MACHINE LEARNING

CONTENTS

3. REGRESSION

LINEAR REGRESSION

3.1.1. SIMPLE LINEAR REGRESSION

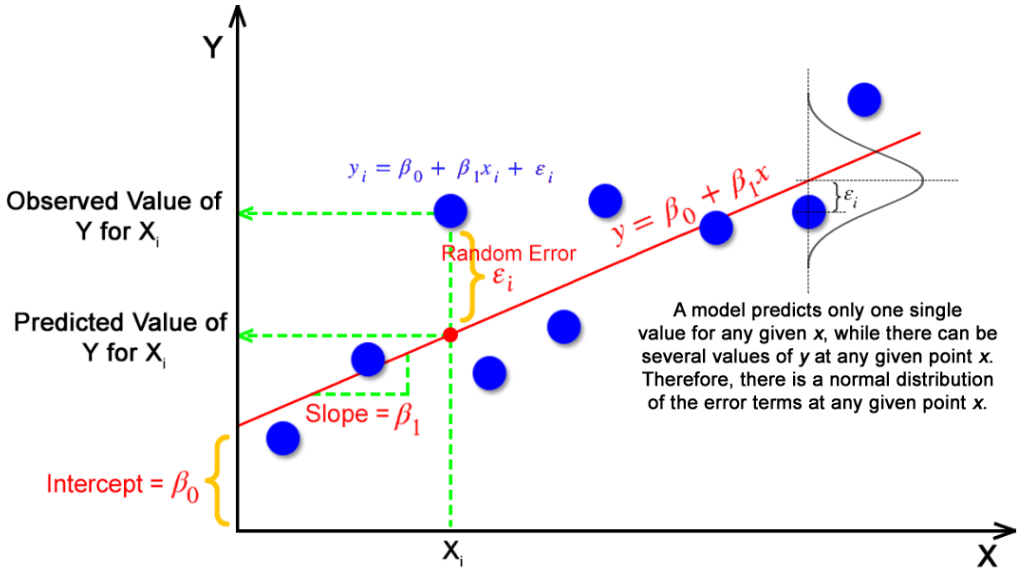
Models use machine learning algorithms, in which the machine learns from the data just like humans learn from their experiences. Machine learning models can be broadly divided into two categories based on the learning algorithm which can further be classified based on the task performed and the nature of the output.

1. Supervised learning methods : Past data with labels is used for building the model.
    a. Regression : The output variable to be predicted is a continuous variable, e.g. scores of a student.
    b. Classification : The output variable to be predicted is a categorical variable, e.g. classifying incoming emails as spam or ham.

2. Unsupervised learning methods : No predefined labels are assigned to past data.
    a. Clustering : No predefined notion of a label is allocated to groups/clusters formed, e.g. customer segmentation.

Simple Linear Regression

It is a model with only one independent variable. It attempts to explain the relationship between a dependent (output) variable y and an independent (predictor) variable x using a straight line $y = \beta_0 + \beta_1 x$. The following figure explains the above geometrically.



Best Fit Line

The best fit line is a straight line that is the best approximation of the given set of data. The equation for the best fitting line is,

$$y_{Predicted} = \beta_0 + \beta_1 x$$

Cost Function

The line that fits the data best will be the one for which the n prediction errors (one for each observed data point) are as small as possible in some overall sense. One way to achieve this is to use the least squares criterion, which minimizes the sum of all the squared prediction errors ε_i (difference between actual value and predicted value of the dependent variable y), i.e.

$$\underset{J(\beta_0,\beta_1)}{Min} \quad where,\ J(\beta_0,\beta_1) = \sum_{i=1}^{n} \varepsilon_i{}^2 = \sum_{i=1}^{n} \left(y_i - y_{Predicted}\right)^2 = \sum_{i=1}^{n} \left(y_i - \beta_0 - \beta_1 x_i\right)^2$$

The function $J(\beta_0, \beta_1)$ mentioned above, is known as the cost function (also referred to as loss or error function). It is a measure of how wrong the model is in terms of its ability to estimate the relationship between x and y. This function helps us in reaching at optimal values for β_i's by either minimizing or maximizing the function.

Optimizing Cost Function

The cost function can be optimized by using any of the following optimization methods.

1. Closed Form Solution : It is a mathematical process that can be completed in a finite number of operations. A closed form solution is nearly always desirable because it means that a solution can be found efficiently. Unfortunately, closed form solutions are not always possible.

$$\frac{\partial}{\partial \theta} J(\theta) = 0$$

2. Iterative Form Solution : It is a mathematical process that uses an initial guess to generate a sequence of improving approximate solutions for a class of problems, in which the n^{th} approximation is derived from the previous $(n-1)^{th}$ ones. An iterative method is called convergent if the corresponding sequence converges for given initial approximations. Iterative methods are often the only choice for nonlinear equations. However, iterative methods are often useful even for linear problems involving a large number of variables (sometimes of the order of millions), where direct methods would be prohibitively expensive (and in some cases impossible) even with the best available computing power.

    a. First Order (Gradient Descent Method)

$$\frac{\partial}{\partial \theta} J(\theta) = 0 \qquad applying\ iterations$$

    b. Second Order (Newton's Method)

$$\frac{\partial^2}{\partial^2 \theta} J(\theta) = 0 \qquad applying\ iterations$$

Optimizing Cost Function using Gradient Descent Method

Gradient descent is a way to minimize an objective function $J(\theta)$, parameterized by a model's parameters $\theta \in \Re$, by updating the parameters in the opposite direction of the gradient of the objective function $\frac{\partial}{\partial \theta} J(\theta)$ w.r.t. to the parameters. The learning rate η determines the size of the steps taken to reach a (local/global) minimum. In other words, one follows the direction of the slope of the surface created by the objective function downhill until a valley is reached.

Mathematically,

1. Start at point θ_0

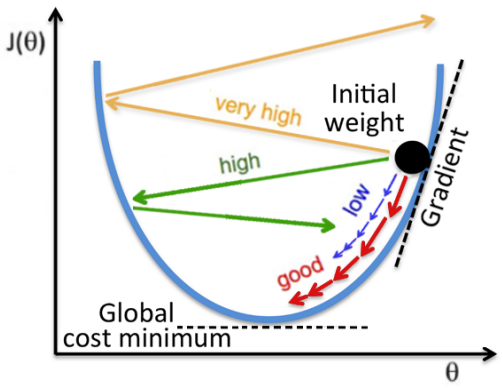
2. Move to new points $\theta_1, \theta_2, \theta_3 \ldots\ldots$ such that,

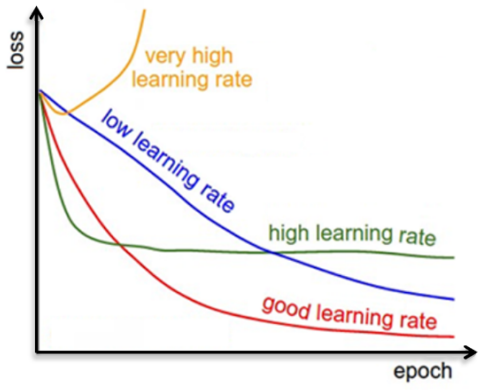
$$\theta_{t+1} = \theta_t - \eta \frac{\partial J(\theta)}{\partial \theta}\Bigg|_{\theta = \theta_{t+1}}$$

$$where,\ J(0)\ J(1)\ J(2)\ \ldots\ldots\ follows\ a\ monotonic\ sequence$$

3. Move till the sequence θ_t converges to the local minimum.

The term η, called the Learning Rate controls the steps taken in downward direction in each iteration. If it is too low, the algorithm may take longer to reach the minimum value. On the other hand, if it is high, the algorithm may overstep the minimum value. The following figures demonstrate the different scenarios one can encounter while configuring the learning rate.





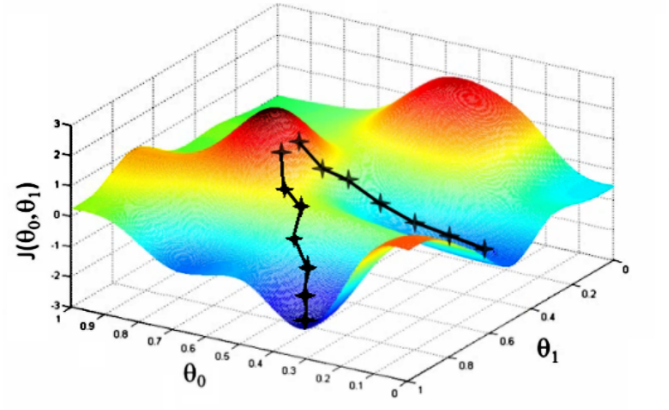
The various variations of Gradient Descent are,

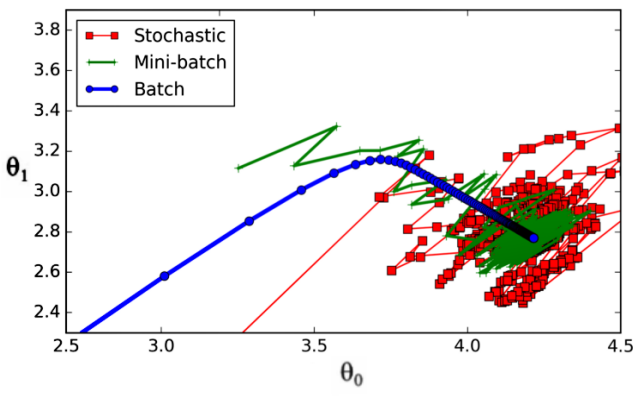
1. Batch or Vanilla gradient descent : It computes the gradient of the cost function w.r.t. to the parameters θ for the entire training dataset.

2. Stochastic gradient descent : In contrast to batch gradient descent it performs a parameter update for each training example θ_t.

3. Mini-Batch gradient descent : It takes the best of both worlds and performs an update for every mini-batch of n training examples.

The following figures show the paths taken by various gradient descent variations for reaching the local minima.





Strength of Linear Regression Model

The strength of any linear regression model can be assessed using various metrics. These metrics usually provide a measure of how well the observed outcomes are being replicated by the model, based on the proportion of total variation of outcomes explained by the model.

The various metrics are,

1. Coefficient of Determination or R-Squared (R2)
2. Root Mean Squared Error (RSME) and Residual Standard Error (RSE)

Coefficient of Determination or R-Squared (R2)

R-Squared is a number which explains what portion of the given data variation is explained by the developed model. It is basically the square of the Pearson's R correlation value between the variables. It always takes a value between 0 & 1. Overall, the higher the R-squared, the better the model fits the data.

Mathematically it can be represented as,

$$R^2 = 1 - \frac{RSS}{TSS}$$

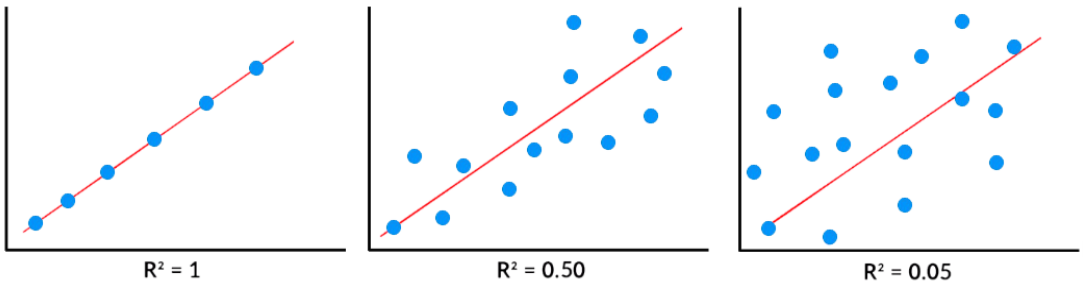
1. Residual Sum of Squares (RSS) is defined as the total sum of error across the whole sample. It is the measure of the difference between the expected and the actual output. A small RSS indicates a tight fit of the model to the data. Mathematically RSS is,

$$RSS = \sum_{i=1}^{n} \left(y_i^{Actual} - y_i^{Predicted} \right)^2 = \sum_{i=1}^{n} \left(y_i - \left(\beta_o + \beta_1 x_i \right) \right)^2$$

2. Total Sum of Squares (TSS) is defined as the sum of errors of the data points from the mean of the response variable. Mathematically TSS is,

$$TSS = \sum_{i=1}^{n} \left(y_i - \bar{y} \right)^2$$

The following figures show the significance of R2.



Root Mean Squared Error (RSME) and Residual Standard Error (RSE)

The Root Mean Squared Error is the square root of the variance of the residuals. It indicates the absolute fit of the model to the data i.e. how close the observed data points are to the model's predicted values. Mathematically it can be represented as,

$$RMSE = \sqrt{\frac{RSS}{n}} = \sqrt{\sum_{i=1}^{n} \left(y_i^{Actual} - y_i^{Predicted} \right)^2 / n}$$

To make this estimate unbiased, one has to divide the sum of the squared residuals by the degrees of freedom rather than the total number of datapoints in the model. This term is then called the Residual Standard Error. Mathematically it can be represented as,

$$RSE = \sqrt{\frac{RSS}{df}} = \sqrt{\sum_{i=1}^{n} \left(y_i^{Actual} - y_i^{Predicted} \right)^2 / (n-2)}$$

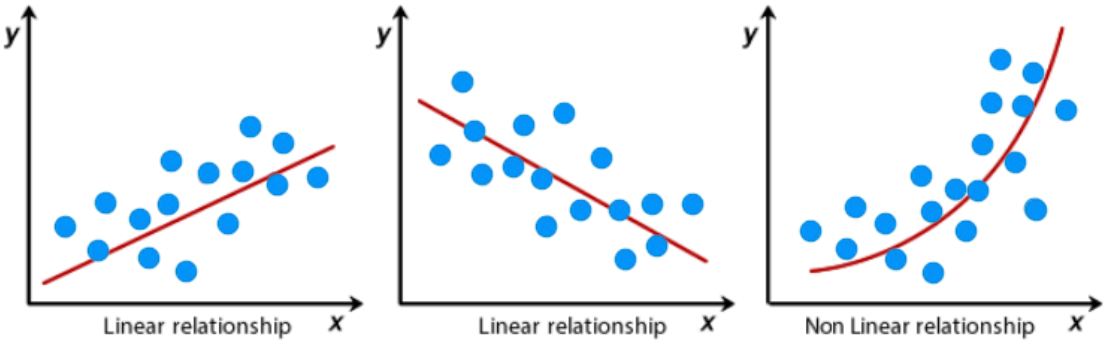
The R-squared is a relative measure of fit, whereas the RMSE is an absolute measure of fit. Because the value of RMSE depends on the units of the variables (i.e. it is not a normalized measure), it can change with the change in the unit of the variables. Thus, R-squared is a better measure than RSME.

Assumptions of Simple Linear Regression

Regression is a parametric approach, meaning it makes assumptions about the data for the purpose of analysis. Due to its parametric side, regression is restrictive in nature and fails to deliver good results with data sets which don't fulfill its assumptions. Therefore, for a successful regression analysis, it's essential to validate the following assumptions.

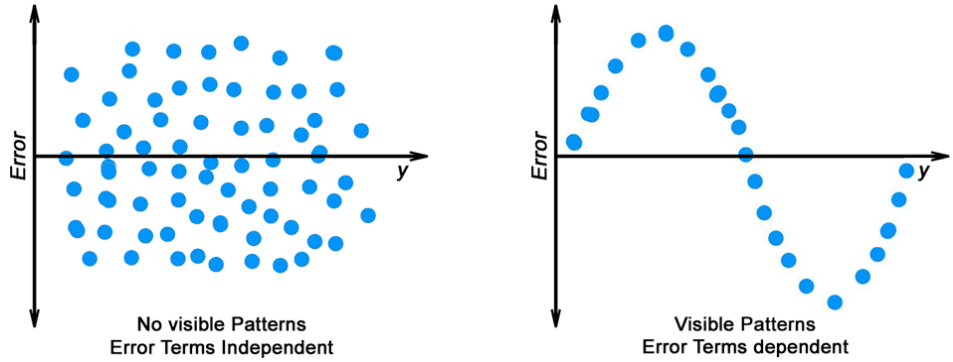
1. Linearity of residuals : There needs to be a linear and additive relationship between the dependent variable and independent variable(s). A linear relationship suggests that a change in response y due to one unit change in x is constant, regardless of the value of x. An additive relationship suggests that the effect of x on y is independent of other variables.

   If a linear model is fit to a non-linear, non-additive data set, then the regression algorithm would fail to capture the trend mathematically, thus resulting in an inefficient model. Also, this will result in erroneous predictions on an unseen data set.



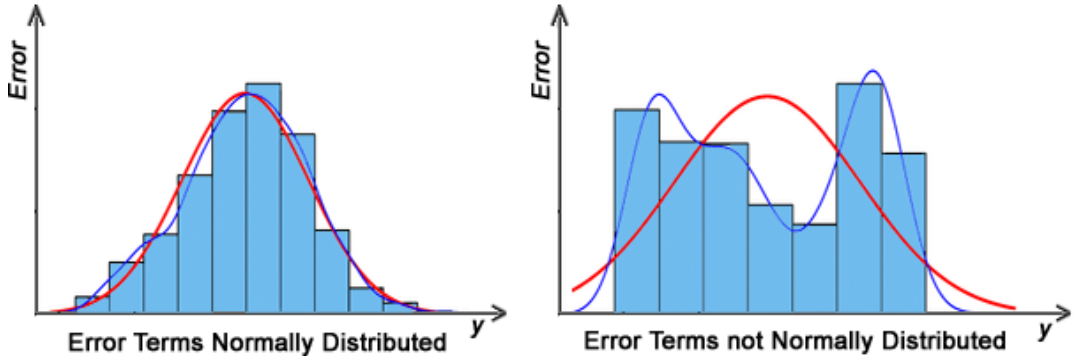
2. Independence of residuals : The error terms should not be dependent on one another (like in a time-series data wherein the next value is dependent on the previous one). There should be no correlation between the residual (error) terms. Absence of this phenomenon is known as Autocorrelation.

   Autocorrelation causes the confidence intervals and prediction intervals to be narrower. Narrower confidence interval means that a 95% confidence interval would have lesser probability than 0.95 that it would contain the actual value of coefficients.



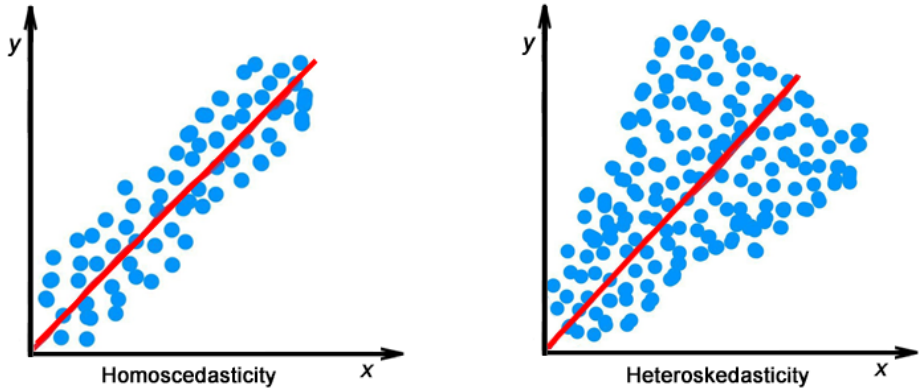
3. Normal distribution of residuals : The mean of residuals should follow a normal distribution with mean equal to zero or close to zero. This is done in order to check whether the selected line is actually the line of best fit or not.

   If the error terms are non-normally distributed, the confidence intervals may become too wide or narrow. Once the confidence interval becomes unstable, it leads to difficulty in estimating coefficients based on minimisation of least squares. This also suggests that there are a few unusual data points which must be studied closely to make a better model.



4. Equal variance of residuals : The error terms must have constant variance. This phenomenon is known as homoscedasticity. The presence of non-constant variance is referred to heteroskedasticity.

   The presence of non-constant variance in the error terms results in heteroskedasticity. Generally, non-constant variance arises in the presence of outliers or extreme leverage values and when these values get too much weight, they disproportionately influence the model's performance. Thereby, causing the confidence interval to be unrealistically wide or narrow for the out of sample predictions.



Hypothesis Testing in Linear Regression

Every time a linear regression is performed, one needs to test whether the fittest line (the coefficient β_1) is significant or not. And in comes the idea of Hypothesis Testing on β_1 .

1. Null Hypothesis (H_0) : $\beta_1 = 0$
2. Alternate Hypothesis (H_A) : $\beta_1 \neq 0$
3. t-score = $(X - \mu) / (s/\sqrt{n}) = (\widehat{\beta}_1 - 0) / SE(\widehat{\beta}_1)$
4. p-value is calculated from the cumulative probability for the given t-score using the t-table.
5. If the p-value is less than 0.05, the null hypothesis is rejected stating β_1 is significant.

Python Code - Simple Linear Regression using Statsmodels

Visualize Data

```
>> sns.pairplot(df,x_vars=[columns],y_vars=column,size=4,aspect=1,kind='scatter')
```

Check Collinearity

```
>> sns.heatmap(df.corr(), cmap="YlGnBu", annot=True)
```

Create Train & Test Data

```
>> from sklearn.model_selection import train_test_split
>> X_train,X_test,y_train,y_test = train_test_split(X, y, train_size=0.7,
                                    test_size=0.3, random_state=100)
```

Scaling Features

```
>> from sklearn.preprocessing import StandardScaler, MinMaxScaler
>> scaler = StandardScaler() # Any one of the scaling methods can be used
>> scaler = MinMaxScaler() # Any one of the scaling methods can be used
>> X_train_scaled = scaler.fit_transform(X_train.values.reshape(-1,1))
>> X_test_scaled = scaler.transform(X_test.values.reshape(-1,1))
#  It is a general convention in scikit-learn that observations are rows, while
features are columns. This is needed only when using a single feature
>> import statsmodels.api as sm
>> X_train_scaled_sm = sm.add_constant(X_train_scaled)
>> X_test_scaled_sm = sm.add_constant(X_test_scaled)
# In statsmodels intercept variable needs to be added explicitly
```

Train Model

```
>> model = sm.OLS(y_train, X_train_scaled_sm).fit()
```

Analyze Model

```
>> model.params
>> model.summary()
# Statsmodel provides an extensive summary of various metrics
>> plt.scatter(X_train_scaled, y_train)
>> plt.plot(X_train_scaled, coeff_A + coeff_B * X_train_scaled, 'r')
```

Analyze Residuals

```
>> y_train_pred = model.predict(X_train_scaled_sm)
>> residual = (y_train - y_train_pred)
>> sns.distplot(residual, bins = 15) # Checking if residuals are normally distributed
>> plt.scatter(X_train_scaled, residual) # Checking for independence of residuals
```

Predict

```
>> y_pred = model.predict(X_test_scaled_sm)
```

Evaluate Model

```
>> from sklearn.metrics import mean_squared_error
>> from sklearn.metrics import r2_score
```

```
>> rsme = np.sqrt(mean_squared_error(y_test, y_pred))
>> r_squared = r2_score(y_test, y_pred)
```

Visualize Model

```
>> plt.scatter(X_test_scaled, y_test)
>> plt.plot(X_test_scaled, coeff_A + coeff_B * X_test_scaled, 'r')
```

Python Code - Simple Linear Regression using Scikit-Learn

Visualize Data

```
>> sns.pairplot(df,x_vars=[columns],y_vars=column,size=4, aspect=1,kind='scatter')
```

Check Collinearity

```
>> sns.heatmap(df.corr(), cmap="YlGnBu", annot=True)
```

Create Train & Test Data

```
>> from sklearn.model_selection import train_test_split
>> X_train,X_test,y_train,y_test = train_test_split(X, y, train_size=0.7,
                                    test_size=0.3, random_state=100)
```

Scaling Features

```
>> from sklearn.preprocessing import StandardScaler, MinMaxScaler
>> scaler = StandardScaler() # Any one of the scaling methods can be used
>> scaler = MinMaxScaler() # Any one of the scaling methods can be used
>> X_train_scaled = scaler.fit_transform(X_train.values.reshape(-1,1))
>> X_test_scaled = scaler.transform(X_test.values.reshape(-1,1))
#  It is a general convention in scikit-learn that observations are rows, while
features are columns. This is needed only when using a single feature
```

Train Model

```
>> from sklearn.linear_model import LinearRegression
>> model = LinearRegression()
>> model.fit(X_train_scaled, y_train)
```

Analyze Model

```
>> coeff_A = model.intercept_
>> coeff_B = model.coef_[0]
>> plt.scatter(X_train_scaled, y_train)
>> plt.plot(X_train_scaled, coeff_A + coeff_B * X_train_scaled, 'r')
```

Analyze Residuals

```
>> y_train_pred = model.predict(X_train_scaled)
>> residual = (y_train - y_train_pred)
>> sns.distplot(residual, bins = 15) # Checking if residuals are normally distributed
>> plt.scatter(X_train, residual) # Checking for independence of residuals
```

Predict

```
>> y_pred = model.predict(X_test_scaled)
```

Evaluate Model

```
>> from sklearn.metrics import mean_squared_error
>> from sklearn.metrics import r2_score
>> rsme = np.sqrt(mean_squared_error(y_test, y_pred))
>> r_squared = r2_score(y_test, y_pred)
```

Visualize Model

```
>> plt.scatter(X_test_scaled, y_test)
>> plt.plot(X_test_scaled, coeff_A + coeff_B * X_test_scaled, 'r')
```

3.1.2. MULTIPLE LINEAR REGRESSION

The term multiple in multiple linear regression gives a fair idea in itself. It represents the relationship between one dependent variable (response variable) and several independent variables (explanatory variables). The objective of multiple regression is to find a linear equation that can best determine the value of the dependent variable y for different values independent variables in x.

Multiple Linear Regression

Most of the concepts in multiple linear regression are quite similar to those in simple linear regression. It is basically an extension of the earlier equation $y = \beta_0 + \beta_1 x$ to add more factors.

Best Fit Line

The equation for the best fitting line with k predictors is,

$$y_{Predicted} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_1 x_1 + \ldots\ldots + \beta_k x_k$$

The above equation can be interpreted as the coefficient β_i for any independent variable x_i gives the amount of increase in mean response of y per unit increase in the variable x_i provided all the other predictors are held constant. The model now fits a hyperplane instead of a line.

Cost Function

$$\frac{Min}{J\left(\beta_0,\beta_1,..\beta_k\right)} \quad where,\ J\left(\beta_0,\beta_1,..\beta_k\right) = \sum_{i=1}^{n} \varepsilon_i{}^2 = \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{k}\beta_j x_{ij}\right)^2$$

Optimizing Cost Function

1. Batch Gradient Descent Method : β can be solved for using the Batch Gradient Descent algorithm as follows,

Repeat till convergence is reached such that,

$$\beta_0 := \beta_0 - \eta\frac{\partial}{\partial\beta_0}J\left(\beta_0,\beta_1\ldots\beta_k\right) = \beta_0 - \eta\frac{2}{n}\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{k}\beta_j x_{ij}\right).x_{i0}$$

$$\beta_1 := \beta_1 - \eta\frac{\partial}{\partial\beta_1}J\left(\beta_0,\beta_1\ldots\beta_k\right) = \beta_1 - \eta\frac{2}{n}\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{k}\beta_j x_{ij}\right).x_{i1}$$

.

.

$$\beta_k := \beta_k - \eta\frac{\partial}{\partial\beta_k}J\left(\beta_0,\beta_1\ldots\beta_k\right) = \beta_k - \eta\frac{2}{n}\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{k}\beta_j x_{ij}\right).x_{ik}$$

$$simultaneously\ updating\ for\ each\ j = (0,1,\ldots k)$$

2. Normal Equation : It is an analytical approach to solving the cost function. One can directly find the value of β without using Gradient Descent. This approach is an effective and a time-saving option when working with a dataset with small features. β is given by,

$$\beta = (X^T X)^{-1} X^T y \quad where, \ \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ . \\ . \\ \beta_k \end{bmatrix} X = \begin{bmatrix} 1 & x_{11} & x_{12} & . & . & x_{1k} \\ 1 & x_{21} & x_{22} & . & . & x_{2k} \\ . & . & . & & & . \\ . & . & . & & & . \\ 1 & x_{n1} & x_{n2} & . & . & x_{nk} \end{bmatrix} y = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_n \end{bmatrix}$$

This of course works only if the inverse exists. If the inverse does not exist, the normal equations can still be solved, but the solution may not be unique. The inverse of $X^T X$ exists, if the columns of X are linearly independent (i.e. no column can be written as a linear combination of the other columns).

Assumptions of Multiple Linear Regression

All the four assumptions made for Simple Linear Regression (Linearity of residuals, Independence of residuals, Normal distribution of residuals, Equal variance of residuals) still hold true for Multiple Linear Regression along with a few new additional assumptions.

1. Overfitting : When more and more variables are added to a model, the model may become far too complex and usually ends up with memorizing all the data points in the training set. This phenomenon is known as overfitting of a model. Overfitting causes the model to become specific rather than generic. This usually leads to high training accuracy and very low test accuracy.

2. Multicollinearity : It is the phenomenon where a model with several independent variables, may have some variables interrelated. When the variables are correlated to each other, they can easily explain each other and thus, their presence in the model becomes redundant.

3. Feature Selection : With more variables present, selecting the optimal set of predictors from the pool of given features (many of which might be redundant) becomes an important task for building a relevant and better model.

Multicollinearity

Multicollinearity refers to the phenomenon of having related predictor variables in the input dataset. As multicollinearity makes it difficult to find out which variable is actually contributing towards the prediction of the response variable, it leads one to conclude incorrectly the effects (strong/weak) of a variable on the target variable. Thus, it is a big issue when one is trying to interpret the model. Though it does not affect the precision of the predictions, it is essential to properly detect and deal with the multicollinearity present in the model, as random removal of any of these correlated variables from the model cause the coefficient values to swing wildly and even change signs. Multicollinearity can be detected using the following methods.

1. Pairwise Correlations : Checking the pairwise correlations between different pairs of independent variables can throw useful insights in detecting multicollinearity.

2. Variance Inflation Factor (VIF) : Pairwise correlations may not always be useful as it is possible that just one variable might not be able to completely explain some other variable but some of the variables combined might be able to do that. Thus, to check these sorts of

relations between variables, one can use VIF. VIF basically explains the relationship of one independent variable with all the other independent variables.
VIF is given by,

$$VIF_i = \frac{1}{1 - R_i^2}$$

where,

i refers to the i^{th} variable which is being represented as a linear combination of the rest of the independent variables.

The common heuristic followed for the VIF values is,

a. $VIF > 10$ indicates definite removal of the variable.
b. $VIF > 5$ indicates variable needs inspection.
c. $VIF < 5$ indicates the variable is good to go.

Once multicollinearity has been detected in the dataset then this can be dealt using the following methods.

1. Highly correlated variables can be dropped.
2. Business Interpretable variables can be picked up.
3. New interpretable features can be derived using the correlated variables.
4. Variable transformations can be done using PCA (Principal Component Analysis) or PLS (Partial Least Squares).

Dummy Variables

While dealing with multiple variables, there may be some categorical variables that might turn out to be useful for the model. So, it is essential to handle these variables appropriately in order to get a good model. These can be dealt with using the following methods.

1. Label Encoding : It is used to transform the non-numerical categorical variables to numerical labels. Numerical labels are always between 0 and $n-1$. This is usually performed on ordinal categorical variables.

2. One Hot Encoding : One way to deal with the nominal categorical variables is to create n new columns (dummy variables) each indicating whether that level exists or not using a zero or one. For example for a variable say, Relationship with three levels namely, Single, In a relationship and Married, one can create a dummy table like the following.

Relationship Status	Single	In a relationship	Married
Single	1	0	0
In a relationship	0	1	0
Married	0	0	1

Dummy Variable Trap is a scenario where there are attributes which are highly correlated (multicollinear). After using one hot encoding, any one of the dummy variables can be predicted with the help of the other remaining dummy variables. Hence, one dummy variable is always highly correlated with the other remaining dummy variables. Thus, one needs to exclude one of the dummy variables before proceeding with the model.

3. Dummy Encoding : Another way to deal with the nominal categorical variables is to create $n-1$ dummy variables rather than n as is done in case of one hot encoding. Taking the same example as above one can drop the Single status from among the dummy variables.

Relationship Status	Single (base)	In a relationship (x_1)	Married (x_2)
Single	1	0	0
In a relationship	0	1	0
Married	0	0	1

It can be clearly seen that there is no need of defining three different levels as even after dropping one of the levels, one can still be able to explain the three levels. Here, the Single status is the base state and the other two statuses represent the effect of that state vs the base state. This can be mathematically represented as follows.

$$y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \begin{cases} \beta_0 & if\ \ i^{th}\ status\ is\ Single \\ \beta_0 + \beta_1 & if\ \ i^{th}\ status\ is\ In\ Relationship \\ \beta_0 + \beta_2 & if\ \ i^{th}\ status\ is\ Married \end{cases}$$

Feature Scaling

Another important aspect to consider is feature scaling. When there are many independent variables in a model, a lot of them might be on very different scales. This leads to a model with very weird coefficients, which are difficult to interpret. Thus, one needs to scale the features for ease of interpretation of the coefficients as well as for the faster convergence of gradient descent methods. One can scale the features using the following methods.

1. Standardization : The variables are scaled in such a way that their mean is zero and standard deviation is one.

$$x' = \frac{x - \bar{x}}{\sigma} \quad where\ \ \bar{x}\ is\ mean,\ \sigma\ is\ standard\ deviation$$

2. Min Max Scaling : The variables are scaled in such a way that all the values lie between zero and one.

$$x' = \frac{x - min(x)}{max(x) - min(x)}$$

It is important to note that scaling just affects the coefficients and none of the other parameters like t-statistic, F statistic, p-values, R-square, etc. Also the dummy variables are usually never scaled for the ease of their interpretation.

Feature Selection

There may be quite a few potential predictors for the model, but choosing the correct features (not the redundant ones but the ones that add some value to the model) is quite essential. To get the optimal model, one can always try all the possible combinations of independent variables and check which model fits the best. But this method is obviously time-consuming and infeasible. Hence, one needs some other method to get a decent model. Following is a list of different approaches for choosing the best set of predictors that will give the least test error.

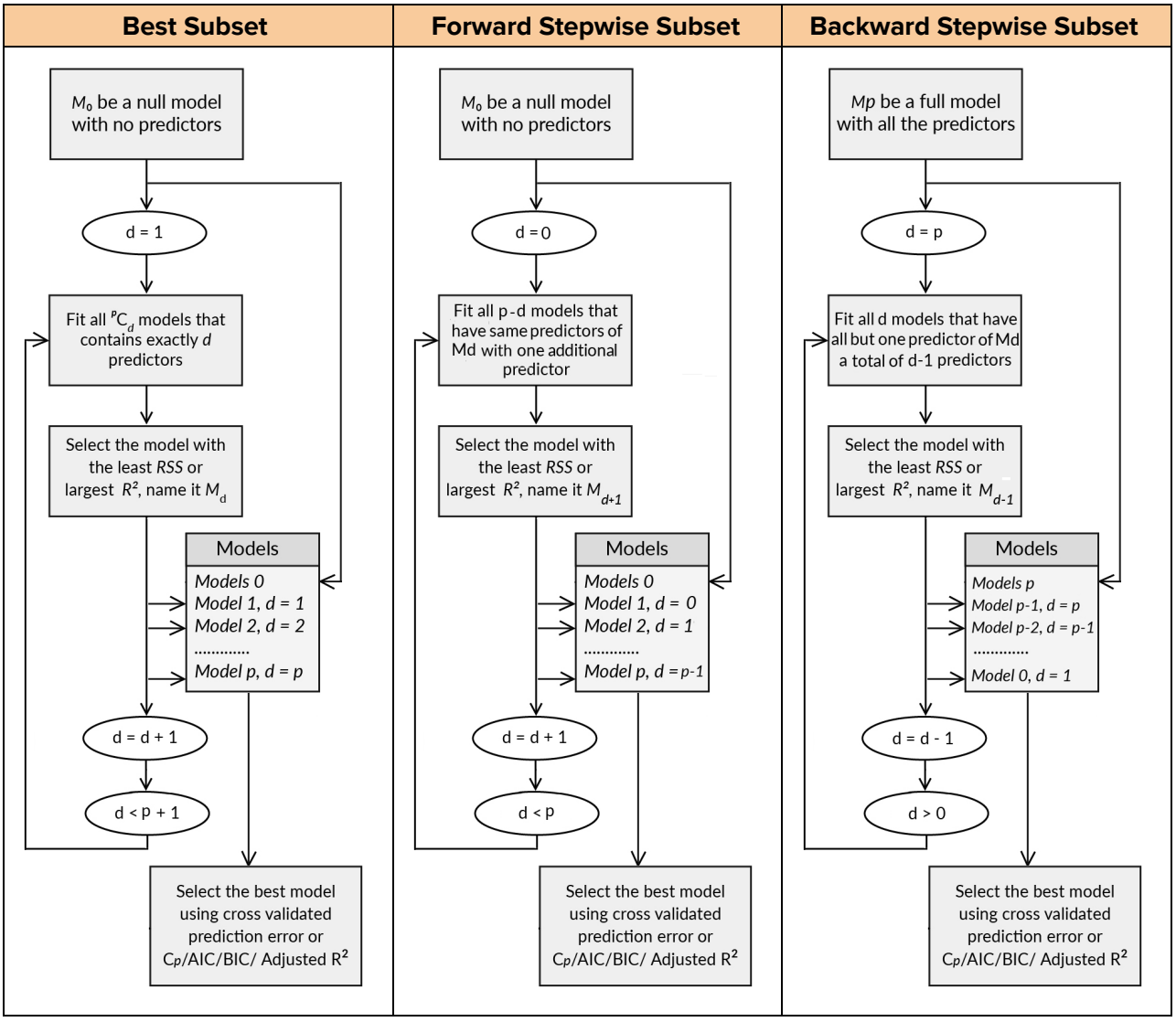
1. Manual Feature Elimination Approach
    a. Build the model with all the features
    b. Drop the features that are least helpful in prediction (high p-value)
    c. Drop features that are redundant (using correlations, VIF)
    d. Rebuild the model
    e. Repeat the above steps till a good model is reached

2. Automated Feature Elimination Approach
    a. Recursive Feature Elimination (RFE)
    b. Best Subset Selection method
    c. Forward/Backward Stepwise Selection method
    d. Lasso Regularization

3. Balanced Feature Elimination Approach
    a. Use automated feature elimination for the coarse tuning (i.e. remove most of the features to drastically bring down the number of features to a minimum number)
    b. Use manual feature elimination for fine tuning on the remaining small set of potential variables

Automated Feature Selection

The following table provides a comparison of the feature selection algorithms.



Total number of models to be built is 2^p	Total number of models to be built is $1+\frac{p(p+1)}{2}$	Total number of models to be built is $1+\frac{p(p+1)}{2}$
It cannot be used when $p>40$ as it becomes computationally infeasible.	It can be applied even when $n<p$ where n is the number of observations.	It cannot be applied when $n<p$ where n is the number of observations, as a full model cannot be fit.

Model Assessment

While creating the best model for any problem statement, one ends up choosing from a set of models which gives the least test error. Hence, the test error, and not only the training error, needs to be estimated in order to select the best model. Besides, selecting the best model to obtain decent predictions, one also needs to maintain a balance between keeping the model simple and explaining the highest variance (i.e. keeping as many required variables as possible).

R-squared is never used for comparing the models as the value of R2 increases with the increase in the number of predictors (even if these predictors did not add any value to the model). Thus, in came the idea to penalize the model for every predictor variable (if the variable did not improve the model more than would be expected by chance) being added to the model. This can be done in the following two ways.

1. Use metrics which take into account both model fit and simplicity, such metrics are Mallow's Cp, Adjusted R2, AIC and BIC.
2. Estimate the test error via a validation set or a cross-validation approach.

For a model where,

n is the number of training examples,
d is the number of predictors,
σ^2 is the estimate of the variance of training error,
D is the entire dataset,
M is the model,
$ln\,P(D|M)$ is the model likelihood,

$$TSS\left(\sum_{i=1}^{n}(y_i-\bar{y})^2\right)=ESS\left(\sum_{i=1}^{n}(\hat{y_i}-\bar{y})^2\right)+RSS\left(\sum_{i=1}^{n}(\hat{y_i}-y_i)^2\right)$$

Mallow's CP

$$C_p=\frac{1}{n}\left(RSS+2d\sigma^2\right)$$

AIC (Akaike Information Criterion)

$$AIC=\frac{1}{n\sigma^2}\left(RSS+2d\sigma^2\right)\approx -2\ln P(D|M)+2k$$

BIC (Bayesian Information Criterion)

It is valid for sample size much larger than the number of parameters in the model.

$$BIC=\frac{1}{n}\left(RSS+\ln(n)\,d\sigma^2\right)\approx -2\ln P(D|M)+k\ln(n)$$

Adjusted R2

It can have a negative value, if the predictors do not explain the dependent variables at all such that $RSS \sim TSS$.

$$Adjusted\ R^2 = 1 - \frac{RSS/(n-d-1)}{TSS/(n-1)} = 1 - (1-R^2)\left(\frac{n-1}{n-d-1}\right)$$

Python Code - Multiple Linear Regression

Visualize Data

```
>> sns.pairplot(df)
>> sns.boxplot(x='column', y='column1', hue='column2', data=df)
```

Prepare Data

```
>> def binary_map(x):
       return x.map({'yes': 1, 'no': 0})

>> df[column_list] = df[column_list].apply(binary_map)
# Handling the categorical variables using custom binary encoding
>> from sklearn.preprocessing import LabelEncoder
>> label_encoder = LabelEncoder()
>> df[column_list] = label_encoder.fit_transform(df[column_list])
# Handling of ordinal categorical variables using label encoding
>> dummy_df = pd.get_dummies(df[column_list], drop_first=True)
>> df = pd.concat([df, dummy_df], axis=1)
>> df.drop([column_list], axis=1, inplace=True)
# Handling the nominal categorical variables using dummy method
>> from sklearn.preprocessing import OneHotEncoder
>> one_hot_encoder = OneHotEncoder(handle_unknown='ignore')
>> one_hot_encoder.fit_transform(df)
# Handling the nominal categorical variables using one hot encoding. It gives a
sparse matrix as output
```

Create Train & Test Data

```
>> from sklearn.model_selection import train_test_split
>> X_train,X_test,y_train,y_test = train_test_split(X, y, train_size=0.7,
                                   test_size=0.3, random_state=100)
```

Scaling Features

```
>> from sklearn.preprocessing import StandardScaler
>> scaler = StandardScaler()
>> X_train_scaled[column_list] = scaler.fit_transform(X_train[column_list])
>> X_test_scaled[column_list] = scaler.transform(X_test[column_list])
```

Train Model

```
>> from sklearn.linear_model import LinearRegression
>> model = LinearRegression()
>> model.fit(X_train, y_train)
```

Automated Feature Selection (RFE)

```
>> from sklearn.feature_selection import RFE
>> rfe = RFE(model, no_of_features_to_be_selected)
>> rfe = rfe.fit(X_train, y_train)
>> list(zip(X_train.columns,rfe.support_,rfe.ranking_))
>> selected_features = X_train.columns[rfe.support_]
>> insignificant_features = X_train.columns[~rfe.support_]
```

Analyze Model using Statsmodels

```
>> import statsmodels.api as sm
>> X_train_rfe = X_train[selected_features]
>> X_train_rfe = sm.add_constant(X_train_rfe)
# In statsmodels intercept variable needs to be added explicitly
>> model_1 = sm.OLS(y_train,X_train_rfe).fit()
>> model_1.summary()
```

Checking VIF

```
>> from statsmodels.stats.outliers_influence import variance_inflation_factor
>> vif = pd.DataFrame()
>> vif['VIF'] = [variance_inflation_factor(X_train_rfe.values, i)
                for i in range(X_train_rfe.shape[1])]
>> vif['VIF'] = round(vif['VIF'], 2)
>> vif['Features'] = X_train_rfe.columns
>> vif = vif.sort_values(by='VIF', ascending=False)
```

Manual Feature Selection

```
# Remove only one feature at a time
# Remove the feature with p-value > 0.05
# Remove the feature with VIF > 5
>> selected_features.remove(feature_to_be_removed)
>> X_train_manual = X_train[selected_features]
>> X_train_manual = sm.add_constant(X_train_manual)
# Repeat steps Analyze Model using Statsmodels, Checking VIF and Manual Feature
Selection till all the features are within proper parameters and the model
performance metrics are acceptable
```

Train Final Model

```
>> model_final = LinearRegression()
>> model_final.fit(X_train_manual, y_train)
```

Analyze Residuals

```
>> y_train_pred = model_final.predict(X_train_manual)
>> residual = (y_train - y_train_pred)
>> sns.distplot(residual, bins = 15)
# Checking if residuals are normally distributed
>> plt.scatter(X_train_manual, residual)
# Checking for independence of residuals
```

Predict

```
>> X_test = X_test[selected_features]
>> X_test = sm.add_constant(X_test)
>> y_pred = model_final.predict(X_test)
```

Evaluate Model

```
>> from sklearn.metrics import mean_squared_error
>> from sklearn.metrics import r2_score
>> rsme = np.sqrt(mean_squared_error(y_test, y_pred))
>> r_squared = r2_score(y_test, y_pred)
```

3.1.3. INDUSTRY RELEVANCE OF LINEAR REGRESSION

Some of the important properties of Linear Regression are as follows,

1. In statistical modelling, linear regression is a process of estimating the relationships among variables. The focus here is to establish the relationship between a dependent variable and one or more independent variable(s).

2. Regression explains the change in value of dependent variable with the change in values of one predictor, holding the other predictors static.

3. Regression only shows relationship, i.e. correlation and not causality.

4. Regression analysis is widely used for Forecasting and Prediction. It does guarantee interpolation (predict value within the range of data used for model building) but not necessarily extrapolation.

5. Linear regression is a form of parametric regression.

Prediction vs Projection

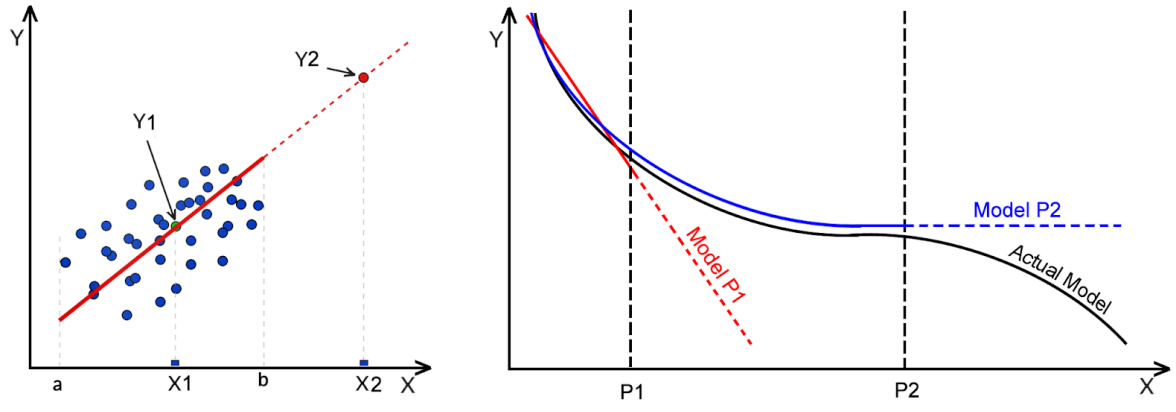
Although prediction and projection sound synonymous but they are different applications in analytics.

Prediction	Projection/Forecast
Identification of predictor variables and measurement of their impact is the main aim.	Finding the final projected result or forecasted value is the main aim.
No new assumptions are made other than those of linear regression.	Forecasts for the next day are made with the assumption that everything remains the same today. Inclusion of any new incidents changes the forecast for the next day.
Simplicity of the model is important.	Accuracy of the model is important and not the model.

Interpolation vs Extrapolation

The basic difference between interpolation and extrapolation is given in the following table,

Interpolation	Extrapolation
Model is used to predict the value of a dependent variable on independent values that lie within the range of data one already has.	Model is used to predict the dependent variable on the independent values that lie outside the range of the data the model was built on.



In the preceding figure when one predicts $Y1$ for $X1$, which lies between a and b, it is called interpolation. On the other hand, extrapolation would be extending the line to predict $Y2$ for $X2$

which lies outside the range on which the linear model was trained. Also one can easily visualize that for the models $P1$ and $P2$ which were trained on values of X lying within $P1$ and $P2$ respectively, upon extrapolation will give incorrect predictions.

Parametric vs Non-Parametric

The basic difference between parametric and non-parametric is given in the following table,

Parametric	Non-Parametric
The number of parameters is fixed with respect to the sample size, simplifying the function to a known form.	The number of parameters can grow with the sample size, freeing it to learn any functional form from the training data.
Simpler and faster to train requiring less training data.	Flexible and highly accurate, but slower to train requiring a lot of training data.
Eg: Simple Neural Networks, Naive Bayes, Logistic Regression, etc.	Eg: k-Nearest Neighbors, Decision Trees like CART and C4.5, SVM. etc.

Constrained Minimisation vs Unconstrained Minimisation

The basic difference between constrained minimisation and unconstrained minimisation is given in the following table,

Constrained Minimisation	Unconstrained Minimisation
It is the process of optimizing a cost function with respect to some variables in the presence of constraints on those variables. The constraints can be either a hard (needs to satisfy) or a soft (penalties for not satisfying) constraint.	It is the process of optimizing a cost function with respect to some variables without any constraints on those variables.
Eg: Ridge/Lasso Regression, SVM etc.	Eg: Linear/Logistic Regression etc.

ADVANCED REGRESSION

3.2.1. GENERALIZED LINEAR REGRESSION

In linear regression problems the dependent variable is always linearly related to the predictor variables. But when this is not the case (i.e the relationship is not linear), the Generalised Regression is used to tackle the problem.

General Equation

$$F(x) = a_0 + a_1 f_1(x) + a_2 f_2(x) + \ldots\ldots + a_k f_k(x)$$

Feature Engineering

Raw attributes, as these appear in the training data, may not be the ones best suited to predict the response variable values. Thus, in comes the derived features, which is a combination of two or more raw attributes and/or transformations of individual raw attributes. These combinations and transformations could be linear (only multiplication by a constant or addition) or nonlinear.

Linear and Non-Linear Model

The term linear in linear regression refers to the linearity in the coefficients, i.e. the target variable y is linearly related to the model coefficients. It does not require that y should be linearly related to the raw attributes or features i.e a model is linear if y is linearly (only multiplication by a constant or addition) related to the coefficients whereas the feature functions could be non-linear.

Data points : $x_1,\ x_2,\ x_3 \ldots\ldots,\ x_d$

Features : $\Phi_1(x),\ \Phi_2(x),\ \Phi_3(x) \ldots\ldots., \Phi_k(x)$

Model : $y = a_0 + a_1\Phi_1(x) + a_2\Phi_2(x) + \ldots\ldots a_k\Phi_k(x)$. Also represented as $y = Xa$, where,

$$y = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_n \end{bmatrix} \quad X = \begin{bmatrix} 1 & \phi_1(x_1) & \phi_2(x_1) & . & . & \phi_k(x_1) \\ 1 & \phi_1(x_2) & \phi_2(x_2) & . & . & \phi_k(x_2) \\ . & . & . & & & . \\ . & . & . & & & . \\ 1 & \phi_1(x_n) & \phi_2(x_n) & . & . & \phi_k(x_n) \end{bmatrix} \quad a = \begin{bmatrix} a_0 \\ a_1 \\ . \\ . \\ a_k \end{bmatrix}$$

Solution : On minimizing the cost function,

$$\frac{Min}{a_0, a_1, a_2, \ldots\ldots, a_k} \quad \sum_{i=1}^{n} \left(y_i - \left(a_0, a_1, a_2, \ldots, a_k \right) \begin{bmatrix} 1 \\ \phi_1(\vec{x}_i) \\ \phi_2(\vec{x}_i) \\ . \\ . \\ \phi_k(\vec{x}_i) \end{bmatrix} \right)^2$$

The final solution is given by $a = (X^T X)^{-1} X^T y$

Feature Matrix

The following matrix provides the feature matrix.

$$X_{(n \times k)} = \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \phi_3(x_1) & . & . & \phi_k(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \phi_3(x_2) & . & . & \phi_k(x_2) \\ . & . & . & & & . \\ . & . & . & & & . \\ \phi_1(x_n) & \phi_2(x_n) & \phi_3(x_n) & . & . & \phi_k(x_n) \end{bmatrix}$$

where,

n *is the number of datapoint in the training dataset*

K *is the number of features*

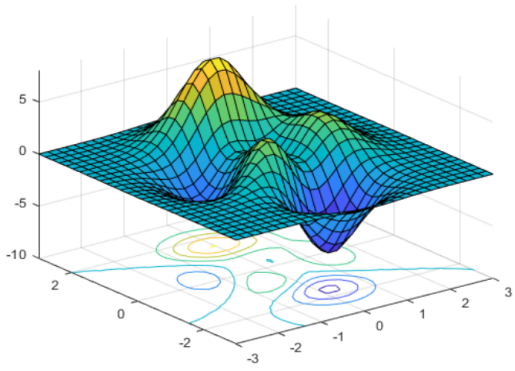
Python Code - Generalized Regression

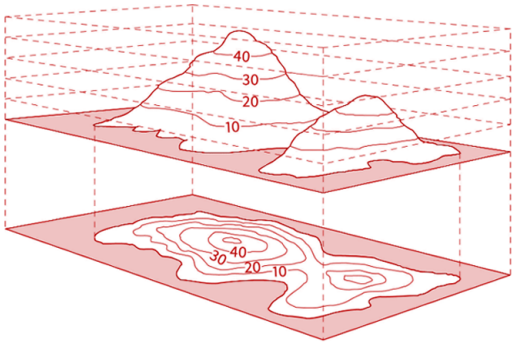
```
>> from sklearn.preprocessing import PolynomialFeatures
>> from sklearn.linear_model import LinearRegression
>> from sklearn.pipeline import make_pipeline
>> from sklearn import metrics
>> pipeline = make_pipeline(PolynomialFeatures(degree), LinearRegression())
>> pipeline.fit(X_train, y_train)
>> y_pred=pipeline.predict(X_test)
>> metrics.r2_score(y_test,y_pred)
```

3.2.2. REGULARIZED REGRESSION

The predictive models need to be as simple as possible, but no simpler as there is an important relationship between the complexity of a model and its usefulness in a learning context. The simpler models are usually more generic, widely applicable and require fewer training samples for effective training in comparison to the more complex models. Thus, the Regularization process is used to create an optimally complex model, i.e. a model which is as simple as possible while performing well on the training data. Through regularization, one tries to strike the delicate balance between keeping the model simple, yet not making it too naive to be of any use.

Contour





For any function $f(x,y)$ when a third axis say z is used to plot the output values of the function and then these values are connected, we get a surface. Now on slicing this surface by a plane and joining all the points where the function value is same gives us a contour.

Regularization Regression

In regularization regression, an additional regularization term is added to the cost function along with the error term, while minimizing the cost function as follows.

$$\frac{Min}{\alpha}\left[\sum_{i=1}^{n}\left(y_i - \alpha\begin{bmatrix}1\\ \phi_1(\vec{x}_i)\\ \vdots\\ \phi_k(\vec{x}_i)\end{bmatrix}\right)^2 + Regularization\ Term\right]$$

Where Regularization terms can be as follows,

1. Ridge Regression (Sum of squares of coefficients) : $\lambda\,\Sigma\alpha^2$
2. Lasso Regression (Sum of absolute values of coefficients) : $\lambda\,\Sigma|\alpha|$

The final solution changes to,

$$\vec{a} = (X^TX + \lambda I)^{-1}X^Ty$$

Ridge & Lasso Regularization

Ridge regression almost always has a matrix representation for the solution while Lasso regression requires iterations to get to the final solution. Thus, making Lasso regression computationally more intensive.

The cost function for both ridge and lasso can also be represented as $\Sigma(\omega^T x_i - y_i)^2 + \lambda R(\omega)$ where, ω is the model parameters (coefficients) and λ is the regularisation hyperparameter and $R(\omega)$ in case of ridge is $\Sigma\,\omega_i^2$ while for lasso is $\Sigma\,|\omega_i|$.

Now if θ^* is the best model that one ends up getting, which is given as $\theta^* = argmin[E(\theta) + \lambda R(\theta)]$, then the lasso regression results in a sparse solution (i.e many of the model coefficients automatically become exactly 0 or $\omega_i = 0$). The sparsity increases with the increase in λ, where sparsity of a model is defined by the number of parameters in θ^* that are exactly equal to 0.

It is already known that the error term with a regularised term is given as $E(\theta) + \lambda R(\theta)$. Now, if an optimal solution is obtained using λ_1 with θ_1 and another using λ_2 with θ_2, where $\lambda_1 > \lambda_2$, one can conclude,

$$E(\theta_1) + \lambda_1 R(\theta_1) \le E(\theta_2) + \lambda_1 R(\theta_2) \quad as\ \theta_1 is\ the\ optimal\ solution\ for\ \lambda_1$$

and

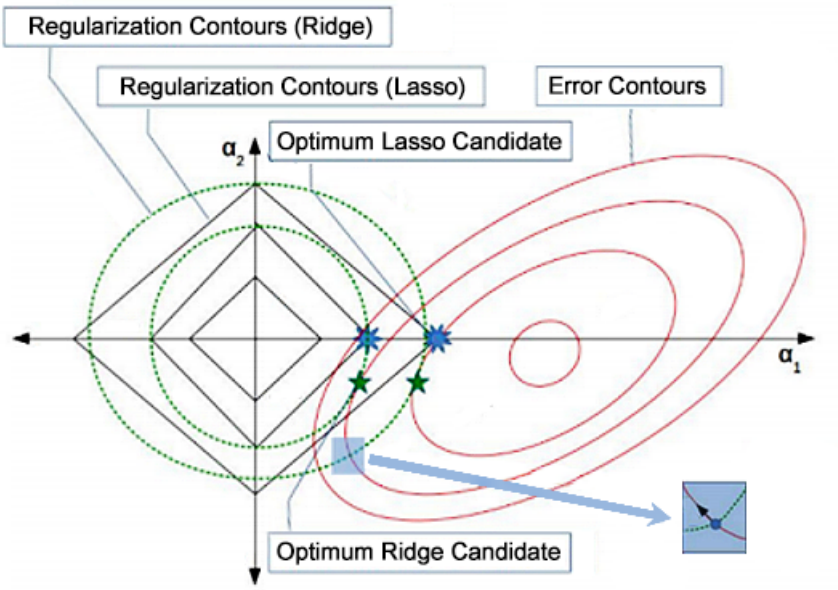
$$E(\theta_2) + \lambda_2 R(\theta_2) \le E(\theta_1) + \lambda_2 R(\theta_1) \quad as\ \theta_2 is\ the\ optimal\ solution\ for\ \lambda_2$$

On solving these two equations it can be concluded $E(\theta_1) \ge E(\theta_2)$ and $R(\theta_1) \le R(\theta_2)$

Thus, one can infer that with the increase in the value of λ, the error term increases and the regularization term decreases and vice-versa. It suggests that higher values of λ does not allow the model to become complex by clamping it down forcefully whereas, lower values of λ ($\rightarrow 0$) allows the model to bring the error down irrespective of the complexity of the model. Briefly, L1 and L2 regularisation, also called Lasso and Ridge respectively, use the L-1 norm ($\Sigma |\omega_i|$) and L-2 norm ($\Sigma \omega_i^2$) respectively as the penalty terms.

Graphical representation of Ridge & Lasso Regularization

The following figure is a schematic illustration of the error (red lines) and regularisation terms (green dotted curves for ridge regression and solid blue lines for lasso regression) contours. The two axes represent α_1 and α_2 respectively.



At every crossing one could move along the arrow (black arrow in the zoomed image) shown to keep the error term same and reduce the regularisation term, giving a better solution. Thus, for the optimum solution for α (sum of the error and regularisation terms is minimum), the corresponding regularisation contour and the error contour must touch each other tangentially and not cross.

The blue stars highlight the touch points between the error contours and the lasso regularisation contours. The green stars highlight the touch points between the error contours and the ridge regularisation terms. The picture illustrates the fact that because of the corners in the lasso contours (unlike ridge regression), the touch points are more likely to be on one or more of the axes. This implies that the other coefficients become zero. Hence, lasso regression also serves as a feature selection method, whereas ridge regression does not.

Python Code - Regularized Regression

```
>> from sklearn.preprocessing import Ridge
>> hyper_params = {'alpha': [parameters]}
>> ridge = Ridge()
>> model_cv = GridSearchCV(estimator=ridge, param_grid=hyper_params, cv=folds,
                           scoring='neg_mean_absolute_error', verbose=1,
                           return_train_score=True)
>> model_cv.fit(X_train, y_train)
```

```
>> cv_results = pd.DataFrame(model_cv.cv_results_)
>> plt.plot(cv_results['param_alpha'], cv_results['mean_train_score'])
>> plt.plot(cv_results['param_alpha'], cv_results['mean_test_score'])
>> ridge = Ridge(alpha=alpha_value_selected)
>> ridge.fit(X_train, y_train)
>> ridge.coef_
```

PRINCIPLES OF MODEL SELECTION

3.3.1. PRINCIPLES OF MODEL SELECTION

There have always been situations where a model performs well on training data but not on the test data. Add to it the confusion about which model to choose for a given problem. Problems like these frequently arise irrespective of the choice of model, data or the problem itself. Some thumb rules and general pointers on selecting the appropriate models are discussed as follows.

Occam's Razor

It states when in dilemma choose the simpler model. A simpler model can be any of these.

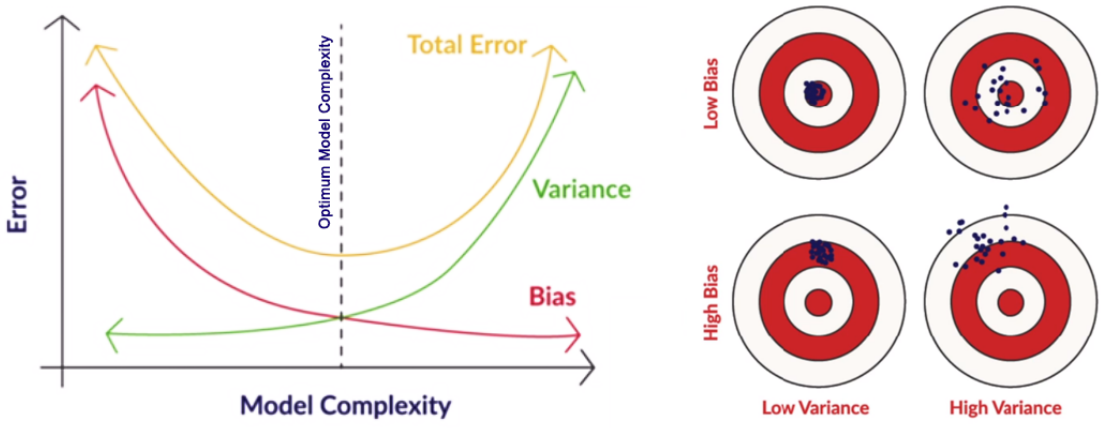
1. Number of parameters required to specify the model completely. The model $y = ax_1 + bx_2$ is simpler than the model $y = ax_1 + bx_2 + cx_3$.
2. The degree of the function, if it is a polynomial.
3. Size of the best-possible representation of the model. The expression $0.552984567x + 932.4710001276$ could be considered to be more complex than $2x + 10$.
4. The depth or size of a decision tree.

Advantages of simpler models.

1. Simpler models are usually more generic and are more widely applicable. One who understands a few basic principles of a subject (simple model) well, is better equipped to solve any new unfamiliar problem than someone who has memorized an entire set of specific problems (complex model).
2. Simpler models require fewer training samples for effective training than the more complex ones and are consequently easier to train.
3. Simpler models are more robust as these are not as sensitive to the specifics of the training data set in comparison to their more complex counterparts.
4. Simpler models make more errors in the training set but give better results on test samples. Whereas complex models lead to overfitting and work very well for the training samples, but fail miserably when applied to test samples.

Bias-Variance Tradeoff

The following figure represents the Bias-Variance tradeoff.



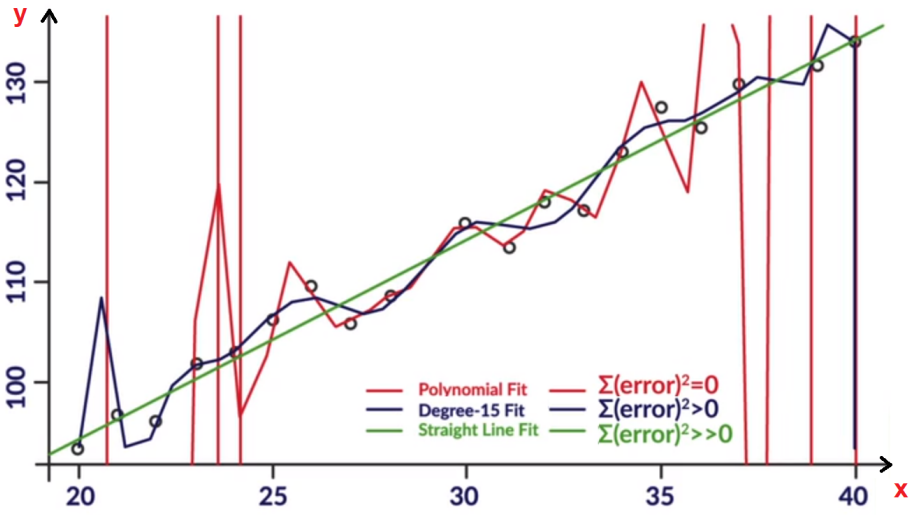
Bias : It quantifies how accurate is the model likely to be on future (test) data. Complex models, assuming there is enough training data available, can do a very accurate job of prediction. Models that are too naive, are very likely to do badly.

Variance : It refers to the degree of changes in the model itself with respect to changes in the training data or how sensitive is a model to the changes in the training data.

As the complexity increases, bias reduces and variance increases. The aim is to find the optimal model.

Overfitting

A model memorizing the data rather than intelligently learning the underlying trends in it is called overfitting. It happens because it is possible for the model to memorize data, which is a problem as the real test happens on unseen real world data. In the following figure the higher degree polynomial fit (red line) shows an overfit model.

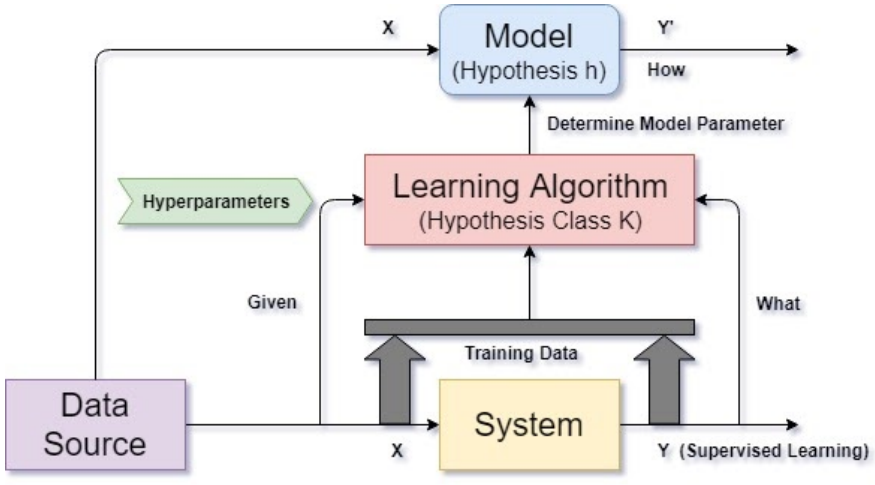


Regularization

It is the process of deliberately simplifying models to achieve the correct balance between keeping the model simple and yet not too naive.

Machine Learning Scenario

The following figure shows a typical Machine Learning scenario.



There is a data source and an underlying system that needs to be replicated. For example the data source could be an email repository. The system is a human, who is trying to segregate spam emails from the rest. There is a Learning Algorithm that takes samples from the data source (this will constitute the training data) and the expected responses from the system and comes up with a model that behaves a lot like the system being replicated. The inputs are denoted by the vector x, the expected system output as y and model output as y'. The learning algorithm works with a hypothesis class K (each learning algorithm searches for the best possible model from a fixed class of models specific to the algorithm). The output of the model is a specific hypothesis or model. Hyperparameters govern the way the learning algorithm produces the model.

3.3.2. MODEL EVALUATION

Once the class of models to be used has been decided, one needs to evaluate it. Following are some of the evaluation strategies for situations when there is abundant or limited (or little) training data.

Hyperparameters

These are the parameters that the algorithm designer passes to the learning algorithm in order to control the complexity of the final model. This in term fine tunes the behaviour of the learning algorithm and has a lot of bearing on the final model produced. In short it governs the behaviour of the algorithm.

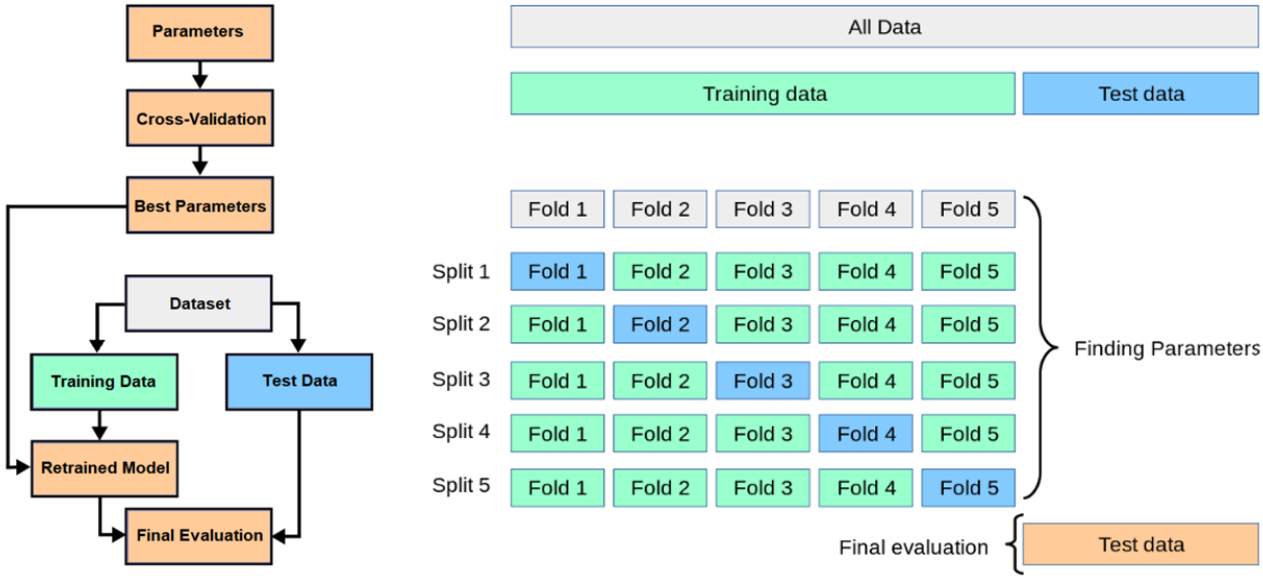
Data Classification

The data is divided into three parts as follows,

1. Training data : data on which model trains.
2. Validation data : data on which model is evaluated for various hyperparameters (prevents peeking of test data).
3. Test data : data on which model predicts.

K-Fold Cross Validation

When the data is limited, the CV is used. In the basic approach, called k-fold CV, the training set is split into k smaller sets. The following figure represents the k-fold CV process.



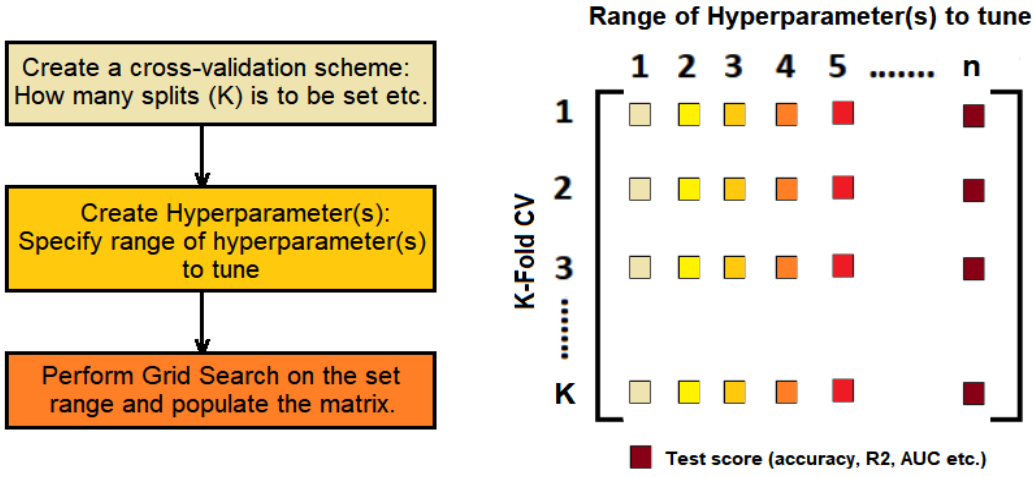
The following steps are taken for each of the k folds.

1. A model is trained using $k-1$ of the folds as training data.
2. The resulting model is validated on the remaining part of the data.

The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop. The model with the best performance is then selected.

Grid Search Cross-Validation

Hyperparameter tuning is one of the most essential tasks in the model building process. For a model to perform best on the data, one needs to tune the hyperparameter(s). For example in the case of linear regression, the number of features is a hyperparameter that can be optimised. One can use the Grid Search Cross Validation to tune the hyperparameter(s) in python. The grid is a matrix which is populated on each iteration. The model is trained using a different value of the hyperparameter each time. The estimated score is then populated in the grid shown in the following figure.



Leave One Out (LOO) Cross Validation

This strategy takes each data point as the test sample once and trains the model on the rest $n-1$ data points. Thus, it trains on n total models. It utilizes the data well since each model is trained on $n-1$ samples, but it is computationally expensive.

Leave P-Out (LPO) Cross Validation

This strategy creates all possible splits after leaving P samples out. For n data points there are nCp possible train test splits.

Stratified K-Fold Cross Validation

This strategy is used for classification problems. It ensures that the relative class proportion is approximately preserved in each train and validation fold. It is very important when there is huge class imbalance (98% to 2%).

Python Code - Cross Validation

Multiple Polynomial Features

```
>> from sklearn.preprocessing import PolynomialFeatures
>> from sklearn.linear_model import LinearRegression
```

```
>> from sklearn.pipeline import make_pipeline
>> model = make_pipeline(PolynomialFeatures(degree),LinearRegression())
# Creates feature and then feeds to the model
>> model.fit(X_train, y_train)
```

K-Fold CV

```
>> from sklearn.model_selection import KFold
>> from sklearn.model_selection import cross_val_score
>> lm = LinearRegression()
>> folds = KFold(n_splits=5, shuffle=True, random_state=100)
>> scores = cross_val_score(lm,X_train,y_train,cv=folds,scoring='r2')
>> scores = cross_val_score(lm, X_train, y_train, cv=folds,
                            scoring='mean_squared_error')
```

Grid Search CV

```
>> hyper_params = [{'n_features_to_select': [parameters]}]
>> lm.fit(X_train, y_train)
>> rfe = RFE(lm)
>> from sklearn.model_selection import GridSearchCV
>> model_cv = GridSearchCV(estimator=rfe, param_grid=hyper_params, cv=folds,
                           scoring='r2', verbose=1, return_train_score=True)
>> model_cv.fit(X_train, y_train)
>> scores = pd.DataFrame(model_cv.cv_results_)
```

Bootstrapping Method

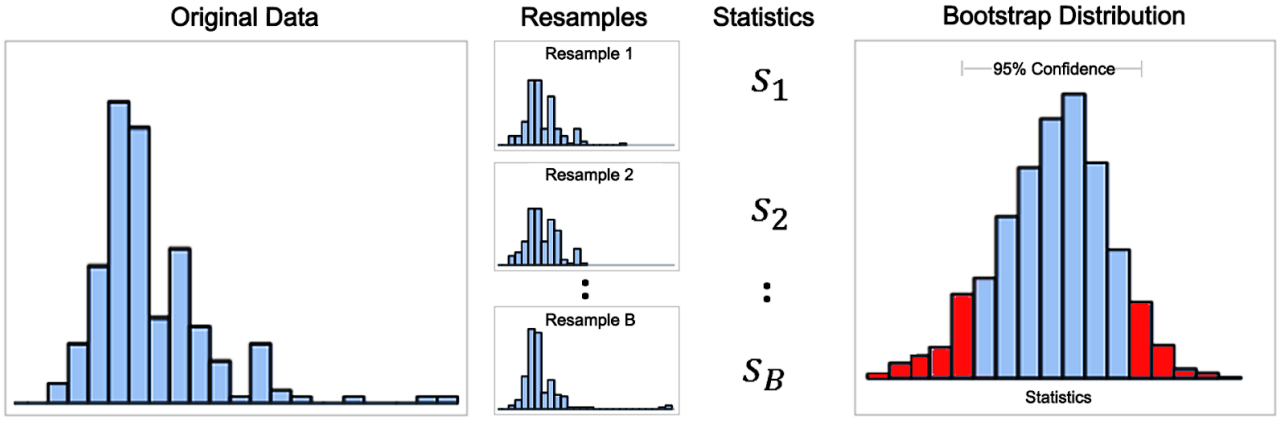
It is very important to both present the expected skill of a machine learning model as well as the confidence intervals for that model skill. Confidence intervals provide a range of model skills and a likelihood that the model skill will fall between the ranges when making predictions on new data. A robust way to calculate confidence intervals for machine learning algorithms is to use the bootstrap method. Bootstrap refers to random sampling with replacement. This is a general technique for estimating statistics mean and standard deviation from the dataset (such as R-squared, Adjusted R2, AIC, BIC, etc.) that can be used to calculate empirical confidence intervals, regardless of the distribution of skill scores. Calculation of confidence intervals with the bootstrap involves the following steps.

1. Calculation of Population of Statistics :
   The first step is to use the bootstrap procedure to resample the original data a number of times and calculate the statistic of interest. The dataset is sampled with replacement (i.e. each time an item is selected from the original dataset, it is not removed, allowing that item to possibly be selected again for the sample). The statistic is calculated on the sample and is stored so as to build up a population of the statistic of interest. The number of bootstrap repeats defines the variance of the estimate, the more the better.

2. Calculation of Confidence Interval :
   The second step is to calculate the confidence intervals. This is done by first ordering the population of statistics and then selecting the values at the chosen percentile for the confidence interval. For example, if a confidence interval of 95% is chosen, then for the calculated $1,000$ statistics from an ordered $1,000$ bootstrap samples, the lower bound would be the $25th$ value and the upper bound would be the $975th$ value. Here a non-parametric confidence interval is calculated which does not make any assumption about the functional form of the distribution of the statistic. This confidence interval is often called the empirical confidence interval.

3. Bootstrap Model Performance :
   The bootstrap method can also be used to evaluate the performance of machine learning algorithms without the need for any test/validation sample. Usually, the size of the sample (training sample) taken in each iteration may be limited to 60% or 80% of the available data. This means that there will always be some observations in the dataset that are not included in the training sample. These are called the out of bag (OOB) samples. A model can then be trained on the data sample of each bootstrap iteration and evaluated on the out of bag samples of that iteration to give a performance statistic. These performance statistics can be collected and the confidence intervals be calculated.

The following figure shows a graphical representation of the above mentioned steps.



Python Code - Bootstrapping

Configure Bootstrap

```
>> n_iterations = no_of_iterations
>> n_size = int(len(df) * percentage_of_sample)
```

Run Bootstrap

```
>> from sklearn.utils import resample
>> from sklearn.metrics import accuracy_score
>> stats = list()
>> values = df.values
>> for i in range(n_iterations):
>>     train = resample(values, n_samples=n_size)
>>     X_train = train[:,:-1]
>>     y_train = train[:,-1]
>>     test = numpy.array([x for x in values if x.tolist() not in train.tolist()])
>>     X_test = test[:,:-1]
>>     y_test = test[:,-1]
>>     model = any_machine_learning_model()
>>     model.fit(X_train, y_train)
>>     predictions = model.predict(X_test)
>>     score = accuracy_score(y_test, predictions)
>>     stats.append(score)
```

Plot Stats

```
>> from matplotlib import pyplot
>> pyplot.hist(stats)
>> pyplot.show()
```

Find Confidence Intervals

```
>> alpha = 0.95
>> p = ((1.0-alpha)/2.0) * 100
>> lower = max(0.0, numpy.percentile(stats, p))
>> p = (alpha+((1.0-alpha)/2.0)) * 100
>> upper = min(1.0, numpy.percentile(stats, p))
>> print('confidence interval', alpha*100, lower*100, upper*100))
```

4. CLASSIFICATION

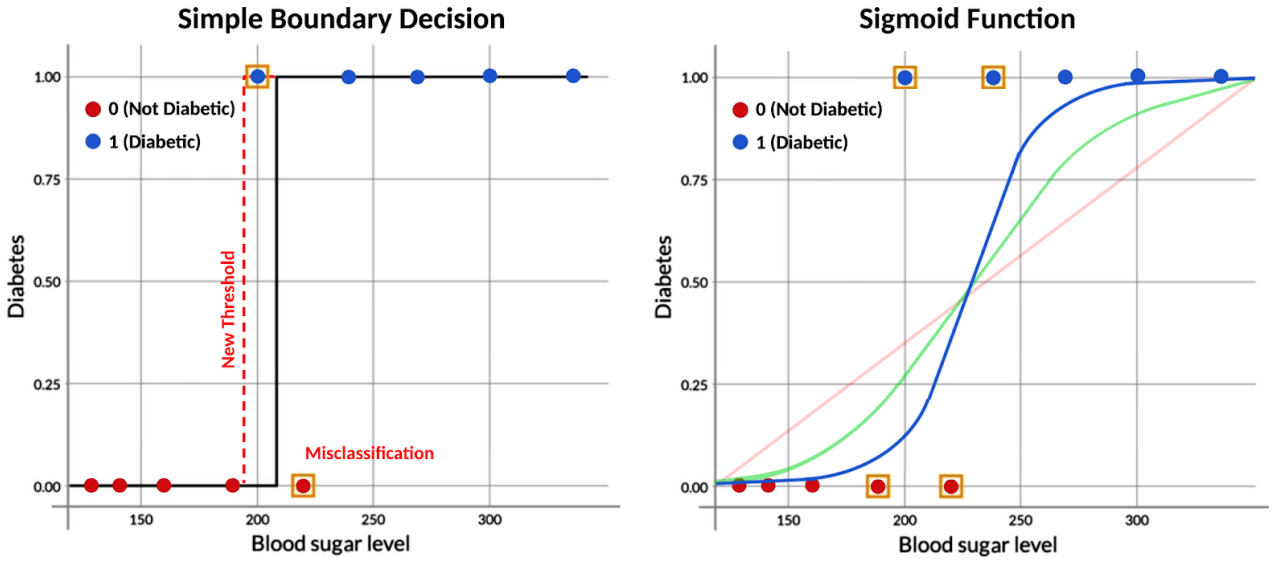
LOGISTIC REGRESSION

4.1.1. LOGISTIC REGRESSION

Linear regression models can make predictions for continuous variables. But if the output variable is categorical, then one has to move to classification models. Logistic Regression is a supervised classification model, which can make predictions from labelled data (i.e. if the target variable is categorical).

Binary Logistic Regression

It is a classification problem in which the target variable has only two possible values, or in other words, two classes. Some examples of binary classification being, determining whether a particular customer will default on a loan or not, whether an email is spam or not, etc. Now let's consider the diabetes example i.e predicting whether a person has diabetes or not, based on that person's blood sugar level. As can be seen from the following figure, a simple boundary decision approach is able to differentiate most of the data points in their respective classes. But, it is going to be too risky to decide the class blatantly on the basis of a cutoff, as especially in the middle, the person could basically belong to any class, diabetic or non-diabetic. So, with the change of the threshold values the number of misclassifications also keeps changing. Thus, with this kind of solution there is a higher chance of misclassification.



Sigmoid Curve

As a simple boundary decision method is not going to work for the diabetic example, one needs to come up with some other method. A curve having extremely low values in the start, extremely high values in the end and intermediate values in the middle would be a good choice. The sigmoid curve has got all these required properties, but so does also a straight line. The main reason for not using the straight line is that it is not steep enough. As can be easily seen from the figure above, the sigmoid curve has low values for a lot of points, then the values rise all of a sudden and at last there are a lot of high values. Whereas, for a straight line, the values rise from low to high very uniformly, and hence, the boundary region (where the probabilities transition from high to low) is not present.

Best Fit Line

The best fit line is a sigmoid curve that is the best approximation dividing the two classes. The equation for the best fit line is given by the sigmoid curve or the logit equation,

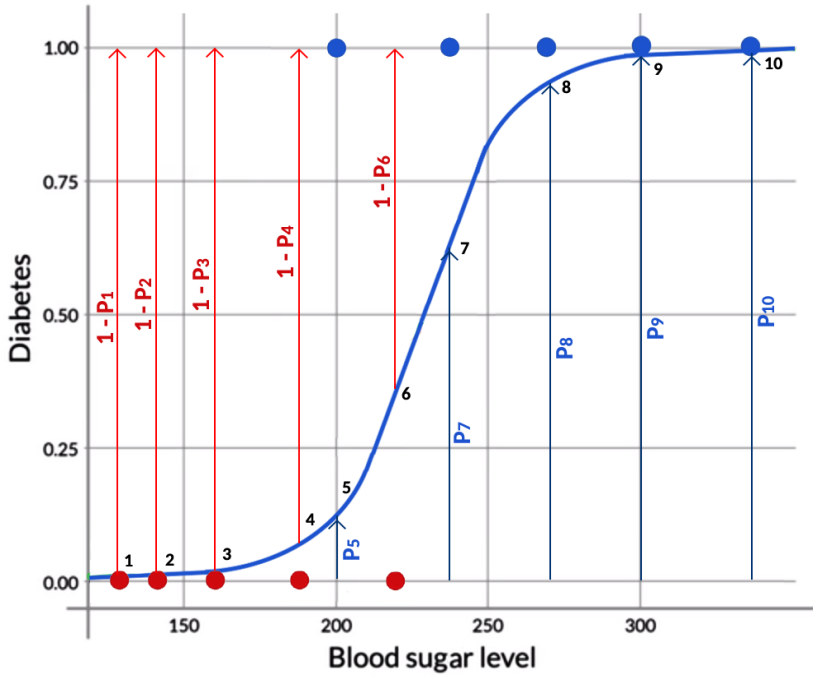
$$P = 1 \Big/ 1 + e^{-\left(\beta_0 + \beta_1 x\right)}$$

However, it is not the only equation that has this form, there is also the probit form of logistic regression, given by,

$$P = \phi^{-1}\left(\beta_0 + \beta_1 x\right)$$

Also, there is the cloglog form of logistic regression, given by,

$$P = \log\left(-\log\left(1-\left(\beta_0 + \beta_1 x\right)\right)\right)$$



For the diabetes example, the best fitting combination of β_0 and β_1 will be the one which will have lower probabilities for non-diabetic persons and higher probabilities for diabetic persons. This can be done by maximizing the product of all the probabilities, i.e.

$$Max\ \left[\left(1-P_1\right)\left(1-P_2\right)\left(1-P_3\right)\left(1-P_4\right)\left(1-P_6\right)\right]\cdot\left[\left(P_5\right)\left(P_7\right)\left(P_8\right)\left(P_9\right)\left(P_{10}\right)\right]$$

This product of probabilities is called the likelihood function. The cost function can be derived using the likelihood function.

Likelihood Function

Consider a dataset $(x_1, x_2, \ldots . x_n)$ with probability density function $f(x_1, \theta)$, then the likelihood function is given by the joint density of the dataset which by independence is equal to the product of the marginal densities,

$$L\left(\theta \mid x_1, x_2, \ldots x_n\right) = f\left(x_1, x_2, \ldots x_n \mid \theta\right) = \prod_{i=1}^{n} f\left(x_i, \theta\right)$$

Cost Function

For a discrete function,

$$f(x_i, \theta) = \theta^{x_i}(1-\theta)^{1-x_i} \quad where, \quad x_i = 0, 1$$

The cost function is given by,

$$J(x, \theta) = L(\theta \mid x) = \prod_{i=1}^{n} \theta^{x_i}(1-\theta)^{1-x_i} = \theta^{\sum_{i=1}^{n} x_i}(1-\theta)^{n-\sum_{i=1}^{n} x_i}$$

$$or, \quad \log J(x, \theta) = \left(\sum_{i=1}^{n} x_i\right)\log\theta + \left(n - \sum_{i=1}^{n} x_i\right)\log(1-\theta)$$

For a continuous function,

$$f(x_i, \theta) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\left(\frac{(x_i-\mu)^2}{2\sigma^2}\right)} \quad where, \quad \theta = f(\mu, \sigma^2)$$

The cost function is given by,

$$J(x, \theta) = L(\theta \mid x) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\left(\frac{(x_i-\mu)^2}{2\sigma^2}\right)} = \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n e^{-\left(\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2\right)}$$

$$or, \quad \log J(x, \theta) = n\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2$$

Optimizing Cost Function

The regression coefficients are usually estimated using maximum likelihood estimation. The maximum likelihood estimator, denoted by θ_{mle} is the value of θ that maximizes the likelihood function $L(\theta|x)$. Unlike linear regression with normally distributed residuals, it is not possible to find a closed-form expression for the coefficient values that maximize the likelihood function, so an iterative process must be used instead. It is often quite difficult to directly maximize $L(\theta|x)$. But, it is usually much easier to maximize the log-likelihood function $log\, L(\theta|x)$. Since log is a monotonic function, the value of the θ that maximizes $log\, L(\theta|x)$, also maximizes $L(\theta|x)$. Thus, one can also define θ_{mle} as the value of θ that solves,

$$\frac{max}{\theta} \log L(\theta|x) = \log \prod_{i=1}^{n} f(x_i, \theta) = \sum_{i=1}^{n} \log f(x_i, \theta)$$

For a discrete function,

$$\theta_{mle} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

For a continuous function,

$$\mu_{mle} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad and \quad \sigma^2_{mle} = \frac{1}{n}\sum_{i=1}^{n} \left(x_i - \mu_{mle}\right)^2$$

Odds and Log Odds

This best fit line function gives the relationship between P (the probability of target variable) and x (the independent variables). But, this form of the function is not very intuitive i.e the relationship between P and x is so complex that it becomes difficult to understand what kind of trend exists between the two. However, on converting the function to a slightly different form, one can achieve a much more intuitive relationship. Upon taking the natural logarithm on both sides of the equation one gets,

$$\log\left(\frac{P}{1-P}\right) = \beta_0 + \beta_1 x \quad where, \ \frac{P}{1-P} = Odds \quad and \quad \log\left(\frac{P}{1-P}\right) = Log\ Odds$$

Odds are defined as the ratio of probability of success to the probability of failure. So, if the odds of success is $4\ (0.8/0.2)$, it shows that the odds of success (80%) has an accompanying odds of failure (20%). Whereas , Log odds is the logarithm of the odds i.e. $Log(4) = 1.386$.

Multivariate Logistic Regression

Multivariate Logistic regression is just an extension of the Univariate Logistic regression. So, the form is identical to Univariate Logistic regression, but with more than one covariate. The logit equation is given by,

$$P = 1 \Big/ 1 + e^{-\left(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \ldots\ldots + \beta_n x_n\right)}$$

Strength of Logistic Regression Model

The strength of any logistic regression model can be assessed using various metrics. These metrics usually provide a measure of how well the observed outcomes are being replicated by the model, based on the proportion of total variation of outcomes explained by the model. The various metrics used to measure a model are,

1. Confusion Matrix
2. ROC Curve
3. Gain and Lift Chart
4. KS - Statistic
5. Gini Coefficient

Confusion Matrix

A confusion matrix is an $n \times n$ matrix, where n is the number of classes being predicted. The following figure represents a confusion matrix for two classes.

Confusion Matrix		Target		*Precision* *TP/(TP+FP)*
		Positive	**Negative**	
Model	**Positive**	True Positive	False Positive	*Positive Predictive Value* *TP/(TP+FP)*
	Negative	False Negative	True Negative	*Negative Predictive Value* *TN/(FN+TN)*
Recall *TP/(TP+FN)*		*Sensitivity* *TP/(TP+FN)*	*Specificity* *TN/(FP+TN)*	**Accuracy** **(TP+TN)/(TP+FP+FN+TN)**

Accuracy of the model is the proportion of the total number of predictions that were correct.

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

For a model with an accuracy of about 90% (which looks good), on revaluation of the confusion matrix, one could see that there were still a lot of misclassifications present. Thus, other new discriminative metrics were brought in.

1. Positive Predictive Value / Precision : Proportion of positive cases correctly identified.
   $$Positive\ Predictive\ Value\ or\ Precision = TP/(TP + FP)$$

2. Negative Predictive Value : Proportion of negative cases correctly identified.
   $$Negative\ Predictive\ Value = TN/(TN + FN)$$

3. Sensitivity / Recall : Proportion of actual positive cases correctly identified.
   $$Sensitivity\ or\ Recall = TP/(TP + FN)$$
4. Specificity : Proportion of actual negative cases correctly identified.
   $$Specificity = TN/(TN + FP)$$

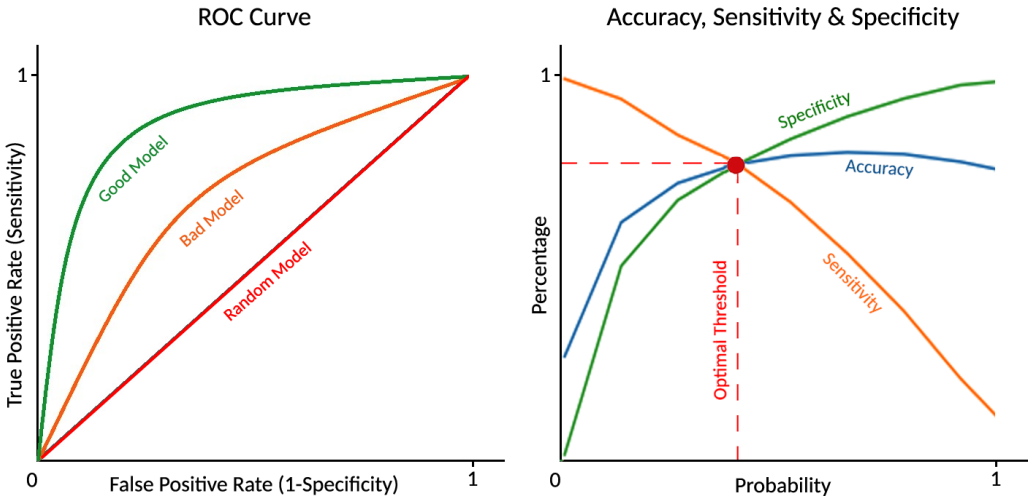
5. True Positive Rate : Proportion of actual positive cases correctly identified.
   $$True\ Positive\ Rate = TP/(TP + FN)$$

6. False Positive Rate : Proportion of actual negative cases incorrectly identified.
   $$Specificity = FP/(FP + TN)$$

7. F1 Score : It is the harmonic mean of Precision and Recall.
   $$F1\ Score = 2 \times (Precision \times Recall)/(Precision + Recall)$$

ROC Curve

The ROC or Receiver Operating Characteristics curve is the plot between True Positive Rate and the False Positive Rate, or simply, a tradeoff between sensitivity and specificity. The biggest advantage of using the ROC curve is that it is independent of the change in proportion of responders. This is because it has both the axes derived out from the columnar calculations of confusion matrix, where the numerator and denominator of both x and y axis change on a similar scale for any change in the response rate. The following figure shows the more the ROC curve is towards the upper-left corner (i.e the more is the area under the curve), the better is the model.



As can be seen, from the accuracy, sensitivity and specificity tradeoff, that when the probability thresholds are very low, the sensitivity is very high and the specificity is very low. Similarly, for larger probability thresholds, the sensitivity values are very low but the specificity values are very high. One could choose any cut-off point based on which of these metrics is required to be high (like if one wants to capture the positives better then some accuracy could be let off for the sake of higher sensitivity and a lower cut-off be chosen). It is completely dependent on the situation. But the optimal cut-off point (where accuracy, sensitivity and specificity meet) can give a fair idea of how the thresholds should be chosen.

Gain and Lift Chart

Before going to the Gain and Lift chart one needs to understand the decile. The following steps are used to get the decile chart.

1. Probability of each observation is calculated.
2. The probabilities are ranked in decreasing order.
3. Deciles are built with each group having almost 10% of the observations.
4. The response rates are calculated at each decile.

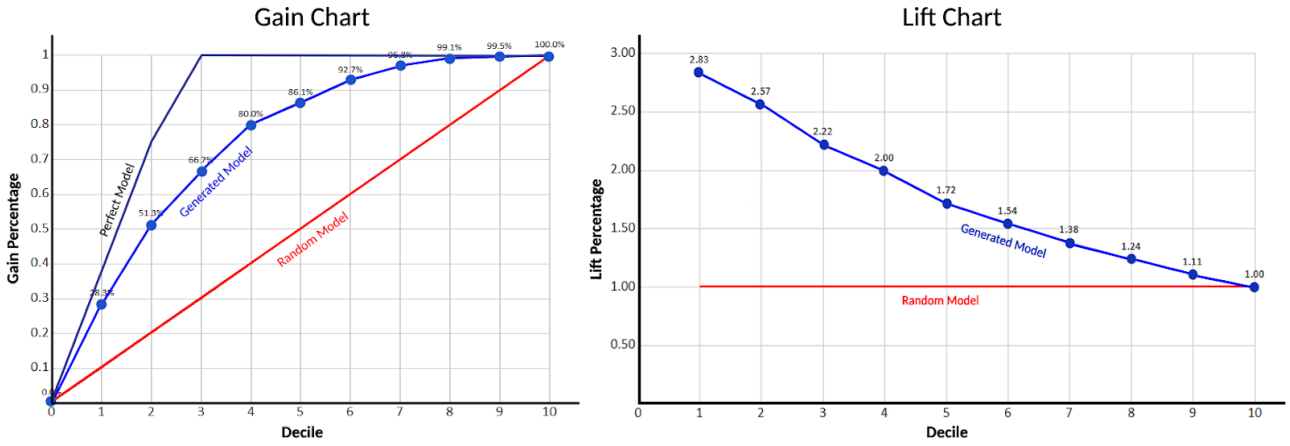
Gain Chart : The gain of a given decile level is the proportion of the cumulative number of targets till that decile to the total number of targets in the dataset.

Lift Chart : The lift of a given decile level is the proportion of gain percentage to the random expectation percentage at that decile level or simply put it measures how much better one can expect to do with the model compared to a random model.

The following figures give a representation of the Gain and Lift charts.

Decile	Data Points	Class 1	Cum Class 1	% Cum Class 1	Class 0	Cum Class 0	%Cum Class 0	Gain Random Model	% Gain Class 1	Lift Class 1	K-S Statistic
1	211	159	159	28.3%	52	52	3.4%	10%	28.3%	2.83	25.0%
2	211	129	288	51.3%	82	134	8.7%	20%	51.3%	2.57	42.7%
3	211	86	374	66.7%	125	259	16.7%	30%	66.7%	2.22	49.9%
4	211	75	449	80.0%	136	395	25.5%	40%	80.0%	2.00	54.5%
5	211	34	483	86.1%	177	572	36.9%	50%	86.1%	1.72	49.2%
6	211	37	520	92.7%	174	746	48.2%	60%	92.7%	1.54	44.5%
7	211	23	543	96.8%	188	934	60.3%	70%	96.8%	1.38	36.5%
8	211	13	556	99.1%	198	1132	73.1%	80%	99.1%	1.24	26.0%

9	211	2	558	99.5%	209	1341	86.6%	90%	99.5%	1.11	12.9%
10	211	3	561	100%	208	1549	100%	100%	100%	1.00	0.0%
	2110	561			1549						



K-S Statistic

K-S or Kolmogorov-Smirnov statistic is an indicator of how well the model discriminates between the two classes i.e a measure of the degree of separation between the positive and negative distributions. It is equal to 0% for the random model, and 100% for the perfect model. Thus, the KS statistic gives an indicator of where the model lies between the two.

A good model is one for which the K-S statistic treads the following conditions,

1) It is equal to 40% or more.
2) It lies in the top deciles, i.e. 1st, 2nd, 3rd or 4th.

Gini Coefficient

The Gini coefficient is nothing but a fancy word and is given by,

$$Gini = AUC = Area\ Under\ ROC\ Curve$$

Python Code - Logistic Regression

Model

```
>> import statsmodels.api as sm
>> links = sm.families.links
>> model = sm.GLM(y_train,(sm.add_constant(X_train)),
                  family=sm.families.Binomial(link=links.logit))
# link can take any of the value logit, probit or cloglog
>> model.fit().summary()

>> from sklearn.linear_model import LogisticRegression
>> model = LogisticRegression(class_weight='balanced')
# For balancing the imbalanced classes
>> model.fit(X_train, y_train)

Predict
>> y_pred = model.predict(X_test)

Analyze Model
>> from sklearn import metrics
```

```
>> metrics.accuracy_score(y_test, y_pred)
# Accuracy
>> from sklearn.metrics import confusion_matrix
>> confusion_matrix = confusion_matrix(y_test, y_pred)
# Confusion Matrix
>> from sklearn.metrics import classification_report
>> classification_report(y_test, y_pred)
# Precision, Recall and F1-score
>> from sklearn.metrics import roc_auc_score
>> from sklearn.metrics import roc_curve
>> logit_roc_auc = roc_auc_score(y_test, y_pred)
>> fpr, tpr, thresholds = roc_curve(y_test, y_pred[:,1])
# ROC-AUC Curve
>> plt.plot(fpr,tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
```

4.1.2. CHALLENGES IN LOGISTIC REGRESSION

Logistic regression is a widely used technique in various industries because of two main reasons,

1. It is very easy to understand and offers an intuitive explanation of the variables.

2. The output (i.e. the probabilities) has a linear relationship with the log of odds, which can be very useful for explaining results to stakeholders. Usually scores are used instead of log odds. Scores are just another way of reporting the findings in a more elegant way than the log odds by using a user defined function to scale the log odds into a very presentable manner. For example, one can use the user defined function $Score = 500 + (20 \times log(odds)/log(2))$ to find the scores from log odds.

But there are certain nuances that need to be taken care of before proceeding for a Logistic regression model for getting better results. Following are the various nuances discussed in detail.

Sample Selection

Selecting the right sample is very essential for solving any business problem. One should look out for the following mentioned errors while selecting a sample.

1. Seasonal or cyclic fluctuations population : The seasonal fluctuations in the business (such as new year sales, economic ups and downs etc.) needs to be taken care of while building the samples. A wide range of data should be selected representing each type of seasonal fluctuations.

2. Representative population : The sample should be representative of the population on which the model is going to be applied in the future rather than all the types of population.

3. Rare incidence population : For rare events samples (such as fraud transactions, anomaly detection etc.), the samples should be stratified and balanced before being used for modelling.

Segmentation

Usually even after having selected the samples with all due considerations, there are chances that the model may not perform well on the chosen sample data set. Assuming that no new sample is

available, the performance of the model can still be improved upon by performing the segmentation of the population.

It is used when there are different segments in the dataset. The notion being that the behaviour of different segments of the population depend on different variables (such as a student defaulting on loan depends on factors like course enrolled for, institute enrolled in, parents' income etc. while a salaried person defaulting on loan depends on factors like marital status, income, etc.). As the predictive patterns across these segments are very different, it makes more sense to build different child models for each of these segments rather than building one single parent model. Now, a combined model of these child models gives a better result than a single model. It should be noted that segmentation should only be done on a variable which is more likely to provide different sets of predictors for the model.

Variable Transformation

It is already known that the categorical variables need to be transformed into dummies and numeric variables be standardised before proceeding with any modelling. However, one could also convert the numeric variables into dummy variables. The major advantage offered by dummies especially for continuous variables is that they stabilize the model i.e small variations in the variables will not have a very big impact on a model built using dummies whereas it will still have a sizable impact on a model built using continuous variables as is. But, it also has a certain disadvantage of the data getting compressed into very few categories resulting in information loss and data clumping.

Apart from creating dummies, there are also other techniques for transforming variables. The commonly used techniques used for transforming variables are,

1. WOE (Weight of Evidence) Transformation
2. Interaction Variables
3. Splines
4. Mathematical Transformation
5. PCA (Principal Component Analysis)

WOE (Weight of Evidence) Transformation

WOE can be calculated for both the categorical as well as continuous data. For continuous data the WOE is usually calculated after having binned the continuous data.

$$WOE = log\,(Percentage\ of\ Events\ in\ bucket) - log\,(Percentage\ of\ Non\ Events\ in\ bucket)$$

The following figure gives a representation of the WOE and IV calculated.

Bins	Number of Events	Number of Non Events	WOE	Information Value
0-5	627	744	-1.19	0.33
6-19	947	489	-0.36	0.03
20-39	1095	320	0.21	0.01
40-59	1121	217	0.62	0.06
60-72	1384	99	1.62	0.35
Total	5174	1869		0.78

Once the WOE values have been calculated, it is important to note that the WOE values should follow an increasing or decreasing trend across the bins. If the trend is not monotonic, then the buckets/bins (coarse buckets) need to be compressed and the WOE values calculated again. It can

be seen that WOE does something similar to creating dummy variables (i.e. replacing a range of values with an indicative variable). It is just that, instead of replacing it with a simple 0 or 1 (which is not thought out at all), it is getting replaced with a well thought out WOE value (which reduces the chances of undesired score clumping).

The advantages of using WOE are,

1. WOE reflects the group identity, i.e. it captures the general trend of distribution of both the events and non-events.

2. NA values can be treated with WOE values. However, one can replace the NA bucket with a bucket which shows similar WOE values. Thus, WOE helps in treating missing values logically for both categorical and continuous variables.

3. WOE makes the model more stable as small changes in the continuous variables will not impact the input so much.

The only disadvantage of WOE is that one might end up doing some score clumping.

Rank Ordering : It is a simple line graph of percentage of events against deciles (scoring bins). On calculating the percentage of events in each decile group, one can easily see the event rate monotonically decreasing. It means that the model predicts the highest number of events in the first decile and then goes progressively down.

IV (Information Value) : It is an important indicator of predictive power. It mainly helps in understanding how the binning of variables should be done. The binning should be done such that the WOE trend across bins is monotonic and the IV (information value) should be high. IV is given by,

$$IV = WOE \times (Percentage\ of\ Events\ in\ bucket - Percentage\ of\ Non\ Events\ in\ bucket)$$

Interaction Variables

Combining various variables to create one variable which gives a good meaning is called an interaction variable. The interaction variables can be built on a judgement call based upon a deep understanding of the business or by building a small decision tree. Indeed, it is only because of incorporation of these interaction variables that Random Forests and Neural networks are able to outperform the Logistic Regression.

Splines

A spline is basically obtained by fitting a polynomial over the WOE values. Using a spline offers high predictive power, but it may result in unstable models.

Mathematical Transformation

Mathematical transformations such as x^2, x^3 and $log\ x$ are commonly used under this type of transformation. But these transformations are not very easy to explain or very intuitive.

PCA (Principal Component Analysis)

It is a very important and effective transformation technique. It transforms the variables into components that are orthogonal to each other or to say the variables are grouped and modified and

put into a few bunches such that each bunch is not correlated with the other. But these transformations are also not very easy to explain or very intuitive.

4.1.3. IMPLEMENTATION OF LOGISTIC REGRESSION

Once the data is prepared for modelling by performing tasks such as sample selection, segmentation, and variable transformation techniques, then the model is built. Next comes the model evaluation, validation and governance.

Model Evaluation

The model performance measures are grouped under three broad classes.

1. Discriminatory Power : It measures how good the model is at separating out the classes. Various metrics being used for measuring are,

    a. KS - Statistic
    b. Gini Coefficient
    c. Rank Ordering
    d. Sensitivity
    e. Specificity

2. Accuracy : It measures how accurately the model is able to predict the classes. Various metrics being used for measuring are,

    a. Sensitivity
    b. Specificity
    c. Comparing Actual v/s Predicted Log Odds

3. Stability : It measures how stable the model is in predicting unseen data. Various metrics being used for measuring are,

    a. Performance Stability : Results of in-sample validation approximately match those of out-of-time validation.
    b. Variable Stability : The sample used for model building has not changed too much and has the same general characteristics. PSI (Population Stability Index) is used for the same.

Model Validation

The model can be validated using the following methods,

1. In Sample validation
2. Out of Time validation
3. K-Cross validation

Model Governance

The process of tracking the model over time as it is being used is referred to as Model Governance. The tracking is done by evaluating the model on the basis of the ongoing samples and comparing it with the previous such evaluation result. The following figure represents the metrics of a model performance over time.

Quarter	Performance Metric (Gini)	Action Taken
2014 Q4	0.84	
2015 Q1	0.82	
2015 Q2	0.80	
2015 Q3	0.78	
2015 Q4	0.75	
2016 Q1	0.72	Recalibration
2016 Q2	0.80	
2016 Q3	0.76	
2016 Q4	0.71	Rebuilding
2017 Q1	0.83	

As can be seen, when the model's performance dropped to 0.72 for the first time, building of a new model was avoided. Basically, the model was just recalibrated (i.e. only the coefficients of the variables were updated), which resulted in a slight increase in the performance. However, the next time the performance dropped to a low value of 0.71, the model was rebuilt (i.e. an entire new model was built with new sample data).

NAIVE BAYES

4.2.1. BAYES' THEOREM

Naive Bayes is a probabilistic classifier which returns the probability of a test point belonging to a class rather than the label of the test point.

Building Blocks of Bayes Theorem - Probability

The probability of an event is a measure of the chance that the event will occur as a result of an experiment i.e. the number of ways the event can occur divided by the total number of possible outcomes. The probability of an event E is given by,

$$P(E) = Number\ of\ favourable\ outcomes \,/\, Total\ number\ of\ possible\ outcomes$$

For example, the probability that a card is a Four is,

$$P(Four) = 4/52 = 1/13$$

Building Blocks of Bayes Theorem - Joint Probability

Joint probability is a statistical measure that calculates the likelihood of two events occurring together and at the same point in time. The joint probability of two events A and B is given by,

$$P(A,B) = P(A \cap B) = P(B \cap A) = P(A) \times P(B)$$

For example, the probability that a card is a Four and Red is,

$$P(Four \cap Red) = P(Four) \times P(Red) = (4/52) \times (26/52) = 1/26$$

Building Blocks of Bayes Theorem - Conditional Probability

Conditional probability is a measure of the probability of an event occurring given that another event has occurred. The probability of event A given that event B has already occurred is given by,

$$P(A \mid B) = P(A \cap B) \,/\, P(B)$$

For example, the probability that a card is a Four given that the card is Red is,

$$P(Four \mid Red) = P(Four \cap Red) \,/\, P(Red) = P(Four) \times P(Red) \,/\, P(Red) = (4/52) = 1/13$$

Bayes Theorem

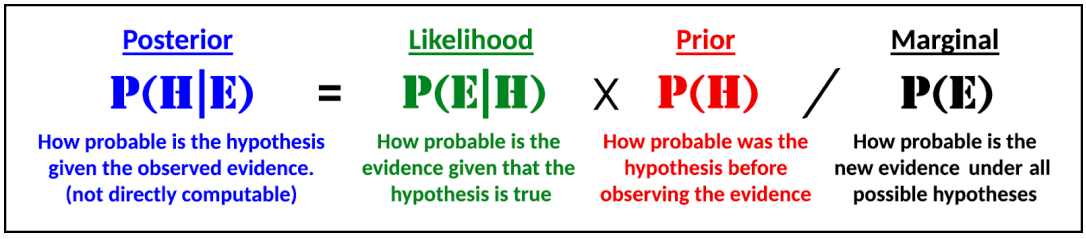
Bayes Theorem gives the probability of an event, based on prior knowledge of conditions that might be related to the event. For example, if diabetes is related to age, then, using Bayes' theorem, a person's age can be used to more accurately assess the probability that they have diabetes, compared to the assessment of the probability of diabetes made without the knowledge of the person's age. If the probability of an event B occurring given that event A has already occurred and individual probabilities of A and B are known, then the probability of event A given B has already occurred can be found out using Bayes Theorem given by,

$$P(A \mid B) = P(B \mid A) \times P(A) \,/\, P(B)$$

or

$$P(B \mid A) = P(A \mid B) \times P(B) \,/\, P(A)$$

The above equation can also be defined in terms prior, posterior and likelihood as follows,



4.2.2. NAIVE BAYES

Naive Bayes 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.

Assumptions for Naive Bayes

The fundamental Naive Bayes assumptions are,

1. Independence : No pair of features are dependent. For example, the colour of apple being red has nothing to do with the apple being round and the apple being round has no effect on the apple being of 3 inches in diameter. Hence, the features are assumed to be independent. i.e. $P(A,B) = P(A) \times P(B)$

2. Equal contribution to the outcome : Each feature is given the same weightage (or importance). For example, knowing only the colour and the size alone cannot predict the outcome accurately. Hence, none of the attributes are irrelevant and assumed to be contributing equally to the outcome.

The assumptions made by Naive Bayes are not generally correct in real-world situations. In-fact, the independence assumption is never correct but often works well in practice.

Naive Bayes Theorem

On applying the above assumptions to the Bayes theorem one gets,

$$P(C_k \mid x_1, x_2, \ldots x_n) = \frac{P(x_1, x_2, \ldots x_n \mid C_k)\, P(C_k)}{P(x_1, x_2, \ldots x_n)} = \frac{P(x_1 \mid C_k)\, P(x_2 \mid C_k) \ldots P(x_n \mid C_k)\, P(C_k)}{P(x_1)\, P(x_1) \ldots P(x_n)}$$

where,

C_k *denotes the* k^{th} *class*

x_n *denotes the the* n^{th} *feature of the data point*

The value of the denominator $P(x_1)P(x_2)...P(x_n)$ remains the same for all the classes for any given data point. Thus, the denominator behaving as a scaling factor out here can be ignored without affecting the final outcome giving,

$$P\left(C_k \mid x_1, x_2, \ldots x_n \right) = P\left(C_k \right) \prod_{i=1}^{n} P\left(x_i \mid C_k \right)$$

Prior Probability : $P(C_k)$ is known as the prior probability. It is the probability of an event occurring before the collection of new data i.e. our prior beliefs before the collection of specific information. Prior plays an important role while classifying using Naive Bayes, as it highly influences the class of the new test point.

Likelihood Function : $P(x_i|C_k)$ represents the likelihood function. It gives the likelihood of a data point occurring in a class i.e. updates our prior beliefs with the new information.

Posterior Probability : $P(C_k|x_1, x_2, ...x_n)$ is called the posterior probability, which is finally compared for the classes and the test point is assigned to the class for which the posterior probability is greater i.e. the final outcome is a balanced combination of the prior beliefs and case-specific information.

Naive Bayes Classifier

In the Naive Bayes theorem only the independent feature model had been derived, i.e. the Naive Bayes probability model. Now, one can easily build the Naive Bayes Classifier by combining this model with a decision rule. One of the most common rules is to pick the hypothesis that is the most probable i.e a class whose posterior is greater than the posterior of the other classes. This is known as the Maximum A Posteriori or MAP decision rule. The Bayes classifier that assigns a class label $y = C_k$ to an observation x is given by,

$$y = \underset{k \in \{1, 2, \ldots K\}}{Argmax} P\left(C_k \right) \prod_{i=1}^{n} P\left(x_i \mid C_k \right)$$

4.2.3. NAIVE BAYES FOR TEXT CLASSIFICATION

Naive Bayes is most commonly used for text classification in applications such as predicting spam emails, classifying text into categories such as politics, sports, lifestyle etc. In general, Naive Bayes has proven to perform well in text classification applications. The following segments explain the Multinomial Naive Bayes Classifier and Bernoulli Naive Bayes Classifier for the categorical data using an example of Naive Bayes document classifier.

Document Classifier Data

1. **Test Data**

Index	Document	Class
0	Harvard is a great educational institution.	education
1	Educational greatness depends on ethics.	education
2	A story of great ethics and educational greatness.	education
3	Titanic is a great cinema.	cinema
4	A good movie depends on a good story.	cinema

The above documents can be converted into features by breaking the sentences into words and putting the unique words into a bag. While selecting the unique words the stopwords

(such as a, an, the, is etc.) are removed as these are inconsequential in terms of providing any discriminatory information helpful in the classification process.

2. **Vocabulary Dictionary**

Dictionary with Stop Words	
0	and
1	cinema
2	depends
3	educational
4	ethics
5	good
6	great
7	greatness
8	institution
9	is
10	movie
11	of
12	on
13	titanic
14	story
15	harvard

Stop Words	
0	and
9	is
11	of
12	on

Final Vocabulary	
0	cinema
1	depends
2	educational
3	ethics
4	good
5	great
6	greatness
7	institution
8	movie
9	titanic
10	story
11	harvard

Now that the vocabulary is ready, next it can be converted into a Bag of Word Representation.

3. **Bag Of Words (BOW) Representation**

Vocabulary		D0	D1	D2	Total	P(W|C)	D3	D4	Total	P(W|C)
0	cinema	0	0	0	0	0/13	1	0	1	1/8
1	depends	0	1	0	1	1/13	0	1	1	1/8
2	educational	1	1	1	3	3/13	0	0	0	0/8
3	ethics	0	1	1	2	2/13	0	0	0	0/8
4	good	0	0	0	0	0/13	0	2	2	2/8
5	great	1	0	1	2	2/13	1	0	1	1/8
6	greatness	0	1	1	2	2/13	0	0	0	0/8
7	institution	1	0	0	1	1/13	0	0	0	0/8
8	movie	0	0	0	0	0/13	0	1	1	1/8
9	titanic	0	0	0	0	0/13	1	0	1	1/8
10	story	0	0	1	1	1/13	0	1	1	1/8
11	harvard	1	0	0	1	1/13	0	0	0	0/8
Total Words Count		4	4	5	13		3	5	8	

Bag of Word Representation basically breaks down the sentences available into words and makes the order of the words irrelevant as if the words were put in a bag and shuffled. All that one is concerned about is the number of occurrences of the words present (for Multinomial Naive Bayes) or if a word is present or not (for Bernoulli Naive Bayes).

Multinomial Naive Bayes Classifier

The Multinomial Naive Bayes Classifier is given by,

$$P\left(C_k \mid w_1, w_2, \ldots w_n \right) \propto P\left(C_k \right) \prod_{i=1}^{n} P\left(w_i \mid C_k \right)$$

Using the above BOW table and the Multinomial Naive Bayes classifier let's classify some documents.

Document : "Great Story"		
Class : education	**Class : cinema**	**Result**
Prior : **P(education) = 3/5** **Likelihood :** **P(Great | education) = 2/13** **P(Story | education) = 1/13** **Posterity :** **Likelihood x Prior = 0.007**	**Prior :** **P(cinema) = 2/5** **Likelihood :** **P(Great | cinema) = 1/8** **P(Story | cinema) = 1/8** **Posterity :** **Likelihood x Prior = 0.006**	**As** **P(education | document) > P(cinema | document)** **Thus, it can be concluded that the document belongs to education**

Document : "Good Story"		
Class : education	**Class : cinema**	**Result**
Prior : **P(education) = 3/5** **Likelihood :** **P(Good | education) = 0/13** **P(Story | education) = 1/13** **Posterity :** **Likelihood x Prior = 0**	**Prior :** **P(cinema) = 2/5** **Likelihood :** **P(Good | cinema) = 2/8** **P(Story | cinema) = 1/8** **Posterity :** **Likelihood x Prior = 0.012**	**As can be seen, one has encountered the zero probability problem. It needs to be solved.**

Multinomial Laplace Smoothing

Now having encountered the zero probability problem i.e. the probability of a word which has never appeared in a class (though it may have appeared in the dataset in another class) is 0, one needs to fix the same. This is where the technique Laplace smoothing comes into play. In this technique the words with frequency 0 in a particular class is assigned to a new frequency value of α along with adding the same value α to the frequency of all the other words in the class. This in turn changes the probability of each of the word in the particular class to,

$$P(W) = (Frequency\ of\ Word + \alpha) / (\alpha \times Size\ of\ Vocabulary)$$

Taking the value of $\alpha = 1$ and applying Laplace Smoothing to it, the new BOW table can be converted as given in the following table,

Vocabulary		D0	D1	D2	Total	P(W|C)	D3	D4	Total	P(W|C)
0	cinema	0	0	0	0+1	1/13	1	0	1+1	2/20
1	depends	0	1	0	1+1	2/25	0	1	1+1	2/20
2	educational	1	1	1	3+1	4/25	0	0	0+1	1/20
3	ethics	0	1	1	2+1	3/25	0	0	0+1	1/20
4	good	0	0	0	0+1	1/25	0	2	2+1	3/20
5	great	1	0	1	2+1	3/25	1	0	1+1	2/20
6	greatness	0	1	1	2+1	3/25	0	0	0+1	1/20
7	institution	1	0	0	1+1	2/25	0	0	0+1	1/20
8	movie	0	0	0	0+1	1/25	0	1	1+1	2/20
9	titanic	0	0	0	0+1	1/25	1	0	1+1	2/20
10	story	0	0	1	1+1	2/25	0	1	1+1	2/20
11	harvard	1	0	0	1+1	2/25	0	0	0+1	1/20
Total Words Count		4	4	5	25		3	5	20	

Now using the new modified BOW table and the Multinomial Naive Bayes classifier let's classify another document.

Document : “Pretty Good Story”		
Class : education	**Class : cinema**	**Result**
Prior : P(education) = 3/5 **Likelihood :** **P(Pretty | education) = Ignore** **P(Good | education) = 1/25** **P(Story | education) = 2/25** **Posterity :** **Likelihood x Prior = 0.001**	**Prior : P(cinema) = 2/5** **Likelihood :** **P(Pretty | cinema) = Ignore** **P(Good | cinema) = 3/20** **P(Story | cinema) = 2/20** **Posterity :** **Likelihood x Prior = 0.006**	**As** **P(cinema | document) > P(education | document)** **Thus, it can be concluded that the document belongs to cinema**

As can be seen if there are words occurring in the test document which are not a part of the vocabulary, then they will not be considered as a part of the feature vector since it only considers the words that are part of the dictionary. Thus, these new words are to be completely ignored.

Bernoulli Naive Bayes Classifier

The Bernoulli Naive Bayes Classifier is given by,

$$P(C_k \mid w_1, w_2, \ldots w_n) \propto P(C_k) \prod_{i=1}^{n} \left[d_i \, P(w_i \mid C_k) + (1 - d_i)(1 - P(w_i \mid C_k)) \right]$$

$$where,$$

$$d_i \; can \; be \; either \; 0 \; or \; 1 \; (i^{th} \; feature \; is \; present \; in \; the \; document \; or \; not)$$

The most fundamental difference in Bernoulli Naive Bayes Classifier is that unlike the Multinomial way which is concerned about the number of occurrences of the word in the class, in Bernoulli one is just concerned about whether the word is present or not. The bag of words representation in this case is just a 0 or 1 rather than the frequency of occurrence of the word. The BOW representation for Bernoulli Naive Bayes is as follows,

Vocabulary		D0	D1	D2	Total	P(W|C)	D3	D4	Total	P(W|C)
0	cinema	0	0	0	0	0/3	1	0	1	1/2
1	depends	0	1	0	1	1/3	0	1	1	1/2
2	educational	1	1	1	3	3/3	0	0	0	0/2
3	ethics	0	1	1	2	2/3	0	0	0	0/2
4	good	0	0	0	0	0/3	0	1	1	1/2
5	great	1	0	1	2	2/3	1	0	1	1/2
6	greatness	0	1	1	2	2/3	0	0	0	0/2
7	institution	1	0	0	1	1/3	0	0	0	0/2
8	movie	0	0	0	0	0/3	0	1	1	1/2
9	titanic	0	0	0	0	0/3	1	0	1	1/2
10	story	0	0	1	1	1/3	0	1	1	1/2
11	harvard	1	0	0	1	1/3	0	0	0	0/2

Using the above BOW table and the Bernoulli Naive Bayes classifier let's classify some documents.

Document : "Very good educational institution"							
Vocabulary			Class : education		Class : cinema		Result
			P(W|C)	Likelihood	P(W|C)	Likelihood	
0	cinema	0	0/3	(0x0/3)+(1-0)(1-0/3)	1/2	(0x1/2)+(1-0)(1-1/2)	The word Very is not there in the vocabulary so it has been ignored. As the Likelihood of word good is becoming 0, it gives rise to the zero probability problem.
1	depends	0	1/3	(0x1/3)+(1-0)(1-1/3)	1/2	(0x1/2)+(1-0)(1-1/2)	
2	educational	1	3/3	(1x3/3)+(1-1)(1-3/3)	0/2	(1x0/2)+(1-1)(1-0/2)	
3	ethics	0	2/3	(0x2/3)+(1-0)(1-2/3)	0/2	(0x0/2)+(1-0)(1-0/2)	
4	good	1	0/3	(1x0/3)+(1-1)(1-0/3)	1/2	(1x1/2)+(1-1)(1-1/2)	
5	great	0	2/3	(0x2/3)+(1-0)(1-2/3)	1/2	(0x1/2)+(1-0)(1-1/2)	
6	greatness	0	2/3	(0x2/3)+(1-0)(1-2/3)	0/2	(0x0/2)+(1-0)(1-0/2)	
7	institution	1	1/3	(1x1/3)+(1-1)(1-1/3)	0/2	(1x0/2)+(1-1)(1-0/2)	
8	movie	0	0/3	(0x0/3)+(1-0)(1-0/3)	1/2	(0x1/2)+(1-0)(1-1/2)	
9	titanic	0	0/3	(0x0/3)+(1-0)(1-0/3)	1/2	(0x1/2)+(1-0)(1-1/2)	
10	story	0	1/3	(0x1/3)+(1-0)(1-1/3)	1/2	(0x1/2)+(1-0)(1-1/2)	
11	harvard	0	1/3	(0x1/3)+(1-0)(1-1/3)	0/2	(0x0/2)+(1-0)(1-0/2)	
			Prior : P(education) = 3/5 Posterity : Likelihood x Prior = 0.001		Prior : P(cinema) = 2/5 Posterity : Likelihood x Prior = 0.006		

Bernoulli Laplace Smoothing

Now having encountered the zero probability problem, Laplace smoothing is applied to the BOW representation. The process remains the same as in case of Multinomial Laplace Smoothing, but with a slight change in the formula, which is given as,

$$P(W) = (Count\ of\ Word + \alpha) / (Documents\ in\ Class + 2)$$

Vocabulary		D0	D1	D2	Total	P(W|C)	D3	D4	Total	P(W|C)
0	cinema	0	0	0	0+1	1/5	1	0	1+1	2/4
1	depends	0	1	0	1+1	2/5	0	1	1+1	2/4
2	educational	1	1	1	3+1	4/5	0	0	0+1	1/4
3	ethics	0	1	1	2+1	3/5	0	0	0+1	1/4
4	good	0	0	0	0+1	1/5	0	1	1+1	2/4
5	great	1	0	1	2+1	3/5	1	0	1+1	2/4
6	greatness	0	1	1	2+1	3/5	0	0	0+1	1/4
7	institution	1	0	0	1+1	2/5	0	0	0+1	1/4
8	movie	0	0	0	0+1	1/5	0	1	1+1	2/4
9	titanic	0	0	0	0+1	1/5	1	0	1+1	2/4
10	story	0	0	1	1+1	2/5	0	1	1+1	2/4
11	harvard	1	0	0	1+1	2/5	0	0	0+1	1/4

Now using the new modified BOW table and the Bernoulli Naive Bayes classifier let's classify the same document.

Document : "Very good educational institution"							
Vocabulary			Class : education		Class : cinema		Result
			P(W|C)	Likelihood	P(W|C)	Likelihood	
0	cinema	0	1/5	(0x1/5)+(1-0)(1-1/5)	2/4	(0x2/4)+(1-0)(1-2/4)	The word Very is not there in the vocabulary so it has been ignored. As P(education | document) > P(cinema | document) Thus, it can be concluded that the document belongs to education
1	depends	0	2/5	(0x2/5)+(1-0)(1-2/5)	2/4	(0x2/4)+(1-0)(1-2/4)	
2	educational	1	4/5	(1x4/5)+(1-1)(1-4/5)	1/4	(1x1/4)+(1-1)(1-1/4)	
3	ethics	0	3/5	(0x3/5)+(1-0)(1-3/5)	1/4	(0x1/4)+(1-0)(1-1/4)	
4	good	1	1/5	(1x1/5)+(1-1)(1-1/5)	2/4	(1x2/4)+(1-1)(1-2/4)	
5	great	0	3/5	(0x3/5)+(1-0)(1-3/5)	2/4	(0x2/4)+(1-0)(1-2/4)	
6	greatness	0	3/5	(0x3/5)+(1-0)(1-3/5)	1/4	(0x1/4)+(1-0)(1-1/4)	
7	institution	1	2/5	(1x2/5)+(1-1)(1-2/5)	1/4	(1x1/4)+(1-1)(1-1/4)	
8	movie	0	1/5	(0x1/5)+(1-0)(1-1/5)	2/4	(0x2/4)+(1-0)(1-2/4)	
9	titanic	0	1/5	(0x1/5)+(1-0)(1-1/5)	2/4	(0x2/4)+(1-0)(1-2/4)	
10	story	0	2/5	(0x2/5)+(1-0)(1-2/5)	2/4	(0x2/4)+(1-0)(1-2/4)	
11	harvard	0	2/5	(0x2/5)+(1-0)(1-2/5)	1/4	(0x1/4)+(1-0)(1-1/42)	
			Prior : P(education) = 3/5 Posterity : Likelihood x Prior = 0.00027		Prior : P(cinema) = 2/5 Posterity : Likelihood x Prior = 0.00018		

Gaussian Naive Bayes Classifier

It is used to classify the continuous data. While dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a normal (or Gaussian) distribution. For example, suppose the training data contains a continuous attribute x and the data is segmented into K classes. If μ_k is the mean and σ_k is the variance for values in x associated with class C_k, then for an observation v the probability distribution of v given a class C_k is given by,

$$P\left(x = v \mid C_k\right) = \frac{1}{\sqrt{2\pi\sigma_k^2}}\, e^{-\left(\frac{\left(v-\mu_k\right)^2}{2\sigma_k^2}\right)}$$

Python Code - Naive Bayes

Document Processing

```
>> train_docs['Class'] = train_docs.Class.map({'class_1':0, 'class_2':1})
>> train_array = train_docs.values
>> X_train = train_array[:,0]
>> y_train = train_array[:,1].astype('int')
```

Bag of Words REpresentation

```
>> from sklearn.feature_extraction.text import CountVectorizer
>> vec = CountVectorizer(stop_words='english')
>> vec.fit(X_train)
>> vocabulary = vec.vocabulary_
>> features = vec.get_feature_names()
>> X_transformed = vec.transform(X_train)
# Compressed sparse matrix
>> matrix = X_transformed.toarray()
# Sparse matrix
>> pd.DataFrame(X_transformed.toarray(), columns=features)
# Converting matrix to dataframe
```

Train Model

```
>> from sklearn.naive_bayes import MultinomialNB
# Multinomial Naive Bayes
>> model = MultinomialNB()
>> model.fit(X_transformed, y_train)

>> from sklearn.naive_bayes import BernoulliNB
# Bernoulli Naive Bayes
>> model = BernoulliNB()
>> model.fit(X_train_transformed,y_train)
```

Predict

```
>> test_docs['Class'] = test_docs.Class.map({'class_1':0, 'class_2':1})
>> test_array = test_docs.values
>> X_test = test_array[:,0]
>> y_test = test_array[:,1]
>> X_test_transformed = vec.transform(X_test)
>> X_test = X_test_transformed.toarray()
>> y_pred_class = model.predict(X_test_tranformed)
# Predict Class
>> y_pred_proba =model.predict_proba(X_test_tranformed)
# Predict probability
```

Analyze Model

```
>> from sklearn import metrics
>> print("PRECISION SCORE :",metrics.precision_score(y_test, y_pred_class))
>> print("RECALL SCORE :", metrics.recall_score(y_test, y_pred_class))
>> print("F1 SCORE :",metrics.f1_score(y_test, y_pred_class))
```

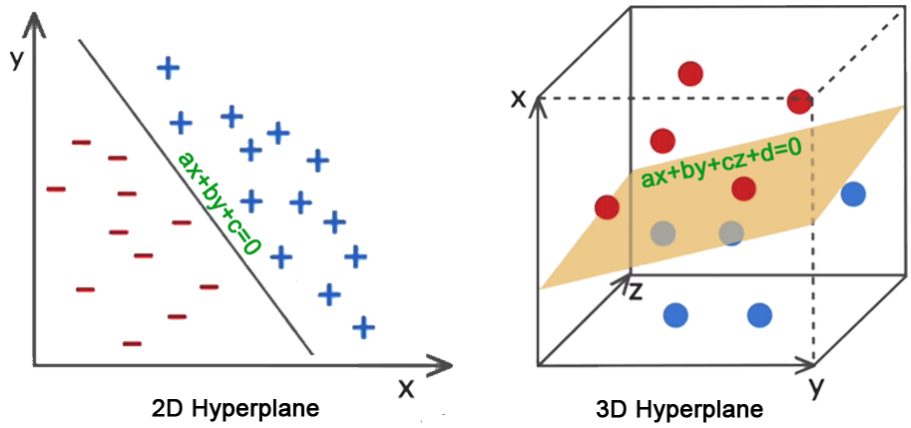
SUPPORT VECTOR MACHINES

4.3.1. MAXIMAL MARGIN CLASSIFIER

The well-known classification models, i.e. logistic regression and naive Bayes though widely used as default prediction and text classification, find limited use in more complex classification problems, such as image classification. But Support Vector Machine models are more capable of dealing with such complex problems, where models such as logistic regression typically fail. SVMs are mostly used for classification tasks, but they can also be used for regression. SVMs belong to the class of linear machine learning models (logistic regression is also a linear model). The model also requires all its attributes to be numeric.

Hyperplanes

Essentially, it is a boundary which separates the data set into its classes. It could be lines, 2D planes, or even n-dimensional planes. In general, the hyperplane in 2D can be represented by a line, whereas a hyperplane in 3D can be represented by a plane as shown in the following figure.



The dimension of a hyperplane is always equal to the $number\ of\ features - 1$. If the value of any point is positive it would mean that the set of values of the features is in one class; however, if it is negative then it belongs to the other class. A value of zero would imply that the point lies on the hyperplane.

Linear Discriminator

The generalized equation of the hyperplane is given as,

$$W_0 + W_1 x_1 + \ldots\ldots + W_d x_d = \sum_{i=1}^{d} W_i x_i + W_0 = 0 \quad where,\ x\ is\ feature\ and\ W\ is\ coefficient$$

Maximal Margin Classifier

There can be multiple possible hyperplanes, which perfectly divide any two classes. But the optimum hyperplane is the one that maintains the largest possible equal distance from the nearest points of both the classes. This is also referred to as a maximal margin classifier. One can think of the margin as a band that the hyperplane has on both its sides.

The distance of any data point from a hyperplane is given by,

$$d = \vec{w} \cdot \vec{x}_i$$

$$where,$$

$$w = (w_0, w_1, \ldots\ldots w_n) \quad is\ the\ normalized\ coeffficient\ vector\ (i.e\ \ \Sigma w_j^2 = 1)$$

$$x_i = (1, x_1, x_2, \ldots\ldots x_n)_i\ is\ the\ i^{th}\ datapoint\ vector$$

If the training data is linearly separable, then two parallel hyperplanes can be selected, which separate the two classes of data, such that the distance between them is as large as possible. The region bounded by these two hyperplanes is called the margin M, and the Maximum Margin Hyperplane is the hyperplane that lies halfway between these two parallel hyperplanes. Also it needs to be ensured that each data point must lie on the correct side of the margin. This can be mathematically expressed as,

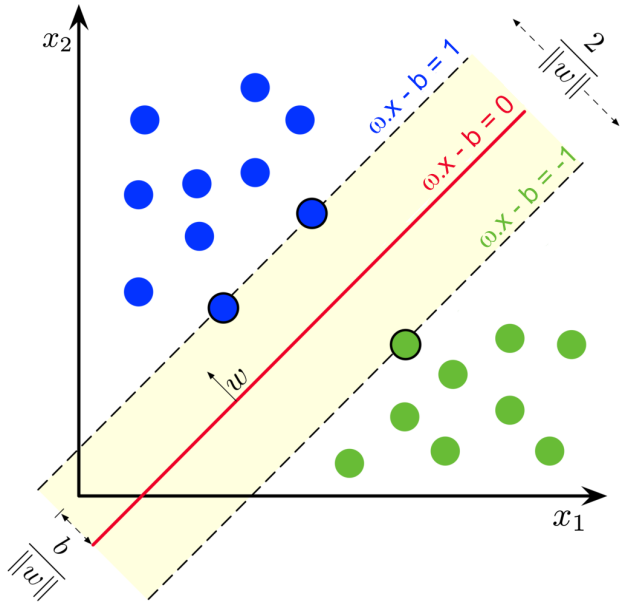
$$y_i \cdot \left(\vec{w} \cdot \vec{x}_i\right) \geq M$$

where, y_i is the label of the i^{th} datapoint (such as $Class\_1\ (+1)$ & $Class\_2\ (-1)$)

Thus, the final optimisation problem becomes,

$$Among\ all\ choices\ of\ \vec{w},\ \frac{Maximize}{M}\ such\ that \begin{cases} \sum_{j=1}^{n} (w_j)^2 = 1 \\ y_i \cdot \left(\vec{w} \cdot \vec{x}_i\right) \geq M \end{cases}$$

The following figure explains it geometrically.



Maximal Margin Classifier is possible only on datasets which are perfectly linearly separable, so it has a rather limited applicability. Thus, in order to classify data points which are partially intermingled one cannot use this classifier.

Support Vectors

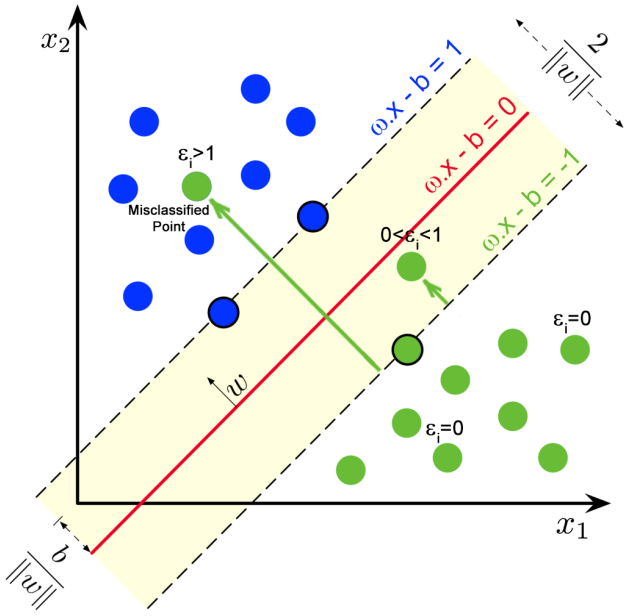
One can observe in the above geometric diagram, that the Maximum Margin Hyperplane is completely determined by the x_i, which lie nearest to it. These x_i's are called support vectors. Thus, the support vectors are the points that lie close to the hyperplane and are the only points that are used in constructing the hyperplane.

4.3.2. SOFT MARGIN CLASSIFIER

The hyperplane that allows certain points to be deliberately misclassified is called the Soft Margin Classifier (also called Support Vector Classifier). Similar to the Maximal Margin Classifier, the Soft Margin Classifier also maximises the margin, but at the same time allows some points to be misclassified.

Slack Variable

In order to allow certain points to be deliberately misclassified, one has to include a new variable called as slack variable ε. A slack variable ε is used to control the misclassifications. It tells where an observation is located relative to the margin and hyperplane. The following figure explains it geometrically.



1. For data points which are at a distance of more than the margin M, i.e. at a safe distance from the hyperplane, the value of the slack variable is 0.

2. For data points correctly classified but falling inside the margin M (or violating the margin), the value of the slack variable lies between 0 and 1.

3. For data points incorrectly classified (i.e. violating the hyperplane), the value of the slack variable is greater than 1.

Thus, the slack variable ε takes a value between 0 to infinity. Lower values of slack are better than higher values ($\varepsilon = 0$ correct classification, $\varepsilon > 1$ incorrect classification, $0 < \varepsilon < 1$ classifies correctly but violates the margin).

Soft Margin Classifier

It has already been proven that the Maximal Margin Classifier can be expressed as,

$$y_i \cdot \left(\vec{w} \cdot \vec{x}_i \right) \geq M$$

If the above condition is imposed on any model, then it implies that each point in the dataset has to be at least a distance of M away from the hyperplane. But unfortunately, hardly any available real datasets are so easily and perfectly separable. Thus, to relax the constraint, one has to include the slack variable ε_i for each datapoint i, modifying the above expression to,

$$y_i \cdot \left(\vec{w} \cdot \vec{x}_i \right) \geq M\left(1 - \varepsilon_i \right)$$

With the inclusion of slack variable ε, the optimisation problem now becomes,

$$\textit{Among all choices of } \vec{w}, \quad \frac{\textit{Maximize}}{M} \textit{ such that } \begin{cases} \sum_{j=1}^{n} w_j^{\,2} = 1 \\ y_i \cdot \left(\vec{w} \cdot \vec{x}_i \right) \geq M\left(1 - \varepsilon_i \right) \end{cases}$$

$$\textit{Provided, the total error allowed } \sum_{i=0}^{n} \varepsilon_i < C$$

Cost of Misclassification 'C'

In the equation $\Sigma\varepsilon_i < C$, given above C is the cost of misclassification and is equal to the sum of all the values of slack variables, i.e it represents the cost of violations to the margin and the hyperplane. Using the notion of the slack variable, one can easily compare any number of Support Vector Classifiers. The cost of misclassification C can be measured for the hyperplanes of all the classifiers, and the one with the least sum of epsilons be chosen as the best fit classifier.

When C is large, the slack variables can be large, i.e. larger number of data points are allowed to be misclassified or to violate the margin. Thus, one gets a hyperplane where the margin is wide and misclassifications are allowed. These models tend to be more flexible, more generalisable, and less likely to overfit (high bias). On the other hand, when C is small, the individual slack variables are forced to be small, i.e. not many data points are allowed to fall on the wrong side of the margin or the hyperplane. Thus, the margin is narrow and there are few misclassifications. These models tend to be less flexible, less generalisable, and more likely to overfit (high variance).

Python Code - SVM

SVM

```
>> from sklearn.svm import SVC
>> model = SVC(C = 1)
# Hyperparameter C used in the SVM formulation (in theory) and the C in the SVC()
functions are the inverse of each other. C is analogous to the penalty imposed for
misclassification, i.e. a higher C will force the model to classify most (training)
data points correctly (and thus, overfit).
>> model.fit(X_train, y_train)
```

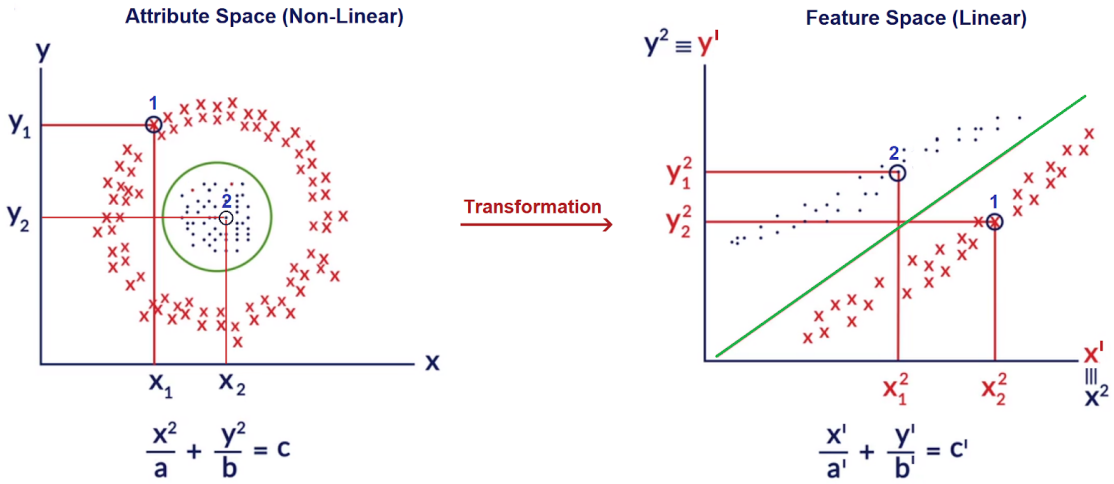
Grid Search SVM

```
>> hyper_params = [{'C': [parameters]}]
>> model = SVC()
>> model_cv = GridSearchCV(estimator=model, param_grid=hyper_params, cv=folds,
                           scoring='accuracy', verbose=1, return_train_score=True)
>> model_cv.fit(X_train, y_train)
>> scores = pd.DataFrame(model_cv.cv_results_)
```

4.3.3. KERNELS

Sometimes it is not possible to imagine a linear hyperplane that can separate the classes reasonably well. So, one needs to tweak the linear SVM model and enable it to incorporate nonlinearity in some way. The Kernels serve this purpose, i.e they enable the linear SVM model to separate non linearly separable data points. The Kernels do not change the linearity of the SVM model, rather these are toppings over the linear SVM model, which somehow enables the model to separate nonlinear data.

Feature Transformation



A nonlinear boundary can be transformed into a linear boundary by applying certain functions to the original attributes to create new features. The original space $(X,\ Y)$ is called the original attribute space, and the transformed space $(X',\ Y')$ is called the feature space.

The preceding diagram represents one such transformation. Now in the above example, the transformation was very neat as the shape of the separator was known to be a circle. But in the real scenario it is going to be very tough to guess the shape (functional form) of the separator, other than that it is linear or nonlinear. Thus, one has to eventually go for a n degree polynomial transformation for m number of attributes.

For example let's consider a 2 degree polynomial transformation of 3 attributes. Solving this, one ends up with 10 features in the new feature space as shown in the following figure.

$$a_1X^2 + a_2Y^2 + a_3Z^2 + a_4XY + a_5YZ + a_6ZX + a_7X + a_8Y + a_9Z + C = 0$$

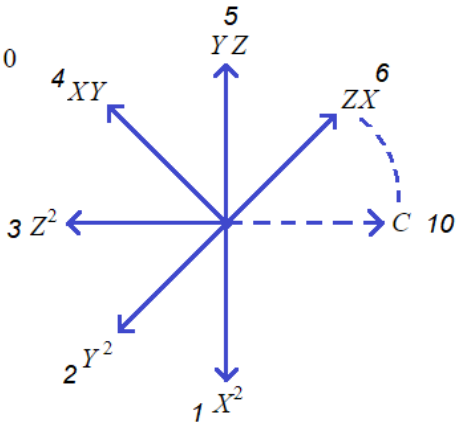
Transformation from 2-D Attribute Space

(X, Y)

↓

10-D Feature Space

$(X^2, Y^2, Z^2, XY, YZ, ZX, X, Y, Z, C)$



Thus, it can be concluded that as the number of attributes increases, the number of dimensions in the transformed feature space also increases exponentially.

The Kernel Trick

Due to the exponential rise in the number of the dimensions during feature transformation, the modelling (i.e. the learning process) becomes computationally expensive. This problem is solved by the use of the Kernel trick. The kernels don't do the feature transformation explicitly (which is a computationally difficult task, as discussed above), rather they use a mathematical hack to do this implicitly.

It is a mathematical fact (Dual Space Optimisation Theory) that for all the linear models in order to find a best fit model, the learning algorithm only requires the pairwise dot product of the observations $(x_i^T \, . \, x_j)$, rather than individual data points x_i or x_j.
Replacing these attribute vectors with their corresponding feature vectors gives the function K defined as follows,

$$\vec{x_i}^T . \vec{x_j} \rightarrow \phi\left(\vec{x_i}\right)^T . \phi\left(\vec{x_j}\right) = K\left(\vec{x_i}, \vec{x_j}\right)$$

This function K is known as the Kernel function. Thus, one just needs to know only the Kernel function and not the mapping function itself, for transforming the non-linear attributes into linear features. The benefits of this implicit transformation are as follows.

1. Manually finding the mathematical transformation is now not required to convert a nonlinear to a linear feature space.

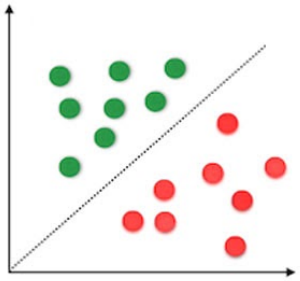
2. Computationally heavy transformations are no longer required.

Some of the most popular types of kernel functions are as follows.

1. Linear Kernel : This is the same as the support vector classifier, or the hyperplane, without any transformation at all.

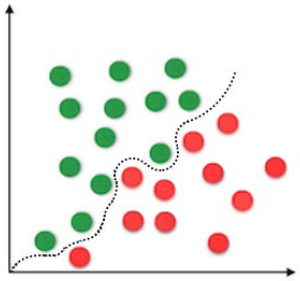
2. Polynomial Kernel : It is capable of creating nonlinear, polynomial decision boundaries.

3. Radial Basis Function Kernel (RBF) : This is the most complex one, which is capable of transforming highly nonlinear feature spaces to linear ones. It is even capable of creating elliptical (i.e. enclosed) decision boundaries.



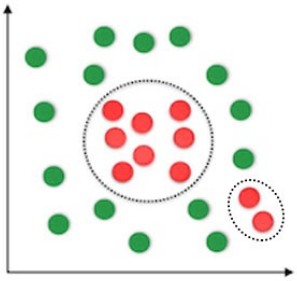
Linear Kernel

$$K(x_i, x_j) = x_i \cdot x_j$$



Polynomial Kernel

$$K(x_i, x_j) = (x_i \cdot x_j + 1)^p$$



RBF Kernel

$$K(x_i, x_j) = e^{-\gamma(x_i - x_j)^2}$$

Thus, one can think of a kernel as a black box. The original attributes are passed into the black box, and it returns the transformed attributes (in a higher dimensional feature space). The SVM algorithm is then shown only the transformed, linear feature space, where it builds the linear classifier as usual.

Choosing a Kernel Function

Choosing the appropriate kernel is very important for building a model of optimum complexity. If the kernel is highly nonlinear, the model is likely to overfit. On the other hand, if the kernel is too simple, then it may not fit the training data well. Usually, it is difficult to choose the appropriate kernel by visualising the data or using exploratory analysis. Thus, cross-validation (or hit-and-trial, if 2-3 types of kernels are chosen) is often a good strategy.

For the polynomial, rbf and sigmoid kernels, the hyperparameter gamma controls the amount of non-linearity in the model. As gamma increases, the model becomes more non-linear, and thus model complexity increases. High values of gamma lead to overfitting (especially for higher values of C).

Python Code - Kernel SVM

```
Kernel SVM
>> from sklearn.svm import SVC
>> model = SVC(C=1, kernel='linear')
# Kernels can be any one of  'linear', 'poly', 'rbf', 'sigmoid' or  'precomputed'.
>> model.fit(X_train, y_train)

Grid Search Kernel SVM
>> hyper_params = [{'C':[parameters], 'gamma':[parameters]}]
>> model = SVC(kernel="rbf")
# hyperparameter gamma is available for 'poly', 'rbf' and 'sigmoid' kernel
>> model_cv = GridSearchCV(estimator=model, param_grid=hyper_params, cv=folds,
                           scoring='accuracy', verbose=1, return_train_score=True)
>> model_cv.fit(X_train, y_train)
>> scores = pd.DataFrame(model_cv.cv_results_)
```

4.3.4. SUPPORT VECTOR REGRESSION

Support Vector Machines (SVMs) are well known in classification problems. The same SVM concepts can be generalized to become applicable to regression problems. Although less popular than SVM, Support Vector Regression (SVR) models have been proven to be an effective tool in real-value function estimation. SVR supports both linear and non-linear regression.

Support Vector Regression (SVR)

The Support Vector Machine can be used as a regression method, maintaining all the main features that characterize the algorithm (i.e. maximal margin), with only a few minor changes. The main objective is to fit as many instances as possible between the lines while limiting the margin violations. SVM generalization to SVR is accomplished by introducing an error insensitive region around the function bordered by the hyperplanes. The objective is to find out the hyperplanes that contain most of the training instances with the absolute error being less than or equal to a specified margin called the maximum error M. Also, for any values that fall outside of the margin, certain slack ε can be provided for deviation from the margin. Although, these deviations have the potential to exist, still one would like to minimize them as much as possible.

However, the main idea is always the same, i.e. minimizing error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated, giving the objective function as,

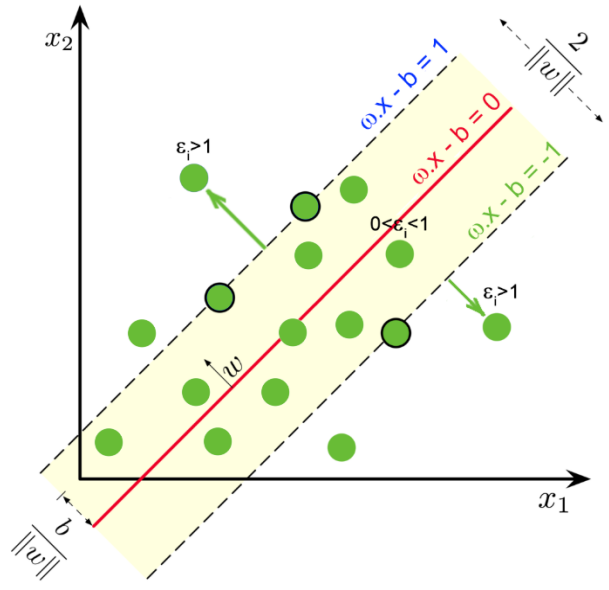
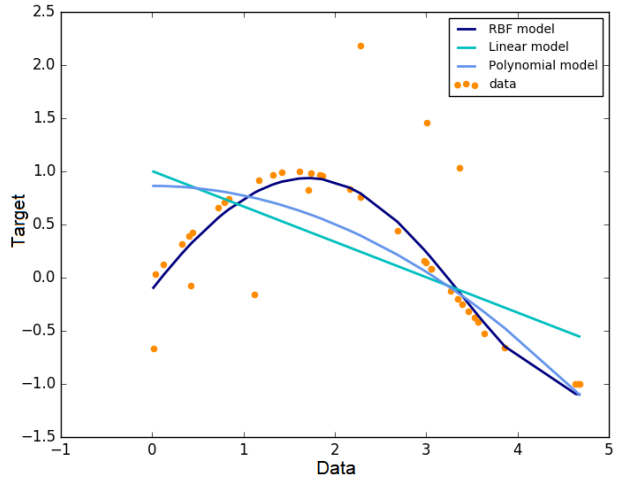
$$y_i \cdot \left(\vec{w} \cdot \vec{x}_i \right) \leq M\left(1 + \varepsilon_i \right)$$

And the final optimisation problem as,

$$Among\ all\ choices\ of\ \vec{w},\ \frac{Minimize}{M}\ such\ that \begin{cases} \sum_{j=1}^{n} w_j^{\,2} = 1 \\ y_i \cdot \left(\vec{w} \cdot \vec{x}_i \right) \leq M\left(1 + \varepsilon_i \right) \end{cases}$$

$$Provided,\ the\ total\ error\ allowed\ \sum_{i=0}^{n} \varepsilon_i < C$$

Similarly, the nonlinear SVR makes use of the kernel functions to transform the data into a higher dimensional feature space, which then makes it possible to perform the linear separation of the data. The following figures explain the linear SVR geometrically and give an example of SVR using linear, polynomial and RBF kernels.

Geometric interpretation of SVR	Various SVR kernels
	

Python Code - SVR

SVR

```
>> from sklearn.svm import SVR
>> model = SVR(C=1, epsilon=0.2, kernel='linear')
# Kernels can be any one of  'linear', 'poly', 'rbf', 'sigmoid' or  'precomputed'.
>> model.fit(X_train, y_train)
```

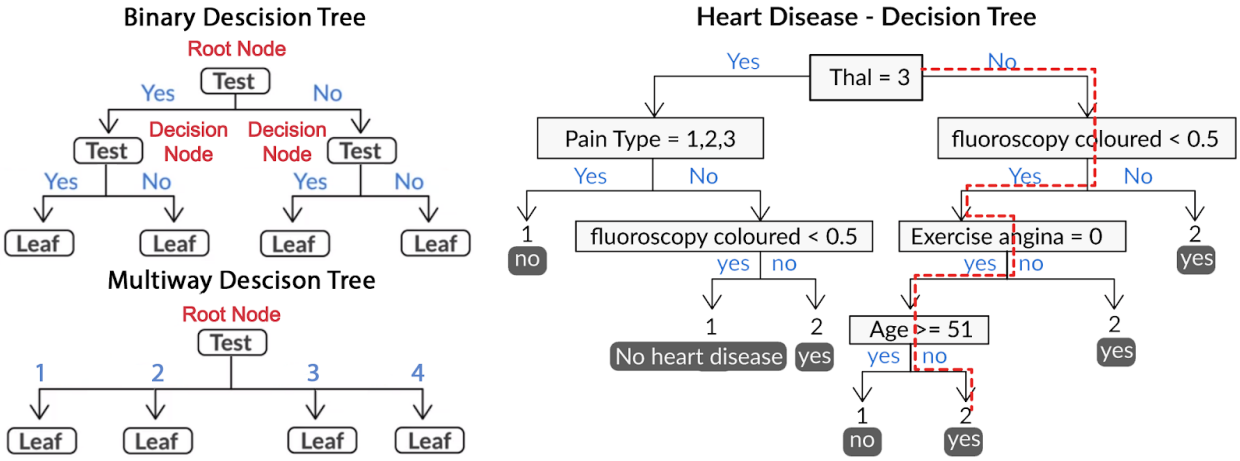
TREE MODELS

4.4.1. DECISION TREES

With high interpretability and an intuitive algorithm, decision trees mimic the human decision making process and excel in dealing with categorical data. Unlike other algorithms like logistic regression or SVMs, decision trees do not find a linear relationship between the independent and the target variable, rather, they can be used to model highly nonlinear data. With decision trees, one can easily explain all the factors leading to a particular decision/prediction. Hence, they are easily understood by business people.

Decision Trees

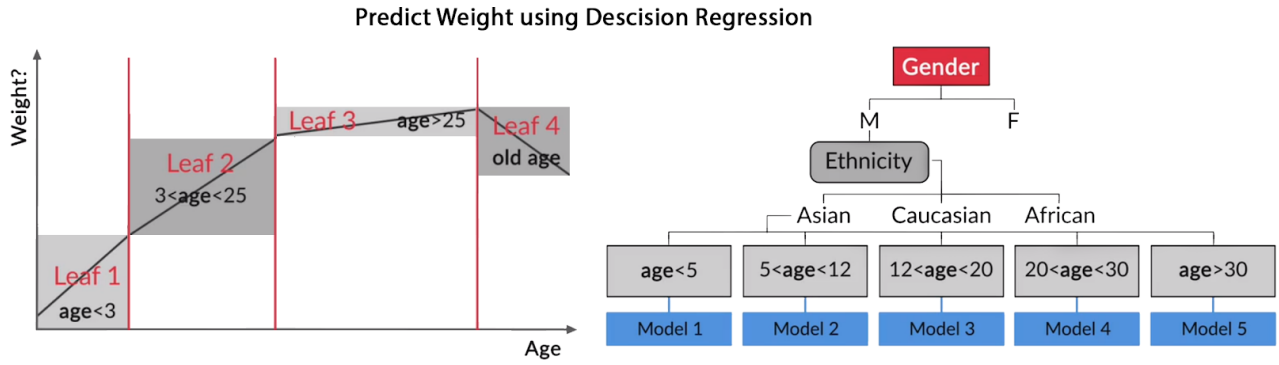
A decision tree uses a tree-like model (resembling an upside-down tree) to make predictions. It is also very similar to decision making in real life (asking a series of questions in a nested if-then-else structure to arrive at a decision). A decision tree splits the data into multiple sets. Then, each of these sets is further split into subsets to arrive at a decision.



If a test splits the data into two partitions it is called a binary decision tree and if it splits the data into more than two partitions, it is called a multiway decision tree. The decision trees are easy to interpret (i.e. almost always, one can identify the various factors that lead to the decision).

Regression with Decision Trees

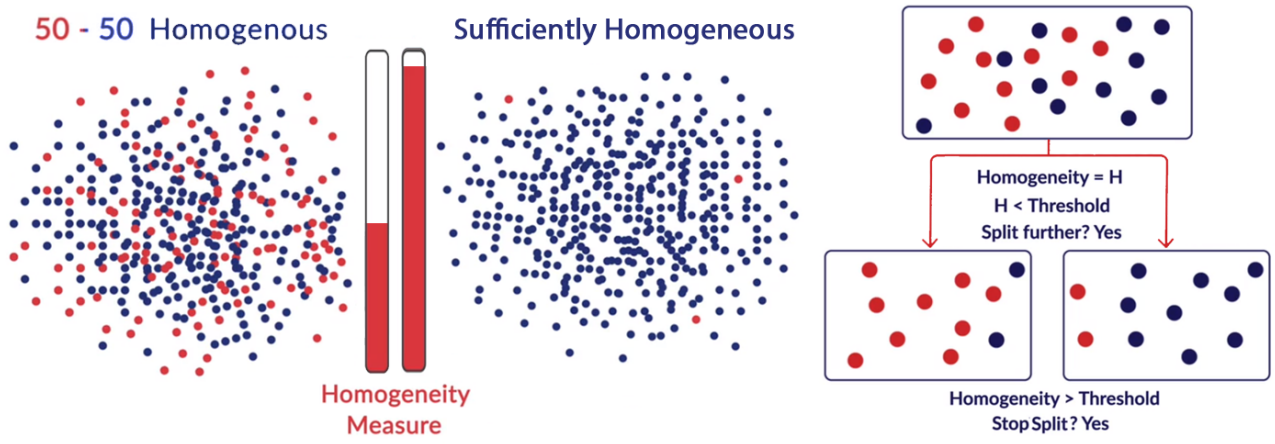
There are cases where one cannot directly apply linear regression to solve a regression problem. The linear regression fits only one model to the entire data set. But there may be cases where the requirement might be to split the data set into multiple subsets and then apply the linear regression to each subset separately. This can be done by using the decision tree to split the data into multiple subsets first. Then a linear regression can be applied to each leaf. The difference between decision tree classification and decision tree regression is that in regression, each leaf represents a linear regression model, as opposed to a class label.



4.4.2. ALGORITHMS FOR DECISION TREE CONSTRUCTION

Homogeneity

A partition which contains data points with all the labels identical (i.e. label X), then the entire partition can be classified as label X (i.e. homogeneous data set). But, in real world data sets, one never gets a completely homogeneous data set (or even nodes after splitting). Thus, one always tries the best to split the nodes such that the resulting nodes are as homogenous as possible (i.e. the generated partitions result in homogeneous data points). For classification tasks, a data set is completely homogeneous if it contains only a single class label. The following figure shows the generic algorithm followed by the decision trees.



The algorithm goes on step by step, picking an attribute and splitting the data such that the homogeneity increases after every split. Usually, the attribute that results in the maximum increase in homogeneity is chosen for splitting. A split that gives a homogenous subset is much more desirable than the one that results in a $50-50$ distribution (in the case of two labels). The splitting is stopped when the resulting leaves are sufficiently homogenous. Here, sufficiently homogeneous means that most of the data points in the set should belong to the same class label. Thus, the decision tree construction problem gets narrowed down to, splitting of the dataset such that the homogeneity of the resulting partitions is maximum.

Gini Index

Gini index is a measure of homogeneity that measures the degree or probability of a particular variable being correctly classified when it is randomly chosen, i.e. if all the elements belong to a single class, then it can be called pure. The degree of Gini index varies between 0 and 1, where 0 denotes that the elements are randomly distributed across various classes and 1 denotes that all elements belong to a certain class. A Gini Index of 0.5 denotes equally distributed elements into some classes. Thus, the higher the homogeneity, the higher the Gini index.

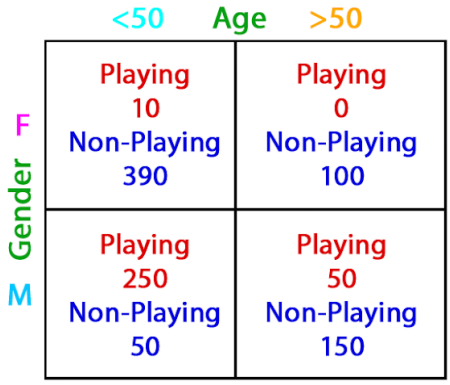
Gini index of a partition D with n labels is given by,

$$Gini_D = \sum_{i=1}^{n} P_i^2 \qquad where,\ P_i \text{ is the probability of a point with label } i \text{ in partition } D$$

While calculating the Gini index of an attribute after splitting, one has to multiply the Gini index of the partition with the fraction of data points of the respective partition. Gini index for an attribute A having n partitions is given by,

$$Gini\ (A)\ = \sum_{i=1}^{n} Fraction\ of\ \ Total\ Observations\ in\ Partition_i \times Gini_{A=i}$$

Following is an example where one has to make a choice between two attributes for splitting, Age and Gender.



Split on Gender

$$\text{Gini Index} = \left[\frac{500}{1000}\right] \times \left[\left(\frac{10}{500}\right)^2 + \left(\frac{490}{500}\right)^2\right] + \left[\frac{500}{1000}\right] \times \left[\left(\frac{300}{500}\right)^2 + \left(\frac{200}{500}\right)^2\right]$$

$$= \left[\frac{1}{2}\right] \times \left[\left(\frac{1}{50}\right)^2 + \left(\frac{49}{50}\right)^2\right] + \left[\frac{1}{2}\right] \times \left[\left(\frac{3}{5}\right)^2 + \left(\frac{2}{5}\right)^2\right] = 0.74$$

Split on Age

$$\text{Gini Index} = \left[\frac{700}{1000}\right] \times \left[\left(\frac{260}{700}\right)^2 + \left(\frac{440}{700}\right)^2\right] + \left[\frac{300}{1000}\right] \times \left[\left(\frac{50}{300}\right)^2 + \left(\frac{250}{300}\right)^2\right]$$

$$= [0.7] \times \left[\left(\frac{26}{70}\right)^2 + \left(\frac{44}{70}\right)^2\right] + [0.3] \times \left[\left(\frac{1}{6}\right)^2 + \left(\frac{5}{6}\right)^2\right] = 0.59$$

On computing, it was found that Gender is a better attribute to split on as it yields a higher value of Gini index as compared to Age. This means that gender gives a better split that helps in distinguishing between football players and non-football players. This is intuitive as well, splitting on gender is expected to be more informative than age because football is usually more popular among males. Thus, one can split the dataset on the attribute Gender.

Entropy

The idea here is to use the notion of entropy (lack of order or predictability) for measuring the homogeneity. Entropy quantifies the degree of disorder in the data. Similar to the Gini index, its value also varies from 0 to 1. If a data set is completely homogenous, then the entropy of such a data set is 0, i.e. there's no disorder. The lower the entropy (or higher the Gini index), the lesser the disorder, and the greater homogeneity. One needs to note that Entropy is a measure of disorderness, while the Gini index is a measure of homogeneity in the data set.

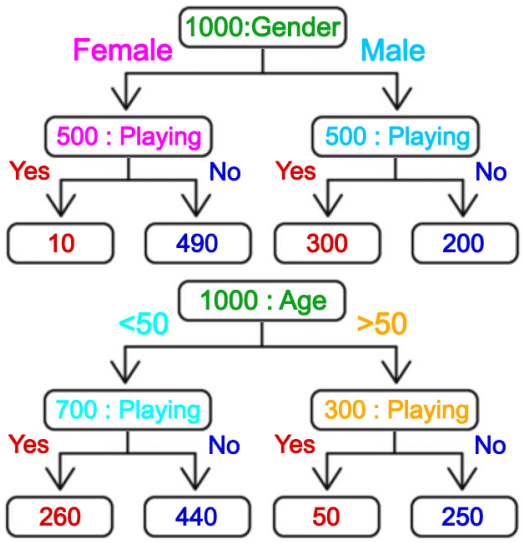
Entropy of partition D with n labels is given by,

$$\varepsilon[D] = -\sum_{i=1}^{n} P_i \log_2 P_i \qquad where,\ P_i\ is\ the\ probability\ of\ \ a\ point\ with\ label\ i\ in\ partition\ D$$

While calculating the Entropy of an attribute after splitting, one has to multiply the Entropy of the partition with the fraction of data points of the respective partition. Entropy for an attribute A having n partitions is given by,

$$\varepsilon[D_A] = -\sum_{i=1}^{n} Fraction\ of\ \ Total\ Observations\ in\ Partition_i \times \varepsilon[D_{A=i}]$$

Following is the same example used for the Gini index, where one has to make a choice between two attributes for splitting, Age and Gender.



Entropy of Original/Parent Data Set

$$\varepsilon[D] = -\left[\left(\frac{310}{1000}\right)\log_2\left(\frac{310}{1000}\right) + \left(\frac{690}{1000}\right)\log_2\left(\frac{690}{1000}\right)\right] = 0.89$$

Split on Gender

$$\varepsilon[D_{Gender}] = -\left(\frac{500}{1000}\right)\times\left[\left(\frac{10}{500}\right)\log_2\left(\frac{10}{500}\right) + \left(\frac{490}{500}\right)\log_2\left(\frac{490}{500}\right)\right]$$
$$-\left(\frac{500}{1000}\right)\times\left[\left(\frac{300}{500}\right)\log_2\left(\frac{300}{500}\right) + \left(\frac{200}{500}\right)\log_2\left(\frac{200}{500}\right)\right] = 0.56$$

Split on Age

$$\varepsilon[D_{Age}] = -\left(\frac{700}{1000}\right)\times\left[\left(\frac{260}{700}\right)\log_2\left(\frac{260}{700}\right) + \left(\frac{440}{700}\right)\log_2\left(\frac{440}{700}\right)\right]$$
$$-\left(\frac{300}{1000}\right)\times\left[\left(\frac{50}{300}\right)\log_2\left(\frac{50}{300}\right) + \left(\frac{250}{300}\right)\log_2\left(\frac{250}{300}\right)\right] = 0.86$$

On computing, it was found that Gender is a better attribute to split on as it yields a lower value of Entropy as compared to Age. Thus, one can split the dataset on the attribute Gender.

Information Gain

Information gain is another measure of homogeneity which is based on the decrease in entropy after a data set is split on an attribute. It measures by how much entropy has decreased between the parent set and the partitions obtained after splitting. Information gain of an attribute A with a partition D is given by,

$$Gain\ (D,A) = \varepsilon[D] - \varepsilon[D_A]$$

where,

$\varepsilon[D]$ *is the entropy of the parent set* D,

$\varepsilon[D_A]$ *is the entropy of the partitions obtained by splitting on attribute* A

As the information gain is equal to the entropy change from the parent set to the partitions, it is maximum when the entropy of the parent set minus the entropy of the partitions is maximum. Thus, while splitting one should choose an attribute such that the information gained is maximum.

Chi-Square

It measures the significance of the relationship between sub-nodes and parent node. Chi-Square of an attribute A with a partition D is given by,

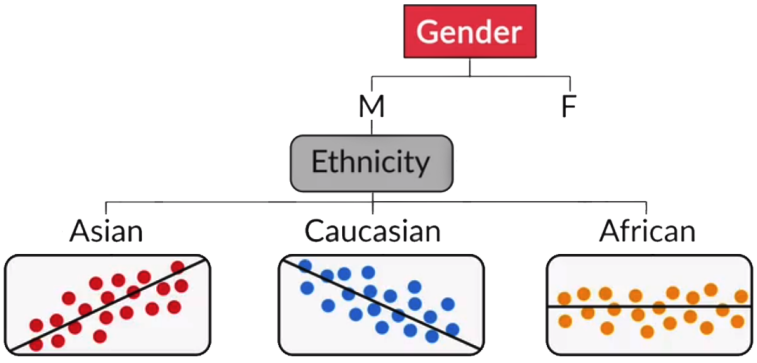
$$\chi_A{}^2 = \frac{\sum\left(D_{Observed\ Value} - D_{Expected\ Value}\right)^2}{D_{Expected\ Value}}$$

Following is the same example used for the Gini index, where one has to make a choice between two attributes for splitting, Age and Gender. On computing, it was found that Gender is a better attribute to split on as it yields a higher Chi-squared value as compared to Age. Thus, one can split the dataset on the attribute Gender.

Node	Observed Playing		Expected Playing		Chi-Square	
	Yes	No	Yes	No	Yes	No
Male	300	200	250	250	3.16	3.16
Female	10	490	250	250	15.16	15.16
Chi-Square for Split on Gender					36.64	
Age < 50	260	440	350	350	4.81	4.81
Age > 50	50	250	150	150	8.16	8.16
Chi-Square for Split on Age					25.94	

R-squared Splitting

So far the splitting has been done only on discrete target variables. But, the splitting can also be done for continuous output variables too. One can calculate the R-squared values of the data sets (before and after splitting) in a similar manner to what is done for linear regression models. The data is split such that the R-squared of the partitions obtained after splitting is greater than that of the original or parent data set. In other words, the fit of the model should be as good as possible after splitting.



Calculating the R-squared at each node

Generic Algorithm for Decision Tree Construction

Following are the steps involved in decision tree construction.

1. The decision tree first decides on an attribute to split on.

2. For the selection of the attribute, it measures the homogeneity of the nodes before and after the split using various measures such as Gini Index, Entropy, Information Gain, Chi-square, R-squared (for regression models) etc.

3. The attribute with maximum homogeneous data set is then selected for splitting.

4. This whole cycle is repeated till one gets a sufficiently homogeneous data set.

There are two widely used Decision Tree algorithms for different applications,

1. CART (Classification and Regression Trees) : It creates a binary tree (a tree with a maximum of two child nodes for any node in the tree) using measures such as Gini Index, Entropy, Information Gain or R-squared. With CART it is not always appropriate to visualise the important features in a dataset as the binary trees tend to be much deeper and more

complex than a non-binary tree (a tree which can have more than two child nodes for any node in the tree). It is best suited for prediction (supervised learning) tasks.

2. CHAID (Chi-square Automatic Interaction Detection) : It creates non-binary trees using the measure Chi-square. CHAID trees tend to be shallower and easier to look at and understand the important drivers (features) in a business problem.

Python Code - Decision Tree Classification

```
Create Train Data
>> X = df.drop('dependent_variable_column',axis=1)
>> y = df['dependent_variable_column']

Train Model
>> from sklearn.tree import DecisionTreeClassifier
>> model = DecisionTreeClassifier(max_depth=level_of_depth)
>> model.fit(X, y)

Visualize Decision Tree Structure
>> from IPython.display import Image
>> from sklearn.externals.six import StringIO
>> from sklearn.tree import export_graphviz
>> import pydotplus, graphviz
>> features = list(X.columns[0:])
>> dot_data = StringIO()
>> export_graphviz(model, out_file=dot_data, feature_names=features,
                   filled=True, rounded=True)
>> graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
>> graph.write_pdf("decision_tree_structure.pdf")
# Creates the pdf with the tree in the current directory
```

4.4.3. TRUNCATION AND PRUNING

Advantages of Decision Trees

1. Predictions made by a decision tree are easily interpretable.

2. A decision tree does not assume anything specific about the nature of the attributes in a data set (as is done by Linear/Logistic regression and SVM). It can seamlessly handle all kinds of data may it be numeric, categorical, string, boolean etc.

3. It does not require normalisation since it has to only compare the values within an attribute.

4. Decision trees often give an idea of the relative importance (importance of attribute decreases as one travels from parent node towards leaf) of the explanatory attributes that are used for prediction.

Disadvantages of Decision Trees

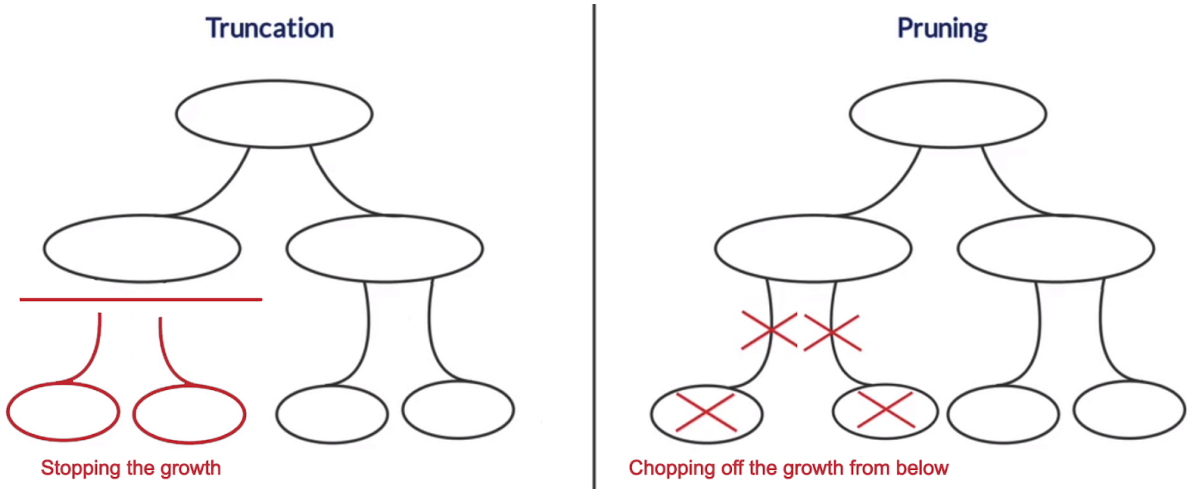
1. Decision trees tend to overfit the data. If allowed to grow without any check on its complexity, a tree will keep splitting till it has correctly classified all the data points in the training set i.e each leaf will denote a datapoint.

2. Decision trees tend to be very unstable, which is an implication of overfitting. A few changes in the data can change a tree considerably.

As can be understood from above, the Decision Trees have a strong tendency to overfit the data. So for practical purposes one has to incorporate certain regularization measures to ensure that the decision tree built does not become more complex than is necessary and starts to overfit. Following are the two ways of regularization of decision trees.

1. Truncation of the decision tree during the training (growing) process, prevents the tree from degenerating into being complex (i.e with one leaf for every data point in the training dataset). Various decision tree stopping criterion can be used to decide the truncation of the tree while performing this pre-pruning process.

2. Pruning of the decision tree in a bottom-up fashion starting from the leaves, after having allowed the tree to grow to any complexity, also prevents the tree from degenerating into a complex one. This post-processing step is more commonly used to avoid overfitting in practical implementations.



Tree Truncation

There are several ways to truncate the decision trees before they start to overfit. Some of these are listed as follows.

1. Minimum Size of the Partition for a Split : Stop partitioning further when the current partition is small enough.

2. Minimum Change in Homogeneity Measure : Do not partition further when even the best split causes an insignificant change in the purity measure (difference between the current purity and the purity of the partitions created by the split).

3. Limit on Tree Depth : If the current node is farther away from the root than a threshold, then stop partitioning further.

4. Minimum Size of the Partition at a Leaf : If any of the partitions after a split has fewer than this threshold minimum, then do not consider the split.

5. Maximum Number of Leaves in the Tree : If the current number of the bottom-most nodes in the tree exceeds this limit then stop partitioning.

Tree Pruning

One of the most popular approaches to pruning called the Reduced Error Pruning, typically uses the validation set (a set of labelled data points kept aside from the original training dataset) for pruning. Following is the algorithm for the approach.

1. Starting at the leaves, each node test (non-leaf) node is considered for pruning.

2. Pruning is done for a node, by removing the entire subtree below the node, making it a leaf. The leaf is then assigned the label of the majority class (or the average of the values in case it is regression) among the training data points that passed through that node.

3. Pruning of a node is done only if the decision tree obtained after the pruning has an accuracy that is no worse on the validation dataset than the tree prior to pruning. This ensures that the parts of the tree which got added due to accidental irregularities in the data are removed, as these irregularities are not likely to repeat.

4. Pruning is stopped when the removal of a node (any of the remaining nodes), decreases the accuracy of the model in the validation dataset.

Python Code - Decision Tree Regularization

Label Encoding Categorical Variables

```
>> from sklearn import preprocessing
>> le = preprocessing.LabelEncoder()
>> df = df.apply(le.fit_transform)
# Decision trees can handle categorical variables. However, label encoding is still
done to get a standard format for sklearn to build the tree.
```

Create Train Data

```
>> X = df.drop('dependent_variable_column',axis=1)
>> y = df['dependent_variable_column']
```

Grid Search

```
>> from sklearn.tree import DecisionTreeClassifier
>> hyper_params = [{'max_depth' : parameters,
                    'min_samples_leaf' : parameters,
                    'min_samples_split' : parameters,
                    'max_features' : parameters,
                    'max_leaf_nodes' : parameters,
                    'criterion' : ['entropy', 'gini']}]

# max_depth: maximum depth of the tree.
# min_samples_leaf: minimum number of samples required to be at a leaf node
# min_samples_split: minimum no. of samples reqd
# max_features: no. of features to consider when looking for the best split
# max_leaf_nodes: maximum number of possible leaf nodes
# criterion: function to measure the quality of a split
>> model = DecisionTreeClassifier()
>> model_cv = GridSearchCV(estimator=model, param_grid=hyper_params,
                           cv=folds, verbose=1)
>> model_cv.fit(X_train, y_train)
>> scores = pd.DataFrame(model_cv.cv_results_)
>> scores.best_score_
>> scores.best_estimator_
```

Train Model

```
>> model_final = DecisionTreeClassifier(max_depth=best_parameter,
                                        min_samples_leaf=best_parameter,
                                        min_samples_split=best_parameter,
                                        max_features=best_parameter,
                                        max_leaf_nodes=best_parameter,
                                        criterion=best_parameter)
>> model_final.fit(X, y)
>> model_final.score(X_test,y_test)
```

Visualize Decision Tree Structure

```
>> from IPython.display import Image
>> from sklearn.externals.six import StringIO
>> from sklearn.tree import export_graphviz
>> import pydotplus, graphviz
>> features = list(X.columns[0:])
>> dot_data = StringIO()
>> export_graphviz(model_final, out_file=dot_data,
                   feature_names=features, filled=True, rounded=True)
>> graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
>> Image(graph.create_png())
```

Test Model

```
>> from sklearn.metrics import classification_report, confusion_matrix
>> y_pred = model_final.predict(X_test)
>> classification_report(y_test, y_pred)
>> confusion_matrix(y_test,y_pred)
```

4.4.4. ENSEMBLES

Ensembles are a group of things viewed as a whole rather than individually. In ensembles, a collection of models is used to make predictions, rather than individual models. The random forest is an ensemble made by the combination of a large number of decision trees. In principle, ensembles can be made by combining all types of models such as logistic regression, neural network or a few decision trees all working in unison.

Diversity and Acceptability

Ensembles of models are somewhat analogous to teams of individual players with different acceptable (at least better than a regular person) skill sets. For an ensemble to work, each model of the ensemble should comply with the following conditions.

1. Each model should be diverse : Diversity ensures that the models serve complementary purposes, which means that the individual models make predictions independent of each other. The advantages of diversity are different depending on the type of ensemble. For example, in a random forest even if some trees overfit, the other trees in the ensemble will neutralise the effect. Also the independence among the trees results in a lower variance of the ensemble compared to a single tree.

2. Each model should be acceptable : Acceptability implies that each model should at least be better than a random model.

Following is an example of how the ensemble works. Considered here is a binary classification problem where the response variable is either a 0 or 1. There is an ensemble of three models,

where each model has an accuracy of $0.7\,(70\,\%)$. The following table shows all the possible cases that can occur while classifying a test data point as 1 or 0.

Case	Result of each Model			Result of Ensemble	Probability
	Model 1	Model 2	Model 3		
1	Correct	Correct	Correct	Correct	0.7 x 0.7 x 0.7 = 0.343
2	Incorrect	Correct	Correct	Correct	0.3 x 0.7 x 0.7 = 0.147
3	Correct	Incorrect	Correct	Correct	0.7 x 0.3 x 0.7 = 0.147
4	Correct	Correct	Incorrect	Correct	0.7 x 0.7 x 0.3 = 0.147
5	Incorrect	Incorrect	Correct	Incorrect	0.3 x 0.3 x 0.7 = 0.063
6	Incorrect	Correct	Incorrect	Incorrect	0.3 x 0.7 x 0.3 = 0.063
7	Correct	Incorrect	Incorrect	Incorrect	0.7 x 0.3 x 0.3 = 0.063
8	Incorrect	Incorrect	Incorrect	Incorrect	0.3 x 0.3 x 0.3 = 0.027

As can be seen from the preceding table,
Probability of the ensemble being correct $= 0.343 + 0.147 + 0.147 + 0.147 = 0.784$
Probability of the ensemble being wrong $= 0.027 + 0.063 + 0.063 + 0.063 = 0.216$

Thus, one can see that an ensemble of just three models gives a boost to the accuracy from 70% to 78.4%. One can also notice that the ensemble has a higher probability of being correct and a lower probability of being wrong than any of the individual models ($0.78 > 0.700$ and $0.216 < 0.300$). In this way, one can also calculate the probabilities of the ensemble being correct and incorrect with 5, 10, 100, 1000 and even a million individual models. The difference in probabilities keeps on increasing with an increase in the number of models, thus improving the overall performance of the ensemble. In general, the more the number of models, the higher the accuracy of an ensemble is.

An ensemble makes a decision by taking the majority vote. It means that if there are n models in the ensemble, and more than half of them give the correct answer, the ensemble will make a correct decision else it will make a wrong decision. This majority voting method performs better on unseen data than any of the individual n models as,

1. Firstly, as each of the individual models is acceptable (i.e. the probability of each model being wrong is less than 0.5), one can easily show that the probability of the ensemble being wrong (i.e. the majority vote going wrong) will be far less than that of any individual model.

2. Secondly, the ensembles avoid getting misled by the assumptions made by individual models. For example, ensembles (particularly random forests) successfully reduce the problem of overfitting as the chances are extremely low that more than half of the models have overfitted. Ensembles ensure that one does not put all their eggs in one basket.

The two most popular ensemble methods are,

1. Bagging : Training a bunch of individual models in a parallel way. Each model is trained by a random subset of the data.

2. Boosting : Training a bunch of individual models in a sequential way. Each individual model learns from mistakes made by the previous model.

4.4.5. BAGGING (RANDOM FORESTS)

Random forest is an ensemble model using bagging as the ensemble method and decision tree as the individual model. The great thing about random forests is that they almost always outperform a decision tree in terms of accuracy. This is the reason why it is one of the most popular machine learning algorithms.

Bagging Algorithm

Bagging or Bootstrapped Aggregation is an ensemble method. It is used when the goal is to reduce the variance of a decision tree classifier. Bootstrapping means to create several random subsets of data (about $30-70\%$) from training samples chosen randomly with replacement. Each collection of subset data is then used to build their tree using a random sample of features while splitting a node (random choice of attributes ensure that the prominent features do not appear in every tree, thus ensuring diversity). Aggregation implies combining the results of different models present in the ensemble, resulting in an ensemble of different models. Average of all the predictions from different trees are then used which is more robust than a single decision tree classifier. It needs to be kept in mind that bagging is just a sampling technique and is not specific to random forests. Following is the algorithm for the same.

1. Consider there are n observations and k features in a training data set. A sample from the training data set is taken randomly with replacement.

2. A subset of k features are selected randomly and whichever feature gives the best split is used to split the node iteratively.

3. The tree is grown to the largest and not pruned (as may be done in constructing a normal tree classifier).

4. The above steps are repeated N times, to construct N trees in the forest. As each tree is constructed independently, it is possible to construct each tree in parallel.

5. The final prediction is given based on the aggregation (majority score) of predictions from the ensemble of the N number of trees.

For measuring the accuracy of the model the OOB (Out-Of-Bag) error can be used. For each bootstrap sample, there is one third of data which is not used in the creation of a tree (i.e. it was out of the sample). This data is referred to as the out of bag data. In order to get an unbiased measure of the accuracy of the model over test data, out of bag error is used.

OOB (Out-Of-Bag) Error

One needs to always avoid violating the fundamental tenet of learning, that is not to test a model on what it has been trained on. The OOB error uses the above principle. It is calculated by using each observation n_i of the dataset n as a test observation. Since, each tree is built on a random bootstrap sample which is about $30-70\%$ of the dataset, for every observation n_i there will be those trees which will not have used the observation n_i in their bootstrap sample. Thus, this observation n_i can be used as a test observation by such trees. All such trees then predict on this observation n_i and the final OOB error is calculated by aggregating the error on each observation

n_i. The OOB error is as good as the cross validation error. In fact, it has been proven that using an OOB estimate is as accurate as using a test data set of a size equal to the training set. Thus, the OOB error completely omits the need for set-aside test data.

Advantages of Bagging

Some of the advantages of random forests are given as follows.

1. Higher resolutions (a smoother decision boundary) in the feature space due to trees being independent of each other. Trees are independent of each other due to the diversity created in each tree by the use of random subsets of features (i.e. not all attributes are considered while making each tree).

2. Reduction in overfitting as the chances are extremely low that more than half of the models have overfitted in the ensemble.

3. Increased stability as the final prediction is given by the aggregation of a large number of trees. The decision made by a single tree (on unseen data) depends highly on the training data since trees are unstable. In a forest, even if a few trees are unstable, averaging out their decisions decreases the mistakes made because of these few unstable trees. A random forest usually always has a lower model variance than an ordinary individual tree.

4. Immunity from the curse of dimensionality. Since each tree does not consider all the features, the feature space reduces. This makes the algorithm immune to a large feature space causing computational and complexity issues. This in turn also allows it to handle a large number of input variables effectively without the necessity of variable deletion.

5. Maintains accuracy for missing data by having an effective method for estimating missing data even when a large proportion of the data are missing.

6. Allows parallelizability, as each tree can be built separately owing to the fact that each of the trees are independently built on different data and attributes. This allows one to make full use of the multi-core CPU to build random forests.

7. Uses all the data to train the model. There is no need for the data to be split into training and validation samples as one can calculate the OOB (Out-of-Bag) error using the training set which gives a really good estimate of the performance of the forest on unseen data.

Even though bagging has many advantages, at times it may not give precise values for the classification and regression model. It is because the final prediction is based on the mean predictions from subset trees.

Time taken to build a Forest

The time taken to construct a forest of S trees, on a dataset which has k features and n observations depend on the following factors.

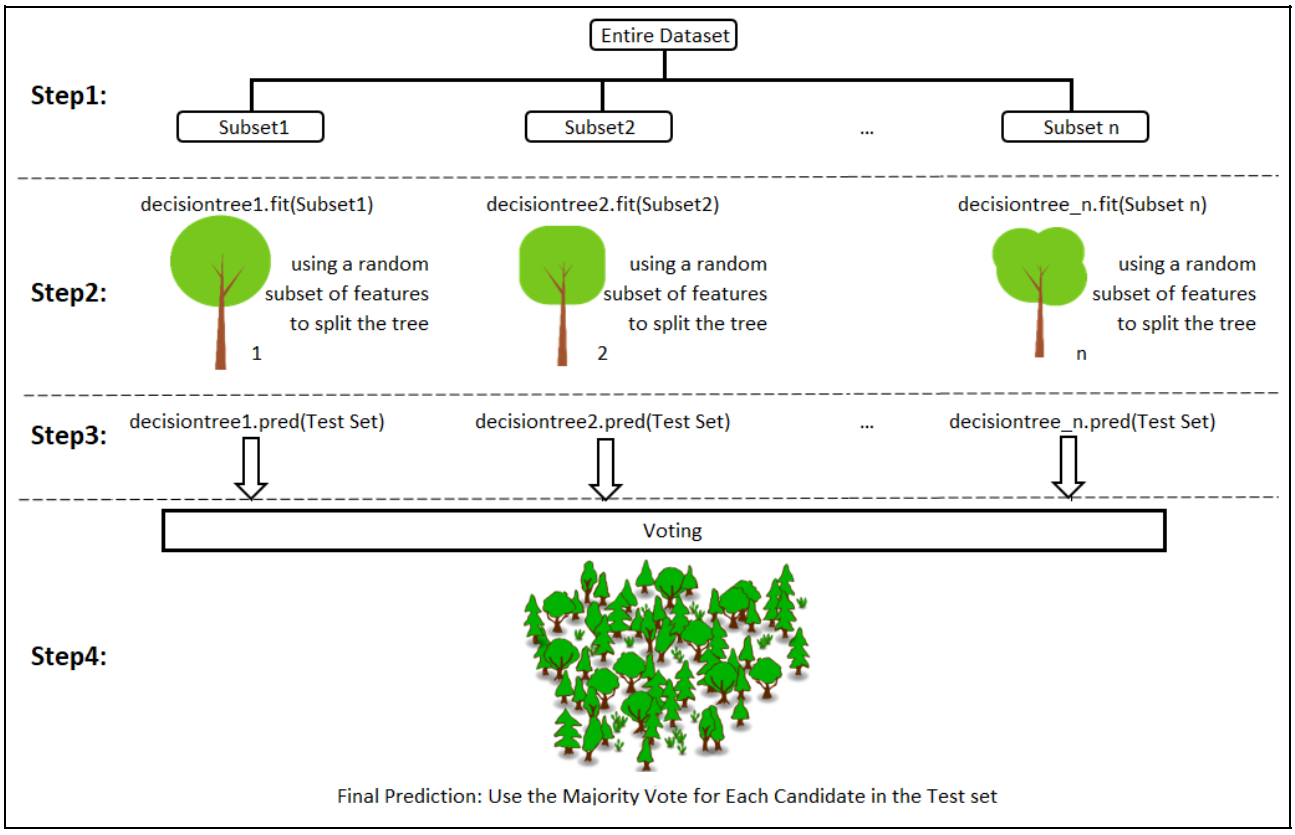
1. Number of Trees : The time is directly proportional to the number of trees. The time required to build each tree is proportional to the time spent in creating the $log(n)$ levels of trees. But this time can be reduced by creating the trees in parallel.

2. Size of Bootstrap Sample : Generally the size of a bootstrap sample is $30-70\%$ of n. The smaller the size the faster it takes to create a forest.

3. Size of Subset of Features for splitting a Node : Usually, this is taken as $\sqrt{k}$ in case of classification and $k/3$ in case of regression.

Random Forest

The following figure gives a representation of random forests using bagging.



Python Code - Random Forest

```
Grid Search
>> from sklearn.ensemble import RandomForestClassifier
>> hyper_params = [{'n_estimators' : parameters,
                    'max_depth' : parameters,
                    'min_samples_leaf' : parameters,
                    'min_samples_split' : parameters,
                    'max_features' : parameters,
                    'max_leaf_nodes' : parameters,
                    'criterion' : ['entropy', 'gini']}]
# n_estimators: number of trees in the forest
# max_depth: maximum depth of the tree
# min_samples_leaf: minimum number of samples required to be at a leaf node
# min_samples_split: minimum no. of samples reqd
# max_features: no. of features to consider when looking for the best split
# max_leaf_nodes: maximum number of possible leaf nodes
# criterion: function to measure the quality of a split
>> model = RandomForestClassifier()
>> model_cv = GridSearchCV(estimator=model, param_grid=hyper_params,
               cv=folds, n_jobs=-1, verbose=1)
>> model_cv.fit(X_train, y_train)
```

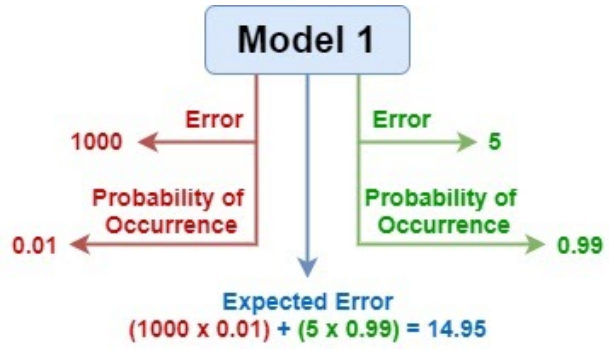
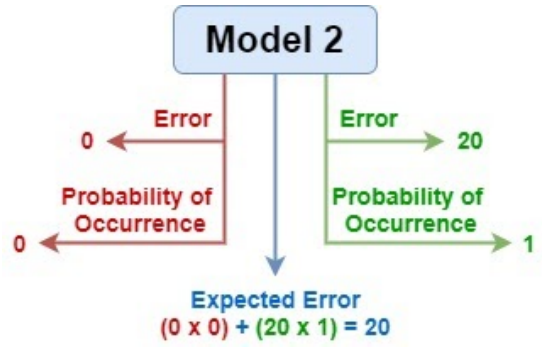
```
>> model_cv.best_score_
>> model_cv.best_params_

Train Model
>> model_final = RandomForestClassifier(bootstrap=True,
                                        n_estimators=best_parameter,
                                        max_depth=best_parameter,
                                        min_samples_leaf=best_parameter,
                                        min_samples_split=best_parameter,
                                        max_features=best_parameter,
                                        max_leaf_nodes=best_parameter,
                                        criterion=best_parameter)
>> model_final.fit(X, y)
>> model_final.score(X_test,y_test)

Test Model
>> from sklearn.metrics import classification_report, confusion_matrix
>> y_pred = model_final.predict(X_test)
>> classification_report(y_test, y_pred)
>> confusion_matrix(y_test,y_pred)
```

4.4.6. BOOSTING

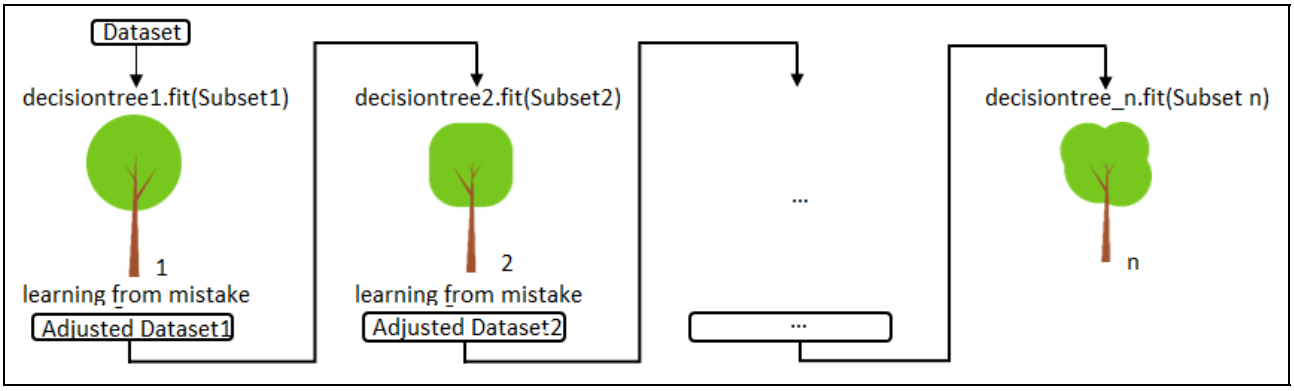
Boosting is a very powerful idea in the field of machine learning. Some of the most popular boosting algorithms being used are AdaBoost, Gradient Boosting and XGBoost.

Boosting

The key idea of boosting is to create an ensemble which makes high errors only on the less frequent data points. Consider the following example showing the expected errors of two models.

	
This model works extremely well for the majority of the data points but goes horribly wrong for few data points.	This model works consistently for all the data points but is not preferable as it is worse than the first model.

As can be seen the aim is to choose a model which minimises the total expected error. Boosting leverages the fact that one can build a series of models specifically targeted at the data points which have been incorrectly predicted by the other models in the ensemble. If a series of models keep reducing the average error, one will have an ensemble having extremely high accuracy. To summarize boosting is a way of generating a strong model from a weak learning algorithm (SVM, regression, and other techniques are algorithms which are used to create models). The following figure gives a representation of the same.



Usually a weak learning algorithm produces a model that does marginally better than a random guess (a random guess has a $50\,\%$ chance of being right) having around 60 to $70\,\%$ chance of being correct. The final objective of boosting algorithms is to create a strong model by making an ensemble of such weak models. Weak learners can be created by applying regularisation rules.

AdaBoost Algorithm

AdaBoost stands for Adaptive Boosting. The AdaBoost algorithm improves the performance by boosting the probability of some points while suppressing the probability of others. Consider a model $M = A(T, D)$, which uses an algorithm A over a training data set T having uniform distribution D (the weight assigned to each data point). Now, with each new model, the distribution of the data is changed such that it boosts the probability of points that are incorrectly classified and suppresses the probability of points that are correctly classified. In other words, the distribution changes after every iteration. This distribution is used for the calculation of the objective function that needs to be minimized while optimising the model. In other words, the objective function is the expected value of the loss which transforms to the following objective function when the distribution is not uniform.

$$Objective\ Function = \frac{Min}{\mathscr{L}} E_D\left[\mathscr{L}\left(h\left(\vec{x}_i\right), y_i\right)\right] = \frac{Min}{\mathscr{L}} \sum_{i=1}^{n} P_D\left(x_i\right).\mathscr{L}\left(h\left(x_i\right), y_i\right)$$

So, there are essentially two major steps involved in the AdaBoost algorithm,

1. Modification of the current distribution to create a new distribution to generate a new model.
2. Calculation of the weights given to each of the models to get the final ensemble.

Following is the algorithm for the same.

1. The probabilities of the distribution having n datapoints is initialized as,

$$P_{D_t}\left(x_i\right) = \frac{1}{n}$$

2. An algorithm h_t is trained on the training data using the respective probabilities.

3. The algorithm is trained such that it performs well on all the data points. Performing well means that the t^{th} algorithm h_t should have a low expected loss $(0-1)$ on the data points for the current distribution $P_{D_t}(x_i)$. i.e,

$$h_t = \underset{\mathscr{L}}{Min}\ E_{D_t}\left[\mathscr{L}\left(h(\vec{x}_i), y_i\right)\right] = \underset{\mathscr{L}}{Min} \sum_{i=1:y_i \neq h(x_i)}^{n} P_{D_t}\left(\mathscr{L}\left(h(\vec{x}_i), y_i\right)\right)$$

This expected loss is called the weighted loss because the loss is not computed on the instances in the training set directly, but rather on the weighted instances in the training set, i.e. the error committed by h_t given by,

$$\in_t = \sum_{i=1:y_i \neq h(x_i)}^{n} P_{D_t}(x_i)$$

4. Based on the performance of the of the algorithm h_t, a weight is assigned to it given by,

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \in_t}{\in_t}\right)$$

The higher the value of ϵ_t the the larger the denominator and smaller the numerator and thus smaller will be the value of α_t. Thus, if the algorithm h_t, has a high error ϵ_t, then it is assigned a smaller weight (i.e. it will contribute less to the output of the ensemble).

5. Once the t^{th} algorithm is added to the ensemble, the training set distribution is recomputed to assign each data point with a probability proportional to how well the current ensemble H_t performs on the training set, i.e.

$$P_{D_{t+1}}(x_i) = \frac{P_{D_t}(x_i) \cdot e^{-\alpha_t y_i h_t(x_i)}}{z_t}$$

$where,$

$$z_t\ is\ the\ normalizing\ factor \sum_{i=1}^{n} P_{D_t}(x_i) \cdot e^{-\alpha_t y_i h_t(x_i)}$$

It can be seen that if $h_t(x_i) = y_i$, then $y_i\, h_t(x_i) = 1$, which means that $e^{-\alpha_t y_i h_t(x_i)} = e^{-\alpha_t}$. Similarly, if $h_t(x_i) \neq y_i$, then $y_i\, h_t(x_i) = -1$, which means that $e^{-\alpha_t y_i h_t(x_i)} = e^{\alpha_t}$. Thus, the exponential term is smaller if the algorithm's prediction agrees with the true value, i.e. higher probability is assigned to the i^{th} data point if the algorithm h_t is wrong on x_i.

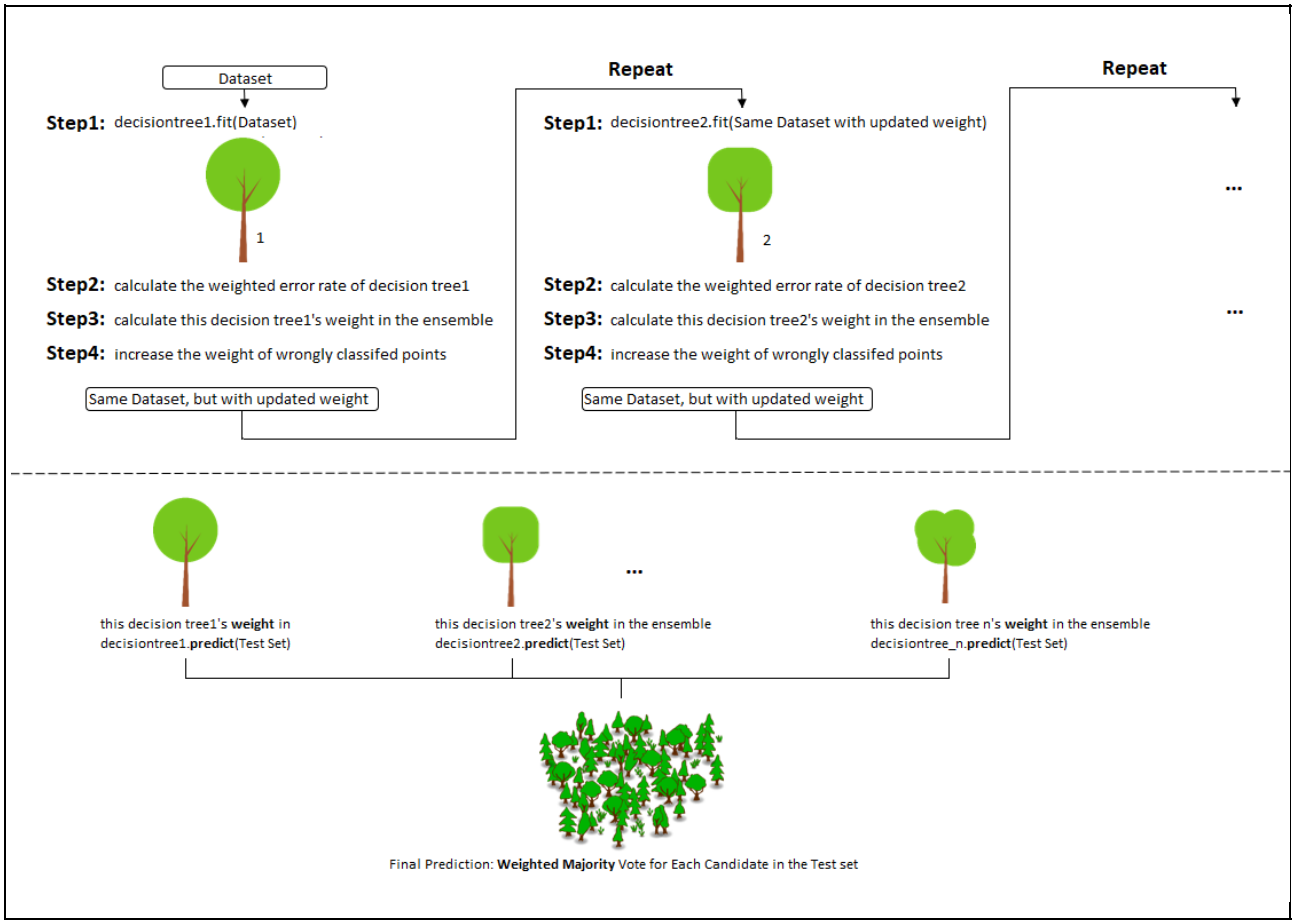
The idea behind this is that the $(t+1)^{th}$ algorithm will correct the errors that the first t algorithms made on the training set. More specifically, after selecting the first t algorithms, it is ensured that the $(t+1)^{th}$ algorithm performs well on those data points for which the first t algorithms had performed poorly.

6. The above steps starting from $Step-2$ to $Step-5$ is repeated N times, to construct an ensemble of N models. Each model is dependent on the previous model so these are constructed sequentially.

7. The final model is given by,

$$H = \sum_{t=1}^{N} \alpha_t h_t$$

It should be noted that before applying the AdaBoost algorithm, one should remove the Outliers. Since, AdaBoost tends to boost up the probabilities of misclassified points and there is a high chance that outliers will be misclassified, it will keep increasing the probability associated with the outliers and make the progress difficult. The following figure gives a representation of AdaBoost algorithm.



Python Code - AdaBoost Algorithm

Train Model

```
>> from sklearn.tree import DecisionTreeClassifier
# Base estimator shallow decision tree : a weak learner with max_depth as 2
>> model_0 = DecisionTreeClassifier(max_depth=2, random_state=100)
# Training the shallow decision tree
>> model_0.fit(X_train, y_train)
>> from sklearn.ensemble import AdaBoostClassifier
# Training the adaboost model
>> model = AdaBoostClassifier(base_estimator=model_0, n_estimators = no_trees)
>> model.fit(X_train, y_train)
```

Predict

```
>> y_pred = model.predict(X_test)
```

Evaluate Model

```
>> from sklearn.metrics import accuracy_score
>> score = metrics.accuracy_score(y_test, y_pred)
```

Grid Search AdaBoost

```
>> hyper_params = {'base_estimator__max_depth':[parameters],
                   'n_estimators':[parameters]}
>> model_0 = DecisionTreeClassifier()
>> model = AdaBoostClassifier(base_estimator=model_0)
>> model_cv = GridSearchCV(estimator=model, param_grid=hyper_params, cv=folds,
                           scoring='roc_auc', verbose=1, return_train_score=True)
>> model_cv.fit(X_train, y_train)
>> scores = pd.DataFrame(model_cv.cv_results_)
```

Gradient Boosting Algorithm

The Gradient Boosting algorithm improves the performance by learning from the mistake (i.e. residual error) directly, rather than updating the weights of data points. Consider a crude model $F_0(x_i)$ which gives a certain output $y_i^0 = y_{avg}$ which is the mean of all the target values. Now, next model $F_1(x_i)$ is trained over the residue $y_i - y_i^0$, i.e. the target variable for model $F_1(x_i)$ is the residual $y_i - y_i^0 = y_i - F_0(x_i)$ and not the actual target variable y_i. Similarly, the subsequent models $F_t(x_i)$ are trained over the residue $y_i - y_i^{t-1} = y_i - [F_0(x_i) + F_1(x_i) + \ldots + F_{t-1}(x_i)]$. The idea being to basically close in on the gap between the output of the earlier models and the expected output, with every new model. This strategy of closing in on the residues with each model can be seen as taking a step towards the negative gradient of the loss function similar to what is done in gradient descent. So, there are essentially two major steps involved in the Gradient Boosting algorithm,

1. Finding the residual and fitting a new model on the residual.
2. Adding the new model to the older model and continuing the next iteration.

Following is the algorithm for the same.

1. A crude model $F_0(x_i)$ is initialized having a constant value as the output.

2. The loss $L(y_i, F_t(x_i))$ for the current model can be computed using various loss functions depending upon the kind of model being used. AdaBoost uses an exponential loss function. As the squared error loss functions are very sensitive to outliers, alternate loss functions, such as the Huber loss criterion can be used. The Huber loss criterion is defined as follows,

$$\mathscr{L}\left(y,\ F_t\right) = \begin{cases} \left(y - F_t\right)^2 & if\ \left|\left(y - F_t\right)\right| \le \delta \\ 2\delta\left|\left(y - F_t\right)\right| - \delta^2 & otherwise \end{cases}$$

3. In order to reduce this loss an incremental model h_{t+1} is generated, which is fitted on the target values given by the negative gradient of the current loss function or pseudo-residuals, i.e,

$$h_{t+1} = \frac{Min}{\mathscr{L}} \sum_{i=1}^{n} \mathscr{L}\left(-\frac{\partial \mathscr{L}\left(y_i,\ F_t\left(x_i\right)\right)}{\partial F_t\left(x_i\right)},\ h_{t+1}\left(x_i\right)\right)$$

In simple words an algorithm h_{t+1} is trained using the training set $\{x_i, y_i'\}$ where y_i' is,

$$y_i' = -\frac{\partial \mathscr{L}\left(y_i,\ F_t(x_i)\right)}{\partial F_t(x_i)}$$

4. Once the model h_{t+1} is determined, the next model F_{t+1} is computed as,

$$F_{t+1} = F_t + \lambda_t h_{t+1}$$

5. The above steps starting from $Step-2$ to $Step-4$ is repeated N times, to construct an ensemble of N models. Each model is dependent on the previous model so these are constructed sequentially.

6. The final model is given by,

$$H = F_N = F_0 + \sum_{t=1}^{N} \lambda_t h_t = F_{N-1} + \lambda_t h_N$$

λ_t is the learning rate having values typically between 0 and 1. Smaller values of λ_t lead to a larger number of trees because with a slower learning rate, larger number of trees are required to reach the minima. This, in turn, leads to longer training time. On the other hand, if λ_t is large, a lesser number of trees will be required, but there's the risk that the minima might actually be missed altogether because of the long strides.

XGBoost Algorithm

Extreme Gradient Boosting (XGBoost) is similar to the Gradient Boosting framework but a more efficient and advanced implementation of Gradient Boosting algorithm. It has become one of the most popular algorithms due to its robust accuracy. Both XGBoost and GBM follow the principle of gradient boosted trees, but XGBoost uses a more regularised model formulation to control overfitting, which gives it better performance, which is why it's also known as regularised boosting technique.

Any ideal machine learning model has an objective function which is a sum of the Loss function L and regularization Ω. The Loss function controls the predictive power of the algorithm and the regularization Ω controls its simplicity. The Gradient Boosting algorithm has its objective function as only the Training Loss while the XGBoost algorithm has an objective function that constitutes a loss function evaluated over all predictions and sum of regularization terms for all predictors. XGBoost algorithm is the same as the Gradient Boosting algorithm, just that the model being fit at each iteration is a decision tree. Hence, there is a change in $Step-3$ and $Step-4$. Following is the algorithm for the same.

1. A crude model $F_0(x_i)$ is initialized having a constant value as the output.

2. The loss $L(y_i, F_t(x_i))$ for the current model can be computed using various loss functions depending upon the kind of model being used. AdaBoost uses an exponential loss function. As the squared error loss functions are very sensitive to outliers, alternate loss functions, such as the Huber loss criterion can be used. The Huber loss criterion is defined as follows,

$$\mathscr{L}\left(y,\ F_t\right) = \begin{cases} \left(y - F_t\right)^2 & if\ \left|\left(y - F_t\right)\right| \le \delta \\ 2\delta\left|\left(y - F_t\right)\right| - \delta^2 & otherwise \end{cases}$$

3. In order to reduce this loss a shallow decision tree D_t is generated having J_t terminal nodes, which is fitted on the target values given by the negative gradient of the current loss function or pseudo-residuals, i.e,

$$D_t = \frac{Min}{\mathscr{L}} \sum_{i=1}^{n} \mathscr{L}\left(-\frac{\partial \mathscr{L}\left(y_i,\ F_t\left(x_i\right)\right)}{\partial F_t\left(x_i\right)},\ D_t\left(x_i\right)\right)$$

   In simple words an algorithm h_{t+1} is trained using the training set $\{x_i, y_i'\}$ where y_i' is,

$$y_i' = -\frac{\partial \mathscr{L}\left(y_i,\ F_t\left(x_i\right)\right)}{\partial F_t\left(x_i\right)}$$

   The terminal nodes R_t generated in the decision tree D_t is given by,

$$R_t = \sum_{j=1}^{J_t} R_j$$

4. The incremental model h_{t+1} constitutes of all the Ω_j's where each Ω_j is given by,

$$\Omega_j = \frac{Min}{\mathscr{L}} \sum_{i=1:x_i \in R_j}^{n} \mathscr{L}\left(y_i,\ F_t\left(x_i\right) + D_t\left(x_i\right)\right)$$

   Thus,

$$h_{t+1} = \sum_{j=1}^{J_t} \Omega_j$$

5. Once the model h_{t+1} is determined, the next model F_{t+1} is computed as,

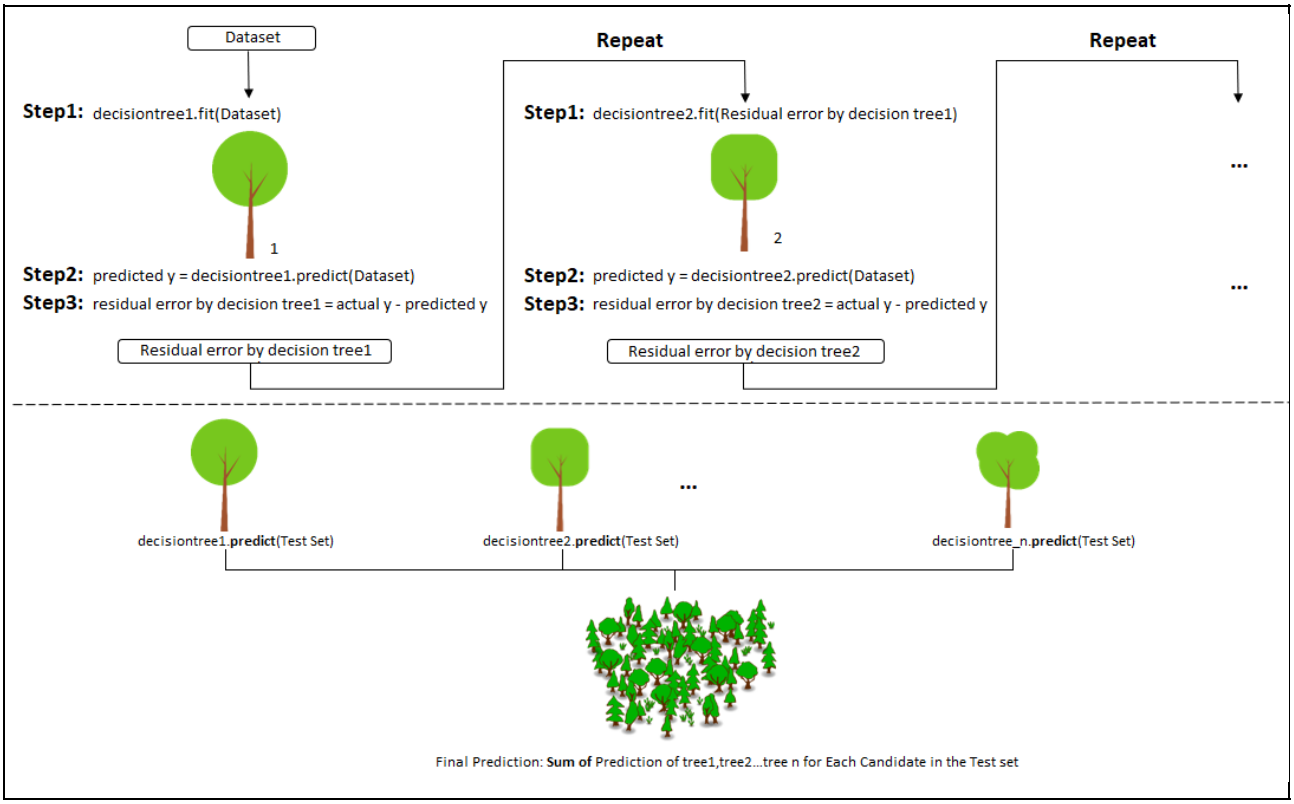
$$F_{t+1} = F_t + \lambda_t h_{t+1}$$

6. The above steps starting from $Step-2$ to $Step-5$ is repeated N times, to construct an ensemble of N models. Each model is dependent on the previous model so these are constructed sequentially.

7. The final model is given by,

$$H = F_N = F_0 + \sum_{t=1}^{N} \lambda_t h_t = F_{N-1} + \lambda_t h_N$$

λ_t is the learning rate (also known as shrinkage) is used to regularize the gradient tree boosting algorithm. Some other ways of regularization are explicitly specifying the number of trees T and doing subsampling. It should be noted that one should not tune both λ_t and T together since a high λ_t already implies a low value of T and vice-versa. Subsampling is the process of training the model in each iteration on a fraction of data (similar to how random forests build each tree). A typical value of subsampling is 0.5 while it ranges from 0 to 1. In random forests, subsampling is very critical to ensure diversity among the trees, since otherwise, all the trees will start with the same training data and therefore look similar. But, this is not a big problem in boosting since each tree is any way built on the residual and gets a significantly different objective function than the previous one. Gradient Boosting has a tendency to overfit with an increasing number of trees (it can be kept on check by consistently monitoring the validation data accuracy). But, this issue is automatically addressed in XGBoost as it uses a more regularised model formulation. The following figure gives a representation of the Gradient Boosting algorithm.



Advantages of XGBoost

1. Parallel Computing : XGBoost, by default uses all the cores of the machine enabling its capacity to do parallel computation.

2. Regularization : The biggest advantage of XGBoost is that it uses regularization and controls the overfitting and simplicity of the model which gives it better performance.

3. Enabled Cross Validation : XGBoost is enabled with internal Cross Validation function.
4. Missing Values : XGBoost is designed to handle missing values internally. The missing values are treated in such a manner that if there exists any trend in missing values, it is captured by the model.

5. Flexibility : XGBoost is not just limited to regression, classification, and ranking problems, it supports user-defined objective functions as well. Furthermore, it supports user-defined evaluation metrics as well.

Python Code - Gradient Boosting Algorithm

Train Model

```
>> from sklearn.ensemble import GradientBoostingClassifier
>> model = GradientBoostingClassifier(max_depth=2, n_estimators=200)
>> model.fit(X_train, y_train)
```

Predict

```
>> y_pred = model.predict(X_test)
```

Evaluate Model

```
>> from sklearn.metrics import accuracy_score
>> score = metrics.accuracy_score(y_test, y_pred)
```

Grid Search Gradient Boosting

```
>> hyper_params = {'learning_rate':[parameters], 'subsample':[parameters]}
>> model_cv = GridSearchCV(estimator=model, param_grid=hyper_params, cv=folds,
                           scoring='roc_auc', verbose=1, return_train_score=True)
>> model_cv.fit(X_train, y_train)
>> scores = pd.DataFrame(model_cv.cv_results_)
```

Python Code - XGBoost Algorithm

Train Model

```
>> import xgboost as xgb
>> from xgboost import XGBClassifier
>> from xgboost import plot_importance
>> params = {'learning_rate': 0.2,'max_depth': 2,'n_estimators':200,'subsample':0.6,
             'objective':'binary:logistic'}
>> model = XGBClassifier(params = params)
>> model.fit(X_train, y_train)
```

Predict

```
>> y_pred = model.predict_proba(X_test)
```

Evaluate Model

```
>> from sklearn.metrics import roc_auc_score
>> score = roc_auc_score(y_test, y_pred[:, 1])
>> importance = dict(zip(X_train.columns, model.feature_importances_))
```

Grid Search Gradient Boosting

```
>> hyper_params = {'learning_rate':[parameters], 'subsample':[parameters]}
>> model = XGBClassifier(max_depth=2, n_estimators=200)
>> model_cv = GridSearchCV(estimator=model, param_grid=hyper_params, cv=folds,
                           scoring='roc_auc', verbose=1, return_train_score=True)
>> model_cv.fit(X_train, y_train)
>> scores = pd.DataFrame(model_cv.cv_results_)
```

4.4.7. TREE REGRESSION

Decision Trees are divided into Classification and Regression Trees. Classification trees are used to separate the dataset into classes belonging to the response variable. Whereas, the Regression trees are needed when the response variable is numeric or continuous. The Decision Tree Regression is both a non-linear and non-continuous model.

Decision Tree Regression

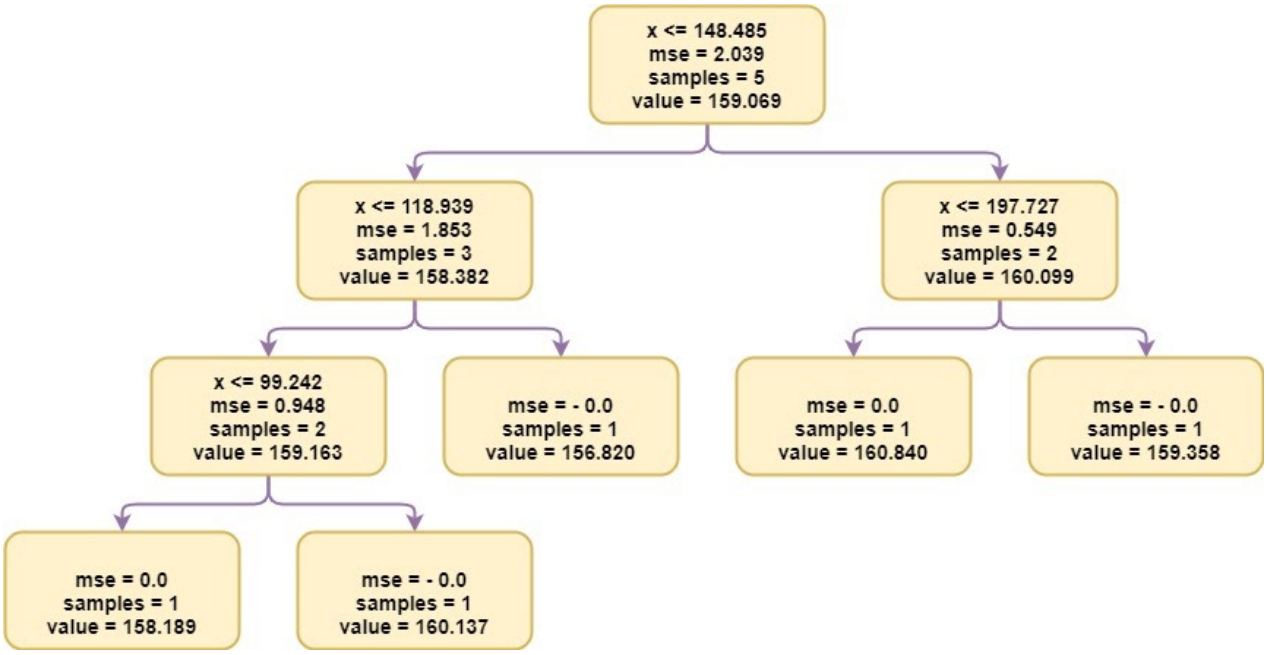
Decision Tree for classification uses entropy, gini index, etc. as the criterion for the measure of impurity. However, to use a decision tree for regression one needs an impurity metric that is suitable for continuous variables. So, the impurity measurement is defined using the weighted mean squared error (MSE) of the children nodes instead. If N_t is the number of training samples at a node t, D_t is the training subset at node t, y_i is the true target value and y_t is the predicted target value (the sample mean), then MSE is given by,

$$MSE(t) = \frac{1}{N_t} \sum_{i=1:i \in D_t}^{N_t} \left(y_i - y_t \right)^2$$

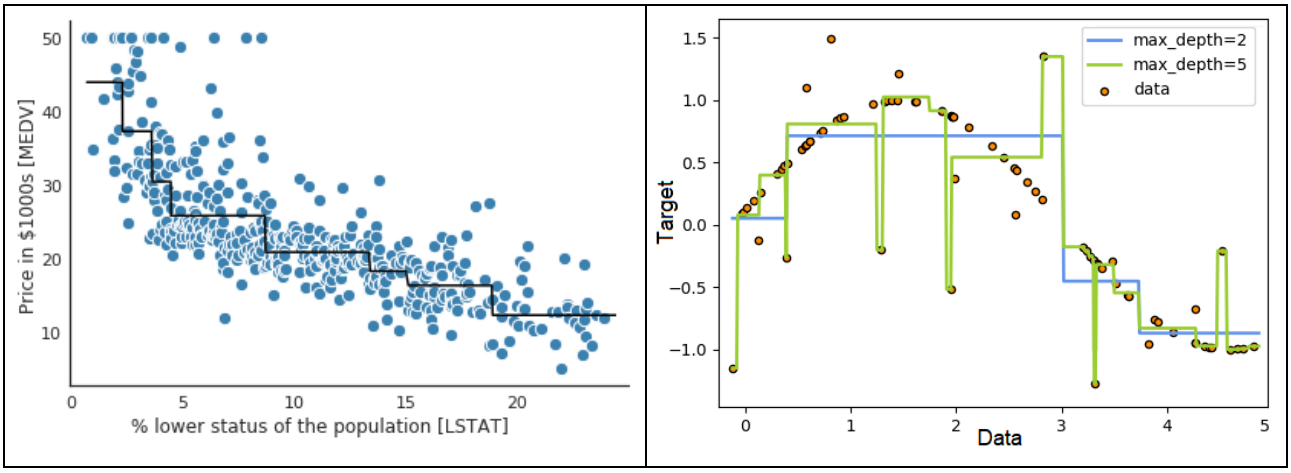
$where,$

$$y_t = \frac{1}{N_t} \sum_{i=1:i \in D_t}^{N_t} y_i$$

The following figure shows an example of Decision Tree Regression with a random sample set having five data points.



The following figure shows examples of predictions of some Decision Tree Regression models.



Python Code - Decision Tree Regression

Train Model

```
>> from sklearn.tree import DecisionTreeRegressor
>> model = DecisionTreeRegressor(max_depth=2)
>> model.fit(X_train, y_train)
```

Predict

```
>> y_pred = model.predict(X_test)
```

Evaluate Model

```
>> from sklearn.metrics import mean_squared_error
>> rsme = np.sqrt(mean_squared_error(y_test, y_pred))
```

Random Forest Regression

The Random Forest Regression is a meta estimator that fits a number of decision tree regressors on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

Python Code - Random Forest Regression

Train Model

```
>> from sklearn.ensemble import RandomForestRegressor
>> model = RandomForestRegressor(n_estimators=10, max_depth=2, random_state=0)
>> model.fit(X_train, y_train)
```

Predict

```
>> y_pred = model.predict(X_test)
```

Evaluate Model

```
>> from sklearn.metrics import mean_squared_error
>> rsme = np.sqrt(mean_squared_error(y_test, y_pred))
```

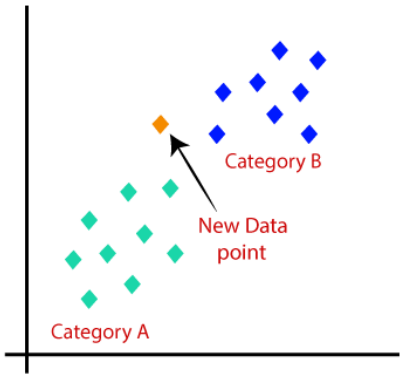
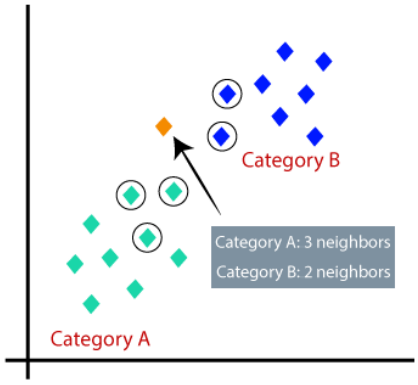
K-NEAREST NEIGHBOR (KNN)

4.5.1. K-NEAREST NEIGHBOR (KNN)

K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection. It is commonly used because of its ease of interpretation and low calculation time. It is widely disposable in real-life scenarios since it is non-parametric, i.e. it does not make any underlying assumptions about the distribution of data

KNN Algorithm

The KNN algorithm works on the principle that similar things exist in close proximity. In other words, *"Birds of a feather flock together"*. Suppose, there is an image of a creature that looks similar to a cat and a dog, but the aim is to find whether it is a cat or dog. In order to identify the creature, one has to compare the similarities of the given creature with the existing details about cats and dogs. Finally, based on the most similar features one can tag the creature either as a cat or a dog. Similarly the KNN algorithm hinges on this idea of similarity (also known as distance, proximity, or closeness). The K-NN algorithm assumes the similarity between the new case/data and available cases and puts the new case into the category that is most similar to the available categories. It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset. Consider the following figure showing two categories A and B, with a new data point x needing to be categorized correctly to one of A or B.

	
In order to classify the data point x, the five nearest neighbours are considered having the minimum distance from x.	As can be seen three of the nearest neighbors are from category A, hence the data point x must belong to category A.

Following is the algorithm for the same.

1. The value of K is initialized.

2. For each training data point x_i, the distance d_i from the test data point is computed. The Euclidean distance being the most popular method can be used as the distance metric. The other metrics that can be used are Chebyshev, cosine, etc. The Euclidean distance between two data points x and y is given by,

$$d(x,y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_n - y_n)^2}$$

3. The top K number of nearest neighbors to the test data point are chosen and their corresponding labels gathered.

4. The test data point is assigned to the mode of the K labels for a classification problem and to the mean of the K labels for a regression problem.

5. The above steps starting from $Step-2$ to $Step-4$ is repeated for all the test data points.

Choosing Number of Neighbors 'K'

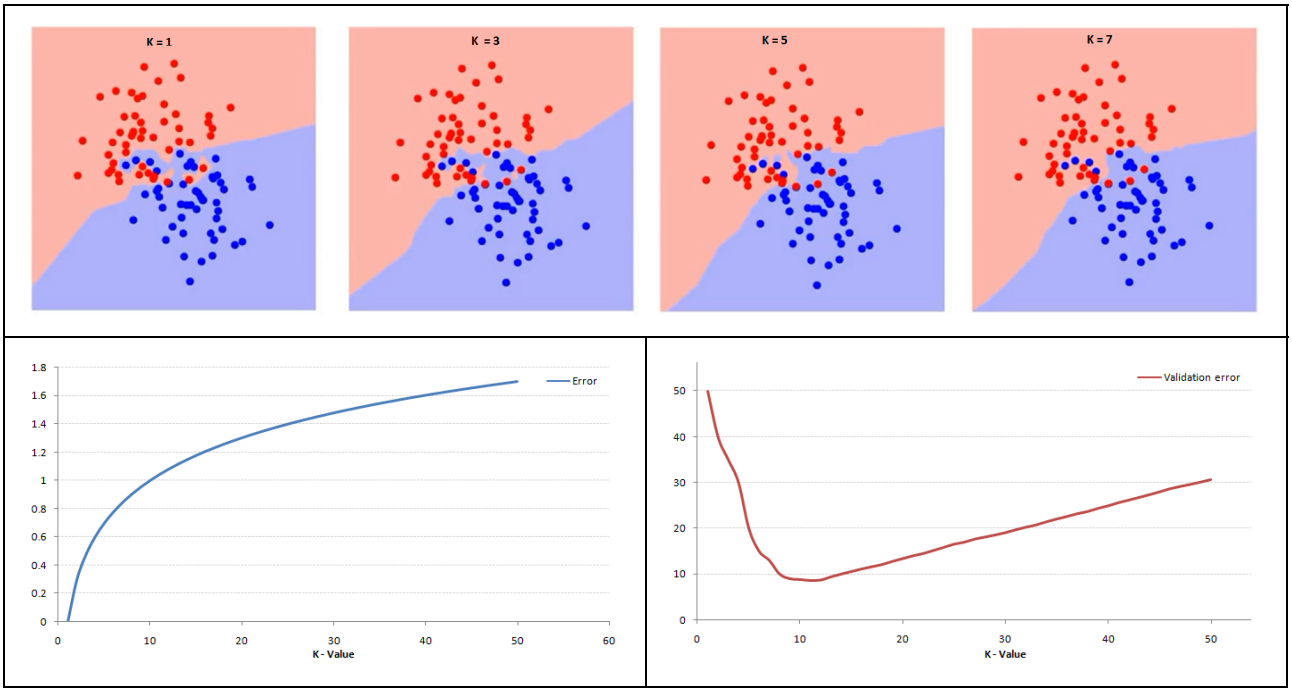
There is no particular way to determine the best value for K, so one needs to run the KNN algorithm several times with different values of K and choose the one that reduces the number of errors one encounters while maintaining the algorithm's ability to accurately make predictions on unseen data. However, the most preferred value for K is 5. Following are some of the things that needs to be taken care of while choosing an optimal value of K.

1. As the value of K is decreased to 1, the predictions become less stable. For example if the test data point is surrounded by several reds and one green (with the green being the single nearest neighbor), then KNN would incorrectly predict that the query point is green even though it is most likely red. A very low value for K such as $K=1$ or $K=2$, can be noisy and lead to the effects of outliers in the model.

2. As the value of K is increased, the predictions become more stable due to majority voting or averaging, and thus, KNN is more likely to make more accurate predictions (up to a certain point). Eventually, for higher values of K one witnesses an increase in the number of errors. It is at this point one can deduce that the value of K has been pushed too far. Large values for K are good, but it may find some difficulties

3. In cases where a majority vote is being taken (such as picking the mode in a classification problem) among labels, usually K is taken as an odd number to have a tiebreaker.

The following figure gives an idea about the same.



Advantages of KNN

1. It is simple and easy to implement as there is no need to build a model, tune several parameters, or make additional assumptions.

2. It is versatile as it can be used for classification, regression and search.

3. It is robust to the noisy training data and is more effective if the training data is large.

Disadvantages of KNN

1. It is always required to determine the value of K which might be complex some time.

2. It gets significantly slower as the number of examples and/or predictors/independent variables increase. The computation cost is high because of calculating the distance between the data points for all the training samples.

Python Code - K-Nearest Neighbor

Train Model

```
>> from sklearn.neighbors import KNeighborsClassifier
>> model = KNeighborsClassifier(n_neighbors=5)
>> model.fit(X_train, y_train)
```

Predict

```
>> y_pred = model.predict(X_test)
```

Evaluate Model

```
>> from sklearn.metrics import confusion_matrix
>> cm = confusion_matrix(mean_squared_error(y_test, y_pred))
```

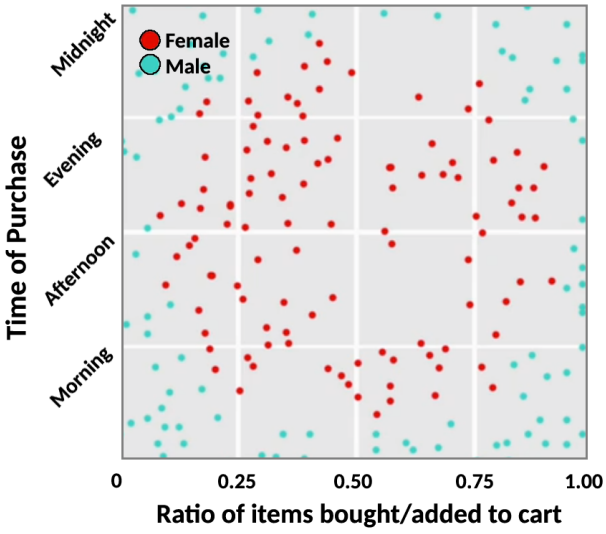
CONSIDERATIONS FOR MODEL SELECTION

4.5.1. CONSIDERATIONS FOR MODEL SELECTION

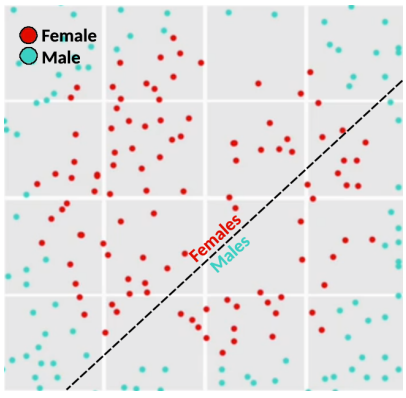
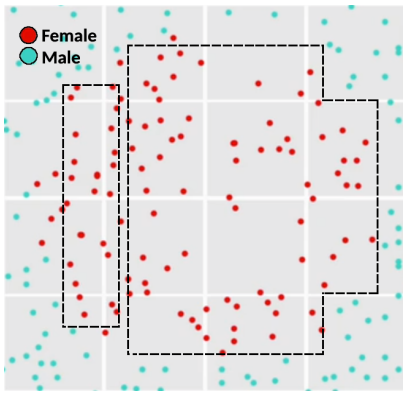
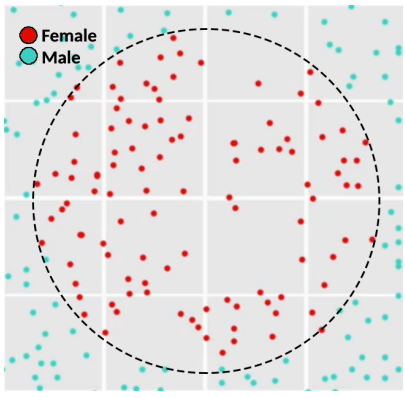
When it comes to the selection of a suitable model, there are a lot of algorithms to choose from. One could always apply all the models and then compare their results. But it is neither always feasible to apply all the models nor is there enough time to try all the available options. Thus, there needs to be some guiding principles behind the choice of models rather than to just use the hit-and-trial approach.

Comparing Different Machine Learning Models

The business problem being considered here is to determine the gender of each online consumer and the age group to which they belong for an ecommerce company. The problem is being solved using Logistic Regression, Decision Trees and SVM. The following figure shows the distribution of the consumer.



The following figures show the performance of the various models used over the data.

		
Logistic Regression : The diagonal line is clearly not able to differentiate the two classes, with a lot of misclassifications.	**Decision Trees/Random Forests : It does better differentiation, but can potentially overfit with increase in the trees.**	**SVM : It does a really good job of differentiating the two classes almost perfectly.**

The following gives the comparison between the three algorithms.

Advantages	Disadvantages
Logistic Regression	
It has a widespread industry use and can be efficiently implemented across different tools. It offers convenient probability scores for outputs. It addresses the issue of multicollinearity in the data using regularisation.	It does not perform well when the features space is too large or when there are a lot of categorical variables in the data. It requires the nonlinear features to be transformed to linear features for efficiency. It relies on entire data i.e. even a small change in the data, changes the model.
Decision Trees	
It is easy to interpret because of the intuitive decision rules. It can handle nonlinear features well and also takes into account the interaction between the variables.	It is highly biased towards the training set and overfits it more often than not. It does not have a meaningful probabilistic output score as the output. Splitting with multiple linear decision boundaries is not always efficient. It is not possible to predict beyond the range of the response variable in the training data in a regression problem.
Random Forests	
It gives a good estimate of model performance on unseen data using the OOB error. It does not require pruning of trees and hardly ever overfits the training data. It is not affected by outliers because of the aggregation strategy.	It too has the problem of not predicting beyond the range of the response variable (being derived from trees). It often does not predict the extreme values because of the aggregation strategy.
Support Vector Machines	
It can handle large feature space. It can handle nonlinear feature interaction. It does not rely on the entire dimensionality of the data for the transformation.	It is not efficient in terms of computational cost when the number of observations is large. It is tricky and time-consuming to find the appropriate kernel for a given data.

End-to-End Modelling

One can easily get overwhelmed by the choices of the algorithms available for classification. The following steps provide a more general rule followed while going about modelling data.

1. Modelling is usually started off with Logistic Regression. This serves two very important purposes,
    a. It acts as a baseline (benchmark) model.
    b. It gives an idea about the important variables.

2. Next, the Decision Trees are used for modelling and the performances are compared. While building a Decision Tree, one should choose the appropriate method, CART for predicting and CHAID for a driver analysis. If the performance is not up to the standards, then the Decision Trees are used to model again, but with only the important variables identified in the Logistic Regression model and re-evaluated.

3. Finally, if the performance is still not to the mark, then one can go for more complex models such as Random Forests and SVM's, keeping the time and resource constraints in mind. These models being computationally expensive and time consuming are usually avoided or always used at last.

The following flowchart gives the steps the end to end modelling.

5. CLUSTERING

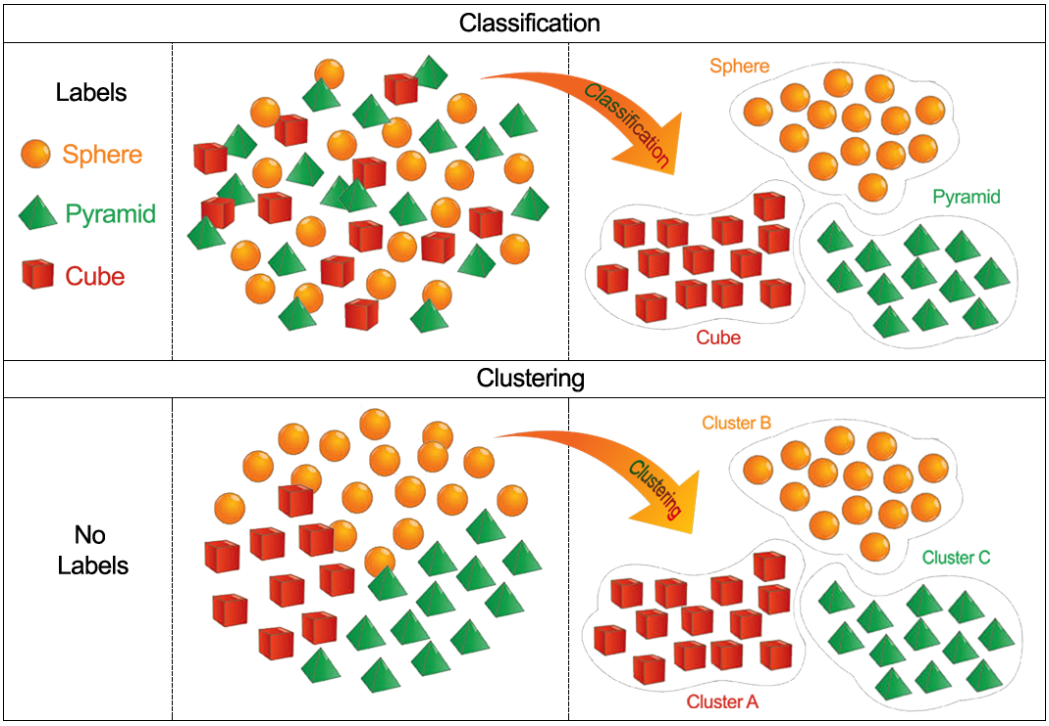
CLUSTERING

5.1.1. CLUSTERING

All the algorithms discussed till now (such as Linear Regression, Logistic Regression, Naive Bayes, SVM, Decision Tree etc.), belong to the Supervised machine learning algorithms, which make use of labelled data to make predictions. Next to be discussed are Unsupervised machine learning algorithms. In Unsupervised learning, prediction is of no importance as there are no target or outcome variables. Rather, the objective is to discover interesting patterns in the data. Clustering is an example of Unsupervised machine learning technique. It is used to place the data elements into related groups without any prior knowledge of the group definitions.

Clustering

Clustering refers to a very broad set of techniques for finding subgroups or segments in a data set. Clustering techniques use the raw data to form clusters based on common factors among various data points. The following figure shows the difference between clustering and classification.



For successful segmentation,

1) The segments formed need to be stable (i.e. the same data point should not fall under different segments upon segmenting the data on the same criteria).
2) The segments should have intra-segment homogeneity (i.e. the behaviour in a segment needs to be similar).
3) The segments should have inter-segment heterogeneity (i.e. the behaviour between segments needs to be different).

There are mainly three types of segmentation used for customer segmentation.

1) Behavioural segmentation : Segmentation is based on the actual behavioural patterns displayed by the person.
2) Attitudinal segmentation : Segmentation is based on the beliefs or intentions of the person, which may not translate into similar action.

3) Demographic segmentation : Segmentation is based on the person's profile and uses information such as age, gender, residence locality, income, etc.

Behavioural Segmentation

The commonly used customer behavioural segmentation techniques are as follows.

RFM	RPI	CDJ
R : Recency (Time since last purchase/visit) F : Frequency (Total number of purchases/visits) M : Monetary (Total revenue generated)	R : Relationship (Past interaction) P : Persona (Type of customer) I : Intent (Intention at the time of purchase)	CDJ : Consumer Decision Journey Based on the customer's life journey with the product.

5.1.2. K-MEANS CLUSTERING

K-means clustering is a simple yet elegant approach for partitioning a data set into K distinct, non-overlapping clusters. In order to perform K-means clustering, one must first specify the desired number of clusters K and then the K-means algorithm assigns each observation to exactly one of the K clusters.

K-Means Algorithm

For a dataset where,

N is the total no of observations

K is the total number of desired clusters

C_k denotes the set containing the observations in k^{th} cluster

The following algorithm is applied to create K clusters.

1. Initialisation : The cluster centers for each of the K clusters are randomly picked. These can either be from any of the N observations or totally a different point.

2. Assignment : Each observation n is assigned to the cluster whose cluster center is the closest to it. The closest cluster center is found using the squared Euclidean distance.
   The equation for the assignment step is as follows.

$$C_k = Argmin \left\{ \sum_{k=1}^{K} \sum_{i=1}^{N} \left(x_i - \mu_k \right)^2 \right\}$$

   The observations are to be assigned in such a way that they satisfy the following conditions.

   a. $C_1 \cup C_2 \cup \ldots C_k = \{1, 2, \ldots, N\}$ i.e. each observation belongs to at least one of the K clusters.

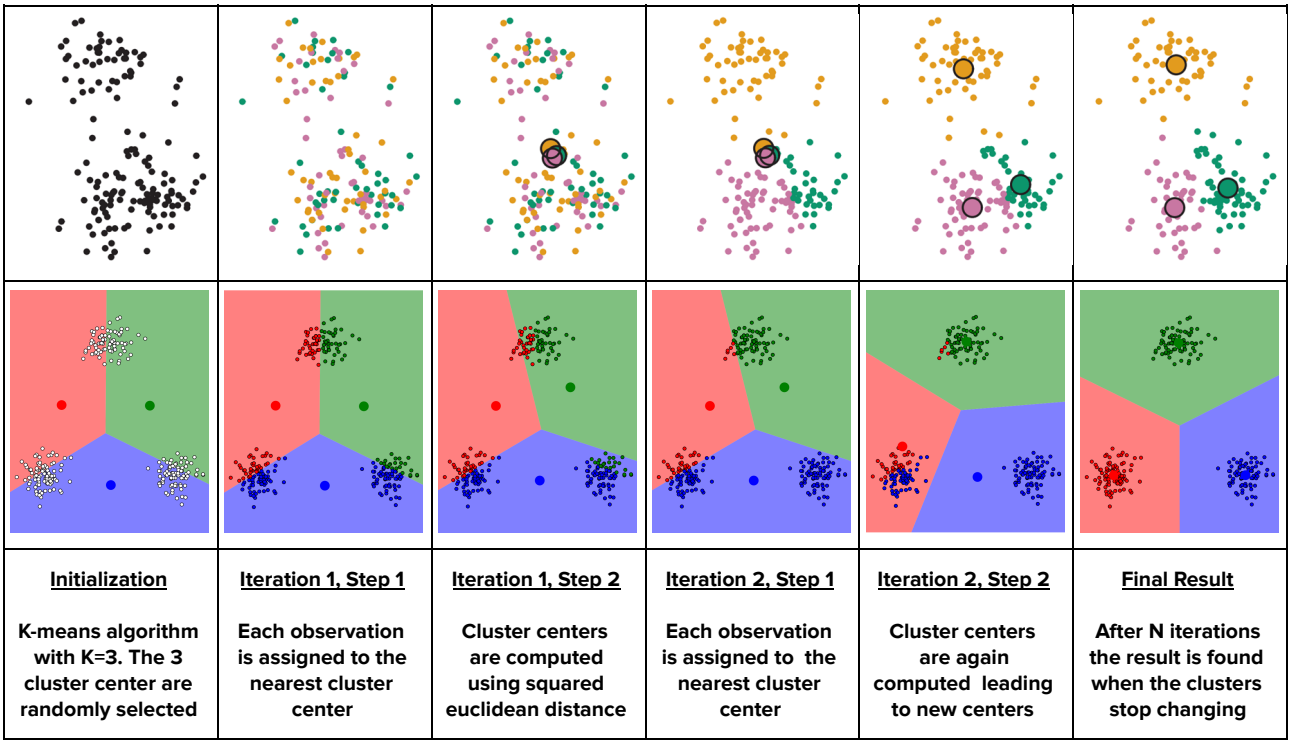
   b. $C_k \cap C_{k'} = \varphi \; where \; k \neq k'$ i.e. no observation belongs to more than one cluster.

3. Optimisation : For each of the K clusters, the cluster center is computed such that the k^{th} cluster center is the vector of the p feature means for the observations in the k^{th} cluster. The equation for the optimisation step is as follows.

$$\mu_k = \frac{1}{|C_k|} \sum_{i \in C_k} x_i$$

4. Iteration : The process of assignment and optimisation is repeated until there is no change in the clusters or possibly until the algorithm converges.

The following figures are a graphical representation of the K-Means algorithm.



Initialization	Iteration 1, Step 1	Iteration 1, Step 2	Iteration 2, Step 1	Iteration 2, Step 2	Final Result
K-means algorithm with K=3. The 3 cluster center are randomly selected	Each observation is assigned to the nearest cluster center	Cluster centers are computed using squared euclidean distance	Each observation is assigned to the nearest cluster center	Cluster centers are again computed leading to new centers	After N iterations the result is found when the clusters stop changing

Cost Function

In the K-Means algorithm it is known that the centroids being computed are the cluster means for each feature and are the constants that minimize the sum of squared deviations. Added to this, the reallocation of the observations help in improving the above result until the result no longer changes (i.e optimum point is reached). Thus, giving the cost function.

$$\frac{Min}{J(C_1, C_2, \ldots C_k)} \quad where,\ J(C_1, C_2, \ldots C_k) = \sum_{k=1}^{K} \sum_{i \in C_k} (x_i - \mu_k)^2$$

Optimizing Cost Function

The K-Means algorithm repeatedly minimize the function concerning cluster assignment C_k given μ and then minimize the function concerning μ given the cluster assignment C_k using a coordinate descent method. The K-Means cost function is usually a non convex function (i.e. local minima is not equal to the global minima). So, the coordinate descent is not guaranteed to converge to the global

minimum and can rather converge to a local minima. Thus, the final results can be highly dependent on the initial cluster assignment of each observation. For this reason, it is important to run the algorithm multiple times with different initial configurations to check for the robustness of the final result.

K-Means++ Algorithm

In the K-Means algorithm during the initialisation step the cluster centers are selected randomly. This may lead to the algorithm converging at a local minima. So, one can choose the cluster centers smartly during the initialisation to achieve the global minima using the K-Means++ algorithm. K-Means++ is just an initialisation procedure for K-Means for picking the initial cluster centers using an algorithm that tries to initialise the cluster centers that are far apart from each other.

The following algorithm is applied to select the cluster centers.

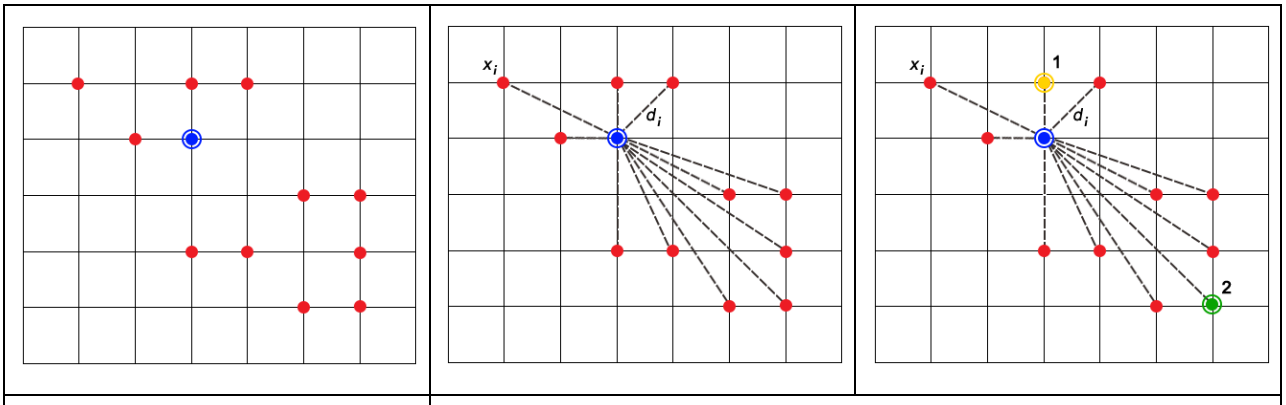
6. One of the data points is randomly chosen as a cluster center.

7. For each data point x_i, the distance d_i is computed. It is the distance between x_i and the nearest center that had already been chosen.

8. The next cluster center is chosen using the weighted probability distribution where a point x is chosen with probability proportional to d_i^2.

9. Steps 2 and 3 are repeated until K cluster centers have been chosen.

The following figure is a graphical representation of the K-Means++ algorithm.



Randomly chosen cluster center out of the 12 data points.	**Data point 1 is the least likely next cluster center as proportional distance d_i^2 is the smallest for this data point.** **Datapoint 2 is the most likely next cluster center as proportional distance d_i^2 is the largest for this data point.**

Practical Considerations for K-Means Clustering

Some of the points to be considered while implementing the K-Means algorithm are as follows.

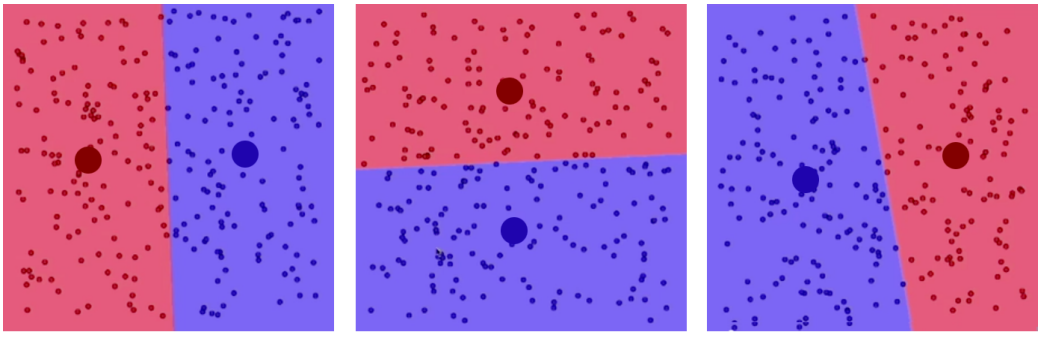
1. The number of clusters (value of K) into which the data points are to be divided has to be predetermined.

2. Since the distance metric used in the clustering process is the squared Euclidean distance between the data points, it is important to ensure that the attributes with a larger range of

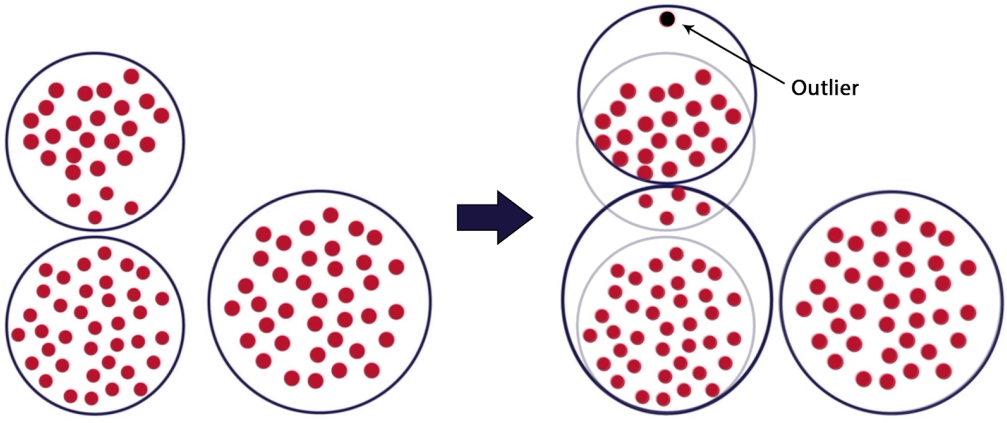
values do not out-weight the attributes with smaller range. Thus, scaling down of all attributes to the same normal scale helps in this process. Scaling helps in making the attributes unit-free and uniform.

3. The K-Means algorithm cannot be used when dealing with categorical data as the concept of distance for categorical data doesn't make much sense.

4. The choice of the initial cluster centres can have an impact on the final cluster formation. The following figure shows three cases (for the same dataset) with different sets of initial cluster centers giving different clusters at the end.



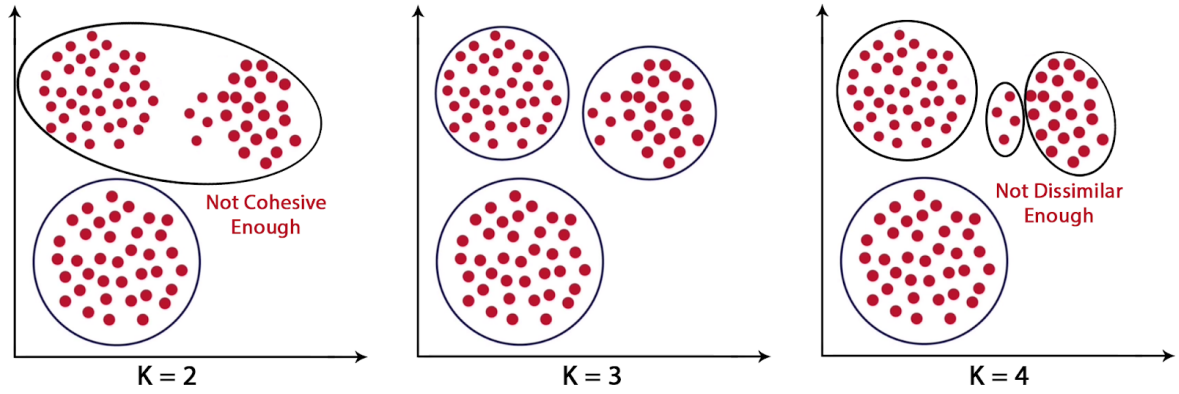
5. Since the K-Means algorithm tries to allocate each of the data points to one of the clusters, outliers have a serious impact on the performance of the algorithm and prevent optimal clustering.



6. There is a chance that the algorithm may not converge in the given number of iterations, so one needs to always check for convergence.

Choosing Number of Clusters 'K'

Choosing the value of K in the K-Means algorithm is of utmost importance as lower values of K can cause the clusters not to be cohesive enough and higher values of K can lead to clusters not dissimilar enough. The same is depicted in the following figure.



There are a number of pointers available that can help in deciding the number of K for a K-means algorithm.

Silhouette Analysis

Silhouette analysis is an approach to choosing the value of K for the K-Means algorithm. The Silhouette Analysis or Silhouette Coefficient is a measure of how similar a data point is to its own cluster (cohesion) compared to other clusters (separation).

For a given dataset where,

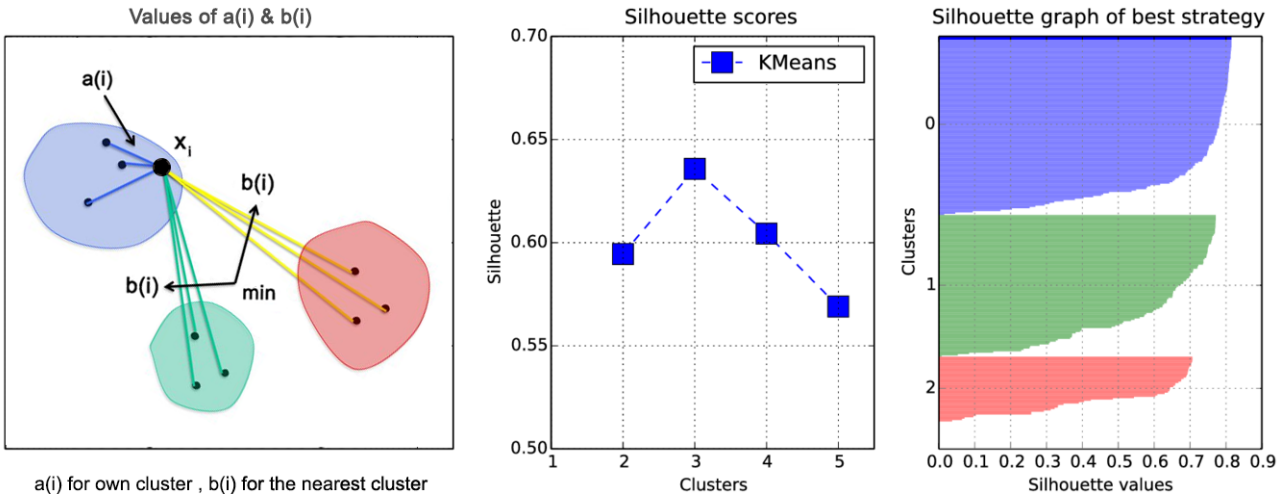
a_i is the average distance from own cluster
b_i is the average distance from the nearest neighbour cluster

Silhouette Value is given by,

$$S(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \quad or \quad S(i) = \begin{cases} 1 - \frac{a(i)}{b(i)}, & if\ a(i) < b(i) \\ 0, & if\ a(i) = b(i) \\ \frac{b(i)}{a(i)}, & if\ a(i) > b(i) \end{cases}$$

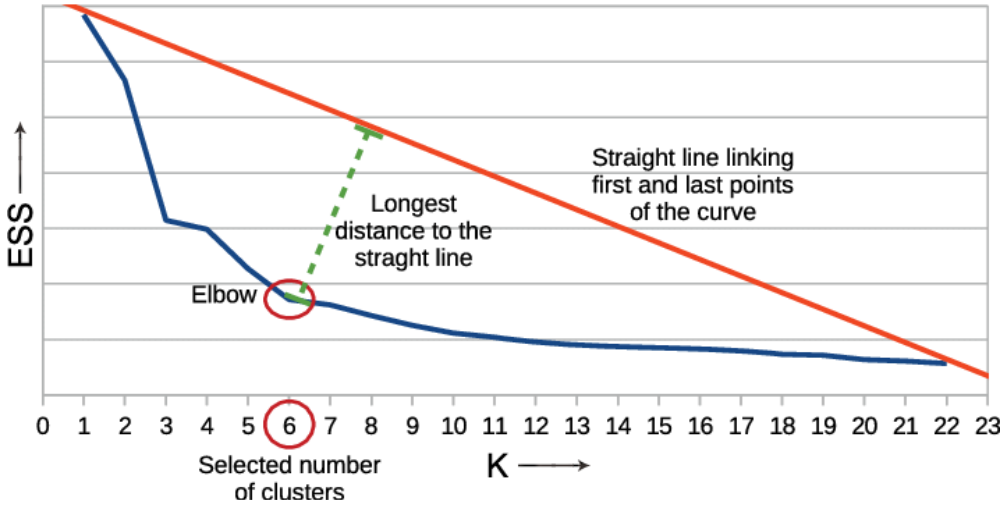
The value of $S(i)$ varies between -1 and 1. For $S(i)$ to be close to 1, it is required for $a(i) \ll b(i)$. As $a(i)$ is a measure of dissimilarity of x_i with its own cluster, a small value of $a(i)$ means it is well matched. Furthermore, a large value of $b(i)$ implies that x_i is badly matched to its neighbouring cluster. Thus, $S(i)$ values close to 1 means that the data is appropriately clustered. If $S(i)$ is close to -1, it implies that x_i would be more appropriate if it were clustered in its neighbouring cluster. An $S(i)$ value close to 0 means that x_i is on the border of two natural clusters.

The average $S(i)$ over all the points of a cluster measures how tightly grouped all the points in the cluster are. Thus the average $S(i)$ over all data of the entire dataset is a measure of how appropriately the data has been clustered. If there are too many or too few clusters, some of the clusters will typically display much narrower silhouettes than the rest. Thus silhouette plots and averages are used to determine the natural number of clusters within a dataset as shown in the following figure.



Elbow Curve Method

The Elbow method is another approach for finding the appropriate number of clusters in a dataset by interpreting and validating the consistency within a cluster. The idea of the elbow method is to run K-Means clustering on the dataset for a range of values of K while calculating the Sum of Squared Errors (SSE) for each value of K. On plotting a line chart of the SSE for each value of K, an arm like plot is formed as shown in the following figure. The elbow on the arm is the value of K, that is the best.



The intention is to always have a small SSE, but as the SSE tends to decrease towards 0 with the increase in K (the SSE is 0 when K is equal to the number of data points in the dataset, because then each data point is its own cluster, and there is no error between it and the center of its cluster). Thus, the goal is to choose a small value of K which still has a low SSE, and the elbow usually represents the point from where the returns start diminishing with increasing values of K. However, the elbow method doesn't always work well, especially if the data is not very clustered. In such cases one can try the Silhouette analysis, or even reevaluate whether clustering is the right thing to do.

Cluster Tendency

Before applying the clustering algorithm to any given dataset, it is important to check whether the given data has some meaningful clusters or not (i.e the data is not random). This process of evaluating the data for its feasibility for clustering is known as the Clustering Tendency. The clustering algorithms will return K clusters even when the datasets do not have any meaningful

clusters. So, one should always check the clustering tendency before proceeding for clustering. To check the cluster tendency, one can use the Hopkins test.

Hopkins Test

The Hopkins test is used to assess the clustering tendency of a data set by measuring the probability that a given data set is generated by a uniform data distribution. In other words, it tests the spatial randomness of the data.

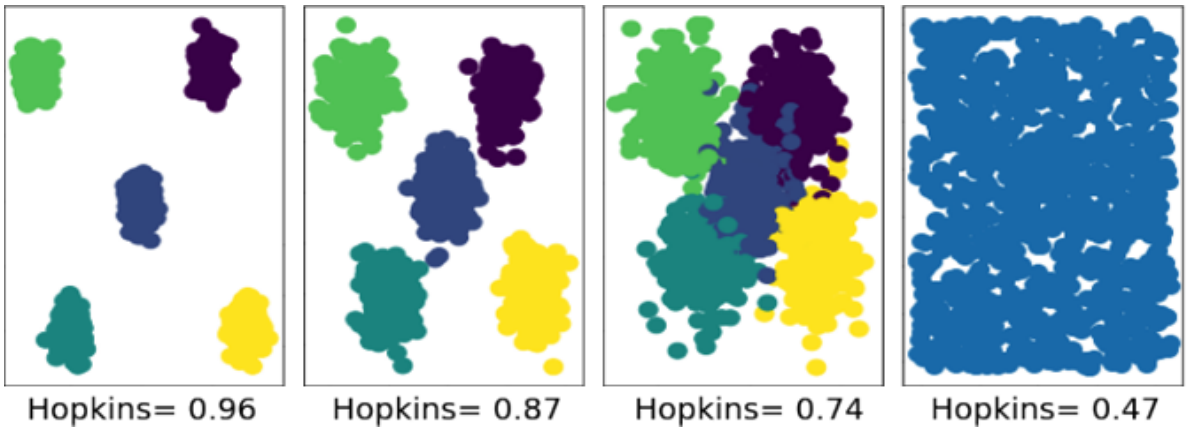
For a dataset where,

X is a set of n data points
p_i is a member of a random sample (without replacement) of $m \ll n$ data points.
x_i is the distance of $p_i \in X$, from its nearest neighbour in X
Y is a set of m uniformly randomly distributed data points (similar to X)
q_i is a member of a data set Y
y_i is the distance of $q_i \in Y$, from its nearest neighbour in X

Hopkins Test is given by,

$$H = \frac{\sum_{i=1}^{m} y_i}{\sum_{i=1}^{m} y_i + \sum_{i=1}^{m} x_i}$$

The value of H varies between 0 and 1. If X were uniformly distributed, then Σx_i and Σy_i would be close to each other, and thus H would be about 0.5. However, if X were to have clusters, then the distances for artificial points Σy_i would be substantially larger than for the real ones Σx_i, thus increasing the value of H. A value for H between 0.01 to 0.3 indicates the data is regularly spaced, around 0.5 indicates the data is random and more than 0.7 indicates a clustering tendency at 90% confidence level. The following figure shows the clustering tendency for various Hopkins factor.



Python Code - K-Means Clustering

```
Behavioural Segmentation (RFM)
>> data = RFM[['recency_col','frequency_col','monetary_col']]
>> Q1 = data['rfm_col'].quantile(0.25)
>> Q3 = data['rfm_col'].quantile(0.75)
```

```
>> IQR = Q3 - Q1
>> data = data[(data['rfm_col'] >= Q1-1.5*IQR) &
               (data['rfm_col'] <= Q3+1.5*IQR)]
>> data = pd.DataFrame(standard_scaler.fit_transform(data))
>> data.columns = ['recency_col','frequency_col','monetary_col']
```

Cluster Tendency Analysis using Hopkins Statistics

```
>> from math import isnan
>> from random import sample
>> from numpy.random import uniform
>> from sklearn.neighbors import NearestNeighbors
>> def hopkins(X):
>>     d = X.shape[1]
>>     n = len(X)
>>     m = int(0.1 * n)
>>     nbrs = NearestNeighbors(n_neighbors=1).fit(X.values)
>>     rand_X = sample(range(0, n, 1), m)
>>     yi = []
>>     xi = []
>>     for j in range(0, m):
>>         y_dist, _ = nbrs.kneighbors(uniform(np.amin(X, axis=0),
                        np.amax(X, axis=0), d).reshape(1, -1), 2,
                        return_distance=True)
>>         yi.append(y_dist[0][1])
>>         x_dist, _ = nbrs.kneighbors(X.iloc[rand_X[j]].values.reshape(1,-1),
                                       2, return_distance=True)
>>         xi.append(x_dist[0][1])
>>     H = sum(yi) / (sum(yi) + sum(xi))
>>     if isnan(H):
>>         print(yi, xi)
>>         H = 0
>>     return H
>> hopkins(data)
```

Choosing K using Silhouette Analysis

```
>> from sklearn.cluster import KMeans
>> from sklearn.metrics import silhouette_score
>> sse = []
>> for k in range(min_K, max_K):
>>     kmeans = KMeans(n_clusters=k).fit(data)
>>     sse.append([k, silhouette_score(data, kmeans.labels_)])
>> plt.plot(pd.DataFrame(sse)[0], pd.DataFrame(sse)[1]);
```

Choosing K using Elbow Curve

```
>> ssd = []
>> for k in range(min_K, max_K):
>>     kmeans = KMeans(n_clusters = k, max_iter=50)
>>     kmeans.fit(data)
>>     ssd.append(kmeans.inertia_)
>> plt.plot(ssd)
```

Build Model

```
>> model = KMeans(n_clusters = best_k, max_iter=50)
>> model.fit(data)
```

Analyse Model

```
>> data.index = pd.RangeIndex(len(data.index))
>> data = pd.concat([data, pd.Series(model.labels_)], axis=1)
>> data.columns = ['id','recency_col','frequency_col','monetary_col','cluster_id']
>> data['recency_col'] = data['recency_col'].dt.days
>> cluster_R = pd.DataFrame(data.groupby(['cluster_id']).recency_col.mean())
```

```
>> cluster_F = pd.DataFrame(data.groupby(['cluster_id']).frequency_col.mean())
>> cluster_M = pd.DataFrame(data.groupby(['cluster_id']).monetary_col.mean())
>> data_plot = pd.concat([pd.Series(range(k)),cluster_R,cluster_F, cluster_M],axis=1)
>> data_plot.columns = ['cluster_id','recency_col','frequency_col','monetary_col']
>> sns.barplot(x=data_plot['cluster_id'], y=data_plot['rfm_col'])
```

5.1.3. HIERARCHICAL CLUSTERING

Hierarchical clustering is yet another algorithm for unsupervised clustering. Here, instead of pre-defining the number of clusters, one has to first visually describe the similarity or dissimilarity between the different data points and then decide the appropriate number of clusters on the basis of these similarities or dissimilarities. The output of the hierarchical clustering algorithm resembles an inverted tree-shaped structure, called the dendrogram.

Hierarchical Clustering Algorithm

In hierarchical clustering, the data is not partitioned into a particular cluster in a single step. Instead, a series of partitions/merges take place, which may run from a single cluster containing all objects to n clusters with each cluster containing a single object.

The following algorithm is applied to create hierarchical clusters for a dataset with n items.

1. Initialisation : The $n \times n$ distance (similarity) matrix is calculated, which gives the distance of each data point from the other.

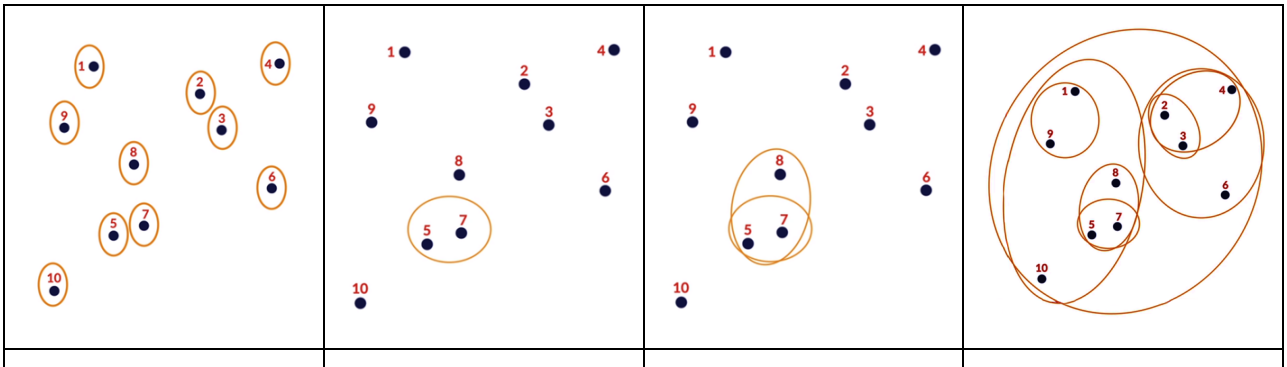
2. Assignment : The clustering is started by assigning each item to its own cluster, such that if there are n items, then the total number of clusters created is also n, with each cluster containing just one item.

3. Optimisation : The closest (most similar) pair of clusters is searched for using the $n \times n$ distance matrix. This pair is then merged into a single cluster, so that now there is one less cluster (i.e $n-1$).

4. Computation : The distances (similarities) between the new cluster and each of the old clusters are computed.

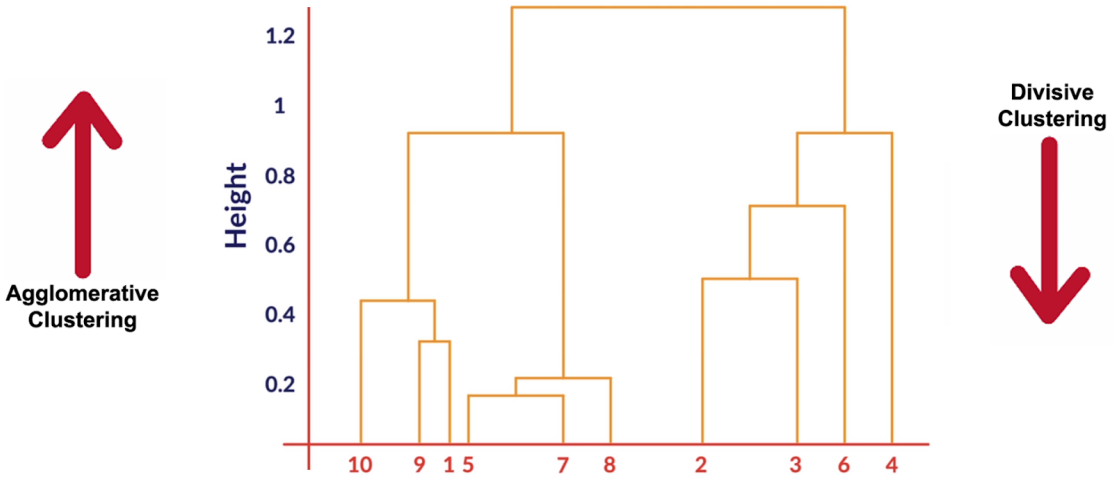
5. Iteration : The process of optimisation and computation are repeated until all the items are clustered into a single cluster of size n.

The following figures are a graphical representation of the hierarchical clustering algorithm.



1. Assignment	2. Optimisation	3. Computation	4. Final Result

Thus, at the end a dendrogram is formed, that shows which data points are grouped together in which cluster at what distance. One can look at what stage an element is joining a cluster and hence see how similar or dissimilar it is to the rest of the cluster (if it joins at the higher height, it is quite different from the rest of the group). The following figure shows a dendrogram that starts with all the data points as separate clusters and indicates at what level of dissimilarity any two clusters were joined.



The y-axis of the dendrogram is some measure of dissimilarity or distance at which clusters join. In the dendrogram given, samples 5 and 7 are the most similar and so join to form the first cluster, followed by samples 1 and 10. The last two clusters to fuse together to form the final single cluster are $10-9-1-5-7-8$ and $2-3-6-4$. There are two types of hierarchical clustering algorithms.

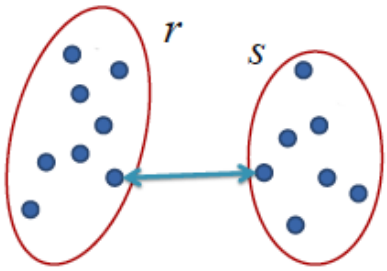
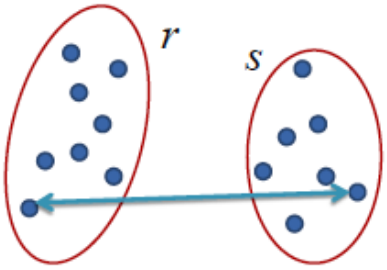
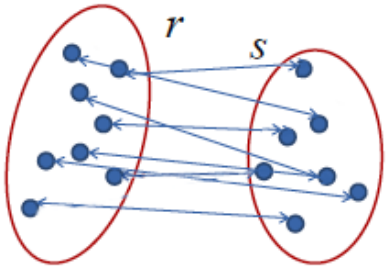
1. Agglomerative : It is a bottom-up algorithm which starts with n distinct clusters and iteratively merge (or agglomerate) pairs of clusters (which have the smallest dissimilarity) until all clusters have been merged into a single cluster.

2. Divisive : It is a top-down algorithm which proceeds by splitting (or dividing) a single cluster (resulting in the biggest dissimilarity) recursively until n distinct clusters are reached. Even though divisive algorithms are more complex than agglomerative algorithms, they produce more accurate hierarchies in certain scenarios. The bottom-up algorithms make clustering decisions based on local patterns without initially taking into account the global distribution, which cannot be undone. Whereas, top-down algorithms benefit from having the complete information about the global distribution while partitioning.

Linkages

In a clustering algorithm, before any clustering is performed, it is required to determine the proximity matrix (displaying the distance between each cluster) using a distance function. This proximity or linkage between any two clusters can be measured by the following three distance functions.

Single Linkage	Complete Linkage	Average Linkage
The distance between two clusters is defined as the shortest distance between two points in each cluster.	The distance between two clusters is defined as the longest distance between two points in each cluster.	The distance between two clusters is defined as the average distance between each point in one cluster to

		every point in the other cluster.
		
$L(r,s) = min\left(D\left(x_{ri}, x_{sj}\right)\right)$	$L(r,s) = max\left(D\left(x_{ri}, x_{sj}\right)\right)$	$L(r,s) = \frac{1}{n_r n_s}\sum_{i=1}^{n_r}\sum_{j=1}^{n_s} D\left(x_{ri}, x_{sj}\right)$
It suffers from chaining. In order to merge two groups, only one pair of points is required to be near to each other, irrespective of the location of all other points. Therefore clusters can be too spread out, and not compact enough.	It suffers from crowding. Its score is based on the worst case dissimilarity between pairs, making a point to be closer to points in other clusters than to points in its own cluster. Thus, clusters are compact, but not far enough apart.	It tries to strike a balance between the two, by using mean pairwise dissimilarity. So the clusters tend to be relatively compact as well as relatively far apart.

The type of linkage to be used is usually decided after having a look at the data. One convenient way to decide is by looking at how the dendrogram looks. Usually, single linkage types produce dendrograms which are not structured properly, whereas complete or average linkages produce dendrograms which have a proper tree-like structure.

Dendrogram

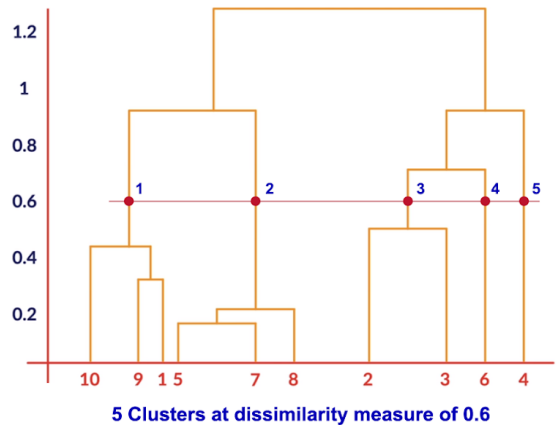
Determining the number of groups in a cluster analysis is usually the primary goal. To determine the cutting section, various methods can be used.

1. By means of observation or experience based on the knowledge of the researcher. For example, based on appearance differences, clusters can be divided into groups.

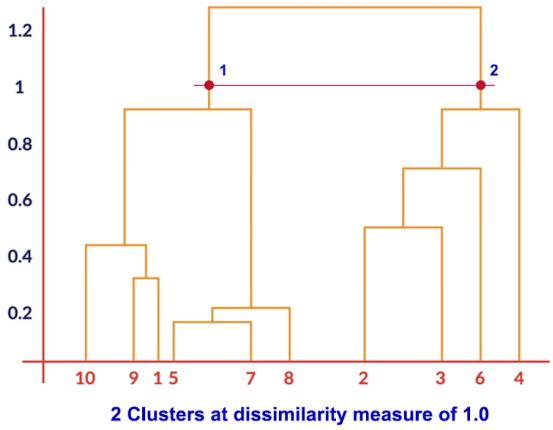
2. Using statistical conventions, the dendrogram can be cut where the difference is most significant. Another technique is to use the square root of the number of observations. Yet another technique is to use at least 70% of the distance between the two groups.

3. Using the function discrimination and classification based on discrimination function.

Thus, the use of the method for cutting a dendrogram depends on the purpose. Typically, one can look for natural groupings defined by long stems. Once the appropriate level is decided upon, one can obtain the clusters by cutting the dendrogram at that appropriate level. The number of vertical lines intersecting the cutting line represents the number of clusters. The following figures show the various clusters obtained at various levels of dissimilarity.



5 Clusters at dissimilarity measure of 0.6



2 Clusters at dissimilarity measure of 1.0

One of the major advantages of Hierarchical clustering is that the number of clusters K is not required to be pre-defined as is the case in K-Means clustering. The Hierarchical clusterings usually produce better clusters, but are time-consuming and computationally intensive, due to which the K-Means clusterings are generally used for larger datasets.

While performing a clustering activity, it is always advisable to first use the Hierarchical clustering (dendrograms) to visualize the clusters from a business standpoint and determine the right value of K. Then calculate the centroids of these K clusters using the mean of each Hierarchical cluster. And finally use K-Means clustering with K being the number of clusters and the computed centroids as the seeds for these K clusters (helps the algorithm to converge faster than in case of random seed selection).

Python Code - Hierarchical Clustering

```
Behavioural Segmentation (RFM)
>> data = RFM[['recency_col','frequency_col','monetary_col']]
>> Q1 = data['rfm_col'].quantile(0.25)
>> Q3 = data['rfm_col'].quantile(0.75)
>> IQR = Q3 - Q1
>> data = data[(data['rfm_col'] >= Q1-1.5*IQR) &
               (data['rfm_col'] <= Q3+1.5*IQR)]
>> data = pd.DataFrame(standard_scaler.fit_transform(data))
>> data.columns = ['recency_col','frequency_col','monetary_col']

Build Model
>> from scipy.cluster.hierarchy import linkage
>> from scipy.cluster.hierarchy import dendrogram
>> model = linkage(data, method = 'single', metric='euclidean')
# method can be any of the values single, complete or average
>> dendrogram(model)
>> plt.show()

Cut Dendrogram
>> from scipy.cluster.hierarchy import cut_tree
>> clusterCut = pd.Series(cut_tree(model, n_clusters = 5).reshape(-1,))

Analyse Model
>> data = pd.concat([RFM, clusterCut], axis=1)
>> data.columns = ['id','recency_col','frequency_col','monetary_col','cluster_id']
>> data['recency_col'] = data['recency_col'].dt.days
>> cluster_R = pd.DataFrame(data.groupby(['cluster_id']).recency_col.mean())
>> cluster_F = pd.DataFrame(data.groupby(['cluster_id']).frequency_col.mean())
>> cluster_M = pd.DataFrame(data.groupby(['cluster_id']).monetary_col.mean())
>> data_plot = pd.concat([pd.Series(range(k)),cluster_R,cluster_F, cluster_M],axis=1)
```

```
>> data_plot.columns = ['cluster_id','recency_col','frequency_col','monetary_col']
>> sns.barplot(x=data_plot['cluster_id'], y=data_plot['rfm_col'])
```

5.1.4. K-MODES CLUSTERING

The K-Means clustering can handle only numerical or continuous data, but not categorical data, and the reason is that of the difference in the dissimilarity measure the K-Means algorithm uses. In comes the K-modes clustering algorithm based on the K-means paradigm which uses dissimilarity score and modes to form clusters of categorical data.

K-Modes Algorithm

The following algorithm is applied to select the clusters.

1. Initialisation : The cluster centers for each of the K clusters are randomly picked.

2. Assignment : Each observation n is assigned to the cluster whose cluster center is the closest to it. The closest cluster center is found using the dissimilarity score (i.e. quantification of the total mismatches between two objects, the smaller this number, the more similar the two objects) rather than the squared Euclidean distance. The equation for the assignment step is as follows.

$$C_k = Argmin \left\{ \sum_{k=1}^{K} \sum_{i=1}^{N} \delta\left(x_i, K_k\right) \right\} \quad where,\ \delta\ is\ the\ dissimilarity\ function$$

3. Optimisation : For each of the K clusters, the cluster center is computed such that the k^{th} cluster center is the vector of the p feature modes (rather than the means) for the observations in the k^{th} cluster. A mode is a vector of elements that minimizes the dissimilarities between the vector itself and each object of the data.

4. Iteration : The process of assignment and optimisation is repeated until there is no change in the clusters or possibly until the algorithm converges.

Python Code - K-Modes Clustering

Labelling the Categorical Variables

```
>> from sklearn import preprocessing
>> le = preprocessing.LabelEncoder()
>> data = data.apply(le.fit_transform)
```

Build Model

```
>> from kmodes.kmodes import KModes
>> model = KModes(n_clusters=k, init = 'Cao', n_init = 1, verbose=1)
# init can be any of the values Cao, Huang or random
>> clusters = model.fit_predict(data)
>> cluster_centroids_df = pd.DataFrame(model.cluster_centroids_)
>> cluster_centroids_df.columns = data.columns
```

Analyse Model

```
>> data = data.reset_index()
>> clusters_df = pd.DataFrame(clusters)
>> clusters_df.columns = ['cluster_predicted']
>> data = pd.concat([data, clusters_df], axis = 1).reset_index()
>> data = data.drop(['index', 'level_0'], axis = 1)
```

```
>> cluster_k = data[data.cluster_predicted==k]
>> category = pd.DataFrame(cluster_k['category'].value_counts())
>> sns.barplot(x=category.index, y=category['category'])
```

5.1.5. K-PROTOTYPE CLUSTERING

The K-Means clustering can handle only numerical or continuous data. The K-Modes clustering can handle categorical data. But if a data set has both numerical and categorical values, then the K-Prototype clustering comes into the picture. It is a simple combination of K-Means and K-Modes in clustering mixed attributes.

K-Prototype Algorithm

The following algorithm is applied to select the clusters.

1. Initialisation : The cluster centers for each of the K clusters are randomly picked.

2. Assignment : Each observation n is assigned to the cluster whose cluster center is the closest to it. The closest cluster center is found using both the squared Euclidean distance and the dissimilarity score. The equation for the assignment step is as follows.

$$C_k = Argmin \left\{ \sum_{k=1}^{K} \left(\sum_{i=1}^{p} \left(x_i - \mu_k \right)^2 + \gamma \sum_{i=p+1}^{N} \delta\left(x_i, K_k \right) \right) \right\}$$

3. Optimisation : For each of the K clusters, the cluster center is computed such that the k^{th} cluster center is the vector of the p feature modes/means for the observations in the k^{th} cluster.

4. Iteration : The process of assignment and optimisation is repeated until there is no change in the clusters or possibly until the algorithm converges.

Python Code - K-Prototype Clustering

```
Build Model
>> from kmodes.kprototypes import KPrototypes
>> model = KPrototypes(n_clusters=k, init='Cao')
# init can be any of the values Cao, Huang or random
>> clusters = model.fit_predict(data, categorical=['categorical_col'])
>> cluster_centroids_df = pd.DataFrame(model.cluster_centroids_)
>> cluster_centroids_df.columns = data.columns

Analyse Model
>> data['cluster_id'] = clusters
>> data_plot = pd.DataFrame(data['cluster_id'].value_counts())
>> sns.barplot(x=data_plot.index, y=data_plot['cluster_id'])
```

5.1.6. DB SCAN CLUSTERING

DBSCAN is a density-based clustering algorithm that divides a data set into subgroups of high-density regions. DBSCAN groups together the points that are close to each other based on a distance measurement (usually Euclidean distance) and a minimum number of points. It also marks as outliers the points that are in low-density regions. DBSCAN looks for densely packed observations and makes no assumptions about the number or shape of the clusters. The DBSCAN

algorithm is used to find associations and structures in the data that are usually hard to find manually. DBSCAN algorithm requires two parameters,

1. EPS (ε) : EPS is the distance parameter that defines the radius to search for nearby neighbours (imagine each data point having a circle with radius EPS drawn around it). The value of EPS has a significant impact on the results. For smaller values of EPS, decidedly fewer data points will be considered in one cluster, and a large part of the data will not be clustered. The un-clustered data points will be considered as outliers as they do not satisfy the number of points to create a dense region. For larger values of EPS, apparently no real clusters will be formed as all of them will merge in the same cluster. The EPS should be chosen based on the distance of the dataset (using k-distance graph), but in general small EPS values are preferable.

2. MinPoints or MinSamples : Min Samples are the number of minimum points to form a dense region or cluster (i.e if min_samples is 5, then there needs to be at least 5 points to form a dense cluster). Minimum points can be selected from some dimensions (D) in the data set, as a general rule min points $\geq D + 1$.

DBSCAN Algorithm

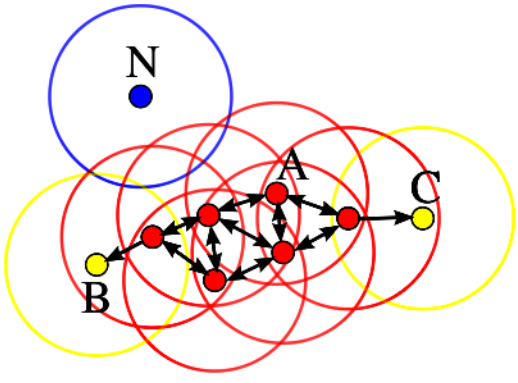
The following algorithm is applied to select the clusters.

1. Initialisation : A random observation n is selected.

2. Assignment : If the observation n has the minimum number of close neighbours, it is considered as a part of the cluster.

3. Grouping : The process of assignment is repeated recursively for all of the n's neighbours, then neighbours neighbour and so on. These are the cluster's core members.

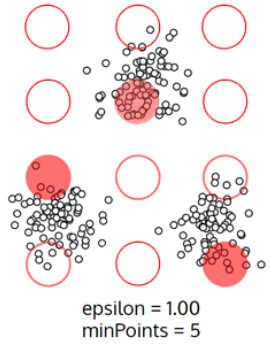
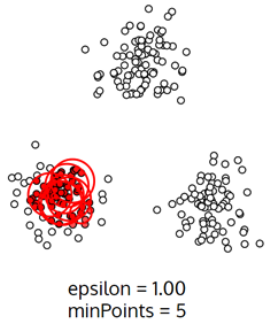
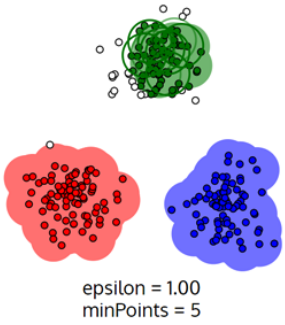
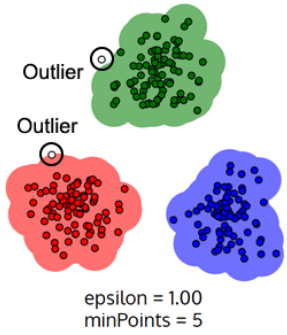
   A point p is a core point if at least the minimum number of points are within the distance ε of it (including p). A point q is directly reachable from p if point q is within distance ε from core point p. A point q is also reachable from p if there is a path $p_1, p_2, \ldots p_n$ with $p_1 = p$ and $p_n = q$, where each p_{i+1} is directly reachable from p_i (i.e. all points on the path must be core points, with the possible exception of q). All points not reachable from any other points are outliers. If p is a core point, then it forms a cluster together with all points (core or non-core) that are reachable from it. Each cluster contains at least one core point where the non-core points can be a part of the cluster forming its edge, since they cannot be used to reach more points. The following figure represents the same.



The minimum number of points is taken as 4. Point A and the other red points are core points, as the area surrounding these points in a radius of ε contain at least 4 points (including the point itself). As all these points are reachable from one another, they form a single cluster. Points B and C are not core points, but are still reachable from A (via other core points), thus they belong to the cluster as well. Point N is an outlier as it is neither a core point nor is reachable directly via other points.

4. Iteration : Once neighbourhood iteration runs out of observations, the process of assignment and grouping is repeated until there is no change in the clusters or possibly until the algorithm converges.

Finally The observations not a part of core are either assigned to a nearby cluster or marked as outliers. The following figures are a graphical representation of the DBSCAN algorithm.

			
1. Initialisation	**2. Assignment**	**3. Grouping**	**4. Final Result**

Python Code - DBSCAN Clustering

```
Build Model
>> from sklearn.cluster import DBSCAN
>> model = DBSCAN(eps=radius, min_samples=n)
>> clusters = model.fit(data)
>> clusters.labels_
```

5.1.7. GAUSSIAN MIXTURE MODEL

In a K-Means algorithm, every data point is assigned to a cluster. This assignment of data points to a cluster can be of two types.

Hard Clustering

In hard clustering, each data point is assigned to any one cluster completely (i.e. there is no consideration of uncertainty in the assignment of a data point). The limitations with hard clustering is that it tends to cluster even the data points in one of the clusters, even if the data point doesn't follow the clustering trend completely. For example while clustering a set of customers into two groups high value and low value customers, one may end up clustering the average value customers to either one of the clusters.

Soft Clustering

In soft clustering, the assignment of the data point to a cluster is based on the probability or likelihood of that data point existing in that cluster. For soft clustering, an algorithm called GMMs or Gaussian Mixture Models are used.

Gaussian Mixture Model

If a model has some hidden, not observable parameters, then one should use GMM. This is because, this algorithm assigns a probability to each point to belong to the certain cluster, instead of

assigning a flag that the point belongs to the certain cluster as in the classical K-Means. Then, GMM produces non-convex clusters, which can be controlled with the variance of the distribution. In fact, K-Means is a special case of GMM, such that the probability of one point to belong to a certain cluster is 1, and all other probabilities are 0, and the variance is 1, which is a reason why K-Means produce only spherical clusters.

Advantages of GMM

The GMM has two major advantages over K-Means as follows,

1. GMM is a lot more flexible in terms of cluster covariance : K-Means is actually a special case of GMM in which each cluster's covariance along all dimensions approaches 0. This implies that a point will get assigned only to the cluster closest to it. With GMM, each cluster can have unconstrained covariance structure (think of rotated or elongated distribution of points in a cluster, instead of spherical as in K-Means). As a result, cluster assignment is much more flexible in GMM than in K-Means.

2. GMM model accommodates mixed membership : Another implication of the covariance structure is that GMM allows for mixed membership of points to clusters. In K-Means, a point belongs to one and only one cluster, whereas in GMM a point belongs to each cluster to a different degree. The degree is based on the probability of the point being generated from each cluster's (multivariate) normal distribution, with cluster center as the distribution's mean and cluster covariance as its covariance. Depending on the task, mixed membership may be more appropriate (e.g. news articles can belong to multiple topic clusters) or not (e.g. organisms can belong to only one species).

GMM or EM (Expectation-Maximisation) Algorithm

The expectation-maximization (EM) algorithm is a way to find maximum-likelihood estimates for model parameters when the data is incomplete, has missing data points, or has unobserved (hidden) latent variables. It works by choosing random values for the missing data points and using those guesses to estimate a second set of data. The new values are used to create a better guess for the first set, and the process continues until the algorithm converges on a fixed point.

For a dataset where,

N is the total no of observations
K is the total number of desired clusters
C_k denotes the set containing the observations in k^{th} cluster
$\overline{\mu_k}$ is the mean of C_k
Σ_k is the covariance of C_k
φ_k is the weightage for C_k such that $\sum_{i=1}^{K} \varphi_i = 1$

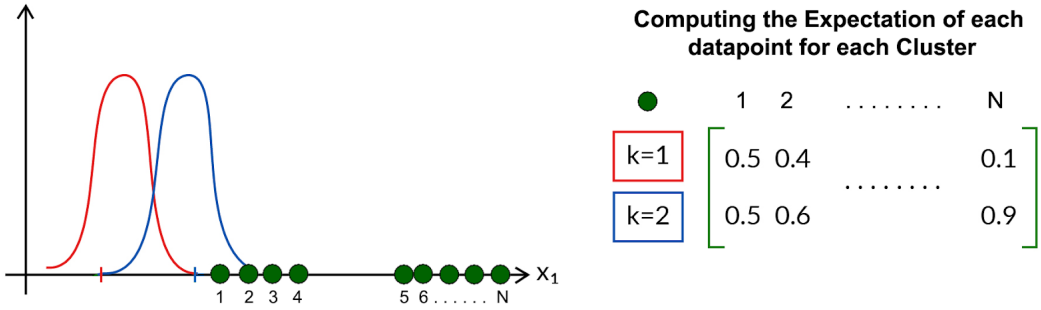
The following algorithm is applied to create K clusters.

1. Initialisation : Initial values are assigned to the parameters μ_k, φ_k and Σ_k by guessing some values.

2. Expectation : Expectations of each data point $x_i \in N$ for each component C_k are calculated for given the model parameters μ_k, φ_k and Σ_k. The probability or expectation that x_i is generated by component C_k is given by,

$$\Upsilon_{ik} = \frac{\varphi_k \, \mathcal{N}\left(\vec{x}_i \,\middle|\, \vec{\mu}_k, \Sigma_k \right)}{\sum_{j=1}^{k} \varphi_j \, \mathcal{N}\left(\vec{x}_j \,\middle|\, \vec{\mu}_j, \Sigma_j \right)}$$

The following figure shows the computation of the expectations.

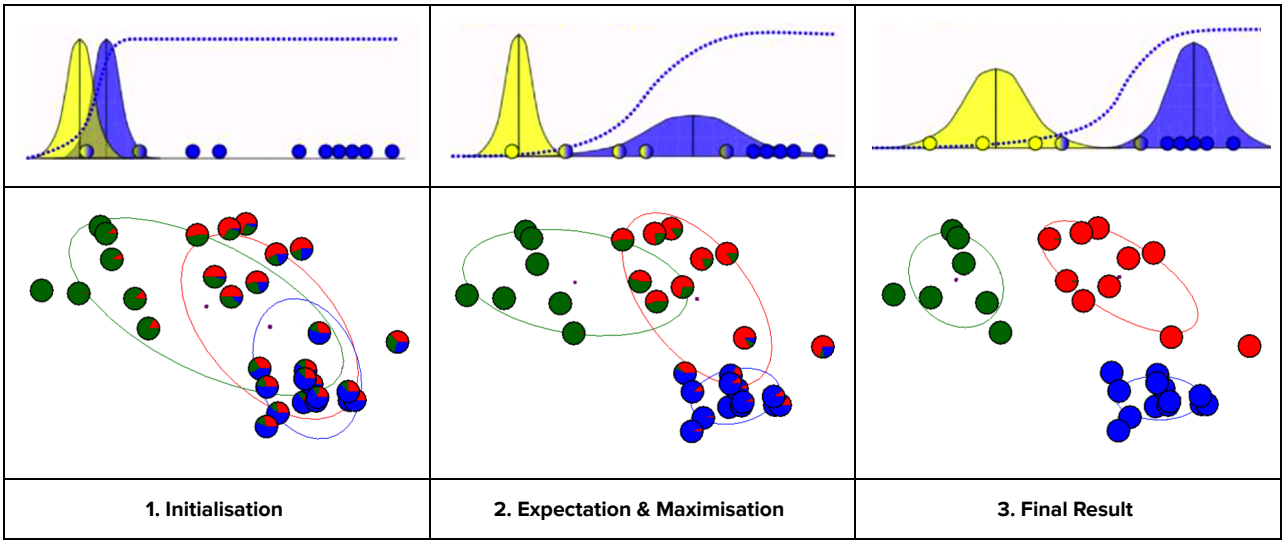


3. Maximisation : For each of the components C_k, the values of μ_k, φ_k and Σ_k are updated so as to maximize the expectations calculated above with respect to the model parameters. The equations for the maximisation step are as follows.

$$\varphi_k = \sum_{i=1}^{N} \frac{\Upsilon_{ik}}{N}, \qquad \mu_k = \frac{\sum_{i=1}^{N} \Upsilon_{ik} x_i}{\sum_{i=1}^{N} \Upsilon_{ik}}, \qquad \Sigma_k = \frac{\sum_{i=1}^{N} \Upsilon_{ik} \Sigma_i}{\sum_{i=1}^{N} \Upsilon_{ik}}$$

4. Iteration : The entire iterative process is repeated until the algorithm converges, giving a maximum likelihood estimate. Intuitively, the algorithm works because knowing the component assignment C_k for each x_i makes solving for μ_k, φ_k and Σ_k easy, while knowing μ_k, φ_k and Σ_k makes inferring $p(C_k|x_i)$ easy. The expectation step corresponds to the latter while the maximization step corresponds to the former. Thus, by alternating between the values which are assumed fixed or known, the maximum likelihood estimates of the non-fixed values are calculated in an efficient manner.

The following figures represent the 2-dimensional and 3-dimensional GMM algorithm.



1. Initialisation	2. Expectation & Maximisation	3. Final Result

Python Code - GMM Clustering

Build Model

```
>> from sklearn.mixture import GMM
>> model = GMM(n_components=k, random_state=n)
# Behaves similar to K-Means model
>> model = GMM(n_components=k, covariance_type='full', random_state=n)
>> clusters = model.fit(data)
```

PRINCIPAL COMPONENT ANALYSIS

5.2.1. PRINCIPAL COMPONENT ANALYSIS

Everytime one encounters a dataset for use in machine learning, invariably one has to deal with the most common problems like outlier treatment, multicollinearity, large number of features etc.

Let's consider a situation where in a logistic regression setting, there are a lot of correlated variables present in the dataset. In order to handle this, one has to use feature elimination techniques to remove the correlated features. But each time a feature is dropped, some information is also lost along with it. Even after all this, there may still be a fairly large number of features which may be potentially related, and can lead to unstable regression models. Similarly, consider another situation where EDA is being performed on a dataset with n records and p features. In order to visualize the dataset one has to go through at least $p(p-1)/2$ plots (even with 20 features it will be at least 190 plots). Thus, one can conclude that the need of the hour is to lower the number of features that are not correlated, without losing information contained in the dataset. This is where PCA comes into play.

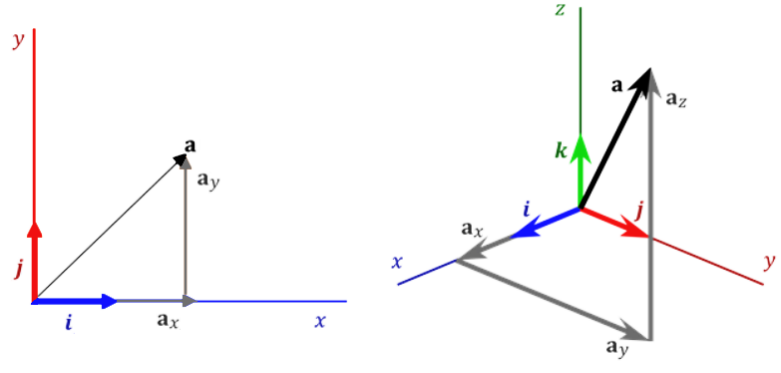
Principal Component Analysis (PCA)

Principal Component Analysis or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set. Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize and make analyzing data much easier and faster for machine learning algorithms without extraneous variables to process.

So to sum up, the idea of PCA is simple: reduce the number of variables of a data set, while preserving as much information as possible. PCA has several uses like,

1. Dimensionality reduction
2. Data visualization and Exploratory Data Analysis
3. Creation of uncorrelated features that can be used as input to a prediction model
4. Uncovering latent variables/themes/concepts
5. Noise reduction in the dataset

Building Blocks of PCA - Basis of Space



1. *x* and *y* axes are the dimensions.
2. *i* is a unit vector in the x direction, *j* is a unit vector in the Y direction.
3. Any point in the 2D space can be expressed in terms of *i* and *j*.
4. The *i* and *j* vectors are the basis of space.
5. *i* and *j* are independent i.e. *i* can't be expressed in terms of *j* and vice versa.
6. The same idea can be extended to any number of dimensions.

A set B of elements (vectors) in a vector space V is called a basis, if every element of V can be written in a unique way as a linear combination of elements of B. The coefficients of this linear

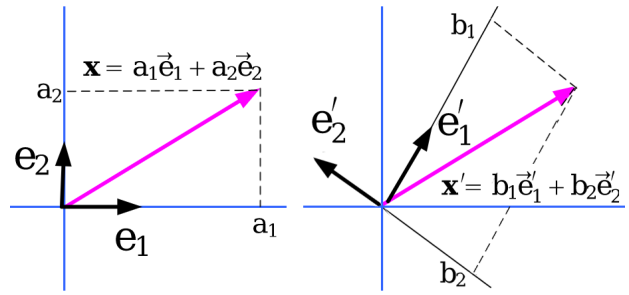
combination are referred to as coordinates on B of the vector. The elements of a basis are called basis vectors. Equivalently B is a basis if its elements are linearly independent and every element of V is a linear combination of elements of B. A vector space can have several bases, however all the bases have the same number of elements, called the dimension of the vector space. The following figure gives a graphical representation.

For a dataset or dataframe or table or matrix,

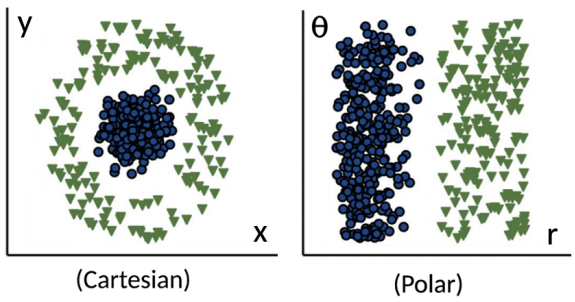
1. Each row (observation) is a point in space.
2. Each column is a basis vector i.e any point (observation) is represented in terms of columns.

Building Blocks of PCA - Basis Transformation

A basis for a vector space of dimension n is a set of n vectors (a_1, a_2 .... a_n), with the property that every vector in the space can be expressed as a unique linear combination of the basis vectors. Since it is often desirable to work with more than one basis for a vector space, it is of fundamental importance to be able to easily transform coordinate-wise representations of vectors and operators taken with respect to one basis to their equivalent representations with respect to another basis. This process of converting the information from one set of basis to another is called basis transformation.



Transformation by rotation of basis



Transformation by change of basis

For a dataset or dataframe or table or matrix, It is the process of representing the data in new columns different from the original (usually for convenience, efficiency or just from common sense).

Building Blocks of PCA - Variance

Consider the following situations where a dataset has four attributes such that,

1. Attribute 1 has only one single value in all the records.
2. Attribute 2 has all the same values except one record.
3. Attribute 3 has a fair amount of variation in the values of the records.
4. Attribute 4 has a large amount of variation in the values of the records.

While using the above data for building a model one can easily drop the attributes 1 and 2, because these attributes have no information to add (all the values are the same). Thus, one can acknowledge that variance is equivalent to information. Therefore, variables which capture variance in the data, are the variables that capture the information in the data.

Building Principal Components

Combining the above building blocks, PCA can be defined as a technique which takes the high-dimensional data and uses the dependencies between the variables to represent it in a more tractable, lower-dimensional basis, without losing too much information. Or to summarise, PCA finds the principal components $z_1, z_2 \ldots\ldots, z_n$ such that each principal component is independent (i.e. perpendicular) to each other and captures as much variance as possible. The principal components are linear combinations of the original features.

The following steps are applied to select the principal components.

1. Initialisation : The first principal component, capturing most variance is calculated as,

   $$Z_1 = \varphi_{11}X_1 + \varphi_{21}X_2 + \ldots. + \varphi_{p1}X_p \quad \left(\varphi \textit{ is such that the variance on } Z_1 \textit{ is maximum}\right)$$

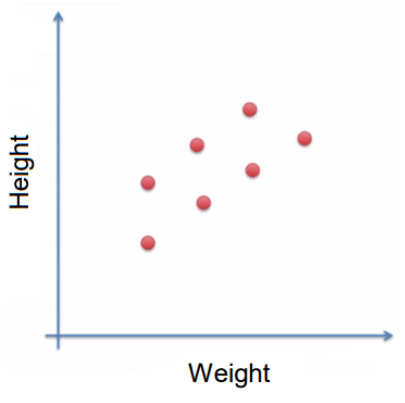
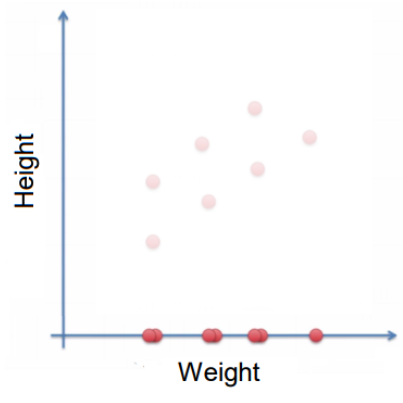
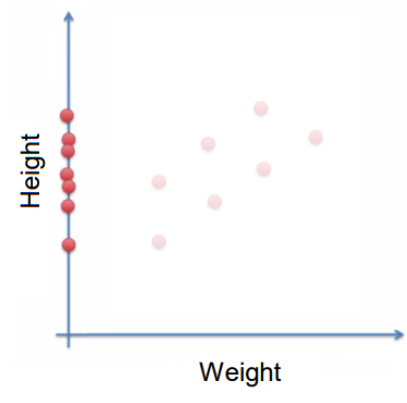
   It is approximately the line in which the projection of the data points is the most spread out. Or mathematically, it's the line that maximizes the variance (the average of the squared distances from the projected data points to the origin).

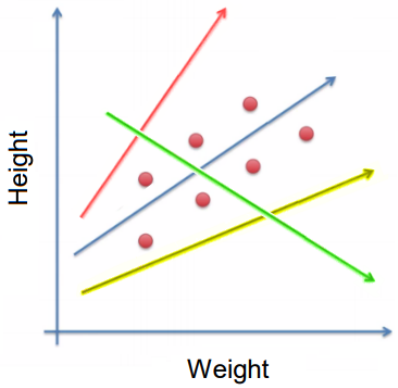
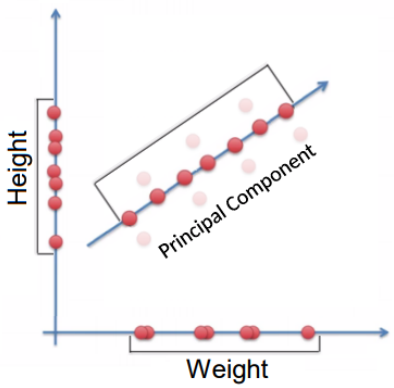
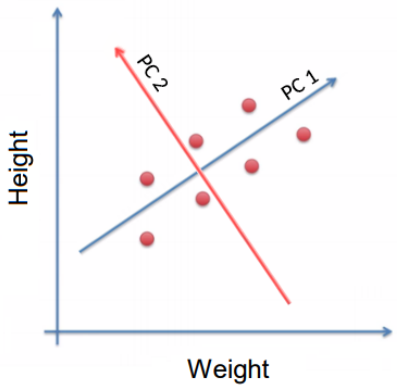
2. Assignment : The second principal component is calculated in the same way, with the condition that it is uncorrelated with (i.e., perpendicular to) the first principal component and that it accounts for the next highest variance.

3. Iteration : The process of assignment is repeated until a total of p principal components have been calculated, equal to the original number of variables. Each additional component captures incremental variance.

The above steps detect p Principal Components which is equal to the number of variables in the dataset. If one chooses to represent the data with k components, such that $k < p$, then it is called dimensional reduction.

The following figures are a graphical representation of the detecting principal components.

		
Objective is to find vectors on which the projected data has the maximum variance.	**On projecting upon the Weight axis, the vertically aligned data points get overlapped (i.e. the vertical variation is not captured).**	**On projecting upon the Height axis, the horizontally aligned points get overlapped (i.e. horizontal variation is not captured).**

		
Purely horizontal or vertical axis does not suffice the variation in both directions. Thus, an angled line is required.	**The line closest to the data retains the maximum variation in the original data points. In fact, PCA finds lines/planes/surfaces closest to data points.**	**There is still some variance that is left in the perpendicular direction to the first PC. This forms the second PC.**

PCA Algorithm (Eigen Decomposition Method)

The following algorithm is applied to select the principal components using the Eigen Decomposition method.

1. A covariance matrix is a matrix that explains both the spread (variance in diagonal elements) and the orientation (covariance in off-diagonal elements) of the dataset. Thus, the original matrix X is first centered and scaled and then the covariance matrix Σ is computed using the following formula,

$$\Sigma = X^T . X = \begin{bmatrix} \sigma(X_1, X_1) & \sigma(X_1, X_2) & \cdots & \sigma(X_1, X_n) \\ \sigma(X_2, X_1) & \sigma(X_2, X_2) & \cdots & \sigma(X_1, X_n) \\ \vdots & \vdots & & \vdots \\ \sigma(X_n, X_1) & \sigma(X_n, X_2) & \cdots & \sigma(X_n, X_n) \end{bmatrix}$$

2. In order to represent the covariance matrix with a vector and its magnitude, one can usually always find a vector v that points into the direction of the largest spread of the data with magnitude equal to the spread (variance) in that direction. As any linearly transformed square matrix ($N \times N$) can be completely defined by its eigenvectors and eigenvalues, Eigen Decomposition is performed on the covariance matrix to get the eigenvalues and eigenvectors by using the following concept.

   An eigenvector v of a linear operator A (described by a square matrix), is a vector that is mapped to a scaled version of itself, i.e. $Av = \lambda v$, where λ is the corresponding eigenvalue. Using this property any matrix A of rank r, can be represented as,

$$AV = V\Lambda$$
$$where,\ \Lambda\ is\ an\ (r \times r)\ diagonal\ matrix\ of\ r\ non\ zero\ Eigen\ values$$
$$V\ is\ an\ (n \times r)\ matrix\ of\ Eigen\ vectors$$

For a full rank matrix i.e $r = n$, A can be represented as,

$$A = V \Lambda V^{-1}$$

And for a symmetric positive definite matrix similar to the covariance matrix Σ above, it can be represented as,

$$\Sigma = V \Lambda V^{T}$$

3. These eigenvectors give the principal components of the original matrix and the eigenvalues denote the amount of variance explained by the eigenvectors. Higher the eigenvalues, higher is the variance explained by the corresponding eigenvector. These eigenvectors are orthonormal (i.e. they are unit vectors and are perpendicular to each other).

PCA Algorithm (Singular Value Decomposition Method)

The SVD is an extension of the Eigen Decomposition method. In short it is a generalization of the Eigen Decomposition method for a positive semidefinite normal matrix (for example, a symmetric matrix with positive eigenvalues) to any $m \times n$ matrix. To summarise, SVD helps to decompose an $m \times n$ matrix A into three component matrices,

1. Projection $z = V^{T}x$ into an r-dimensional space, where r is the rank of A.
2. Element-wise multiplication with r singular values σ_i, i.e. $z' = \Sigma z$.
3. Transformation $y = Uz'$ to the m-dimensional output space.

Combining the three component matrices, A can be re-written as,

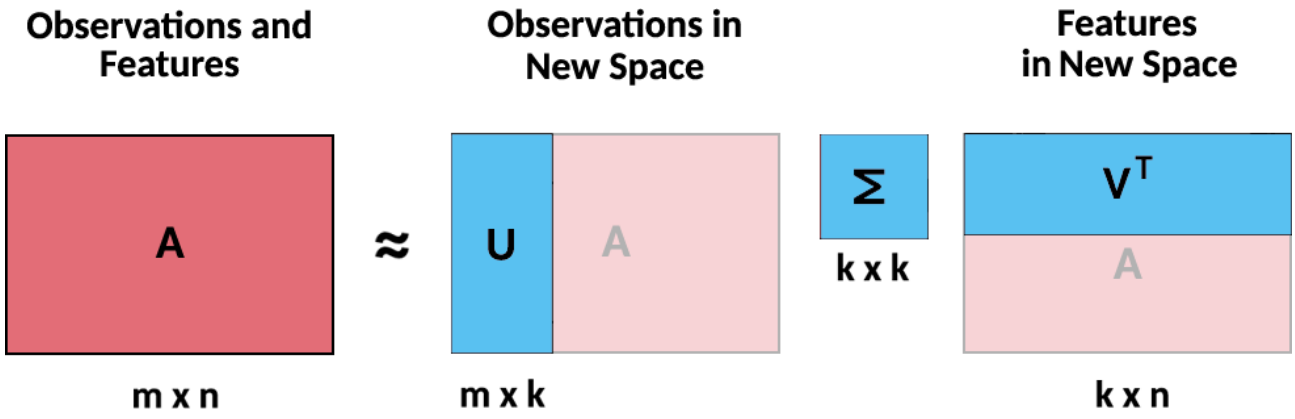
$$A = U \Sigma V^{T}$$

$$where, \quad U \; is \; an \; (m \times r) \; orthonormal \; matrix \; spanning \; A's \; column \; space$$

$$\Sigma \; is \; an \; (r \times r) \; diagonal \; matrix \; of \; \text{singular values}$$

$$V \; is \; an \; (n \times r) \; orthonormal \; matrix \; spanning \; A's \; row \; space$$

The following figure shows the algorithm applied to select the principal components using Singular Value Decomposition method.

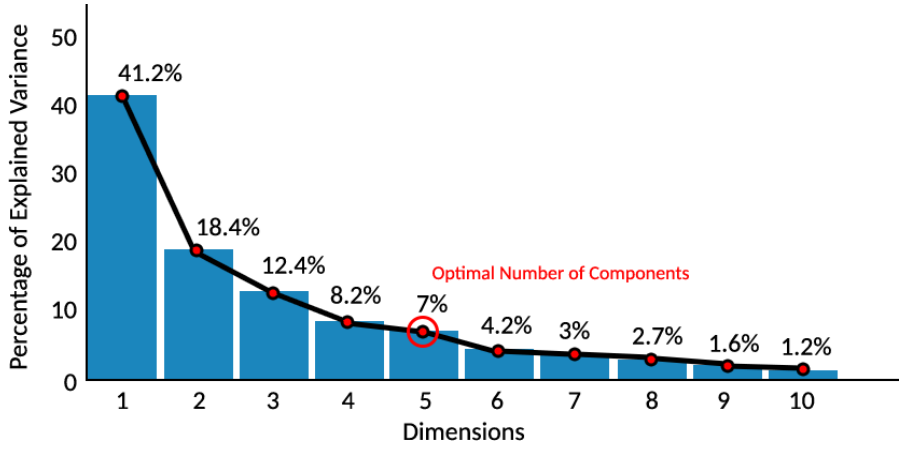


U is an $m \times k$ matrix	Rows to Themes. Maps the users to the new k themes/latent variables.
Σ is a diagonal $k \times k$ matrix	Weightages of Themes. Represents the strength of each feature.
V^T is a $k \times n$ matrix	Themes to Columns. Maps each of the k themes/latent variables to the original n features.

Thus, by decomposing one can easily remove the columns/features which have little to no theme strength but still preserve the variance level. This is basically the process of dimensionality reduction. When $k < n$, dimension reduction is said to be achieved as the $m \times n$ dataset with n original features is reduced to an $m \times k$ dataset with k principal components.

Scree Plots

Scree plots are line plots of the eigenvalues of factors or principal components in an analysis. These are used to determine the most important themes or the directions with the maximum amount of variance, by plotting the total cumulative variance against each principal component. The plot displays the eigenvalues in a downward curve, with the eigenvalues ordered from the largest to the smallest. The elbow of the graph where the eigenvalues seem to level off is deemed as the optimal point and the components to the left of this point are considered to be significant. The following figure shows a scree plot.



Practical Considerations for PCA

Some of the points to be considered while implementing PCA are as follows.

1. PCA being very scale sensitive, the data needs to be centered and scaled before feeding to a classifier model.

2. PCA being inherently a linear transformation method, it significantly boosts the performance when followed by linear models (like logistic regression). But, it can still give significant speed boosts to non-linear methods, by reducing the number of variables.

3. If the data already has largely uncorrelated features, a forced reduction in dimensionality using PCA can lead to severe loss of information. Thus, one needs to be careful while capturing the significant variance.

4. PCA requires the principal components to be uncorrelated/orthogonal/perpendicular, but sometimes there is the necessity of the data to be correlated so as to represent it. This may cause issues. Thus, one can use ICA (Independent Component Analysis), even though it is several times slower than the PCA.

5. PCA assumes that the low variance components are not very useful, which can lead to the loss of valuable information in a supervised learning situation (mainly for highly imbalanced classes)

6. PCA is used widely for image compression and data transmission. Based on the application the number of components are chosen such that the essence of the data (that is essential to the task) is retained.

The following figure represents the use of PCA for dimensionality reduction.

		
Original Image, k = 512	**Compressed Image, k = 256**	**Compressed Image, k = 128**
		
Compressed Image, k = 64	**Compressed Image, k = 32**	**Compressed Image, k = 16**

As can be seen, even after having reduced the number of dimensions from a much higher number of dimensions $k = 512$ to a very low number of dimensions $k = 16$, the image is still pretty much distinct and interpretable. Indeed, one can see that it is a very huge gain. Thus, one can conclude that despite there being some drawbacks in PCA, it is still a very efficient and powerful technique to reduce the complexity and discover hidden patterns present in the data.

Python Code - PCA

PCA

```
>> from sklearn.decomposition import PCA
>> from sklearn.decomposition import IncrementalPCA
>> model = PCA(n_components=number_of_components, svd_solver='randomized')
>> model = IncrementalPCA(n_components=number_of_components,
                          svd_solver='randomized')
>> model.fit(X_train)
>> model.components_
>> model.explained_variance_ratio_
```

Scree Plot

```
>> plt.plot(np.cumsum(model.explained_variance_ratio_))
>> plt.xlabel('number of components')
>> plt.ylabel('cumulative explained variance')
```

LINEAR DISCRIMINANT ANALYSIS

5.3.1. LINEAR DISCRIMINANT ANALYSIS

Linear Discriminant Analysis is a dimensionality reduction technique used as a preprocessing step in Machine Learning and pattern classification applications.