DATA PREPROCESSING

```
#importing libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#importing datasets
dataset = pd.read_csv('Data.csv')
X= dataset.iloc[:,:-1].values
Y = dataset.iloc[:, 3].values

#taking care of missing data
from sklearn.preprocessing import Imputer
imputer = Imputer(missing_values="NaN",strategy="mean",axis =0)
imputer = imputer.fit(X[:,1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])

#encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X = LabelEncoder()
X[:, 0]=labelencoder_X.fit_transform(X[:,0])
onehotencoder= OneHotEncoder(categorical_features = [0])
X=onehotencoder.fit_transform(X).toarray()
labelencoder_Y = LabelEncoder()
Y = labelencoder_Y.fit_transform(Y)

#splitting dataset into training 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_x = StandardScaler()
```

```
X_train = sc_x.fit_transform(X_train)
X_test = sc_x.transform(X_test)
```

➔ Feature scaling is used normalize the data in a particular range

SIMPLE LINEAR REGRESSION

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#importing datasets
dataset = pd.read_csv('Salary_Data.csv')
X= dataset.iloc[:,:-1].values
Y = dataset.iloc[:, 1].values

from sklearn.model_selection import train_test_split
X_train,X_test, Y_train, Y_test = train_test_split(X,Y,test_size= 1/3, random_state=0)

#fitting simple linear regression to the training set

from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, Y_train)

#predicting the test results
#we create a vector of all the predicted values
y_pred = regressor.predict(X_test)

#visualizing the training set results
#x-axis user experience, y-axis salary
plt.scatter(X_train,Y_train, color= 'red')
plt.plot(X_train, regressor.predict(X_train), color='blue')
plt.title('Salary vs Experience(Training Set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()

#visualizing the test set results
#x-axis user experience, y-axis salary
plt.scatter(X_test,Y_test, color= 'pink')
plt.plot(X_train, regressor.predict(X_train), color='violet')
plt.title('Salary vs Experience(Test Set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

MULTIPLE LINEAR REGRESSION

$$Y = c + m_1 * x_1 + m_2 * x_2 + m_3 * x_3 + \ldots\ldots\ldots\ldots$$

Assumptions of a linear regression:

1) Linearity
2) Homoscedasticity
3) Multivariate normality
4) Independence of errors
5) Lack of multicollinearity

➔ Dummy variable trap: Always omit one dummy variable

➔ 5 methods of building models

- All-in
- Backward Elimination
- Forward selection
- Bidirectional elimination

} Stepwise regression

- Score comparison

Backward elimination:

Step-1: Select a significance level to stay in model (SL=0.05)

Step-2: Fit the full model with all possible predictors

Step-3: Consider the predictor with highest p-value, if p>SL, go to step-4 otherwise go to FIN

Step-4: Remove predictor

Step-5: Fit the model without this variable

Forward Selection:

Step-1: Select a significance level to enter the model (SL =0.05)

Step-2: Fit all simple regression models $y \sim x_n$ select the one with the lowest p-value

Step-3: Keep this variable and fit all possible models with one extra predictor added to the one(s) you already have

Step-4: Consider the predictor with lowest p-value. If p<SL go to step-3, otherwise go to FIN

Bidirectional Elimination:

Step-1: Select a significance level to enter and stay in the model (SL_{enter} =0.05 , SL_{stay} = 0.05)

Step-2: Perform the next step of Forward selection (new Variables must have p<SL_{enter} to enter)

Step-3: Perform all steps of backward elimination (Old variables must have p<SL_{stay} to stay)

Step-4: No new variables can enter and no old variables can exit

Your model is ready

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

dataset = pd.read_csv('50_Startups.csv')
X= dataset.iloc[:,:-1].values
y= dataset.iloc[:, 4 ].values

from sklearn.preprocessing import LabelEncoder,OneHotEncoder
labelencoder_X= LabelEncoder()
X[:,3]= labelencoder_X.fit_transform(X[:,3])
onehotencoder= OneHotEncoder(categorical_features = [3])
X=onehotencoder.fit_transform(X).toarray()

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)

#Avoiding dummy variable trap
X=X[:,1:]

#Fitting multiple linear regression to the training set
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(X_train,y_train)

#predicting the test results
y_pred= regressor.predict(X_test)

#Y= c + m1*x1 + m2*x2 + m3*x3 + …….
# we represent like m0*x0 + m1*x1 + m2*x2 + ....mn*xn, where x0=1(always)
#So we are adding a column of 1's at beginning
import statsmodels.api as sm
X=np.append(arr= np.ones((50,1)).astype(int), values = X, axis=1)
```

X_opt = X[:, [0,1,2,3,4,5]]

regressor_OLS = sm.OLS(endog = y, exog= X_opt).fit()

regressor_OLS.summary()

Covariance Type: nonrobust

	coef	std err	t	P>|t|	[0.025	0.975]
const	5.013e+04	6884.820	7.281	0.000	3.62e+04	6.4e+04
x1	198.7888	3371.007	0.059	0.953	-6595.030	6992.607
x2	-41.8870	3256.039	-0.013	0.990	-6604.003	6520.229
x3	0.8060	0.046	17.369	0.000	0.712	0.900
x4	-0.0270	0.052	-0.517	0.608	-0.132	0.078
x5	0.0270	0.017	1.574	0.123	-0.008	0.062

#2nd attribute has highest p value

X_opt = X[:, [0,1,3,4,5]]

regressor_OLS = sm.OLS(endog = y, exog= X_opt).fit()

regressor_OLS.summary()

	coef	std err	t	P>|t|	[0.025	0.975]
const	5.011e+04	6647.870	7.537	0.000	3.67e+04	6.35e+04
x1	220.1585	2900.536	0.076	0.940	-5621.821	6062.138
x2	0.8060	0.046	17.606	0.000	0.714	0.898
x3	-0.0270	0.052	-0.523	0.604	-0.131	0.077
x4	0.0270	0.017	1.592	0.118	-0.007	0.061

#1st attribute has highest p value

X_opt = X[:, [0,3,4,5]]

regressor_OLS = sm.OLS(endog = y, exog= X_opt).fit()

regressor_OLS.summary()

Covariance Type: nonrobust

	coef	std err	t	P>|t|	[0.025	0.975]
const	5.012e+04	6572.353	7.626	0.000	3.69e+04	6.34e+04
x1	0.8057	0.045	17.846	0.000	0.715	0.897
x2	-0.0268	0.051	-0.526	0.602	-0.130	0.076
x3	0.0272	0.016	1.655	0.105	-0.006	0.060

#remove 4th attriubute

X_opt = X[:, [0,3,5]]

regressor_OLS = sm.OLS(endog = y, exog= X_opt).fit()

regressor_OLS.summary()

Covariance Type:		nonrobust				
	coef	std err	t	P>|t|	[0.025	0.975]
const	4.698e+04	2689.933	17.464	0.000	4.16e+04	5.24e+04
x1	0.7966	0.041	19.266	0.000	0.713	0.880
x2	0.0299	0.016	1.927	0.060	-0.001	0.061

#remove 5th attribute

X_opt = X[:, [0,3]]

regressor_OLS = sm.OLS(endog = y, exog= X_opt).fit()

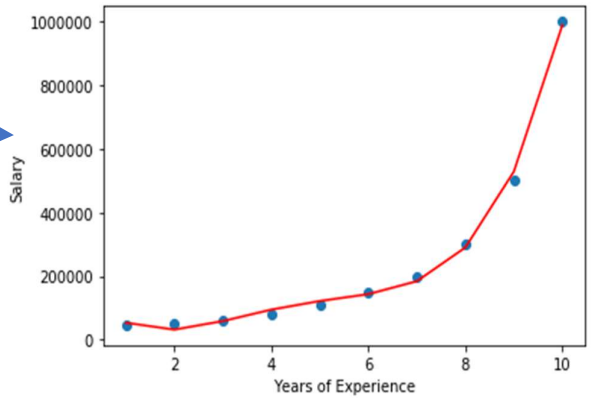
regressor_OLS.summary()

Covariance Type:		nonrobust				
	coef	std err	t	P>|t|	[0.025	0.975]
const	4.903e+04	2537.897	19.320	0.000	4.39e+04	5.41e+04
x1	0.8543	0.029	29.151	0.000	0.795	0.913

Polynomial Linear Regression

$$Y= c_0 + m_1x_1^2 + m_2x_2^2 + m_3x_3^3 + \ldots\ldots\ldots\ldots\ldots + m_nx_n^n$$

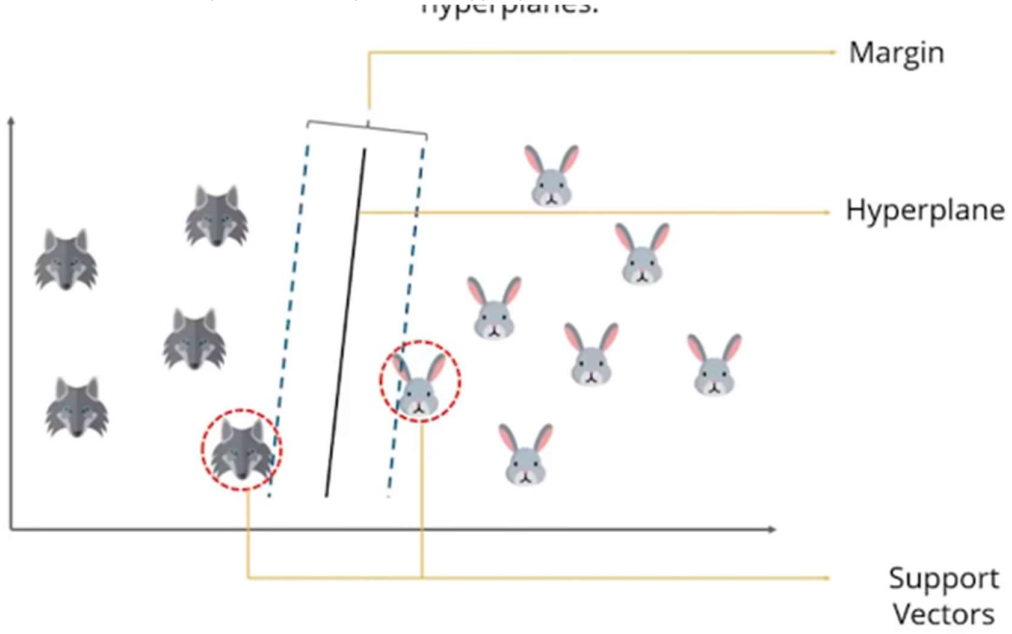
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset= pd.read_csv('Position_Salaries.csv')
X = dataset.iloc[:, 1:2 ]
y = dataset.iloc[:, 2]

from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
lin_reg= LinearRegression()
pol_reg=PolynomialFeatures( degree = 4)
X_poly = pol_reg.fit_transform(X)
lin_reg.fit(X_poly,y)
pol_reg.fit(X_poly,y)

plt.scatter(X,y)
plt.plot(X, lin_reg.predict(X_poly), color='red')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()

arr = np.arange(1, dtype = float).reshape(-1,1)
arr[0]=6.5
lin_reg.predict(pol_reg.fit_transform(arr))
```



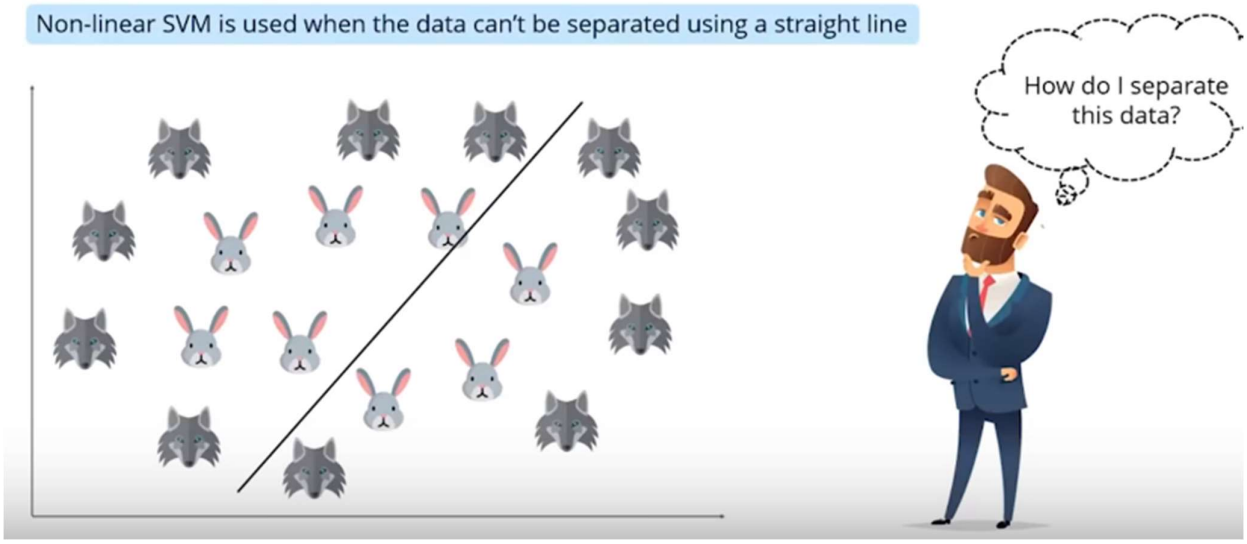

```
In [48]: arr = np.arange(1, dtype = float).reshape(-1,
    ...: arr[0]=6.5
    ...: lin_reg.predict(pol_reg.fit_transform(arr))
Out[48]: array([158862.45265153])
```

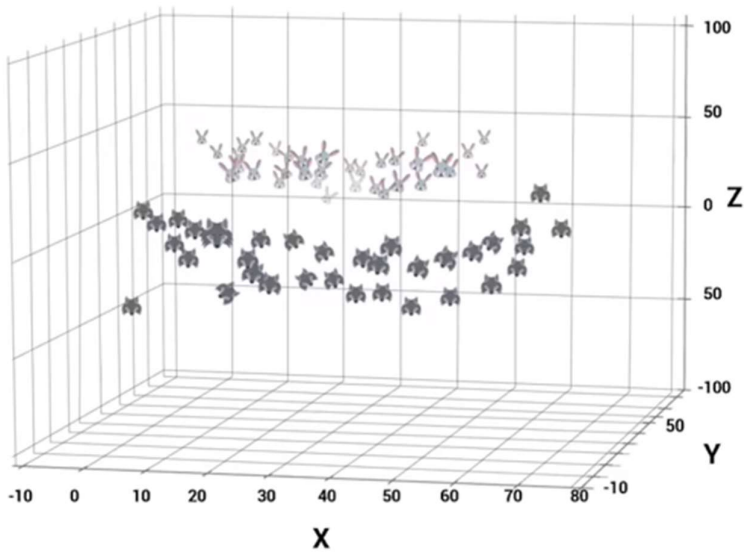
Support Vector Regression

- It is used for face recognition, cancer classification
- SVM is a supervised classification that separates data using hyperplanes i.e., data is divided into various segments and each segment contains one kind of data
- It is mainly used for classification purpose where in if you want to classify the data into two different segments depending on features of the data
- It can used for classification and regression
- SVM can be used to classify non-linear data by using kernel trick, kernel trick is to transfer the data into another dimension that you can easily draw a hyperplane between different classes of the data



- It is always suggested that the margin should be large so that it best separates the data into two different segments
- Now a question comes ,





- Now we plot the data in 3-dimensional plane

REAL WORLD USE-CASE OF SVM CLASSIFICATION:

- It is used to classify the cancer
- The dataset consists of 2000 transmembrane protein samples & about 50-200 gene samples it contains both colon cancer samples and normal colon tissue samples
- Even for very small data samples svm is very accurate, it outperforms naïve bayes algorithm in every case

Additional notes

- SVR performs linear regression in higher dimensional space
- Here each datapoint in the training represents it's own dimension. When you evaluate between a test point and point in a training set , the resultant gives you a coordinate of your test point in that dimension
- We get a vector after evaluating the test points for all points In our training set, and this vector is representation of test set in higher dimensional space
- Once we have that vector , we use it to perform a linear regression

➔ The vectors X are used to define a hyperplane that seperates the two different classes in your solution
➔ These vectors are used to perform linear regression.
➔ ***The vectors closest to the test point are referred to as support vectors***

How to build SVR?

- Collect a training set T = { X, Y }
- Choose a kernel and it's parameters as well as any regularization needed
- From the correlational matrix, K
- Train your machine exactly or approximately , to get contraction coefficients $\alpha = \{\alpha_i\}$
- Use these coefficients , create your estimator f(X, α, x^*) = y^*
- Next step is to choose a kernel
    - ✓ Guassian Regularization
    - ✓ Noise

Correlation Matrix

- $$K_{i,j} = \exp\left(\sum_k \theta_k \left|x_k^i - x_k^j\right|^2\right) + \epsilon\delta_{i,j}$$

The main part of the algorithm $\bar{K}\vec{\alpha} = \vec{y}$

$\vec{y}$ is the vector of values corresponding to your training set,

$\bar{K}$ is your correlation matrix

$\vec{\alpha}$ is a set of unknowns we need to solve for.

- $$\vec{\alpha} = \bar{K}^{-1}\vec{y}$$
  - Once $\vec{\alpha}$ parameters known - form the estimator
  - we use the coefficients we found during the optimization step and the kernel we started off with.
  - To estimate the value y* for a test point, $\vec{x^*}$ - compute the correlation vector $\vec{k}$,
  - $y^* = \vec{\alpha} \cdot \vec{k}$

$$k_i = \exp\left(\sum_k \theta_k \left|x_k^i - x_k^\star\right|^2\right)$$

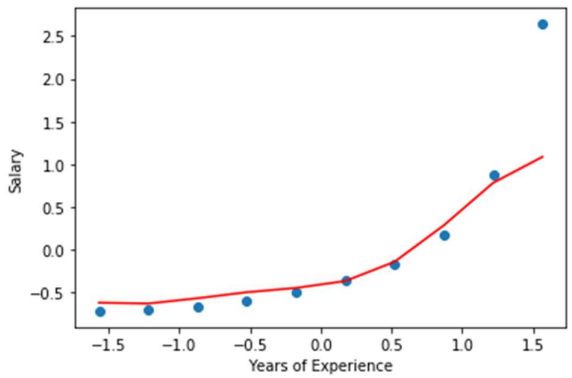
Brief Overview about SVR :

SVR has different regression goal compared to linear regression. In linear regression we are trying to minimize the error between the prediction and data. In SVR our goal is to make sure that errors don't exceed the threshold

//code

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt


dataset= pd.read_csv('Position_Salaries.csv')

X = dataset.iloc[:, 1:2 ].values

y = dataset.iloc[:, 2:3].values
```

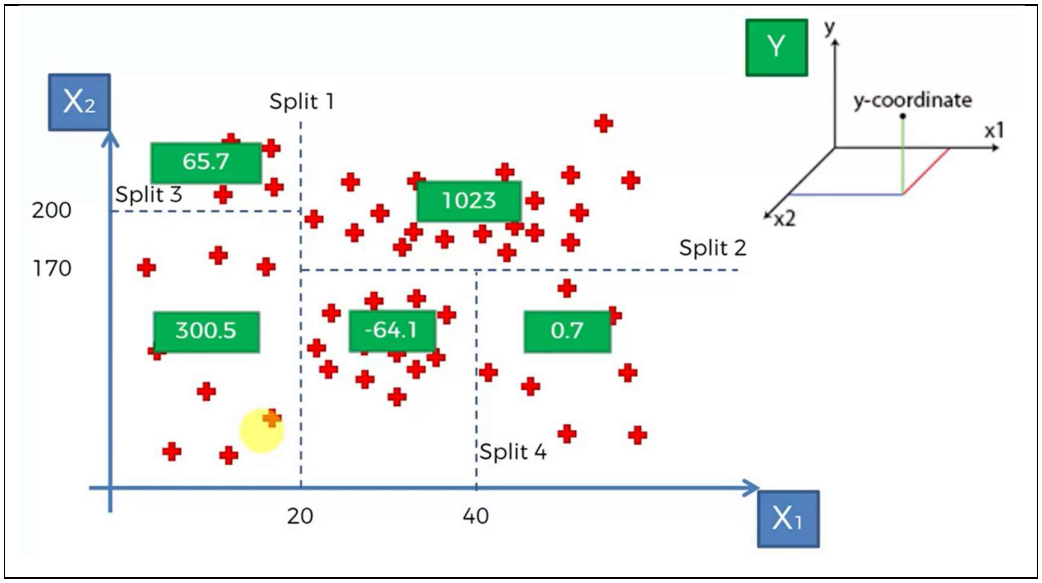
```
#feature scaling ( Most of the algorithms use feauture scaling
#inbuit but svr class is less oftenly used so we need to
# feauture scale our data manually)
from sklearn.preprocessing import StandardScaler
SC_X = StandardScaler()
SC_y = StandardScaler()
X= SC_X.fit_transform(X)
y= SC_y.fit_transform(y)

#fitting SVR to dataset
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(X,y)

#plotting svr

plt.scatter(X,y)
plt.plot(X, regressor.predict(X), color='red')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

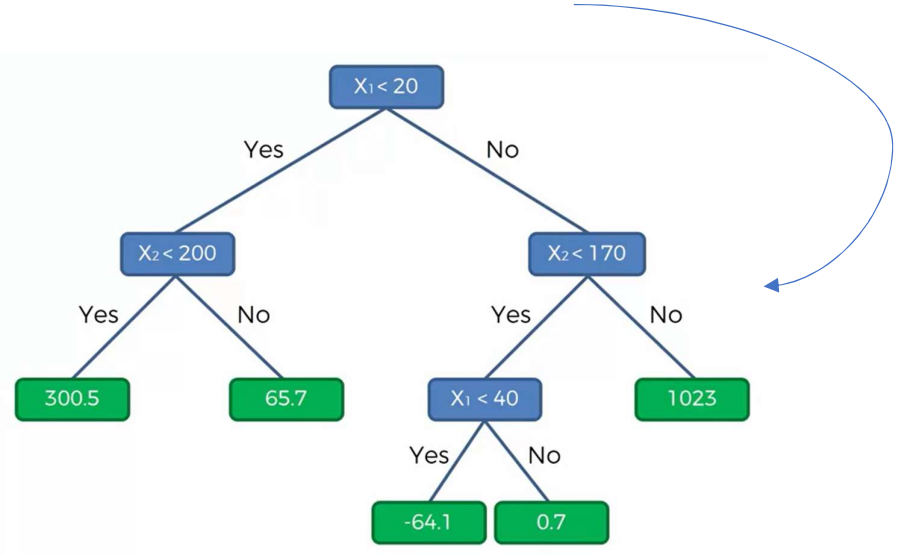



```
#since we applied feature scaling to the data, we can't simply give input as 6.5, it will give us a wrong prediction
#regressor.predict(SC_X.transform(arr))  by executing it gives us scaled output, so we need to inverse
k= SC_X.transform(np.array([[13]]))
y_pred=SC_y.inverse_transform(regressor.predict(k))
```

DECISION TREE INTUITION

- Here we have two independent variables say, x1 & x2 and a dependant variable y in other dimension
- The algorithm includes multiples splits on the independent variables so that each set upon splitting , splits into two leaves
- Example, we have 100 points , and a split happens on x=30 then the dataset is divided into two leaves each leave having 30 and 70 points, and the process continues until the no.of points in any leave must be greater than a threshold value, if less further splitting can't be done on that particular node





While we are coding we get odd results, The model is non-continous, since in the dataset {1,2,3,4,5,6,7,8,9}, we have points like above so in the intervals in between it will take the average of the points

If we give input 6 o/p = 150000

If we give input 7 o/p= 200000

If we give any value in between [6, 6.5] o/p = 150000

If we give any value in between (6.5 ,7] o/p = 200000

// code is exactly same as previous simple regression

//technique

Level	Salary
1	45000
2	50000
3	60000
4	80000
5	110000
6	150000
7	200000
8	300000
9	500000
10	1000000

RANDOM FOREST INTUITION

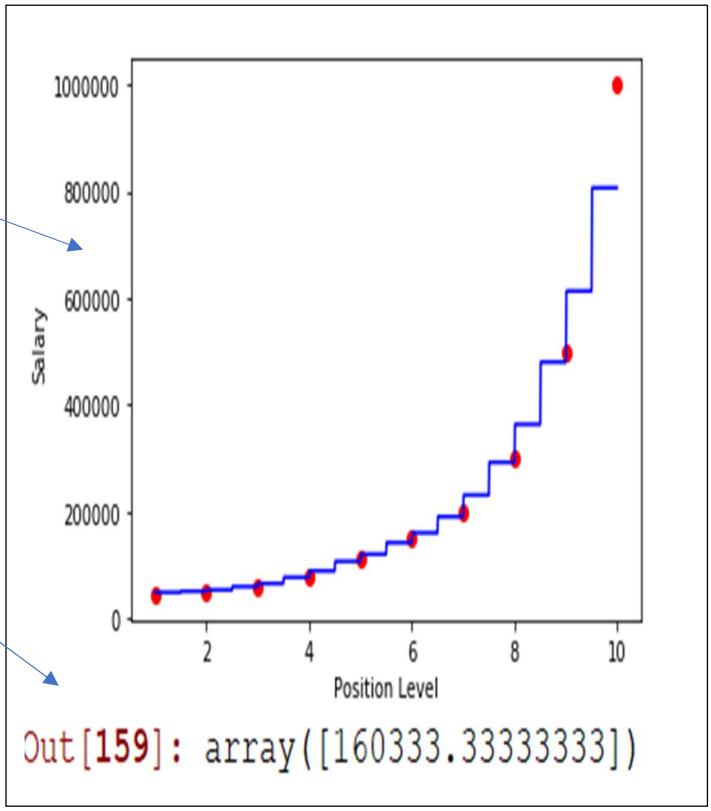
Step-1: Pick a random K data points from the training sets

Step-2: Build the decision tree associated to these K data points

Step-3: Choose the number Ntree of trees you want to build and repeat step-1 & step-2

Step-4: For a new data point , make each one of your Ntree trees predict the value of Y to for the data point in question, and assign the new data point the average across all the predicted Y values

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt


dataset= pd.read_csv('Position_Salaries.csv')

X = dataset.iloc[:, 1:2 ]

y = dataset.iloc[:, 2]


from sklearn.ensemble import RandomForestRegressor

regressor= RandomForestRegressor(n_estimators=10 , random_state=0)

regressor.fit(X,y)


X1= dataset.iloc[:, 1 ]

X_grid = np.arange(min(X1),max(X1),0.01)

X_grid = X_grid.reshape((len(X_grid),1))

plt.scatter(X,y,color='red')

plt.plot(X_grid, regressor.predict(X_grid),color='blue')

plt.xlabel('Position Level')

plt.ylabel('Salary')

plt.show()


regressor.predict(np.array([[6.5]]))
```



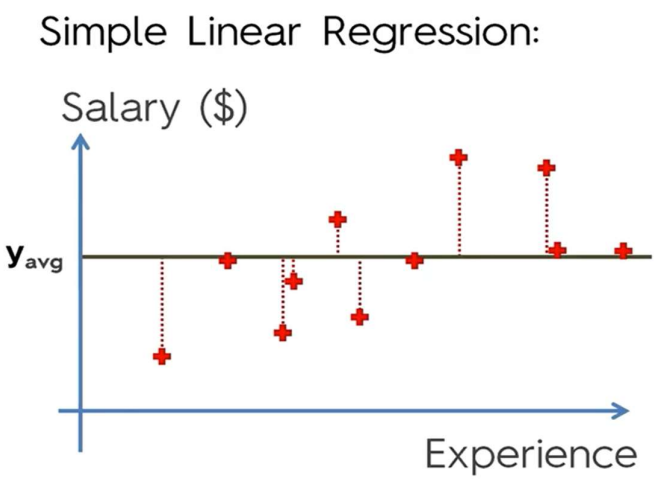
Here n_estimators is no.of trees in your model,

It's assumed that higher the no.of trees in the model, higher the steps in the graph, but it's not true

Here the steps in the graph are better chosen to predict the overall outcome

EVALUATING REGRESSION MODELS PERFORMANCE

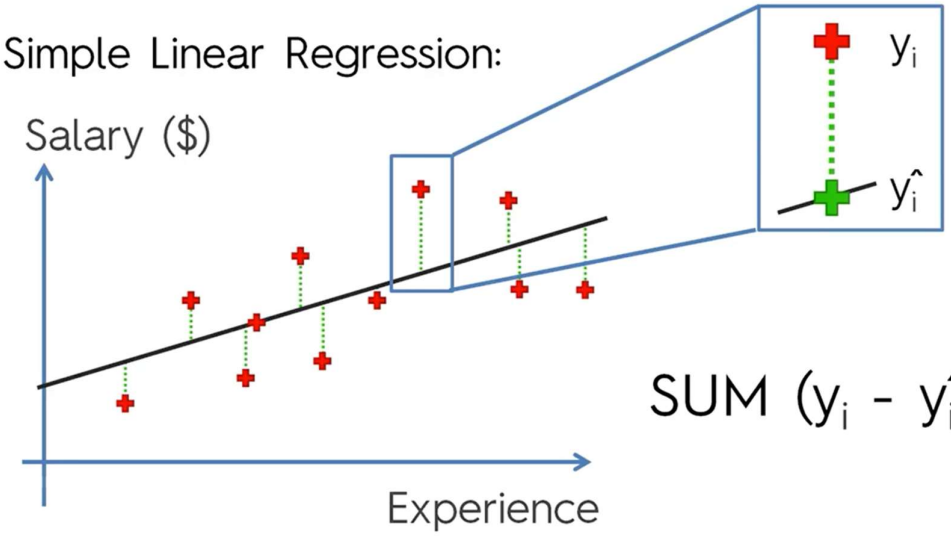
R-squared intuition:



$$SS_{res} = SUM\ (y_i - \hat{y_i})^2$$

$$SS_{tot} = SUM\ (y_i - y_{avg})^2$$

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$



$$SUM\ (y_i - \hat{y_i})^2 \rightarrow min$$

- Higher the r^2 the best is the plotted line
- r^2 can also be negative, if the line fits as worst as ever to your data then numerator has very high value , so fraction exceeds 1
- $r^2 = 1$ happens when the line passes exactly all through the points in your dataset, hence numerator = 0
- Overall, it tells how well your model has fitted to the data than the average line that you could draw through the data
- R^2 -> Goodness of fit (Greater is better)

Disadvantages of R^2:

- R-Square value will never be decreased, let's suppose a model has a single variable ($y = b_0 + b_1X_1$)
- And now we try to add a new variable ($y = b_0 + b_1X_1 + b_2X_2$)
- Suppose adding this variable helps our model to best fit the plot to data, then R-square will increase(numerator increases) else if that variable is making our model even worse then we try to remove that variable from the model hence R-square remains unchanged, So R-square is never going to decrease
- Adding a new variable has always a slight correlation between dependent and independent variable
- To overcome this problem, we introduce adjusted R-square method

Adjusted R-square intuition:

$$R^2 = 1 - \left((1-R^2) * (n-1)/(n-p-1)\right)$$

P – number of regressors

N – sample size

Understanding our models even better using Adjusted R-square:

1.

```
Call:
lm(formula = Profit ~ R.D.Spend + Administration + Marketing.Spend +
    State, data = dataset)

Residuals:
   Min     1Q Median     3Q    Max
-33504  -4736     90   6672  17338

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)      5.008e+04  6.953e+03   7.204 5.76e-09 ***
R.D.Spend        8.060e-01  4.641e-02  17.369  < 2e-16 ***
Administration  -2.700e-02  5.223e-02  -0.517    0.608
Marketing.Spend  2.698e-02  1.714e-02   1.574    0.123
State2           4.189e+01  3.256e+03   0.013    0.990
State3           2.407e+02  3.339e+03   0.072    0.943
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9439 on 44 degrees of freedom
Multiple R-squared:  0.9508,    Adjusted R-squared:  0.9452
F-statistic: 169.9 on 5 and 44 DF,  p-value: < 2.2e-16
```

2.

```
Call:
lm(formula = Profit ~ R.D.Spend + Administration + Marketing.Spend,
    data = dataset)

Residuals:
   Min     1Q Median     3Q    Max
-33534  -4795     63   6606  17275

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)      5.012e+04  6.572e+03   7.626 1.06e-09 ***
R.D.Spend        8.057e-01  4.515e-02  17.846  < 2e-16 ***
Administration  -2.682e-02  5.103e-02  -0.526    0.602
Marketing.Spend  2.723e-02  1.645e-02   1.655    0.105
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9232 on 46 degrees of freedom
Multiple R-squared:  0.9507,    Adjusted R-squared:  0.9475
F-statistic:   296 on 3 and 46 DF,  p-value: < 2.2e-16
```

3.

```
lm(formula = Profit ~ R.D.Spend + Marketing.Spend, data = dataset)

Residuals:
   Min     1Q Median     3Q    Max
-33645  -4632   -414   6484  17097

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     4.698e+04  2.690e+03  17.464   <2e-16 ***
R.D.Spend       7.966e-01  4.135e-02  19.266   <2e-16 ***
Marketing.Spend 2.991e-02  1.552e-02   1.927     0.06 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9161 on 47 degrees of freedom
Multiple R-squared:  0.9505,    Adjusted R-squared:  0.9483
```

4.

```
Call:
lm(formula = Profit ~ R.D.Spend, data = dataset)

Residuals:
   Min     1Q Median     3Q    Max
-34351  -4626   -375   6249  17188

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 4.903e+04  2.538e+03   19.32   <2e-16 ***
R.D.Spend   8.543e-01  2.931e-02   29.15   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9416 on 48 degrees of freedom
Multiple R-squared:  0.9465,    Adjusted R-squared:  0.9454
F-statistic: 849.8 on 1 and 48 DF,  p-value: < 2.2e-16
```

Explanation:

Look at

Image	*R-square*	*Adjusted R-square*
1	0.9508	0.9452
2	0.9507	0.9475
3	0.9505	0.9483
4	0.9465	0.9454

Number of variables Increasing

It implies that if we add on more and more variables to our model, R^2 tells us the model fits even better, but adjusted R-square tells exactly which model fits the best

In image-3 after removing a variable adjusted R^2 decreased, it means that variable plays a key role in our model

Interpreting Linear Regression Coefficients:

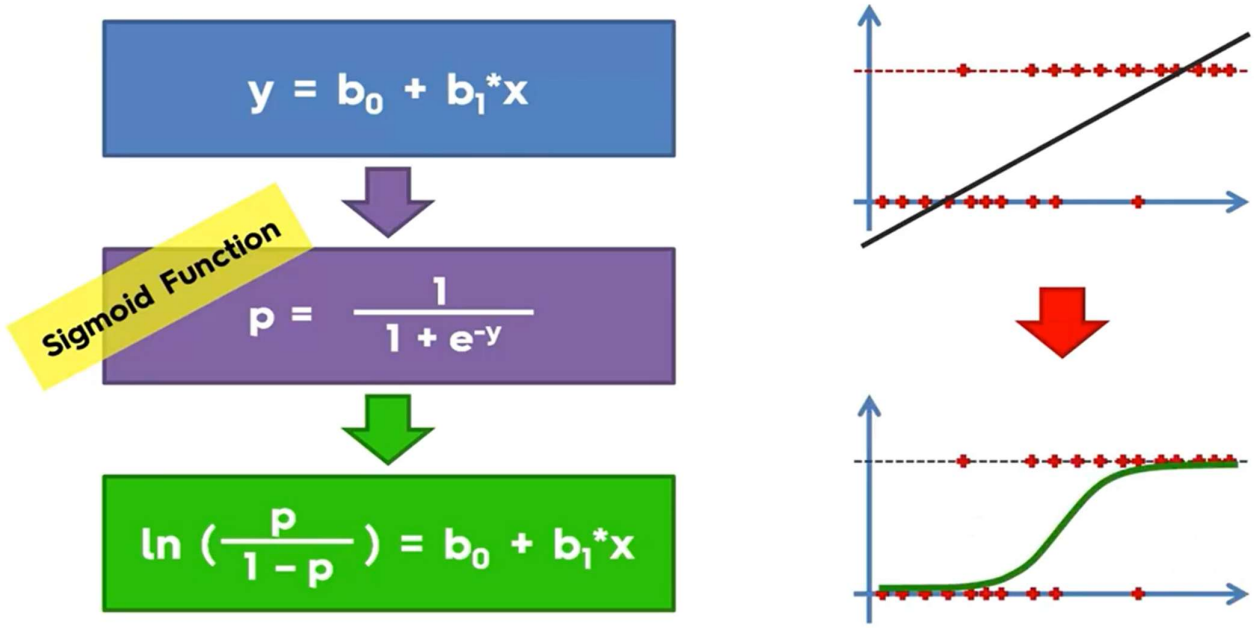
1. If sign Is positive there is a direct relationship between independent and dependent variable, else vice-versa (Example, look at image no.1 administration coefficient, the sign is negative it implies if one increases another decrease, if we increase value of R.D spend the profit will be increasing)

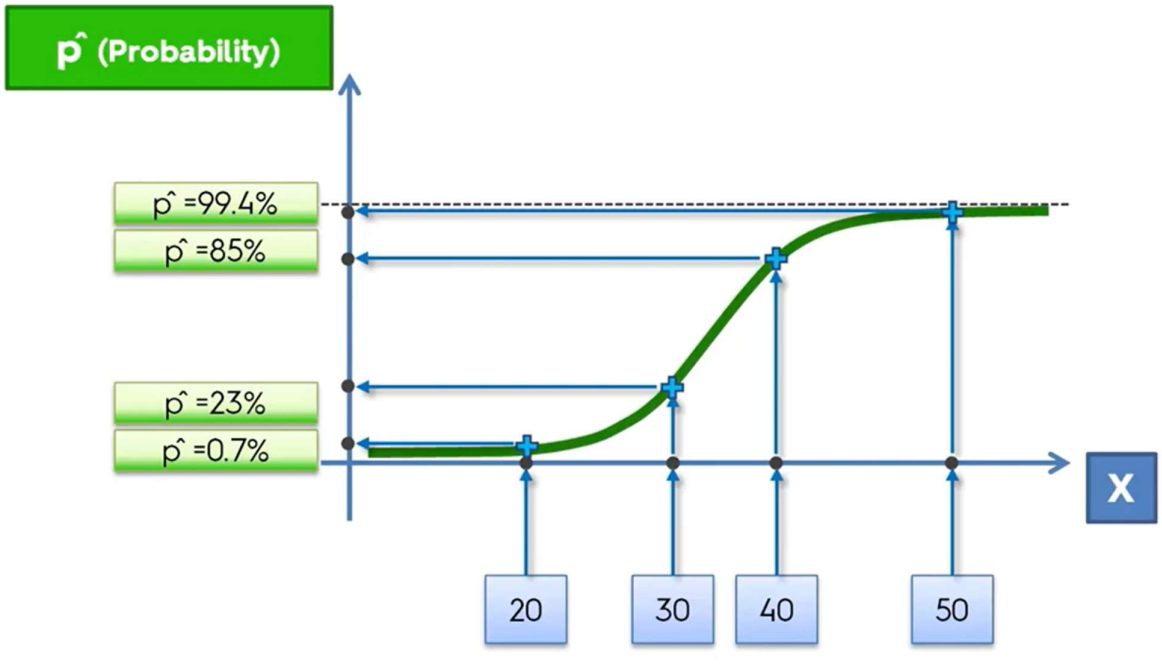
Logistic Regression

- Logistic Regression produces results in a binary format. Outcome should be categorical /discrete
- Here outcome will be only 0/1
- Sigmoid curve can convert any values from -inf to +inf into discrete values
- Shape of sigmoid curve is 'S' Shaped
- We introduce threshold value which indicates the probability of winning / loosing

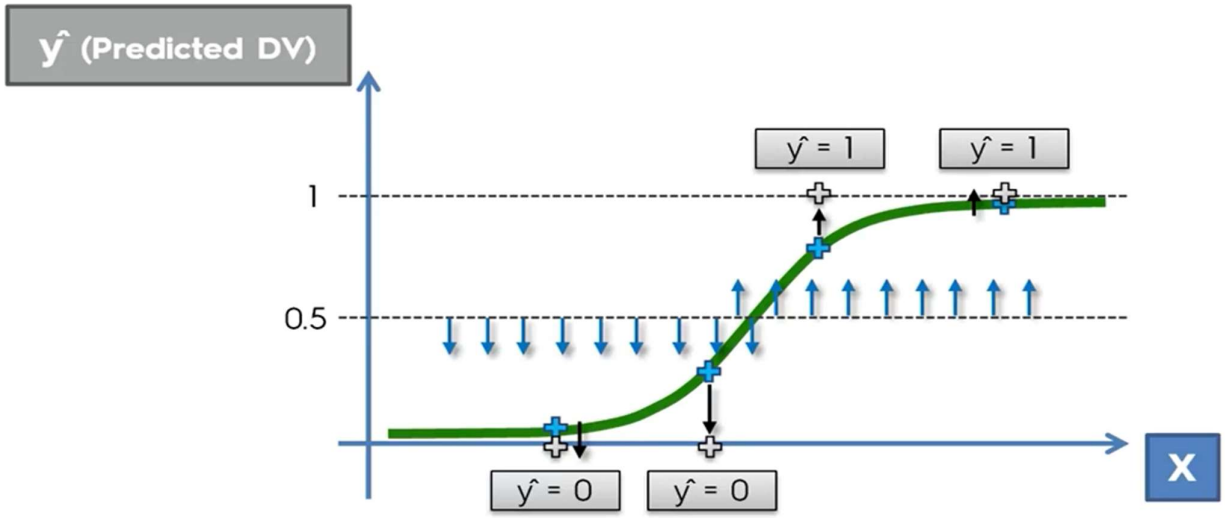
USE CASES:

1. Weather Predictions
2. Classification Problems
3. Determine illness





But here in logistic regression we are not going to predict the probability, but we are just going to predict the Y value
So, we take up a line exactly at y=0.5, and all the predictions above this line will be projected upwards and rest projected downwards




```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt


dataset= pd.read_csv('Social_Network_Ads.csv')

X= dataset.iloc[:,[2,3]].values

y= dataset.iloc[:, 4].values


from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.25,random_state =0)


from sklearn.preprocessing import StandardScaler

sc_X = StandardScaler()

sc_y = StandardScaler()

X_train = sc_X.fit_transform(X_train)

X_test = sc_X.transform(X_test)


from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state=0)

classifier.fit(X_train,y_train)

y_pred =classifier.predict(X_test)
```

#making the confusion matrix

from sklearn.metrics import confusion_matrix

cm= confusion_matrix(y_test,y_pred)

cm - NumPy array

	0	1
0	65	3
1	8	24

Confusion matrix

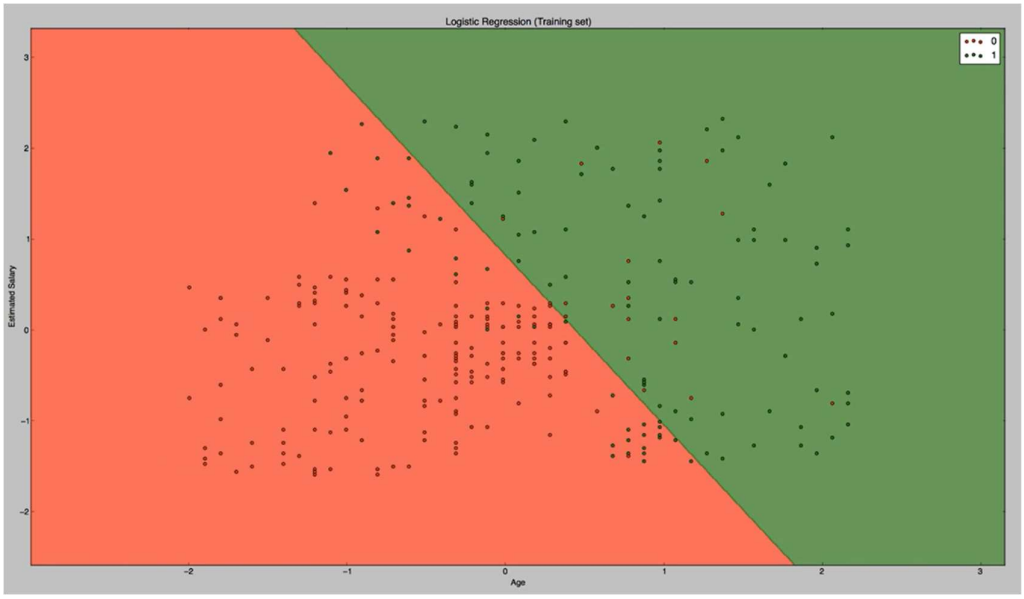
No.of correct predictions = 65+24 = 89

No.of wrong predictions = 8+3 = 11

Accuracy= 0.89

```
# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

O/p:

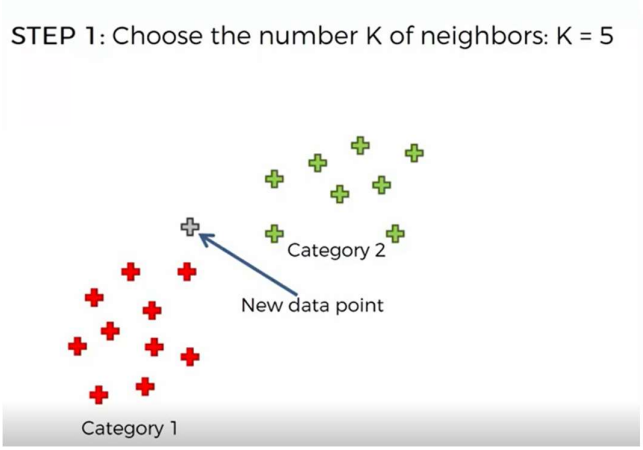


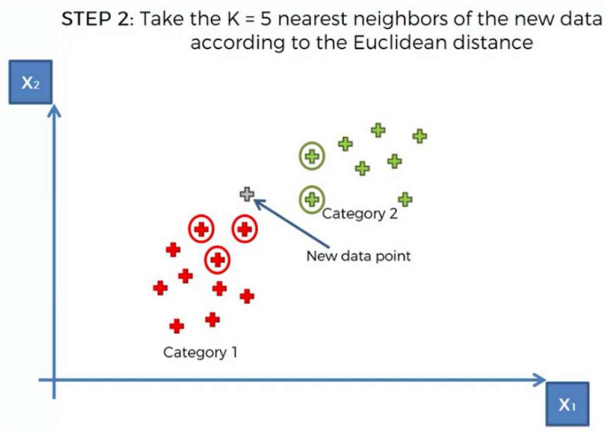
K-NN INTUITION

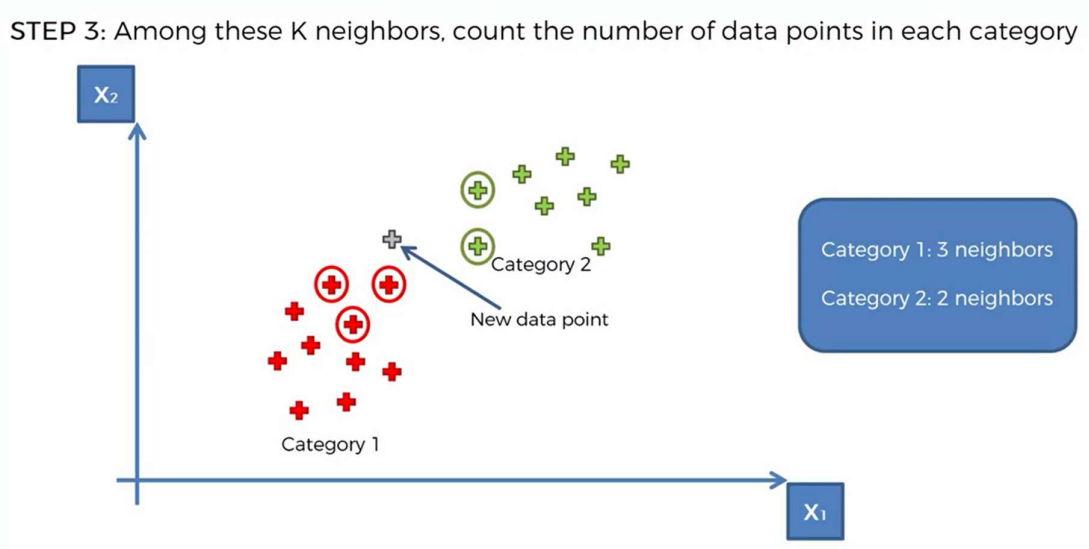
Algorithm:

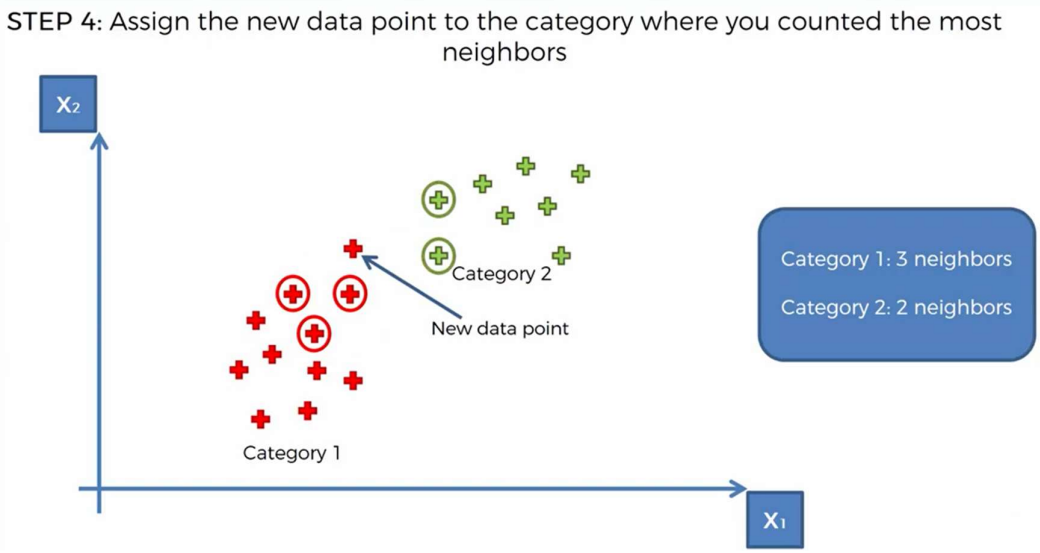
1. Choose the number of K- Neighbours
2. Take the K nearest neighbours of the new data point, according to the Euclidean distance
3. Among the K neighbours, count the number of data points in each category
4. Assign the new data point to the category where you counted the most neighbours

Example:









from sklearn.preprocessing import StandardScaler

sc_X = StandardScaler()

sc_y = StandardScaler()

X_train = sc_X.fit_transform(X_train)

X_test = sc_X.fit_transform(X_test)

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors=5,metric='minkowski', p=2)

classifier.fit(X_train,y_train)

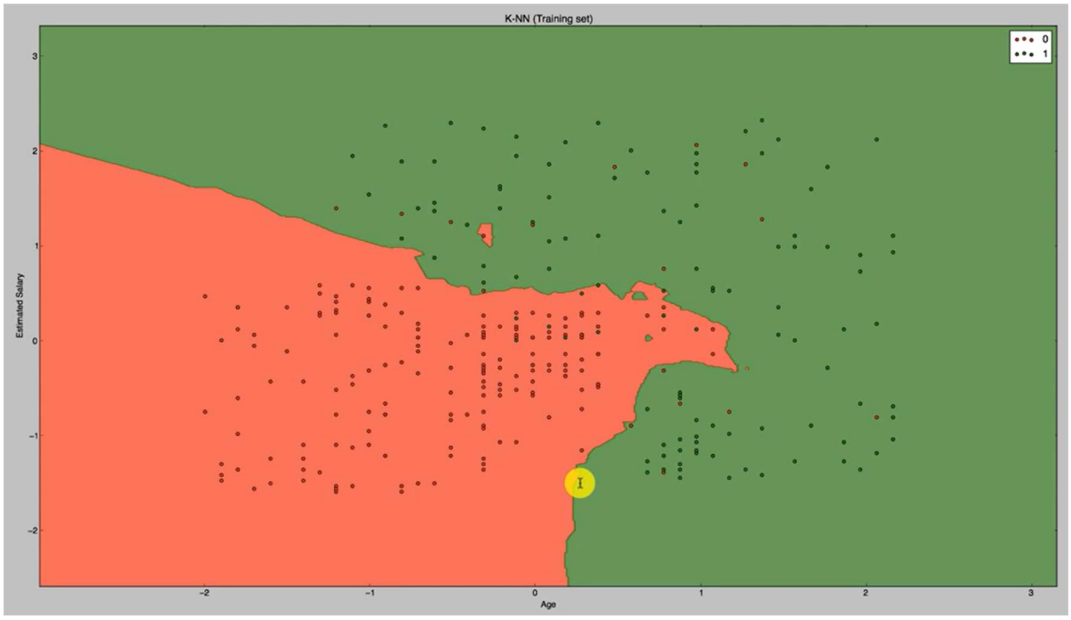
y_pred= classifier.predict(X_test)

from sklearn.metrics import confusion_matrix

cm= confusion_matrix(y_test,y_pred), //// Accuracy = 93%

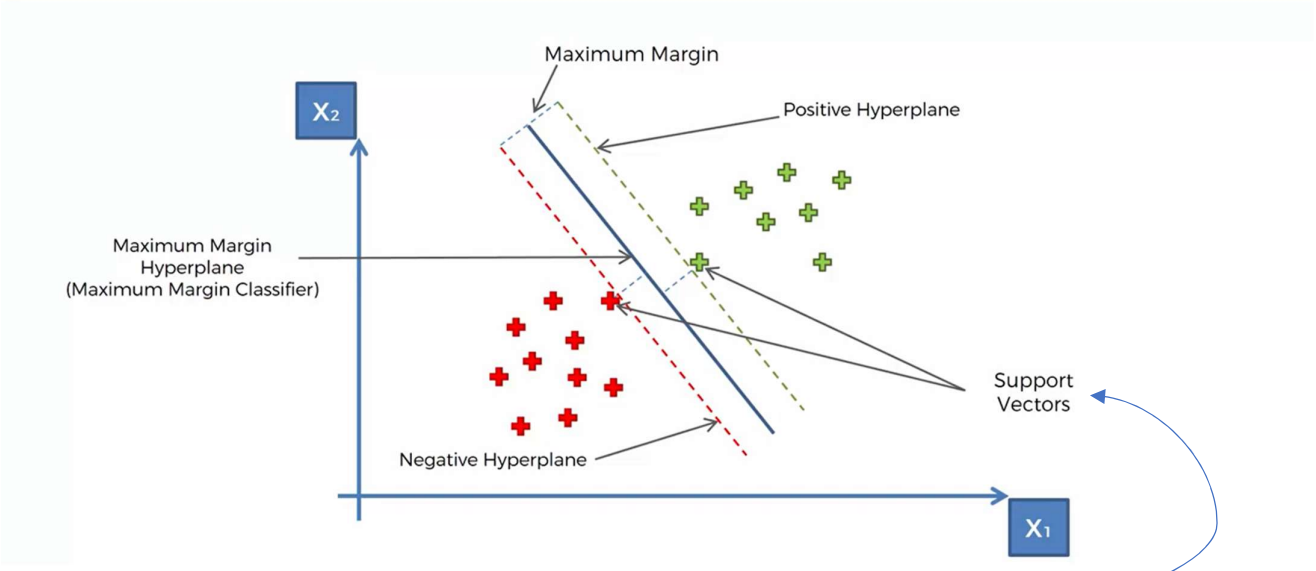
```
# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, s
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, s
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shap
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('K-NN (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

O/P:



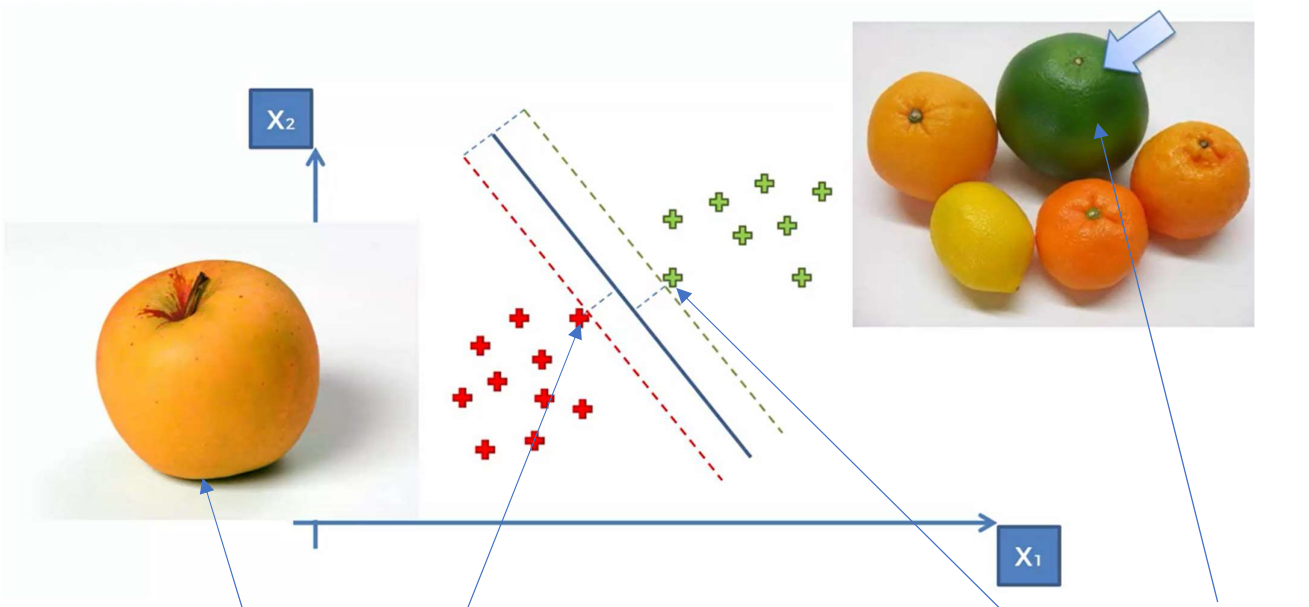
SUPPORT VECTOR MACHINE CLASSIFICATION

Hyperplanes



That hyperplane is chosen such that the sum distance from both the support vectors is maximum, And the whole plane is entirely chosen purely based on those support vectors

What's so special about SVM's?



Apple looking like an orange

(This is a support vector)

Orange which is not

Looking like orange

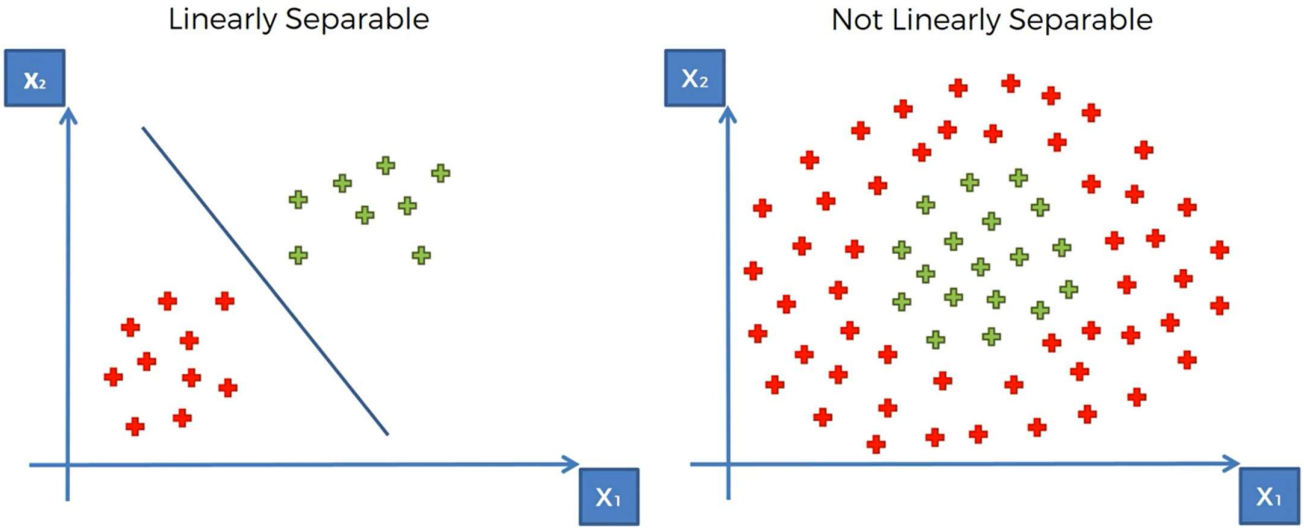
(Support vector)

These Support vectors are close to the boundary, It is a much risky type of algorithm and it produces very good results compared to non-svm classification algorithms

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].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 = 0.25, 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)

from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0 )
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

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

o/p: accuracy = 90%

KERNEL SVM



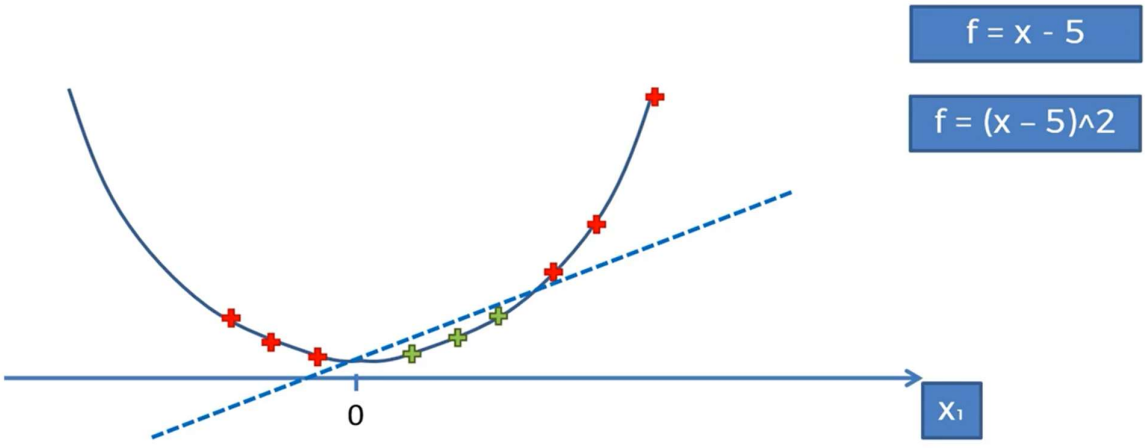
So, we are going to A higher-dimensional space and we get a linearly separable data set

How, we make dataset into higher dimensional space, let's have an example,

1. We have initial data like this,

2. We applied a function y = x-5

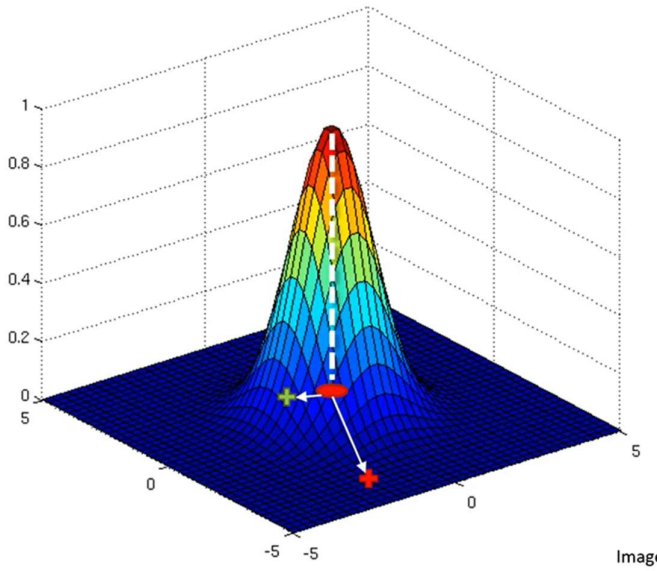
3. We applied a function y = $(x-5)^2$ * we can separate our data



Disadvantages:

Mapping to a higher dimensional space can be a highly computive-intensive, so we use kernel-trick which give exactly same results

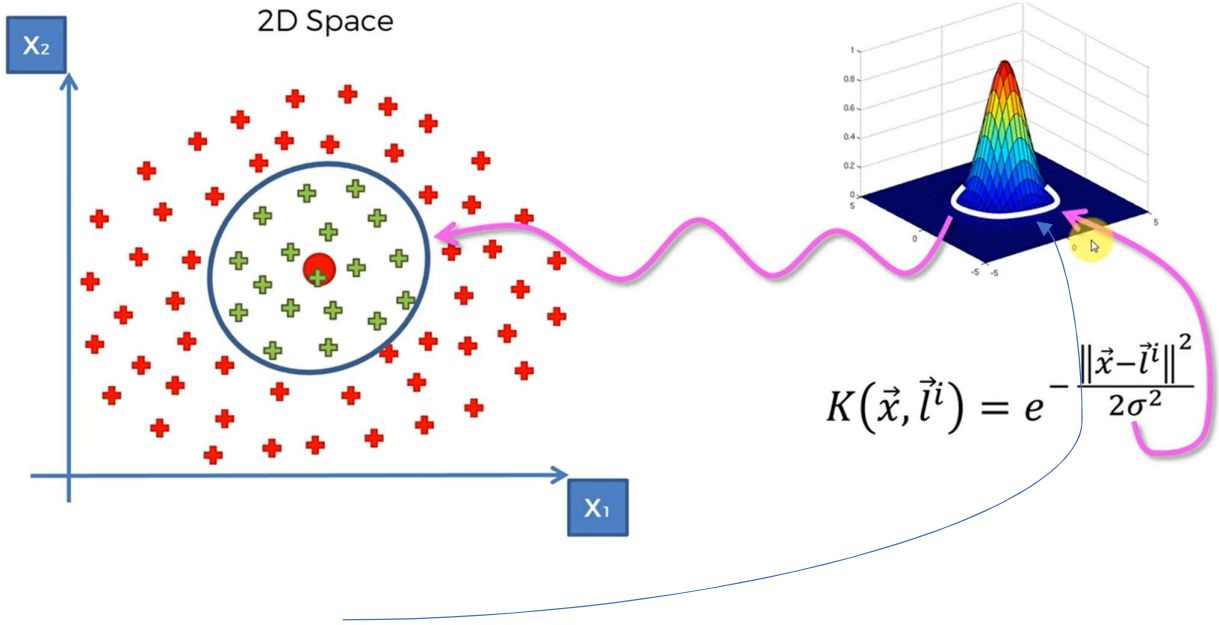
Working of kernel trick: (Gaussian RBF Kernel)



$$K(\vec{x}, \vec{l}^i) = e^{-\frac{\|\vec{x}-\vec{l}^i\|^2}{2\sigma^2}}$$

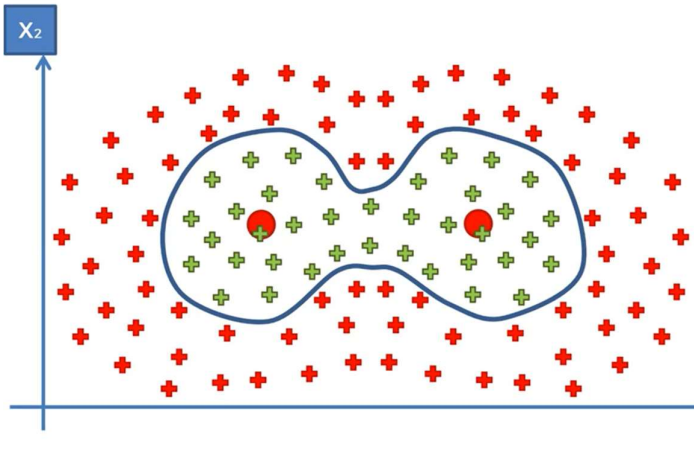
Image source: http://www.cs.toronto.edu/~duvenaud/cookbook/ir

$(x-l)^2$ represents distance of the point from centre, the resultant value is plotted above the XY plane



If sigma increases, the circumference increases (Directly proportional)

More complex example,



$$K(\vec{x}, \vec{l}^1) + K(\vec{x}, \vec{l}^2)$$

(Simplified Formula)

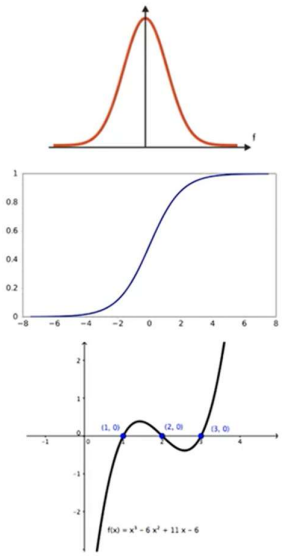
Green when:

$$K(\vec{x}, \vec{l}^1) + K(\vec{x}, \vec{l}^2) > 0$$

Red when:

$$K(\vec{x}, \vec{l}^1) + K(\vec{x}, \vec{l}^2) = 0$$

TYPES OF KERNEL FUNCTIONS:



Gaussian RBF Kernel

$$K(\vec{x}, \vec{l}^i) = e^{-\frac{\|\vec{x}-\vec{l}^i\|^2}{2\sigma^2}}$$

Sigmoid Kernel

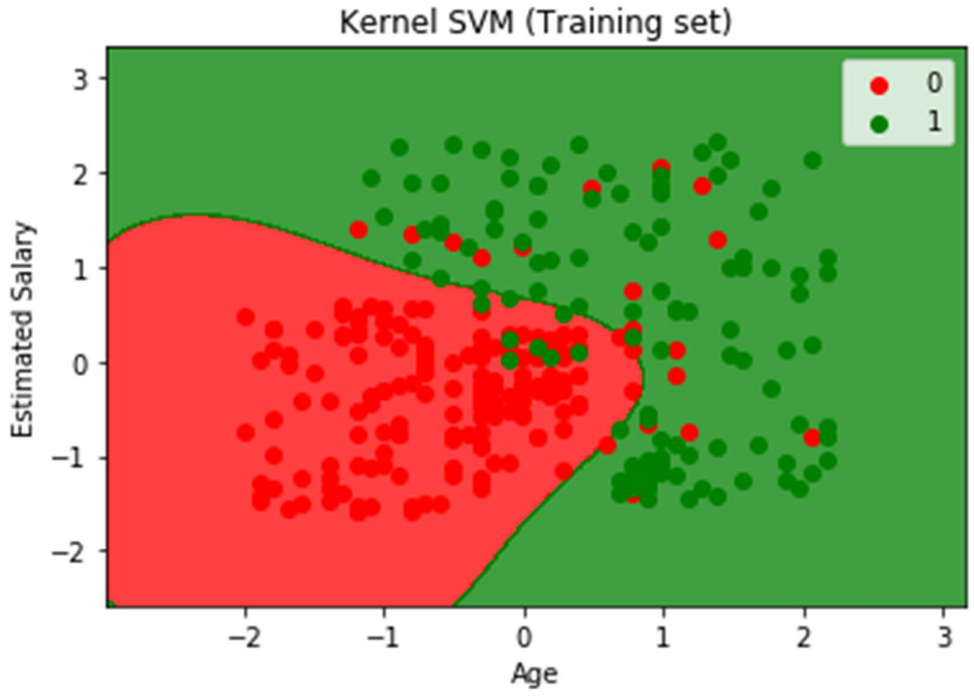
$$K(X, Y) = \tanh(\gamma \cdot X^T Y + r)$$

Polynomial Kernel

$$K(X, Y) = (\gamma \cdot X^T Y + r)^d, \gamma > 0$$

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].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 = 0.25, 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)

from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf',random_state = 0)
classifier.fit(X_train,y_train)
```

```
y_pred= classifier.predict(X_test)


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


from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Kernel SVM (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



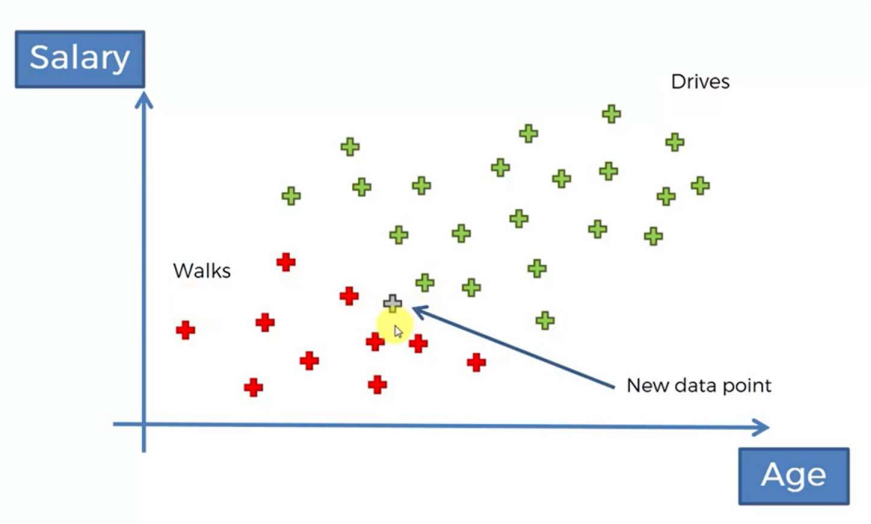
Bayes Theorem

Mach1: 30 wrenches / hr
Mach2: 20 wrenches / hr
Out of all produced parts:
We can SEE that 1% are defective
Out of all defective parts:
We can SEE that 50% came from mach1
And 50% came from mach2
Question:
What is the probability that a part produced by mach2 is defective = ?

-> P(Mach2) = 20/50 = 0.4
-> P(Defect) = 1%
-> P(Mach2 | Defect) = 50%
-> P(Defect | Mach2) = ?

$$P(Defect \mid Mach2) = \frac{P(Mach2 \mid Defect) * P(Defect)}{P(Mach2)}$$

Classifier Intuition:



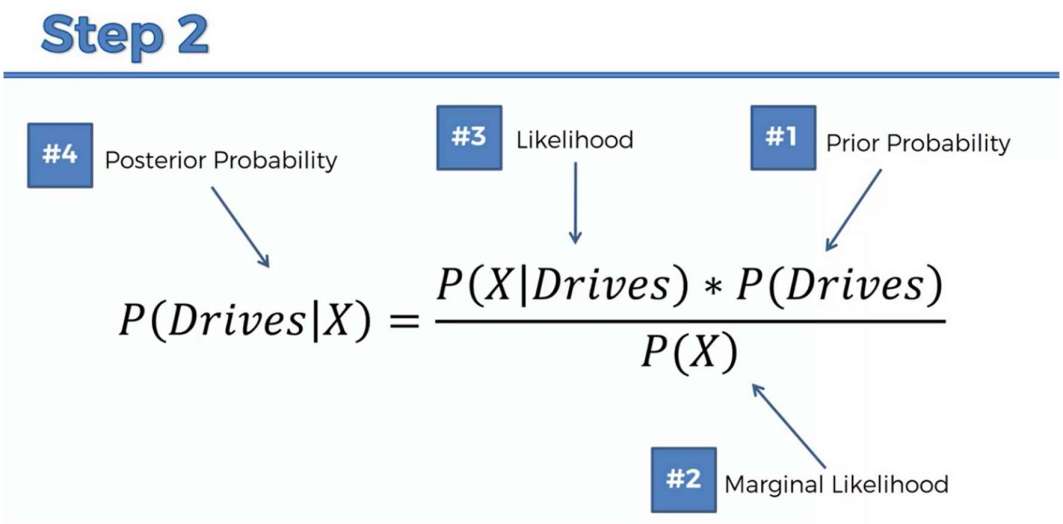
Here assume X is features of the new data point

Step 1

#4 Posterior Probability
#3 Likelihood
#1 Prior Probability

$$P(Walks|X) = \frac{P(X|Walks) * P(Walks)}{P(X)}$$

#2 Marginal Likelihood

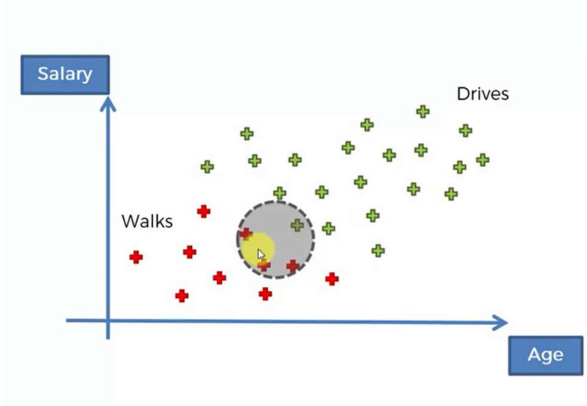


Step-3:

$$P(Walks|X)\ v.s.\ P(Drives|X)$$

Step -1: P(walks/X)

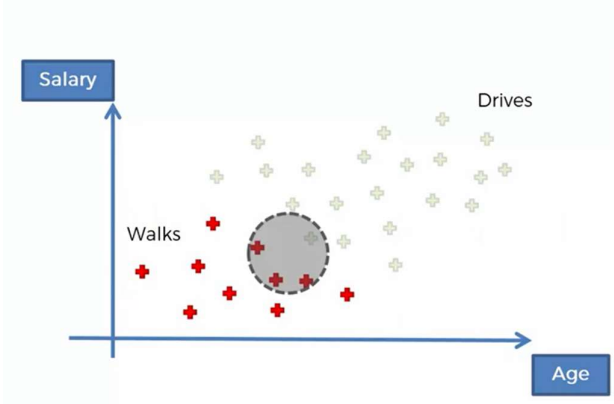
1. We can get it easily = 10/30 (no.of red points / total)
2. How to select marginal likelihood?
   A. We can select a radius of our choice, we can select less / more radius. Now we look all the points inside the circle



#2. P(X)

$$P(X) = \frac{Number\ of\ Similar\ Observations}{Total\ Observations}$$

$$P(X) = \frac{4}{30}$$



#3. P(X|Walks)

$$P(X|Walks) = \frac{Number\ of\ Similar\ Observations\ Among\ those\ who\ Walk}{Total\ number\ of\ Walkers}$$

$$P(X|Walks) = \frac{3}{10}$$

3.

4. P(Walks/X) = $\frac{\left(\frac{3}{10}\right) \times \left(\frac{10}{30}\right)}{\left(\frac{4}{30}\right)}$ = 0.75

Step-2: P(drives/X)

Similarly,

$$P(Drives/X) = \frac{\left(\frac{1}{20}\right) \times \left(\frac{20}{30}\right)}{\left(\frac{4}{30}\right)} = 0.25$$

Step-3: P(Walks/X) > P(Drives/X)

Industrial use of naïve Bayes:

- ✓ News categorisation
- ✓ Spam filtering
- ✓ Object &face recognition
- ✓ Medical Diagnosis
- ✓ Weather Prediction

Types of naïve Bayes:

- ✓ Gaussian
- ✓ Multinomial
- ✓ Bernoulli

```
from sklearn.naive_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(X_train,y_train)


y_pred= classifier.predict(X_test)


from sklearn.metrics import confusion_matrix

cm= confusion_matrix(y_pred,y_test)
```

o/p: 90%

DECISION TREE CLASSIFICATION

```
# Training the Decision Tree Classification model on the Training set
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', 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)
print(cm)
```

→ Accuracy = 91%

```
# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree Classification (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

RANDOM FOREST CLASSIFICATION

Now we are going to learn *ENSEMBLE LEARNING:*

Ensemble learning is when we take multiple machine learning algorithms and put together to make a even big machine learning algorithm, so that the final machine learning model is using leveraging many different other machine learning algorithms of same type

We are going to look at the random forest method which combines a lot of decision tree method

Step-1: Pick a random K data points from the training sets

Step-2: Build the decision tree associated to these K data points

Step-3: Choose the number Ntree of trees you want to build and repeat step-1 & step-2

Step-4: For a new data point , make each one of your Ntree trees predict the category to which the data point belongs, and assign the new data point to the category that wins the majority vote

Applications:

- ➔ Microsoft has developed a device called Kinect, where it can recognise human's body parts, such as where arms are located and etc., to play games,
- ➔ This device uses random forest classification, how humans are moving all around and with the help of grid system it identifies human's moving behaviour
- ➔ Article: "REAL-TIME HUMAN POSE RECOGNITION IN PARTS FROM SINGLE DEPTH IMAGES"

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd


# Importing the dataset

dataset = pd.read_csv('Social_Network_Ads.csv')

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, -1].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 = 0.25, 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)
```

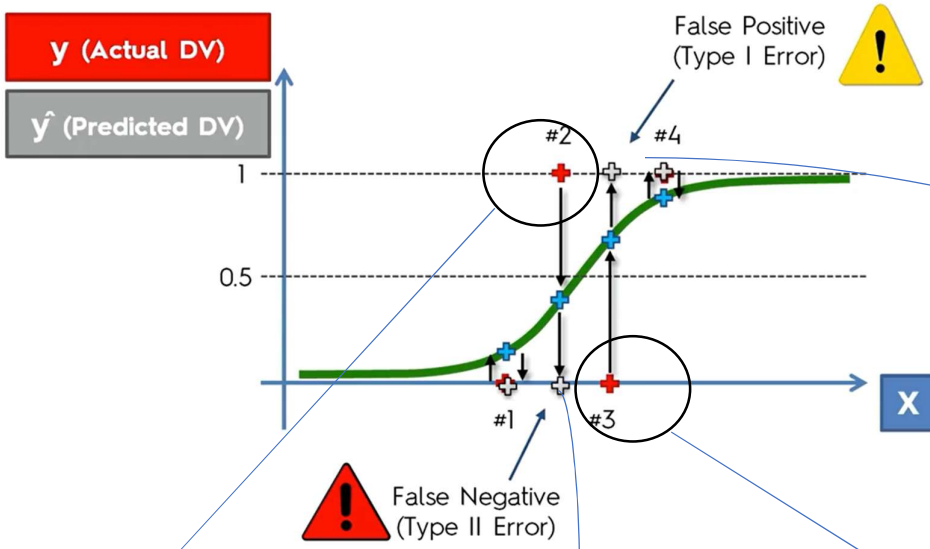
```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=10,criterion ='entropy',random_state=0 )
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

There is an over-fitting for the data set

Over-all kernel svm is best classifier for the dataset , since accuracy is more and no overfitting of data and also it has a smooth curve classifying green and red points

EVALUATING CLASSIFICATION MODELS

FALSE POSITIVES AND FALSE NEGATIVES:



#2

It's actually Yes, but it predicted as No (False negative)

It's most dangerous, Example: Tsunami is not going to come, but after two days, Tsunami came and kills thousands of people

#1, #4

Both these points are predicted correctly

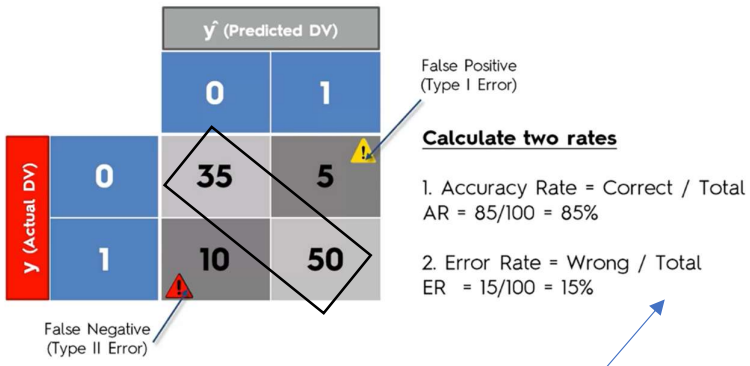
#3

It's actually No, but it predicted as Yes (False Positive)

Example: We can assume it as a warning, straighter example,

2012 end of the world

CONFUSION MATRIX:



Calculate two rates

1. Accuracy Rate = Correct / Total
AR = 85/100 = 85%

2. Error Rate = Wrong / Total
ER = 15/100 = 15%

Here,

0 – 0 Corresponds that actual is No, predicted is No

0 – 1 Corresponds that actual is No, predicted is Yes

1 – 0 Corresponds that actual is Yes, predicted is No

1 – 1 Corresponds that actual is Yes, predicted is Yes

K-MEANS CLUSTERING

Algorithm:

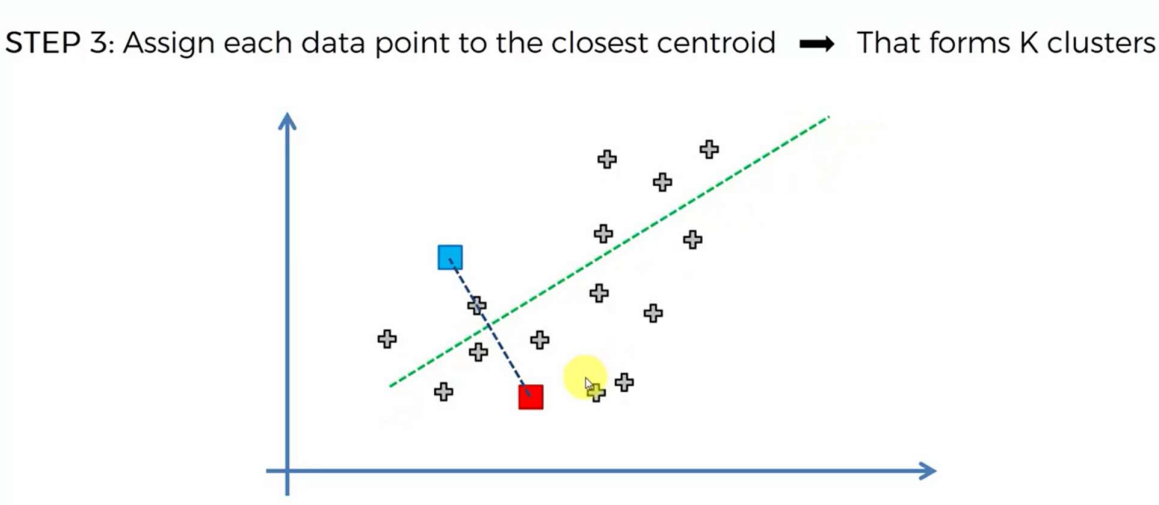
1. Choose the number of K clusters
2. Select at random K points, the centroids
3. Assign each data point to the closest centroid -> Forms K cluster
4. Compute and place the new centroid of each cluster
5. Reassign each data point to the new closest centroid, if any reassignment took place, go to step-4 , otherwise finish

Example:

Step-1: Let, k=2

Step-2: Selected two random points

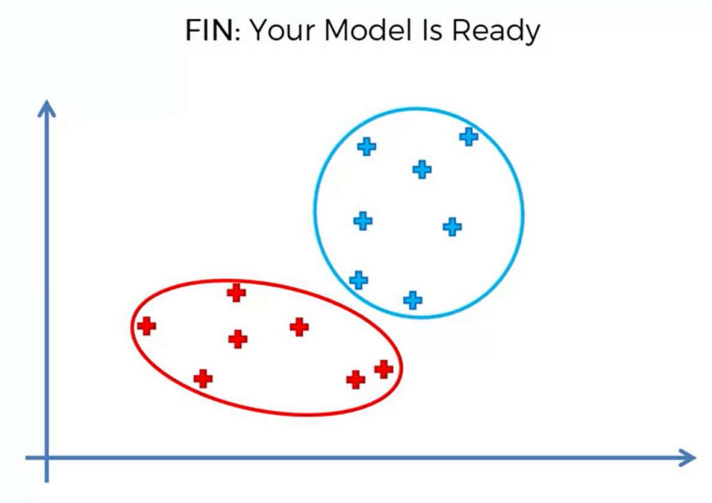
Step-3:



Points above the green line are all classified into blue points, and rest all are red

Step-4:

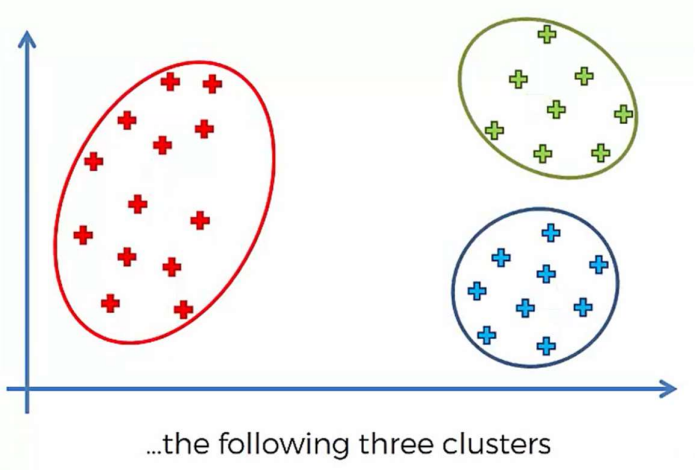
After doing a lot of work



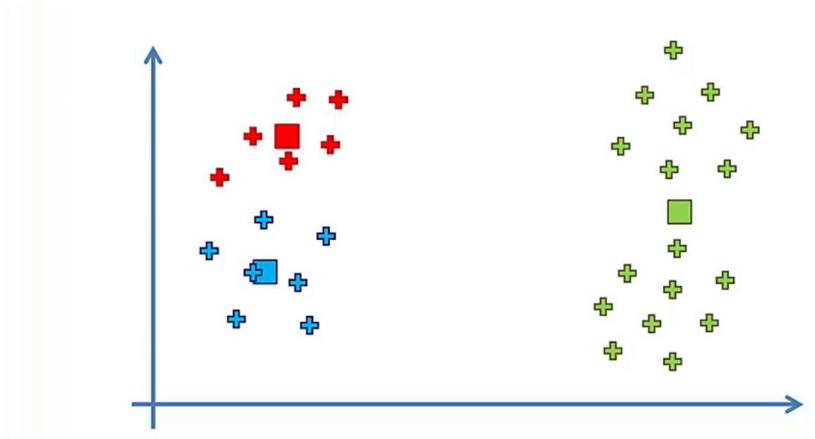
Random Initialization Trap:

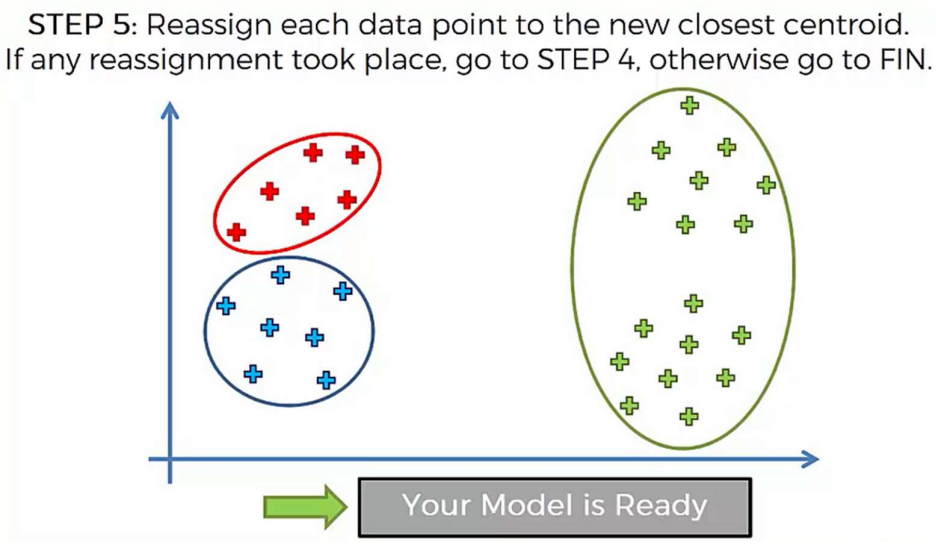
What would happen, if we had a bad random initialization

Assume, this is true dataset



What if we select centroids as below





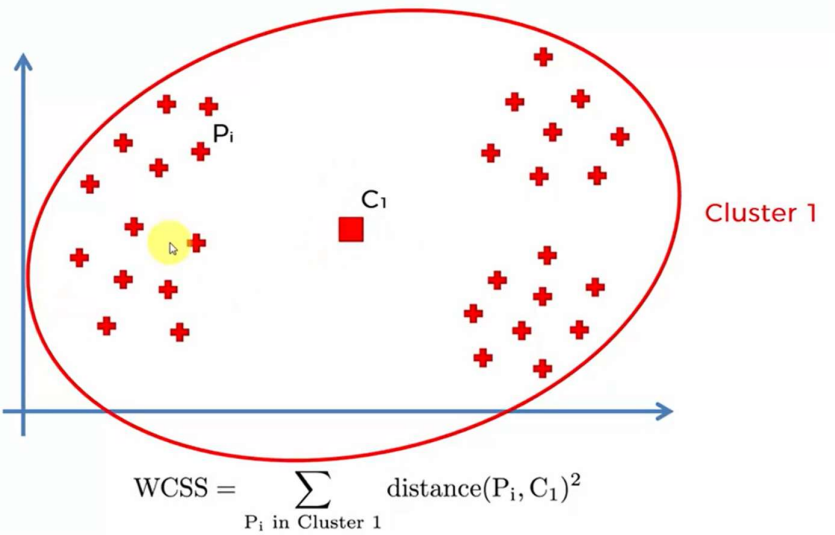
This type of random initialization will degrade the performance of the model,

What is the solution? -> **K-means++**

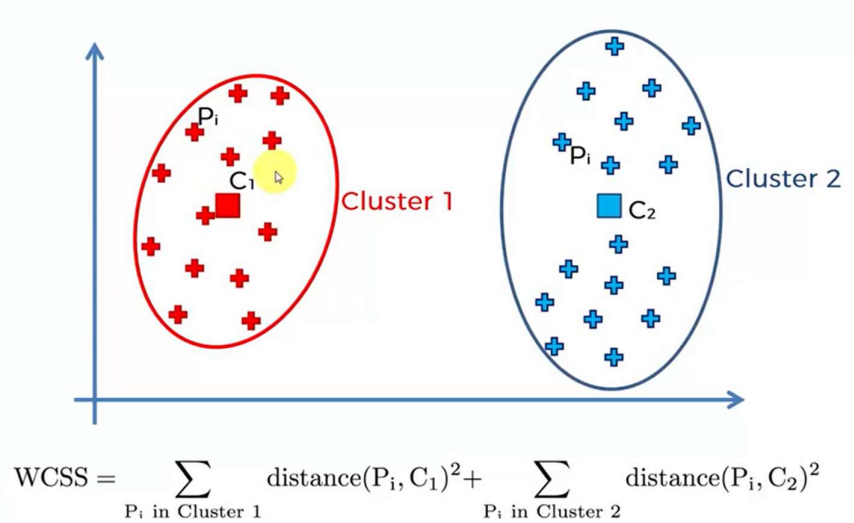
How to choose right number of clusters?

We have a variable called WCSS, based on that we decide how many clusters we can have in an optimal way

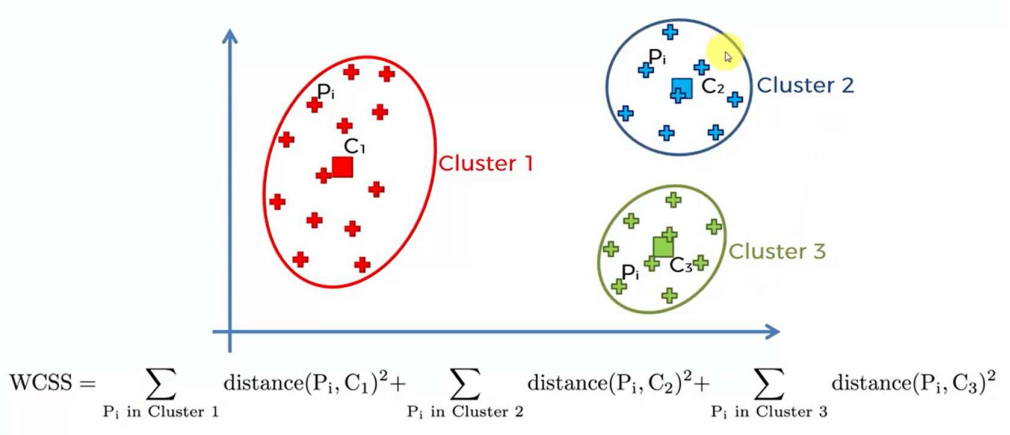
Visual representation:



Here we get a very high value of WCSS, since squared distance from each point to it's centroid will result in a very very high value



Here WCSS will be somewhat less as compared to above



In step-3 , WCSS is even less as compared to step-2

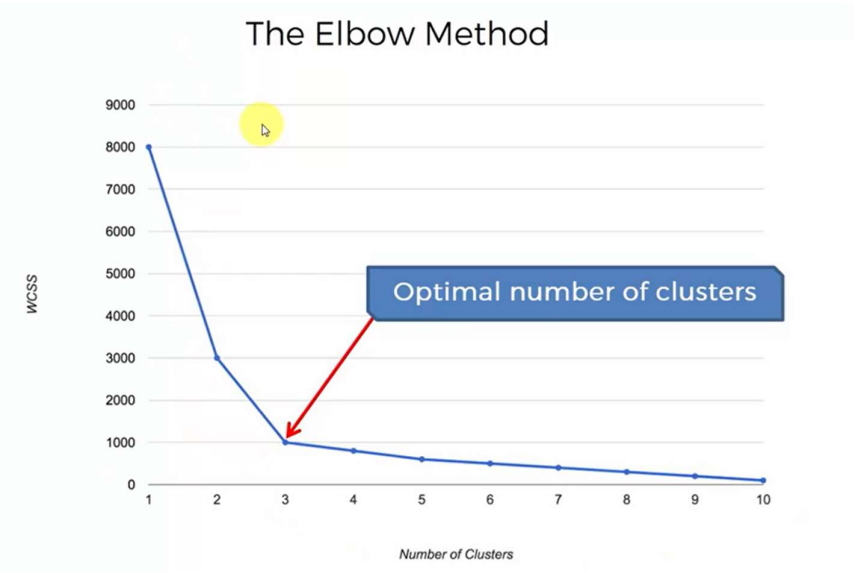
How many clusters can we put maximum in our dataset? Answer Is we can as many clusters as the number of actual data points in our dataset

Example,

If points = 50, clusters $_{MAX}$ = 50 -> Here WCSS = 0

But what is the most optimal number of clusters we can choose?

We take the help of method called “ELBOW” Method



We can observe that,

8000 – 3000 -> Huge drop in WCSS

3000 – 1000 -> Huge drop in WCSS

1000 – 800 -> Less drop in WCSS as compared to previous

But this is not we are going to strictly follow, by the end of day it’s out decision that how many clusters we are going to choose as a data scientist, Take k= 1,2,3,4….. and look what k value best fits to your model

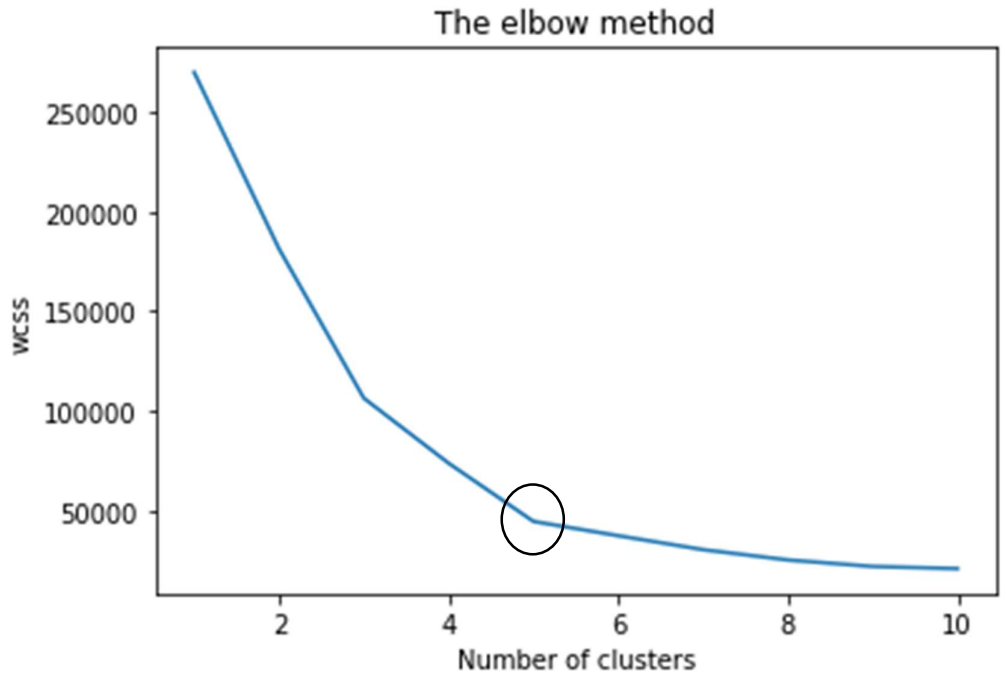
Code: (Very Important)

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd


dataset = pd.read_csv('Mall_customers.csv')


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


# now we are going to find optimal number of clusters
```

#what we are going to do is we test for cluster number = 1 to 10 (Each one), and find what is the wcss value and plot a graph between number of clusters and WCSS value, and as discussed above we try to figure out where is a sudden change in the graph and fix the number of clusters to that value,

Here inertial is an inbuilt method which computes sum of squared distances of all values from centroid to each point

```
from sklearn.cluster import KMeans


wcss=[]
for i in range(1,11):
    kmeans = KMeans(n_clusters=i, init = 'k-means++', max_iter =300, n_init=10, random_state=0)
    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()
```




```
    #applying k-means to the mall dataset
kmeans = KMeans(n_clusters=5, init = 'k-means++', max_iter =300, n_init=10, random_state=0)
#here we use fit_predict method which gives the ouput as to which cluster the client belongs
y_kmeans = kmeans.fit_predict(X)


#visualizing the clusters
plt.scatter(X[y_kmeans==0,0], X[y_kmeans==0,1], c='red', label = 'cluster-1')
plt.scatter(X[y_kmeans==1,0], X[y_kmeans==1,1], c='blue', label = 'cluster-2')
plt.scatter(X[y_kmeans==2,0], X[y_kmeans==2,1], c='green', label = 'cluster-3')
```

```
plt.scatter(X[y_kmeans==3,0], X[y_kmeans==3,1], c='cyan', label = 'cluster-4')
plt.scatter(X[y_kmeans==4,0], X[y_kmeans==4,1], c='violet', label = 'cluster-5')
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1], c='yellow', label='centroids')
plt.title('Clusters of clients')
plt.xlabel('Income')
plt.ylabel('Spending score')
plt.legend()
plt.show()
```

Note : X[y_kmeans == i, 0/1] -> for a given point first we are comparing with which cluster It belongs from [0,4]

Then 0 corresponds 1st attribute of X, 1 corresponds 2nd attribute of X

	0
39	3
40	4
41	3
42	4
43	1
44	4
45	3
46	1
47	1
48	1
49	1
50	1
51	1

Y_kmeans array

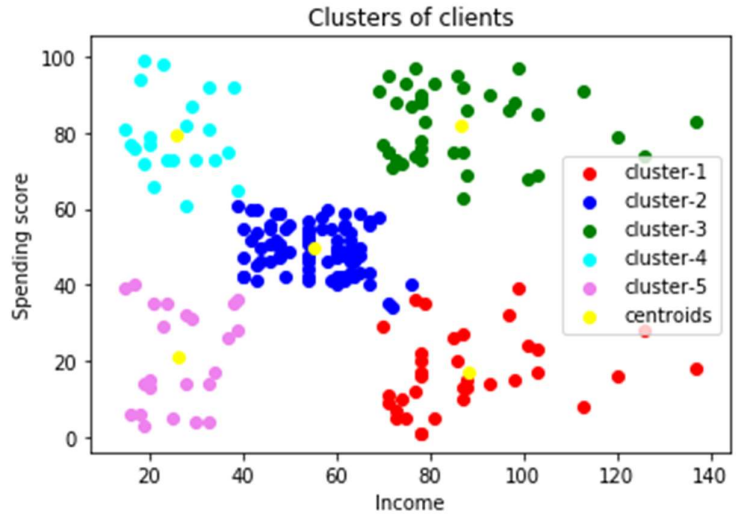
	0	1
0	15	39
1	15	81
2	16	6
3	16	77
4	17	40
5	17	76
6	18	6
7	18	94
8	19	3
9	19	72
10	19	14
11	19	99
12	20	15

X matrix

At index I, if first it checks

i^{th} datapoint belongs to which cluster,

Then it gives the index 0 or 1 of x matrix



HIERARCHICAL CLUSTERING

We can think this algorithm is a backward approach of k-means clustering,

1. Here each and every point in the dataset is assumed as an individual cluster-> n clusters (points in dataset=n)
2. Two closest clusters in the dataset forms a new cluster -> n-1 clusters
3. This process is repeated until there is only one cluster remaining

Here distance between any two clusters is calculated as

1. Closest points
2. Furthest points
3. Average distance
4. Distance between centroids

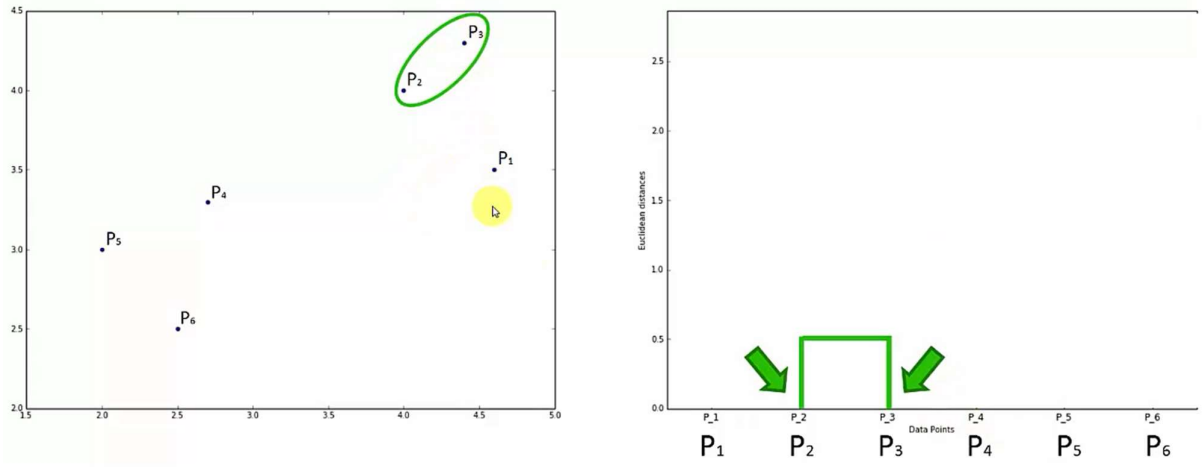
How the hierarchical clustering works is, it maintains the memory of entire process we done and this memory is stored in **"DENDOGRAMS"**

How do **DENDOGRAMS** work?

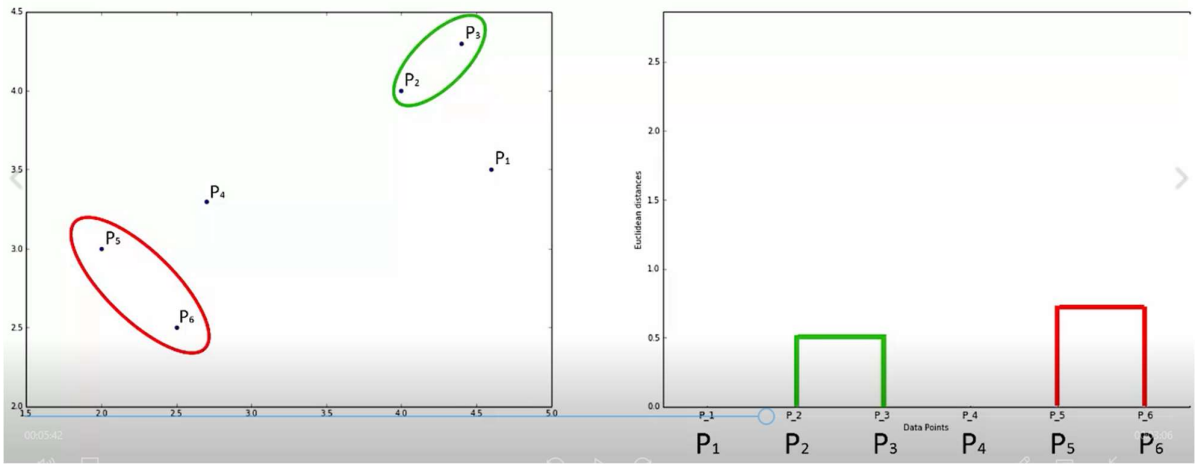
- ➔ Here on horizontal axis we put labels of points and on y-axis we put distance (Euclidian distance)
- ➔ When two clusters are closest to each other we put a horizontal bar, which connects those two points at height of Euclidian distance between those two clusters

  Example,

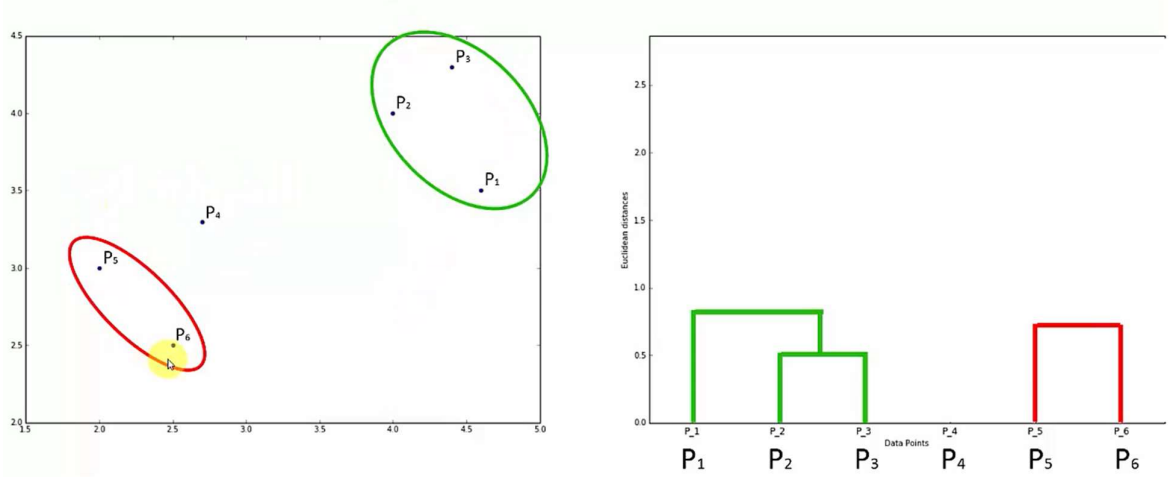
  1)



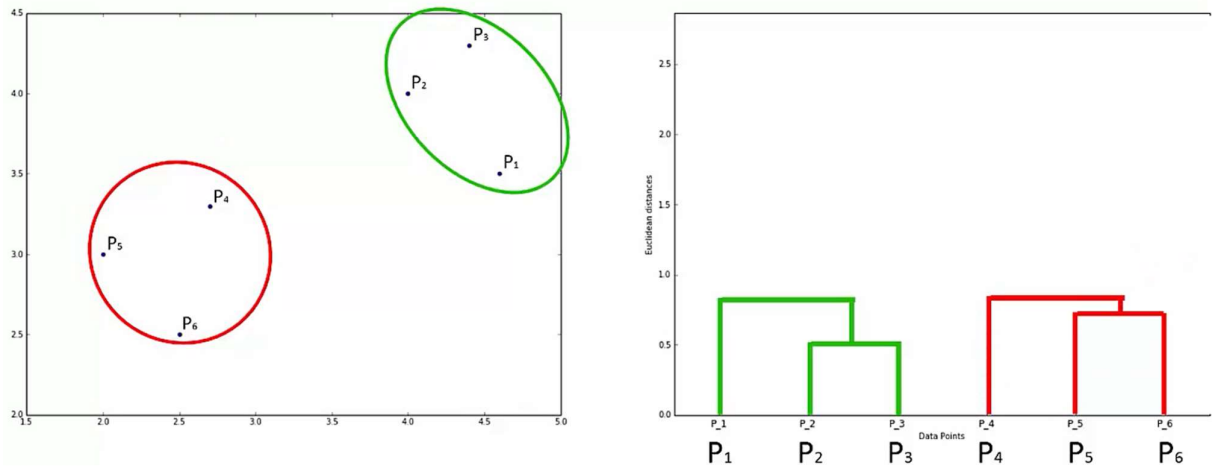
2)



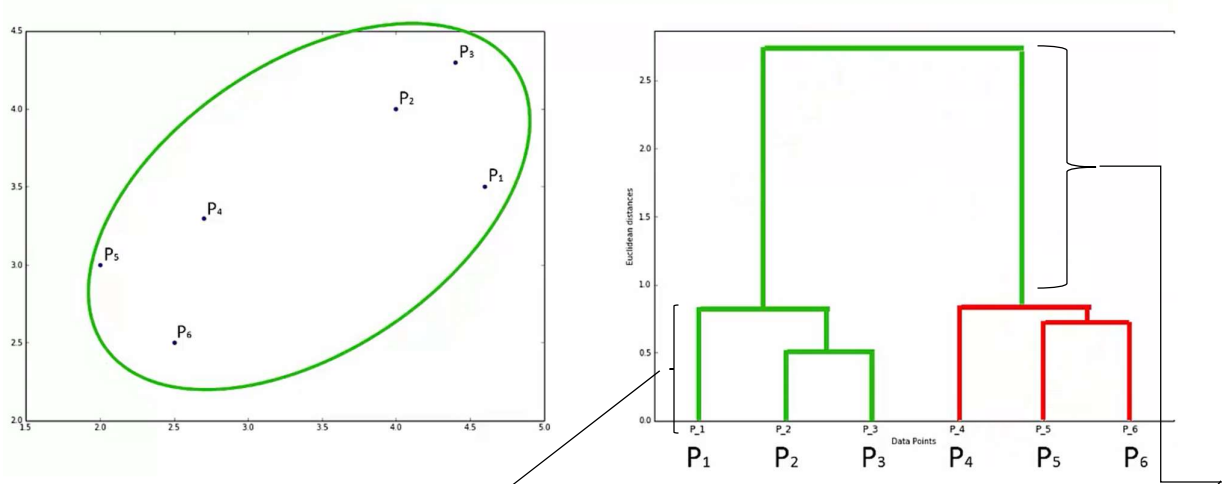
3)



4)



5)



This height is Euclidian distance between, cluster p_1 and cluster (p_2, p_3)

This height is Euclidian distance between, cluster (p_1, p_2, p_3) and cluster (p_4, p_5, p_6)

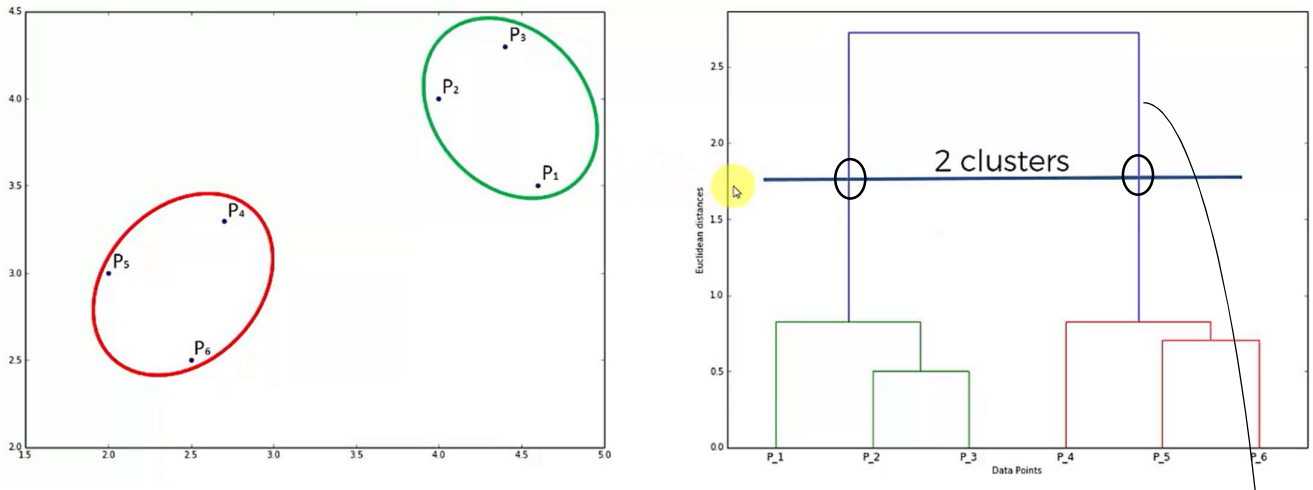
Note: Dissimilarity between any two clusters can be said by using Euclidian distance between those two clusters,

i.e., More the Euclidian distance between two clusters, More the dissimilarity between them

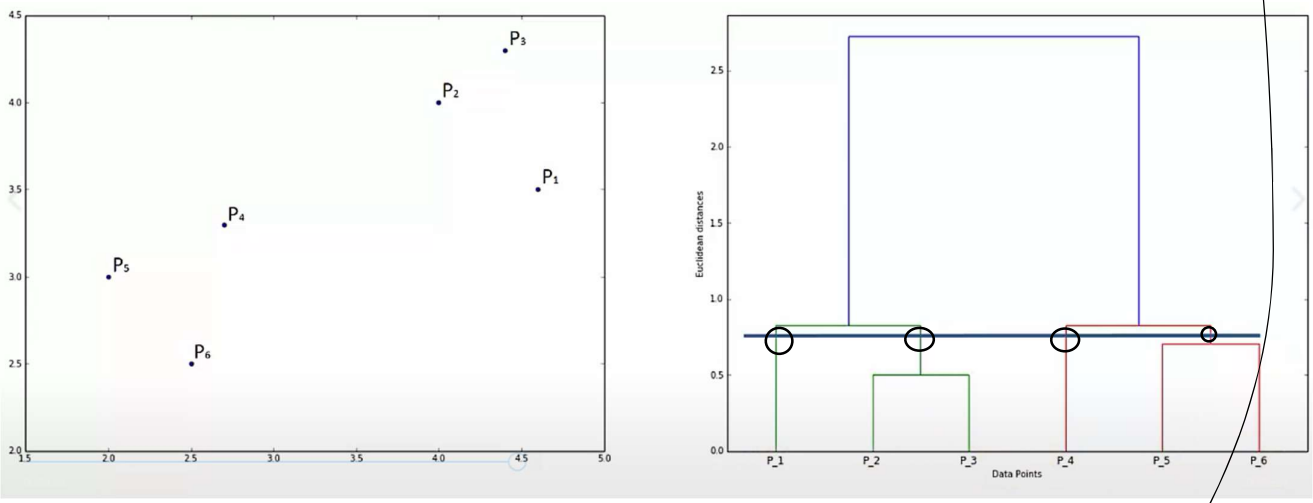
Then, how we choose how many number of clusters should we have in our model?

We put a threshold value of dissimilarity and make a line. we can simply say how many number of clusters we have just by looking into the graph and count the number of vertical lines the threshold line crosses

Example,



Clusters = $\{(p_1, p_2, p_3), (p_4, p_5, p_6)\}$



Clusters= $\{p_1, (p_2, p_3), p_4, (p_5, p_6)\}$

FINDING OPTIMAL NUMBER OF CLUSTERS

We should select the which has greatest vertical distance on the dendrogram, basically any line that will not cross any extended horizontal lines,

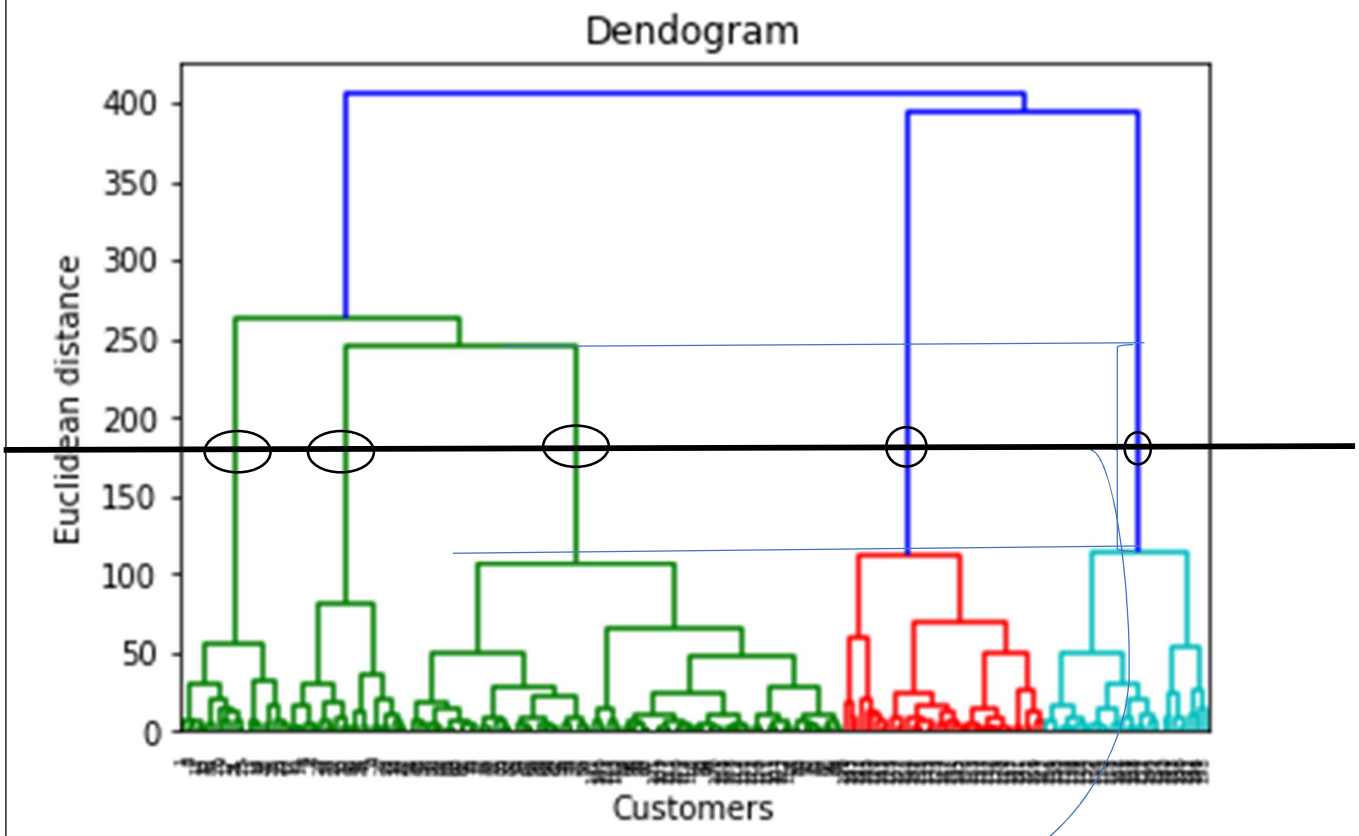
Vertical line passing through,

P_2, P_3 can be considered

P_1 can't be considered because if we extend the horizontal line connecting (p2,p3) it will cross the vertical line passing through P_1

This is the largest distance vertical line

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

dataset= pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:,[3,4]].values

#using dendrogram to find optimal number of clusters
import scipy.cluster.hierarchy as sch
dedrogram = sch.dendrogram(sch.linkage(X,method='ward'))
#ward is a method which minimizes the variance within each cluster
#it's exaclty same as kmeans cluster where we try to minimize sum
#of squares there, here we try to reduce variance with in each cluster
plt.title('Dendogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distance')
plt.show()
```



Longest vertical line, where we don't have any cross overs, therefore, clusters = 5

```
#fitting hierarchical clustering
from sklearn.cluster import AgglomerativeClustering
hc= AgglomerativeClustering(n_clusters=5,affinity = 'euclidean',linkage = 'ward')
y_hc = hc.fit_predict(X)


plt.scatter(X[y_hc==0,0],X[y_hc==0,1],color='blue',label = 'cluster-1')
plt.scatter(X[y_hc==1,0],X[y_hc==1,1],color='red',label = 'cluster-2')
plt.scatter(X[y_hc==2,0],X[y_hc==2,1],color='green',label = 'cluster-3')
plt.scatter(X[y_hc==3,0],X[y_hc==3,1],color='violet',label = 'cluster-4')
plt.scatter(X[y_hc==4,0],X[y_hc==4,1],color='orange',label = 'cluster-5')
plt.title('Clusters of customers')
plt.xlabel('Annaul income')
plt.ylabel('Spending score')
plt.legend()
plt.show()
```

Cluster-5: They have low income and they spend less -> **Sensible customers**

Cluster-4: They have low income and they spend more -> **Careless customers**

Cluster-3: They have high income and they spend more -> **Target customers**

Cluster-2: They have average income and they spend at average rate -> **standard customers**

Cluster-1: They have high income and they spend less -> **careful customers**

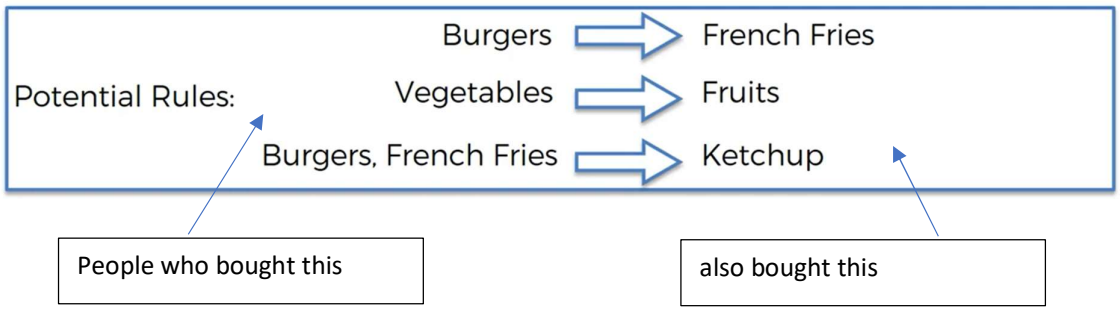
APRIORI ALGORITHM

WHAT IS ASSOCIATION RULE LEARNING ALL ABOUT?

People who bought this, also bought ….

ARL - Market Basket Optimisation

Transaction ID	Products purchased
46578	Burgers, French Fries, Vegetables
98989	Burgers, French Fries, Ketchup
71527	Vegetables, Fruits
78981	Pasta, Fruits, Butter, Vegetables
89192	Burgers, Pasta, French Fries
61557	Fruits, Orange Juice, Vegetables
87923	Burgers, French Fries, Ketchup, Mayo



Terminology

Support(M) = #users watchlists containing M / #users watchlists

Confidence (M_1 -> M_2) = #users watchlists containing M_1 & M_2 / #user watchlists containing M_1

Lift (M_1 -> M_2) = Confidence (M_1 -> M_2)/ Support(M_2)

Illustration:

Let's assume there are **100** people, suppose let's say **10** out of **100** seen movie '**A'**, Support = **10%**

Let's assume **40** who have watched movie '**B'** and out of **40** who have seen movie '**B'** only **7** people like to see movie '**A'**, Confidence = 7/40 = **17.5%**

Lift = Confidence (B -> A) / support(A) = 17.5%/10% = 1.75

Lift is simply improvement in your prediction

APRIORI ALGORITHM:

1. Set a minimum support and confidence
2. Take all the subsets in transactions having higher support than minimum support
3. Take all the rules of these subsets having higher confidence than minimum confidence
4. Sort the rules by decreasing lift

What is min_length?

- ➔ Here in our dataset, min_length=2 (Because by the definition, if we want to predict output of the customer who bought a certain item also buys another item, we should atleast have two transactions in our dataset

How to set min_support?

- ➔ Here this answer depends on your business goal (let's say product can be purchased frequently 3 or 4 times a day)
- ➔ But what's for sure is that if we manage to find some strong rules about items that are bought at least three or four times a day then by associating them and placing them together customers will be more likely to put them in their basket and therefore more of these products will be purchased and therefore the sales will increase
- ➔ Now after fixing this min_support if rules are not convinced then we can change this min_support later on
- ➔ Now we can also apply these rules for a certain period of time, let's say a week, then 3* 7 = 21 times a product is purchased a week
- ➔ Min_support = 21/7500 = 0.0028 (0.003 approximately)

What is min_confidence?

- ➔ Apriori model is implemented in a package called a Rolls, this package contain default value for min_confidence
- ➔ Here we choose min_confidence as 20%, it means our rule should be correct for at least 1 time out of 5 times
- ➔ And also support=0.003, confidence =0.2 is a good pair

How to choose lift?

- ➔ Rules that have lift at least higher than 3, if we get some rules that have lift above three then these are actually some good rules because you know the lift is a great insight of relevance & strength of rule

```
import pandas as pd

import matplotlib.pyplot as plt

import numpy as np


#our dataset has no header

dataset= pd.read_csv('Market_Basket_Optimisation.csv',header=None)


#apriori algorithm take input as lists of lists, so we

#are converting our dataframe into lists of lists

transactions=[]

for i in range(0,7501):

  st=[]

  for j in range(0,20):

    st.append(str(dataset.values[i,j]))

  transactions.append(st)

"""for i in range(0,7501):
```

transactions.append([str(dataset.values[i,j]) for j in range(0,20)])"""

#Training apriori on the dataset

from apyori import apriori

rules = apriori(transactions, min_support=0.003,min_confidence=0.2,min_lift=3,min_length=2)

#visualizing the results

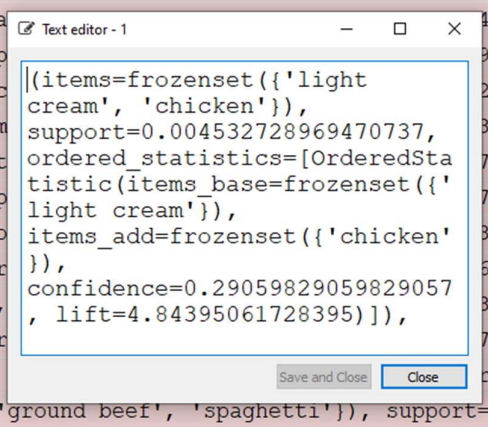
results=list(rules)

result= str(results)

frozensets=result.split("RelationRecord")

o/p:

Index	Type	Size	Value
0	str	1	[
1	str	1	(items=frozenset({'light cream', 'chicken'}), support=0.00453272896947 ...
2	str	1	(items=frozenset({'escalope', 'mushroom cream sauce'}), support=0.0057 ...
3	str	1	(items=frozenset({'escalope', 'pasta'}), support=0.005865884548726837, ...
4	str	1	(items=frozenset({'fromage bla ... 4814 ...
5	str	1	(items=frozenset({'herb & pepp ... 9786 ...
6	str	1	(items=frozenset({'tomato sauc ... 2622 ...
7	str	1	(items=frozenset({'light cream ... 3390 ...
8	str	1	(items=frozenset({'whole wheat ... 7998 ...
9	str	1	(items=frozenset({'shrimp', 'p ... 7, o ...
10	str	1	(items=frozenset({'milk', 'avo ... 3288 ...
11	str	1	(items=frozenset({'milk', 'bur ... 5219 ...
12	str	1	(items=frozenset({'chocolate', ... 3066 ...
13	str	1	(items=frozenset({'milk', 'tur ... 7339 ...
14	str	1	(items=frozenset({'tomatoes', ... t=0 ...
15	str	1	(items=frozenset({'cereals', 'ground beef', 'spaghetti'}), support=0.0 ...
16	str	1	(items=frozenset({'milk', 'ground beef', 'chicken'}), support=0.003866 ...
17	str	1	(items=frozenset({'light cream', 'nan', 'chicken'}), support=0.0045327 ...
18	str	1	(items=frozenset({'milk', 'olive oil', 'chicken'}), support=0.00359952 ...
19	str	1	(items=frozenset({'olive oil', 'chicken', 'spaghetti'}), support=0.003 ...



REINFORCMENT LEARNING

NATURAL LANGUAGE PROCESSING

NLP is concerned with the interactions between computers and human languages. NLP is used to apply ML models to text and language

Example: Teach machines to understand what is spoken and written word is the focus of NLP. Whenever you dictate a word or something else to your phone then that is converted into text

Apple Siri, Amazon Alexa etc.,

Use cases:

1. Sentiment analysis. Identifying mood within large amounts of text
2. Question answering
3. Machine translator or speech recognition system
4. Document summarization
5. Predicting genre of the book

Practical Implementation:

IMPORTING DATASET,

here we use TSV file since in the review one can add comma if we use csv(Comma separated Value) then problem occurs but a review will never consist of TAB even if we press it we go to some other column here reviews can also has double quotes inside which can cause problems, so we try to ignore them

CLEANING THE TEXTS

we clean the text since, at the end of the day we are going to collect a bag of words which contains only essential words non-essential words = the, a, an, on.. simply we collect those words which can tell us whether it's a good review or bad review we also do stemming, taking the root word example Loved, love conveys same so we take only "LOVE" we also get rid of capitals and after doing all that we proceed to

TOKENISATION MODEL (Bag of words of model)

It splits different reviews into different words we take all the words from all the reviews and attribute each word to each column for each review each column will contain the number of times the associated word appears in the review. so, we'll have a lot of zeroes because for each review a lot of words don't appear in the review

```
import pandas as pd

import matplotlib.pyplot as plt

import numpy as np


dataset = pd.read_csv('Restaurant_Reviews.tsv',delimiter ='\t',quoting = 3)


#cleaning the texts

import re


#using sub function we try to eliminate all the numbers from the review

#1st param : it tells we are not going to remove characters that are a-z and A-Z

#2nd param: if we remove a character from the review and the characters to the left and right of it are going to stuck together and doesn't make any sense
```

```
#          it create a word that doesn't make any sense, so now removed character is replaced by ' '(space)


review = re.sub('[^a-zA-z]',' ',dataset['Review'][0])

#converting all characters to lower cases

review= review.lower()


#removing words like prepositions,articles which doesn't make any impact on the review which tells bad or good

import nltk

nltk.download('stopwords')

#this stopwords contains all the words which doesn't tell whether the review is bad or good, simply we don't collect those words in our word bag

#now we loop over our review, and if the word is present in stopwords then we eliminate that word

#we convert string to list so iteration becomes easier

review = review.split()

#importing stopwords package

from nltk.corpus import stopwords

review = [word for word in review if not word in set(stopwords.words('english'))]


#Stemming, we simply apply stemming function to each word, this step can be performed parallely to the above step


from nltk.stem.porter import PorterStemmer

ps=PorterStemmer()


review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]


#now we join all words of the list to a string

#here ' ' implies there will be space appended between the words

review = ' '.join(review)


#we have done for only one review, now we are going to do this for all 1000 review

corpus=[]

corpus.append(review) #since we have already done it for the 1st review

for i in range(1,1000):

   review = re.sub('[^a-zA-z]',' ',dataset['Review'][i])
```

```
review= review.lower()
review = review.split()
review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]
review = ' '.join(review)
corpus.append(review)
```

O/P: Dataset before,

Index	Review	Liked
0	Wow... Loved this place.	1
1	Crust is not good.	0
2	Not tasty and the texture was just nasty.	0
3	Stopped by during the late May bank holiday off Rick Steve recommendation and loved…	1
4	The selection on the menu was great and so were the prices.	1
5	Now I am getting angry and I want my damn pho.	0
6	Honeslty it didn't taste THAT fresh.)	0
7	The potatoes were like rubber and you could tell they had been made up ahead of tim…	0
8	The fries were great too.	1
9	A great touch.	1
10	Service was very prompt.	1
11	Would not go back.	0
12	The cashier had no care what so ever on what I had to say it still ended up being w…	0
13	I tried the Cape Cod ravoli, chicken, with cranberry...mmmm!	1
14	I was disgusted because I was pretty sure that was human hair.	0
15	I was shocked because no signs indicate cash only.	0
16	Highly recommended.	1
17	Waitress was a little slow in service.	0
18	This place is not worth your time, let alone Vegas.	0
19	did not like at all.	0
20	The Burrittos Blah!	0

Dataset after cleaning our data,

0	str	1	wow love place
1	str	1	crust good
2	str	1	tasti textur nasti
3	str	1	stop late may bank holiday rick steve recommend love
4	str	1	select menu great price
5	str	1	get angri want damn pho
6	str	1	honeslti tast fresh
7	str	1	potato like rubber could tell made ahead time kept warmer
8	str	1	fri great
9	str	1	great touch
10	str	1	servic prompt
11	str	1	would go back
12	str	1	cashier care ever say still end wayyy overpr
13	str	1	tri cape cod ravoli chicken cranberri mmmm
14	str	1	disgust pretti sure human hair
15	str	1	shock sign indic cash
16	str	1	highli recommend
17	str	1	waitress littl slow servic
18	str	1	place worth time let alon vega
19	str	1	like

CREATING THE BAG OF WORDS MODEL(Tokenisation)

At the end, we are going to predict whether a review is positive or negative, so it's simply a classification model, and here we have two variables one dependent and another independent

SPARSE MATRIX:

A matrix containing of huge number of zeros. In ML we should try to reduce sparsity as much as possible

Here Max_features=1500 gives us the 1500 most relevant and frequent words to train our algorithm

Code:

```
#creating bag of words model

from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(max_features = 1500) #use of this is we can have a very huge number of colums in our X matrix which leads to sparcity

X = cv.fit_transform(corpus).toarray()

y= dataset.iloc[:,1].values

#From now we use naive bayes or decision tree classification

#we are using naive bayes here


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)


from sklearn.naive_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(X_train,y_train)


y_pred=classifier.predict(X_test)


from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test,y_pred)


from sklearn.metrics import accuracy_score

Accuracy= accuracy_score(y_test,y_pred)
```

o/p: accuracy= 0.73

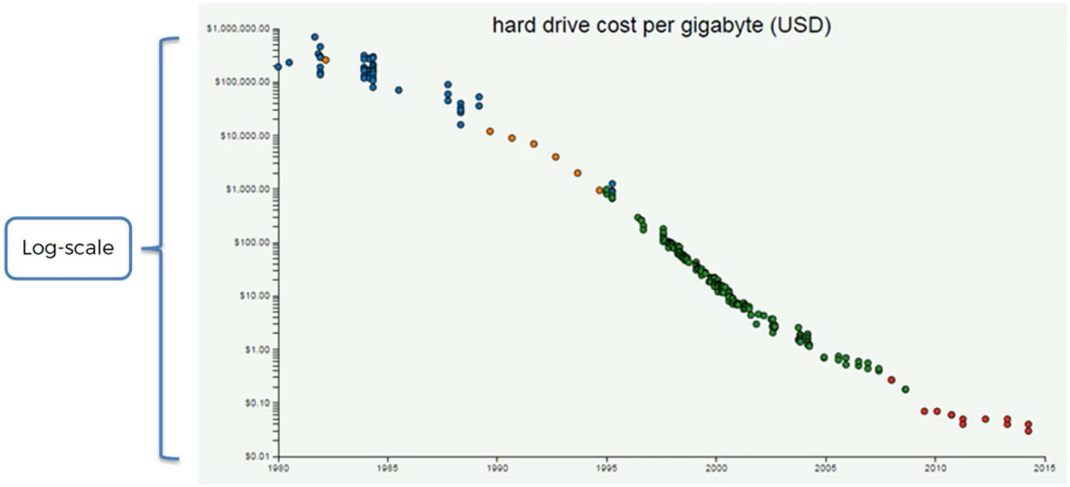
DEEP LEARNING

20 years ago, people don't know even what is internet

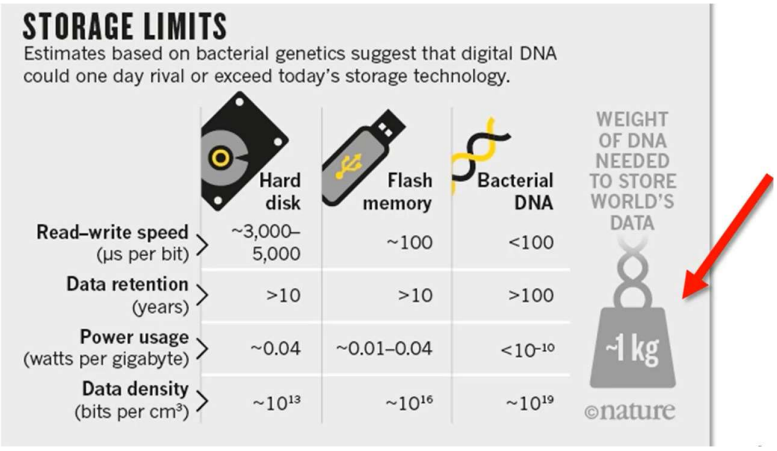
For deep learning, we need two things, first is a lot of data and second is processing power

HOW DATA IS EVOLVED?

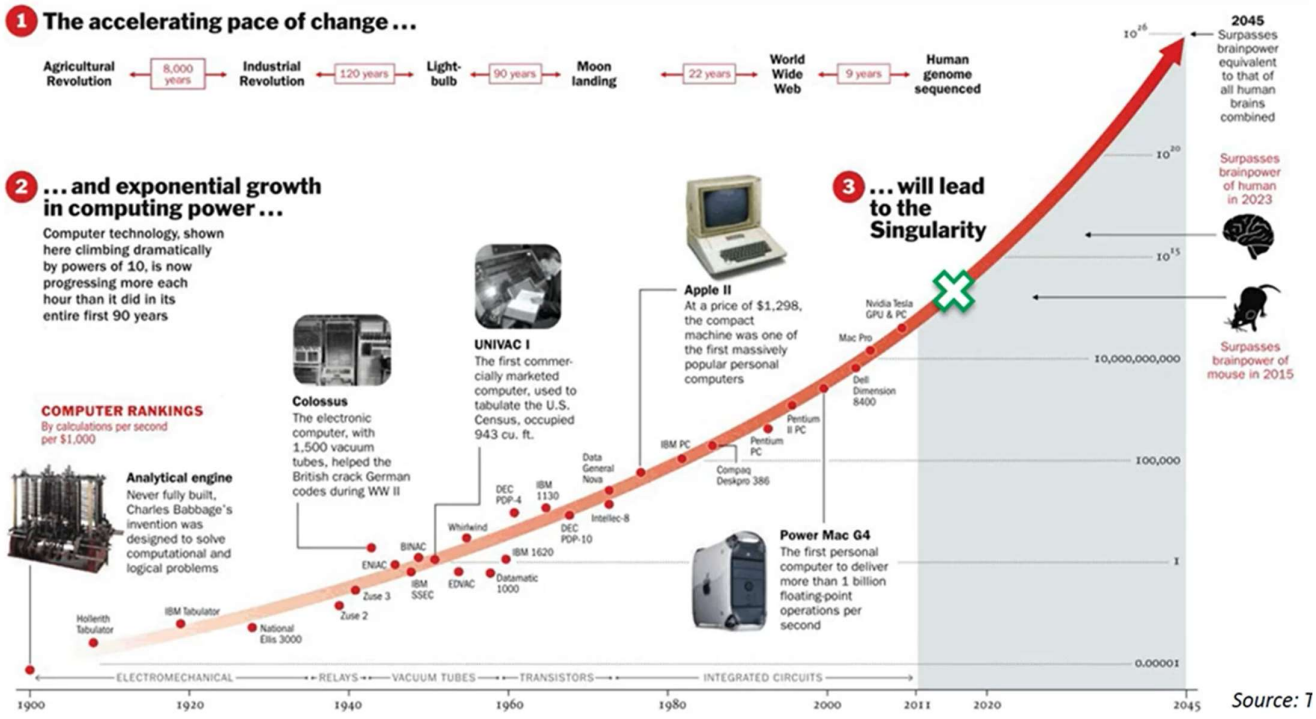
1956	1980	2017
 A company had to pay 2500$ to rent 5MB of storage (Hard drive) for one month		 SanDisk Ultra 256GB MicroSDXC UHS-I Card with Adapter (SDSQUNI-256G-GN6MA). by SanDisk $149.99 $199.99 Prime 21,224 Get it by Wednesday, Mar 22 FREE Shipping on eligible orders Product Features 256GB capacity breakthrough More Buying Choices $147.99 (10 new offers)



Scientists are making research to store data in DNA



- ➔ Machines now had surpassed the computational power of rats
- ➔ It's estimated that machines will surpass the computation power of human's brain in 2023
- ➔ Interesting fact is that in year 2045 it's estimated that power of one machine will equal to power of brains of all the humans combined in the world



"GEOFFREY HINTON" is known as god father of DEEP LEARNING. He works at google and he published a lots of research papers

What we do now using DEEP LEARNING?

1. We try to mimic how human brain operates? Inside a human brain there are approximately 100 billion neurons, each neuron is connected with approximately 1000 of its neighbouring neurons

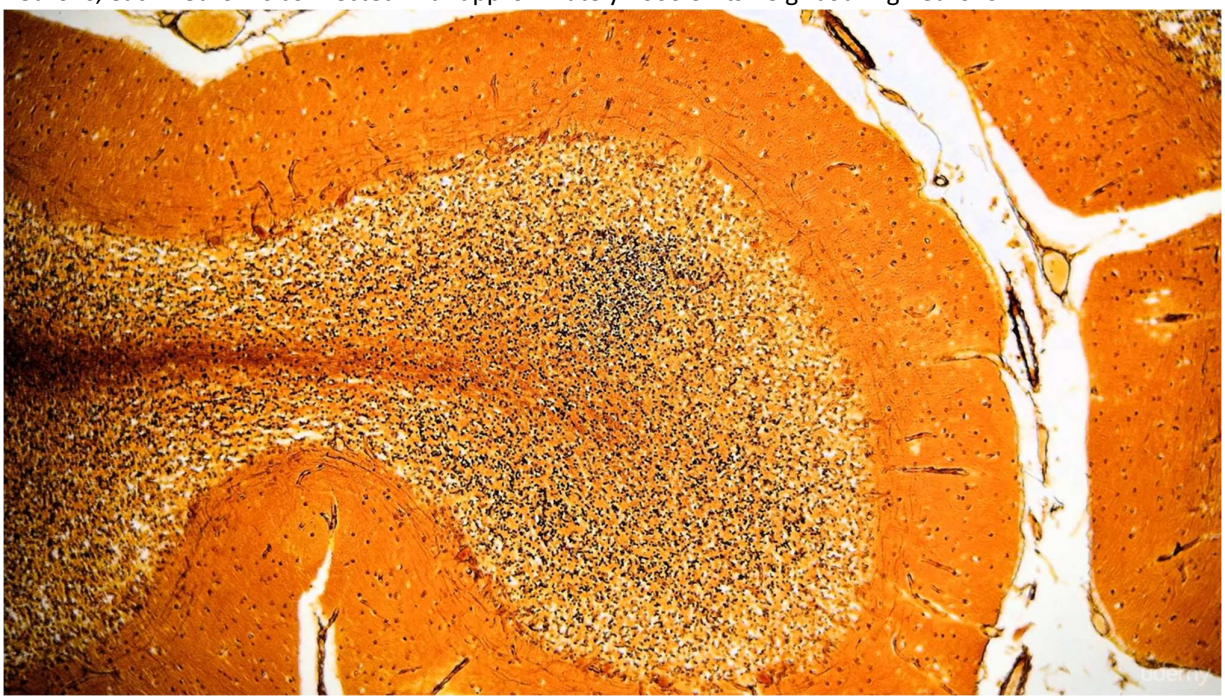
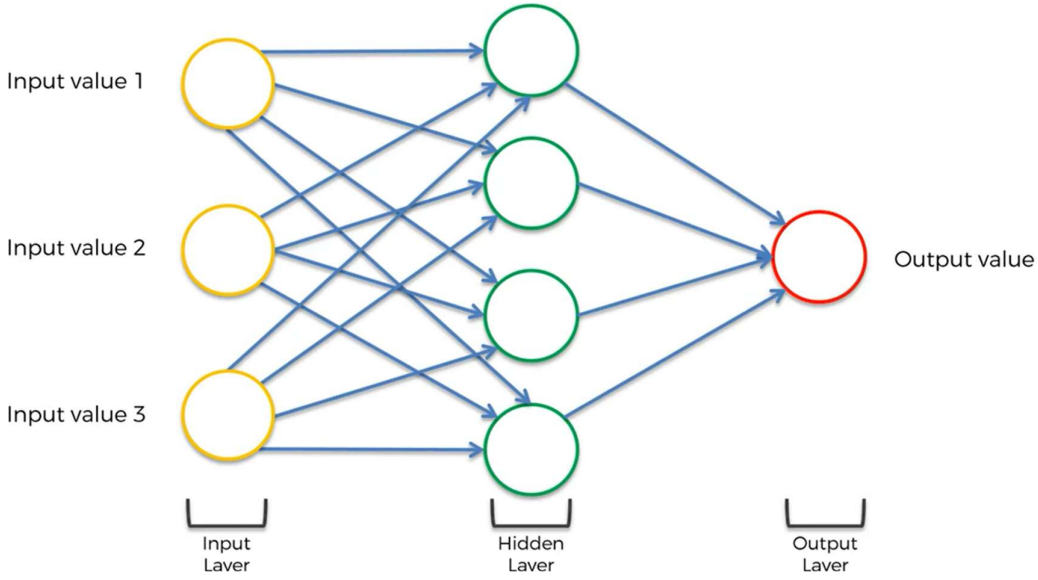


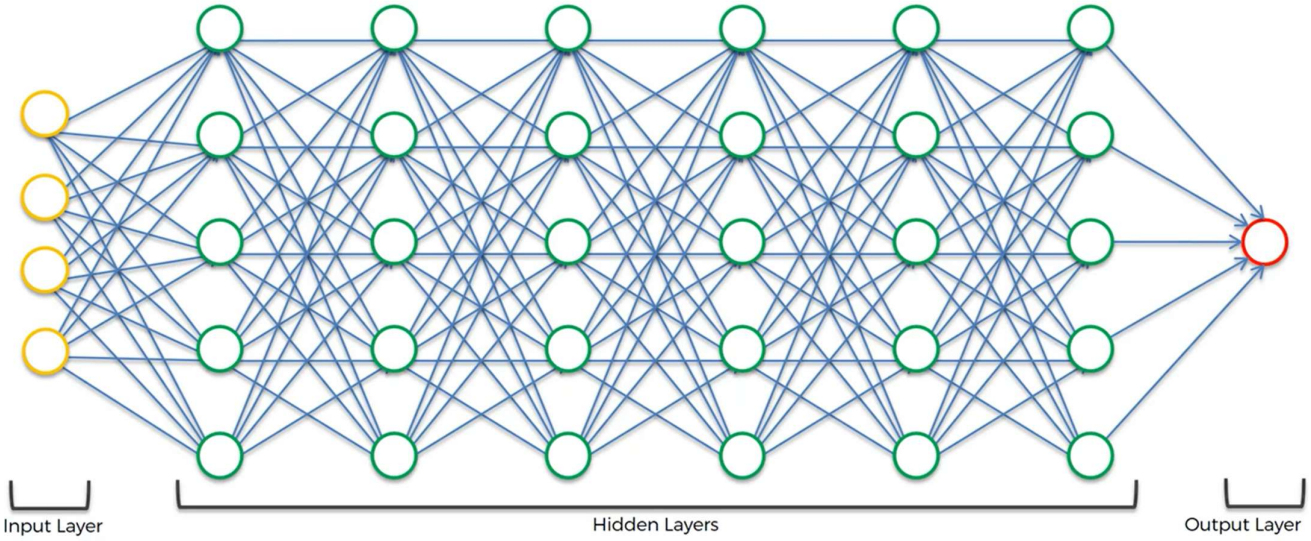
Illustration of a human brain

2. Similarly, we try to construct an artificial similar structure called an Artificial neural net where we have nodes or neurons

Below is referred as shallow learning, where we don't have huge number of hidden layers



Below is referred as deep learning, where we have huge number of hidden layers

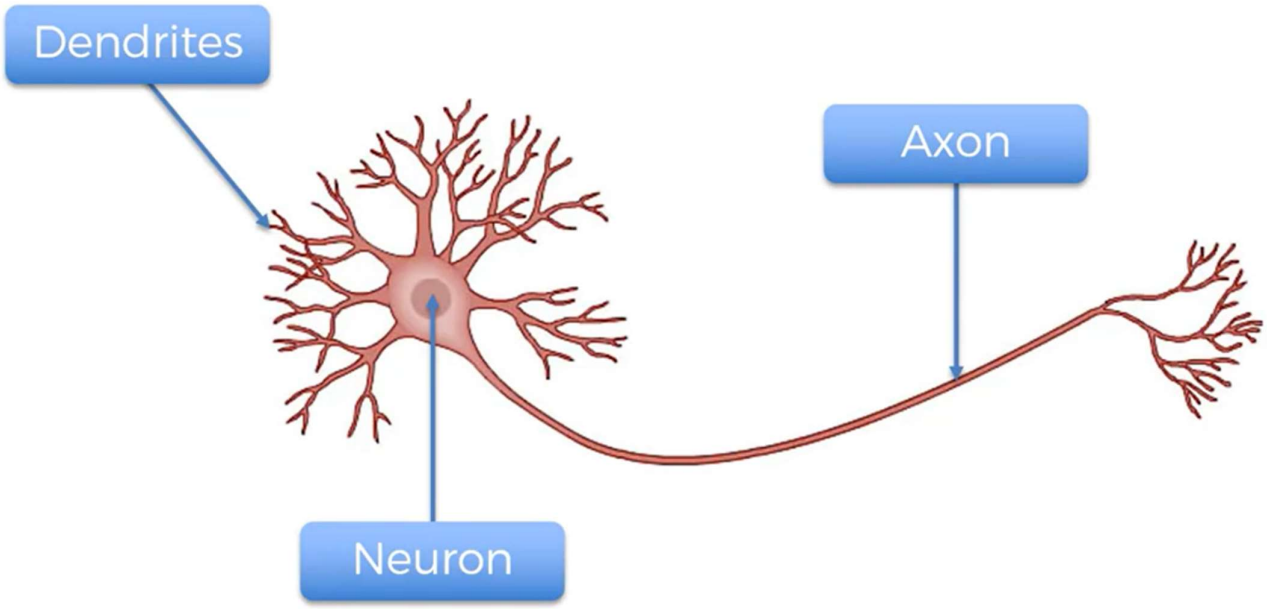


ARTIFICIAL NEURAL NETWORK

What we will learn in this section

- The neuron
- The activation function
- How do neural networks work? (Example)
- How do neural networks learn?
- Gradient descent
- Stochastic Gradient descent
- Backpropagation

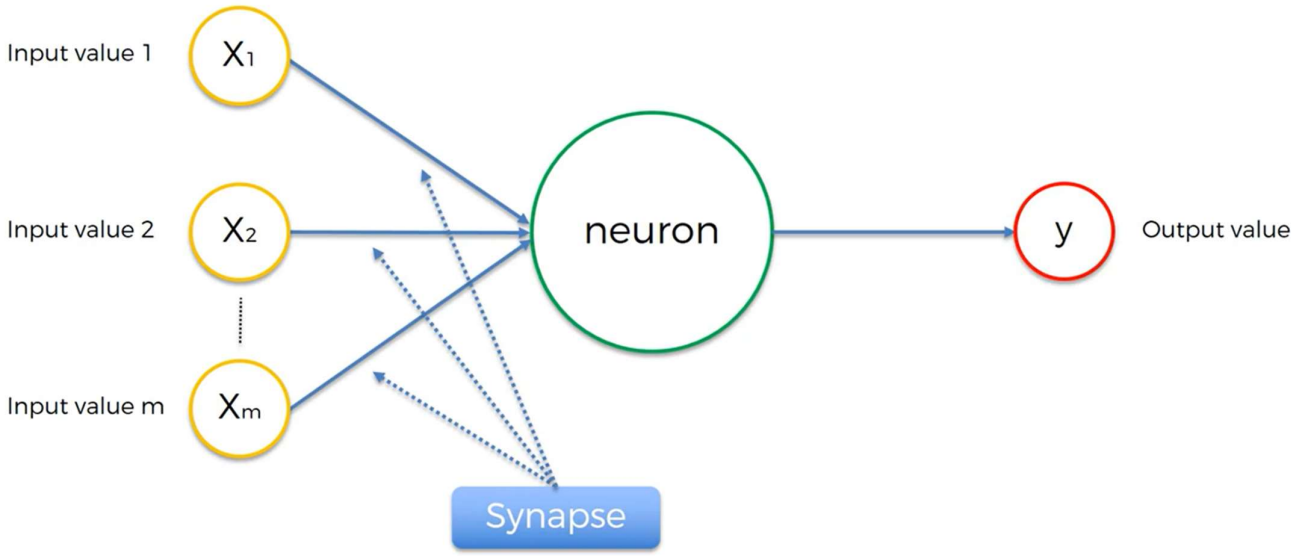
THE NEURON



Dendrites – Receive signals

Axon – Transmits signals

Axon of one neuron connects to dendrites of other neuron, the part where axon of one neuron and dendrite of other neuron meet or touches is called SYNAPSE



We take care of independent variables X_1, X_2, … X_m and make them as standardize (Mean, variance =1) or normalize them and it depends on the situation

What is y? (dependent variable)

- Continuous
- Binary
- Categorical (produces several output values)

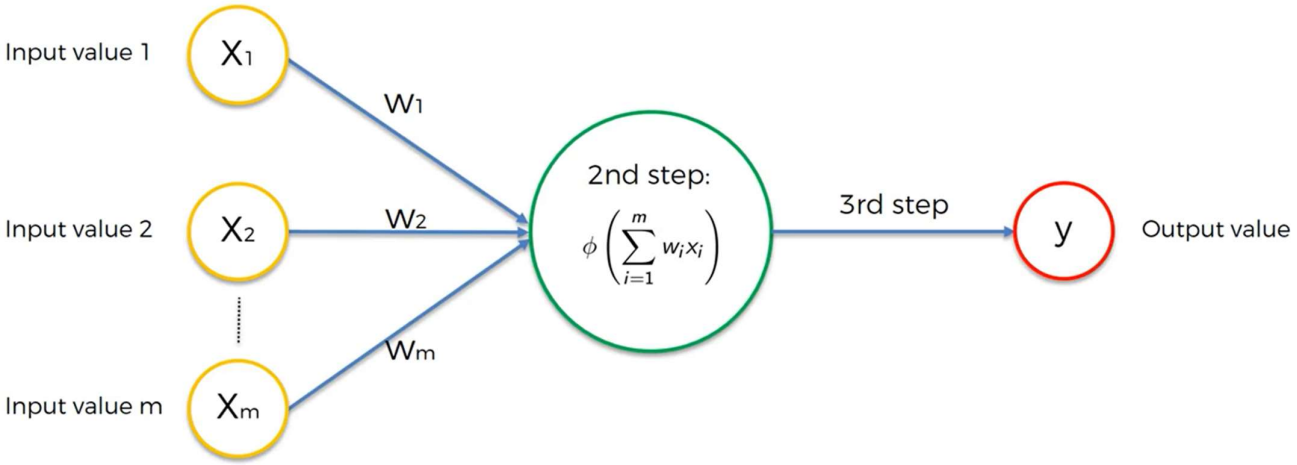
Synapses:

Here all the Synapses will be assigned with weights. Weights are very crucial to artificial neural network and it works functioning because weights are how neural networks learn by adjusting the weights the neural network decides in every single case what single signal is poor and what signal is not important to a neuron, What signal pass along and what signal don't pass along, They are the things that are going to get adjusted during the process of learning and here is where gradient descent and back propagation come into play

What is inside the neuron?

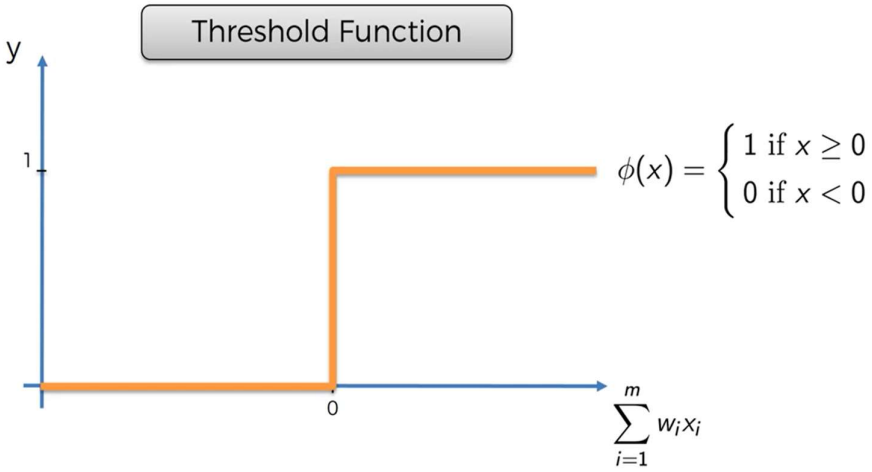
1. All the weights it got gets added up "weighted sum= $(\sum W_i X_i)$"
2. It applies *ACTIVATION FUNCTION* to the weighted sum

THE ACTIVATION FUNCTION

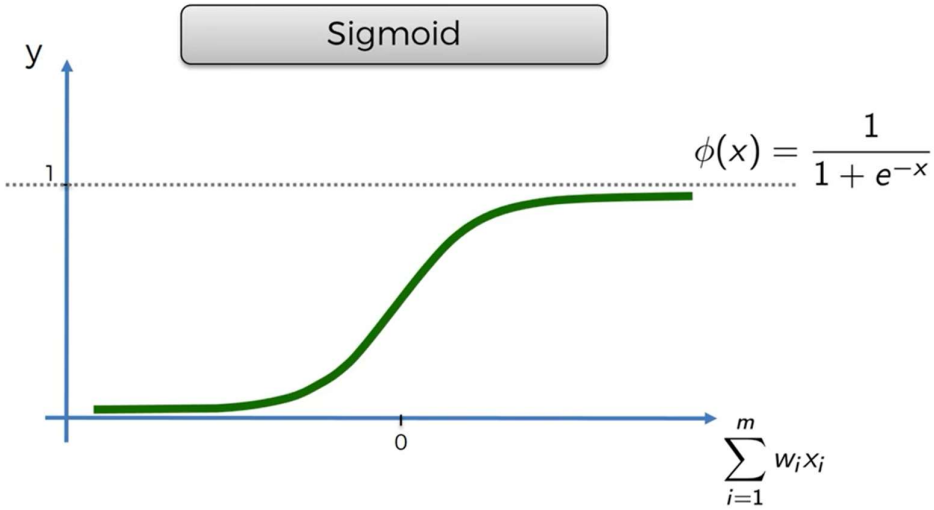


Types of activation functions:

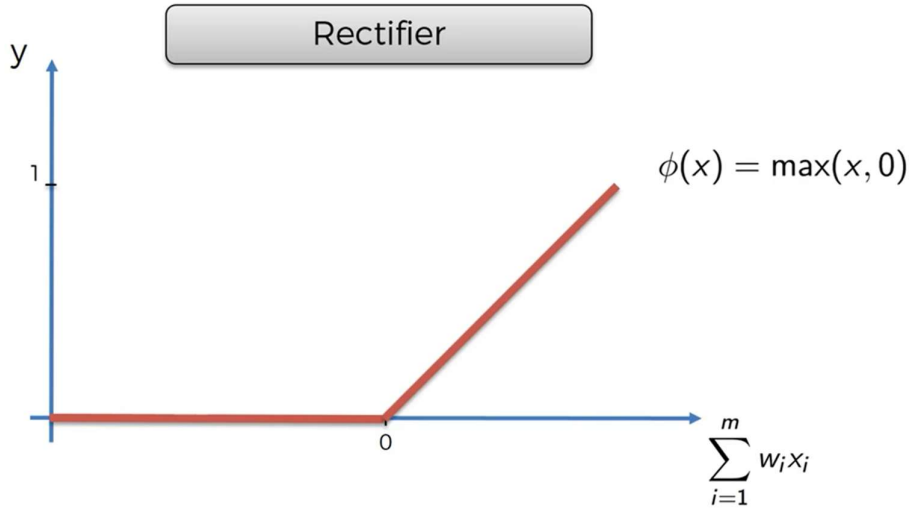
1. Threshold function (Used when Dependent Variable is binary)



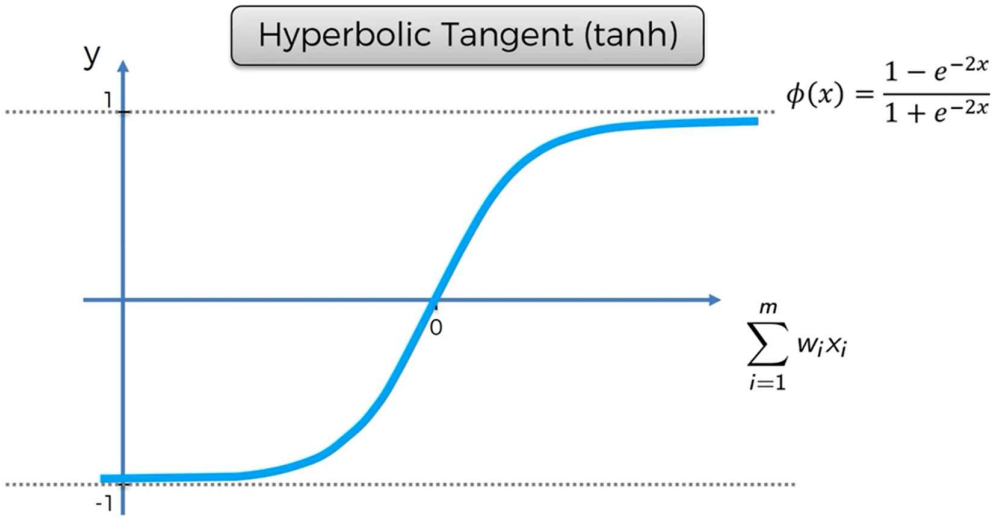
2. Sigmoid function (used when dependent variable is between 0 and 1, we can also classify, used at output layer)



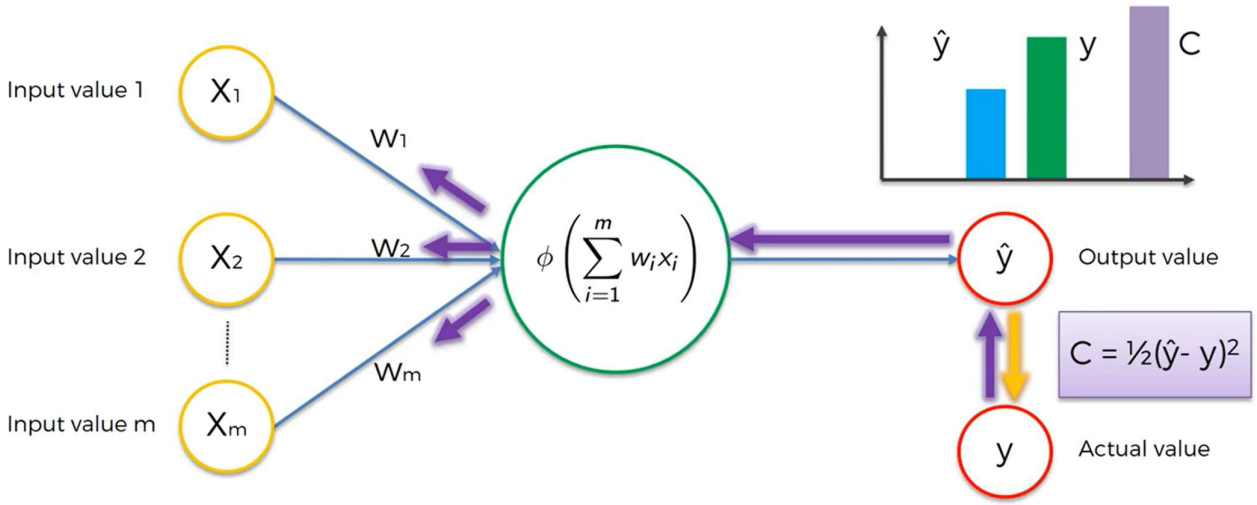
3. Rectifier Function (Most used in ANN, Mostly used In hidden layers)



4. Hyperbolic Tangent(tanh)



HOW DO NEURAL NETWORKS LEARN?



Step-1: Weights are given, all are added up at the neuron

Step-2: Activation function gets applied

Step-3: ANN predicts some output (= y')

Step-4: Now we compare y'(predicted) and y(actual) and we prepare a cost function, which tells us how much difference is there between actual and predicted

Step-5: according to the cost, the information is sent back to the neural network and according to the cost, weights Are going to be adjusted which can decrease the error (Our goal is to decrease the cost function)

We keep on doing this process and keep on feeding the same row of data to the ANN, we constantly update weights

Until the cost function becomes very low (It means error is low)

So, the above process is for only one row of data, then how for several rows of data?

The process is very simple, Take ***only SINGLE NEURAL NETWORK*** (the image have 8 individual ann's but it's purely to understand)

If we have 8 rows of data,

Step-1: take row-1 apply it to ANN note down y'_1;

take row-2 apply it to ANN note down y'_2;

take row-3 apply it to ANN note down y'_3;

take row-4 apply it to ANN note down y'_4;

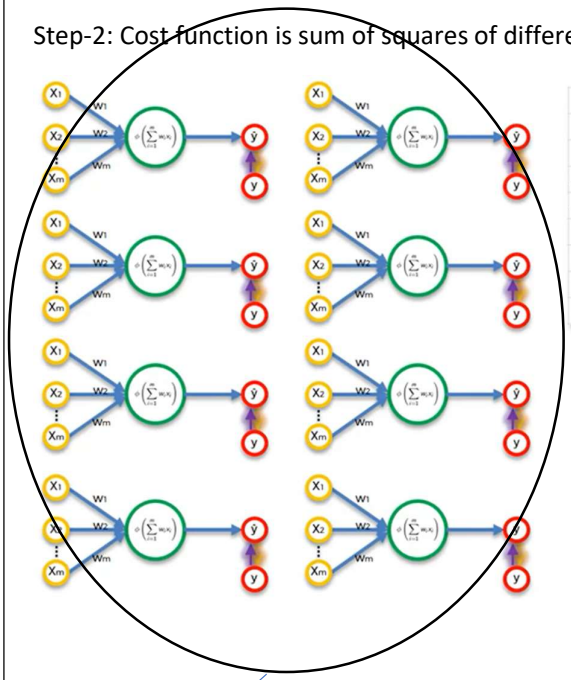
take row-5 apply it to ANN note down y'_5;

take row-6 apply it to ANN note down y'_6;

take row-7 apply it to ANN note down y'_7;

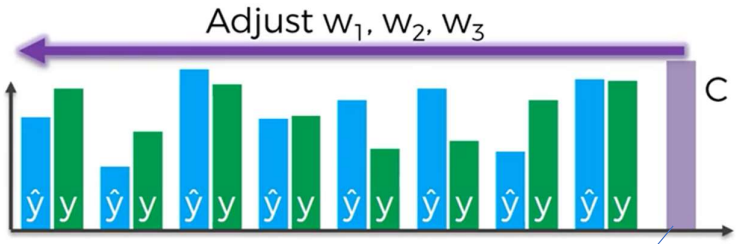
take row-8 apply it to ANN note down y'_8;

Step-2: Cost function is sum of squares of differences of y'_i and y



Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

$$C = \sum \tfrac{1}{2}(\hat{y} - y)^2$$



This is not 8 individual ANN but **IT'S ONLY A SINGLE ANN**

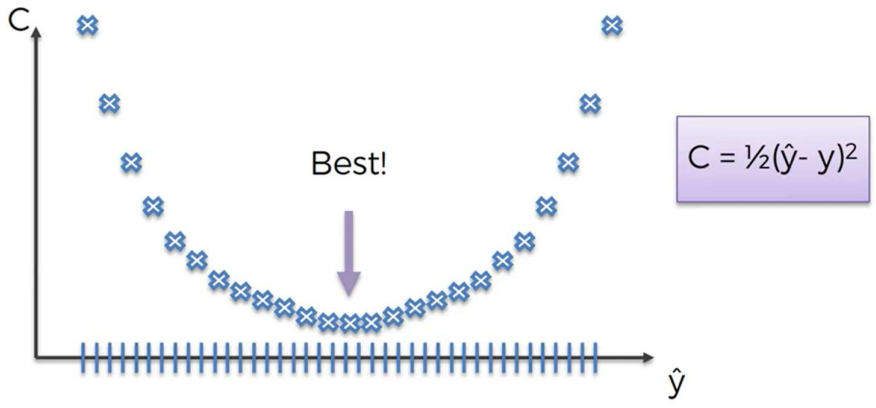
Goal is to minimize the Cost function

NOTE: The above process is known as backpropagation

GRADIENT DESCENT

This is used in order to adjust the weights and also minimize the cost function

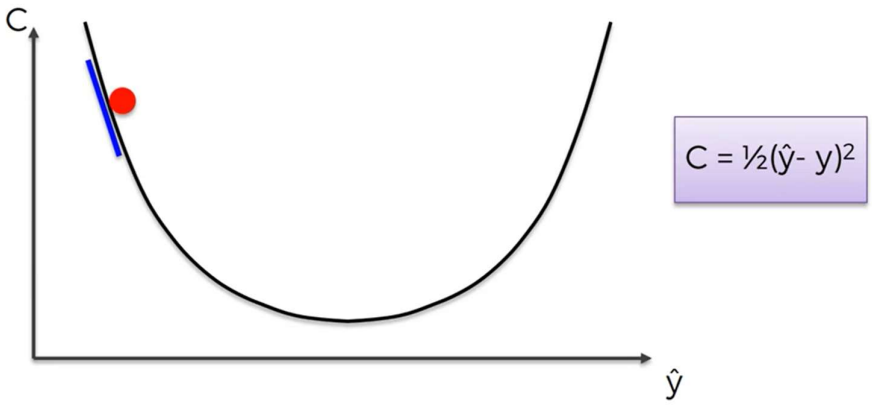
- ➔ A question arises why won't we take each and every y' value compute cost function to each and every y'
- ➔ This is simply a brute force approach



- ➔
- ➔ But for very large datasets, combinations will get way too higher and the present fastest computer in the world too can't solve it, Then how?

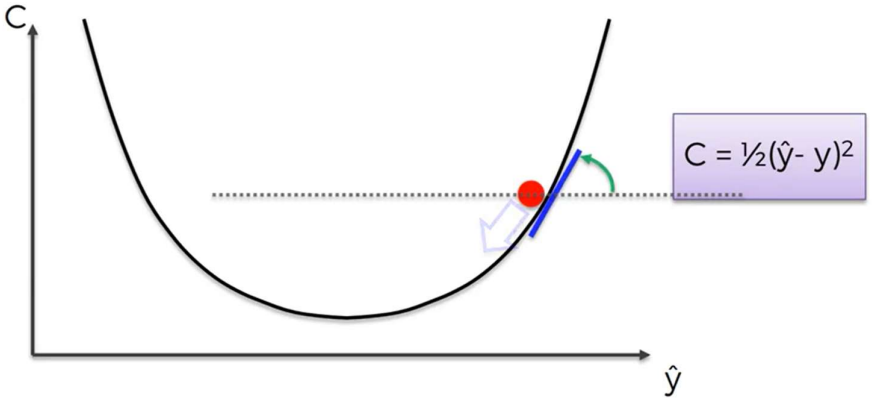
Now gradient descent comes into the picture,

At initial stage, plot a graph between y' and C (=1/2(y'-y)^2) [it's a parabola] and find the slope of the obtained value for C

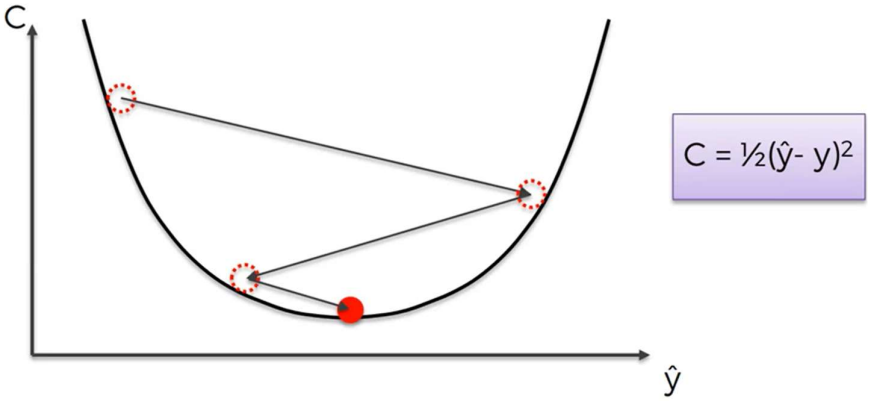


Here the slope is negative so we go to right (Since, negative slope represents downhill)

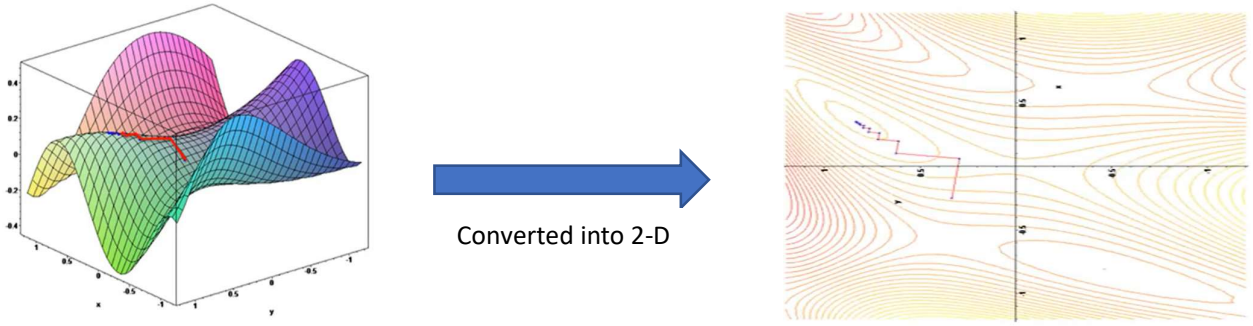
After that, assume we landed up here,



Now slope is positive now go to left, this process continues until we get cost function as minimum as possible

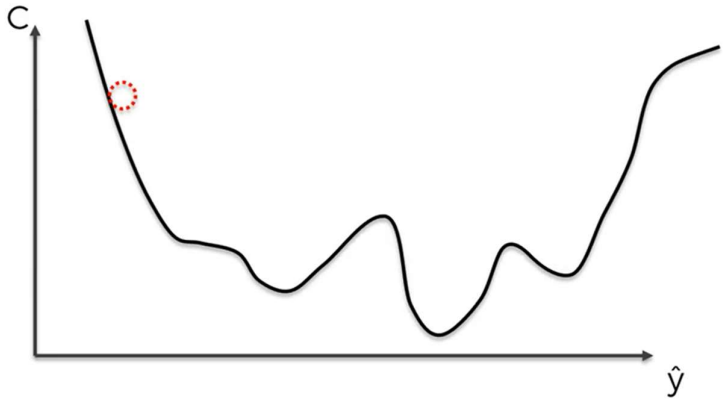


HOW WILL BE THE GRADIENT DESCENT IF WE PLOT IT IN 3-D



STOCHASTIC GRADIENT DESCENT

The above cost function is a convex shaped function simply it's a parabola, what if in some cases our cost function is not a convex shaped curve, what If our cost function looks like this

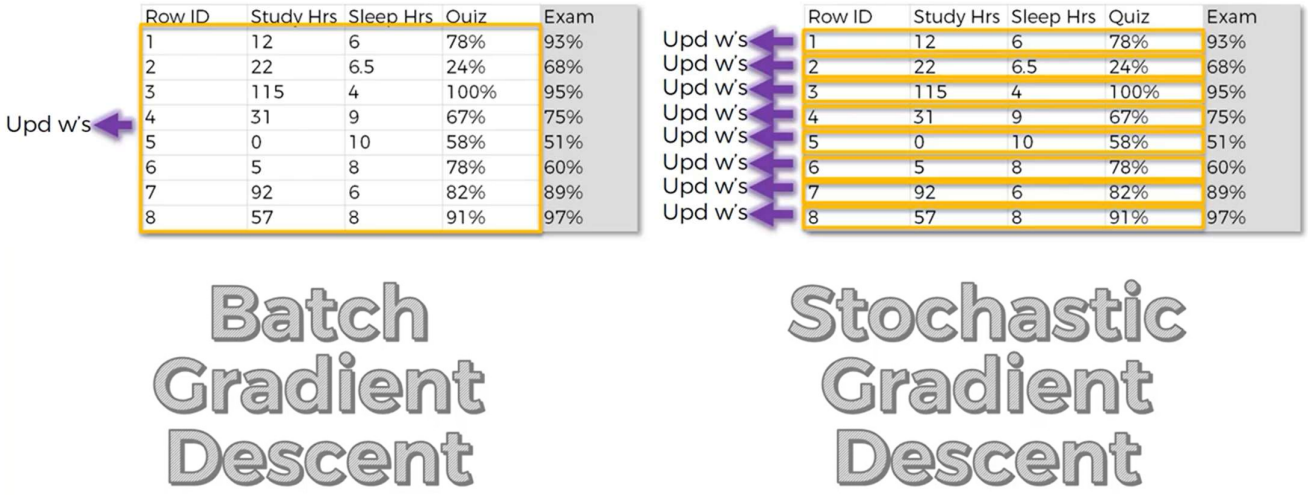


The main difference between normal gradient descent (Batch Gradient Descent) and stochastic GD is that,

In normal we plot every y' on the graph then compute cost then adjust weights, Where as in stochastic GD

In step-i, We plot y'_i then we adjust weights, (where i=1,2,3….m)

Example, we plot y_1' we compute weights, again we plot y_2' then again, we compute weights and so on



Batch Gradient Descent	**Stochastic Gradient Descent**
Slower when compared	Very faster
have to load up all the data into memory	Doesn't have to load up all the data into memory
Not a row by row, computes entire dataset once then make changes to entire dataset	Row by row execution
Should wait until all of those rules are on together	Need not wait until all of those rules are on together
Heavy algorithm than stochastic	Lighter algorithm
Deterministic algorithm	Sarcastic algorithm (since it's random)

TRAINING THE ANN WITH STOCHASTIC GRADIENT DESIGN

STEP 1: Randomly initialise the weights to small numbers close to 0 (but not 0).

STEP 2: Input the first observation of your dataset in the input layer, each feature in one input node.

STEP 3: Forward-Propagation: from left to right, the neurons are activated in a way that the impact of each neuron's activation is limited by the weights. Propagate the activations until getting the predicted result y.

STEP 4: Compare the predicted result to the actual result. Measure the generated error.

STEP 5: Back-Propagation: from right to left, the error is back-propagated. Update the weights according to how much they are responsible for the error. The learning rate decides by how much we update the weights.

STEP 6: Repeat Steps 1 to 5 and update the weights after each observation (Reinforcement Learning). Or: Repeat Steps 1 to 5 but update the weights only after a batch of observations (Batch Learning).

STEP 7: When the whole training set passed through the ANN, that makes an epoch. Redo more epochs.

Business problem:

We have details of fictional bank customers, Problem is Bank is experiencing unusual churn rates => customers leaving the bank at high rates, bank wants to know why people are leave and why churn rate is high. Your job is to find out the customers and tell about these customers to the bank who are at highest state of risk to leave the bank (Judgment graphic segmentation model)

Here we should install three special libraries

- **Theano Library**: It's an open source numerical computational library, it can run not only cpu but also in gpu

  GPU is much more useful when we are doing deep neural networks, for simple neural

  Networks CPU is enough

- **Tensor flow library**: It's also an open source numerical computational library which can run numerical

  Computations. Run on both cpu and gpu. Originally developed by google brain team

  It's presently under the licence of apache

These two libraries are used mostly for research and development purposes for deep neural networks, it means if we want to use these two libraries for deep learning you would use them to build a deep neural network from scratch

Therefore, we are not going to use Theano & tensor flow library, we use a library called "keras" library which wraps up above two libraries

Like sklearn which implement ML models, here keras implements deep learning models

How to install these libraries

- ➔ Open start menu and type anaconda prompt
- ➔ Type this command "conda install keras"

Algorithm intuition with dataset

1. Use dense function to randomly initialize weights number close to 0
2. Here dataset contains 11 features
3. Use rectifier function as activation function for hidden layers and sigmoid activation function for output layer
   Which gives us the probability whether the customer leaves the bank or not
4. Compare predicted result with actual, compute cost function
5. Back propagate and update weights,
6. Repeat steps 1-5
7. When whole training set passed through ANN, that makes an epoch, re repeat more epochs

Parameters for dense function:

1. How many nodes are chosen? Simply hidden layer nodes = (#input layer nodes + #output layer nodes)/2

   In dataset 11 input and 1 output so we choose dimension of hidden layer as 6 (number of nodes =6)

   Units=6

2. kernel_initializer='uniform' -> adds weights uniformly
3. activation='relu' -> rectifier activation function
4. input_dim = 11 (This step is used for only once, since first layer in a sequential model must get input_shape,

   for next hidden layers, we don't need to spend this argument)

➔ Same for all the hidden layers except argument -4
   - classifier.add(Dense(units=6, kernel_initializer='uniform', activation='relu'))

➔ For o/p layer we choose units=1(based on model, since we require whether customer leaves or not it has either 0 or 1 o/p we have dimension of o/p layer =1)

   And also, replace activation function as sigmoid

   classifier.add(Dense(units=1, kernel_initializer='uniform', activation='sigmoid'))

Note: If we have dimension in o/p layer >1 then units >1 and activation = 'softmax' -> softmax is also a sigmoid function which is applied to dependent variable of more than 2 category

COMPILING THE ANN:

1. *Optimizer* is simply the algorithm you want to use to find the optimal set of weights in neural networks
   Optimizer = 'adam' (which is a efficient type of stochastic gradient descent)
2. If your dependent variable has a binary outcome then loss = binary_crossentropy
   If your dependent variable has More than 2 categories then loss=Categorical_crossentropy
3. Metrics is just a criterion that chosen to evaluate your model, we use accuracy criterion,
   Here we used list('[]') since metrics parameter is expecting a list of metrics and we are inputting a single metric

Note:

1 Epoch = 1 Forward pass + 1 Backward pass for ALL training samples.

Batch Size = Number of training samples in 1 Forward/1 Backward pass. (With increase in Batch size, required memory space increases.)

Number of iterations = Number of passes i.e. 1 Pass = 1 Forward pass + 1 Backward pass (Forward pass and Backward pass are not counted differently.)

Example : If we have 1000 training samples and Batch size is set to 500, it will take 2 iterations to complete 1 Epoch.

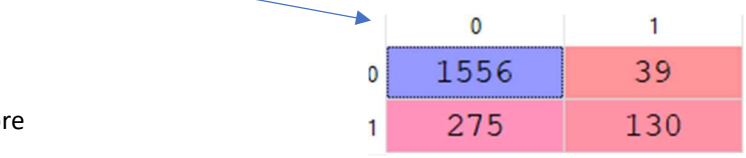
How to choose results of confusion matrix

- ➔ Y_pred will give the probabilities of a customer leaving the bank, but we don't need this probabilities but we need the predicted results in a binary format,
- ➔ So we need to choose a threshold, value over the threshold is 1 and below is 0

CODING PART

```
import pandas as pd

import matplotlib.pyplot as plt

import numpy as np


dataset= pd.read_csv('Churn_Modelling.csv')

X=dataset.iloc[:,3:13].values

y=dataset.iloc[:,13].values


from sklearn.preprocessing import LabelEncoder, OneHotEncoder


label_encoder_X1 = LabelEncoder()

X[:,1]= label_encoder_X1.fit_transform(X[:,1])

label_encoder_X2 = LabelEncoder()

X[:,2]= label_encoder_X2.fit_transform(X[:,2])

onehotencoder = OneHotEncoder(categorical_features = [1])

X= onehotencoder.fit_transform(X).toarray()

X= X[:,1:]


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)


from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)


#importing the keras libraries and packages

import keras

from keras.models import Sequential
```

```
from keras.layers import Dense

#Initialzing the ann - defining it as a sequence of layers
#can be done in two ways, either defining as a sequence of layers
#or defining a graph
classifier = Sequential()

#adding the input layer and first hidden layer
#dense function will take care of step-1, look algorithm
classifier.add(Dense(units=6, kernel_initializer='uniform', activation='relu', input_dim = 11 ))

#adding the second hidden layer
classifier.add(Dense(units=6, kernel_initializer='uniform', activation='relu'))

#adding the output layer
classifier.add(Dense(units=1, kernel_initializer='uniform', activation='sigmoid'))

#compiling the ANN
classifier.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])

#fitting the ANN to training set, the accuracy you observe here is on train set
classifier.fit(X_train,y_train,batch_size=10,nb_epoch=100)

#predicting the test set,it predicts the probability to leave the bank
y_pred = classifier.predict(X_test)
y_pred = (y_pred>0.5)
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test,y_pred)

#accuracy score
from sklearn.metrics import accuracy_score
Accuracy= accuracy_score(y_test,y_pred)
```



	0	1
0	1556	39
1	275	130

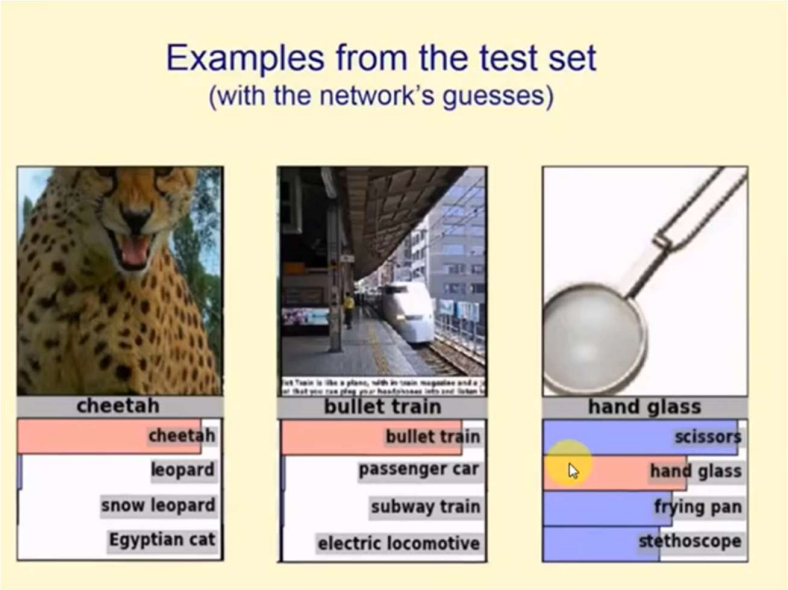
Accuracy = 0.843

CONVOLUTIONAL NEURAL NETWORKS

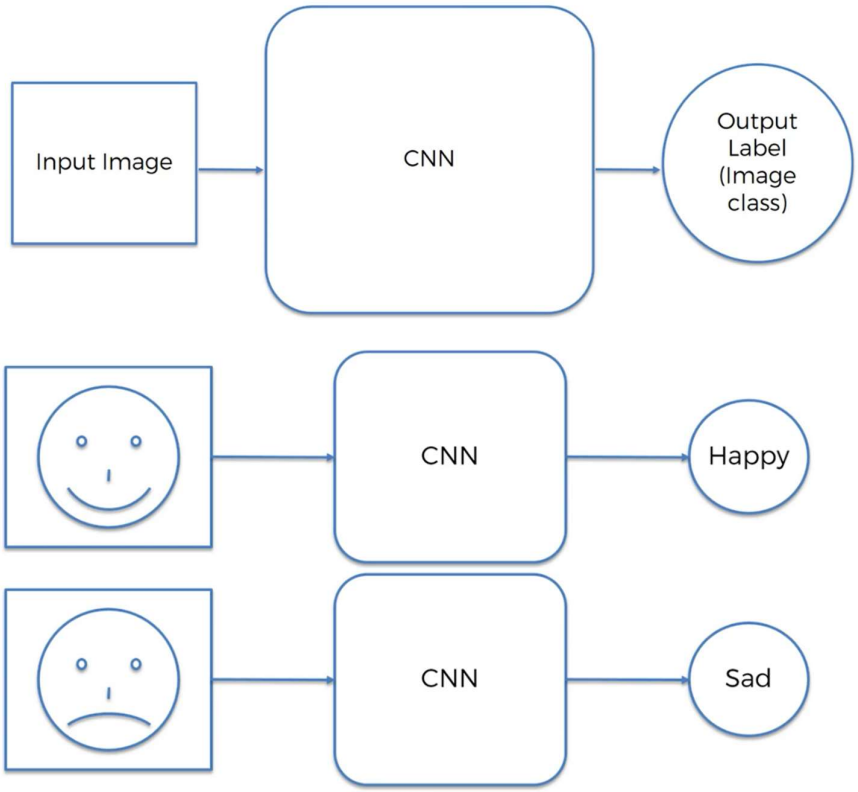
What will we learn in this section?

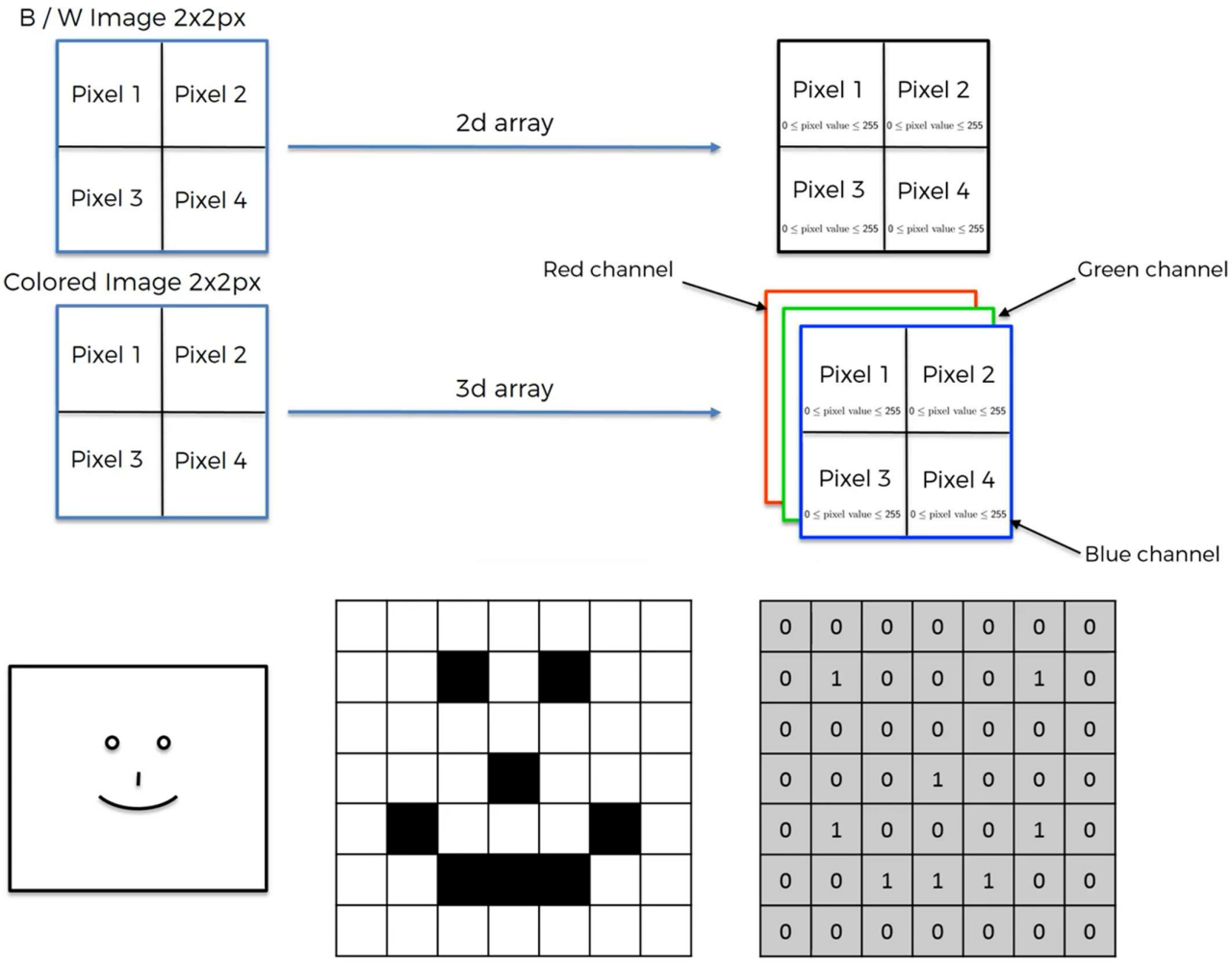
- What are convolutional neural networks
- Convolutional operation
- RELU layer
- Pooling
- Flattening
- Full connection

WHAT ARE CONVOLUTIONAL NEURAL NETWORKS



"***YANN LECUN***" is a god father of Convolutional neural networks, he's a student of ***GEOFFREY HINTON*** and he's director of Facebook artificial intelligence centre



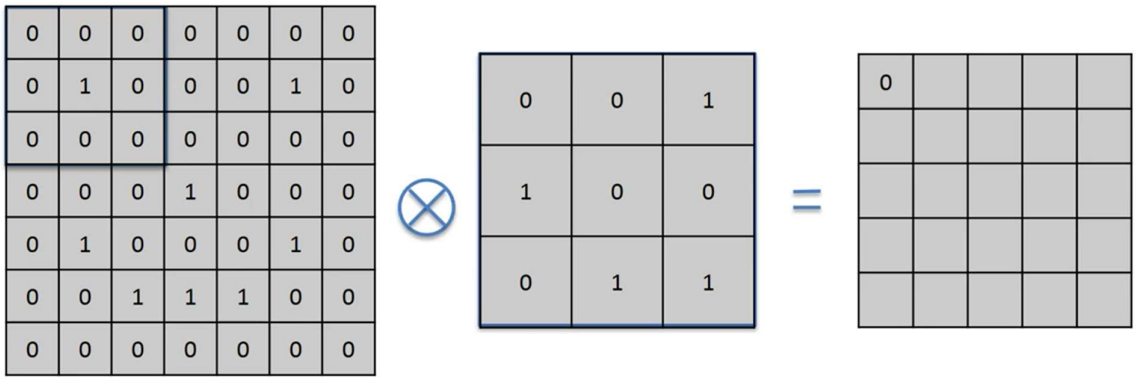


0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

STEPS WE ARE GOING THROUGH THESE IMAGES

1. Convolution
2. Max pooling
3. Flattening
4. Full connection

CONVOLUTION



Feature detector also known as “FILTER”

What we’ve done is we put the feature detector/filter exactly on the input image of first 3*3 size (highlighted one)

And we multiply corresponding value and sum up all the values, nothing matches so final result = 0

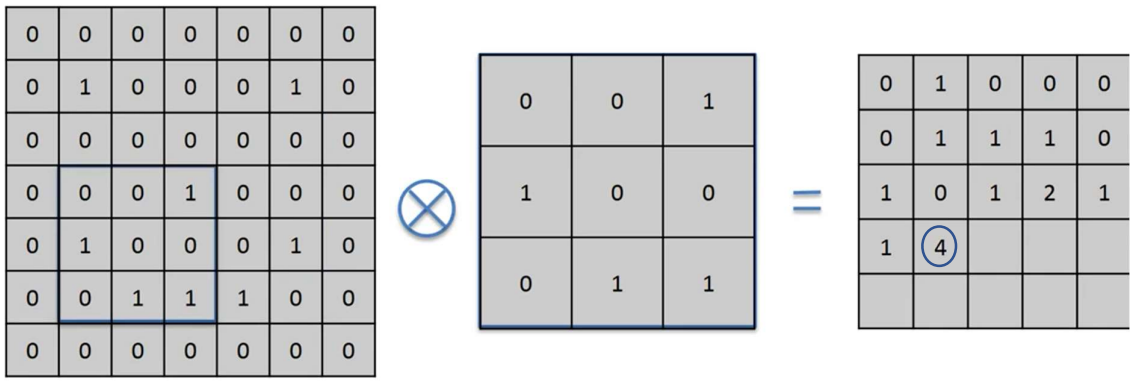
0*0 + 0*0 + 0*1 = 0

0*1 + 1* 0 + 0*0 = 0

0*0 + 0*1 + 0*1 =0

} 0 + 0 + 0 = final value (Simply final value is how many of the cells matched up)

Another illustration,

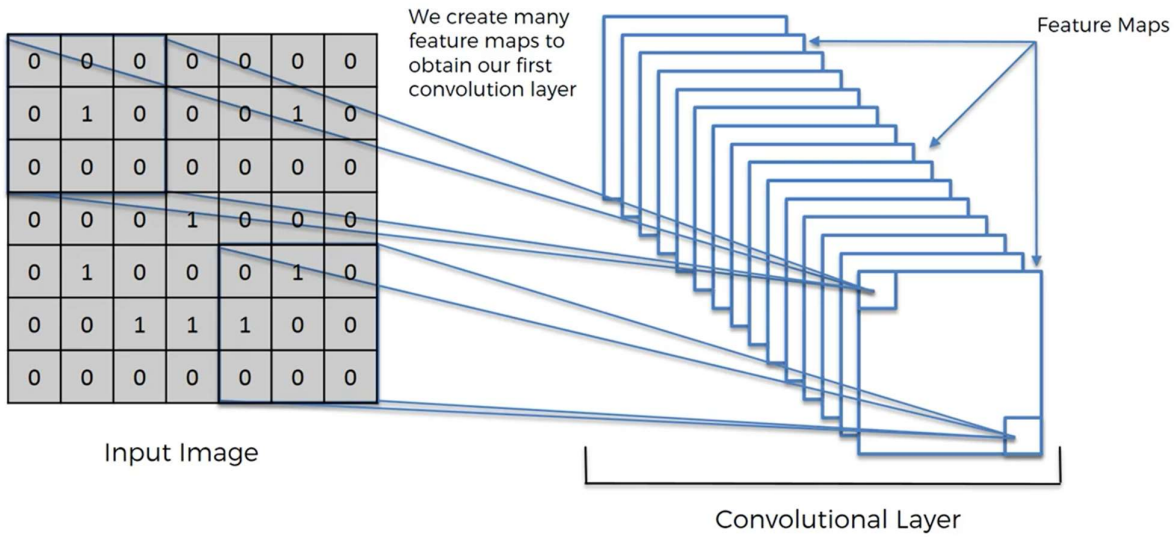


Here 4 values got matched, similarly we fill all the values in the Feature map

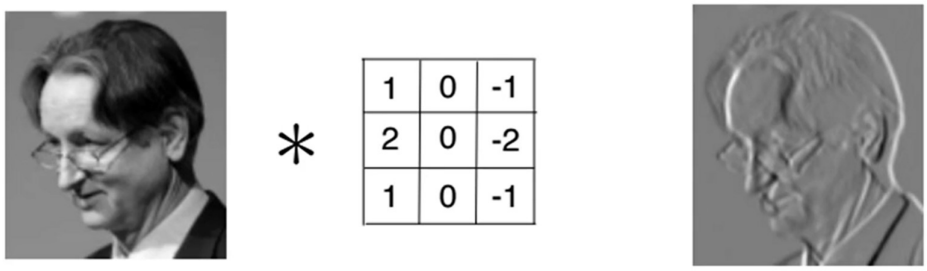
What we have done?

- We reduced the size of the image (according to the size of feature detector size of image gets decided)
- Feature map helps us to detect the important features in an image such as detecting eyes, nose, etc.,
- It allows us to preserve the important features

In practical, we create very many filters



Each of the feature map stores different types of features

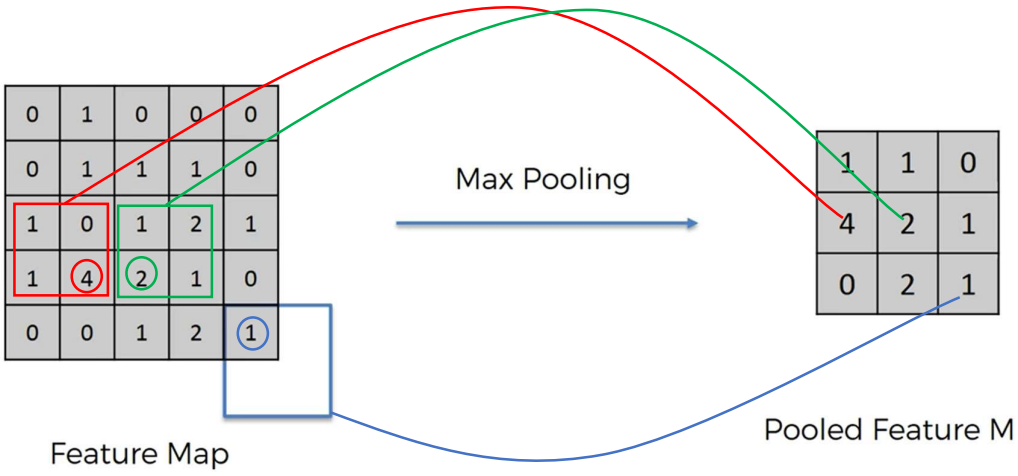


RELU LAYER

Rectifier is used since we increase non-linearity since images are by nature non-linear.

MAX POOLING

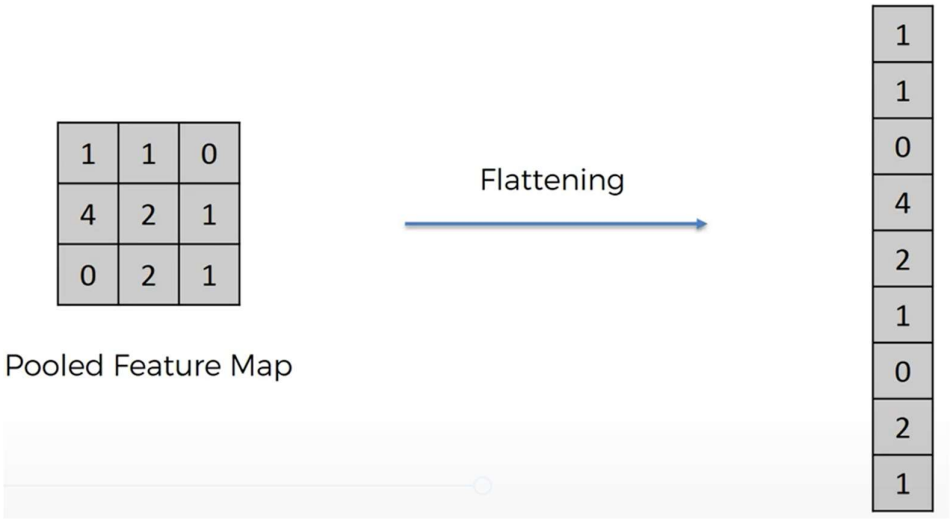
How your model recognises cheetah is there in the image, its angle is different, each image has different exposure level,

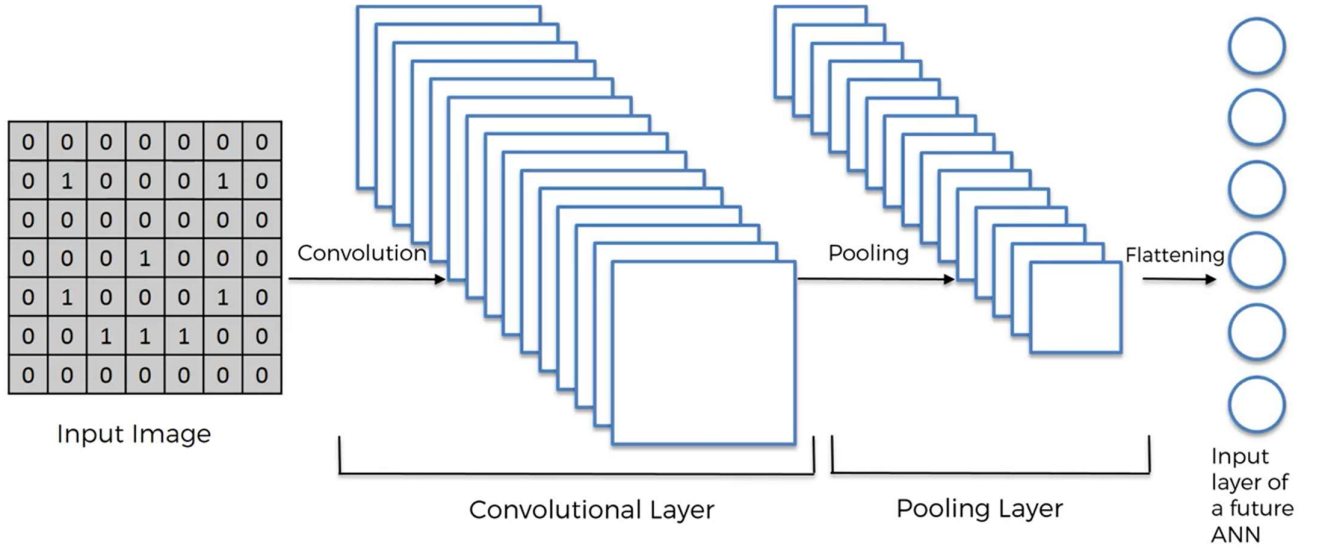


Name it self says (Max pooling) for each 2*2 submatrix we take maximum and put in the corresponding cell of pooled feature matrix

By doing Max Pooling, we preserve features and identify them wherever they are located at in the new image

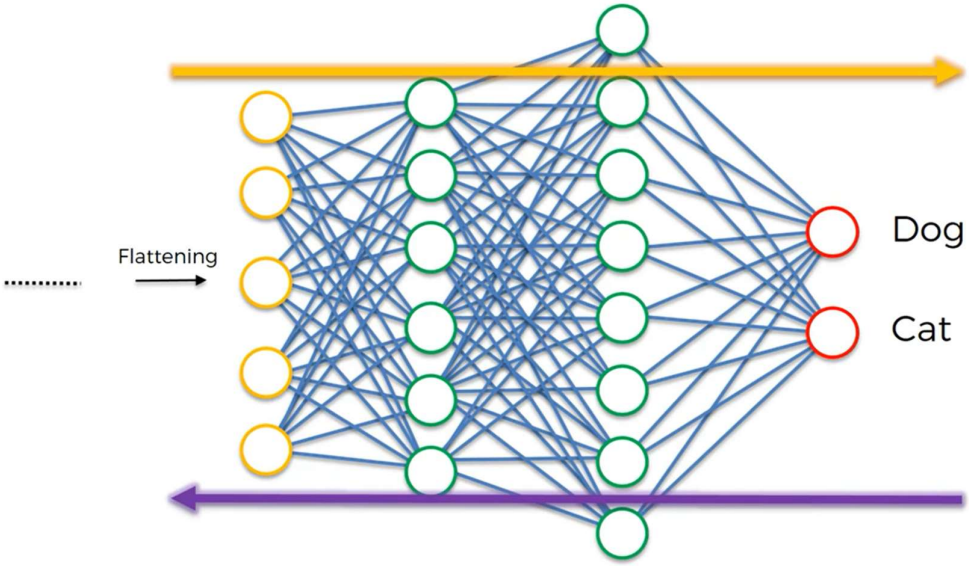
FLATTENING





FULL CONNECTION

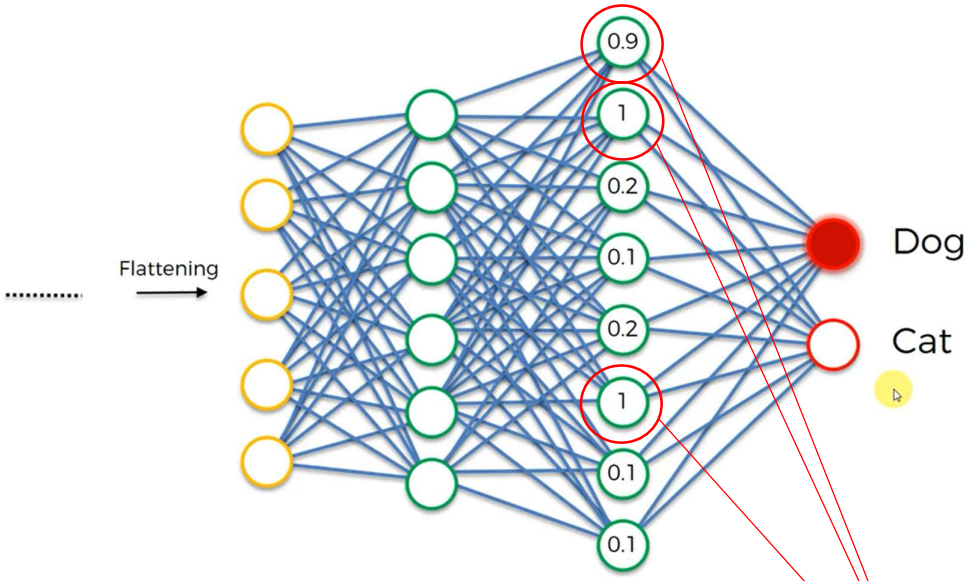
Here, the hidden layers are connected fully unlike the artificial neural networks (where we connect only if its weight is non zero)



The same strategy is followed like artificial neural networks, a prediction is made error value is calculated then it will be back propagated, weights are adjusted and also feature matrix will also be get updated, there will be a lot of math inside it

But the very interesting part is how do these 2 output neurons will work?

Let's have an illustration about how dog's work?



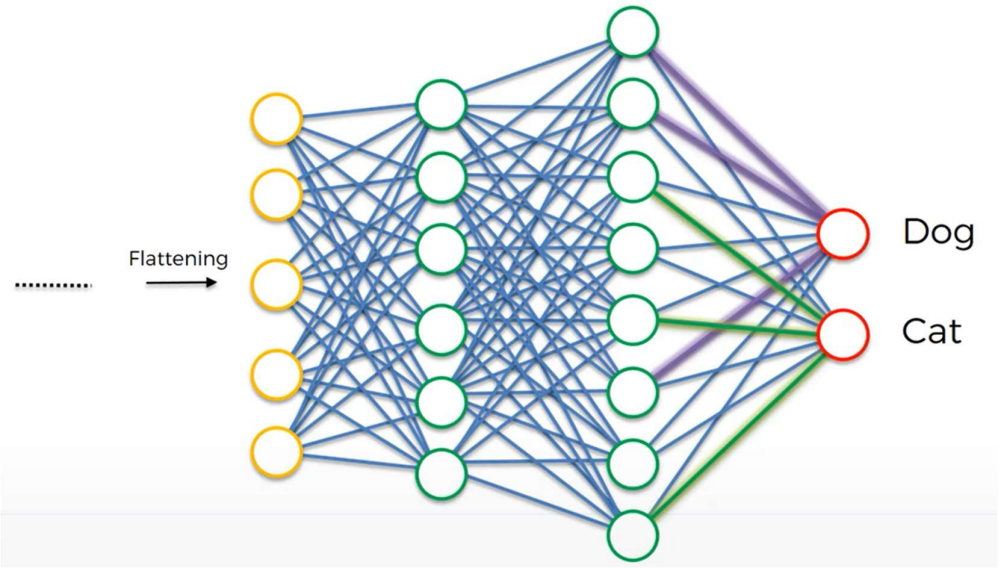
Here, we can observe that nodes with values 0.9,1,1 are strong and confident about some features it found from the input image,

And it's the choice of Dog or cat neuron whether to take into account of those nodes or not. This will be remembered and obtained after lots and lots of training, adjusting the weights etc.,

With these nodes which works for Dog, cat neuron thinks that "Hmm! It's not working from this is not I'm looking for, so just ignore these nodes'

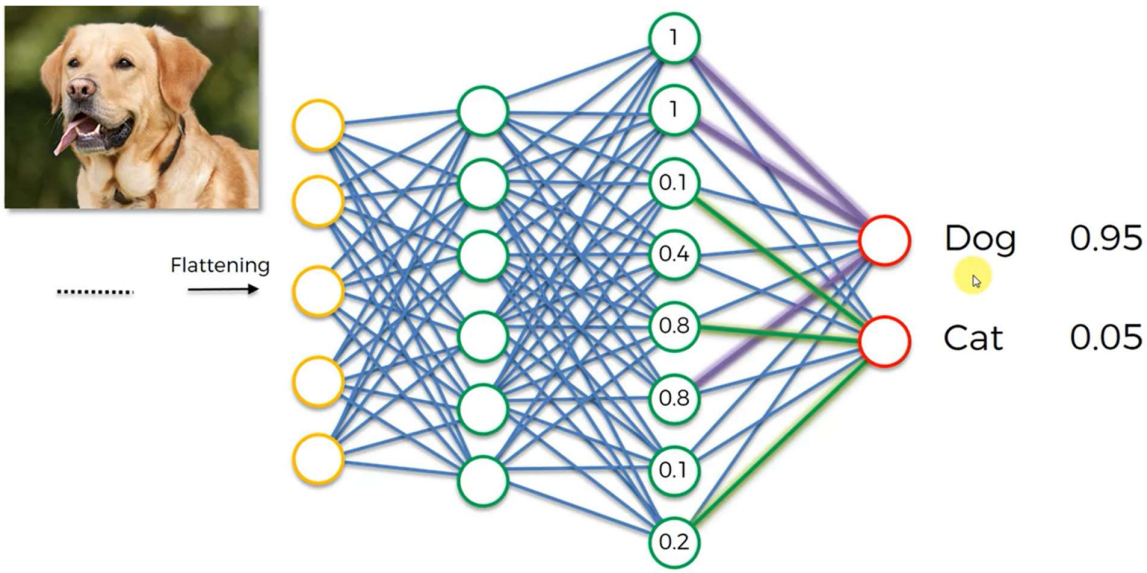
This is how output layer neurons learn

After a lot of training, here's our model looks like



Observer the violet lines over there, those corresponding neurons are the one's which Dog neuron will concentrate on and green one's are concentrated by cat neuron

Suppose a dog image is given as input



3 of the violet nodes of hidden layer has the values of (1,1,0.8) so the probability will be 0.95(assume it as very high)

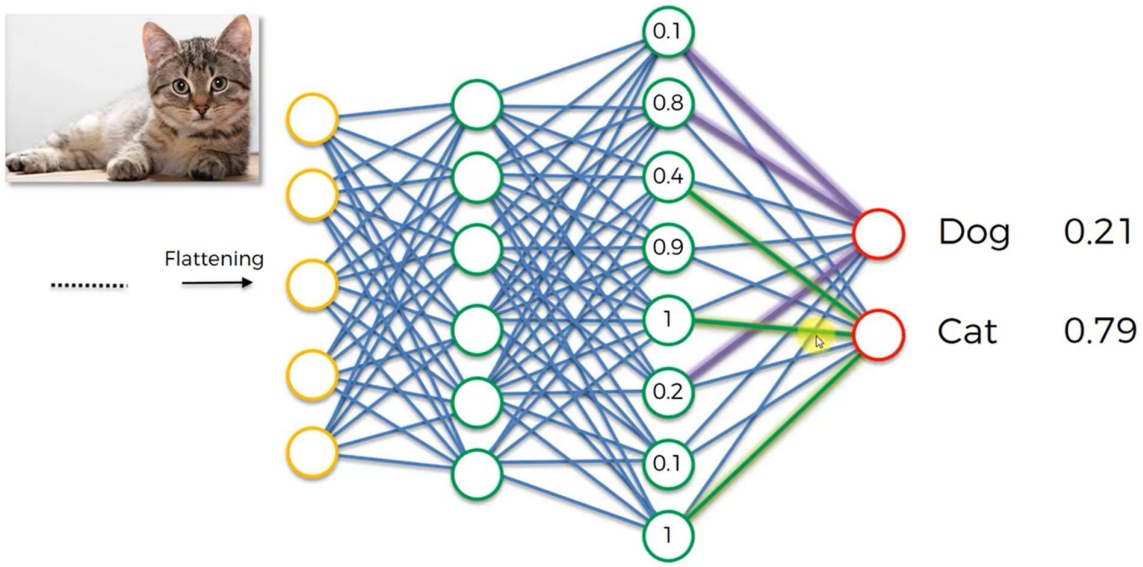
For the same image look at the nodes corresponding to green line, these nodes have values 0.1,0.8,0.2

It says that only one feature is matching (may be eyes or nose, dog has bent ears whereas cat has sharp pointing ears) and the other two features matching probability is less so overall probability = 0.05(not a computed but assume it as very low value)

Here 1st prediction = Dog

2nd prediction = cat

Let's take input as cat this time



Nodes corresponding to Violet lines have values = 0.1,0.8,0.2 => less matching probability (overall = 0.21)

Nodes corresponding to Green lines have values = 0.4,1,1 => two nodes strongly say a feature prob= 0.79

Here 1st prediction = cat

2nd prediction = dog

But remember that dog neuron remembers those violet lines and always vote to those nodes only after a lot and lots of training and back propagation, adjusting weights

Explanation for coding part: (Convolution part)

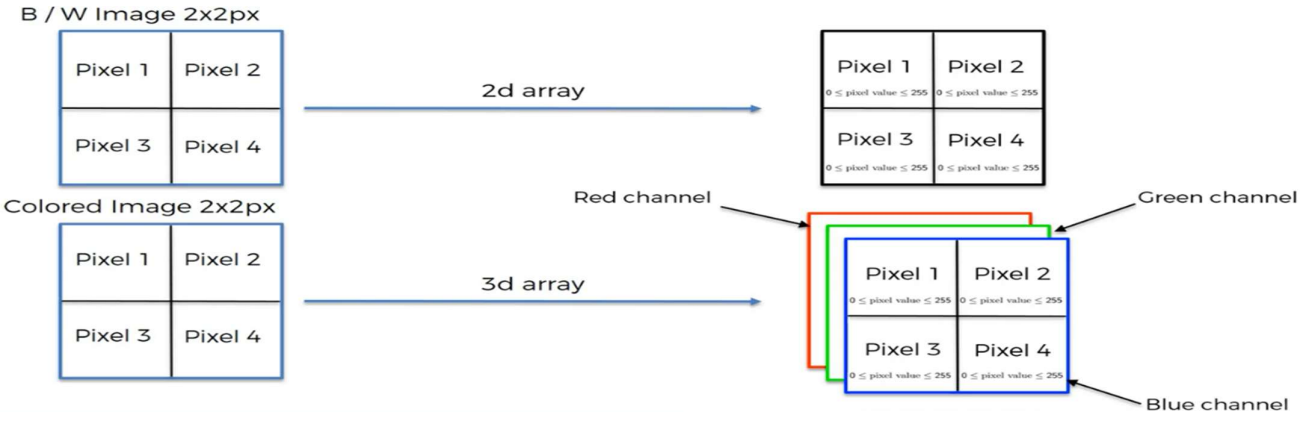
1st argument: number of filters you want to use

2nd argument: number of rows, number of columns

Ex: Convolution2D (32,(3,3)) -> creates 32 feature detectors of 3*3 size (It's used as default)

3rd argument: input_shape -> shape of input image, all our images are converted into this same fixed size

input_shape



Therefore, 3arguments = (64,64,3) -> 3 is number of channels (3 for colour image, 1 for b/w) and 64,64 is dimensions of 2-D array. Chosen as default, we can also choose 256-Dimension size but it'll take a lot and lots of time to process

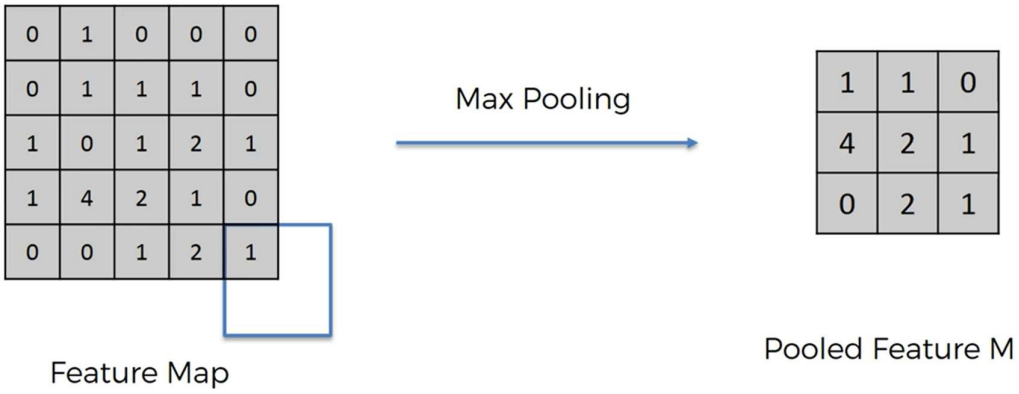
4th argument: to get non-linearity in our images, we use rectifier activation function, Activation ='relu'

Overall: classifier.add(Convolution2D(32,(3,3), input_shape =(64,64,3),activation='relu'))

MAX POOLING:

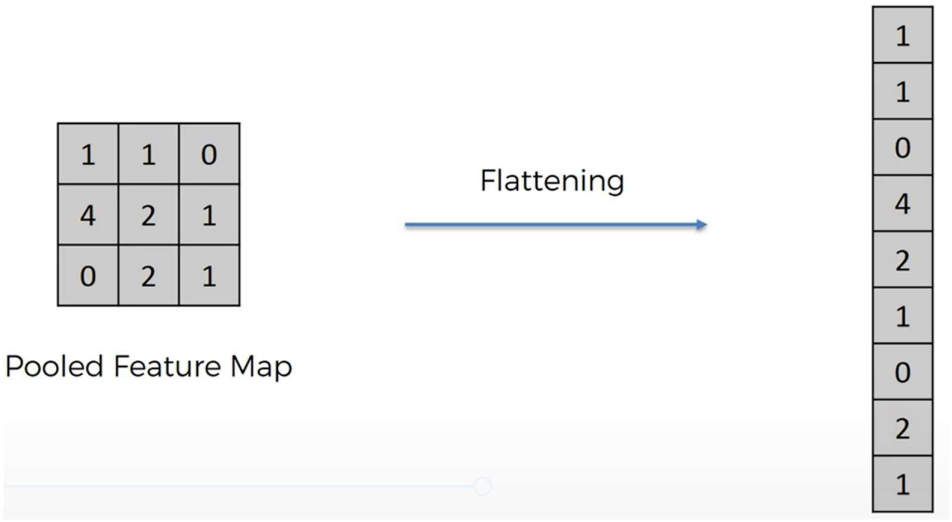
This is used to preserve most of the features and also it helps to reduce the number of nodes in the next step,

If we don't reduce the size of feature maps, we get a lot of nodes, so we use max pooling to reduce complexity and also increase efficiency



1st argument: pool_size= (2,2)

FLATTENING:



Why we haven't taken all the pixels of input image and flatten them into one single vector? Why we done max pooling and convolution?

Answer: If we do this, each pixel of input image will represent independently represent one node in ANN, so we get information's of the pixel itself and we don't get information of how this pixel is spatially connected to the other pixels around it

If we do convolution and pooling step where each feature map represents one feature in convolution step and we max pool each of the feature maps, we preserve highly informational data in the image. It means this huge vector then represent the information of a specific feature of a specific detail of the image.

Example if we want to extract information of ear of a dog, can we see a single pixel in the image and we able to predict this pixel represent ear of a dog, NO! so this convolution operation is able to extract the specific information in the input image

classifier.add(Flatten()) -> we don't need to input any parameters since, keras will understand what are the previous steps

FULL CONNECTION:

1st param: In the artificial neural network, we took number of nodes in the hidden layers as (input+output)/2

What should we take here, since we don't know how many input nodes are going to present,

After a lot of experimentation number of nodes in the hidden layers = 128, => units = 128

2nd param: activation function ='relu'

```
classifier.add(Dense(units=128, activation = 'relu'))


#adding output layer, units=1 since we only predict the
#probability of whether it's dog or cat
classifier.add(Dense(units=1, activation ='sigmoid'))
```

FITTING THE CNN TO THE IMAGES

Before fitting CNN to images, we need to make image augmentation, if we don't do that our training set will be overfitted and will produce poor results to test set but great results for training set.

Open google, search keras documentation

CODING PART

```
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense

#initialzing the CNN
classifier = Sequential()

#step-1 convolution
classifier.add(Convolution2D(32,(3,3), input_shape =(64,64,3),activation='relu'))

#step-2 Pooling
classifier.add(MaxPooling2D(pool_size=(2,2)))

#adding a second convolution to increase test set accuracy
#here input to his 2nd convolution neural network is pooled feature maps coming from previous steps therefore
```

```
#we need not include input_shape and keras will know this
classifier.add(Convolution2D(32,(3,3),activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2,2)))

#step-3 flattening
classifier.add(Flatten())

#STEP-4 Full connection

classifier.add(Dense(units=128, activation = 'relu'))

#adding output layer, units=1 since we only predict the
#probability of whether it's dog or cat
classifier.add(Dense(units=1, activation ='sigmoid'))

#compiling the ANN

classifier.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])

#go to this website https://keras.io/api/preprocessing/image/ and copy paste the code
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(
      rescale=1./255,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)
trainingset = train_datagen.flow_from_directory('dataset/train_set',
                                    target_size=(64, 64),
                                    batch_size=32,
                                    class_mode='binary')
```

```
testset = test_datagen.flow_from_directory('dataset/test_set',
                                           target_size=(64, 64),
                                           batch_size=32,
                                           class_mode='binary')
classifier.fit(
    trainingset,
    steps_per_epoch=2000, #number of images present in training set
    epochs=5,
    validation_data=testset,
    validation_steps=400)  #number of images present in test set
```

```
Using TensorFlow backend.
Found 4000 images belonging to 2 classes.
Found 800 images belonging to 2 classes.
Epoch 1/5
2000/2000 [==============================] - 566s 283ms/step - loss: 0.5166 - accuracy: 0.7324 -
val_loss: 0.4026 - val_accuracy: 0.7513
Epoch 2/5
2000/2000 [==============================] - 563s 281ms/step - loss: 0.2410 - accuracy: 0.8983 -
val_loss: 0.6045 - val_accuracy: 0.7538
Epoch 3/5
2000/2000 [==============================] - 562s 281ms/step - loss: 0.0974 - accuracy: 0.9646 -
val_loss: 1.6761 - val_accuracy: 0.7600
Epoch 4/5
2000/2000 [==============================] - 567s 284ms/step - loss: 0.0594 - accuracy: 0.9791 -
val_loss: 0.7156 - val_accuracy: 0.7550
Epoch 5/5
2000/2000 [==============================] - 563s 281ms/step - loss: 0.0430 - accuracy: 0.9851 -
val loss: 0.7813 - val accuracy: 0.7362
```

Test set accuracy

Training set accuracy

For predicting a single image,

```
import cv2
img=cv2.imread('dataset/predict/9.jpg')
img= cv2.resize(img,(64,64))
img=img.reshape(1,64,64,3)
result= classifier.predict_classes(img)
if(result[0][0] == 1):
        print('this may be a dog')
else
        print('this may be a cat')
```

PRINCIPAL COMPONENT ANALYSIS

It's considered to be most used unsupervised algorithm and can be seen as the most popular dimensionality reduction algorithm

It's used for operations such as

1. Noise filtering
2. Visualization
3. Feature Extraction
4. Stock market predictions
5. Gene data analysis

Goal of PCA is

1. Identify patterns in data
2. Detect correlation between variables

Quick summary about PCA

- It's not like linear regression although it may look like it because PCA rather predicting the values, it attempts to learn about the relationship between X and y values
- Finds list of principal axes
- It is highly affected by outliers in the data

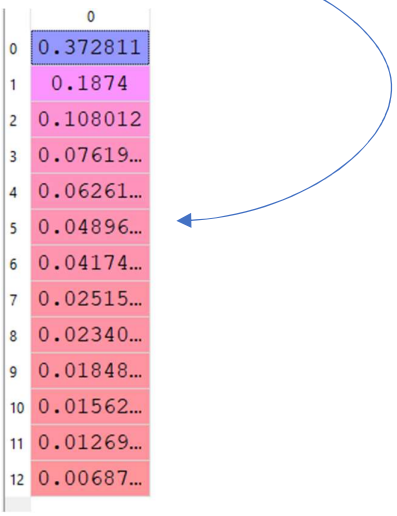
❖ *From m independent variables of your dataset, PCA extracts p<=m new independent variables that explain the most the variance of the dataset* ***regardless of the dependent variable.*** That makes PCA an unsupervised model because of fact that Dependent variable is not considered

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt


dataset= pd.read_csv('Wine.csv')

X=dataset.iloc[:,0:13].values

y=dataset.iloc[:,13].values

from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=0)


from sklearn.preprocessing import StandardScaler

sc= StandardScaler()

X_train = sc.fit_transform(X_train)

X_test= sc.transform(X_test)


from sklearn.decomposition import PCA

pca= PCA(n_components=2) -> we select none at initially
```

X_train=pca.fit_transform(X_train)

X_test=pca.transform(X_train)

explained_version = pca.explained_variance_ratio_



	0
0	0.372811
1	0.1874
2	0.108012
3	0.07619…
4	0.06261…
5	0.04896…
6	0.04174…
7	0.02515…
8	0.02340…
9	0.01848…
10	0.01562…
11	0.01269…
12	0.00687…

So here we take first two principal components which are ranked in decreasing order, according to their variance

37% + 18% = 53%

Now, we change n_components = 2