

MACHINE LEARNING

1. SUPERVISED AND UNSUPERVISED ALGORITHMS

Supervised learning

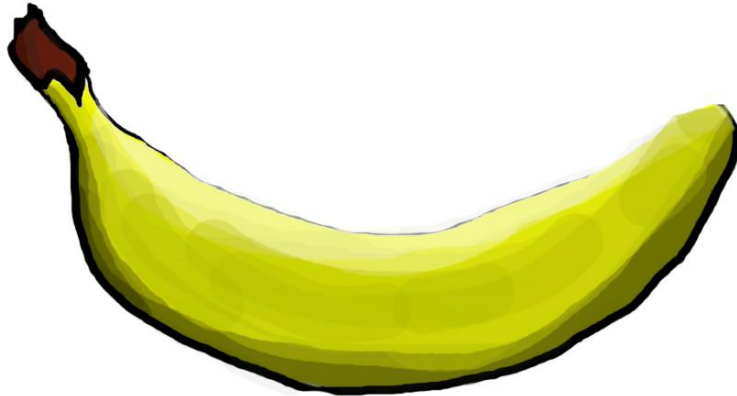
Supervised learning as the name indicates a presence of supervisor as teacher. Basically, supervised learning is a learning in which we teach or train the machine using data which is well labelled that means some data is already tagged with correct answer. After that, machine is provided with new set of examples(data) so that supervised learning algorithm analyses the training data (set of training examples) and produces an correct outcome from labelled data. This is also called as predictive modelling.

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



- If shape of object is rounded and depression at top having color Red then it will be labelled as –**Apple**.
- If shape of object is long curving cylinder having color Green-Yellow then it will be labelled as –**Banana**.

Now suppose after training the data, you have given a new separate fruit say Banana from basket and asked to identify it.



Since machine has already learnt the things from previous data and this time have to use it wisely. It will first classify the fruit with its shape and color, and would confirm the fruit name as BANANA and put it in Banana category. Thus machine learns the things from training data(basket containing fruits) and then apply the knowledge to test data(new fruit).

Supervised learning classified into two categories of algorithms:

- **Classification:** A classification problem is when the output variable is a category, such as “Red” or “blue” or “disease” and “no disease”.
- **Regression:** A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

Unsupervised learning

Unsupervised learning is the training of machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data.

Unlike supervised learning, no teacher is provided that means no training will be given to the machine. Therefore machine is restricted to find the hidden structure in unlabeled data by our-self.

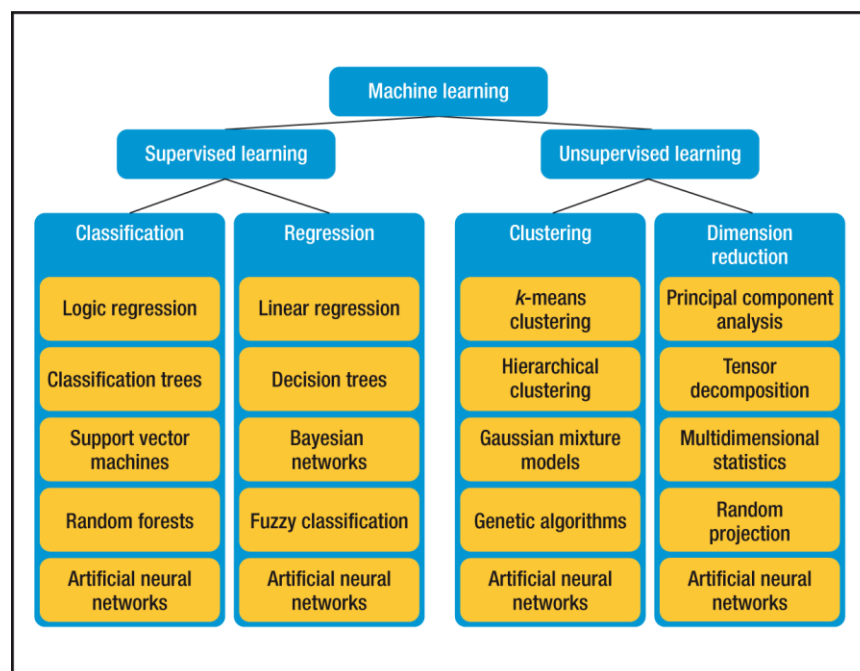
For instance, suppose it is given an image having both dogs and cats which have not seen ever.

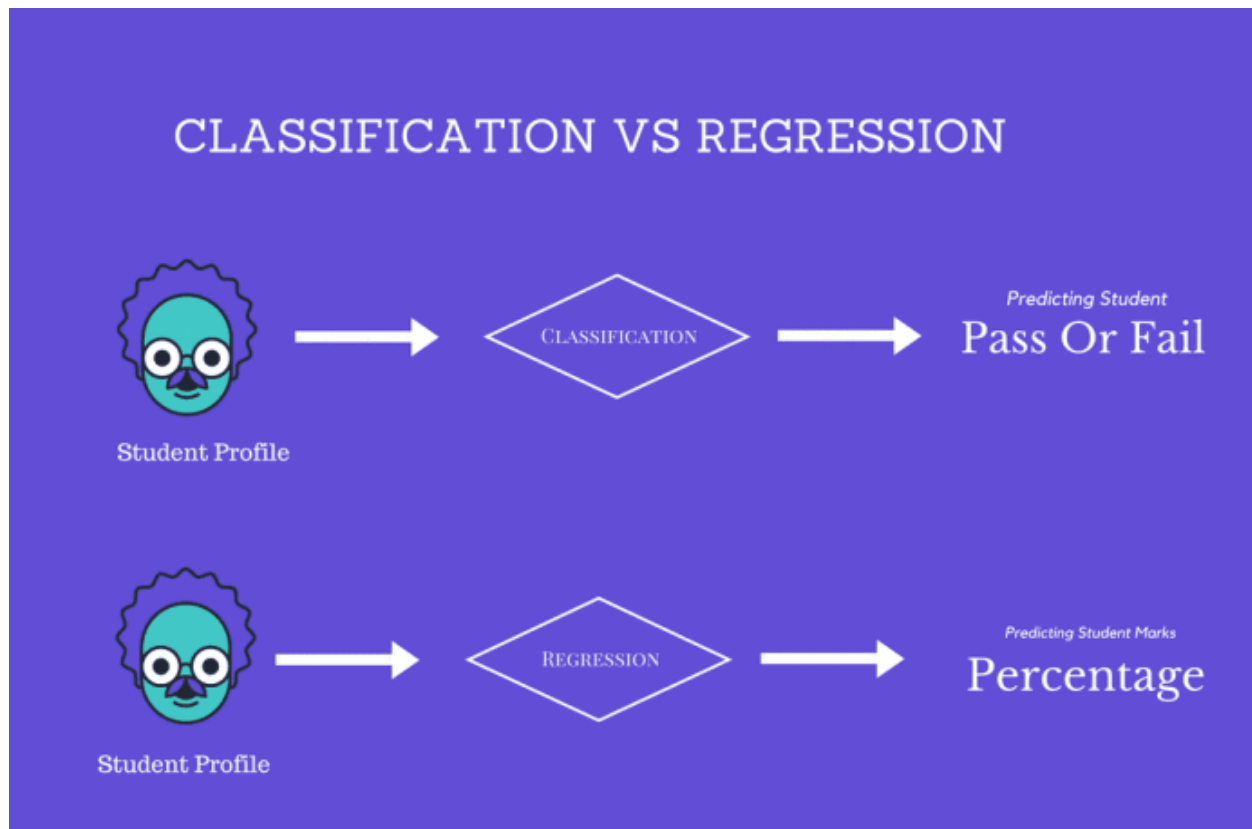


Thus, machine has no any idea about the features of dogs and cat so we can't categorize it in dogs and cats. But it can categorize them according to their similarities, patterns and differences i.e., we can easily categorize the above picture into two parts. First may contain all pics having **dogs** in it and second part may contain all pics having **cats** in it. Here you didn't learn anything before, means no training data or examples.

Unsupervised learning classified into two categories of algorithms:

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





Classification

The main goal of classification is to predict the target class (Yes/ No). If the trained model is for predicting any of two target classes. It is known as binary classification. Considering the student profile to predict whether the student will pass or fail. Considering the customer, transaction details to predict whether he will buy the new product or not. These kind problems will be addressed with binary classification. If we have to predict more the two target classes it is known as multi-classification. Considering all subject details of a student to predict which subject the student will score more. Identifying the object in an image. These kind problems are known as multi-classification problems.

Regression

The main goal of regression algorithms is the predict the discrete or a continues value. In some cases, the predicted value can be used to identify the linear relationship between the attributes. Suppose the increase in the product advantage budget will increase the product sales. Based on the problem difference regression

algorithms can be used. some of the basic regression algorithms are linear regression, polynomial regression ... etc



Classification Example:

Suppose from your past data (**train data**) you come to know that your best friend likes the above movies. Now one new movie (**test data**) released. Hopefully, you want to know your best friend like it or not. If you strongly confirmed about the chances of your friend like the move. You can take your friend to a movie this weekend.

If you clearly observe the problem it is just whether your friend **like or not**. Finding a solution to this type of problem is called as **classification**. This is because we are classifying the things to their belongings (**yes or no, like or dislike**). Keep in mind here we are forecasting **target** class(classification) and the other thing this classification belongs to **Supervised learning**. This is because you are **learning** this from your **train data**.

In this case, the problem is a **binary classification** in which we have to predict whether output belongs to class 1 or class 2 (**class 1 : yes, class 2: no**). As we have discussed earlier we can use classification for predicting more classes too. Like (**Color Prediction: RED, GREEN, BLUE, YELLOW, ORANGE**)

Regression Example:

Suppose from your past data (**train data**) you come to know that your best friend likes the above movies. You also know how many times each particular movie seen by your friend. Now one new movie (**test data**) released. Now you are going to find how many times this newly released movie will your friend watch. It could be 5 times, 6 times, 10 times etc...

If you clearly observe the problem is about finding the **count**, sometimes we can say this as **predicting the value**. Keep in mind, here we are forecasting a **value** (Prediction) and the other thing this prediction also belongs to **Supervised learning**. This is because you are **learning** this from your **train data**.

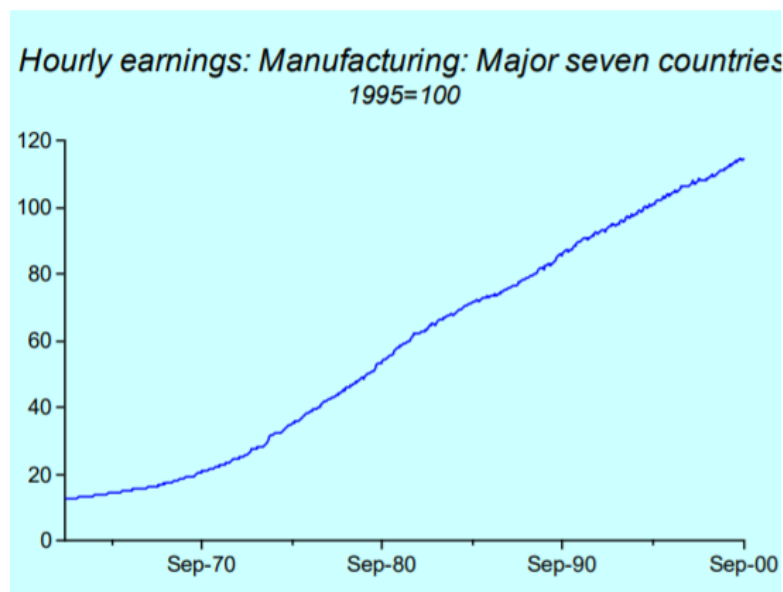
Summary

- If forecasting **target** class (Classification)
- If forecasting a **value** (Regression)

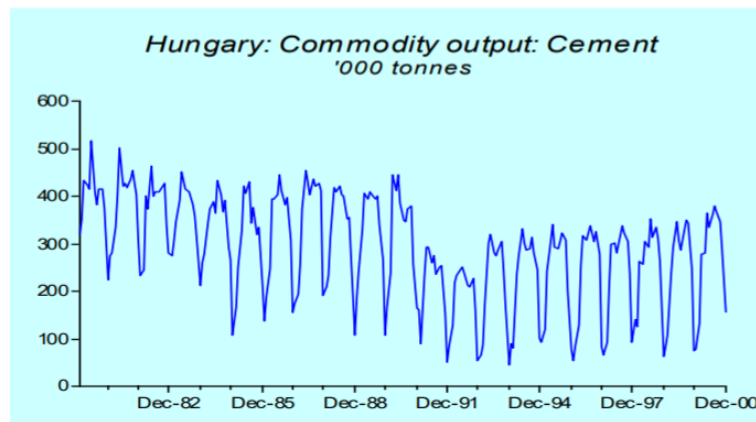
1.2 TIME SERIES ANALYSIS

Definition of Time Series: *An ordered sequence of values of a variable at equally spaced time intervals.*

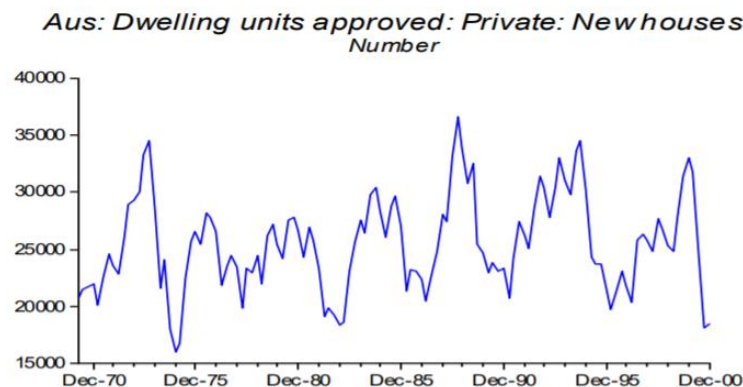
Trend: The long-term general change in the level of the data with a duration of longer than a year. It can be linear, non-linear, i.e. exponential, quadratic



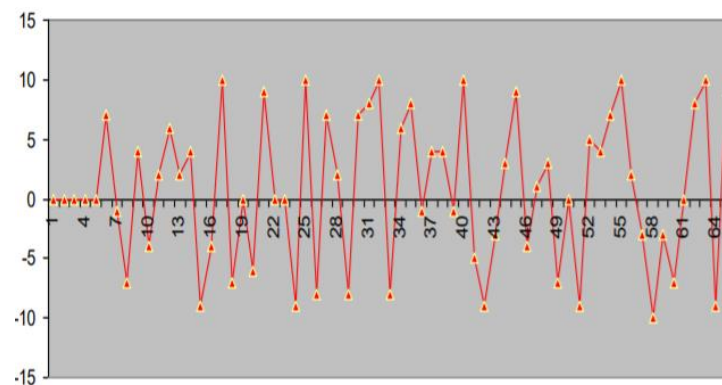
Seasonal variations: Regular wavelike fluctuations of constant length, repeating themselves within a period of no longer than a year.



Cyclical variations: Wavelike movements, quasi regular fluctuations around the long-term trend, lasting longer than a year. Example: Business cycles



Irregular Component: The random variations of the data comprise the deviations of the observed time series from the underlying pattern.



- The four components of a time series (T : trend, S : seasonal, C : cyclical, R : random) can be combined in different ways. Accordingly, the time series model used to describe the observed data (Y) can be

Additive:

$$Y_t = T_t + S_t + C_t + R_t$$

Multiplicative:

$$Y_t = T_t \times S_t \times C_t \times R_t$$

E.g.: If the trend is linear, these two models look as follows:

$$Y_t = (a + bt) + S_t + C_t + R_t$$

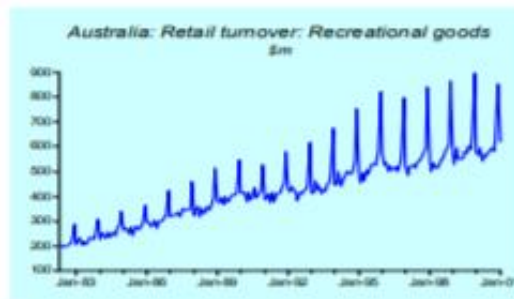
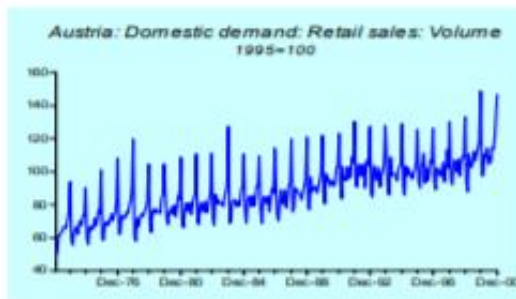
$$Y_t = (a + bt) \times S_t \times C_t \times R_t$$

In an additive model the seasonal, cyclical and random variations are *absolute* deviations from the trend.

In a multiplicative model the seasonal, cyclical and random variations are *relative* (percentage) deviations from the trend.

→ They do not depend on the level of the trend.

→ The higher the trend, the more intensive these variations are.



These time series have an increasing linear trend component, but the fluctuations around this trend have the same intensity;

the fluctuations around this trend are more and more intensive.

Though in practice the multiplicative model is the more popular, both models have their own merits and, depending on the nature of the time series to be analysed, they are equally acceptable.

The additive decomposition is the most appropriate if the magnitude of the seasonal fluctuations, or the variation around the trend-cycle, does not vary with the level of the time series. When the variation in the seasonal pattern, or the variation around the trend-cycle, appears to be proportional to the level of the time series, then a multiplicative decomposition is more appropriate. Multiplicative decompositions are common with economic time series.

An alternative to using a multiplicative decomposition is to first transform the data until the variation in the series appears to be stable over time, then use an additive decomposition. When a log transformation has been used, this is equivalent to using a multiplicative decomposition because $y_t = S_t \times T_t \times R_t$ is equivalent to

$$\log(y_t) = \log(S_t) + \log(T_t) + \log(R_t) + \log(C_t).$$

We will decompose the new orders index for electrical equipment. The data show the number of new orders for electrical equipment (computer, electronic and optical products) in the Euro area (16 countries). The data have been adjusted by working days and normalised so that a value of 100 corresponds to 2005.



Figure 6.1

Figure 6.1 shows the trend-cycle component, T_t , in red and the original data, y_t , in grey. The trend-cycle shows the overall movement in the series, ignoring the seasonality and any small random fluctuations.

Figure 6.2 shows an additive decomposition of these data.

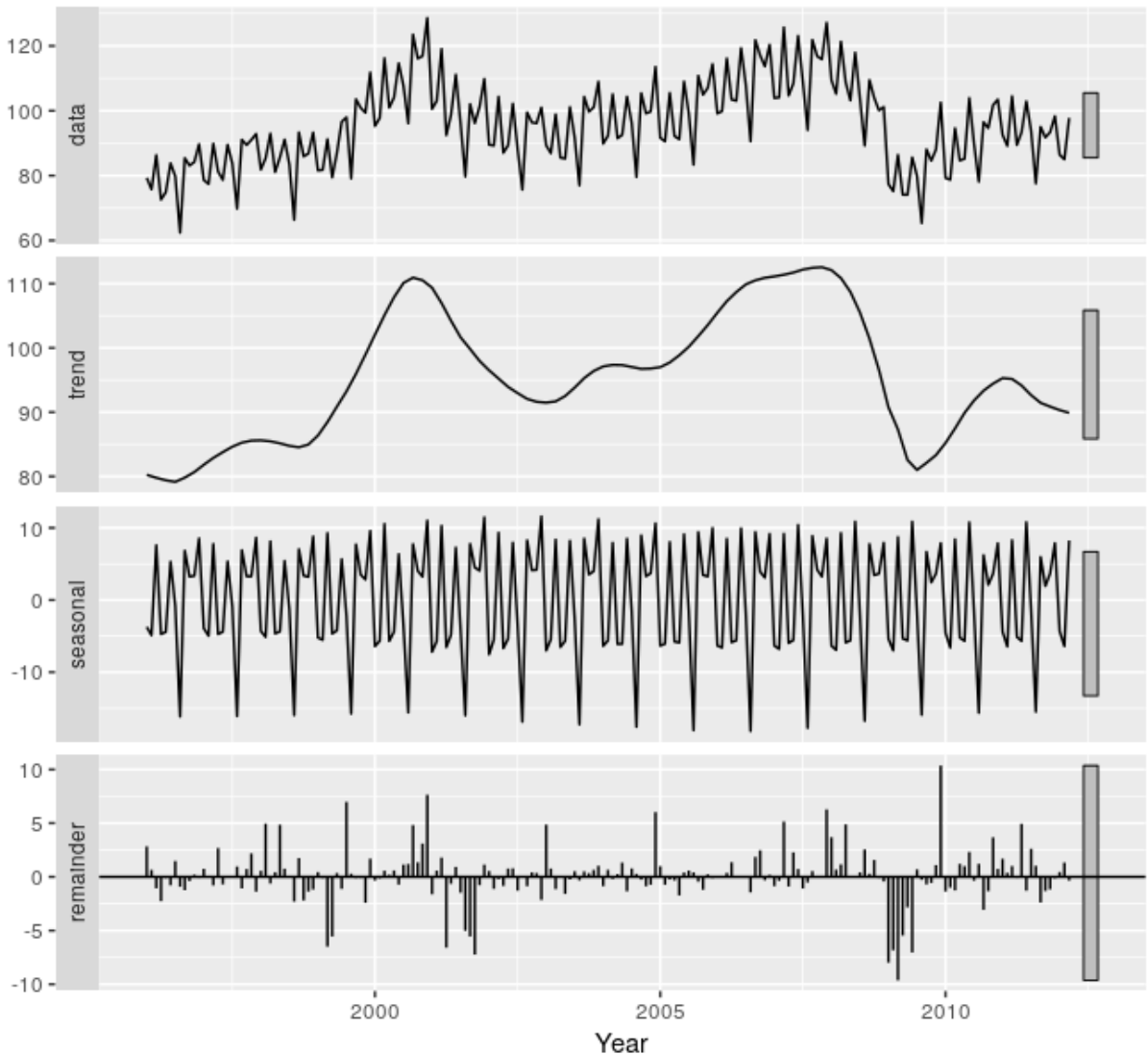


Figure 6.2

The three components are shown separately in the bottom three panels of Figure 6.2. These components can be added together to reconstruct the data shown in the top panel. Notice that the seasonal component changes slowly over time, so that any two consecutive years have similar patterns, but years far apart may have different seasonal patterns. The remainder component shown in the bottom panel is

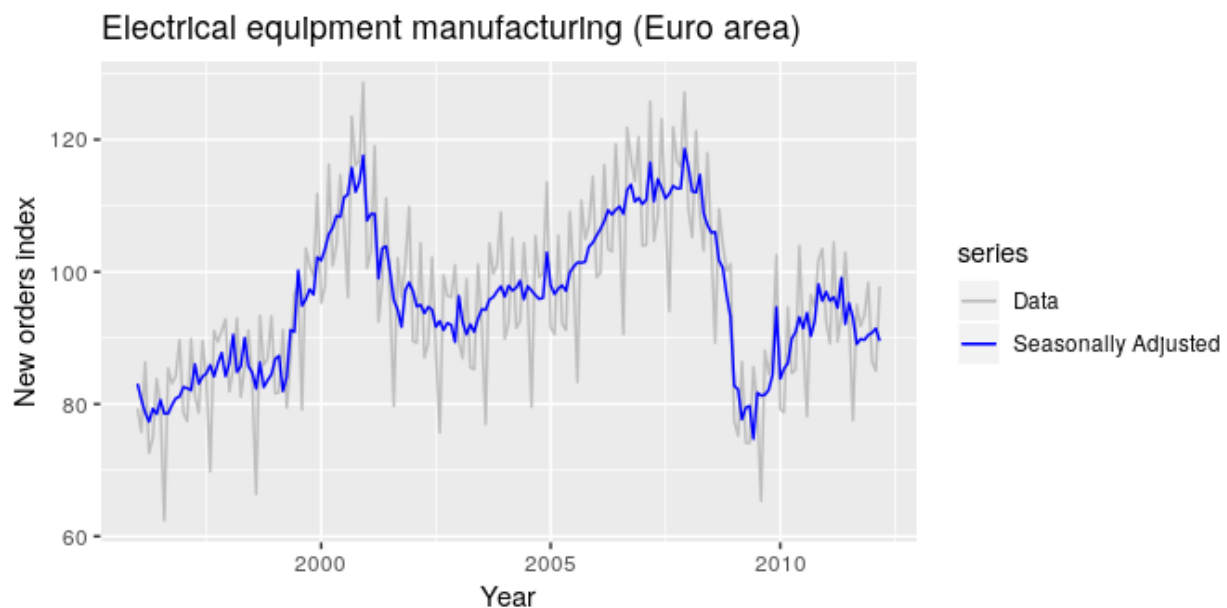
what is left over when the seasonal and trend-cycle components have been subtracted from the data.

The grey bars to the right of each panel show the relative scales of the components. Each grey bar represents the same length but because the plots are on different scales, the bars vary in size. The large grey bar in the bottom panel shows that the variation in the remainder component is small compared to the variation in the data, which has a bar about one quarter the size. If we shrunk the bottom three panels until their bars became the same size as that in the data panel, then all the panels would be on the same scale.

Seasonally adjusted data

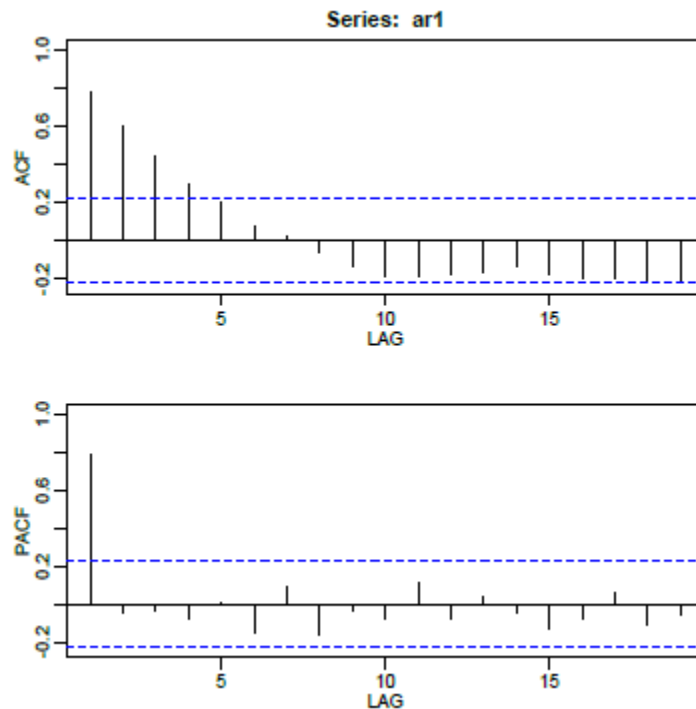
If the seasonal component is removed from the original data, the resulting values are the “seasonally adjusted” data. For an additive decomposition, the seasonally adjusted data are given by $y_t - S_t$, and for multiplicative data, the seasonally adjusted values are obtained using y_t / S_t .

Figure 6.3 shows the seasonally adjusted electrical equipment orders.



If the variation due to seasonality is not of primary interest, the seasonally adjusted series can be useful.

Seasonally adjusted series contain the remainder component as well as the trend-cycle. Therefore, they are not “smooth”, and “downturns” or “upturns” can be misleading. If the purpose is to look for turning points in a series, and interpret any changes in direction, then it is better to use the trend-cycle component rather than the seasonally adjusted data.



IDEAL TREND: Ideal Trend must have decreasing ACF and 1 or 2 “Lags” in PACF

IDEAL SEASONALITY: Cyclically in ACF and few “Lags” with some positive and negative in PACF

IDEAL RANDOM: A spike may or may not be present in PACF even if present it will be small in magnitude.

1.2.1 STATIONARY AND NON-STATIONARY

Loosely speaking a **stationary process** is one whose statistical properties do not change over time.

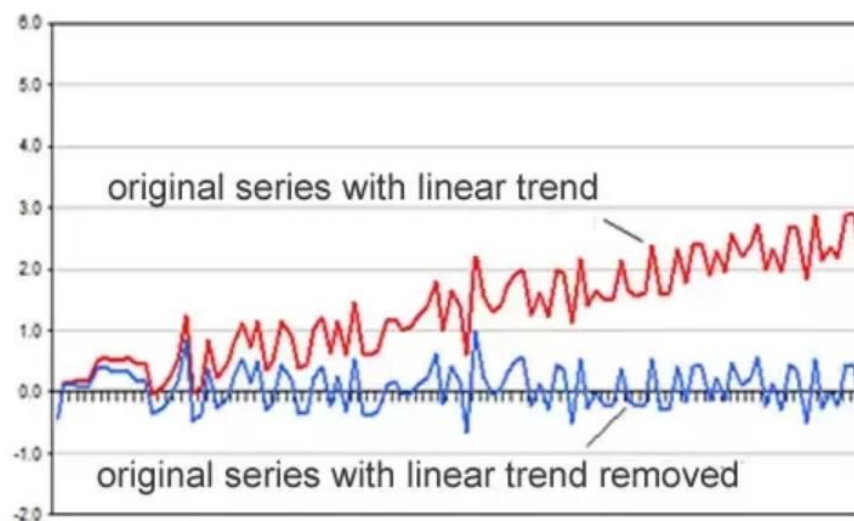
it means that all moments of all degrees (expectations, variances, third order and higher) of the process, anywhere are the same. It also means that the joint

distribution of (X_t, X_s) is the same as (X_{t+r}, X_{s+r}) and hence cannot depend on s or t but only on the distance between s and t , i.e. $s - t$.

Hence, a **non-stationary series** is one whose statistical properties change over time.

Non-stationary data should be first converted into stationary data (for example by trend removal), so that further statistical analysis can be done on the de-trended stationary data. This is so because for example, if the series is consistently increasing over time, the sample mean and variance will grow with the size of the sample, and they will always underestimate the mean and variance in future periods. The usual mean of “de-trending” a series is to fit a regression line and then subtract it from the original data.

Most statistical forecasting methods are based on the assumption that the time series are approximately stationary. A stationary series is relatively easy to predict: you simply forecast that its statistical properties will be the same in the future as they have been in the past.



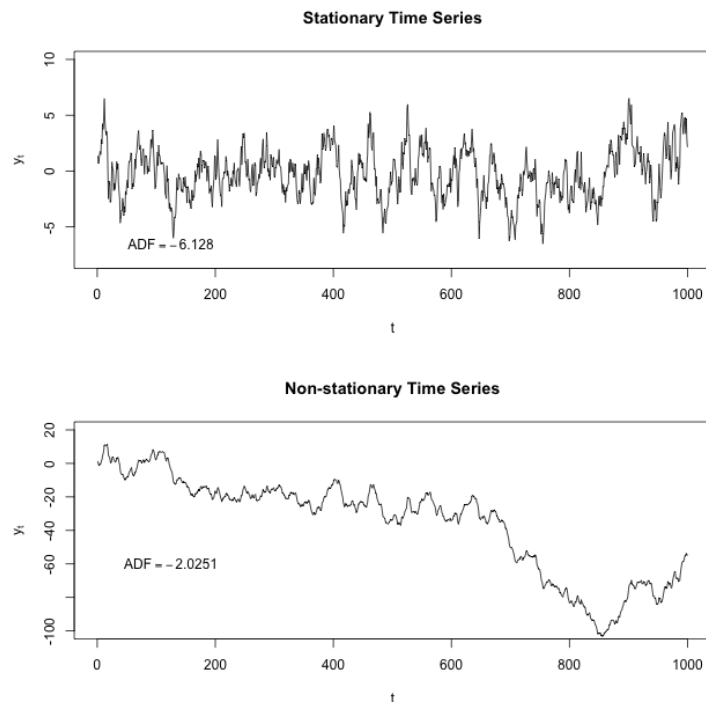
To Convert **NON-STATIONARY** model to **STATIONARY** model we need to perform **DIFFERENCING** until it is stationary.

Differencing in statistics is a transformation applied to time-series data in order to make it stationary. A stationary time series' properties do not depend on the time at which the series is observed.

In order to difference the data, the difference between consecutive observations is computed. Mathematically, this is shown as

$$y'_t = y_t - y_{t-1}$$

Differencing removes the changes in the level of a time series, eliminating trend and seasonality and consequently stabilizing the mean of the time series.



1.2.2 ACF AND PACF

An **autoregressive model** is when a value from a time series is regressed on previous values from that same time series. for example, y_t on y_{t-1}

$$y_t = \beta_0 + \beta_1 y_{t-1} + \epsilon_t.$$

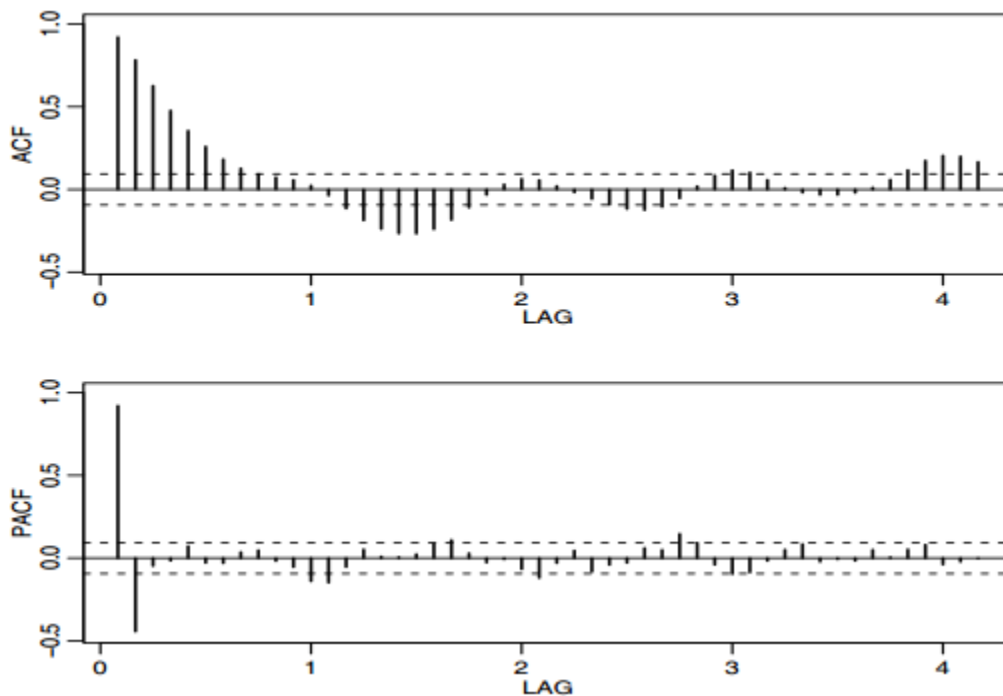
In this regression model, the response variable in the previous time period has become the predictor and the errors have our usual assumptions about errors in a simple linear regression model. The **order** of an autoregression is the number of immediately preceding values in the series that are used to predict the value at the

present time. So, the preceding model is a first-order autoregression, written as AR(1).

If we want to predict y_t this year (y_t) using measurements of global temperature in the previous two years (y_{t-1}, y_{t-2}), then the autoregressive model for doing so would be:

$$y_t = \beta_0 + \beta_1 y_{t-1} + \beta_2 y_{t-2} + \epsilon_t.$$

Then by calculating the correlation of the transformed time series we obtain the **partial autocorrelation function (PACF)**. Where the coefficient of the variable of order k is the value for the k th PACF variable.



1.2.3 MOVING AVERAGES

In time series analysis, the moving-average (MA) model is a common approach for modelling univariate time series. The moving-average model specifies that the output variable depends linearly on the current and various past values of a stochastic (imperfectly predictable) term.

Together with the autoregressive (AR) model, the moving-average model is a special case and key component of the more general ARMA and ARIMA models of time series, which have a more complicated stochastic structure.

SIMPLE MOVING AVERAGE

A simple moving average (SMA) is the unweighted [mean](#) of the previous n data.

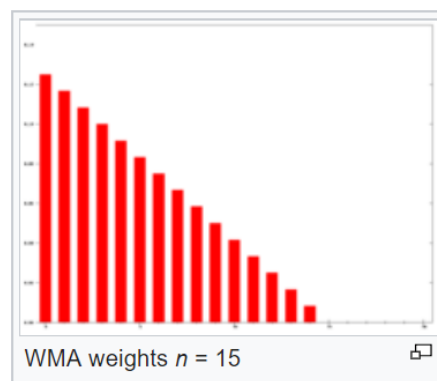
An example of a simple equally weighted running mean for a n -day sample of closing price is the mean of the previous n days' closing prices. If those prices are then the formula next successive price is is

$$\begin{aligned}\bar{p}_{SM} &= \frac{p_M + p_{M-1} + \cdots + p_{M-(n-1)}}{n} \\ &= \frac{1}{n} \sum_{i=0}^{n-1} p_{M-i}\end{aligned}$$

WEIGHTED MOVING AVERAGE

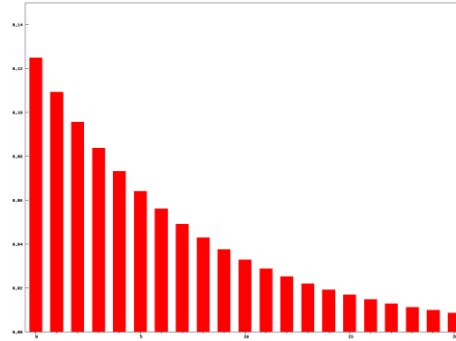
A weighted average is an average that has multiplying factors to give different weights to data at different positions in the sample window. A **weighted moving average** (WMA) has the specific meaning of weights that decrease in arithmetical progression. The more significant number gets the highest weight.

$$WMA_M = \frac{np_M + (n-1)p_{M-1} + \cdots + 2p_{(M-n+2)} + p_{(M-n+1)}}{n + (n-1) + \cdots + 2 + 1}$$



EXPONENTIAL MOVING AVERAGE

An exponential moving average (EMA), also known as an exponentially weighted moving average (EWMA),^[5] is a first-order infinite impulse response filter that applies weighting factors which decrease exponentially. The weighting for each older datum decreases exponentially, never reaching zero.



1.2.4 AR MODEL

The AR involves regressing the variable on its own lagged (i.e., past) values. The autoregressive model specifies that the output variable depends linearly on its own previous values and on a stochastic term (an imperfectly predictable term); thus the model is in the form of a stochastic difference equation.

Together with the moving-average (MA) model, it is a special case and key component of the more general ARMA and ARIMA models of time series,

The notation $AR(p)$ indicates an autoregressive model of order p . The $AR(p)$ model is defined as

$$X_t = c + \sum_{i=1}^p \varphi_i X_{t-i} + \varepsilon_t$$

1.2.5 ARMA MODEL

In the statistical analysis of time series, **autoregressive–moving-average (ARMA) models** provide a parsimonious description of a (weakly) stationary stochastic process in terms of two polynomials, one for the autoregression and the second for the moving average.

Given a time series of data X_t , the ARMA model is a tool for understanding and, perhaps, predicting future values in this series. The model consists of two parts, an autoregressive (AR) part and a moving average (MA) part. The AR part involves regressing the variable on its own lagged (i.e., past) values. The MA part involves modeling the error term as a linear combination of error terms occurring contemporaneously and at various times in the past.

AUTO REGRESSION

The notation $AR(p)$ refers to the autoregressive model of order p . The $AR(p)$ model is written

$$X_t = c + \sum_{i=1}^p \varphi_i X_{t-i} + \varepsilon_t.$$

where $\varphi_1, \dots, \varphi_p$ are **parameters**, c is a constant, and the random variable ε_t is **white noise**.

WEIGHTED MOVING AVERAGE

The notation $MA(q)$ refers to the moving average model of order q :

$$X_t = \mu + \varepsilon_t + \sum_{i=1}^q \theta_i \varepsilon_{t-i}$$

where the $\theta_1, \dots, \theta_q$ are the parameters of the model, μ is the expectation of X_t (often assumed to equal 0), and the $\varepsilon_t, \varepsilon_{t-1}, \dots$ are again, **white noise** error terms.

ARMA

The notation $ARMA(p, q)$ refers to the model with p autoregressive terms and q moving-average terms. This model contains the $AR(p)$ and $MA(q)$ models,

$$X_t = c + \varepsilon_t + \sum_{i=1}^p \varphi_i X_{t-i} + \sum_{i=1}^q \theta_i \varepsilon_{t-i}.$$

1.2.6 ARIMA

ARIMA stands for Autoregressive Integrated Moving Average models. Univariate (single vector) ARIMA is a forecasting technique that projects the future values of a series based entirely on its own inertia. Its main application is in the area of short term forecasting requiring at least 40 historical data points. It works best when your data exhibits a stable or consistent pattern over time with a minimum amount of outliers.

ARIMA Model



ARIMA (2,0,1) $y_t = a_1 y_{t-1} + a_2 y_{t-2} + b_1 \epsilon_{t-1}$

ARIMA (3,0,1) $y_t = a_1 y_{t-1} + a_2 y_{t-2} + a_3 y_{t-3} + b_1 \epsilon_{t-1}$

ARIMA (1,1,0) $\Delta y_t = a_1 \Delta y_{t-1} + \epsilon_t$, where $\Delta y_t = y_t - y_{t-1}$

ARIMA (2,1,0) $\Delta y_t = a_1 \Delta y_{t-1} + a_2 \Delta y_{t-2} + \epsilon_t$ where $\Delta y_t = y_t - y_{t-1}$

To build a time series model issuing ARIMA, we need to study the time series and identify p,d,q

Goodness of fit

- Remember... **Residual analysis** and Mean deviation, Mean Absolute Deviation and Root Mean Square errors?
- Four common techniques are the:

- Mean absolute deviation,
$$MAD = \frac{\sum_{i=1}^n |Y_i - \hat{Y}_i|}{n}$$
- Mean absolute percent error
$$MAPE = \frac{100}{n} \sum_{i=1}^n \frac{|Y_i - \hat{Y}_i|}{Y_i}$$
- Mean square error,
$$MSE = \frac{\sum_{i=1}^n (Y_i - \hat{Y}_i)^2}{n}$$
- Root mean square error.
$$RMSE = \sqrt{MSE}$$

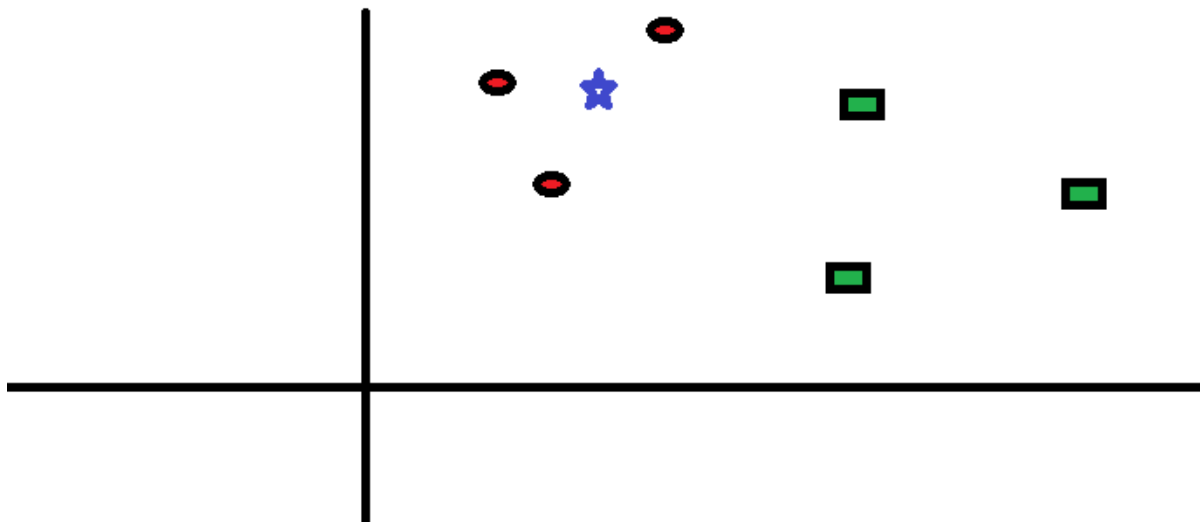
1.3 KNN ALGORITHM

The reason of a bias towards classification models is that most analytical problem involves making a decision.

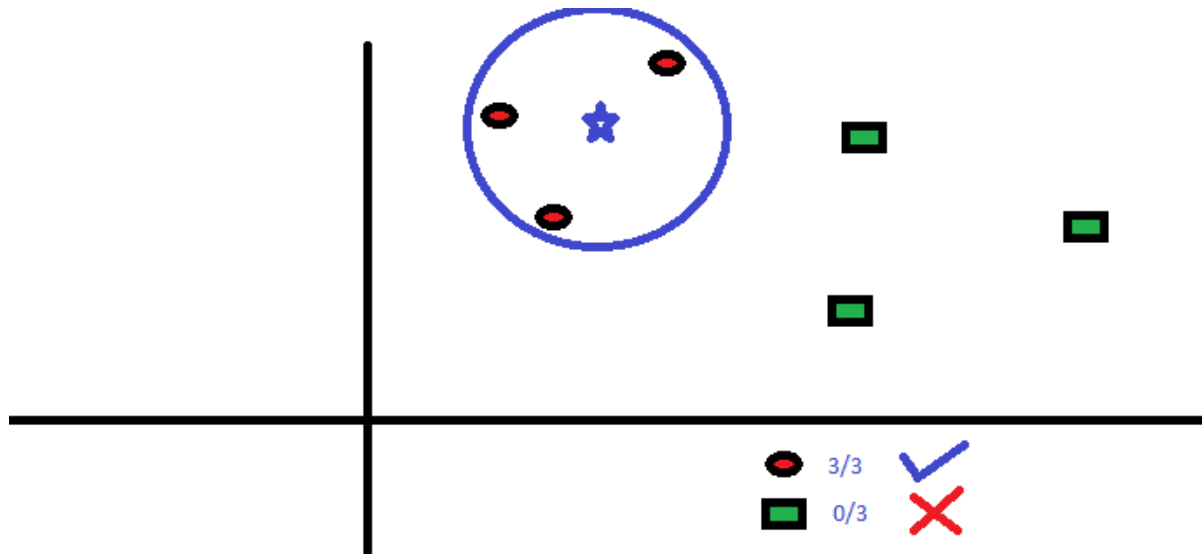
KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. To evaluate any technique we generally look at 3 important aspects:

1. Ease to interpret output
2. Calculation time
3. Predictive Power

Let's take a simple case to understand this algorithm. Following is a spread of red circles (RC) and green squares (GS) :

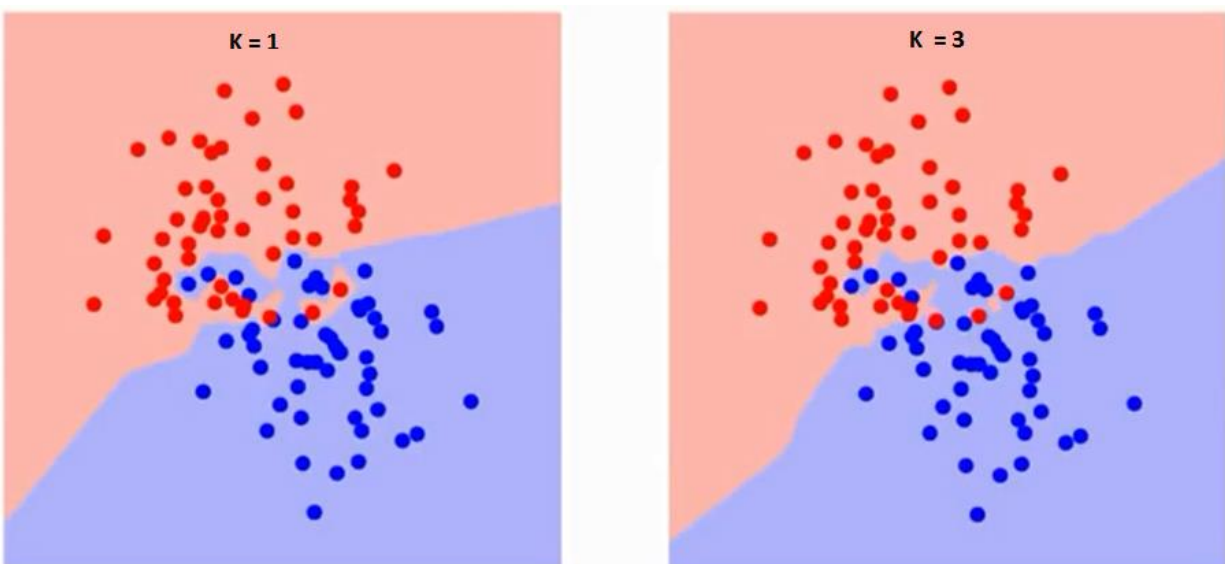


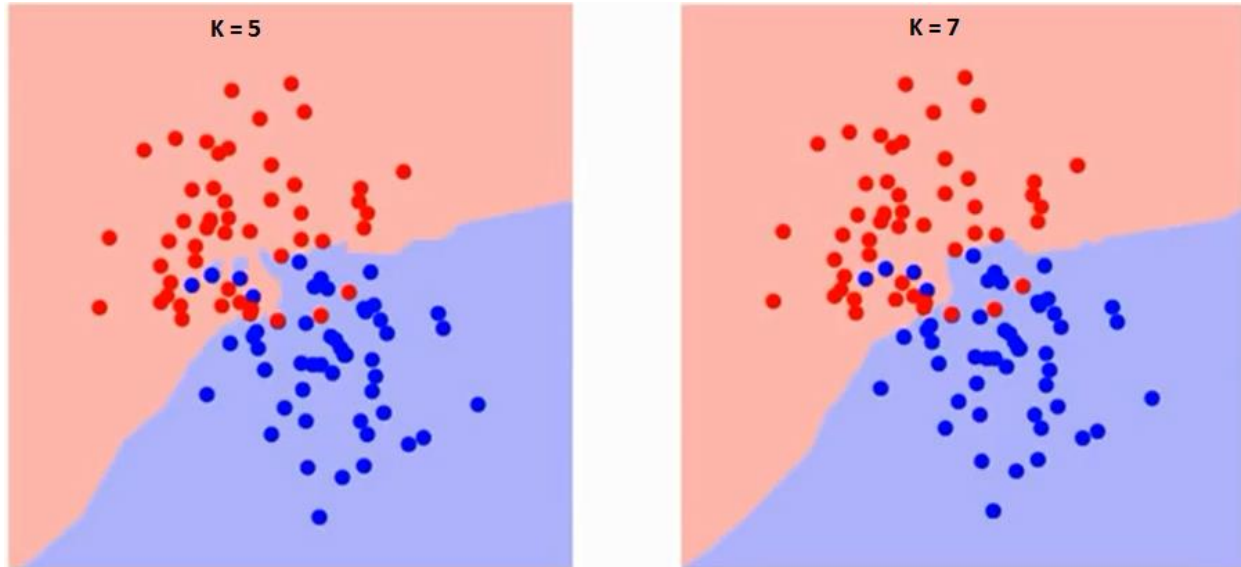
You intend to find out the class of the blue star (BS) . BS can either be RC or GS and nothing else. The “K” in KNN algorithm is the nearest neighbors we wish to take vote from. Let's say $K = 3$. Hence, we will now make a circle with BS as center just as big as to enclose only three datapoints on the plane. Refer to following diagram for more details:



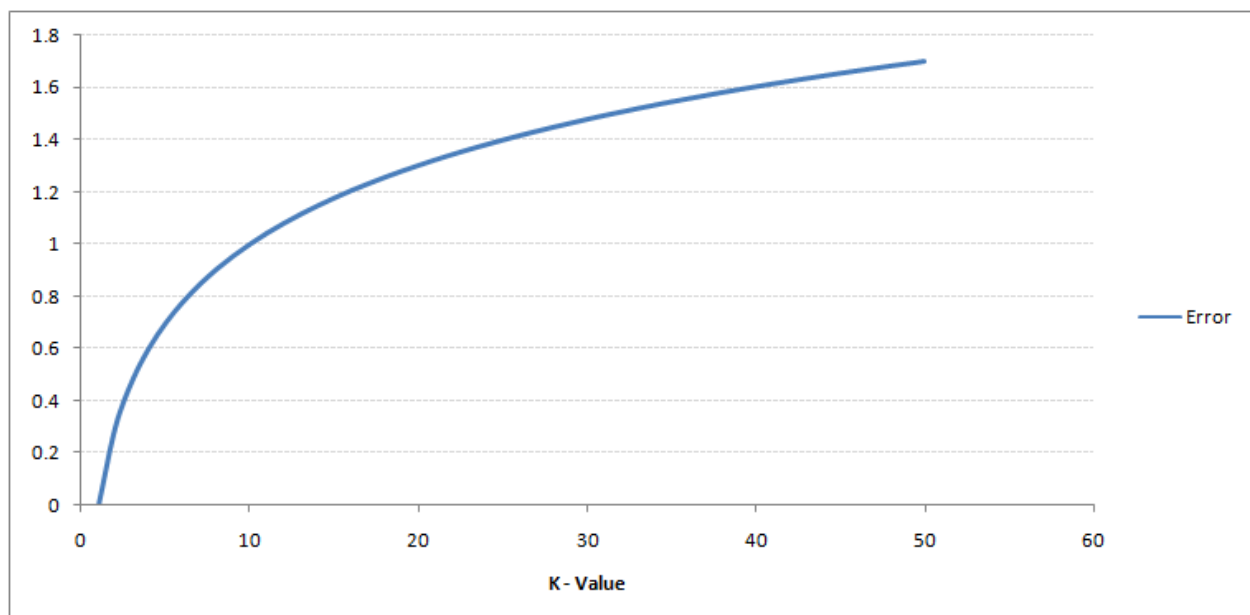
The three closest points (**USING DISTANCE ESTIMATION**) to BS are all RC. Hence, with good confidence level we can say that the BS should belong to the class RC. Here, the choice became very obvious as all three votes from the closest neighbor went to RC. The choice of the parameter K is very crucial in this algorithm.

First let us try to understand what exactly does K influence in the algorithm. If we see the last example, given that all the 6 training observation remain constant, with a given K value we can make boundaries of each class. These boundaries will segregate RC from GS. The same way, let's try to see the effect of value "K" on the class boundaries. Following are the different boundaries separating the two classes with different values of K.



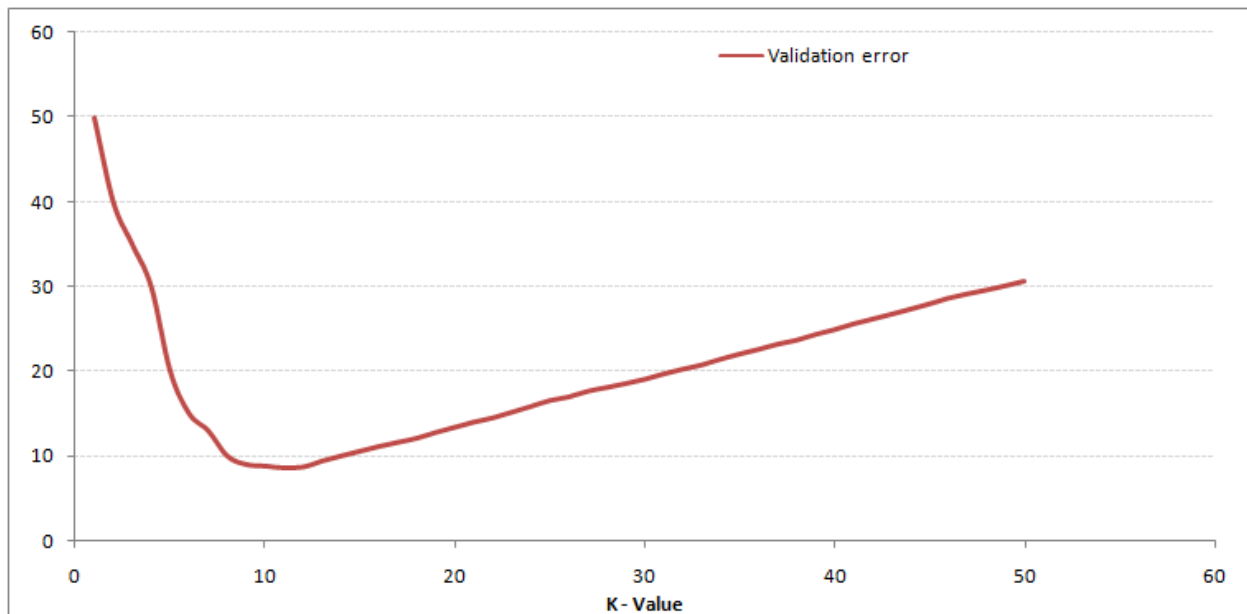


If you watch carefully, you can see that the boundary becomes smoother with increasing value of K . With K increasing to infinity it finally becomes all blue or all red depending on the total majority. The training error rate and the validation error rate are two parameters we need to access on different K -value. Following is the curve for the training error rate with varying value of K :



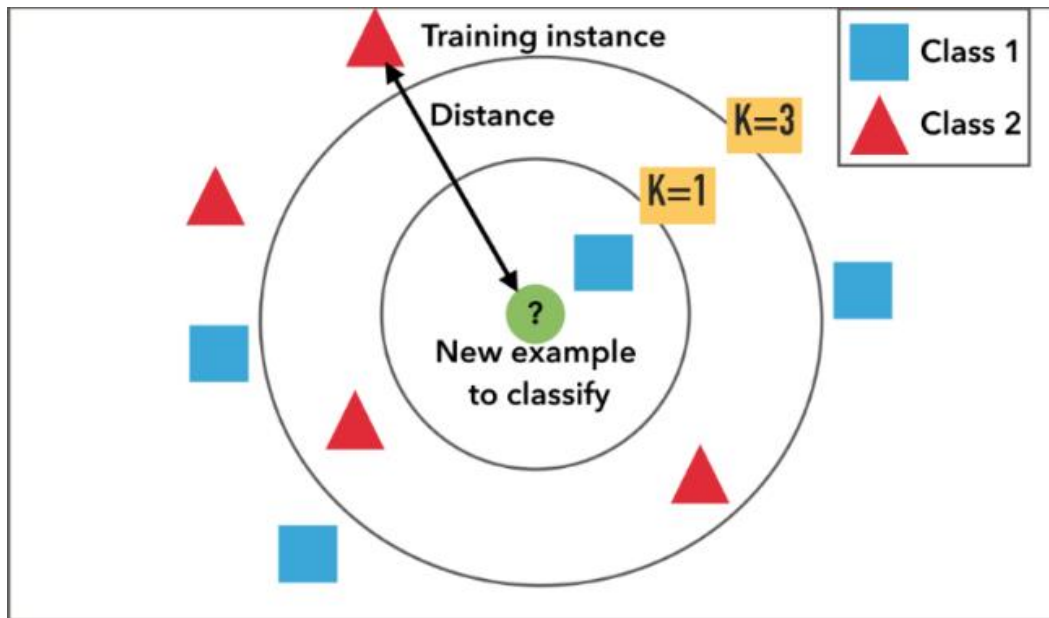
As you can see, the error rate at $K=1$ is always zero for the training sample. This is because the closest point to any training data point is itself. Hence the prediction is always accurate with $K=1$. If validation error curve would have been similar, our

choice of K would have been 1. Following is the validation error curve with varying value of K :



This makes the story more clear. At $K=1$, we were overfitting the boundaries. Hence, error rate initially decreases and reaches a minima. After the minima point, it then increase with increasing K . To get the optimal value of K , you can segregate the training and validation from the initial dataset. Now plot the validation error curve to get the optimal value of K . This value of K should be used for all predictions.

Example 2 of k -NN classification. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If $k = 3$ (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If $k = 5$ (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).



KNN can be used for **classification**—the output is a class membership (predicts a class—a discrete value). An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors. It can also be used for **regression**—output is the value for the object (predicts continuous values). This value is the average (or median) of the values of its k nearest neighbors.

Some pros and cons of KNN

Pros:

- No assumptions about data—useful, for example, for nonlinear data
- Simple algorithm—to explain and understand/interpret
- High accuracy (relatively)—it is pretty high but not competitive in comparison to better supervised learning models
- Versatile—useful for classification or regression

Cons:

- Computationally expensive—because the algorithm stores all of the training data

- High memory requirement
- Stores all (or almost all) of the training data
- Prediction stage might be slow (with big N)
- Sensitive to irrelevant features and the scale of the data

1.4 K-MEANS CLUSTERING

K -means clustering is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K . The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided.

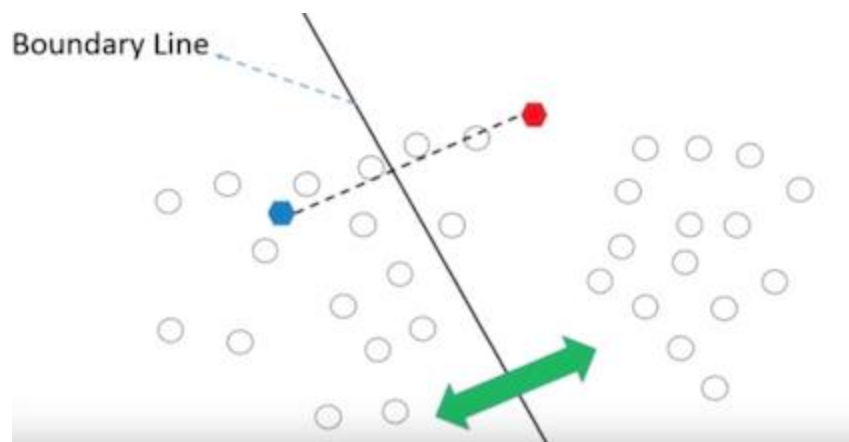
As, you can see, k-means algorithm is composed of 3 steps:

Step 1: Initialization

The first thing k-means does, is **randomly** choose K examples (data points) from the dataset (the 4 green points) as initial centroids and that's simply because it does not know yet where the center of each cluster is. (a centroid is the center of a cluster).

Step 2: Cluster Assignment

Then, all the data points that are the closest (similar) to a centroid will create a cluster. If we're using the Euclidean distance between data points and every centroid, a straight line is drawn between two centroids, then a perpendicular bisector (boundary line) divides this line into two clusters.

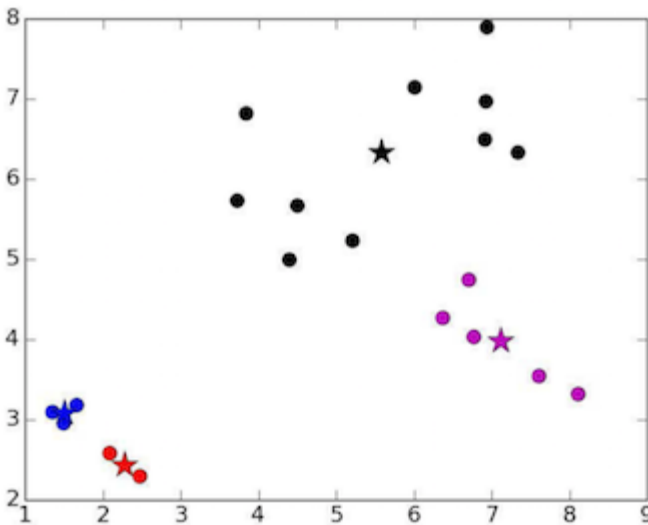


Step 3: Move the centroid

Now, we have new clusters, that need centers. A centroid's new value is going to be the mean of all the examples in a cluster.

We'll keep repeating step 2 and 3 until the centroids stop moving, in other words, K-means algorithm is converged.

Unlucky centroids:



The blue and red stars are unlucky centroids :(. From MIT 6.0002 lecture 12

Choosing poorly the random initial centroids will take longer to converge or get stuck on local optima which may result in bad clustering. in the picture above, the blue and red stars are unlucky centroids.

There are two solutions:

- Distribute them over the space.
- Try different sets of random centroids, and choose the best set.

1.5 DECISION TREE ALGORITHM

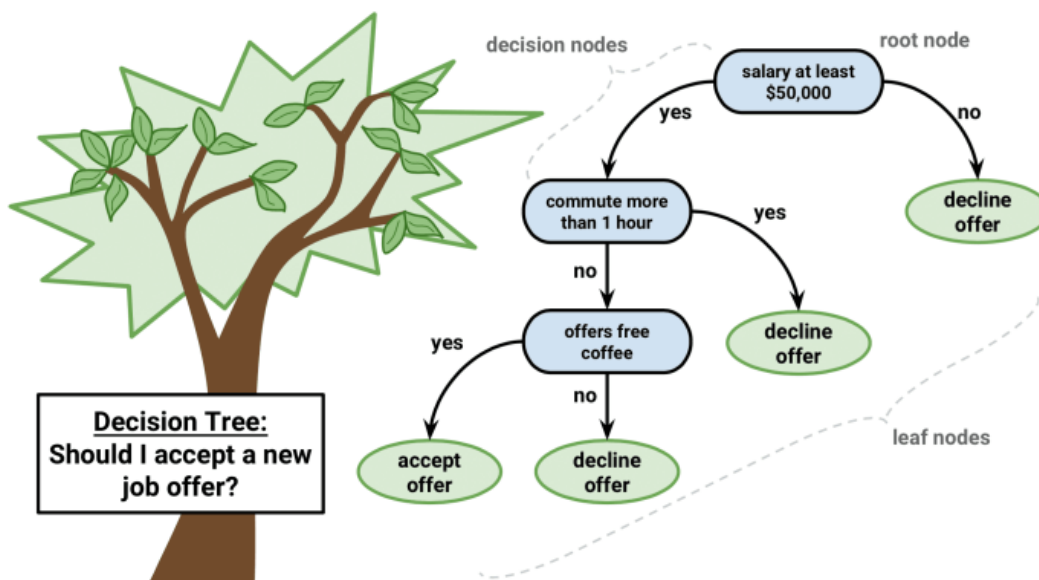
Decision Tree algorithm belongs to the family of [supervised learning algorithms](#). Unlike other supervised learning algorithms, decision tree algorithm can be used for solving [regression and classification problems](#) too.

The general motive of using Decision Tree is to create a training model which can use to predict class or value of target variables by **learning decision rules** inferred from prior data(training data).

The understanding level of Decision Trees algorithm is so easy compared with other classification algorithms. The decision tree algorithm tries to solve the problem, by using tree representation. Each **internal node** of the tree corresponds to an attribute, and each **leaf node** corresponds to a class label.

Decision Tree Algorithm Pseudocode

1. Place the best attribute of the dataset at the **root** of the tree.
2. Split the training set into **subsets**. Subsets should be made in such a way that each subset contains data with the same value for an attribute.
3. Repeat step 1 and step 2 on each subset until you find **leaf nodes** in all the branches of the tree.



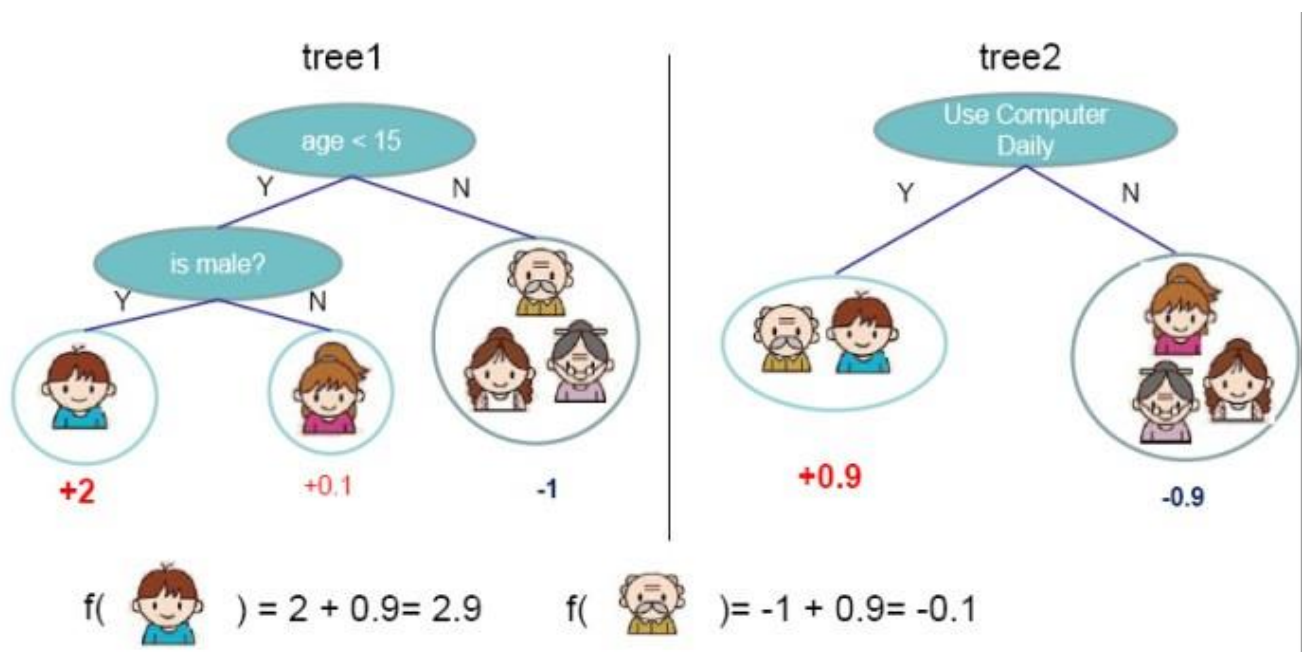
In decision trees, for predicting a class label for a record we start from the **root** of the tree. We compare the values of the root attribute with record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

We continue comparing our record's attribute values with other **internal nodes** of the tree until we reach a **leaf node** with predicted class value. As we know how the modeled decision tree can be used to predict the target class or the value. Now let's understanding how we can create the decision tree model.

Assumptions while creating Decision Tree

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

- At 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.



Decision Trees follow **Sum of Product (SOP)** representation. For the above images, you can see how we can predict **can we accept the new job offer?** and **Use computer daily?** from traversing for the root node to the leaf node.

It's a sum of product representation. The Sum of product(SOP) is also known as **Disjunctive Normal Form**. For a class, every branch from the root of the tree to a leaf node having the same class is a conjunction(product) of values, different branches ending in that class form a disjunction(sum).

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 know the attributes selection. We have different attributes selection measure to identify the attribute which can be considered as the root note at each level.

The popular attribute selection measures:

- Information gain
- Gini index

Attributes Selection

If 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 *criterion* like **information gain, gini index**, etc. 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 gini index, attributes are assumed to be continuous.

1.5.1 INFORMATION GAIN

By using information gain as a criterion, we try to estimate the information contained by each attribute. We are going to use some points deducted from [information theory](#).

To measure the randomness or uncertainty of a random variable X is defined by **Entropy**.

For a binary classification problem with only two classes, positive and negative class.

- If all examples are positive or all are negative then entropy will be zero i.e, low.
- If half of the records are of positive class and half are of negative class then entropy is one i.e, high.

By calculating **entropy measure** of each attribute we can calculate their **information gain**. Information Gain calculates the expected reduction in entropy due to sorting on the attribute. Information gain can be calculated.

To get a clear understanding of calculating **information gain & entropy**, we will try to implement it on a sample data.

Example: Construct a Decision Tree by using “information gain” as a criterion

	A	B	C	D	E
1	4.8	3.4	1.9	0.2	positive
2	5	3	1.6	0.2	positive
3	5	3.4	1.6	0.4	positive
4	5.2	3.5	1.5	0.2	positive
5	5.2	3.4	1.4	0.2	positive
6	4.7	3.2	1.6	0.2	positive
7	4.8	3.1	1.6	0.2	positive
8	5.4	3.4	1.5	0.4	positive
9	7	3.2	4.7	1.4	negative
10	6.4	3.2	4.5	1.5	negative
11	6.9	3.1	4.9	1.5	negative
12	5.5	2.3	4	1.3	negative
13	6.5	2.8	4.6	1.5	negative
14	5.7	2.8	4.5	1.3	negative
15	6.3	3.3	4.7	1.6	negative
16	4.9	2.4	3.3	1	negative

We are going to use this data sample. Let's try to use information gain as a criterion. Here, we have 5 columns out of which 4 columns have continuous data and 5th column consists of class labels.

A, B, C, D attributes can be considered as predictors and E column class labels can be considered as a target variable. For constructing a decision tree from this data, we have to convert continuous data into categorical data.

We have chosen some random values to categorize each attribute:

A	B	C	D
≥ 5	≥ 3.0	≥ 4.2	≥ 1.4
< 5	< 3.0	< 4.2	< 1.4

There are **2 steps for calculating information gain** for each attribute:

1. Calculate entropy of Target.
2. Entropy for every attribute A, B, C, D needs to be calculated. Using information gain formula we will subtract this entropy from the entropy of target. The result is Information Gain.

The entropy of Target: We have 8 records with negative class and 8 records with positive class. So, we can directly estimate the entropy of target as 1.

Variable E	
Positive	Negative
8	8

Calculating entropy using formula:

$$\begin{aligned}
 E(8,8) &= -1 * ((p(+ve)*\log_2(p(+ve))) + (p(-ve)*\log_2(p(-ve)))) \\
 &= -1 * ((8/16)*\log_2(8/16)) + (8/16) * \log_2(8/16)) \\
 &= 1
 \end{aligned}$$

Information gain for Var A

Var A has value ≥ 5 for 12 records out of 16 and 4 records with value < 5 value.

- For Var A ≥ 5 & class == positive: 5/12
- For Var A ≥ 5 & class == negative: 7/12
 - $\text{Entropy}(5,7) = -1 * ((5/12)*\log_2(5/12) + (7/12)*\log_2(7/12)) = 0.9799$
- For Var A < 5 & class == positive: 3/4
- For Var A < 5 & class == negative: 1/4
 - $\text{Entropy}(3,1) = -1 * ((3/4)*\log_2(3/4) + (1/4)*\log_2(1/4)) = 0.81128$

$$\begin{aligned}\text{Entropy}(\text{Target}, A) &= P(\geq 5) * E(5,7) + P(< 5) * E(3,1) \\ &= (12/16) * 0.9799 + (4/16) * 0.81128 = 0.937745\end{aligned}$$

$$\text{Information Gain(IG)} = E(\text{Target}) - E(\text{Target}, A) = 1 - 0.937745 = 0.062255$$

Information gain for Var B

Var B has value ≥ 3 for 12 records out of 16 and 4 records with value < 3 value.

- For Var B ≥ 3 & class == positive: 8/12
- For Var B ≥ 3 & class == negative: 4/12
 - $\text{Entropy}(8,4) = -1 * ((8/12)*\log_2(8/12) + (4/12)*\log_2(4/12)) = 0.39054$
- For Var B < 3 & class == positive: 0/4
- For Var B < 3 & class == negative: 4/4
 - $\text{Entropy}(0,4) = -1 * ((0/4)*\log_2(0/4) + (4/4)*\log_2(4/4)) = 0$

$$\begin{aligned}\text{Entropy}(\text{Target}, B) &= P(\geq 3) * E(8,4) + P(< 3) * E(0,4) \\ &= (12/16) * 0.39054 + (4/16) * 0 = 0.292905\end{aligned}$$

$$\text{Information Gain(IG)} = E(\text{Target}) - E(\text{Target}, B) = 1 - 0.292905 = 0.707095$$

Information gain for Var C

Var C has value ≥ 4.2 for 6 records out of 16 and 10 records with value < 4.2 value.

- For Var C ≥ 4.2 & class == positive: 0/6
- For Var C ≥ 4.2 & class == negative: 6/6
 - Entropy(0,6) = 0
- For Var C < 4.2 & class == positive: 8/10
- For Var C < 4.2 & class == negative: 2/10
 - Entropy(8,2) = 0.72193

$$\begin{aligned}\text{Entropy}(\text{Target}, C) &= P(\geq 4.2) * E(0,6) + P(< 4.2) * E(8,2) \\ &= (6/16) * 0 + (10/16) * 0.72193 = 0.4512\end{aligned}$$

$$\text{Information Gain(IG)} = E(\text{Target}) - E(\text{Target}, C) = 1 - 0.4512 = 0.5488$$

Information gain for Var D

Var D has value ≥ 1.4 for 5 records out of 16 and 11 records with value < 1.4 value.

- For Var D ≥ 1.4 & class == positive: 0/5
- For Var D ≥ 1.4 & class == negative: 5/5
 - Entropy(0,5) = 0
- For Var D < 1.4 & class == positive: 8/11
- For Var D < 1.4 & class == negative: 3/11
 - Entropy(8,3) = $-1 * ((8/11) * \log_2(8/11) + (3/11) * \log_2(3/11))$
= 0.84532

$$\begin{aligned}\text{Entropy}(\text{Target}, D) &= P(\geq 1.4) * E(0,5) + P(< 1.4) * E(8,3) \\ &= 5/16 * 0 + (11/16) * 0.84532 = 0.5811575\end{aligned}$$

$$\text{Information Gain(IG)} = E(\text{Target}) - E(\text{Target}, D) = 1 - 0.5811575 = 0.4188425$$

		Target	
		Positive	Negative
A	≥ 5.0	5	7
	< 5	3	1
Information Gain of A = 0.062255			

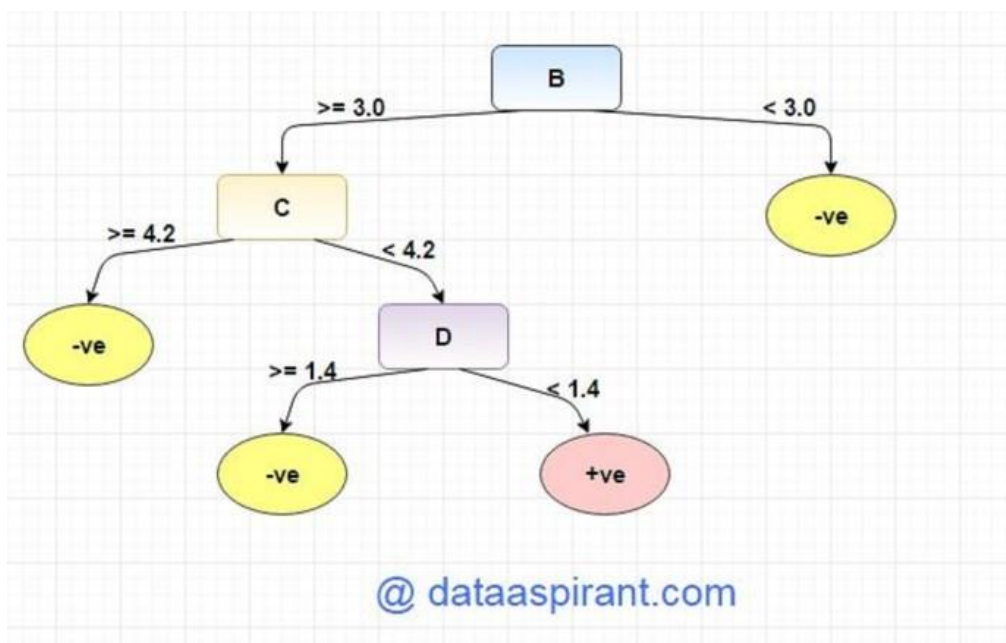
		Target	
		Positive	Negative
B	≥ 3.0	8	4
	< 3.0	0	4
Information Gain of B = 0.7070795			

		Target	
		Positive	Negative
C	≥ 4.2	0	6
	< 4.2	8	2
Information Gain of C = 0.5488			

		Target	
		Positive	Negative
D	≥ 1.4	0	5
	< 1.4	8	3
Information Gain of D = 0.41189			

From the above Information Gain calculations, we can build a decision tree. We should place the attributes on the tree according to their values.

An Attribute with better value than other should position as root and A branch with entropy 0 should be converted to a leaf node. A branch with entropy more than 0 needs further splitting.



1.5.2 GINI INDEX

Gini Index is a metric to measure how often a randomly chosen element would be incorrectly identified. It means an attribute with lower gini index should be preferred.

Example: Construct a Decision Tree by using “gini index” as a criterion

	A	B	C	D	E
1	4.8	3.4	1.9	0.2	positive
2	5	3	1.6	0.2	positive
3	5	3.4	1.6	0.4	positive
4	5.2	3.5	1.5	0.2	positive
5	5.2	3.4	1.4	0.2	positive
6	4.7	3.2	1.6	0.2	positive
7	4.8	3.1	1.6	0.2	positive
8	5.4	3.4	1.5	0.4	positive
9	7	3.2	4.7	1.4	negative
10	6.4	3.2	4.5	1.5	negative
11	6.9	3.1	4.9	1.5	negative
12	5.5	2.3	4	1.3	negative
13	6.5	2.8	4.6	1.5	negative
14	5.7	2.8	4.5	1.3	negative
15	6.3	3.3	4.7	1.6	negative
16	4.9	2.4	3.3	1	negative

We are going to use same data sample that we used for information gain example. Let's try to use gini index as a criterion. Here, we have 5 columns out of which 4 columns have continuous data and 5th column consists of class labels.

A, B, C, D attributes can be considered as predictors and E column class labels can be considered as a target variable. For constructing a decision tree from this data, we have to convert continuous data into categorical data.

We have chosen some random values to categorize each attribute:

A	B	C	D
≥ 5	≥ 3.0	≥ 4.2	≥ 1.4
< 5	< 3.0	< 4.2	< 1.4

Gini Index for Var A

Var A has value ≥ 5 for 12 records out of 16 and 4 records with value < 5 value.

- For Var A ≥ 5 & class == positive: 5/12
- For Var A ≥ 5 & class == negative: 7/12
 - $\text{gini}(5,7) = 1 - ((5/12)^2 + (7/12)^2) = 0.4860$
- For Var A < 5 & class == positive: 3/4
- For Var A < 5 & class == negative: 1/4
 - $\text{gini}(3,1) = 1 - ((3/4)^2 + (1/4)^2) = 0.375$

By adding weight and sum each of the gini indices:

$$\text{gini}(\text{Target}, A) = (12/16) * (0.486) + (4/16) * (0.375) = 0.45825$$

Gini Index for Var B

Var B has value ≥ 3 for 12 records out of 16 and 4 records with value < 3 value.

- For Var B ≥ 3 & class == positive: 8/12
- For Var B ≥ 3 & class == negative: 4/12
 - $\text{gini}(8,4) = 1 - ((8/12)^2 + (4/12)^2) = 0.446$
- For Var B < 3 & class == positive: 0/4
- For Var B < 3 & class == negative: 4/4
 - $\text{gini}(0,4) = 1 - ((0/4)^2 + (4/4)^2) = 0$

$$\text{gini}(\text{Target}, B) = (12/16) * 0.446 + (4/16) * 0 = 0.3345$$

Gini Index for Var C

Var C has value ≥ 4.2 for 6 records out of 16 and 10 records with value < 4.2 value.

- For Var C ≥ 4.2 & class == positive: 0/6
- For Var C ≥ 4.2 & class == negative: 6/6

- $\text{gini}(0,6) = 1 - ((0/8)^2 + (6/6)^2) = 0$
- For Var C < 4.2 & class == positive: 8/10
- For Var C < 4.2 & class == negative: 2/10
- $\text{gini}(8,2) = 1 - ((8/10)^2 + (2/10)^2) = 0.32$

$$\text{gini}(\text{Target}, C) = (6/16) * 0 + (10/16) * 0.32 = 0.2$$

Gini Index for Var D

Var D has value ≥ 1.4 for 5 records out of 16 and 11 records with value < 1.4 value.

- For Var D ≥ 1.4 & class == positive: 0/5
- For Var D ≥ 1.4 & class == negative: 5/5
- $\text{gini}(0,5) = 1 - ((0/5)^2 + (5/5)^2) = 0$
- For Var D < 1.4 & class == positive: 8/11
- For Var D < 1.4 & class == negative: 3/11
- $\text{gini}(8,3) = 1 - ((8/11)^2 + (3/11)^2) = 0.397$

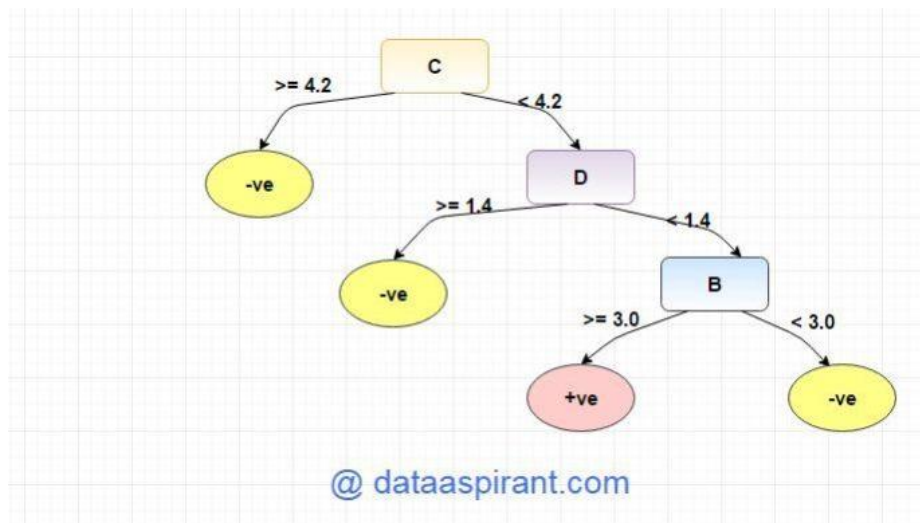
$$\text{gini}(\text{Target}, D) = (5/16) * 0 + (11/16) * 0.397 = 0.273$$

		wTarget	
		Positive	Negative
A	≥ 5.0	5	7
	< 5	3	1
Gini Index of A = 0.45825			

		Target	
		Positive	Negative
B	≥ 3.0	8	4
	< 3.0	0	4
Gini Index of B = 0.3345			

		Target	
		Positive	Negative
C	≥ 4.2	0	6
	< 4.2	8	2
Gini Index of C = 0.2			

		Target	
		Positive	Negative
D	≥ 1.4	0	5
	< 1.4	8	3
Gini Index of D = 0.273			



Overfitting

Overfitting is a practical problem while building a decision tree model. The model is having an issue of overfitting is considered when the algorithm continues to go deeper and deeper in the tree to reduce the training set error but results with an increased test set error i.e., Accuracy of prediction for our model goes down. It generally happens when it builds many branches due to outliers and irregularities in data.

Two approaches which we can use to avoid overfitting are:

- Pre-Pruning
- Post-Pruning

Pre-Pruning

In pre-pruning, it stops the tree construction bit early. It is preferred not to split a node if its goodness measure is below a threshold value. But it's difficult to choose an appropriate stopping point.

Post-Pruning

In post-pruning first, it goes deeper and deeper in the tree to build a complete tree. If the tree shows the overfitting problem then pruning is done as a post-pruning step. We use a cross-validation data to check the effect of our pruning.

Using cross-validation data, it tests whether expanding a node will make an improvement or not.

If it shows an improvement, then we can continue by expanding that node. But if it shows a reduction in accuracy then it should not be expanded i.e, the node should be converted to a leaf node.

Decision Tree Algorithm Advantages and Disadvantages

Advantages:

1. Decision Trees are easy to explain. It results in a set of rules.
2. It follows the same approach as humans generally follow while making decisions.
3. Interpretation of a complex Decision Tree model can be simplified by its visualizations. Even a naive person can understand logic.
4. The Number of hyper-parameters to be tuned is almost null.

Disadvantages:

1. There is a high probability of overfitting in Decision Tree.
2. Generally, it gives low prediction accuracy for a dataset as compared to other machine learning algorithms.
3. Information gain in a decision tree with categorical variables gives a biased response for attributes with greater no. of categories.
4. Calculations can become complex when there are many class labels.

1.6 ENSEMBLE LEARNING

Ensemble is the art of combining diverse set of learners (individual models) together to improvise on the stability and predictive power of the model.

Ensemble learning is a broad topic and is only confined by your own imagination.

Here are the top 4 reasons for a model to be different. They can be different because of a mix of these factors as well:

1. Difference in population



2. Difference in hypothesis



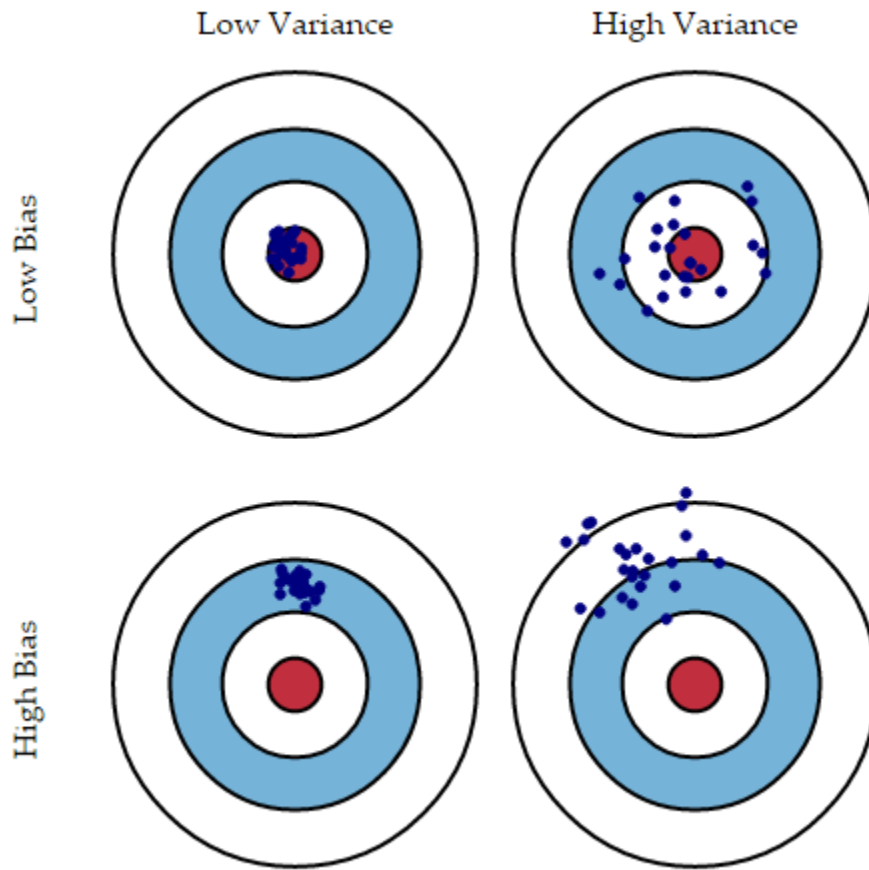
3. Difference in modeling technique



Error in Ensemble Learning (Variance vs. Bias)

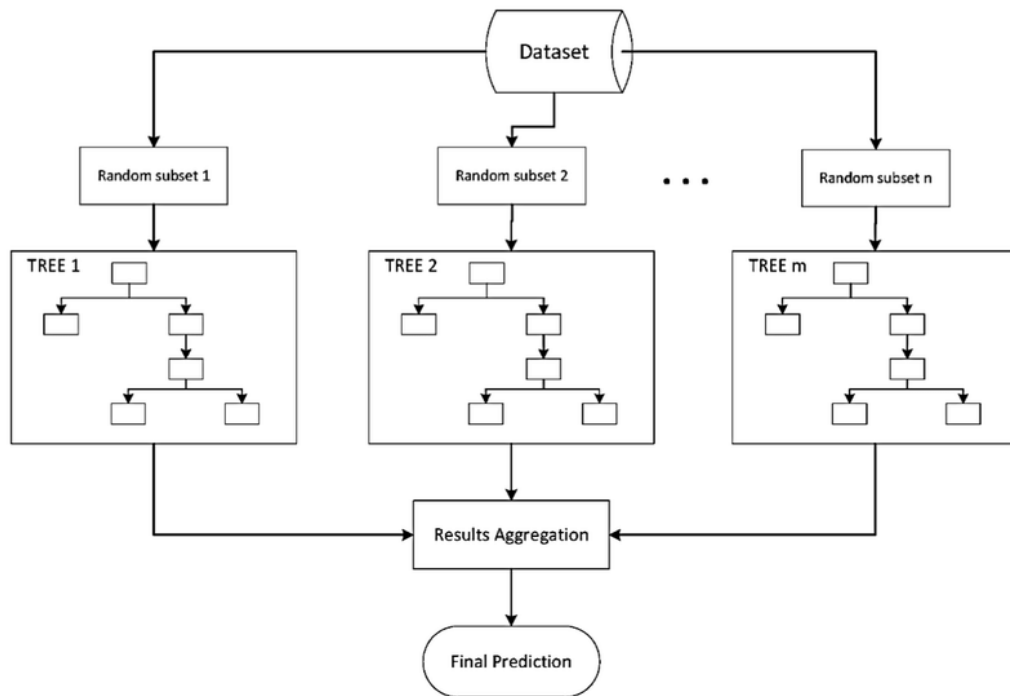
Bias error is useful to quantify how much on an average are the predicted values different from the actual value. A high bias error means we have a under-performing model which keeps on missing important trends.

Variance on the other side quantifies how are the prediction made on same observation different from each other. A high variance model will over-fit on your training population and perform badly on any observation beyond training. Following diagram will give you more clarity (Assume that red spot is the real value and blue dots are predictions) :



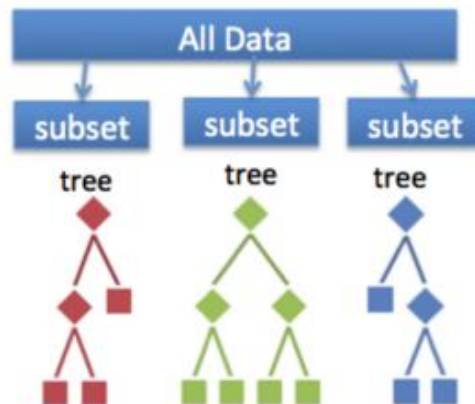
Some Commonly used Ensemble learning techniques

1. **BAGGing**, or *Bootstrap AGG*regating. BAGGing gets its name because it combines *Bootstrapping* and *Aggregation* to form one ensemble model. Given a sample of data, multiple bootstrapped subsamples are pulled. A Decision Tree is formed on each of the bootstrapped subsamples. After each subsample Decision Tree has been formed, an algorithm is used to aggregate over the Decision Trees to form the most efficient predictor. The image below will help explain:



Given a Dataset, bootstrapped subsamples are pulled. A Decision Tree is formed on each bootstrapped sample. The results of each tree are aggregated to yield the strongest, most accurate predictor.

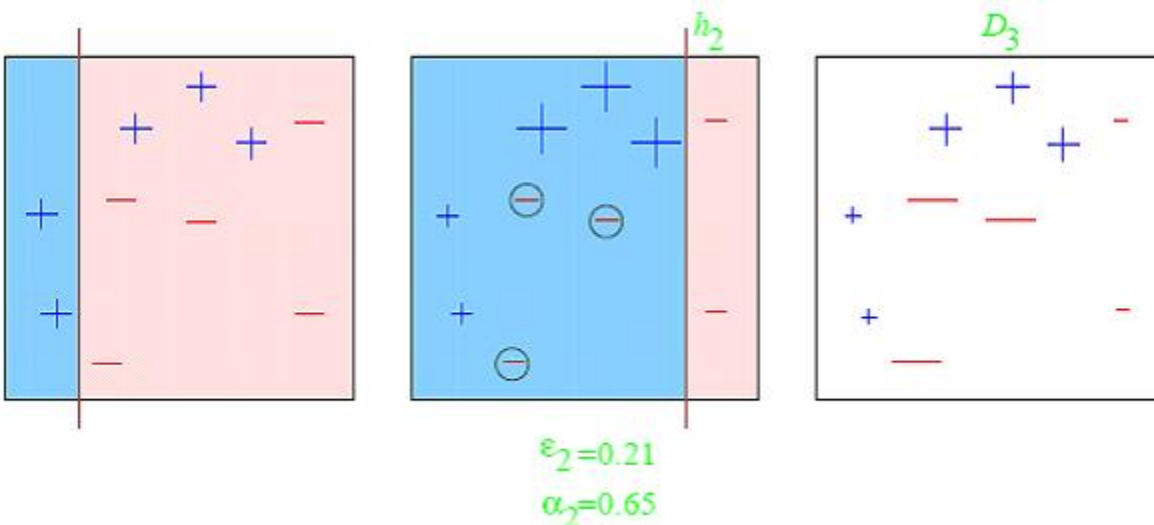
2. Random Forest Models. Random Forest Models can be thought of as BAGGing, with a slight tweak. When deciding where to split and how to make decisions, BAGGed Decision Trees have the full disposal of features to choose from. Therefore, although the bootstrapped samples may be slightly different, the data is largely going to break off at the same features throughout each model. In contrary, Random Forest models decide where to split based on a random selection of features. Rather than splitting at similar features at each node throughout, Random Forest models implement a level of differentiation because each tree will split based on different features. This level of differentiation provides a greater ensemble to aggregate over, ergo producing a more accurate predictor. Refer to the image for a better understanding.



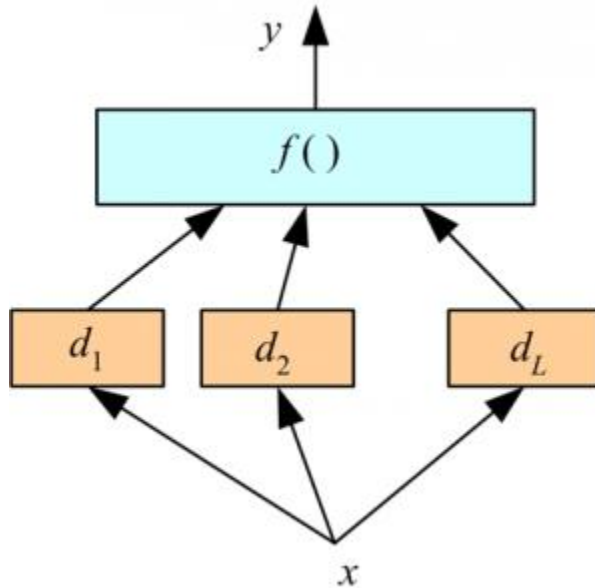
A random forest takes a random subset of features from the data, and creates n random trees from each subset. Trees are aggregated together at end.

Similar to BAGGING, bootstrapped subsamples are pulled from a larger dataset. A decision tree is formed on each subsample. HOWEVER, the decision tree is split on different features (in this diagram the features are represented by shapes).

2. Boosting : Boosting is an iterative technique which adjust the weight of an observation based on the last classification. If an observation was classified incorrectly, it tries to increase the weight of this observation and vice versa. Boosting in general decreases the bias error and builds strong predictive models. However, they may sometimes over fit on the training data.



3. Stacking : This is a very interesting way of combining models. Here we use a learner to combine output from different learners. This can lead to decrease in either bias or variance error depending on the combining learner we use.



1.6 BAYES THEOREM AND NAÏVE BAYES CLASSIFIER

The theorem provides a way to revise existing predictions or theories given new or additional evidence. In finance, Bayes' theorem can be used to rate the [risk](#) of lending money to potential borrowers.

The formula is as follows:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A) \times P(B|A)}{P(B)}$$

Bayes' theorem is also called Bayes' Rule or Bayes' Law.

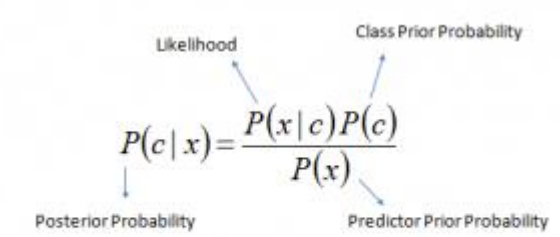
Naïve bayes:

It is a classification technique based on **Bayes' Theorem** with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently

contribute to the probability that this fruit is an apple and that is why it is known as ‘Naive’.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability $P(c|x)$ from $P(c)$, $P(x)$ and $P(x|c)$. Look at the equation below:



The diagram shows the equation $P(c|x) = \frac{P(x|c)P(c)}{P(x)}$ with four labels and arrows: 'Likelihood' points to $P(x|c)$, 'Class Prior Probability' points to $P(c)$, 'Posterior Probability' points to $P(c|x)$, and 'Predictor Prior Probability' points to $P(x)$.

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$
$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \dots \times P(x_n|c) \times P(c)$$

