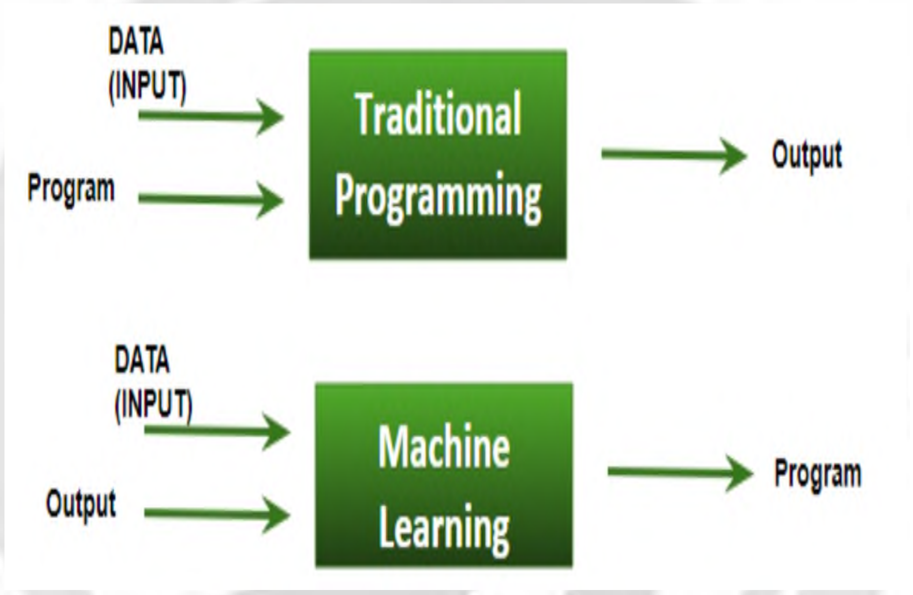
Chapter 3: Machine Learning Algorithms

Machine Learning

Machine learning (ML) is the part of Artificial Intelligence which allows the software applications to become more accurate at predicting outcomes without being explicitly programmed to do so. ML algorithm uses historical data as input to predict new output values.

In other words, we can say it as a field of study that gives the computers the capability to learn without being explicitly programmed.

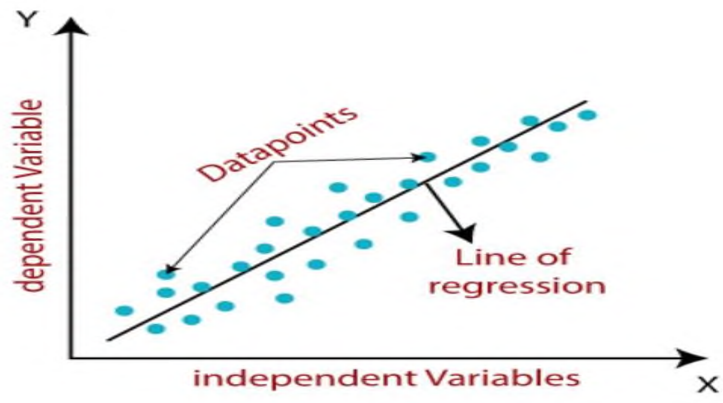


1. <u>Linear Regression Algorithm</u>

Linear regression is one of the most popular used ML algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc.

Linear regression algorithm shows a linear relationship between a dependent (x) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables.



Mathematically, we can represent a linear regression as:

$$y = a_0 + a_1 x + \varepsilon$$

Here,
Y= Dependent Variable (Target Variable)
X= Independent Variable (Predictor Variable)
a0= intercept of the line (Gives an additional degree of freedom)
a1 = Linear regression coefficient (scale factor to each input value).
ε = random error

The values for x and y variables are training datasets for Linear Regression model representation. Linear regression can be further be divided into two types of the algorithm:

- ***Simple Linear Regression*:** If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.
- ***Multiple Linear regressions*:** If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

Cost function

The different values for weights or coefficient of lines (a0, a1) gives the different line of regression, and the cost function is used to estimate the values of the coefficient for the best fit line. Cost function optimizes the regression coefficients or weights. It measures how a linear regression model is performing. We can use the cost function to find the accuracy of the mapping function, which maps the input variable to the output variable. This mapping function is also known as Hypothesis function.

For Linear Regression, we use the Mean Squared Error (MSE) cost function, which is the average of squared error occurred between the predicted values and actual values. It can be written as:

$$MSE= 1\frac{1}{N}\sum_{i=1}^{n}\left(y_i - (a_1x_i + a_0)\right)^2$$

N=Total number of observations
Yi = Actual value
(a1xi+a0) = Predicted value.

Residuals
The distance between the actual value and predicted values is called residual. If the observed points are far from the regression line, then the residual will be high, and so cost function will high. If the scatter points are close to the regression line, then the residual will be small and hence the cost function.

Gradient Descent
Gradient descent is used to minimize the MSE by calculating the gradient of the cost function. A regression model uses gradient descent to update the coefficients of the line by reducing the cost function. It is done by a random selection of values of coefficient and then iteratively update the values to reach the minimum cost function.
The Goodness of fit determines how the line of regression fits the set of observations. The process of finding the best model out of various models is called optimization. It can be achieved by R-squared method.

R-squared is a statistical method that determines the goodness of fit. It measures the strength of the relationship between the dependent and independent variables on a scale of 0-100%. The high value of R-square determines the less difference between the predicted values and actual values and hence represents a good model. It is also called a coefficient of determination, or coefficient of multiple determinations for multiple regressions.

It can be calculated from the below formula:

$$\text{R-squared}=\frac{\text{Explained variation}}{\text{Total Variation}}$$

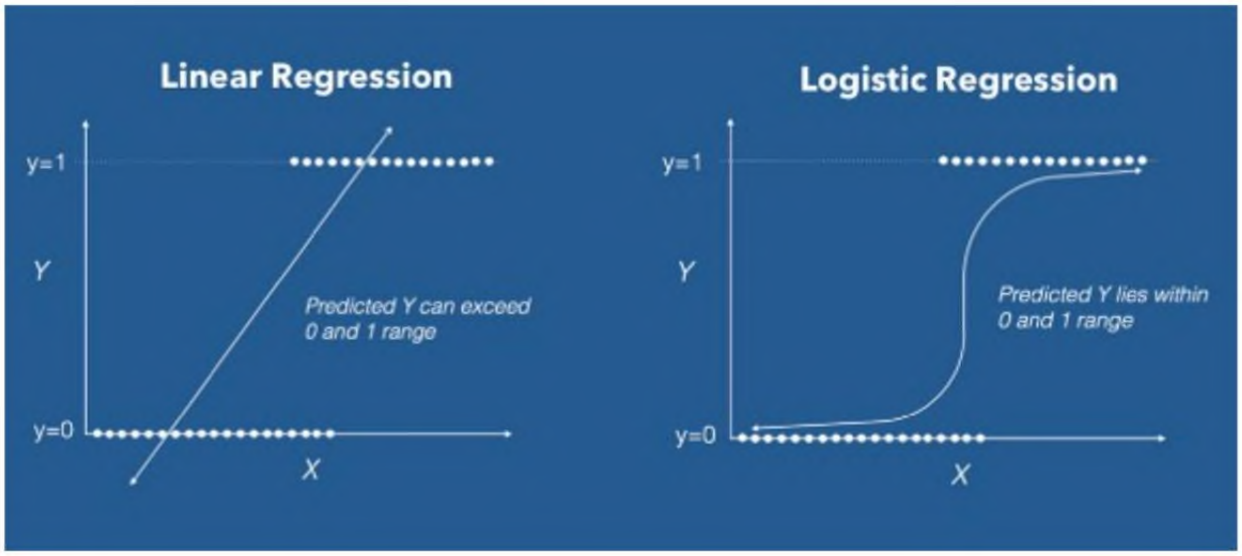
For the above linear equation, MSE can be calculated as:

$$MSE = \frac{1}{n}\Sigma\underbrace{\left(y - \widehat{y}\right)^2}_{\text{The square of the difference between actual and predicted}}$$

2. Logistic Regression Algorithm

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Some of the examples of classification problems are Email spam or not spam, Online transactions Fraud or not Fraud, Tumor Malignant or Benign. Logistic regression transforms its output using the logistic sigmoid function to return a probability value.

Logistic Regression is a Machine Learning algorithm which is used for the classification problems. It is a predictive analysis algorithm and based on the concept of probability.



Sigmoid Function

Regression uses a more complex cost function. This cost function can be defined as the 'Sigmoid function' or also known as the 'logistic function' instead of a linear function.
The hypothesis of logistic regression tends it to limit the cost function between 0 and 1. Therefore linear functions fail to represent it as it can have a value greater than 1 or less than 0 which is not possible as per the hypothesis of logistic regression.

In order to map predicted values to probabilities, we use the Sigmoid function. The function maps any real value into another value between 0 and 1. In machine learning, we use sigmoid to map predictions to probabilities.

$$f(x) = \frac{1}{1 + e^{-(x)}}$$

Gradient Descent

Now the question arises, how do we reduce the cost value? Well, this can be done by using Gradient Descent. The main goal of Gradient descent is to minimize the cost value. i.e., min J(θ).

Now to minimize our cost function we need to run the gradient descent function on each parameter i.e.,

$$\theta j := \theta j - \alpha \frac{\partial}{\partial \theta j} J(\theta)$$

Gradient descent has an analogy in which we have to imagine ourselves at the top of a mountain valley and left stranded and blindfolded, our objective is to reach the bottom of the hill. Feeling the slope of the terrain around you are what everyone would do. Well, this action is analogous to calculating the gradient descent, and taking a step is analogous to one iteration of the update to the parameters.

3. <u>Decision tree Algorithm</u>

Decision tree is one of the supervised non-parametric algorithms. It can be also used for solving the regression and classification problems. This algorithm helps us to predict the value of the target variable by simple learning decision rules from the prior data. We start from the root of the tree by comparing the values of root attribute with the record’s attribute. On the basis of the comparison, we follow the branch corresponding to that values and jump to the next node.

The assumptions while creating a decision tree:

- The whole training set is considered as root.
- The featured values are proffered to be categorical. If the values are continuous then they are converted into discrete.
- Records are distributed recursively on the basis of attribute values.
- Order to placing attributes as root or internal node of the tree is done by using some statistical approach.

The decision of making strategic splits heavily affects a tree’s accuracy. The decision criteria which are made are different for classification and regression trees.

Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that the purity of the node increases with respect to the target variable. The decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.

The algorithm selection is also based on the type of target variables. Let us look at some algorithms used in Decision Trees:

- *ID3*: The ID3 was developed by Ross Quinlan, which is shorthand for "Iterative Dichotomiser 3." This algorithm leverages entropy and information gain as metrics to evaluate candidate splits.
- C4.5: This algorithm is considered a later iteration of ID3, which was also developed by Quinlan. It can use information gain or gain ratios to evaluate split points within the decision trees.
- CART: The term, CART, is an abbreviation for "classification and regression trees" which was introduced by Leo Breiman. This algorithm typically utilizes Gini impurity to identify the ideal attribute to split on. Gini impurity measures how often a randomly chosen attribute is misclassified. When evaluating using Gini impurity, a lower value is more ideal.

Entropy and Information Gain

It's difficult to explain information gain without first discussing entropy. Entropy is a concept that stems from information theory, which measures the impurity of the sample values. It is defined with by the following formula, where:

$$\text{Entropy}(S) = -\sum_{c \in C} p(c)\log_2 p(c)$$

- S represents the data set that entropy is calculated
- c represents the classes in set, S
- p(c) represents the proportion of data points that belong to class c to the number of total data points in set, S

Entropy values can fall between 0 and 1. If all samples in data set, S, belong to one class, then entropy will equal zero. If half of the samples are classified as one class and the other half are in another class, entropy will be at its highest at 1. In order to select the best feature to split on and find the optimal decision tree, the attribute with the smallest amount of entropy should be used. Information gain represents the difference in entropy before and after a split on a given attribute. The attribute with the highest information gain will produce the best split as it's doing the best job at classifying the training data according to its target classification. Information gain is usually represented with the following formula, where:

$$\text{Information Gain}(S,a) = \text{Entropy}(S) - \sum_{\text{vevcalues}(a)} \frac{|S_v|}{|S|}\,\text{Entropy}(S_v)$$

- *a* represents a specific attribute or class label
- *Entropy(S)* is the entropy of dataset, S
- |Sv|/ |S| represents the proportion of the values in S_v to the number of values in dataset, S
- *Entropy* (S_v) is the entropy of dataset, S_v

Gini Impurity

Gini impurity is the probability of incorrectly classifying random data point in the dataset if it were labeled based on the class distribution of the dataset. Similar to entropy, if set, S, is pure – (i.e., belonging to one class) then, its impurity is zero. This is denoted by the following formula:

$$\text{Gini Impurity} = 1 - \sum_i (p_i)^2$$

Advantages and disadvantages of decision tree

The advantages are as follows:

- ***Easy to interpret*:** The Boolean logic and visual representations of decision trees make them easier to understand and consume. The hierarchical nature of a decision tree also makes it easy to see which attributes are most important, which isn't always clear with other algorithms, like neural networks.
- ***Little to no data preparation required*:** Decision trees have a number of characteristics, which make it more flexible than other classifiers. It can handle various data types—i.e., discrete or continuous values, and continuous values can be converted into categorical values through the use of thresholds. Additionally, it can also handle values with missing values, which can be problematic for other classifiers, like Naïve Bayes.
- ***More flexible*:** Decision trees can be leveraged for both classification and regression tasks, making it more flexible than some other algorithms. It's also insensitive to underlying relationships between attributes; this means that if two variables are highly correlated, the algorithm will only choose one of the features to split on.

The disadvantages are as follows:

- ***Prone to overfitting*:** Complex decision trees tend to overfit and do not generalize well to new data. This scenario can be avoided through the processes of pre-pruning or post-pruning. Pre-pruning halts tree growth when there is insufficient data while post-pruning removes subtrees with inadequate data after tree construction.
- ***High variance estimators*:** Small variations within data can produce a very different decision tree. Bagging, or the averaging of estimates, can be a method of reducing variance of decision trees. However, this approach is limited as it can lead to highly correlated predictors.
- ***More costly*:** Given that decision trees take a greedy search approach during construction, they can be more expensive to train compared to other algorithms.
- ***Not fully supported in scikit-learn*:** Scikit-learn is a popular machine learning library based in Python. While this library does have a Decision Tree module, the current implementation does not support categorical variables.

4. Random Forest Algorithm

Random forest is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.

One of the most important features of the Random Forest Algorithm is that it can handle the data set containing continuous variables as in the case of regression and categorical variables as in the case of classification. It performs better results for classification problems.

Before understanding the working of the random forest, we must look into the ensemble technique. Ensemble simply means combining multiple models. Thus, a collection of models is used to make predictions rather than an individual model.

Ensemble uses two types of methods:

- ***Bagging*:** It creates a different training subset from sample training data with replacement & the final output is based on majority voting. For example, Random Forest.
- ***Boosting*:** It combines weak learners into strong learners by creating sequential models such that the final model has the highest accuracy. For example, ADA BOOST, XG BOOST

Bagging, also known as Bootstrap Aggregation is the ensemble technique used by random forest. Bagging chooses a random sample from the data set. Hence each model is generated from the samples (Bootstrap Samples) provided by the Original Data with replacement known as row sampling. This step of row sampling with replacement is called bootstrap. Now each model is trained independently which generates results. The final output is based on majority voting after combining the results of all models. This step which involves combining all the results and generating output based on majority voting is known as aggregation.

Hyperparameters

Hyperparameters are used in random forests to either enhance the performance and predictive power of models or to make the model faster.
Following hyperparameters increases the predictive power:

- ***n_estimators*:** number of trees the algorithm builds before averaging the predictions.
- ***max_features*:** maximum number of features random forest considers splitting a node.
- ***mini_sample_leaf*:** determines the minimum number of leaves required to split an internal node.

Following hyperparameters increases the speed:

- ***n_jobs*:** it tells the engine how many processors it is allowed to use. If the value is 1, it can use only one processor but if the value is -1 there is no limit.

- ***random_state*:** controls randomness of the sample. The model will always produce the same results if it has a definite value of random state and if it has been given the same hyperparameters and the same training data.
- ***oob_score*:** OOB means out of the bag. It is a random forest cross-validation method. In this one-third of the sample is not used to train the data instead used to evaluate its performance. These samples are called out of bag samples.

5. Support Vector Machines (SVM)

Support vector machines are a set of supervised learning methods used for classification, regression, and outlier's detection. All of these are common tasks in machine learning. You can use them to detect cancerous cells based on millions of images or you can use them to predict future driving routes with a well-fitted regression model. There are specific types of SVMs you can use for particular machine learning problems, like support vector regression (SVR) which is an extension of support vector classification (SVC). The main thing to keep in mind here is that these are just math equations tuned to give you the most accurate answer possible as quickly as possible.

SVMs are different from other classification algorithms because of the way they choose the decision boundary that maximizes the distance from the nearest data points of all the classes. The decision boundary created by SVMs is called the maximum margin classifier or the maximum margin hyper plane.

Importance of SVM

SVMs are used in applications like handwriting recognition, intrusion detection, face detection, email classification, gene classification, and in web pages. This is one of the reasons we use SVMs in machine learning. It can handle both classification and regression on linear and non-linear data.
Another reason we use SVMs is because they can find complex relationships between your data without you needing to do a lot of transformations on your own. It's a great option when you are working with smaller datasets that have tens to hundreds of thousands of features. They typically find more accurate results when compared to other algorithms because of their ability to handle small, complex datasets.

Kernel functions

SVM algorithms use a set of mathematical functions that are defined as the kernel. The function of kernel is to take data as input and transform it into the required form. Different SVM algorithms use different types of kernel functions. These functions can be different types.

- ***Linear***
  These are commonly recommended for text classification because most of these types of classification problems are linearly separable. The linear kernel works really well when there are a lot of features, and text classification problems have a lot of features. Linear kernel functions are faster than most of the others and you have fewer parameters to optimize.

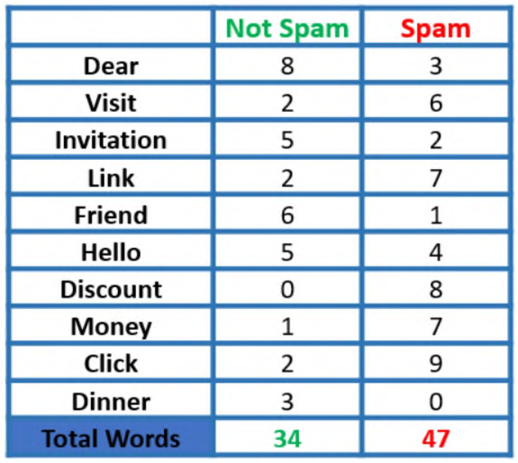
- ***Polynomial***
  The polynomial kernel isn't used in practice very often because it isn't as computationally efficient as other kernels and its predictions aren't as accurate.
- ***Gaussian Radial Basis Function (RBF)***
  One of the most powerful and commonly used kernels in SVMs. Usually the choice for non-linear data. It helps in improving the transformation.
- ***Sigmoid***
  More useful in neural networks than in support vector machines, but there are occasional specific use cases. This function is equivalent to a two-layer, perceptron model of the neural network, which is used as an activation function for artificial neurons.

6. Naive Bayes Classifier Algorithm

Naive Bayes is a classification technique that is based on Bayes' Theorem with an assumption that all the features that predicts the target value are independent of each other. It calculates the probability of each class and then pick the one with the highest probability. It has been successfully used for many purposes, but it works particularly well with natural language processing (NLP) problems. Bayes Theorem describes the probability of an event, based on a prior knowledge of conditions that might be related to that event.

Naive Bayes classifier assumes that the features we use to predict the target are independent and do not affect each other. While in real-life data, features depend on each other in determining the target, but this is ignored by the Naive Bayes classifier. Though the independence assumption is never correct in real-world data, but often works well in practice. so that it is called "Naïve".

Assume we have a bunch of emails that we want to classify as spam or not spam. Our dataset has 15 Not Spam emails and 10 Spam emails. Some analysis had been done, and the frequency of each word had been recorded as shown below:



	Not Spam	Spam
Dear	8	3
Visit	2	6
Invitation	5	2
Link	2	7
Friend	6	1
Hello	5	4
Discount	0	8
Money	1	7
Click	2	9
Dinner	3	0
Total Words	34	47

Let's explore some probabilities:

P(Dear|Not Spam) = 8/34
P(Visit|Not Spam) = 2/34
P(Dear|Spam) = 3/47
P(Visit|Spam) = 6/47
and so on.

Now assume we have the message “Hello friend” and we want to know whether it is a spam or not. Hence, using Bayes’ Theorem

$$P(Not\ Spam|Hello\ Friend) = \frac{P(Hello\ Friend|Not\ Spam) * P(Not\ Spam)}{P(Hello\ Friend)}$$

Let’s ignore the denominator, the formula will look like:

$$P(Not\ Spam|Hello\ Friend) = P(Hello\ Friend|Not\ Spam) * P(Not\ Spam)$$

But, P(Hello friend | Not Spam) = 0, as this case (Hello friend) doesn’t exist in our dataset, i.e. we deal with single words, not the whole sentence, and the same for P(Hello friend | Spam) will be zero as well, which in turn will make both probabilities of being a spam and not spam both are zero, which has no meaning.

We had said that the Naive Bayes assumes that `the features we use to predict the target are independent. So,

$$P(Hello\ Friend|Not\ Spam) = P(Hello|Not\ Spam) * P(Friend|Not\ Spam)$$

$$P(Not\ Spam|Hello\ Friend) = P(Hello|Not\ Spam) * P(Friend|Not\ Spam) * P(Not\ Spam)$$

$$P(Not\ Spam|Hello\ Friend) = \frac{5}{34} * \frac{6}{34} * \frac{15}{25} = 0.0155$$

Now let’s calculate the probability of being spam using the same procedure:

$$P(Spam|Hello\ Friend) = \frac{4}{47} * \frac{1}{47} * \frac{10}{25} = 0.00072$$

Therefore, the message “Hello friend” is not a spam.

Types of Naive Bayes Classifiers

- ***Multinomial*****:** Feature vectors represent the frequencies with which certain events have been generated by a multinomial distribution. For example, the count how often each word occurs in the document. This is the event model typically used for document classification.
- ***Bernoulli*****:** Like the multinomial model, this model is popular for document classification tasks, where binary term occurrence (i.e., a word occurs in a document or not) features are used rather than term frequencies (i.e., frequency of a word in the document).
- ***Gaussian*****:** It is used in classification and it assumes that features follow a normal distribution.

The Zero-Frequency Problem

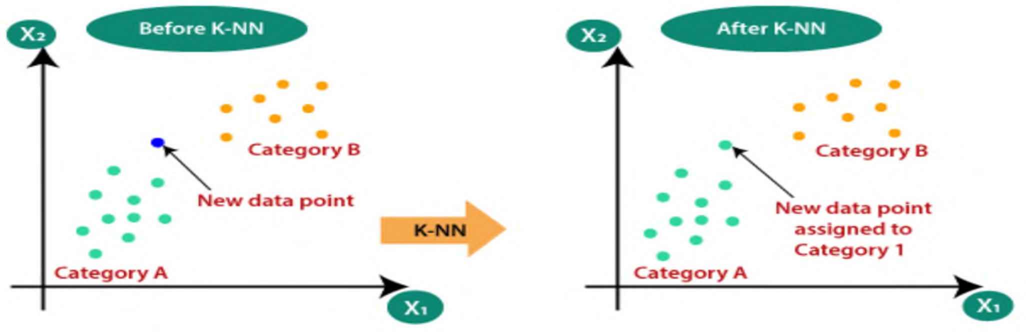
One of the disadvantages of Naïve-Bayes is that if you have no occurrences of a class label and a certain attribute value together then the frequency-based probability estimate will be zero. And this will get a zero when all the probabilities are multiplied.
An approach to overcome this 'zero-frequency problem' in a Bayesian environment is to add one to the count for every attribute value-class combination when an attribute value doesn't occur with every class value.

7. K-Nearest Neighbors Algorithm (KNN)

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. The algorithm's learning is:

- ***Instance-based learning:*** Here we do not learn weights from training data to predict output (as in model-based algorithms) but use entire training instances to predict output for unseen data.
- ***Lazy Learning:*** Model is not learned using training data prior and the learning process is postponed to a time when prediction is requested on the new instance.
- ***Non -Parametric:*** In KNN, there is no predefined form of the mapping function.

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



Determine your distance metrics

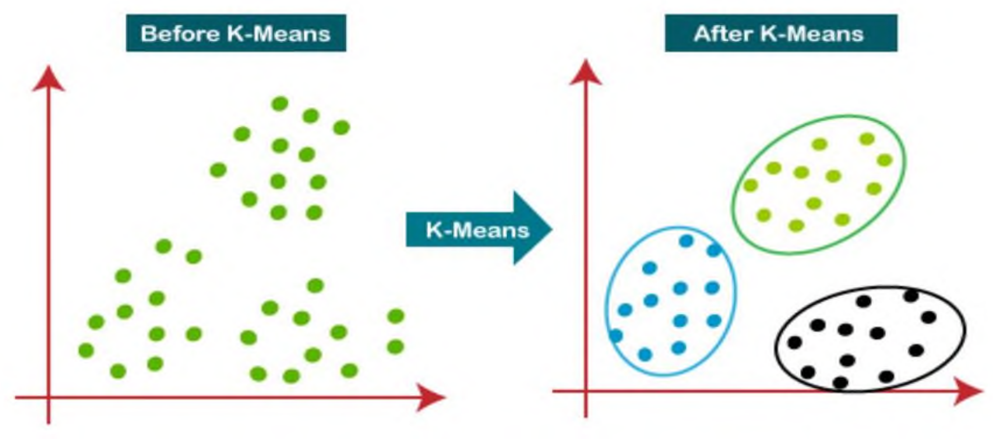
In order to determine which data points are closest to a given query point, the distance between the query point and the other data points will need to be calculated. These distance metrics help to form decision boundaries, which partitions query points into different regions. You commonly will see decision boundaries visualized with diagrams.

- ***Euclidean distance*:** This is the most commonly used distance measure, and it is limited to real-valued vectors. Using the below formula, it measures a straight line between the query point and the other point being measured.
- ***Manhattan distance*:** This is also another popular distance metric, which measures the absolute value between two points. It is also referred to as taxicab distance or city block distance as it is commonly visualized with a grid, illustrating how one might navigate from one address to another via city streets.
- ***Minkowski distance*:** This distance measure is the generalized form of Euclidean and Manhattan distance metrics. The parameter, p, in the formula below, allows for the creation of other distance metrics. Euclidean distance is represented by this formula when p is equal to two, and Manhattan distance is denoted with p equal to one.
- ***Hamming distance*:** This technique is used typically used with Boolean or string vectors, identifying the points where the vectors do not match. As a result, it has also been referred to as the overlap metric.

8. <u>K-Means Algorithm</u>

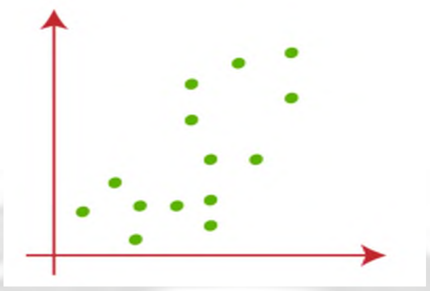
K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

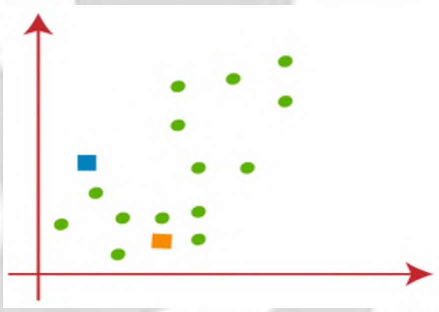


Let’s understand it with an example.

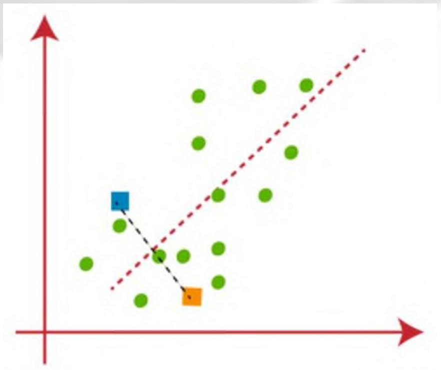
Suppose we have two variables M1 and M2. The x-y axis scatter plot of these two variables is given below:



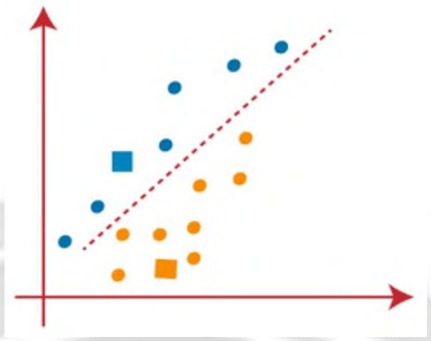
- Let's take number k of clusters, i.e., K=2, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.
- We need to choose some random k points or centroid to form the cluster. These points can be either the points from the dataset or any other point. So, here we are selecting the below two points as k points, which are not the part of our dataset. Consider the below image



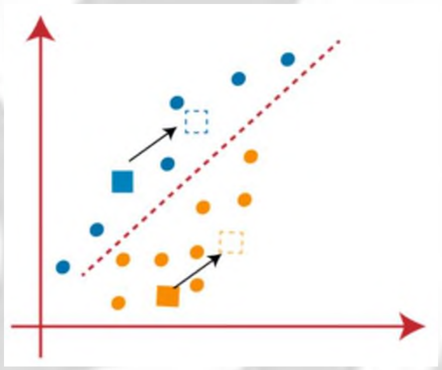
- Now we will assign each data point of the scatter plot to its closest K-point or centroid. We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a median between both the centroids. Consider the below image:



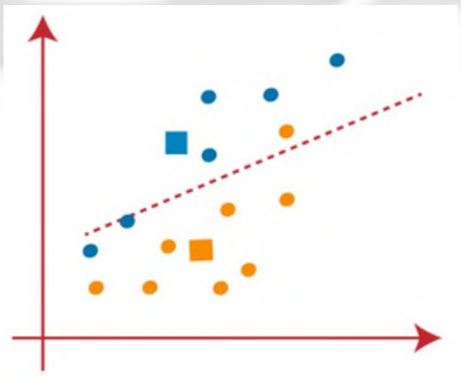
- From the above image, it is clear that points left side of the line is near to the K1 or blue centroid, and points to the right of the line are close to the yellow centroid. Let's color them as blue and yellow for clear visualization.



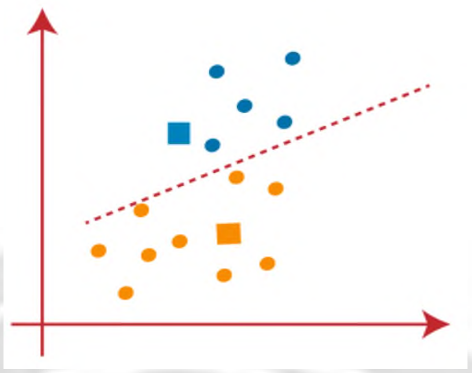
- As we need to find the closest cluster, so we will repeat the process by choosing a new centroid. To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids as below



- Next, we will reassign each datapoint to the new centroid. For this, we will repeat the same process of finding a median line. The median will be like below image

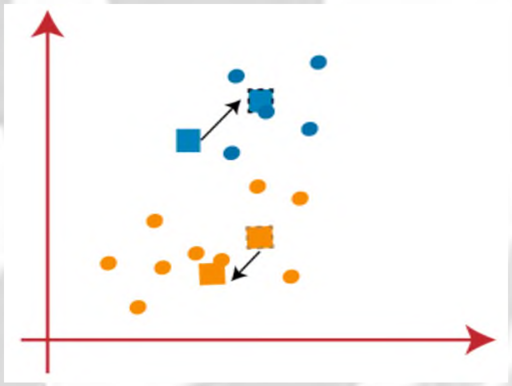


- From the above image, we can see, one yellow point is on the left side of the line, and two blue points are right to the line. So, these three points will be assigned to new centroids.

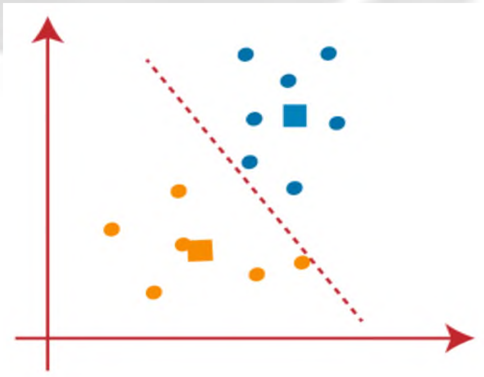


As reassignment has taken place, so we will again go to the step-4, which is finding new centroids or K-points.

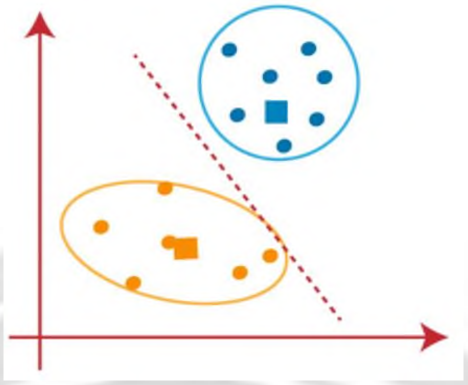
- We will repeat the process by finding the center of gravity of centroids, so the new centroids will be as shown in the below image



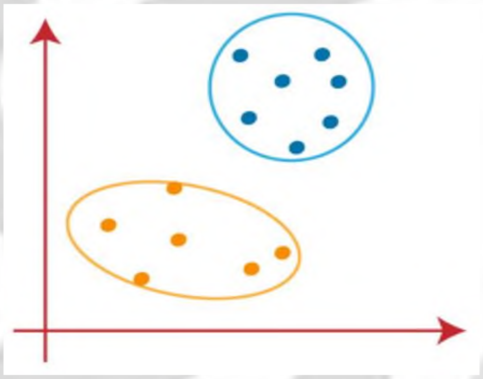
- As we got the new centroids so again will draw the median line and reassign the data points. So, the image will be



- We can see in the above image; there are no dissimilar data points on either side of the line, which means our model is formed. Consider the below image



- As our model is ready, so we can now remove the assumed centroids, and the two final clusters will be as shown in the below image



9. <u>Apriori Algorithm</u>

The primary objective of the apriori algorithm is to create the association rule between different objects. The association rule describes how two or more objects are related to one another. Apriori algorithm is also called frequent pattern mining. Generally, you operate the Apriori algorithm on a database that consists of a huge number of transactions. Let's understand the apriori algorithm with the help of an example; suppose you go to Big Bazar and buy different products. It helps the customers buy their products with ease and increases the sales performance of the Big Bazar. Here, we will discuss the apriori algorithm with examples.

Itemset

A set of items together is called an itemset. If any itemset has k-items it is called a k-itemset. An itemset consists of two or more items. An itemset that occurs frequently is called a frequent itemset.

Thus, frequent itemset mining is a data mining technique to identify the items that often occur together.

What is a Frequent Itemset?

A set of items is called frequent if it satisfies a minimum threshold value for support and confidence. Support shows transactions with items purchased together in a single transaction. Confidence shows transactions where the items are purchased one after the other.

Steps In Apriori

Apriori algorithm is a sequence of steps to be followed to find the most frequent itemset in the given database. This data mining technique follows the join and the prune steps iteratively until the most frequent itemset is achieved. A minimum support threshold is given in the problem or it is assumed by the user.

Step 1: In the first iteration of the algorithm, each item is taken as a 1-itemsets candidate. The algorithm will count the occurrences of each item.

Step 2: Let there be some minimum support, min_sup (eg 2). The set of 1 – itemsets whose occurrence is satisfying the min sup are determined. Only those candidates which count more than or equal to min_sup, are taken ahead for the next iteration and the others are pruned.

Step 3: Next, 2-itemset frequent items with min_sup is discovered. For this in the join step, the 2-itemset is generated by forming a group of 2 by combining items with itself.

Step 4: The 2-itemset candidates are pruned using min-sup threshold value. Now the table will have 2 –itemsets with min-sup only.

Step 5: The next iteration will form 3 –itemsets using join and prune step. This iteration will follow antimonotone property where the subsets of 3-itemsets, that is the 2 –itemset subsets of each group fall in min_sup. If all 2-itemset subsets are frequent then the superset will be frequent otherwise it is pruned.

Step 6: Next step will follow making 4-itemset by joining 3-itemset with itself and pruning if its subset does not meet the min_sup criteria. The algorithm is stopped when the most frequent itemset is achieved.

Methods to improve Apriori efficiency

- ***Hash-Based Technique*:** This method uses a hash-based structure called a hash table for generating the k-item sets and its corresponding count. It uses a hash function for generating the table.
- ***Transaction Reduction*:** This method reduces the number of transactions scanning in iterations. The transactions which do not contain frequent items are marked or removed.

- ***Partitioning*****:** This method requires only two database scans to mine the frequent item sets. It says that for any itemset to be potentially frequent in the database, it should be frequent in at least one of the partitions of the database.
- ***Sampling*****:** This method picks a random sample S from Database D and then searches for frequent itemset in S. It may be possible to lose a global frequent itemset. This can be reduced by lowering the min_sup.
- ***Dynamic Itemset Counting*****:** This technique can add new candidate item sets at any marked start point of the database during the scanning of the database.

10. Principle Component Analysis

Principal Component Analysis, or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize and make analyzing data much easier and faster for machine learning algorithms without extraneous variables to process.

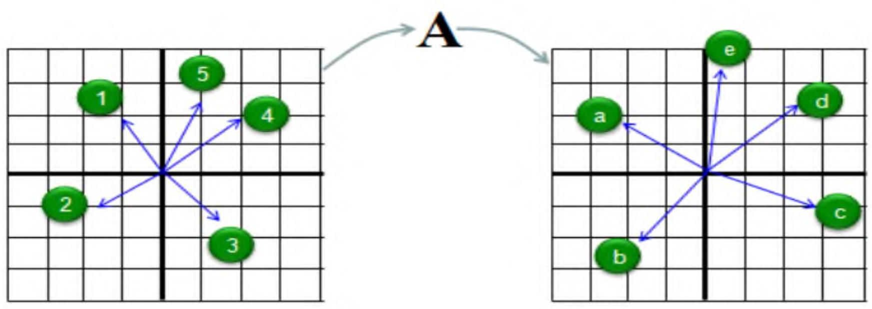
Eigenvector and Eigenvalue

If you think of a Matrix as a geometric transformer, the Matrix usually perform two types of transformational action. One is 'scaling (extend/shrink)' and the other one is 'rotation'. (There are some additional types of transformation like 'shear', 'reflection', but these would be described by special combination of scaling and rotation).

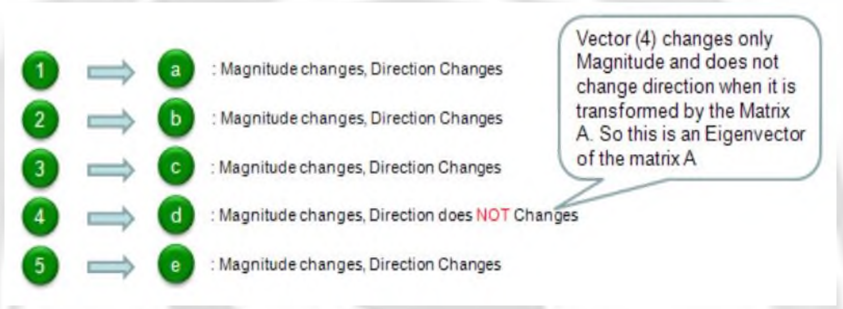
Eigenvector and Eigenvalue can give you the information on the scaling and rotational characteristics of a Matrix. Eigenvector would give you the rotational characteristics and Eigenvalue would give you the scaling characteristics of the Matrix.

When a vector is transformed by a Matrix, usually the matrix changes both direction and amplitude of the vector, but if the matrix applies to a specific vector, the matrix changes only the amplitude (magnitude) of the vector, not the direction of the vector. This specific vector that changes its amplitude only (not direction) by a matrix is called Eigenvector of the matrix.

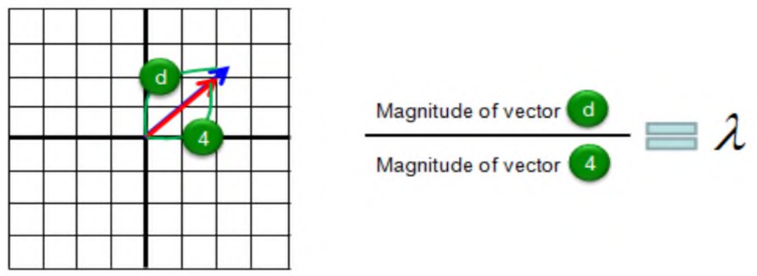
Let me try explaining the concept of eigenvector in more intuitive way. Let's assume we have a matrix called 'A'. We have 5 different vectors shown in the left side. These 5 vectors are transformed to another 5 different vectors by the matrix A as shown on the right side. Vector (1) is transformed to vector (a), Vector (2) is transformed to vector (b) and so on.



Compare the original vector and the transformed vector and check which one has changes both its direction and magnitude and which one changes its magnitude ONLY. The result in this example is as follows. According to this result, vector (4) is the eigenvector of Matrix 'A'.


1 ⇒ a : Magnitude changes, Direction Changes
2 ⇒ b : Magnitude changes, Direction Changes
3 ⇒ c : Magnitude changes, Direction Changes
4 ⇒ d : Magnitude changes, Direction does NOT Changes
5 ⇒ e : Magnitude changes, Direction Changes
Vector (4) changes only Magnitude and does not change direction when it is transformed by the Matrix A. So this is an Eigenvector of the matrix A

Now we know eigenvector changes only its magnitude when applied by the corresponding matrix. Then the question is "How much in magnitude it changes?". Did it get larger? or smaller? exactly how much? The indicator showing the magnitude change is called Eigenvalue. For example, if the eigenvalue is 1.2, it means that the magnitude of the vector gets larger than the original magnitude by 20% and if the eigenvalue is 0.8, it means the vector got smaller than the original vector by 20 %. The graphical presentation of eigenvalue is as follows.



We can use principal component analysis (PCA) for the following purposes:

- To reduce the number of dimensions in the dataset.
- To find patterns in the high-dimensional dataset
- To visualize the data of high dimensionality
- To ignore noise
- To improve classification

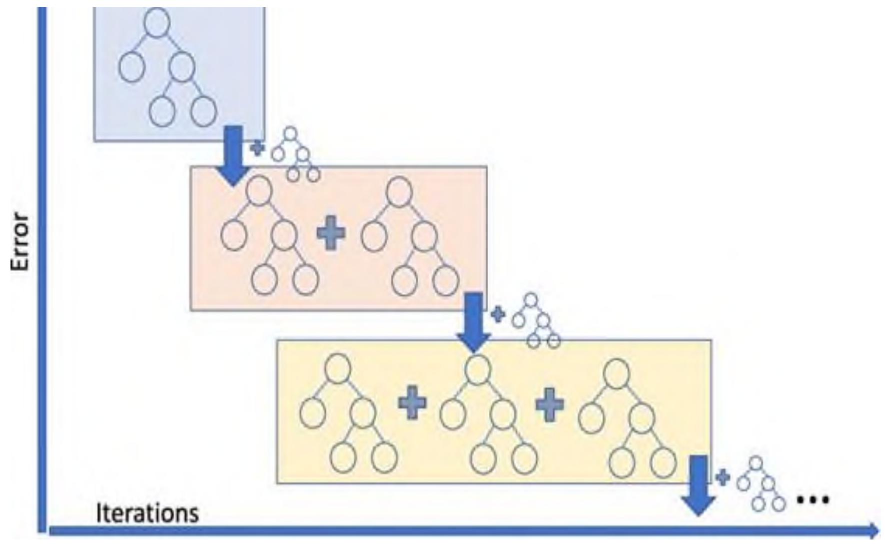
- To gets a compact description
- To captures as much of the original variance in the data as possible

In summary, we can define **principal component analysis (PCA)** as the transformation of any high number of variables into a smaller number of uncorrelated variables called principal components (PCs), developed to capture as much of the data's variance as possible.

11. Boosting Algorithms or Ensemble technique

Ensemble learning or boosting has become one of the most promising approaches for analyzing data in machine learning techniques. The method was initially proposed as ensemble methods based on the principle of generating multiple predictions and average voting among individual classifiers. We have compared two of the popular boosting algorithms, Gradient Boosting and AdaBoost.

Gradient Boosting
Gradient boosting is a type of boosting algorithm. It relies on the intuition that the best possible next model, when combined with previous models, minimizes the overall prediction error. The key idea is to set the target outcomes for this next model in order to minimize the error. Gradient Boosting can be used for both Classification and Regression.



The difference between Gradient Boost and Ada Boost lies in what it does with the underfitted values of its predecessor. Contrary to AdaBoost, which tweaks the instance weights at every interaction, this method tries to fit the new predictor to the *residual errors* made by the previous predictor.

Basically, Gradient Boosting involves three elements:
1. A loss function to be optimized.
2. A weak learner to make predictions.
3. An additive model to add weak learners to minimize the loss function.