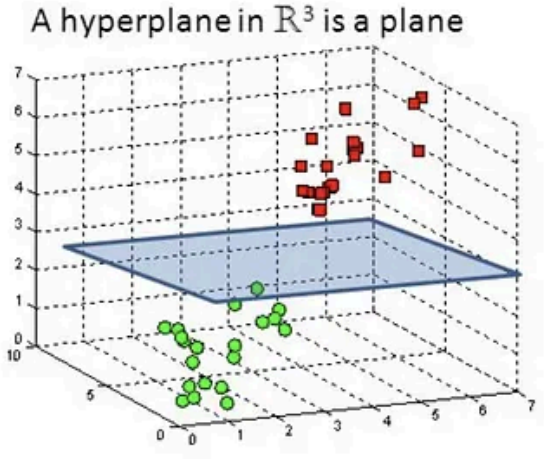
By adding the third dimension, the sample space will become as below image:



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.

Case- Based Reasoning

- Case-based reasoning (CBR) classifiers use a database of problem solutions to solve new problems.
- Unlike nearest-neighbor 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.
- When given a new case to classify, a case-based reasoner will first check if an identical training case exists.
- If one is found, then the accompanying solution to that case is returned.
- If no identical case is found, then the case-based reasoner will search for training cases having components that are similar to those of the new case.
- Conceptually, these training cases may be considered as neighbors of the new case.
- If cases are represented as graphs, this involves searching for subgraphs that are similar to subgraphs within the new case. The case-based reasoner tries to combine the solutions of the neighboring training cases in order to propose a solution for the new case.
- If incompatibilities arise with the individual solutions, then backtracking to search for other solutions may be necessary.
- The case-based reasoner may employ background knowledge and problem-solving strategies in order to propose a feasible combined solution.

Evaluation of a classifier

Confusion Matrix

A confusion matrix is a table that is often used to **describe the performance of a classification model**(or "classifier") on a set of test data for which the true values are known. The confusion matrix itself is relatively simple to understand, but the related terminology can be confusing.

- A **true positive** is an outcome where the model correctly predicts the positive class. Similarly, a **true negative** is an outcome where the model correctly predicts the negative class. (TP)

- A **false positive** is an outcome where the model incorrectly predicts the positive class. And a **false negative** is an outcome where the model incorrectly predicts the negative class. (FP).

Classification Accuracy

Accuracy is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right.Formally, accuracy has the following definition:

Accuracy: Number of correct predictions/ Total number of predictions

For binary classification, accuracy can also be calculated in terms of positives and negatives as follows:

Accuracy=(TP+TN)/(TP+TN+FP+FN)

where TP=true positives, TN= true negatives, FP=false positives, and FN=false negatives.

Precision

- Precision Answers: What proportion of positive identifications was actually correct?
- Precision is defined as:

Precision=TP/(TP+FP)

Recall

- RecallAnswers: What proportion of actual positives was identified correctly? Recall is defined as:

  Recall =TP/(TP+FN)

Area Under the ROC Curve (AUC-ROC):

- The area under the Receiver Operating Characteristic (ROC) curve, which plots the true positive rate against the false positive rate.
- AUC-ROC measures the model's ability to distinguish between positive and negative instances.

F1 Score

- The F1 Score is just a way to have an unique number to represent both the recall and the precision values. It is an harmonic mean between the two and is denoted by the formula below:

$$2 * \frac{Precision * Recall}{Precision + Recall}$$

Matthews Correlation Coefficient (MCC)

- The Matthews Correlation Coefficient (MCC) is a metric used to evaluate the performance of a binary classification model. It takes into account all four components of the confusion matrix (True Positives, True Negatives, False Positives, and False Negatives) and provides a balanced measure of the model's performance, especially in situations where the classes are imbalanced.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

The MCC ranges from -1 to +1, where:

+1 : indicates a perfect prediction.
0 : indicates a performance similar to random chance.
−1: indicates a complete disagreement between predictions and actual values.

Specificity (True Negative Rate):

- The ratio of correctly predicted negative instances to the total actual negative instances.

  Specificity = TN / (TN + FP)

- Specificity measures the ability of the model to correctly identify negative instances.

Cross-validation

- Cross-validation is a statistical technique used in machine learning to assess the performance and generalization ability of a model.
- It helps in evaluating how well a model will perform on an independent dataset by partitioning the available dataset into subsets.
- The primary purpose of cross-validation is to ensure that the model's performance is not overly dependent on a specific training-test split.

- **Training-Test Split:**

- The original dataset is divided into two subsets: a training set and a test set. The model is trained on the training set and then evaluated on the test set to measure its performance.

- **Issues with a Single Split:**

- Depending on the random selection of data points, a single training-test split may lead to biased performance metrics. The model could perform exceptionally well or poorly based on the specific instances in the training or test set.

- **K-Fold Cross-Validation:**

- In K-Fold Cross-Validation, the dataset is divided into K subsets or "folds." The model is trained and evaluated K times, each time using a different fold as the test set and the remaining data as the training set. This process helps in obtaining a more robust estimate of the model's performance.
- For example, in 5-fold cross-validation, the dataset is divided into 5 folds.

The model is trained and tested five times, and the performance metrics are averaged.

- **Leave-One-Out Cross-Validation (LOOCV):**

- A special case of K-Fold Cross-Validation where K is set to the number of data points. In each iteration, a single data point is used as the test set, and the model is trained on the remaining data points. This process is repeated for all data points.

- **Stratified Cross-Validation:**

- In classification tasks with imbalanced class distribution, stratified cross-validation ensures that each fold has a similar distribution of target classes as the entire dataset. This helps prevent bias in the model evaluation.

- **Grid Search and Hyperparameter Tuning:**

- Cross-validation is often used in conjunction with grid search to find the optimal hyperparameters for a model. The hyperparameter values that result in the best cross-validated performance are selected.
- Cross-validation provides a more robust evaluation of a model's performance and helps in detecting issues such as overfitting or underfitting. It is a crucial step in the model development process to ensure that the model's performance estimates are reliable and generalize well to unseen data.