# Linear Regression

Types of Linear Regression:

* Linear Regression
* Logistic Regression
* Polynomial Regression
* Stepwise Regression
* Ridge Regression
* Lasso Regression
* ElasticNet Regression

**‘if you are using regression without regularization, you have to be very special!’**

## Simple Linear Regression:

Simple linear regression has one target variable and one predictor(continues)

The simplest – formula

**y = mx + c**

Linear regression is nothing but a manifestation of this simple equation.

* **y** is the dependent variable i.e. the variable that needs to be estimated and predicted.
* **x** is the independent variable i.e. the variable that is controllable. It is the input.
* **m** is the slope. It determines what will be the angle of the line. It is the parameter denoted as β.
* **c** is the intercept. A constant that determines the value of y when x is 0.

This can also be rewritten in y = β0 + β1X1 + e

In the above β0 is a constant and β1 is co efficient of X1 i.e amount of change in β1 changes X1

## Multi Linear Regression:

When the regression problem has more than one independent variable and one target variable

Formula:

y = β0 + β1X1 + β2X2+ β1X2+ e

X1 – Age

X2 – Gender

X3 – Experience etc. Variables

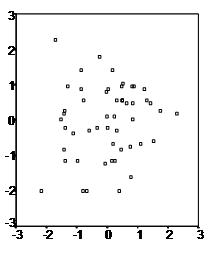
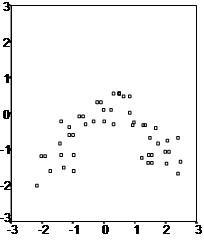
When experience increases the salary increases

Explanation of β0 can be gives as follows: When a graduate passes the college, his exp =0 and min salary is 30K

Assumptions of Multiple Linear Regression:

### Multiple Linear Regression Assumptions

1. First, multiple linear regression requires the **relationship between the independent and dependent variables to be linear**.  The linearity assumption can best be tested with scatterplots.  The following two examples depict a curvilinear relationship (left) and a linear relationship (right).



1. Second, the multiple linear regression analysis requires that the errors between observed and predicted values (i.e., the residuals of the regression) should be **normally distributed**. This assumption may be checked by looking at a histogram or a Q-Q-Plot.  Normality can also be checked with a goodness of fit test (e.g., the Kolmogorov-Smirnov test), though this test must be conducted on the residuals themselves.
2. Third, multiple linear regression assumes that there is **no multicollinearity** in the data.  Multicollinearity occurs when the independent variables are too highly correlated with each other. Multicollinearity means when once independent variable increase there is a increase in other independent variable too.

Multicollinearity may be checked multiple ways:

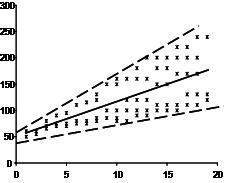
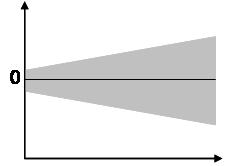
1) Correlation matrix – When computing a matrix of Pearson’s bivariate correlations among all independent variables, the magnitude of the correlation coefficients should be less than .80.

2) Variance Inflation Factor (VIF) – The VIFs of the linear regression indicate the degree that the variances in the regression estimates are increased due to multicollinearity. **VIF values higher than 10 indicate that multicollinearity is a problem.**

If multicollinearity is found in the data, one possible solution is to center the data.  To center the data, subtract the mean score from each observation for each independent variable. However, the simplest solution is to identify the variables causing multicollinearity issues (i.e., through correlations or VIF values) and removing those variables from the regression.

1. The last assumption of multiple linear regression is homoscedasticity.  A scatterplot of residuals versus predicted values is good way to check for homoscedasticity.  There should be no clear pattern in the distribution; if there is a cone-shaped pattern (as shown below), the data is heteroscedastic.

Homoscedasticity describes a situation in which the error term (that is, the “noise” or random disturbance in the relationship between the independent variables and the dependent variable) is the same across all values of the independent variables.  Heteroscedasticity (the violation of homoscedasticity) is present when the size of the error term differs across values of an independent variable



### Model Building:

1. From sklearn

from sklearn.linear\_model import LinearRegression

lm = LinearRegression()

lm.fit(X\_train,y\_train)

from sklearn import metrics

print('MAE:', metrics.mean\_absolute\_error(y\_test, predictions))

print('MSE:', metrics.mean\_squared\_error(y\_test, predictions))

print('RMSE:', np.sqrt(metrics.mean\_squared\_error(y\_test, predictions)))

1. From statsmodel.api

from statsmodels import api as sm

model = sm.OLS(y\_train, X\_train).fit()

ols\_predictions = model.predict(X\_test)

model.summary()

### Model Evaluation:

OLS Summary:



* R2 – always towards one is better, when we add variable the R2 increases
* Adjusted R2 – Adjusted R2 will increase when a unnecessary variable is added, adjusted r2 helps in deciding to keep the number of important variables
* R2 – can be negative, which indicates that the predicted values are lower than the mean of the values, which denotes that the model is doing worst or not necessary
* Co Efficient – All the co efficient should be relatively greater than zero indicating it has some effect
* **Omnibus/Prob(Omnibus)**– a test of the skewness and kurtosis of the residual (characteristic #2). We hope to see a value close to zero which would indicate normalcy. The Prob (Omnibus) performs a statistical test indicating the probability that the residuals are normally distributed. We hope to see something close to 1 here. In this case Omnibus is relatively low and the Prob (Omnibus) is relatively high so the data is somewhat normal, but not altogether ideal. A linear regression approach would probably be better than random guessing but likely not as good as a nonlinear approach.
* **Skew**– a measure of data symmetry. We want to see something close to zero, indicating the residual distribution is normal. Note that this value also drives the Omnibus. This result has a small, and therefore good, skew.
* **Kurtosis**– a measure of "peakiness", or curvature of the data. Higher peaks lead to greater Kurtosis. Greater Kurtosis can be interpreted as a tighter clustering of residuals around zero, implying a better model with few outliers.
* **Durbin-Watson**– tests for homoscedasticity (characteristic #3). We hope to have a value between 1 and 2. In this case, the data is close, but within limits.
* **Jarque-Bera z** of the Omnibus test. In this case we do.
* **Condition Number** **–** This test measures the sensitivity of a function's output as compared to its input (characteristic #4). When we have multicollinearity, we can expect much higher fluctuations to small changes in the data, hence, we hope to see a relatively small number, something below 30. In this case we are well below 30, which we would expect given our model only has two variables and one is a constant.

### Examples from web:

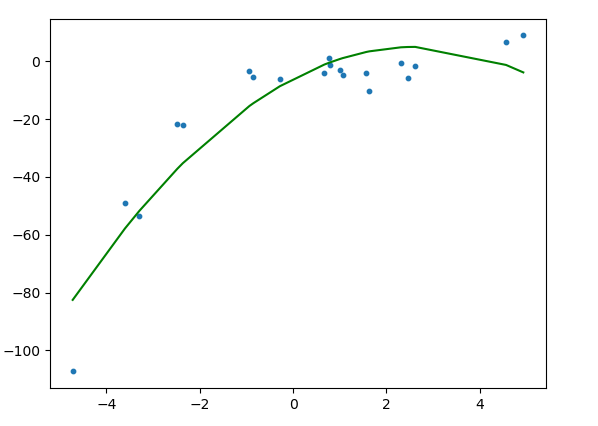
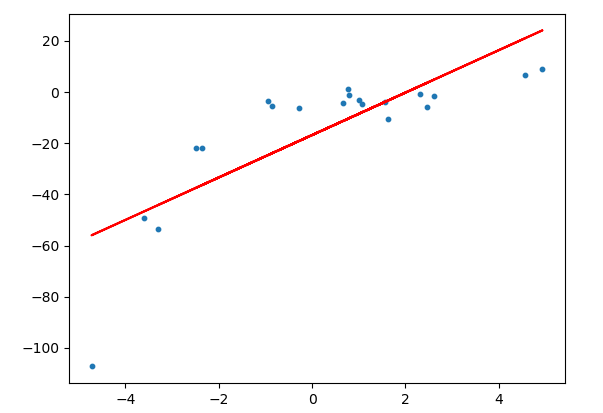
Simple Linear Regression: 

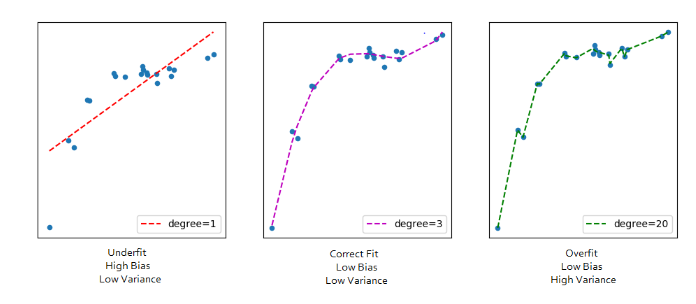
## Polynomial Regression

Description:

When a bet fit line cannot be drawn as a linear line or when the datapoints are not distributed in a linear fashion, Polynomial regression is used to find the best fit line – the line curved over all the data points

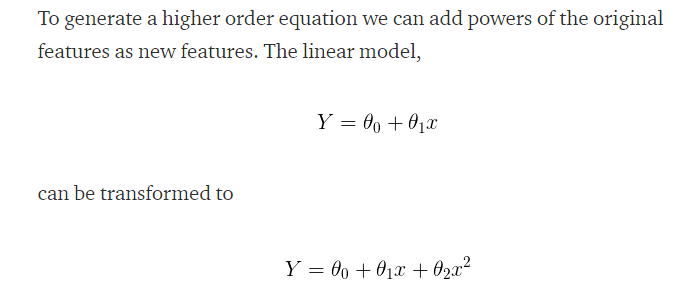
Eg 1





Formula:

Linear regression to Polynomial regression



When degree is 2 then y = β0 + β1x1 + β1x12

When degree is 4 then y = β0 + β1x1 + β1x12 + β1x13 + β1x14

A polynomial regression can be overfitted when the degrees of are not set correctly.

### Model Building:

from sklearn.preprocessing import PolynomialFeatures

poly\_reg = PolynomialFeatures(degree = 4)

X\_poly = poly\_reg.fit\_transform(X)

poly\_reg.fit(X\_poly, y)

lin\_reg\_2 = LinearRegression()

lin\_reg\_2.fit(X\_poly, y)

lin\_reg\_2.predict(poly\_reg.fit\_transform(X\_test))

## Redge, Lasso and Elastinet Regressions:

When working with a dataset that has too many features, these regression techniques can be used, to choose the optimal features for model building. This is a regularization technique meaning model should fit for both linear or non linear models

These techniques work on L1 and L2 regularization method by penalizing the coefficient by λ

1. **Ridge Regression:**
   * Performs L2 regularization, i.e. adds penalty equivalent to **square of the magnitude** of coefficients
   * Minimization objective = LS Obj + α \* (sum of square of coefficients)
2. **Lasso Regression:**
   * Performs L1 regularization, i.e. adds penalty equivalent to **absolute value of the magnitude** of coefficients
   * Minimization objective = LS Obj + α \* (sum of absolute value of coefficients)

Note that here ‘LS Obj’ refers to ‘least squares objective’, i.e. the linear regression objective without regularization.

1. Key Difference

* **Ridge**: It includes all (or none) of the features in the model. Thus, the major advantage of ridge regression is coefficient shrinkage and reducing model complexity.
* **Lasso**: Along with shrinking coefficients, lasso performs feature selection as well. (Remember the ‘*selection*‘ in the lasso full-form?) As we observed earlier, some of the coefficients become exactly zero, which is equivalent to the particular feature being excluded from the model.

Traditionally, techniques like stepwise regression were used to perform feature selection and make parsimonious models. But with advancements in Machine Learning, ridge and lasso regression provide very good alternatives as they give much better output, require fewer tuning parameters and can be automated to a large extend.

2. Typical Use Cases

* **Ridge**: It is majorly used to *prevent overfitting*. Since it includes all the features, it is not very useful in case of exorbitantly high #features, say in millions, as it will pose computational challenges.
* **Lasso**: Since it provides *sparse solutions*, it is generally the model of choice (or some variant of this concept) for modelling cases where the #features are in millions or more. In such a case, getting a sparse solution is of great computational advantage as the features with zero coefficients can simply be ignored.

Its not hard to see why the stepwise selection techniques become practically very cumbersome to implement in high dimensionality cases. Thus, lasso provides a significant advantage.

3. Presence of Highly Correlated Features

* Ridge: It generally works well even in presence of highly correlated features as it will include all of them in the model but the coefficients will be distributed among them depending on the correlation.
* Lasso: It arbitrarily selects any one feature among the highly correlated ones and reduced the coefficients of the rest to zero. Also, the chosen variable changes randomly with change in model parameters. This generally doesn’t work that well as compared to ridge regression.

This disadvantage of lasso can be observed in the example we discussed above. Since we used a polynomial regression, the variables were highly correlated. ( Not sure why? Check the output of data.corr() ). Thus, we saw that even small values of alpha were giving significant sparsity (i.e. high #coefficients as zero).

Along with Ridge and Lasso, Elastic Net is another useful techniques which combines both L1 and L2 regularization. It can be used to balance out the pros and cons of ridge and lasso regression. I encourage you to explore it further.

### Documentation:



# CART

CART – Classification and Regression Trees

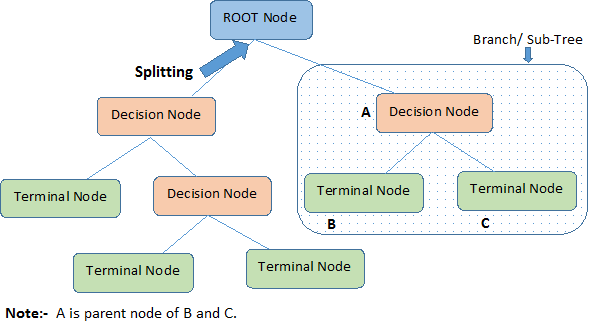
CART contains Decision Tree & Random Forest

## Decision Tree:

Decision Tree follow top down approach, splits the values into non lapsing boxes in multi dimension

**Important Terminology related to Tree based Algorithms**

Let’s look at the basic terminology used with Decision trees:

1. **Root Node:**It represents entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:**It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:**When a sub-node splits into further sub-nodes, then it is called decision node.
4. **Leaf/ Terminal Node:**Nodes do not split is called Leaf or Terminal node.
5. [](https://www.analyticsvidhya.com/wp-content/uploads/2015/01/Decision_Tree_2.png)**Pruning:**When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.
6. **Branch / Sub-Tree:**A sub section of entire tree is called branch or sub-tree.
7. **Parent and Child Node:**A node, which is divided into sub-nodes is called parent node of sub-nodes where as sub-nodes are the child of parent node.

**Advantages**: Easy to understand, Useful for EDA, Less data cleaning

**Disadvantages**: Overfitting and Not fit for continues variables

**How does a node is split:**

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

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

The algorithm selection is also based on type of target variables. Let’s look at the four most commonly used algorithms in decision tree:

Gini index, Chi Square , Information gain using Entropy, Reduction in variance

**Entropy** Example is attached here



Code Sample:

**from** **sklearn.tree** **import** DecisionTreeClassifier

dtree = DecisionTreeClassifier()

dtree.fit(X\_train,y\_train)

DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=None,

max\_features=None, max\_leaf\_nodes=None, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

presort=False, random\_state=None, splitter='best')

predictions = dtree.predict(X\_test)

**from** **sklearn.metrics** **import** classification\_report,confusion\_matrix

print(confusion\_matrix(y\_test,predictions))

## Random Forest:

* Random forest is a ensemble model
* Ensemble is a method of having two are models together (multi decision trees)
* Regression – Average of all models
* Classification – Max Voting or Mode
* Random forest is done using a Boostrap samples is sample with replacement (chances of choosing the same sample is equal to choosing the sample again)
* OOB – Out of Bag, each time when a sample is chosen, left over samples are used to test the chosen sample. Chosen sample is tested against OOB sample, OOB performance is important to get the correct model validation
* Variable importance is for a feature section is based on Gini Index
* Random Forest is mainly used for feature selection, it randomly chooses the feature in each tree, so that the averaging process can reduce the variance by decollerating data trees

### Sample Code:

from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n\_estimators=100)

rfc.fit(X\_train, y\_train)

RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=100, n\_jobs=1,

oob\_score=False, random\_state=None, verbose=0,

warm\_start=False)

rfc\_pred = rfc.predict(X\_test)

### Hyperparameters in RF

Max\_depth 🡪 is the longest path between the root and leaf node, having more depth can improve accuracy, but also can lead to overfit the training set

Min\_sample\_split 🡪 to reduce the overfitting

Min\_samples\_leaf 🡪 the samples are considered only when the min sample leaf is satisfied. Eg. If 5, 5 leaf should be available in a root node to consider for training

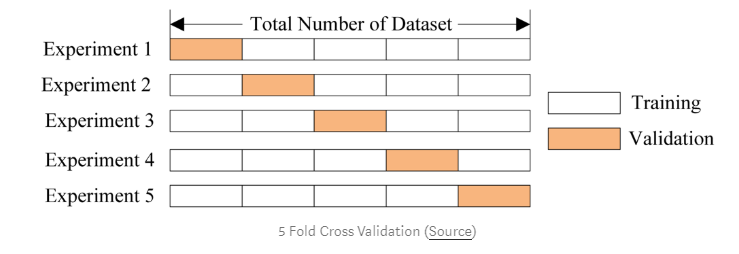
N\_estimators 🡪 how many trees for the forest

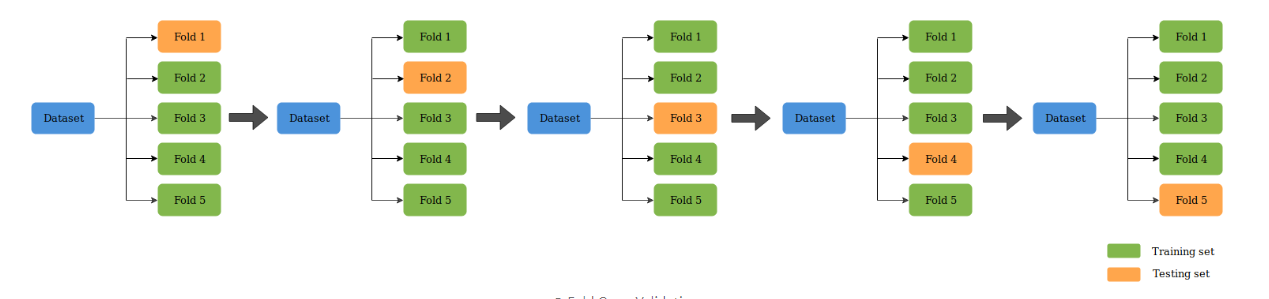
Max\_features 🡪 how many features to be chosen during the random split

# Cross validation

## k-fold

K-Fold CV is where a given data set is split into a ***K*** number of sections/folds where each fold is used as a testing set at some point. Lets take the scenario of 5-Fold cross validation(K=5). Here, the data set is split into 5 folds. In the first iteration, the first fold is used to test the model and the rest are used to train the model. In the second iteration, 2nd fold is used as the testing set while the rest serve as the training set. This process is repeated until each fold of the 5 folds have been used as the testing set.

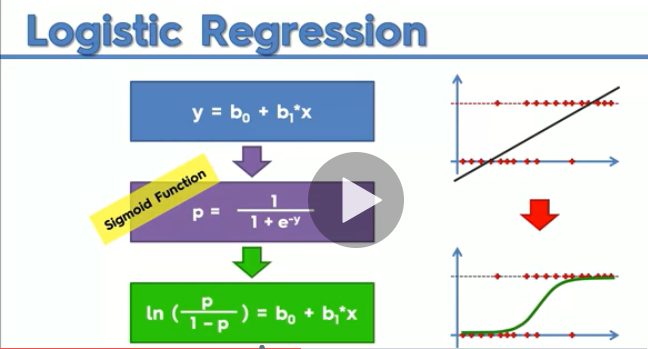


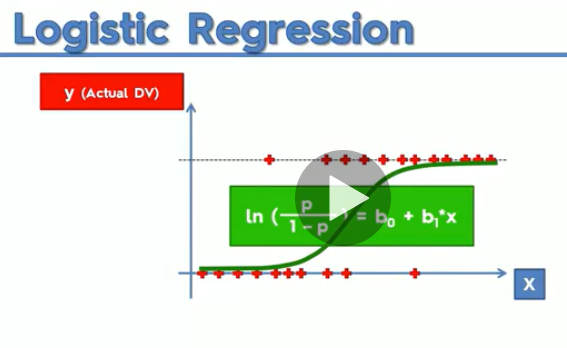


# Logistic Regression

* Logistic Regression is probability score of Linear Regression
* It applies Sigmoid function as cost function, to bound the values between 0 and 1
* Threshold values can be decided by organization level from 0.1 to 1

y = e^(b0 + b1\*x) / (1 + e^(b0 + b1\*x))





### Evaluation:

Confusion Matrix

|  |  |  |
| --- | --- | --- |
|  | **Predicted Positive** | **Predicted Negative** |
| **Actual Positive** | True Positive | False Negative - type2 |
| **Actual Negative** | False Positive - type1 | True Negative |
|  |  |  |
|  | **Predicted Positive** | **Predicted Negative** |
| **Actual Positive** | 8 | 2 |
| **Actual Negative** | 1 | 4 |
|  | 9 | 6 |
|  |  |  |
| Accuracy = TP + TN / total | 8+4/15 | 0.8 |
| Precision - Proportion of correct positive results out of all positive results | 8/9 | 0.888888889 |
| Recall - Proportion of actual positive cases | 8/10 | 0.8 |
| F1 Score - Weighted Average (Harmonic Mean) of Precision and Recall |  |  |
| F1 Score = 2 \* Precision \* Recall /(precistion + recall) | (2\*0.888\*0.8)/(0.88+0.8) | 0.845714286 |

F1 Score tells us which model performs better

ROC – Curve is plot of all possible Thresholds 0.1, 0.2, 0.3 …. 1.0

Higher the AUC, better the model is at pre dicting 0s as 0s and 1s as 1s

### Sample Code

from sklearn.linear\_model import LogisticRegression

logmodel = LogisticRegression()

logmodel.fit(X\_train,y\_train)

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, max\_iter=100, multi\_class='ovr', n\_jobs=1,

penalty='l2', random\_state=None, solver='liblinear', tol=0.0001,

verbose=0, warm\_start=False)

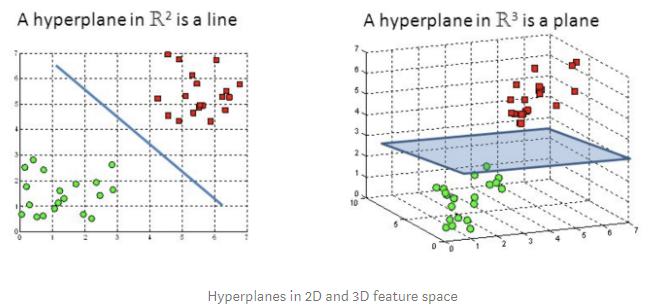
predictions = logmodel.predict(X\_test)

from sklearn.metrics import classification\_report

print(classification\_report(y\_test,predictions))

# Support Vector Machine (SVM)

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points.



Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.

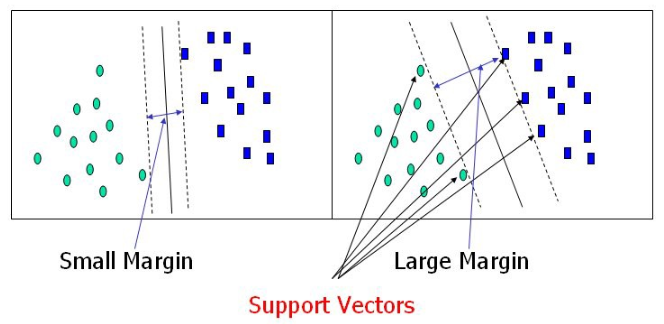
## Cost Function

In the SVM algorithm, we are looking to maximize the margin between the data points and the hyperplane. The loss function that helps maximize the margin is hinge loss.

Maximize the distances between two data points(Margin)

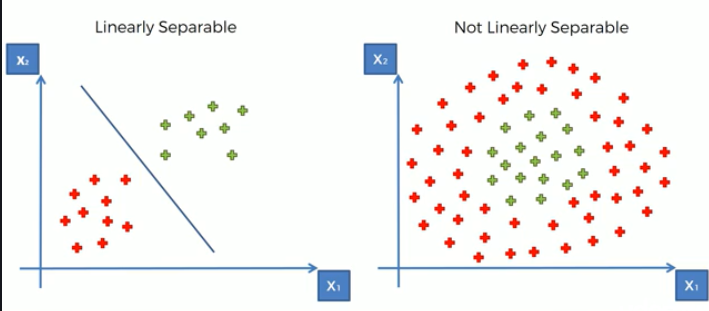
This is a distance based algorithm, so scaling is important.

SVM is different because it looks for very extreme case(apple that does not looks like apple)



## Kernel Trick:

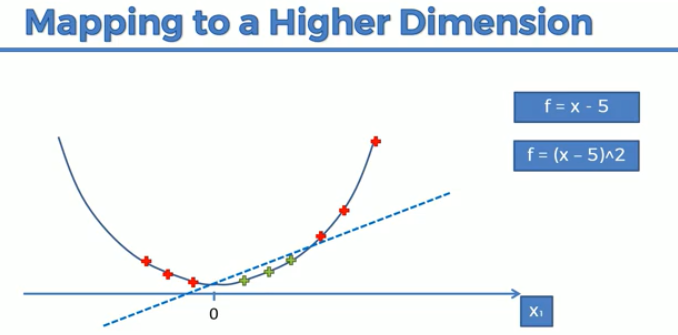
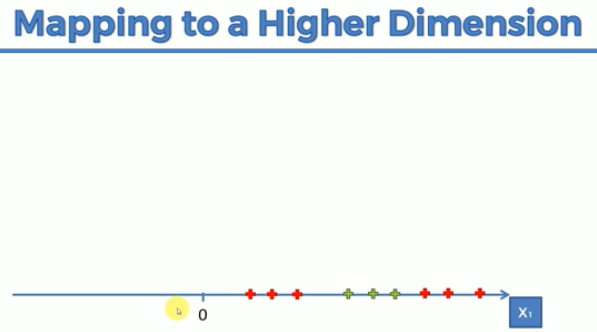
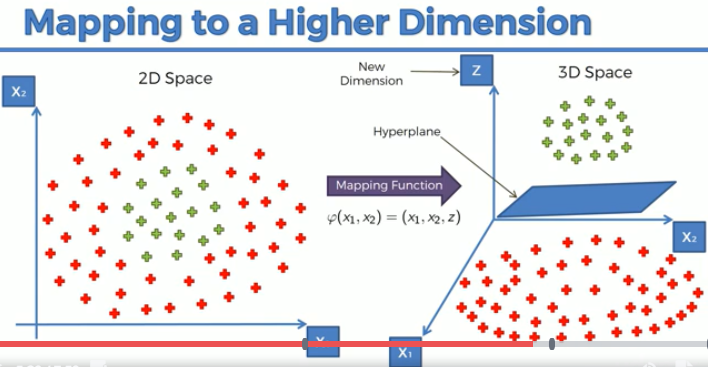
The idea is mapping the non-linear separable data-set into a higher dimensional space where we can find a hyperplane that can separate the samples.



Kernel can be very computational intensive

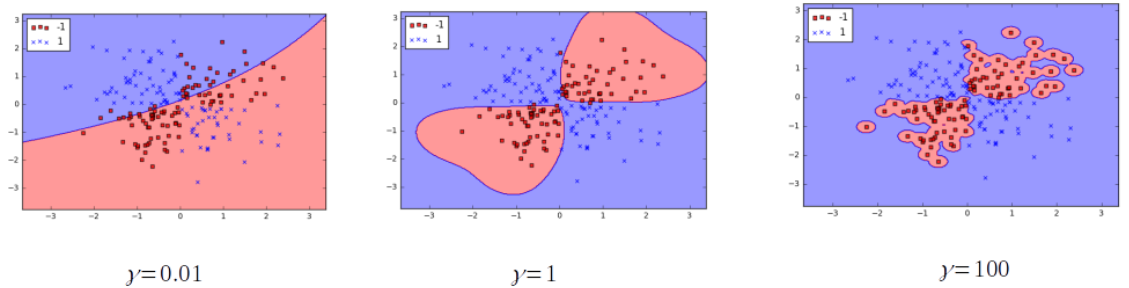
Mapping to Higher Dimention

Mapping a non linear separable data to Higher Dimension using a cost function:

## Types of Kernel Functions

* Linear Kernel SVM
* Polynomial Kernel SVM
* Gaussian Kernel SVM
* Sigmoid Kernel SVM
* Gaussian Radial Kernel SVM (RBF)
* RBF kernel we see that it depends on the Euclidean distance between two points, i.e. if two vectors are closer then this term is small



### Sample Code:

from sklearn.svm import SVC

model = SVC()

model.fit(X\_train,y\_train)

SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape=None, degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, probability=False, random\_state=None, shrinking=True,

tol=0.001, verbose=False)

predictions = model.predict(X\_test)

from sklearn.metrics import classification\_report,confusion\_matrix

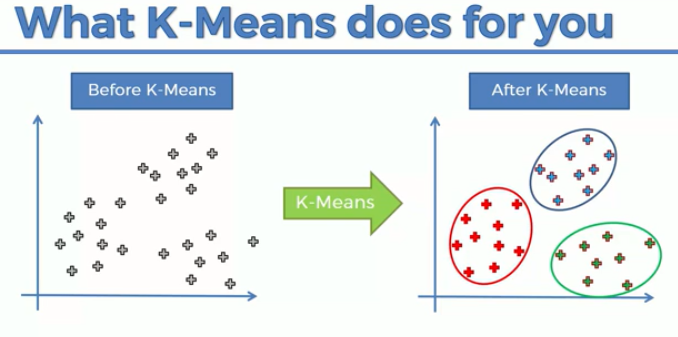
print(confusion\_matrix(y\_test,predictions))

print(classification\_report(y\_test,predictions))

# K Mean Clustering

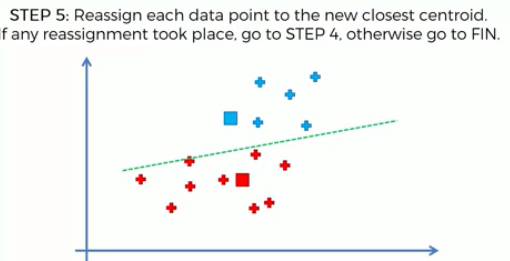
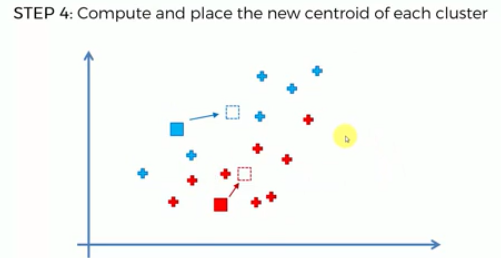
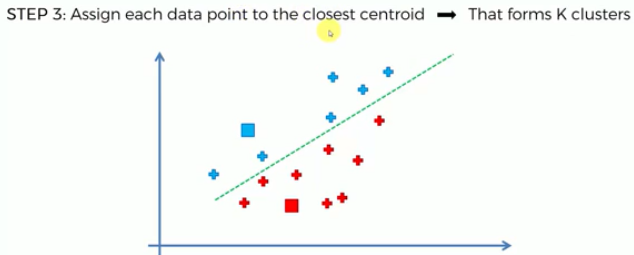
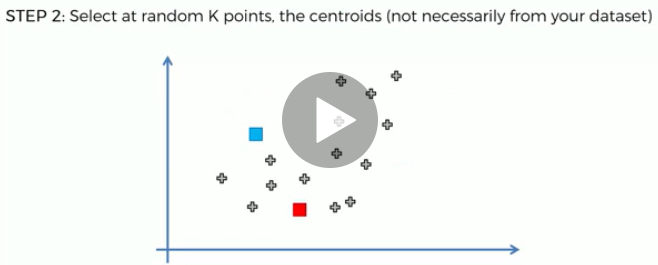
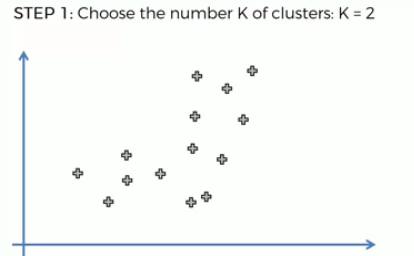
## K means Clustering:

K means clustering is an unsupervised learning, which has no target variable. Used to identify the groups.



### K means steps:

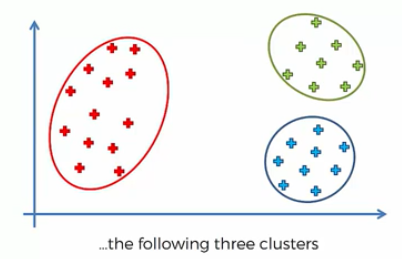
1. Choose the number of clusters
2. Select random K points, which serves as a centroids for the clusters
3. Assign each point to the closet centroid – distance (eludian) that forms cluster
4. Compute and place the new centroid
5. Reassign each data point to the new closest centroid. If any reassignment took place. Go to step 4. Otherwise finish



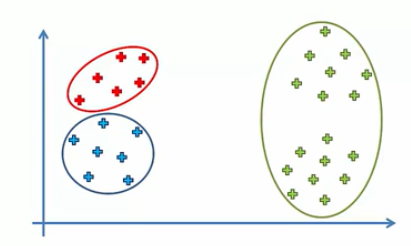
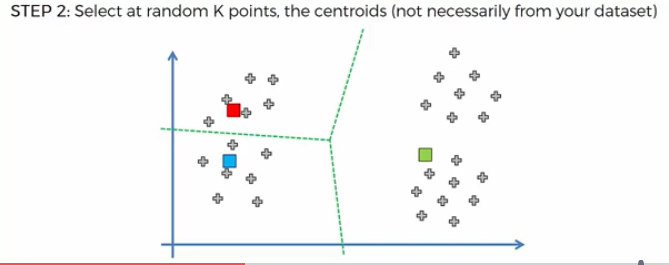
### K Means random initialization trap:

When a bad centroid point is initialized, it can treat to bad centroids and bad clusters like as follows

When we select randomly



When we have a bad random initialization

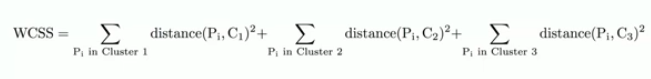


How to combat the random initialization into centroid is **K MEANS ++**

### Selecting number of Clusters:

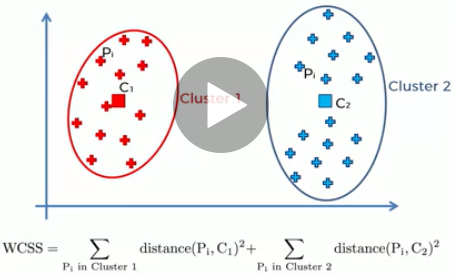
How to select the finding the number of correct clusters

To select the number of clusters the metric used is WCSS (Within Cluster Sum of Square)

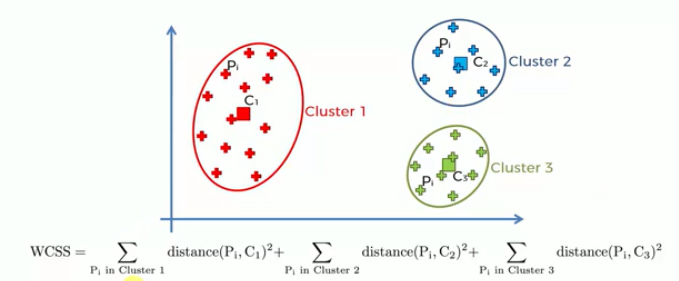


1. Calculate square distance of each point to centroid for all the points
2. Calculate all centroid sum of square points
3. When we have correct clusters, the WCSS tend to reduce

With 2 clusters



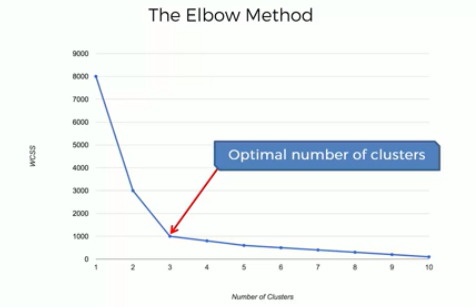
With 3 clusters



We can has as many as clusters a much as data points, then the WCSS distance weight would be zero 0.

### Elbow Method:

Correct no of clusters are chosen by Elbow method



We should choose the optimum number of cluster when there is elbow break, above 3 or 4

### Sample Code:

from sklearn.cluster import KMeans

wcss = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters = i, init = 'k-means++', random\_state = 42)

kmeans.fit(X)

wcss.append(kmeans.inertia\_)

plt.plot(range(1, 11), wcss)

plt.title('The Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()

# Fitting K-Means to the dataset

kmeans = KMeans(n\_clusters = 5, init = 'k-means++', random\_state = 42)

y\_kmeans = kmeans.fit\_predict(X)

# Visualising the clusters

plt.scatter(X[y\_kmeans == 0, 0], X[y\_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')

plt.scatter(X[y\_kmeans == 1, 0], X[y\_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')

plt.scatter(X[y\_kmeans == 2, 0], X[y\_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')

plt.scatter(X[y\_kmeans == 3, 0], X[y\_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')

plt.scatter(X[y\_kmeans == 4, 0], X[y\_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 300, c = 'yellow', label = 'Centroids')

plt.title('Clusters of customers')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.legend()

## Hierarchical Clustering

## Types:

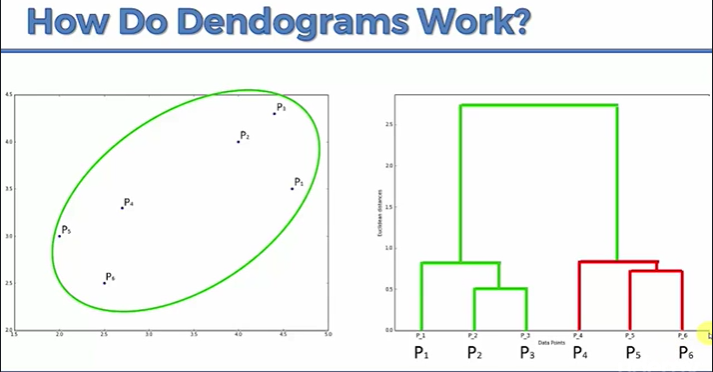
* Agglomerative (Most Common)
* Divisive

## Hierarchical Clustering Steps:

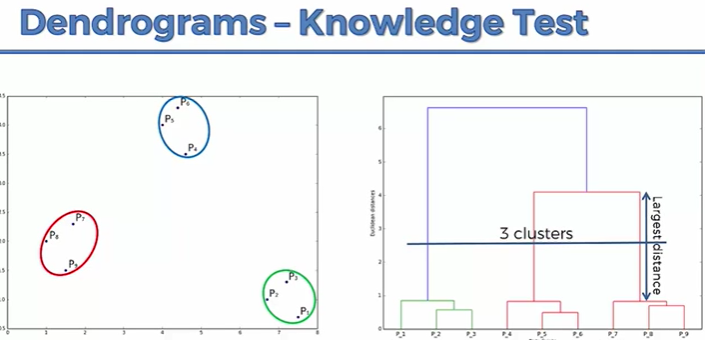
1. Make each data point as a single cluster
2. Take two closet data points and make them as a cluster
3. Take the **two closet clusters** and join them (Measure cluster space using Eludican distance, of farthest points, closest points, or between cluster centroids, etc.)
4. Repeat step3 until we have a single large cluster
5. Finish

### Constructing Dendograms:

Looking at the dendogram, we would come to know, what is the order of clustering



Maximum clusters are chosen by taking the longest distance in the dendograms like below



### Sample code:

dataset = pd.read\_csv('Mall\_Customers.csv')

X = dataset.iloc[:, [3, 4]].values

import scipy.cluster.hierarchy as sch

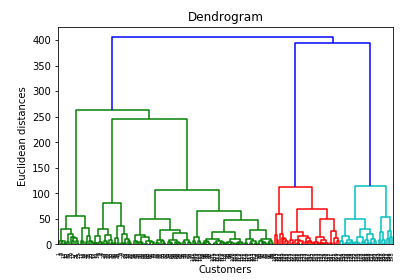
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))

plt.title('Dendrogram')

plt.xlabel('Customers')

plt.ylabel('Euclidean distances')

plt.show()



from sklearn.cluster import AgglomerativeClustering

hc = AgglomerativeClustering(n\_clusters=3, linkage='ward')

y\_hc = hc.fit\_predict(X)

# Dimensionality Reduction

## Principal Component Analysis

* Principle component analysis is used to reduce to reduce the no of variables
* When we have 100’s and 1000s of variables it can result in
* Sparse dataset in higher dimensions
* Overfit the data
* Higher computation effort
* Uses Eigen vectors and eigen values to reduce the dimensionality, higher eigen values are selected
* Used for feature extraction

### Steps:

1. Identify patter in data
2. Identify correlation in data, then combine them, where 2 variables can be explained by a single variable
3. Reduce the dimensions of a d-dimensional dataset by projecting it onto a (k)-dimensional subspace) where K < d
4. Standardize data
5. Obtain eigen values and vectors
6. Sort the Eigen values in descending order
7. Transform the original dataset into new datasets (<https://setosa.io/ev/principal-component-analysis/>)

### Sample Code:

|  |  |  |
| --- | --- | --- |
|  |  | X = dataset.iloc[:, 0:13].values |
|  |  | y = dataset.iloc[:, 13].values |
|  |  |  |
|  |  | # Splitting the dataset into the Training set and Test set |
|  |  | from sklearn.model\_selection import train\_test\_split |
|  |  | X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0) |
|  |  |  |
|  |  | # Feature Scaling |
|  |  | from sklearn.preprocessing import StandardScaler |
|  |  | sc = StandardScaler() |
|  |  | X\_train = sc.fit\_transform(X\_train) |
|  |  | X\_test = sc.transform(X\_test) |
|  |  |  |
|  |  | # Applying PCA |
|  |  | from sklearn.decomposition import PCA |
|  |  | pca = PCA(n\_components = 2) |
|  |  | X\_train = pca.fit\_transform(X\_train) |
|  |  | X\_test = pca.transform(X\_test) |
|  |  | explained\_variance = pca.explained\_variance\_ratio\_ |
|  |  |  |
|  |  | # Fitting Logistic Regression to the Training set |
|  |  | from sklearn.linear\_model import LogisticRegression |
|  |  | classifier = LogisticRegression(random\_state = 0) |
|  |  | classifier.fit(X\_train, y\_train) |
|  |  |  |
|  |  | # Predicting the Test set results |
|  |  | y\_pred = classifier.predict(X\_test) |
|  |  |  |
|  |  | # Making the Confusion Matrix |
|  |  | from sklearn.metrics import confusion\_matrix |
|  |  | cm = confusion\_matrix(y\_test, y\_pred) |