# Classification

Classification is a very important area of supervised machine learning. A large number of important machine learning problems fall within this area. Classification is an area of supervised machine learning that tries to predict which class or category some entity belongs to, based on its features.

For example, you might analyze the employees of some company and try to establish a dependence on the features or variables, such as the level of education, number of years in a current position, age, salary, odds for being promoted, and so on. The set of data related to a single employee is one observation.

There are two main types of classification problems:

1. Binary or binomial classification: exactly two classes to choose between (usually 0 and 1, true and false, or positive and negative)
2. Multiclass or multinomial classification: three or more classes of the outputs to choose from

Methods used for classification are

* Logistic Regression
* k-Nearest Neighbors
* Naive Bayes classifiers
* Support Vector Machines
* Decision Trees
* Random Forests
* Neural Networks

# Classification Performance

Binary classification has four possible types of results:

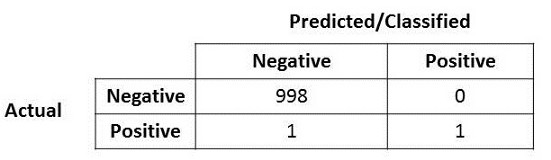
* **True negatives**: correctly predicted negatives (zeros)
* **True positives**: correctly predicted positives (ones)
* **False negatives**: incorrectly predicted negatives (zeros)
* **False positives**: incorrectly predicted positives (ones)

You usually evaluate the performance of your classifier by comparing the actual and predicted outputs and counting the correct and incorrect predictions.

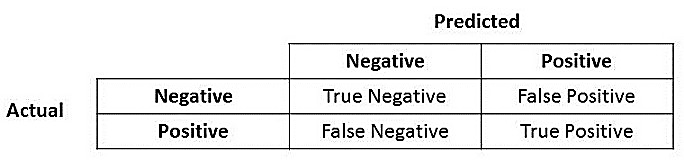
The most straightforward indicator of classification accuracy is the ratio of the number of correct predictions to the total number of predictions (or observations). Other indicators of binary classifiers include the following:

* The **positive predictive** value is the ratio of the number of true positives to the sum of the numbers of true and false positives.
* The **negative predictive** value is the ratio of the number of true negatives to the sum of the numbers of true and false negatives.
* The **sensitivity** (also known as recall or true positive rate) is the ratio of the number of true positives to the number of actual positives.
* The **specificity** (or true negative rate) is the ratio of the number of true negatives to the number of actual negatives.

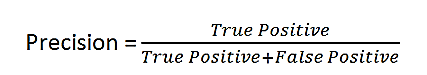
Firstly, let us look at the following confusion matrix. What is the accuracy for the model?



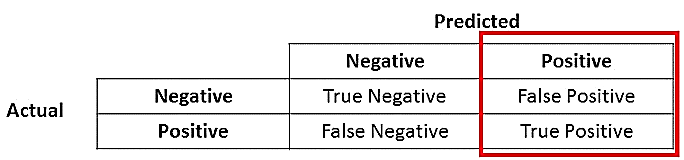
Very easily, you will notice that the accuracy for this model is very high, at 99.9%. what if I mentioned that the positive over here is actually represents terrorist that the model says it’s a non-terrorist? And vice versa.

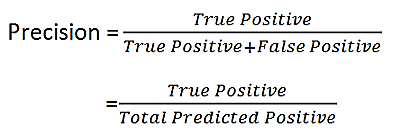


## ****Precision****



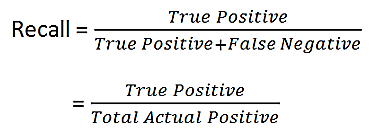
What do you notice for the denominator? The denominator is actually the Total Predicted Positive! So the formula becomes

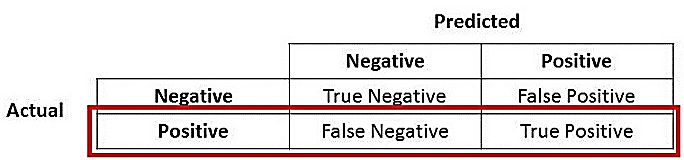




Precision is a good measure to determine, when the costs of False Positive is high. For instance, email spam detection. In email spam detection, a false positive means that an email that is non-spam (actual negative) has been identified as spam (predicted spam). The email user might lose important emails if the precision is not high for the spam detection model.

## ****Recall****





Recall actually calculates how many of the Actual Positives our model capture through labeling it as Positive (True Positive). Applying the same understanding, we know that Recall shall be the model metric we use to select our best model when there is a high cost associated with False Negative.

For instance, in fraud detection or sick patient detection. If a fraudulent transaction (Actual Positive) is predicted as non-fraudulent (Predicted Negative), the consequence can be very bad for the bank.

Similarly, in sick patient detection. If a sick patient (Actual Positive) goes through the test and predicted as not sick (Predicted Negative). The cost associated with False Negative will be extremely high if the sickness is contagious.

## F1 Score

F1 which is a function of Precision and Recall.

C:\Users\Vinsmon TP\Pictures\1_T6kVUKxG_Z4V5Fm1UXhEIw.png

F1 Score is needed when you want to seek a balance between Precision and Recall. Right…so what is the difference between F1 Score and Accuracy then? We have previously seen that accuracy can be largely contributed by a large number of True Negatives which in most business circumstances, we do not focus on much whereas False Negative and False Positive usually has business costs (tangible & intangible) thus F1 Score might be a better measure to use if we need to seek a balance between Precision and Recall AND there is an uneven class distribution (large number of Actual Negatives).

# Logistic Regression

Logistic regression is a fundamental classification technique. It belongs to the group of linear classifiers and is somewhat similar to polynomial and linear regression. Logistic regression is fast and relatively uncomplicated, and it’s convenient for you to interpret the results. Although it’s essentially a method for binary classification, it can also be applied to multiclass problems.

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). In other words, the logistic regression model predicts P(Y=1) as a function of X.

## Logistic Regression Assumptions

* Binary logistic regression requires the dependent variable to be binary.
* For a binary regression, the factor level 1 of the dependent variable should represent the desired outcome.
* Only the meaningful variables should be included.
* The independent variables should be independent of each other. That is, the model should have little or no multi collinearity.
* The independent variables are linearly related to the log odds.
* Logistic regression requires quite large sample sizes.

## Working of Logistic Regression

It is a special case of linear regression where the target variable is categorical in nature. It uses a log of odds as the dependent variable. Logistic Regression predicts the probability of occurrence of a binary event utilizing a logit function.

Linear Regression Equation:

C:\Users\Vinsmon TP\Pictures\image1_ga8gze.png

Where, y is dependent variable and x1, x2 ... and Xn are explanatory variables.

Sigmoid Function:

C:\Users\Vinsmon TP\Pictures\image2_kwxquj.png

Apply Sigmoid function on linear regression:

C:\Users\Vinsmon TP\Pictures\image3_qldafx.png

to find the logistic regression function 𝑝(𝐱) such that the predicted responses 𝑝(𝐱ᵢ) are as close as possible to the actual response 𝑦ᵢ for each observation 𝑖 = 1, …, 𝑛. Remember that the actual response can be only 0 or 1 in binary classification problems! This means that each 𝑝(𝐱ᵢ) should be close to either 0 or 1. That’s why it’s convenient to use the sigmoid function.

Once you have the logistic regression function 𝑝(𝐱), you can use it to predict the outputs for new and unseen inputs, assuming that the underlying mathematical dependence is unchanged.

To get the best weights, you usually maximize the log**-**likelihoodfunction(LLF) for all observations 𝑖 = 1, …, 𝑛. This method is called the maximumlikelihoodestimation and is represented by the equation LLF = Σᵢ(𝑦ᵢ log(𝑝(𝐱ᵢ)) + (1 − 𝑦ᵢ) log(1 − 𝑝(𝐱ᵢ))).

When 𝑦ᵢ = 0, the LLF for the corresponding observation is equal to log(1 − 𝑝(𝐱ᵢ)). If (𝐱ᵢ) is close to 𝑦ᵢ = 0, then log(1 − 𝑝(𝐱ᵢ)) is close to 0. This is the result you want. If (𝐱ᵢ) is far from 0, then log(1 − 𝑝(𝐱ᵢ)) drops significantly. You don’t want that result because your goal is to obtain the maximum LLF. Similarly, when 𝑦ᵢ = 1, the LLF for that observation is 𝑦ᵢ log(𝑝(𝐱ᵢ)). If 𝑝(𝐱ᵢ) is close to 𝑦ᵢ = 1, then log(𝑝(𝐱ᵢ)) is close to 0. If 𝑝(𝐱ᵢ) is far from 1, then log(𝑝(𝐱ᵢ)) is a large negative number.

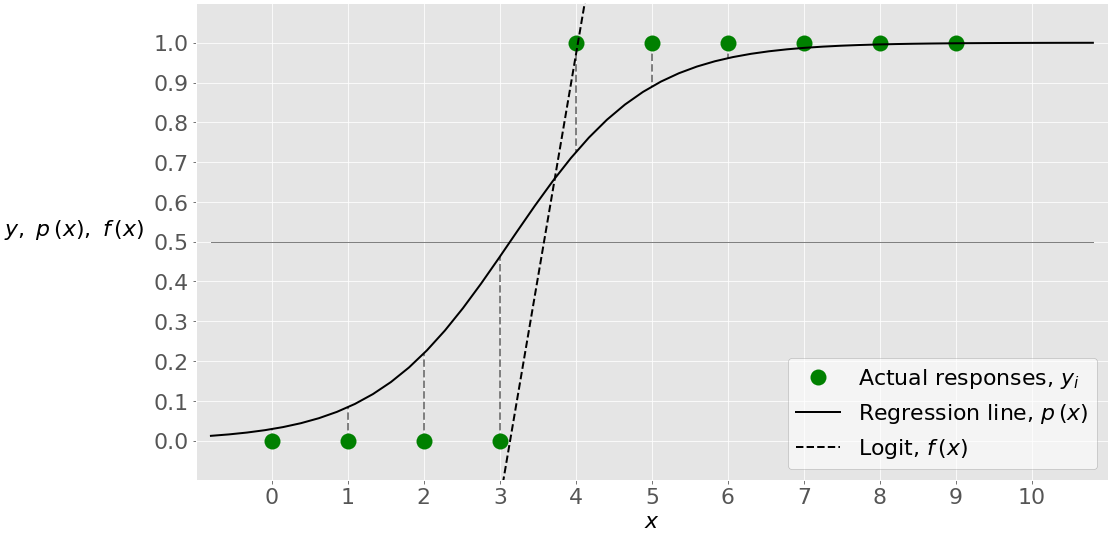
There’s one more important relationship between 𝑝(𝐱) and 𝑓(𝐱), which is that log(𝑝(𝐱) / (1 − 𝑝(𝐱))) = 𝑓(𝐱). This equality explains why 𝑓(𝐱) is the logit. It implies that 𝑝(𝐱) = 0.5 when 𝑓(𝐱) = 0 and that the predicted output is 1 if 𝑓(𝐱) > 0 and 0 otherwise.



Linear regression is estimated using Ordinary Least Squares (OLS) while logistic regression is estimated using Maximum Likelihood Estimation (MLE) approach. The MLE is a "likelihood" maximization method, while OLS is a distance-minimizing approximation method. Maximizing the likelihood function determines the parameters that are most likely to produce the observed data. From a statistical point of view, MLE sets the mean and variance as parameters in determining the specific parametric values for a given model. This set of parameters can be used for predicting the data needed in a normal distribution.

## Single-Variate Logistic Regression

Single-variate logistic regression is the most straightforward case of logistic regression. There is only one independent variable (or feature), which is 𝐱 = 𝑥. This figure illustrates single-variate logistic regression:

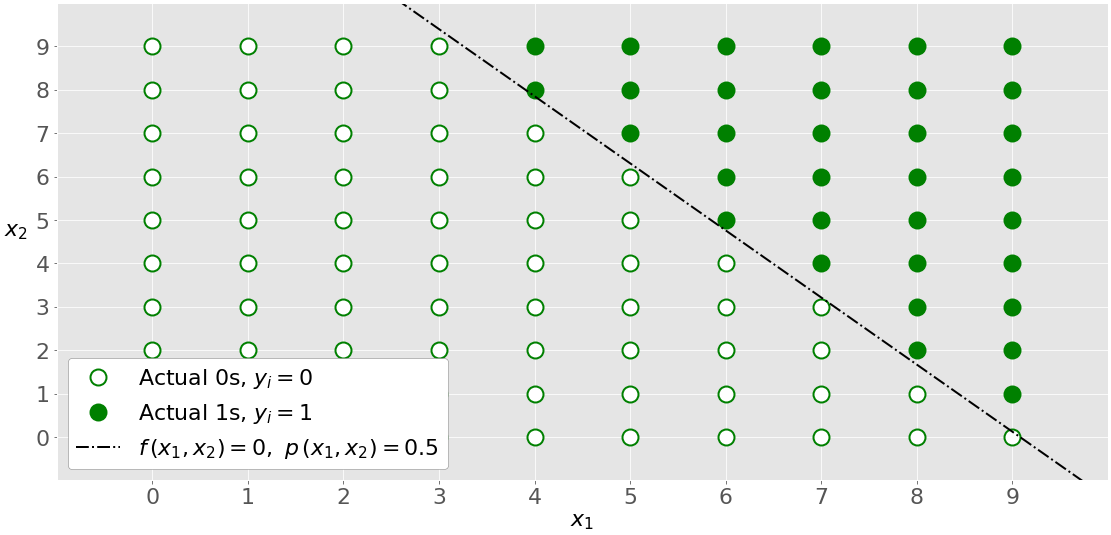
[](https://files.realpython.com/media/log-reg-2.e88a21607ba3.png)

Here, you have a given set of input-output (or 𝑥-𝑦) pairs, represented by green circles. These are your observations. Remember that 𝑦 can only be 0 or 1. For example, the leftmost green circle has the input 𝑥 = 0 and the actual output 𝑦 = 0. The rightmost observation has 𝑥 = 9 and 𝑦 = 1.

Logistic regression finds the weights 𝑏₀ and 𝑏₁ that correspond to the maximum LLF. These weights define the logit 𝑓(𝑥) = 𝑏₀ + 𝑏₁𝑥, which is the dashed black line. They also define the predicted probability 𝑝(𝑥) = 1 / (1 + exp(−𝑓(𝑥))), shown here as the full black line. In this case, the threshold 𝑝(𝑥) = 0.5 and 𝑓(𝑥) = 0 corresponds to the value of 𝑥 slightly higher than 3. This value is the limit between the inputs with the predicted outputs of 0 and 1.

## Multi-Variate Logistic Regression

Multi-variate logistic regression has more than one input variable. This figure shows the classification with two independent variables, 𝑥₁ and 𝑥₂:

[](https://files.realpython.com/media/log-reg-3.b1634d335c4f.png)

The graph is different from the single-variate graph because both axes represent the inputs. The outputs also differ in color. The white circles show the observations classified as zeros, while the green circles are those classified as ones.

Logistic regression determines the weights 𝑏₀, 𝑏₁, and 𝑏₂ that maximize the LLF. Once you have 𝑏₀, 𝑏₁, and 𝑏₂, you can get:

* The logit 𝑓(𝑥₁, 𝑥₂) = 𝑏₀ + 𝑏₁𝑥₁ + 𝑏₂𝑥₂
* The probabilities 𝑝(𝑥₁, 𝑥₂) = 1 / (1 + exp(−𝑓(𝑥₁, 𝑥₂)))

The dash-dotted black line linearly separates the two classes. This line corresponds to (𝑥₁, 𝑥₂) = 0.5 and 𝑓(𝑥₁, 𝑥₂) = 0.

# K Nearest Neighbor

K-Nearest Neighbors, or KNN for short, is a non-parametric, lazy learning algorithm. When we say a technique is non-parametric, it means that it does not make any assumptions about the underlying data. In other words, it makes its selection based off the proximity to other data points regardless of what feature the numerical values represent. Being a lazy learning algorithm implies that there is little to no training phase. ie There is no model to speak of other than holding the entire training dataset. Because no work is done until a prediction is required, KNN is often referred to as a lazy learning method.

## Pros and Cons of KNN

### Pros

* No assumptions about data
* Simple algorithm — easy to understand
* Can be used for classification and regression

### Cons

* High memory requirement — All of the training data must be present in memory in order to calculate the closest K neighbors
* Sensitive to irrelevant features
* Sensitive to the scale of the data since we’re computing the distance to the closest K points

## Algorithm

Step 1: Calculate Euclidean Distance.

Step 2: Get Nearest Neighbors & Pick a value for K (i.e. 5).

Step 3: Make Predictions.

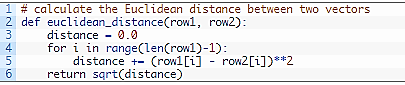
We can calculate the straight line distance between two vectors using the Euclidean distance measure. It is calculated as the square root of the sum of the squared differences between the two vectors.

Euclidean Distance = sqrt(sum i to N (x1\_i – x2\_i)^2)



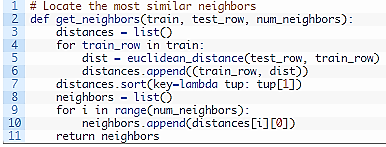
Where x1 is the first row of data, x2 is the second row of data and i is the index to a specific column as we sum across all columns. With Euclidean distance, the smaller the value, the more similar two records will be. A value of 0 means that there is no difference between two records.

Below is a function named euclidean\_distance() that implements this in Python as example.



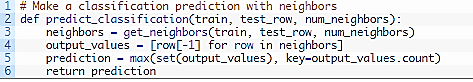
Neighbors for a new piece of data in the dataset are the k closest instances, as defined by our distance measure. To locate the neighbors for a new piece of data within a dataset we must first calculate the distance between each record in the dataset to the new piece of data. We can do this using our distance function prepared above. Once distances are calculated, we must sort all of the records in the training dataset by their distance to the new data. We can then select the top k to return as the most similar neighbors. We can do this by keeping track of the distance for each record in the dataset as a tuple, sort the list of tuples by the distance (in descending order) and then retrieve the neighbors.







The most similar neighbors collected from the training dataset can be used to make predictions. In the case of classification, we can return the most represented class among the neighbors. We can achieve this by performing the max() function on the list of output values from the neighbors. Given a list of class values observed in the neighbors, the max() function takes a set of unique class values and calls the count on the list of class values for each class value in the set.Below is the function named predict\_classification() that implements this.

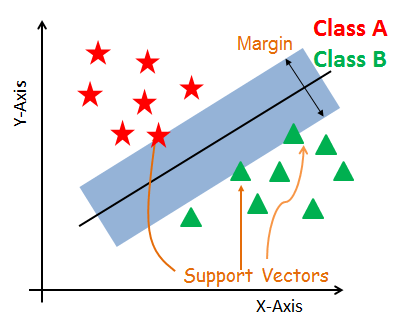


The K Nearest Neighbors algorithm doesn’t require any additional training when new data becomes available. Rather it determines the K closest points according to some distance metric (the samples must reside in memory). Then, it looks at the target label for each of the neighbors and places the new found data point into the same category as the majority. Given that KNN computes distance, it’s imperative that we scale our data. In addition, since KNN disregards the underlying features, it’s our responsibility to filter out any features that are deemed irrelevant.

# Support Vector Machine

SVM offers very high accuracy compared to other classifiers such as logistic regression, and decision trees. It known for its kernel trick to handle nonlinear input spaces. It is used in a variety of applications such as face detection, intrusion detection, classification of emails, news articles and web pages, classification of genes, and handwriting recognition.

Support Vector Machine (SVM) is a supervised machine-learning algorithm capable of performing classification, regression and even outlier detection.



## Support Vectors

Support vectors are the data points, which are closest to the hyperplane. These points will define the separating line better by calculating margins. These points are more relevant to the construction of the classifier.

## Hyperplane

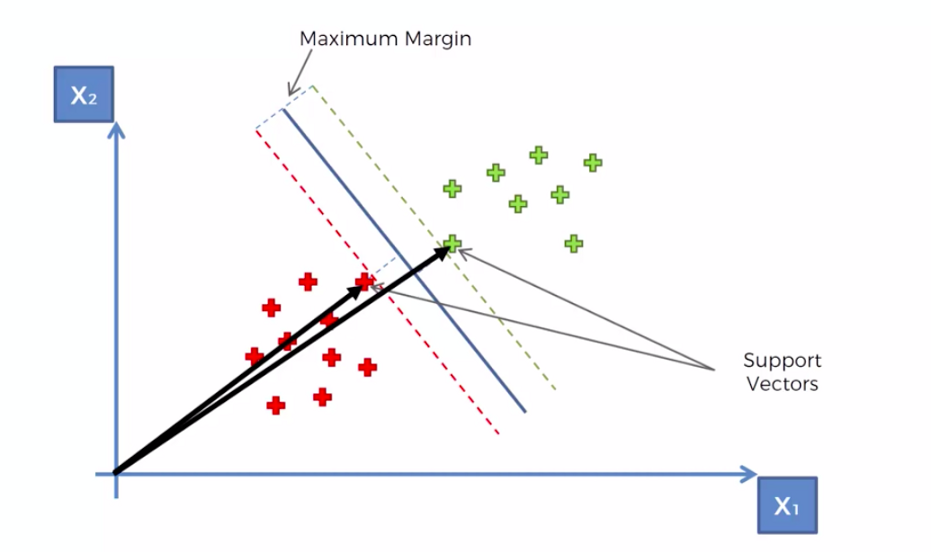
A hyperplane is a decision plane which separates between a set of objects having different class memberships.

## Margin

A margin is a gap between the two lines on the closest class points. This is calculated as the perpendicular distance from the line to support vectors or closest points. If the margin is larger in between the classes, then it is considered a good margin, a smaller margin is a bad margin.

## SVM for linearly separable

The linear SVM classifier works by drawing a straight line between two classes. All the data points that fall on one side of the line will be labeled as one class and all the points that fall on the other side will be labeled as the second. Sounds simple enough, but there’s an infinite amount of lines to choose from. How do we know which line will do the best job of classifying the data? This is where the LSVM algorithm comes in to play. The LSVM algorithm will select a line that not only separates the two classes but stays as far away from the closest samples as possible. In fact, the “support vector” in “support vector machine” refers to two position vectors drawn from the origin to the points which dictate the decision boundary.



 SVM constructs a hyperplane in multidimensional space to separate different classes. SVM generates optimal hyperplane in an iterative manner, which is used to minimize an error. The core idea of SVM is to find a maximum marginal hyperplane(MMH) that best divides the dataset into classes.

The main objective is to segregate the given dataset in the best possible way. The distance between the either nearest points is known as the margin. The objective is to select a hyperplane with the maximum possible margin between support vectors in the given dataset. SVM searches for the maximum marginal hyperplane in the following steps:

1. Generate hyperplanes which segregates the classes in the best way. Left-hand side figure showing three hyperplanes black, blue and orange. Here, the blue and orange have higher classification error, but the black is separating the two classes correctly.
2. Select the right hyperplane with the maximum segregation from the either nearest data points as shown in the right-hand side figure.

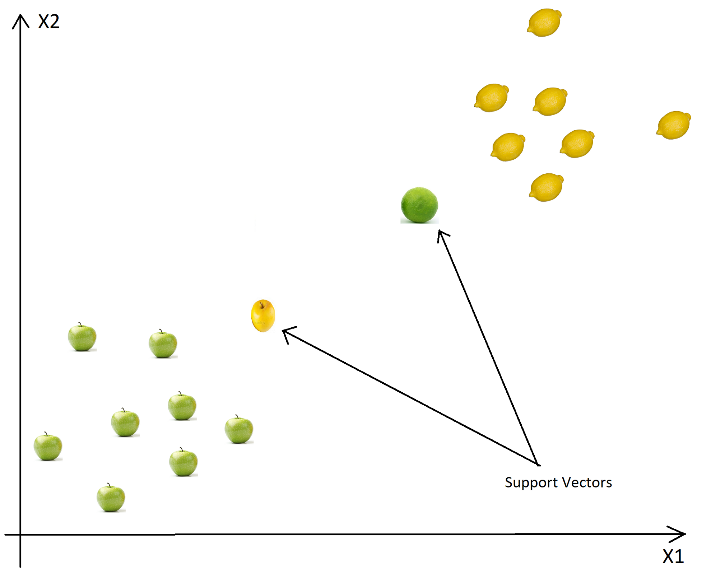


As an example, lets consider two classes, apples and lemons.

Other algorithms will learn the most evident, most representative characteristics of apples and lemons, like apples are green and rounded while lemons are yellow and have elliptic form.

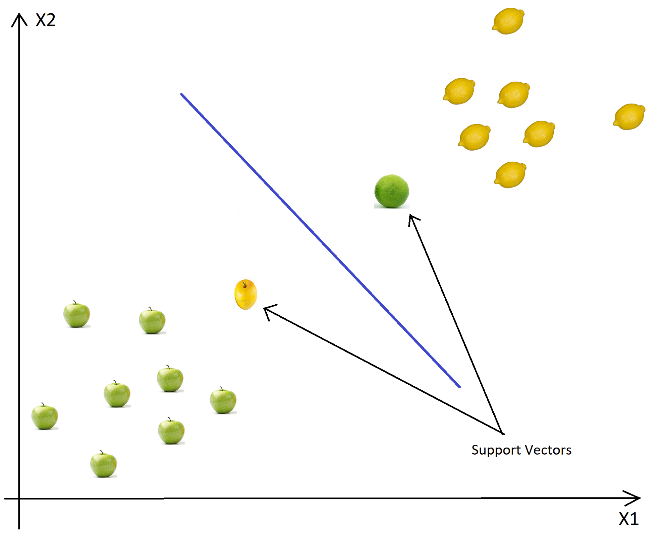
In contrast, SVM will search for apples that are very similar to lemons, for example apples which are yellow and have elliptic form. This will be a support vector. The other support vector will be a lemon similar to an apple (green and rounded). So other algorithms learns the differences while SVM learns similarities.

If we visualize the example above in 2D, we will have something like this:

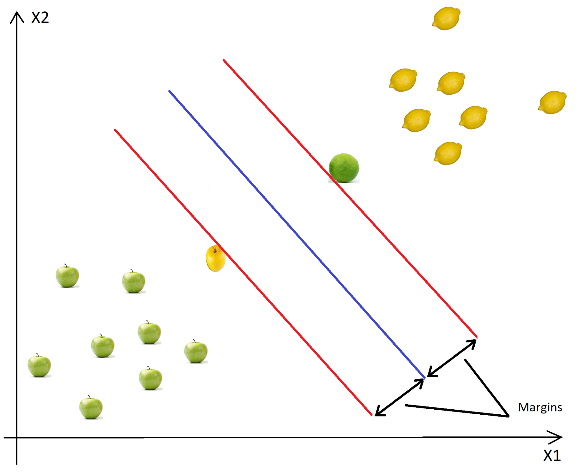


As we go from left to right, all the examples will be classified as apples until we reach the yellow apple. From this point, the confidence that a new example is an apple drops while the lemon class confidence increases. When the lemon class confidence becomes greater than the apple class confidence, the new examples will be classified as lemons (somewhere between the yellow apple and the green lemon).

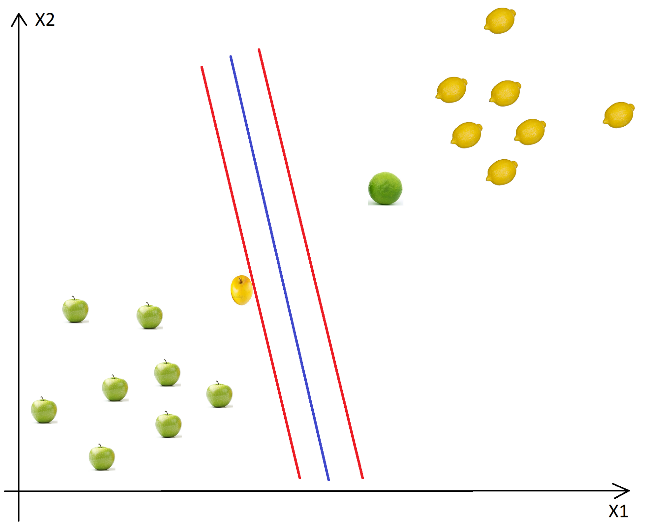
Based on these support vectors, the algorithm tries to find the best hyperplane that separates the classes. In 2D the hyperplane is a line, so it would look like this:



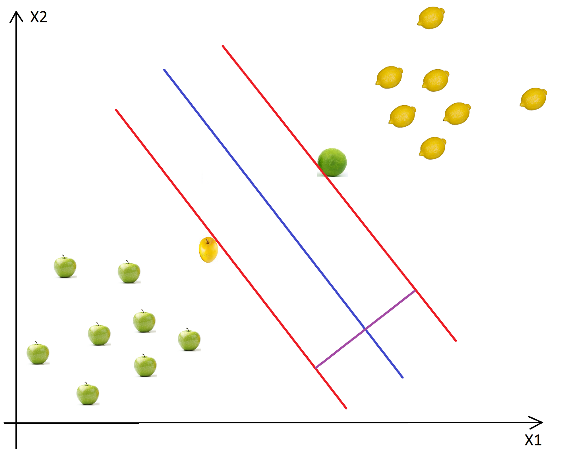
Intuitively the best line is the line that is far away from both apple and lemon examples (has the largest margin). To have optimal solution, we have to maximize the margin in both ways (if we have multiple classes, then we have to maximize it considering each of the classes).



So if we compare the picture above with the picture below, we can easily observe, that the first is the optimal hyperplane (line) and the second is a sub-optimal solution, because the margin is far shorter.



Because we want to maximize the margins taking in consideration all the classes, instead of using one margin for each class, we use a “global” margin, which takes in consideration all the classes. This margin would look like the purple line in the following picture:



This margin is orthogonal to the boundary and equidistant to the support vectors.

So where do we have vectors? Each of the calculations (calculate distance and optimal hyperplanes) are made in vectorial space, so each data point is considered a vector. The dimension of the space is defined by the number of attributes of the examples. To understand the math behind, please read this brief mathematical description of vectors, hyperplanes and optimizations.

All in all, support vectors are data points that defines the position and the margin of the hyperplane. We call them “support” vectors, because these are the representative data points of the classes, if we move one of them, the position and/or the margin will change. Moving other data points won’t have effect over the margin or the position of the hyperplane.

To make classifications, we don’t need all the training data points (like in the case of KNN), we have to save only the support vectors. In worst case all the points will be support vectors, but this is very rare and if it happens, then you should check your model for errors or bugs.

So basically the learning is equivalent with finding the hyperplane with the best margin, so it is a simple optimization problem.

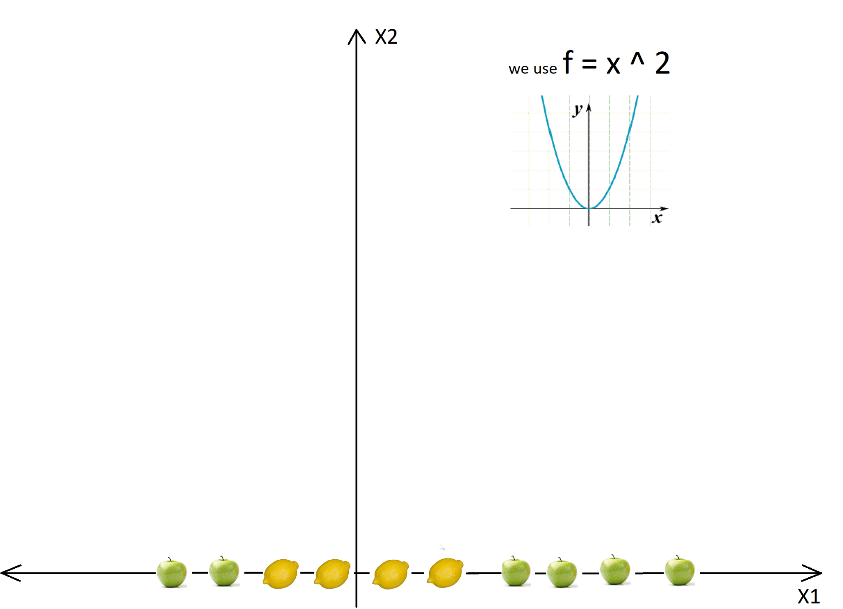
The basic steps of the SVM are:

1. select two hyperplanes (in 2D) which separates the data with no points between them (red lines)
2. maximize their distance (the margin)
3. the average line (here the line half way between the two red lines) will be the decision boundary

This is very nice and easy, but finding the best margin, the optimization problem is not trivial (it is easy in 2D, when we have only two attributes, but what if we have N dimensions with N a very big number)

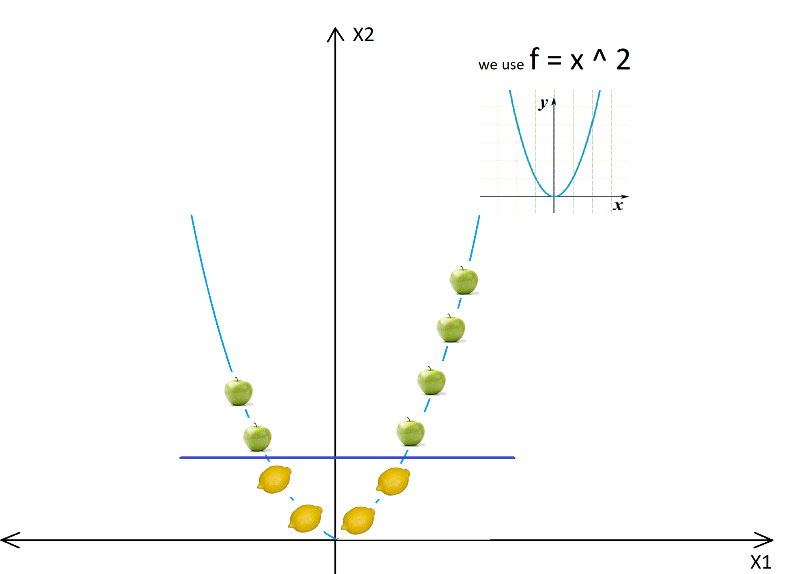
To solve the optimization problem, we use the Lagrange Multipliers.

## SVM for Non-Linear Data Sets



In above case we cannot find a straight line to separate apples from lemons. So how can we solve this problem. We will use the Kernel Trick. The basic idea is that when a data set is inseparable in the current dimensions, add another dimension, maybe that way the data will be separable.

To solve this problem we **should not just blindly add another dimension**, we should transform the space so we generate this level difference intentionally. So after using the kernel and after all the transformations we will get:



So after the transformation, we can easily delimit the two classes using just a single line.

Now we can easily separate the two classes. These transformations are called kernels. Popular kernels are: Polynomial Kernel, Gaussian Kernel, Radial Basis Function (RBF), Laplace RBF Kernel, Sigmoid Kernel, Anove RBF Kernel, etc.

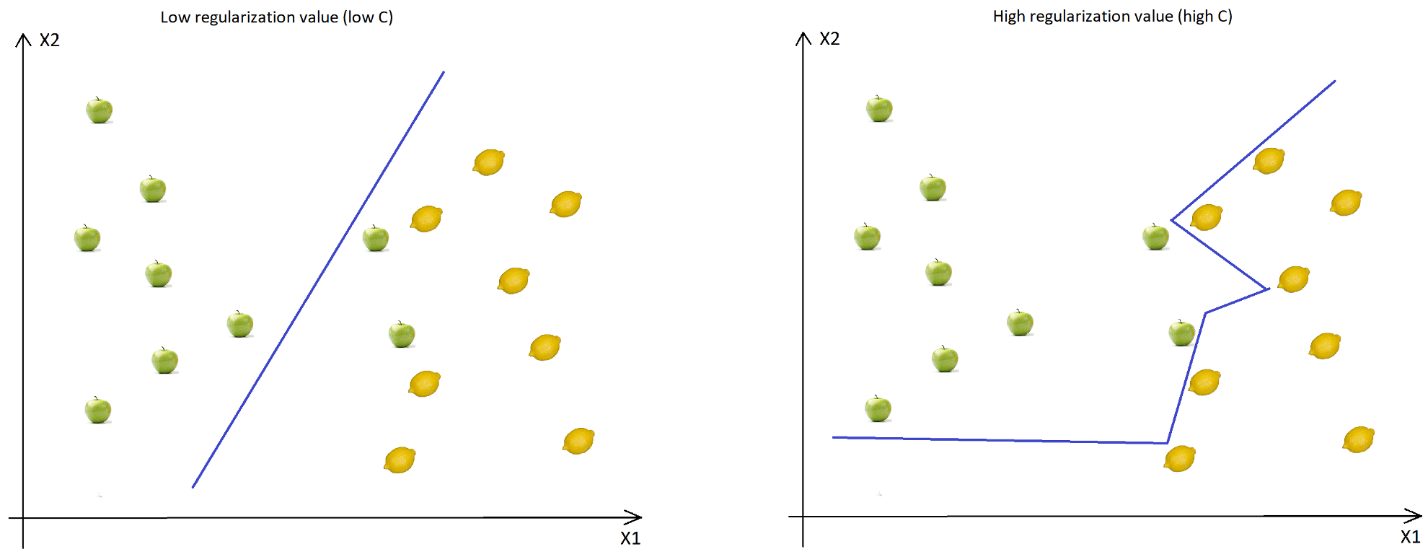
In real life applications we won’t have a simple straight line, but we will have lots of curves and high dimensions. In some cases we won’t have two hyperplanes which separates the data with no points between them, so we need some trade-offs, tolerance for outliers. Fortunately the SVM algorithm has a so-called regularization parameter to configure the trade-off and to tolerate outliers.

## Tuning Parameters

As we saw in the previous section choosing the right kernel is crucial, because if the transformation is incorrect, then the model can have very poor results. As a rule of thumb, always check if you have linear data and in that case always use linear SVM (linear kernel). Linear SVM is a parametric model, but an RBF kernel SVM isn’t, so the complexity of the latter grows with the size of the training set. Not only is more expensive to train an RBF kernel SVM, but you also have to keep the kernel matrix around, and the projection into this “infinite” higher dimensional space where the data becomes linearly separable is more expensive as well during prediction. Furthermore, you have more hyperparameters to tune, so model selection is more expensive as well! And finally, it’s much easier to overfit a complex model!

### Regularization

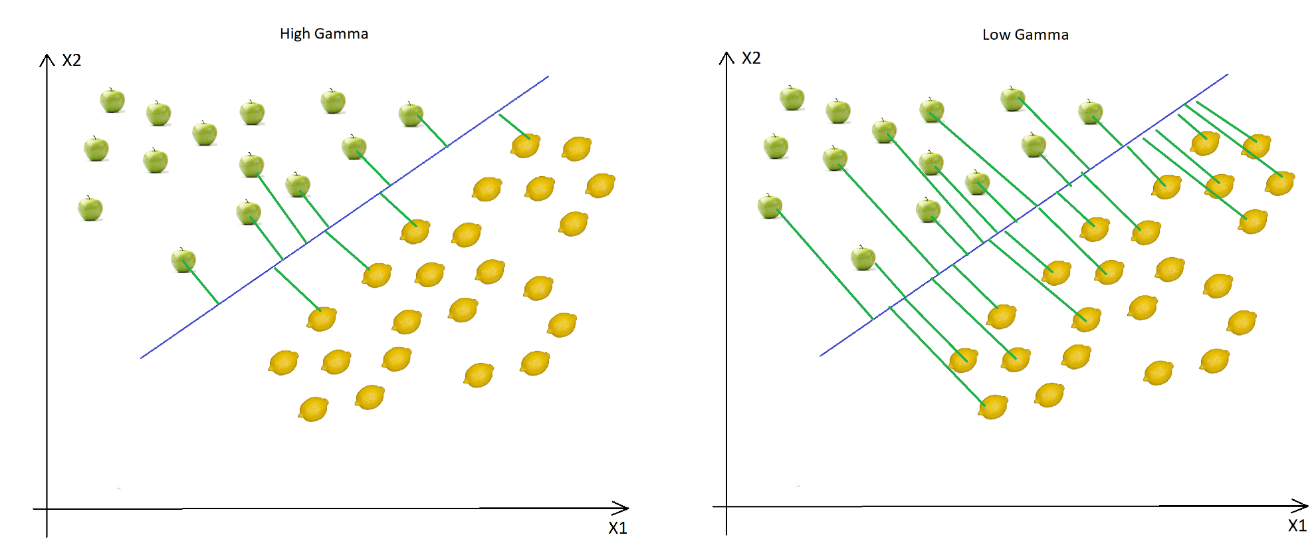
The Regularization Parameter (in python it’s called C) tells the SVM optimization how much you want to avoid miss classifying each training example. If the C is higher, the optimization will choose smaller margin hyperplane, so training data miss classification rate will be lower. On the other hand, if the C is low, then the margin will be big, even if there will be miss classified training data examples. This is shown in the following two diagrams



As you can see in the image, when the C is low, the margin is higher (so implicitly we don’t have so many curves, the line doesn’t strictly follows the data points) even if two apples were classified as lemons. When the C is high, the boundary is full of curves and all the training data was classified correctly. Don’t forget, even if all the training data was correctly classified, this doesn’t mean that increasing the C will always increase the precision (because of overfitting).

### Gamma

The next important parameter is Gamma. The gamma parameter defines how far the influence of a single training example reaches. This means that high Gamma will consider only points close to the plausible hyperplane and low Gamma will consider points at greater distance.



As you can see, decreasing the Gamma will result that finding the correct hyperplane will consider points at greater distances so more and more points will be used (green lines indicates which points were considered when finding the optimal hyperplane).

### Margin

The last parameter is the margin. Higher margin results better model, so better classification (or prediction). The margin should be always maximized.

## Pros

* SVN can be very efficient, because it uses only a subset of the training data, only the support vectors
* Works very well on smaller data sets, on non-linear data sets and high dimensional spaces
* Is very effective in cases where number of dimensions is greater than the number of samples
* It can have high accuracy, sometimes can perform even better than neural networks
* Not very sensitive to overfitting

## Cons

* Training time is high when we have large data sets
* When the data set has more noise (i.e. target classes are overlapping) SVM doesn’t perform well

## Popular Use Cases

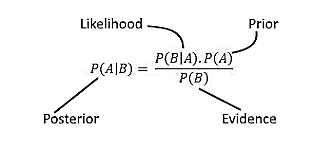
* Text Classification
* Detecting spam
* Sentiment analysis
* Aspect-based recognition
* Aspect-based recognition
* Handwritten digit recognition

# Naïve Bayes

Naive Bayes is the most straightforward and fast classification algorithm, which is suitable for a large chunk of data. Naive Bayes classifier is successfully used in various applications such as spam filtering, text classification, sentiment analysis, and recommender systems. It uses Bayes theorem of probability for prediction of unknown class.

## Bayes Theorem

Bayes’ Theorem is a way of finding a probability when we know certain other probabilities.The formula is:



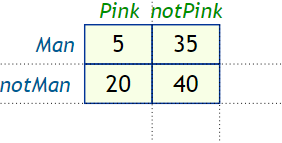
Which tells us: how often A happens *given that B happens*, written **P(A|B)**,When we know: how often B happens *given that A happens*, written **P(B|A)**  and how likely A is on its own, written **P(A)**  and how likely B is on its own, written **P(B)**

Let us say P(Fire) means how often there is fire, and P(Smoke) means how often we see smoke, then:

P(Fire|Smoke) means how often there is fire when we can see smoke  
P(Smoke|Fire) means how often we can see smoke when there is fire

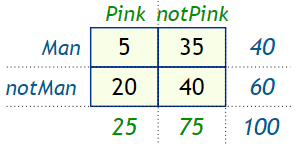
### Example

Imagine 100 people at a party, and you tally how many wear pink or not, and if a man or not, and get these numbers:



Bayes' Theorem is based off just those 4 numbers!

Let us do some totals:



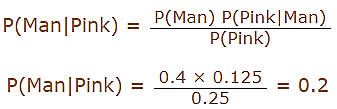
Calculate some probabilities:

* the probability of being a man is P(Man) =  = 0.4
* the probability of wearing pink is P(Pink) =  = 0.25
* the probability that a man wears pink is P(Pink|Man) =  = 0.125

The probability that a person wearing pink is a man **P (Man|Pink) =**

* P(Man) = 0.4,
* P(Pink) = 0.25 and
* P(Pink|Man) = 0.125

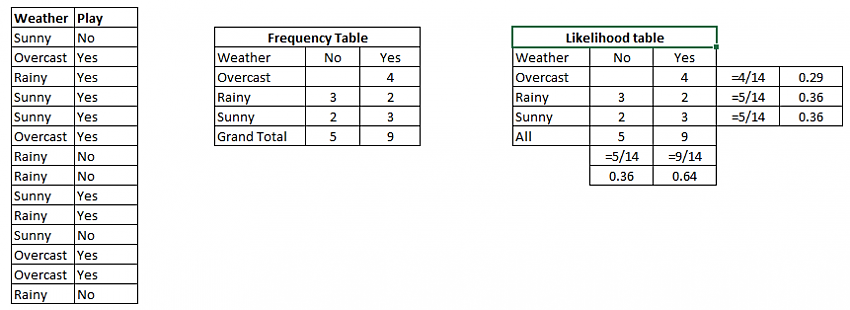
Then discover P (Man|Pink)



## Working of Naive Bayes algorithm

Let’s understand it using an example. Below I have a training data set of weather and corresponding target variable ‘Play’ (suggesting possibilities of playing). Now, we need to classify whether players will play or not based on weather condition. Let’s follow the below steps to perform it.

* Step 1: Convert the data set into a frequency table
* Step 2: Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.



* Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

**Problem:**Players will play if weather is sunny. Is this statement is correct?

We can solve it using above discussed method of posterior probability.

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

Here we have P (Sunny |Yes) = 3/9 = 0.33, P(Sunny) = 5/14 = 0.36, P( Yes)= 9/14 = 0.64

Now, P (Yes | Sunny) = 0.33 \* 0.64 / 0.36 = 0.60, which has higher probability.

Naive Bayes uses a similar method to predict the probability of different class based on various attributes. This algorithm is mostly used in text classification and with problems having multiple classes.

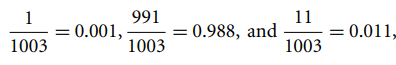
## Zero Probability Problem

Suppose there is no tuple for a risky loan in the dataset, in this scenario, the posterior probability will be zero, and the model is unable to make a prediction. This problem is known as Zero Probability because the occurrence of the particular class is zero.

The solution for such an issue is the Laplacian correction or Laplace Transformation. Laplacian correction is one of the smoothing techniques. Here, you can assume that the dataset is large enough that adding one row of each class will not make a difference in the estimated probability. This will overcome the issue of probability values to zero.

For Example: Suppose that for the class loan risky, there are 1000 training tuples in the database. In this database, income column has 0 tuples for low income, 990 tuples for medium income, and 10 tuples for high income. The probabilities of these events, without the Laplacian correction, are 0, 0.990 (from 990/1000), and 0.010 (from 10/1000)

Now, apply Laplacian correction on the given dataset. Let us add 1 more tuple for each income-value pair. The probabilities of these events:



## Advantages

* It is not only a simple approach but also a fast and accurate method for prediction.
* Naive Bayes has very low computation cost.
* It can efficiently work on a large dataset.
* It performs well in case of discrete response variable compared to the continuous variable.
* It can be used with multiple class prediction problems.
* It also performs well in the case of text analytics problems.
* When the assumption of independence holds, a Naive Bayes classifier performs better compared to other models like logistic regression.

## Disadvantages

* The assumption of independent features. In practice, it is almost impossible that model will get a set of predictors which are entirely independent.
* If there is no training tuple of a particular class, this causes zero posterior probability. In this case, the model is unable to make predictions. This problem is known as Zero Probability/Frequency Problem.

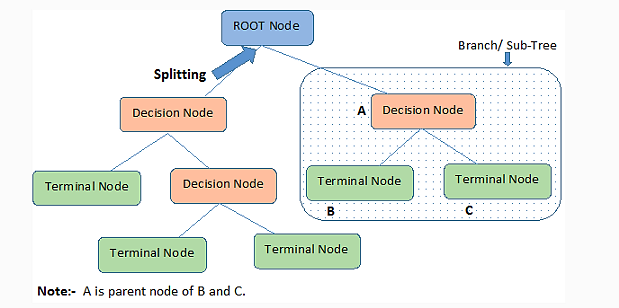
# Decision Tree

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving regression and classification problems too.

The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from prior data(training data).

## Important Terminology related to Decision Trees

* **Root Node**: It represents the entire population or sample and this further gets divided into two or more homogeneous sets.
* **Splitting**: It is a process of dividing a node into two or more sub-nodes.
* **Decision Node**: When a sub-node splits into further sub-nodes, then it is called the decision node.
* **Leaf / Terminal Node**: Nodes do not split is called Leaf or Terminal node.
* **Pruning**: When we remove sub-nodes of a decision node, this process is called pruning. You can say the opposite process of splitting.
* **Branch / Sub-Tree**: A subsection of the entire tree is called branch or sub-tree.
* **Parent and Child Node**: A node, which is divided into sub-nodes is called a parent node of sub-nodes whereas sub-nodes are the child of a parent node.



Each node in the tree acts as a test case for some attribute, and each edge descending from the node corresponds to the possible answers to the test case. This process is recursive in nature and is repeated for every subtree rooted at the new node.

## Assumptions while creating Decision Tree

Below are some of the assumptions we make while using Decision tree:

* In the beginning, the whole training set is considered as the root.
* Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
* 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 primary challenge in the decision tree implementation is to identify which attributes do we need to consider as the root node and each level. Handling this is to know as the attributes selection. We have different attributes selection measures to identify the attribute which can be considered as the root note at each level.

## Attribute Selection Measures

If the dataset consists of N attributes then deciding which attribute to place at the root or at different levels of the tree as internal nodes is a complicated step. By just randomly selecting any node to be the root can’t solve the issue. If we follow a random approach, it may give us bad results with low accuracy.

For solving this attribute selection problem, researchers worked and devised some solutions. They suggested using some criteria like:

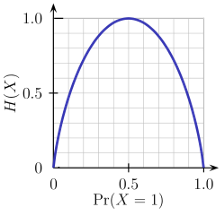
* Entropy,
* Information gain,
* Gini index,
* Gain Ratio,
* Reduction in Variance
* Chi-Square

These criterions will calculate values for every attribute. The values are sorted, and attributes are placed in the tree by following the order i.e, the attribute with a high value (in case of information gain) is placed at the root.

While using Information **Gain as a criterion, we assume attributes to be categorical**, and for the **Gini index, attributes are assumed to be continuous**.

### Entropy

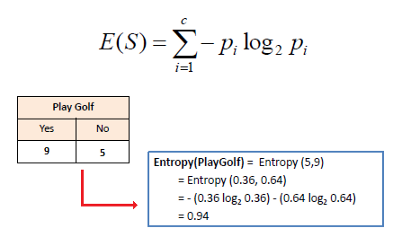
Entropy is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information. Flipping a coin is an example of an action that provides information that is random.



From the above graph, it is quite evident that the entropy H(X) is zero when the probability is either 0 or The Entropy is maximum when the probability is 0.5 because it projects perfect randomness in the data and there is no chance if perfectly determining the outcome.

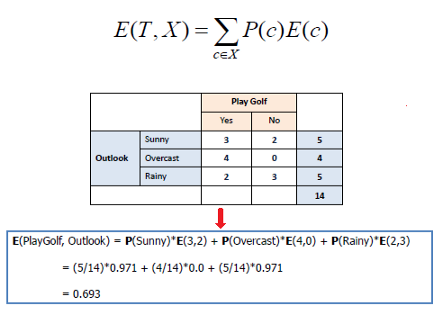
**ID3 follows the rule — A branch with an entropy of zero is a leaf node and A brach with entropy more than zero needs further splitting.**

Mathematically Entropy for 1 attribute is represented as:



Where S → Current state, and Pi → Probability of an event i of state S or Percentage of class i in a node of state S.

Mathematically Entropy for multiple attributes is represented as:



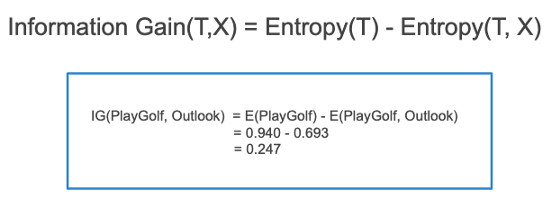
where T→ Current state and X → Selected attribute

### Information Gain

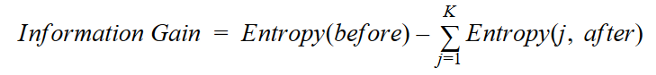
Information gain or IG is a statistical property that measures how well a given attribute separates the training examples according to their target classification. Constructing a decision tree is all about finding an attribute that returns the highest information gain and the smallest entropy.

Information gain is a decrease in entropy. It computes the difference between entropy before split and average entropy after split of the dataset based on given attribute values. ID3 (Iterative Dichotomiser) decision tree algorithm uses information gain.

Mathematically, IG is represented as:



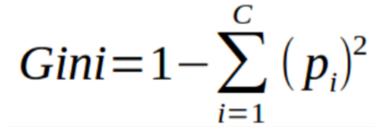
In a much simpler way, we can conclude that:



Where “before” is the dataset before the split, K is the number of subsets generated by the split, and (j, after) is subset j after the split.

### Gini Index

You can understand the Gini index as a cost function used to evaluate splits in the dataset. It is calculated by subtracting the sum of the squared probabilities of each class from one. It favors larger partitions and easy to implement whereas information gain favors smaller partitions with distinct values.



Gini Index works with the categorical target variable “Success” or “Failure”. It performs only Binary splits.

Higher the value of Gini index higher the homogeneity.

**Steps to Calculate Gini index for a split**

1. Calculate Gini for sub-nodes, using the above formula for success(p) and failure(q) (p²+q²).
2. Calculate the Gini index for split using the weighted Gini score of each node of that split.

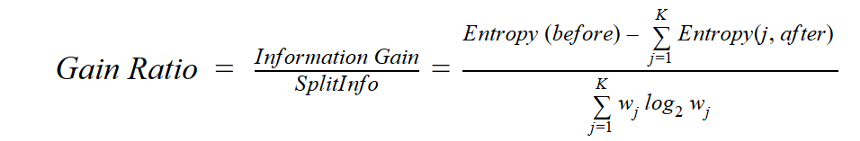
CART (Classification and Regression Tree) uses the Gini index method to create split points.

### Gain ratio

Information gain is biased towards choosing attributes with a large number of values as root nodes. It means it prefers the attribute with a large number of distinct values.

C4.5, an improvement of ID3, uses Gain ratio which is a modification of Information gain that reduces its bias and is usually the best option. Gain ratio overcomes the problem with information gain by taking into account the number of branches that would result before making the split. It corrects information gain by taking the intrinsic information of a split into account.

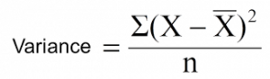
Let us consider if we have a dataset that has users and their movie genre preferences based on variables like gender, group of age, rating, blah, blah. With the help of information gain, you split at ‘Gender’ (assuming it has the highest information gain) and now the variables ‘Group of Age’ and ‘Rating’ could be equally important and with the help of gain ratio, it will penalize a variable with more distinct values which will help us decide the split at the next level.



Where “before” is the dataset before the split, K is the number of subsets generated by the split, and (j, after) is subset j after the split.

### Reduction in Variance

Reduction in variance is an algorithm used for continuous target variables (regression problems). This algorithm uses the standard formula of variance to choose the best split. The split with lower variance is selected as the criteria to split the population:



Above X-bar is the mean of the values, X is actual and n is the number of values.

Steps to calculate Variance:

1. Calculate variance for each node.
2. Calculate variance for each split as the weighted average of each node variance.

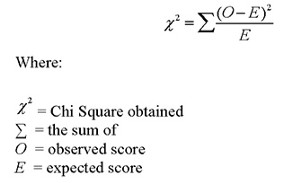
### Chi-Square

The acronym CHAID stands for Chi-squared Automatic Interaction Detector. It is one of the oldest tree classification methods. It finds out the statistical significance between the differences between sub-nodes and parent node. We measure it by the sum of squares of standardized differences between observed and expected frequencies of the target variable.

It works with the categorical target variable “Success” or “Failure”. It can perform two or more splits. Higher the value of Chi-Square higher the statistical significance of differences between sub-node and Parent node.

It generates a tree called CHAID (Chi-square Automatic Interaction Detector).

Mathematically, Chi-squared is represented as:



Steps to Calculate Chi-square for a split:

1. Calculate Chi-square for an individual node by calculating the deviation for Success and Failure both
2. Calculated Chi-square of Split using Sum of all Chi-square of success and Failure of each node of the split

Working of Decision Trees

The decision of making strategic splits heavily affects a tree’s accuracy. The decision criteria 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 → (extension of 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)

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.

Steps in ID3 algorithm:

1. It begins with the original set S as the root node.
2. 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.
3. It then selects the attribute which has the smallest Entropy or Largest Information gain.
4. The set S is then split by the selected attribute to produce a subset of the data.
5. The algorithm continues to recur on each subset, considering only attributes never selected before.

## How to avoid/counter Overfitting in Decision Trees

The common problem with Decision trees, especially having a table full of columns, they fit a lot. Sometimes it looks like the tree memorized the training data set. If there is no limit set on a decision tree, it will give you 100% accuracy on the training data set because in the worse case it will end up making 1 leaf for each observation. Thus this affects the accuracy when predicting samples that are not part of the training set.

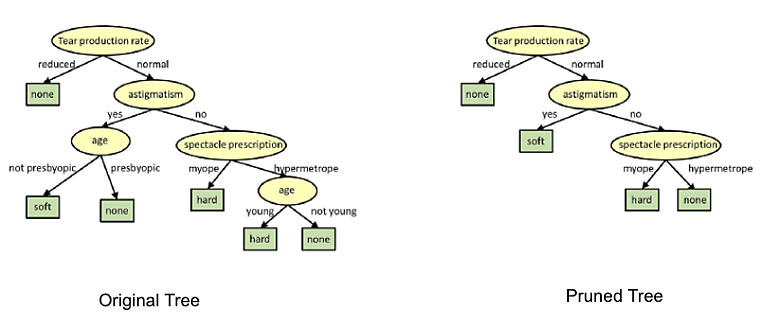
Here are two ways to remove overfitting:

* Pruning Decision Trees.
* Random Forest

### Pruning Decision Trees

The splitting process results in fully grown trees until the stopping criteria are reached. But, the fully grown tree is likely to overfit the data, leading to poor accuracy on unseen data.

In pruning, you trim off the branches of the tree, i.e., remove the decision nodes starting from the leaf node such that the overall accuracy is not disturbed. This is done by segregating the actual training set into two sets: training data set, D and validation data set, V. Prepare the decision tree using the segregated training data set, D. Then continue trimming the tree accordingly to optimize the accuracy of the validation data set, V.



In the above diagram, the ‘Age’ attribute in the left-hand side of the tree has been pruned as it has more importance on the right-hand side of the tree, hence removing overfitting.

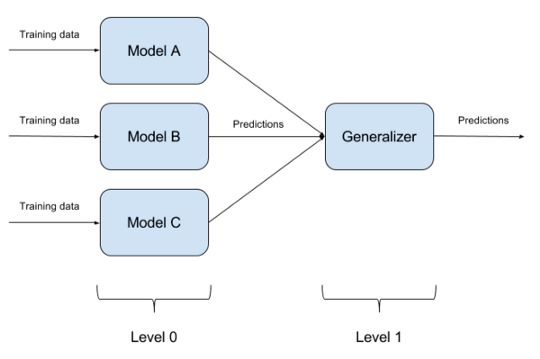
# Random Forest

Random Forest is an example of ensemble learning, in which we combine multiple machine learning algorithms to obtain better predictive performance. The common problem with Decision trees, especially having a table full of columns, they fit a lot. Sometimes it looks like the tree memorized the training data set. If there is no limit set on a decision tree, it will give you 100% accuracy on the training data set because in the worse case it will end up making 1 leaf for each observation. Thus this affects the accuracy when predicting samples that are not part of the training set.

## Ensemble Learning algorithms

Ensemble learning algorithms are meta-algorithms that combine several machine learning algorithms into one predictive model in order to decrease variance, bias or improve predictions.

The algorithm can be any machine learning algorithm such as logistic regression, decision tree, etc. These models, when used as inputs of ensemble methods, are called ”base models”.



Ensemble methods usually produce more accurate solutions than a single model would. This has been the case in a number of machine learning competitions, where the winning solutions used ensemble methods. In the popular Netflix Competition, the winner used an ensemble method to implement a powerful collaborative filtering algorithm. Another example is KDD 2009 where the winner also used ensemble methods.

Ensemble algorithms or methods can be divided into two groups:

1. Sequential ensemble methods — where the base learners are generated sequentially (e.g. AdaBoost).The basic motivation of sequential methods is to exploit the dependence between the base learners. The overall performance can be boosted by weighing previously mislabeled examples with higher weight.
2. Parallel ensemble methods — where the base learners are generated in parallel (e.g. Random Forest).

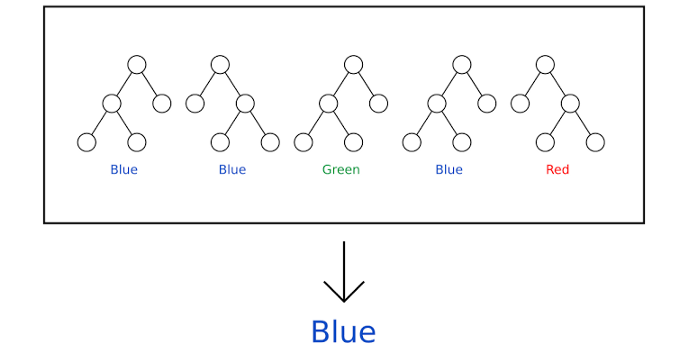
The basic motivation of parallel methods is to exploit independence between the base learners since the error can be reduced dramatically by averaging.

Most ensemble methods use a single base learning algorithm to produce homogeneous base learners, i.e. learners of the same type, leading to homogeneous ensembles.

There are also some methods that use heterogeneous learners, i.e. learners of different types, leading to heterogeneous ensembles. In order for ensemble methods to be more accurate than any of its individual members, the base learners have to be as accurate as possible and as diverse as possible.

## Random Forest algorithm

Random forest is a supervised ensemble learning algorithm that is used for both classifications as well as regression problems. But however, it is mainly used for classification problems. As we know that a forest is made up of trees and more trees mean more robust forest. Similarly, the random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method that is better than a single decision tree because it reduces the over-fitting by averaging the result.



The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds.

**“A large number of relatively uncorrelated models(trees) operating as a committee will outperform any of the individual constituent models.”**

The low correlation between models is the key.

The reason why Random forest produces exceptional results is that the trees protect each other from their individual errors. While some trees may be wrong, many others will be right, so as a group the trees are able to move in the correct direction.

Why the name “Random”?

Two key concepts that give it the name random:

* A random sampling of training data set when building trees.
* Random subsets of features considered when splitting nodes.

How is Random Forest ensuring Model diversity?

Random forest ensures that the behavior of each individual tree is not too correlated with the behavior of any other tree in the model by using the following two methods:

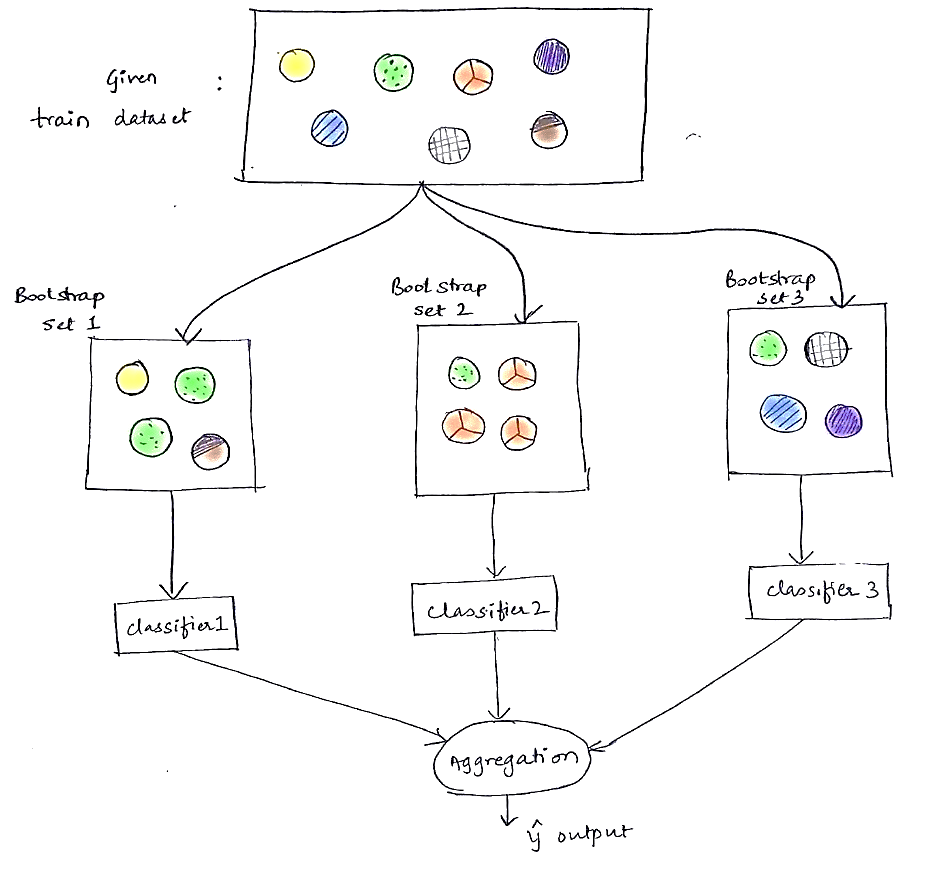
* Bagging or Bootstrap Aggregation
* Random feature selection

## Bagging or Bootstrap Aggregation

Decision trees are very sensitive to the data they are trained on, small changes to the training data set can result in a significantly different tree structure. The random forest takes advantage of this by allowing each individual tree to randomly sample from the dataset with replacement, resulting in different trees. This process is called Bagging.

Note that with bagging we are not subsetting the training data into smaller chunks and training each tree on a different chunk. Rather, if we have a sample of size N, we are still feeding each tree a training set of size N. But instead of the original training data, we take a random sample of size N with replacement.

For example — If our training data is [1,2,3,4,5,6], then we might give one of our trees the list [1,2,2,3,6,6] and we can give another tree a list [2,3,4,4,5,6]. Notice that the lists are of length 6 and some elements are repeated in the randomly selected training data we can give to our tree(because we sample with replacement).

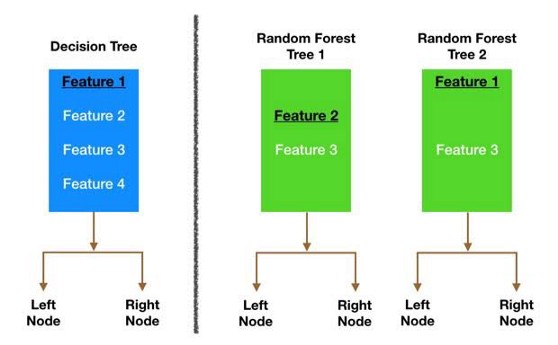


The above figure shows how random samples are taken from the dataset with replacement.

## Random feature selection

In a normal decision tree, when it is time to split a node, we consider every possible feature and pick the one that produces the most separation between the observations in the left node vs right node. In contrast, each tree in a random forest can pick only from a random subset of features. This forces even more variation amongst the trees in the model and ultimately results in low correlation across trees and more diversification.

So in random forest, we end up with trees that are trained on different sets of data and also use different features to make decisions.



And finally, uncorrelated trees have created that buffer and predict each other from their respective errors.

Random Forest creation pseudocode:

1. Randomly select “k” features from total “m” features where k << m
2. Among the “k” features, calculate the node “d” using the best split point
3. Split the node into daughter nodes using the best split
4. Repeat the 1 to 3 steps until “l” number of nodes has been reached
5. Build forest by repeating steps 1 to 4 for “n” number times to create “n” number of trees.