

ML LEARNING

A-Z

- Link to get datasets and codes and slides :
<https://www.superdatascience.com/machine-learning>
- The coding is going to be done on colab or any other ide.

- **MACHINE LEARNING WORKFLOW:**

There are three main steps:



Data Pre-Processing

- Import the data
- Clean the data
- Split into training & test sets



Modelling

- Build the model
- Train the model
- Make predictions



Evaluation

- Calculate performance metrics
- Make a verdict

Importance of splitting the data into training and test set:

- For example : suppose you have to predict the sale prices of the car, and of course in this scenario we have one dependent variable and some independent .
- Splitting of data into training and testing set is generally done in the ratio of 80:20
- Training set is used for training the model and testing is used for testing the trained model.

Feature scaling:

- Feature scaling is always applied to column and never applied across columns.
- There are so many feature scaling techniques but we will see normalization and standardisation
- After feature scaling a column by using normalization all the values in the column all scaled down between 0 and 1
- But by using standardization the values are scaled down upto between -3 and 3.

Normalization

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$





[0 ; 1]

Standardization






$$X' = \frac{X - \mu}{\sigma}$$

[-3 ; +3]

Applying feature scaling to the following problem:

		
	70,000 \$	45 yrs
	60,000 \$	44 yrs
	52,000 \$	40 yrs

(Before feature scaling the salary
And the age column)

		
	1	1
	0.4444	0.8
	0	0

(After feature scaling the columns
of the salary and age)

Step 1: Data preprocessing

Data preprocessing tools:

- Importing the libraries
 - Importing dataset
 - Take care of missing data
 - Encoding categorical data
1. Encoding independent variable
 2. Encoding the decoding variable
- Splitting the dataset into training set and testing set
 - Feature scaling

Dataset used during the course :

Country	Age	Salary	Purchased
France	44	72000	No
Spain	27	48000	Yes
Germany	30	54000	No
Spain	38	61000	No
Germany	40		Yes
France	35	58000	Yes
Spain		52000	No
France	48	79000	Yes
Germany	50	83000	No
France	37	67000	Yes

1. Importing the libraries



```
▼ Importing the libraries  
1 import numpy as np  
2 import matplotlib.pyplot as plt  
3 import pandas as pd
```

- We have to import three libraries that are generally used:
 1. Numpy as it is used for working with array .
 2. Matplotlib as it is used for plotting the graphs efficiently .
 3. Pandas as it is used for importing the dataset for implementation and also used for feature scaling.

2. Importing the dataset

- For importing dataset you have to get data set csv file uploaded on colab then by using the pandas library we can import the csv file.
- By using the `.read_csv` , it will read the entire dataset and will create a data frame.
- The data frame create will have the same number of columns and values in the data frame.
- Next step is to create a matrix of features(independent variables) and dependent(that are generally the last column of the dataset) variable vector.
- For creating X that is the matrix of features we have to take all the rows but only the independent variable columns .
- This is done by using `iloc` that stands for locate indexes function that is used for taking the indexes of the columns and rows we want to extract from the data set.
- In the `iloc` function you will have to mention the rows and the columns for X, y.

```
[2]  1 dataset = pd.read_csv('Data.csv')  
    2 X = dataset.iloc[:, 1:-1].values  
    3 y = dataset.iloc[:, 1].values
```

3. Taking care of missing values

- Missing values in dataset causes errors when training ml model, so to handle them there are several ways to handle the missing values: the first way is to ignore the observations by deleting them and there is a second way as well that is done by using Simple imputer class of sklearn.
- Then to create an object of simple imputer class that works by replacing the values by the mean or mode of the column . In this case we are replacing them by mean value of the column.
- For using the simple imputer class we will have to import it from sklearn libraries impute module.
- Then by using fit method we will connect the imputer with the matrix of feature,
- For doing the replacement we have to use transform method that will replace missing values with the mean/average.



```
1 from sklearn.impute import SimpleImputer
2 imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
3 imputer.fit(X[:, 1:3])
4 X[:, 1:3] = imputer.transform(X[:, 1:3])
```

3. Encoding the categorical data

- Encoding the independent variables

- Encoding of categorical data as it will be difficult for the ml model to compute some correlations between these columns, So these categorical data values are converted into numbers or binary values.
- There are techniques we can use for encoding the categorical data like one hot encoding, etc.
- In this case we are using one hot encoding for encoding the categorical data into three columns for this ColumnTransformer class is used.
- Then by using fit_transformer method you can relate it with feature matrix

```
1 from sklearn.compose import ColumnTransformer
2 from sklearn.preprocessing import OneHotEncoder
3 ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder='passthrough')
4 X = np.array(ct.fit_transform(X))
```

- Encoding the dependent variables
- So for encoding the categorical data of the dependent variable that is to convert the data as yes -1 and no-0)
- We will use labelencoder class from preprocessing module
- This labelencoder will convert the dependent variable values as 0 and 1
- By using fit_transform method you will connect with the dependent variable y.

```
10| 1 from sklearn.preprocessing import LabelEncoder  
    2 le = LabelEncoder()  
    3 y = le.fit_transform(y)
```

NOTE:

Most frequently asked question:

Q.1. Do we have to apply feature scaling before or after splitting the dataset, why?

Ans:

We should apply feature scaling after splitting the dataset, because test set has to be a brand new set on which we will evaluate the machine learning model but feature scaling generally uses a technique of calculating mean and using it but if we apply feature scaling on the original dataset then it will be like a information leak since the feature scaling will use all the data value of the column for calculating mean and standard deviation and that will include all values including the data to be used for testing the model.

4. Splitting dataset into training and testing set

- Splitting the dataset into training and testing is done by using a module of sklearn library that is `model_selection` so from this module we will import `train_test_split` class.
- Then by using it we will split the dataset into 4 values that is `X`(matrix of features) into `x_train` and `x_test` and `y`(dependent variable) into `y_test` ,`y_train`.
- The arguments passed through the `train_test_split` method includes `test_size` indicating the size of the testing set,`random_state` for generating random values.

```
(11) 1 from sklearn.model_selection import train_test_split  
      2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 1)
```

5. Feature scaling

- Feature scaling is a process that allows to put all the features on same scale.
- To avoid dominance of some feature over others.
- The main two feature scaling techniques are : standardisation and normalisation.
- Normalisation is used when you have normal distribution between features, but standardisation works well all the time ,So we generally use standardisation.
- We will see that we have to apply on both X_train and X_test,
- We will get the mean and standard deviation of the values of X_train and apply feature scaling on X_train and by using the same mean and standard deviation from X_train we will apply on X_test to scale the X_test(in order to prevent info leak).
- We do not apply the feature scaling on dummy variables in the feature matrix.
- By using StandardScaler class from sklearn preprocessing we will do scaling.

```
1 from sklearn.preprocessing import StandardScaler
2 sc = StandardScaler()
3 X_train[:, 3:] = sc.fit_transform(X_train[:, 3:])
4 X_test[:, 3:] = sc.transform(X_test[:, 3:])
```

Step 2: Regression

- Simple linear regression

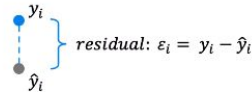
- The equation of simple regression problem.
- By using the equation we can plot the graph and then how do we choose the best slop line,so that is done by using **ordinary least square**.

$$\hat{y} = b_0 + b_1 X_1$$

Diagram illustrating the components of the simple linear regression equation:

- \hat{y} : Dependent variable
- b_0 : y-intercept (constant)
- b_1 : Slope coefficient
- X_1 : Independent variable

Ordinary Least Squares:

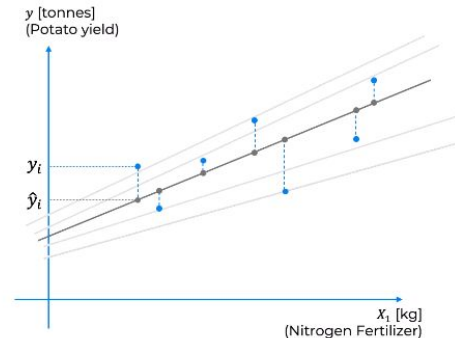


residual: $\varepsilon_i = y_i - \hat{y}_i$

$$\hat{y} = b_0 + b_1 X_1$$

b_0, b_1 such that:

$SUM(y_i - \hat{y}_i)^2$ is minimized



● Building Regression Model

- First step will be data preprocessing that is to import the libraries ,dataset then splitting the dataset.
- For **creating a simple linear regression model** we will start by importing the LinearRegression class from sklearn linear_model
- Then creating the instance of the LinearRegression class will be our step 2 and for simple regression problem we don't have to give any parameter while creating the instance.
- Then for connecting the instance with the training dataset will be done by using fit() method.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
```

Importing the dataset

```
1 dataset = pd.read_csv('Salary_Data.csv')
2 X = dataset.iloc[:, :-1].values
3 y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

```
1 from sklearn.model_selection import train_test_split
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

```
1 from sklearn.linear_model import LinearRegression
2 regressor = LinearRegression()
3 regressor.fit(X_train, y_train)
```

Predicting the Test set results

- **Predicting the test set results**
- According to the dataset used in this method we have to predict the salary for years of experience from the test set. that means we have to pass `X_test` as an parameter to predict function for obtaining predicted value.
- Then the next step is compare actual test result with predicted results , i.e. actual salaries and predicted salaries by plotting a graph.

```
1 y_pred = regressor.predict(X_test)
```

- **Visualising the training set results:**

For visualising the trained set by comparing actual salaries and predicted ones we will matplotlib library pyplot module and in the graph X-axis(years of experience) and y-axis(salaries). The red points represents actual data and blue is the predicted value. And the same model for plotting the test set too.

```
1 plt.scatter(X_train, y_train, color = 'red')
2 plt.plot(X_train, regressor.predict(X_train), color = 'blue')
3 plt.title('Salary vs Experience (Training set)')
4 plt.xlabel('Years of Experience')
5 plt.ylabel('Salary')
6 plt.show()
```

```
1 plt.scatter(X_test, y_test, color = 'red')
2 plt.plot(X_train, regressor.predict(X_train), color = 'blue')
3 plt.title('Salary vs Experience (Test set)')
4 plt.xlabel('Years of Experience')
5 plt.ylabel('Salary')
6 plt.show()
```

Salary vs Experience (Test set)

● Multiple linear regression

- Multiple linear regression is similar to a simple linear regression but there only one difference between them that is the number of independent variable on which y that dependent variable depends. With the same number of slope coefficient as that of the number of independent variables.

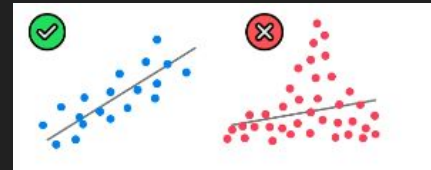
● Assumptions of linear regression

Before using linear regression for any dataset you should make sure that the dataset is fit for the model or not. That's why we assumptions, In total we have 5 assumptions:

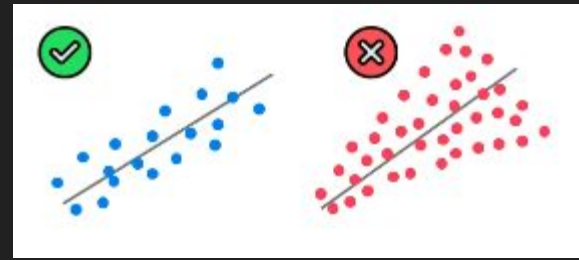
1. Linearity(linear relationship between Y and each X)

$$\hat{y} = b_0 + b_1X_1 + b_2X_2 + \dots + b_nX_n$$

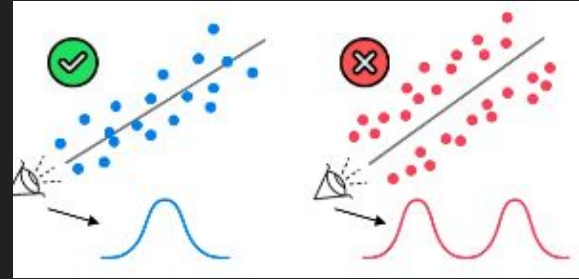
The diagram shows the equation $\hat{y} = b_0 + b_1X_1 + b_2X_2 + \dots + b_nX_n$ with vertical lines connecting terms to labels below. \hat{y} is labeled 'Dependent variable'. b_0 is labeled 'y-intercept (constant)'. b_1 is labeled 'Slope coefficient 1'. X_1 is labeled 'Independent variable 1'. b_2 is labeled 'Slope coefficient 2'. X_2 is labeled 'Independent variable 2'. b_n is labeled 'Slope coefficient n'. X_n is labeled 'Independent variable n'.



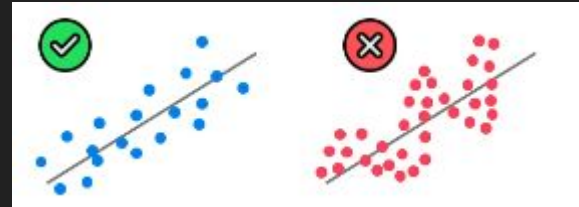
2. Homoscedasticity (equal variance)



3. Multivariate Normality (Normality of error distribution)



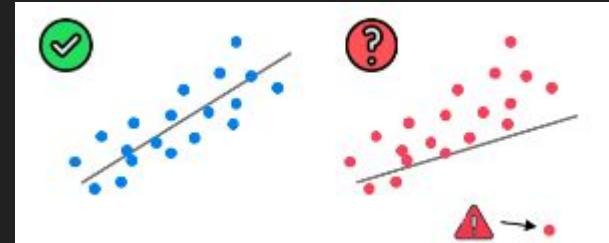
4. Independence (of observation that generally include “no autocorrelation” that in the data there shouldn’t be any pattern if the dataset has some pattern that means the rows are not independent)



5. Lack of Multicollinearity(predictor/features are not correlated with each other)but even if the features are correlated and if proceed to build the model then coefficient estimated are unreliable.

✓ $X_1 \not\sim X_2$ ✗ $X_1 \sim X_2$

6. The outlier check(this is not an assumption but an extra)



- Before using the dataset you should not check the assumptions.

<https://www.superdatascience.com/blogs/assumptions-of-linear-regression>

● Dummy variables

- Dummy variables are the variable we use for representing categorical data.
- These are generally variables or binary values .They are coded as 0 and 1.
- These 0 and 1 represents the presence and absence of the particular categorical value.

State	Dummy Variables	
	New York	California
New York	1	0
California	0	1
California	0	1
New York	1	0
California	0	1

● Dummy variables in Multiple linear regression

					Dummy Variables	
Profit	R&D Spend	Admin	Marketing	State	New York	California
192,261.83	165,349.20	136,897.80	471,784.10	New York	1	0
191,792.06	162,597.70	151,377.59	443,898.53	California	0	1
191,050.39	153,441.51	101,145.55	407,934.54	California	0	1
182,901.99	144,372.41	118,671.85	383,199.62	New York	1	0
166,187.94	142,107.34	91,391.77	366,168.42	California	0	1

$$y = b_0 + b_1 * x_1 + b_2 * x_2 + b_3 * x_3 + b_4 * D_1$$

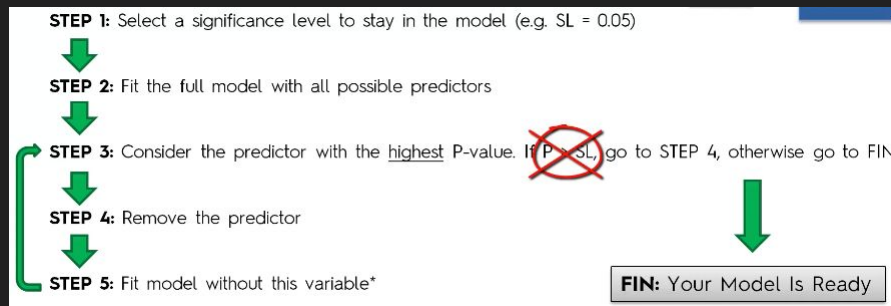
- In this case only new york column is used and the equations of y also change according to the dummy variable used.
- Generally if you have 2 categories then both are variables are never used in the equation at the same time since it will cause multicollinearity and will make the model not able to distinguish the effects of both the dummy variables, this is known as dummy variable trap. (that means always omit one dummy variable every time).

- **P-VALUES and Statistical significance in hypothesis training**

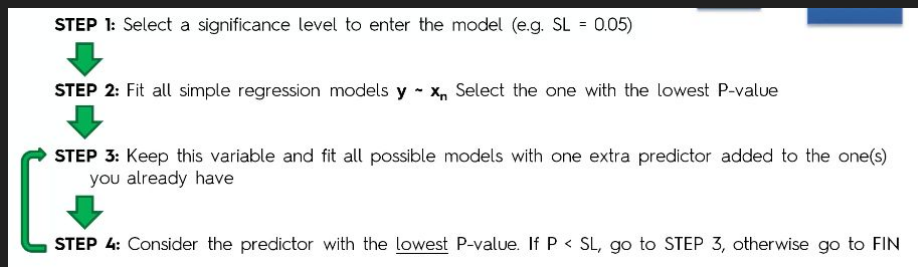
- In **regression analysis**, the **p-value** helps us decide whether a specific **independent variable (feature)** has a **statistically significant impact** on the **dependent variable (output)**.
- It is the probability of **observing the given data (or something more extreme)** **assuming** that the **null hypothesis(it treats everything equally)** is true
- In simpler terms, it tells you how likely it is that the **coefficient (slope)** of a variable is actually **zero** — meaning the variable has **no effect**
- variable is said to be **statistically significant** if its **p-value is less than a chosen threshold (α)**
- **If p-value < 0.05 :**
 - Reject the null hypothesis.
 - The variable **likely has a real impact(output)**
 - It is **statistically significant**
- **If p-value ≥ 0.05 :**
 - Fail to reject the null hypothesis.
 - The variable may **not have a meaningful effect.**

Building a model

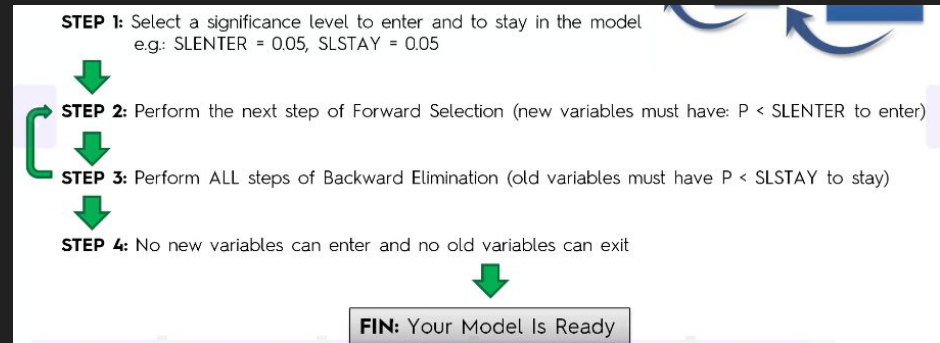
- We should not use all the independent variable for model instead use only some them this is because of two reasons: 1. Garbage in -garbage out, 2. complicated explanation.
- **5 Methods of building models:**
- All-in: Using all the variables.
- Backward Elimination:



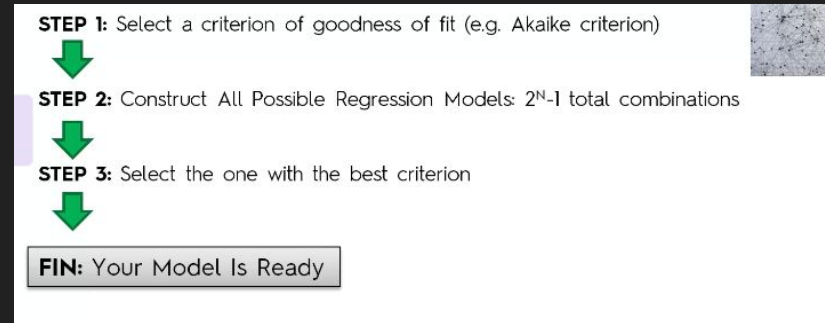
- Forward Selection:



- Bidirectional elimination:



- All possible models:
This is not a good approach



Steps by steps building of Model

- First step is data preprocessing of the dataset.
- Class to build the regression model is same as that of the simple linear regression model.
- The next step will be to calculate the predicted value by using predict method
- For display two vectors that is actual profit and predicted value together we have to use concatenate a numpy function.

```
1 from sklearn.linear_model import LinearRegression
2 regressor = LinearRegression()
3 regressor.fit(X_train, y_train)
```

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, n

```
1 y_pred = regressor.predict(X_test)
2 np.set_printoptions(precision=2)
3 print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1),
```

[[103015.2 103282.38]

● Polynomial regression

- In polynomial regression there will be only one independent variable.
- The equation of polynomial regression is similar to that of multiple linear regression but the only difference is unlike multiple linear regression here in polynomial regression only has one independent variable but with different powers
- This model is still referred as linear in spite of being polynomial. because linear here is referring to coefficients of the equation and not the x or features cause in the polynomial regression x can have a nonlinear relationship with y but in this scenario the value of y depends on coefficients like b_0 b_1 , etc and states that is coefficients are linear.

$$y = b_0 + b_1x_1 + b_2x_1^2 + \dots + b_nx_1^n$$

- **Build the model**
- First step is data preprocessing that includes importing of libraries and dataset.
- Splitting of dataset
- Then constructing the matrix of powered features by using PolynomialFeatures class ,the instance of the class will take a parameter degree representing power.
- Then creating linear regression class for integrating the polynomial x with y .
- Visualising the linear regression model on the dataset for the difference.
- Visualising it on polynomial regression as well.
- As you visualise the dataset you may notice the clear difference in the linear and polynomial linear regression for the particular dataset

```
1 from sklearn.preprocessing import PolynomialFeatures
2 poly_reg = PolynomialFeatures(degree = 2)
3 X_poly = poly_reg.fit_transform(X)
4 lin_reg_2 = LinearRegression()
5 lin_reg_2.fit(X_poly, y)
```

```
1 plt.scatter(X, y, color = 'red')
2 plt.plot(X, lin_reg.predict(X), color = 'blue')
3 plt.title('Truth or Bluff (Linear Regression)')
4 plt.xlabel('Position Level')
5 plt.ylabel('Salary')
6 plt.show()
```

```
1 plt.scatter(X, y, color = 'red')
2 plt.plot(X, lin_reg_2.predict(poly_reg.fit_transform(X)), color = 'blue')
3 plt.title('Truth or Bluff (Polynomial Regression)')
4 plt.xlabel('Position Level')
5 plt.ylabel('Salary')
6 plt.show()
```

fig: Truth or Bluff (Polynomial Regression)

- Predicting the y according to some input value.
- For that you have to use predict method and give input as parameter to it.

▼ Predicting a new result with Linear Regression

```
[10] 1 lin_reg.predict([[6.5]])
```

```
array([330378.78787879])
```

▼ Predicting a new result with Polynomial Regression

```
1 lin_reg_2.predict(poly_reg.fit_transform([[6.5]]))
```

```
array([[15.46245265155]])
```

Support vector regression:

Q. How does Support vector regression differ from linear regression?

Answer:

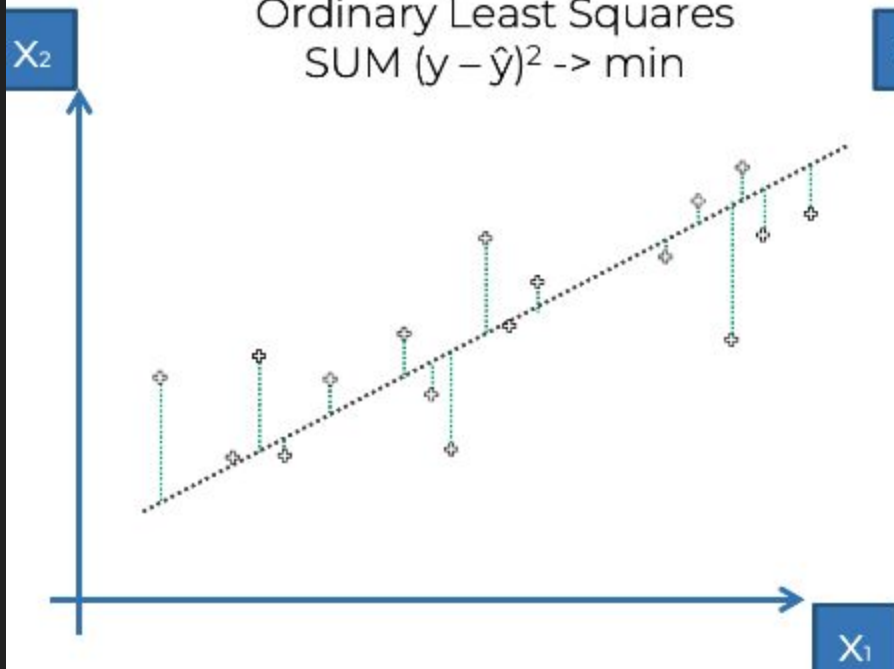
Lets see it by using two graph with some random data points plotted on the graph on graph .

- If we will apply simple regression on the first graph then you will be able to generate the regression line by using ordinary least square method.
- In svr instead of simple line you will see tube ,a tube containing regression in the middle and tube around it the tube has the width of measured vertically ,such that the tube is called epsilon-insensitive tube.the tube says that all the data points that lies within the tube can have and not care about any error inside here.And the points lying outside the tube will calculate the distance between the point and the tube,and for points outside the tube we do care about error .
- The points outside the tube are supporting the structure of the tube.

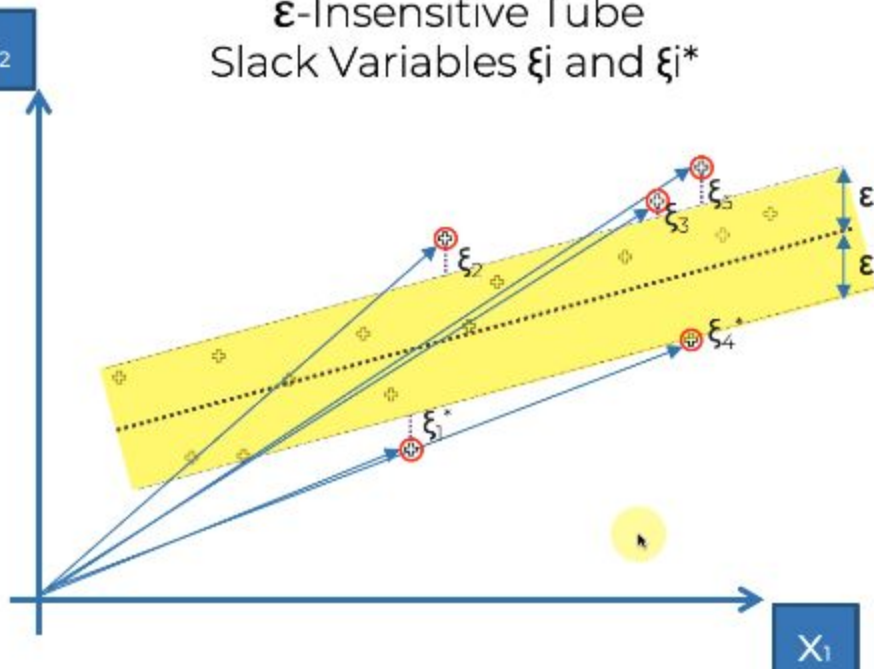
SVM Introduction

$$\frac{1}{2} \|w\|^2 + C \sum_{i=1}^m (\xi_i + \xi_i^*) \rightarrow \min$$

Ordinary Least Squares
 $\text{SUM } (y - \hat{y})^2 \rightarrow \min$



ϵ -Insensitive Tube
Slack Variables ξ_i and ξ_i^*



● Building the SVR Model

- First step is data preprocessing.
- Second step is feature scaling unlike other regression model in this model we have to apply feature scaling.
- But during this case feature scaling we will not be just applied X_{train} and x_{test} it is going to be applied on both Entire X and entire y .
- Training the SVR model on the dataset in which we will use rbf kernel that learn non-linear relationships.

```
1 from sklearn.preprocessing import StandardScaler
2 sc_X = StandardScaler()
3 sc_y = StandardScaler()
4 X = sc_X.fit_transform(X)
5 y = sc_y.fit_transform(y)
```

```
1 from sklearn.svm import SVR
2 regressor = SVR(kernel = 'rbf')
3 regressor.fit(X, y)
```

- Predicting the value of y by giving a single value as input

```
1 sc_y.inverse_transform(regressor.predict(sc_X.transform([[6.5]]).reshape(-1,1))
```

- Visualising the model by using matplotlib

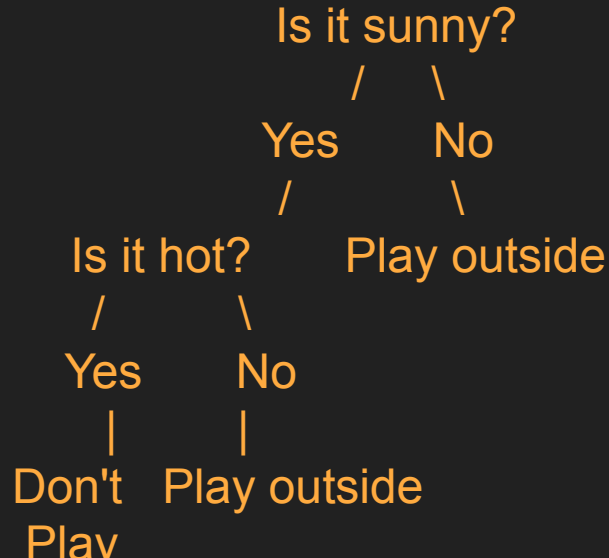
```
1 plt.scatter(sc_X.inverse_transform(X), sc_y.inverse_transform(y), color = 'red')
2 plt.plot(sc_X.inverse_transform(X), sc_y.inverse_transform(regressor.predict(X).reshape(-1,1)), color = 'blue')
3 plt.title('Truth or Bluff (SVR)')
4 plt.xlabel('Position level')
5 plt.ylabel('Salary')
6 plt.show()
```

- Visualising the svr model with higher resolution.

```
1 X_grid = np.arange(min(sc_X.inverse_transform(X)), max(sc_X.inverse_transform(X)), 0.1)
2 X_grid = X_grid.reshape((len(X_grid), 1))
3 plt.scatter(sc_X.inverse_transform(X), sc_y.inverse_transform(y), color = 'red')
4 plt.plot(X_grid, sc_y.inverse_transform(regressor.predict(sc_X.transform(X_grid)).reshape(-1,1)), color = 'blue')
5 plt.title('Truth or Bluff (SVR)')
6 plt.xlabel('Position level')
7 plt.ylabel('Salary')
8 plt.show()
```

Decision Tree:

- Decision tree is a supervised learning algorithm used for both classification and regression problem.
- It splits the data set on the bases of the features that provide the most information by using methods like information gain, etc.
- Decision tree is used for getting the next node of the tree.
- Decision tree regression model is generally used for multiple feature dataset.
- Following is an example of decision where the final result is deciding whether to play outside or not:



- **Step by step Implementation of decision tree.**

- Start by importing the necessary libraries to be used.
- Import the dataset and divide it into X as feature matrix, and y as array of dependent variable.
- We do not have to apply feature scaling in neither decision tree nor random forest regression as to obtain the result we split the data instead of using equation so there is no need to scale the features.
- For the model we have to import decisiontreeregressor class from tree module of sklearn.
- For predicting you just have to use predict method.
- Then visualising the model by using matplotlib

```
[3]: 1 from sklearn.tree import DecisionTreeRegressor
      2 regressor = DecisionTreeRegressor(random_state = 0)
      3 regressor.fit(X, y)
```

Random forest regression:

- Random forest intuition:

STEP 1: Pick at random K data points from the Training set.



STEP 2: Build the Decision Tree associated to these K data points.

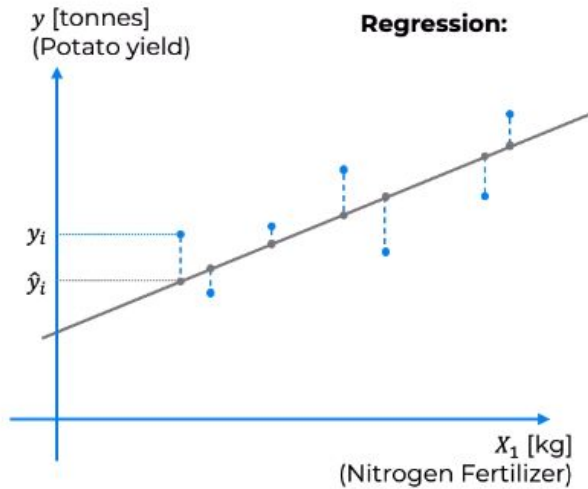


STEP 3: Choose the number N_{tree} of trees you want to build and repeat STEPS 1 & 2

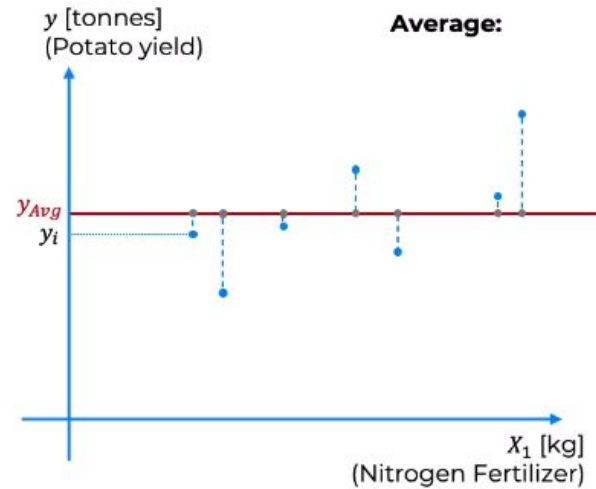


STEP 4: For a new data point, make each one of your N_{tree} trees predict the value of Y to for the data point in question, and assign the new data point the average across all of the predicted Y values.

Evaluating the performance of the model (R square method)



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



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

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

- R square method always gives value between 0-1
- Rule of thumb:
- 1.0 = Perfect fit(suspicious)
- ~0.9 = Very good
- <0.7 = Not great
- <0.4 = Terrible
- <0 = Model makes no sense
- That means the greater the value of R^2 is the good the fit is

Adjusted R square:

$$Adj R^2 = 1 - (1 - R^2) \times \frac{n - 1}{n - k - 1}$$

K= number independent variable(if k increases the value of adjusted r^2 decreases)

N = sample size

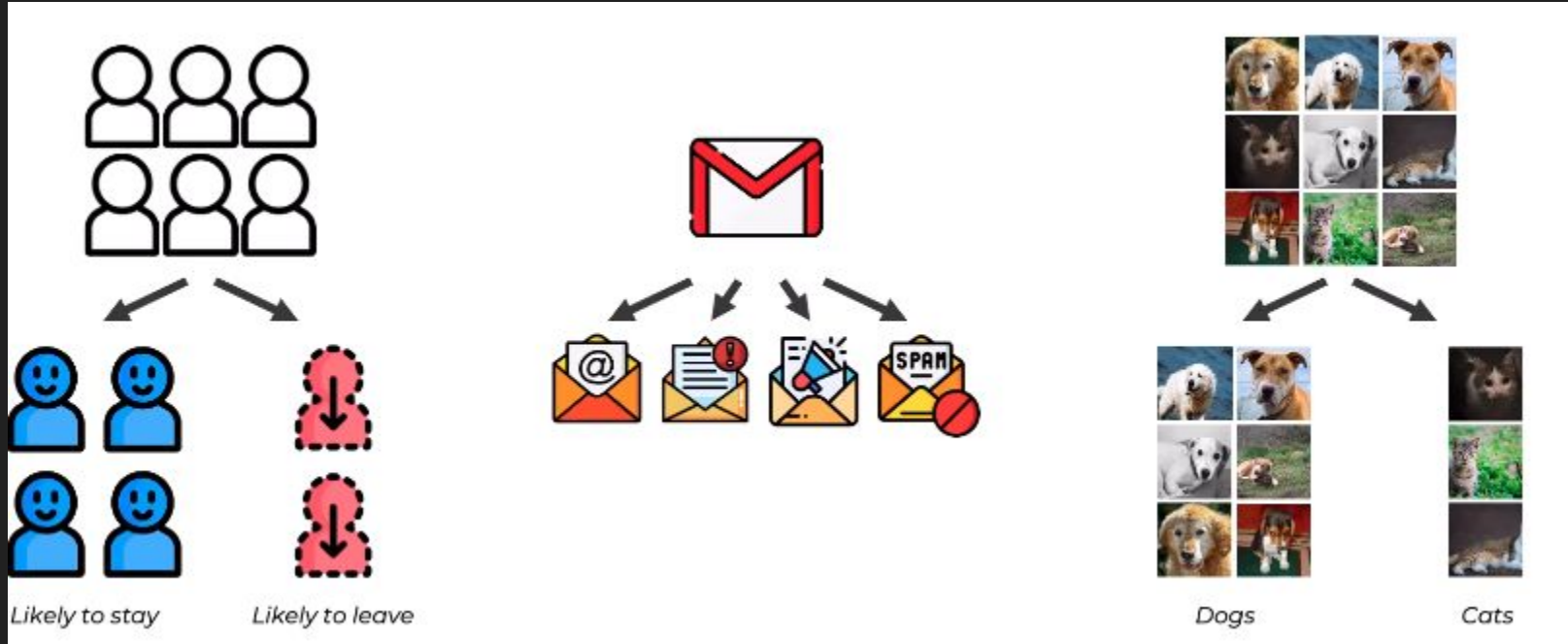
- If you are trying to add a new feature then instead of using r^2 method we can use adjusted r^2 method to evaluate the method.

Model selection:

- Model selection is the process of choosing the best-performing model for a specific dataset.
- It involves evaluation and comparing of different models to identify the one that best fit the dataset and produces the desired output.

CLASSIFICATION MODELS:

- Classification :A machine learning technique to identify the category of new observations based on training data.
- Example:



1. Logistic Regression

- Logistic regression : Predict a categorical dependent variable from a number of independent variables.
- Logistic regression uses sigmoid function for categorising the data.



Will purchase
health insurance:
Yes / No

~



Age



Income



Level of
Education



Family or
Single

$$\ln \frac{p}{1-p} = b_0 + b_1X_1 + b_2X_2 + b_3X_3 + b_4X_4$$

Steps to build the logistic regression model

- The dataset we are using during the implementation is predicting whether the older customers are going to buy a car or not.
- In the dataset we have independent variables(Age,estimated salary) and dependent variable(purchased)
- Then train the Logistic regression model imported from sklearn linear_model module ,then create its instance and apply the fit method to reflect the changes.
- Then by using predict method for particular value.
- Comparing the y_pred and y_test values



```
1 from sklearn.linear_model import LogisticRegression
2 classifier = LogisticRegression(random_state = 0)
3 classifier.fit(X_train, y_train)
```

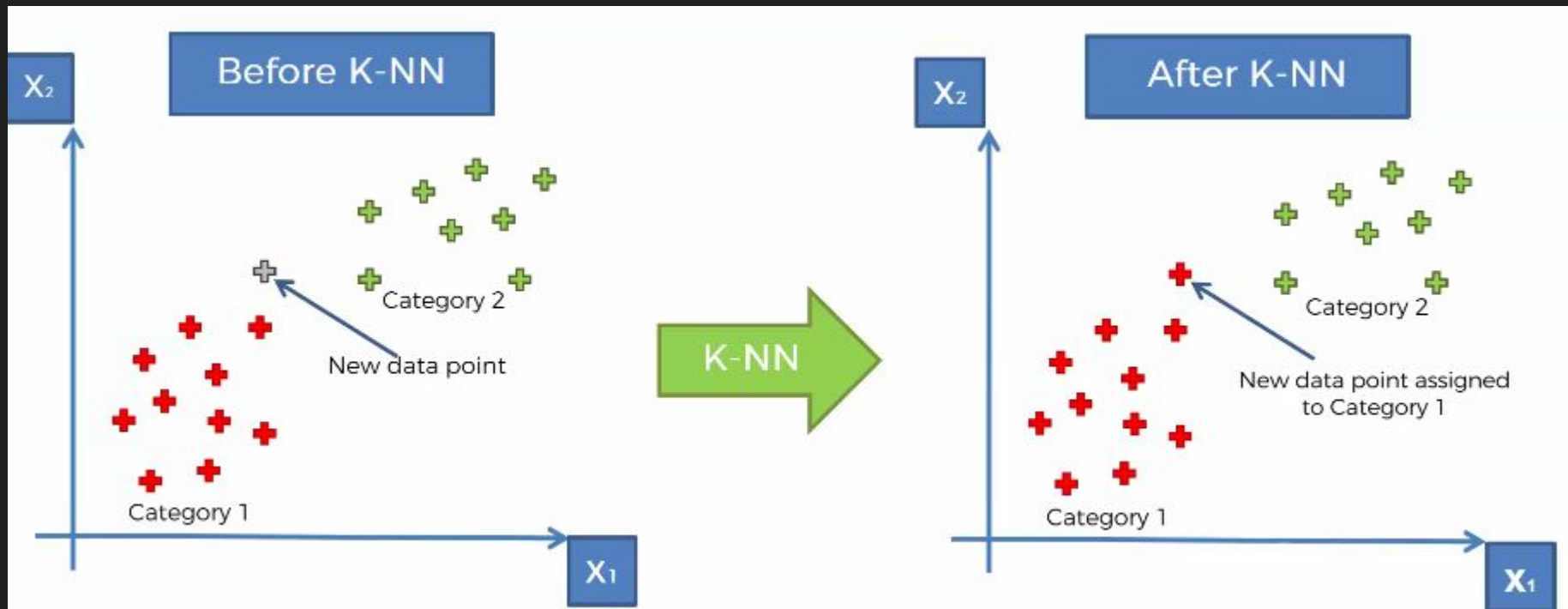
```
1 y_pred = classifier.predict(X_test)
2 print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
[0 0]
```

- Confusion matrix:for the construction of confusion matrix we need to use class confusion_matrix from metrics module of sklearn library.
- Accuracy can be calculated by using accuracy class of metrics module from sklearn library.

```
1 from sklearn.metrics import confusion_matrix, accuracy_score
2 cm = confusion_matrix(y_test, y_pred)
3 print(cm)
4 accuracy_score(y_test, y_pred)
```

2. KNN (K-nearest neighbour)

- K-nearest neighbour is a supervised machine learning algorithm .
- It works by finding the “k” closest data point(neighbours) to a given input and makes a predictions based on the majority class for classification).



Algorithm to build

- Step 1: Selecting the optimal value of k (number of neighbours to be considered while making predictions).
- Step 2: Calculating distance by using different techniques(e.g. Euclidean, manhattan, and minkowski)
- Step 3: finding the nearest neighbours that is calculating the smallest distance between target points and data points.
- Step 4: Assigning the target points into the category nearest to the target point.

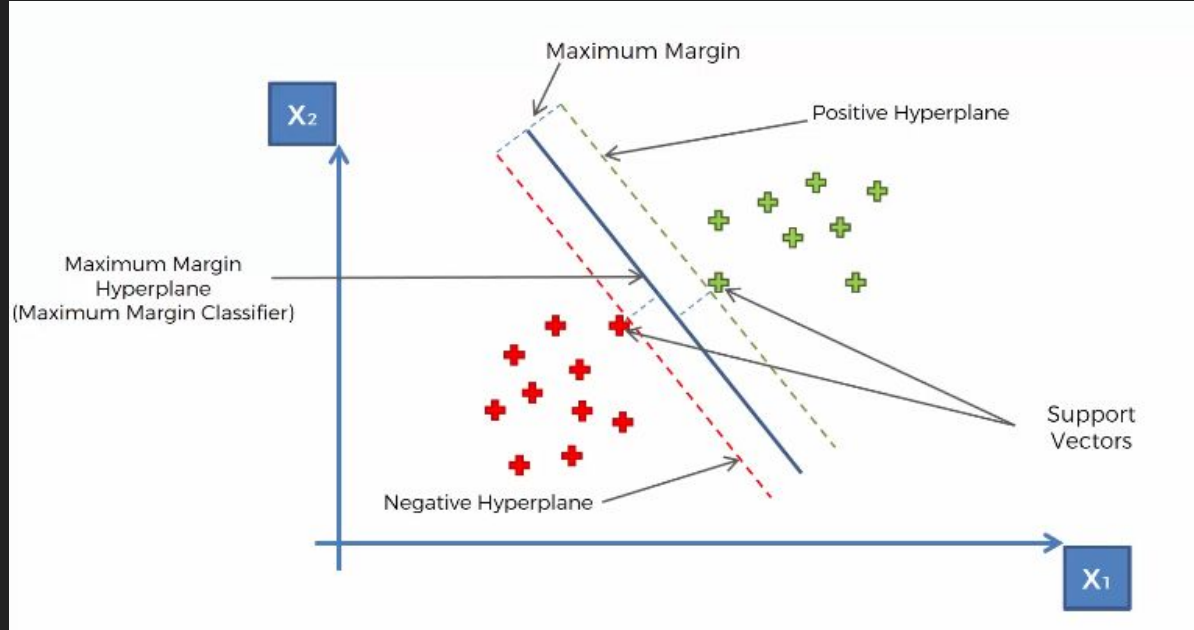
Steps to build the algorithm:

- First step is to preprocess the data that is to import the necessary libraries ,importing dataset.
- Then apply train_test_split to split the dataset into training and testing set .
- Then applying feature scaling on the splitted dataset.
- The training the model on the training set.
- Predicting the value at age 30 and salary-8700 by using predict function.
- Visualising the knn prediction

```
1 from sklearn.neighbors import KNeighborsClassifier
2 classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
3 classifier.fit(X_train, y_train)
```

3. SVM (support vector machine)

- It is a supervised learning algorithm used for both regression and classification.
- SVM aims to find an optimal hyperplane in N-dimension to separate data points into different classes
- The algorithm works by maximising the margin between the closest data points of each class.




Building the SVM model

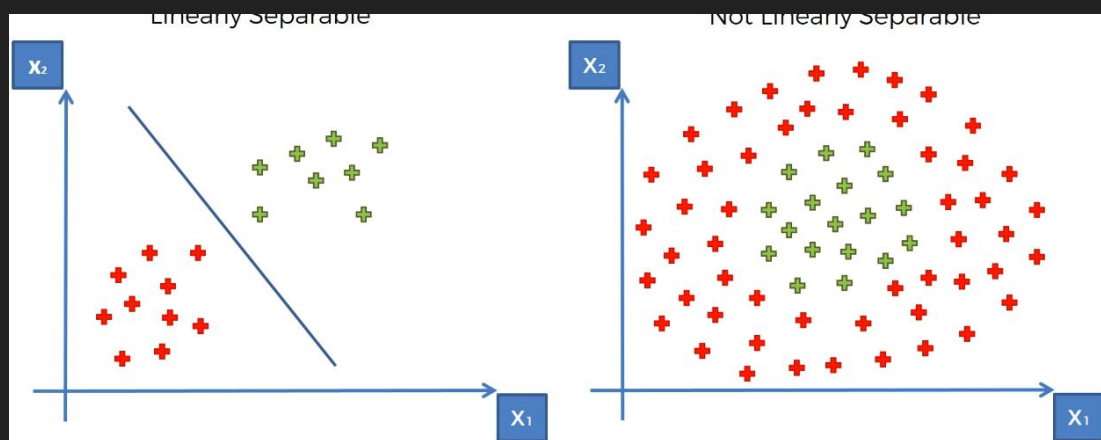
- First step is data preprocessing(importing)
- Second step is splitting the dataset.
- Third is feature scaling the dataset.
- Training the SVM model.
- Predicting the value and test set
- Visualising .

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

4. KERNEL SVM (support vector machine)

Difference between linear-separable and non-linearly separable data.

Feature	Linearly Separable	Nonlinearly Separable	
Separation boundary	Straight line / plane	Curved / complex boundary	
Classifier needed	Linear	Nonlinear	
Simplicity	Easy	More complex	



Why to use Kernel svm?

Solution:

Kernel Svm is a smart trick that helps SVM handle complex cases as in the non-linearly cases where we can't even draw one single decision boundary or linear decision hyperplane.

It works by transforming the data into higher dimension y adding an extra dimension where a straight line can separate the data.

Transforming Nonlinear data for linear separation:

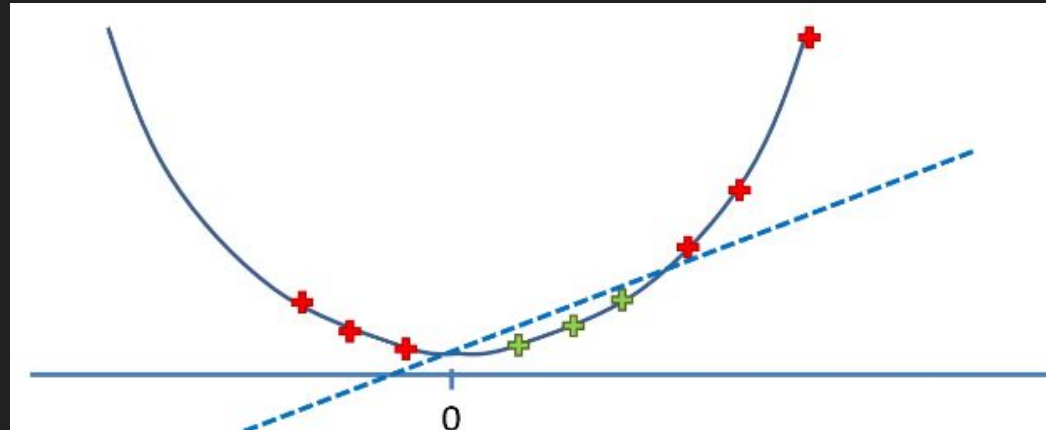
- Taking an example of one dimensional dataset.
- Then mapping to higher dimension.
- Applying the following function on all the data points:

$$f = x - 5$$

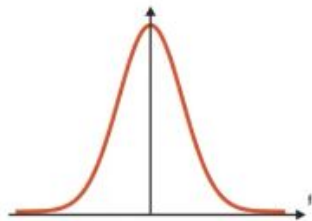
- Now squaring the values :

$$f = (x - 5)^2$$

- After this step your data points will be linearly separable.
- The drawback of this algorithm is that it can be highly compute-intensive

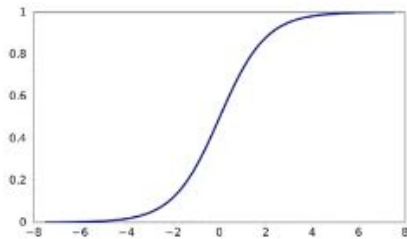


TYPES OF KERNELS:



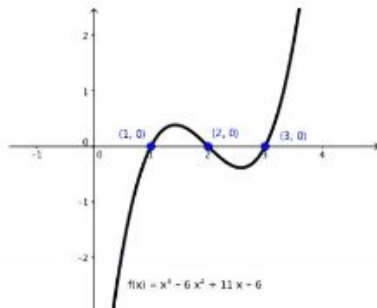
Gaussian RBF Kernel

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



Sigmoid Kernel

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



Polynomial Kernel

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

Building the kernel svm model:

- First step is data preprocessing
- Then second step is to train the model on training dataset ,this step is similar to that of training linear svm
- The only thing that is going change is the kernel parameter.

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

5. Naive Bayes

Q. What is Bayes theorem?

Solution:

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

Bayes theorem is a fundamental concept in probability and statistics used to calculate the probability of an event to happen on the basis of an event that has already occurred.

Example:

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 = ?**

$$\rightarrow P(\text{Mach2}) = 20/50 = 0.4$$

$$\rightarrow P(\text{Defect}) = 1\%$$

$$\rightarrow P(\text{Mach2} \mid \text{Defect}) = 50\%$$

$$\rightarrow P(\text{Defect} \mid \text{Mach2}) = ?$$

$$P(\text{Defect} \mid \text{Mach2}) = \frac{0.5 \quad * \quad 0.01}{0.4} = 0.0125$$

Naive Bayes Algorithm:

Step 1: Calculate the probability of two categories you have.

Step 2: by using bayes theorem you can calculate the probability of the two categories.

Step 3: then compare the probability for both categories.

Step 4: then designate the position .

Important Question why “Naive”?

Solution:

Naive Bayes classifier is called naive because it naively assumes independence among features.

Building the naive bayes classifier :

- Firstly we have to do data preprocessing.
- Second step is to train the GaussianNB model by using the training dataset.
- Third thing is to predict the value of y by giving input .

```
1 y_pred = classifier.predict(X_test)
2 print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

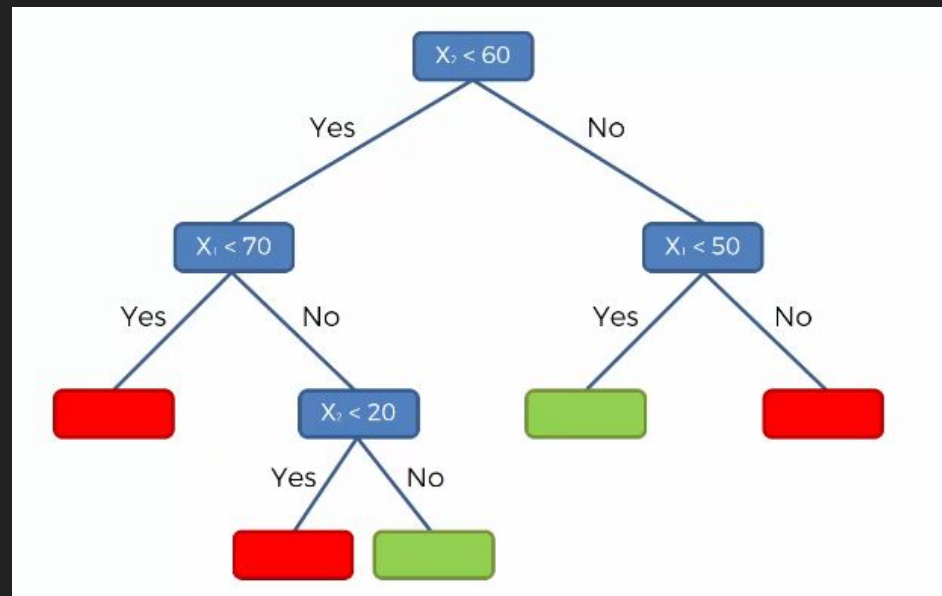
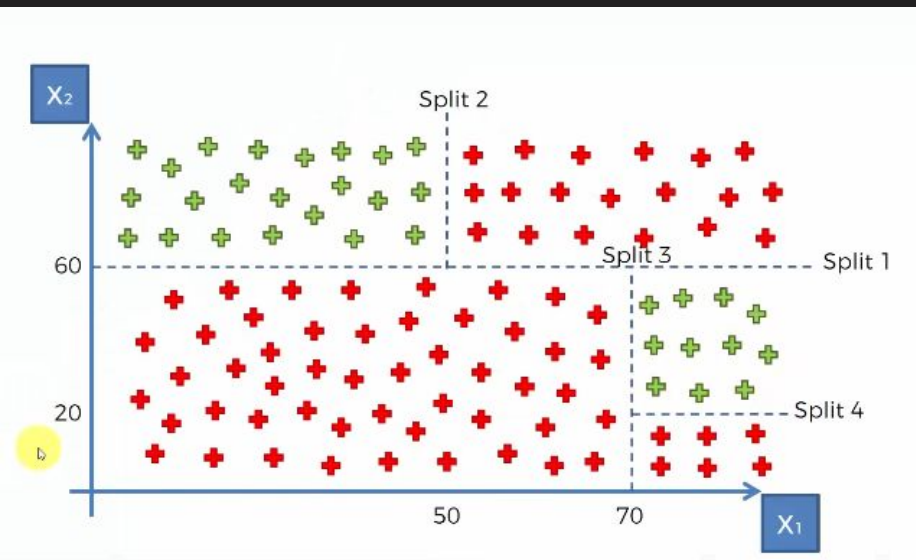
```
1 from sklearn.naive_bayes import GaussianNB
2 classifier = GaussianNB()
3 classifier.fit(X_train, y_train)
```

- Fourth step is to make confusion matrix for y_test and y_pred,
- Fifth step is to check the accuracy of the model.
- Last step is to visualise the model

```
1 from sklearn.metrics import confusion_matrix, accuracy_score
2 cm = confusion_matrix(y_test, y_pred)
3 print(cm)
4 accuracy_score(y_test, y_pred)
```

6. Decision Tree Classification

Example



Implementing Decision Tree Classification model

- First step is to do data preprocessing .
- Training the model DecisionTreeClassifier class from tree module and we have to give entropy as criteria as parameter. .

```
1 from sklearn.tree import DecisionTreeClassifier
2 classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
3 classifier.fit(X_train, y_train)
```

- Predicting the value by giving a value.
- Making confusion matrix and calculating accuracy.
- Visualising the model

7. Random Forest Classification

- It is an ensemble type of algorithm since it runs decision tree classification model a lot of times for getting the majority for result.

STEP 1: Pick at random K data points from the Training set.



STEP 2: Build the Decision Tree associated to these K data points.



STEP 3: Choose the number N_{tree} of trees you want to build and repeat STEPS 1 & 2



STEP 4: For a new data point, make each one of your N_{tree} trees predict the category to which the data points belongs, and assign the new data point to the category that wins the majority vote.

Implementing Decision Tree Classification model

- First step is to do data preprocessing .
- Training the model RandomForestClassifier class from ensemble module and we have to give entropy as criteria and n_estimator as no of trees as parameter.

```
[ ] 1 from sklearn.ensemble import RandomForestClassifier  
    2 classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)  
    3 classifier.fit(X_train, y_train)
```

- Predicting the value by giving a value.
- Making confusion matrix and calculating accuracy.
- Visualising the model

Confusion Matrix:

		Prediction	
		NEG	POS
Actual	NEG	43	12
	POS	4	41

! (in cell: Actual POS, Prediction POS)

! (in cell: Actual POS, Prediction NEG)

Type II Error
(False Negatives)

Type I Error
(False Positives)

Accuracy :

$$AR = \frac{Correct}{Total} = \frac{TN + TP}{Total} = \frac{84}{100} = 84\%$$

Error Rate :

$$ER = \frac{Incorrect}{Total} = \frac{FP + FN}{Total} = \frac{16}{100} = 16\%$$

Recall : Recall measures how well the model identifies all relevant instances.

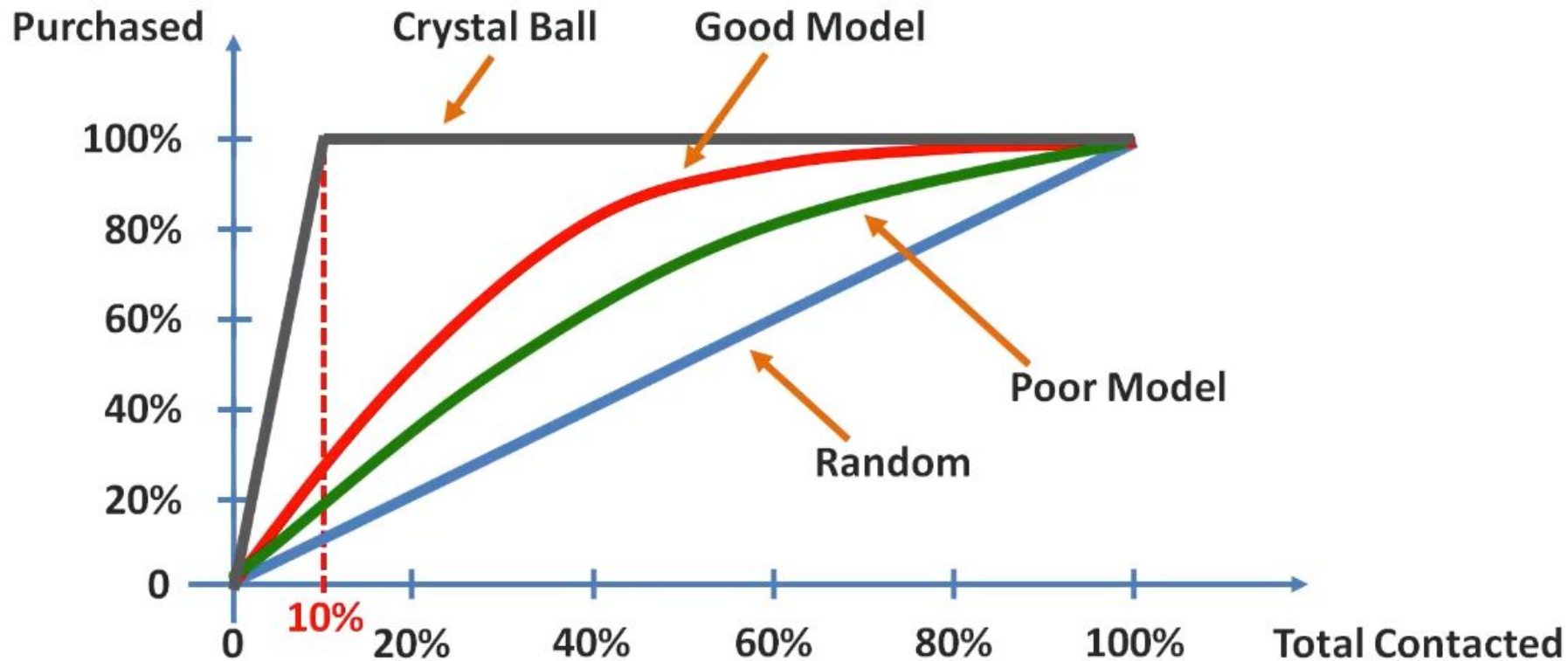
$$Recall = \frac{TP}{TP + FN}$$

Precision: Precision focusses on correctness of positive instances.

$$Precision = \frac{TP}{TP + FP}$$

CAP : Cumulative accuracy profile curve

It is a visualisation tool used for evaluating the performance of classification models.



4. Clustering:

- It is a supervised learning algorithm.
- As it is an unsupervised learning algorithm it does not have any prior information .
- Without any prior knowledge it group the data points into clusters on the basis of their similarities.
- We have two type of clustering technique:
 1. K-means clustering
 2. Hierarchical clustering.

1. K-means clustering

- K-means is a way of automatically group data into k clusters(k is the number you choose or given as an input).
- Steps to do k-means clustering:
 1. First step is to choose k
 2. These k will be placed randomly as they are referred as centroids.
 3. Grouping the points closest to the centroids as group.
 4. Then calculating the averages we will have to change the center of mass
 5. Repeat the steps 3 and 4 again
 6. This process will continue until the centroids will stop moving much.

K-means++

- Using k-means we may have different sets of centroid and this is referred as Random Initialization trap.
- In place of k-means we can use k-means++ method.
- But k-means does not guarantee that there will be no issues.

K-Means++ Initialization Algorithm:

Step 1: Choose first centroid at random among data points

Step 2: For each of the remaining data points compute the distance (D) to the nearest out of already selected centroids

Step 3: Choose next centroid among remaining data points using weighted random selection – weighted by D^2

Step 4: Repeat Steps 2 and 3 until all k centroids have been selected

Step 5: Proceed with standard k-means clustering

Implementing the algorithm:

- First step is data preprocessing that includes importing dataset and libraries.
- Second step is to use elbow method to get the optimal number of clusters .this is to be done by using kmeans class of cluster module and then applying for loop for getting the number of optimal clusters.
- Third step is to train the model by using the same kmeans class.
- Last step is to visualise the plot graph.

```
1 from sklearn.cluster import KMeans
2 wcss = []
3 for i in range(1, 11):
4     kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
5     kmeans.fit(X)
6     wcss.append(kmeans.inertia_)
7 plt.plot(range(1, 11), wcss)
8 plt.title('The Elbow Method')
9 plt.xlabel('Number of clusters')
10 plt.ylabel('WCSS')
11 plt.show()
```

```
1 kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
2 y_kmeans = kmeans.fit_predict(X)

1 print(y_kmeans)
```

```
1 plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
2 plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
3 plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
4 plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
5 plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
6 plt.scatter(kmeans.cluster_centers_[0, 0], kmeans.cluster_centers_[0, 1], s = 300, c = 'yellow', label = 'Centroids')
7 plt.title('Clusters of customers')
8 plt.xlabel('Annual Income (k$)')
9 plt.ylabel('Spending Score (1-100)')
10 plt.legend()
11 plt.show()
```



Because indeed, these are with these customers

2. Hierarchical clustering

- Hierarchical clustering is an unsupervised learning algorithm used to group data points based on their distance and similarities.
- There are two types of hierarchical clustering :
 1. Agglomerative clustering(Bottom-up):each data point is considered as an separate cluster and then we group them on the bases of their closeness.
 2. Divisive clustering(Top-up):All the data points are considered as one big cluster and then also recursively split the cluster until each point is in its own cluster.
- We will use the agglomerative approach
- Dendograms is used for visually representing the structure.

Step by step implementation of agglomerative approach:

STEP 1: Make each data point a single-point cluster → That forms N clusters



STEP 2: Take the two closest data points and make them one cluster → That forms $N-1$ clusters



STEP 3: Take the two closest clusters and make them one cluster → That forms $N-2$ clusters

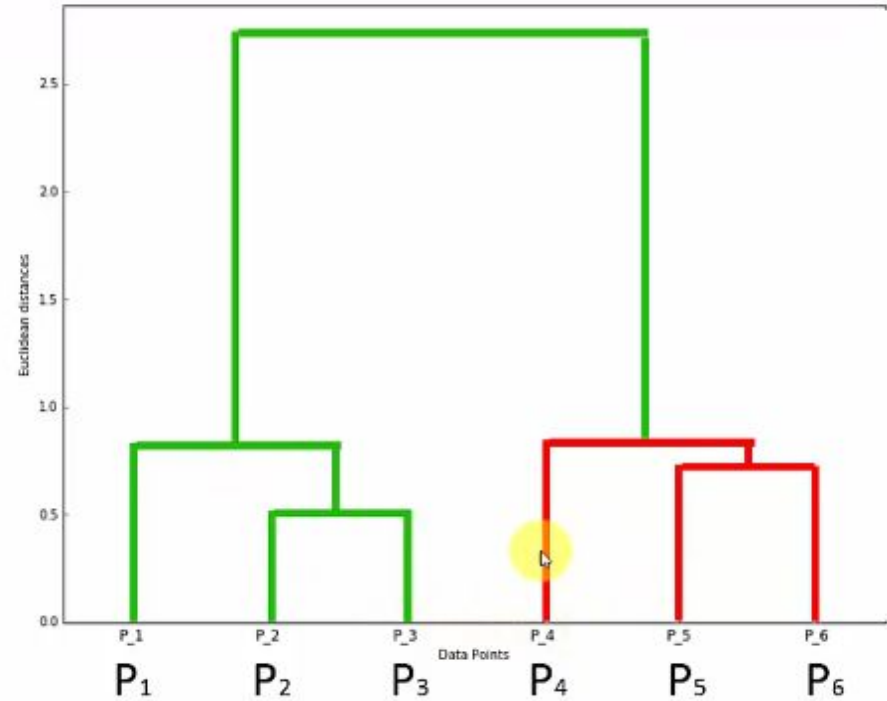
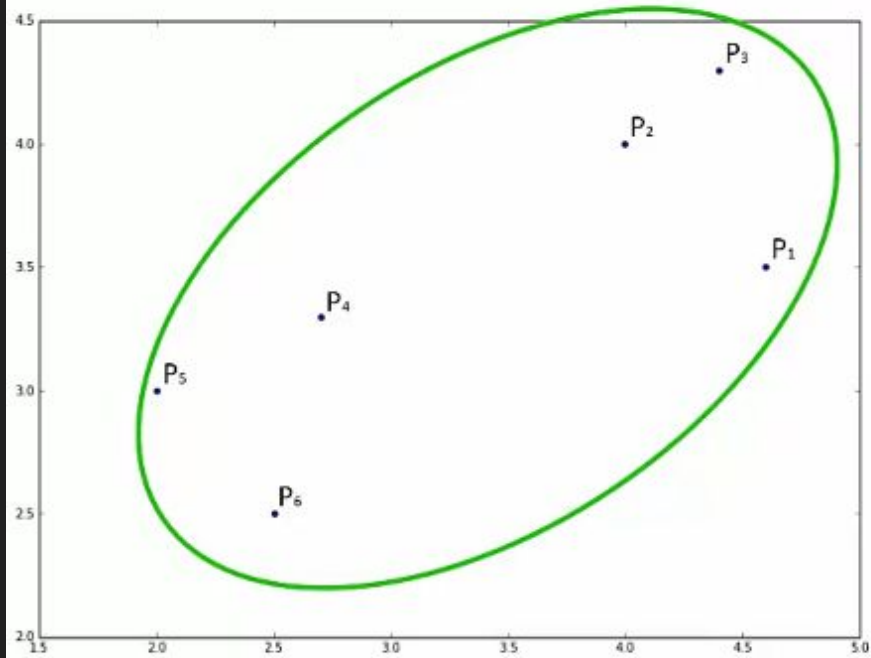


STEP 4: Repeat STEP 3 until there is only one cluster



FIN

How dendrograms work?



Implementation of hierarchical clustering:

- First step is data preprocessing.
- Using dendrograms to find the optimal number of clusters.
- Training the hc model on the dataset by using the class Agglomerativeclustering from module cluster

```
1 import scipy.cluster.hierarchy as sch
2 dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
3 plt.title('Dendrogram')
4 plt.xlabel('Customers')
5 plt.ylabel('Euclidean distances')
6 plt.show()
```

```
[ ] 1 from sklearn.cluster import AgglomerativeClustering
    2 hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = 'ward')
    3 y_hc = hc.fit_predict(X)
```

- Visualising the clusters.

```
[ ] 1 plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
    2 plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
    3 plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
    4 plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
    5 plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
    6 plt.title('Clusters of customers')
    7 plt.xlabel('Annual Income (k$)')
    8 plt.ylabel('Spending Score (1-100)')
    9 plt.legend()
   10 plt.show()
```

Association Rule learning:

1. Apriori:

- It is an classic a algorithm primarily used in association rule learning for market basket analysis and similar tasks.
- so,Apriori is about people who bought something also buy something else,or who did something also did something else.
- Apriori algorithm has three parts :support , confidence, lift
- Let's take an example of data of people who like movies.
- Support : it is taking the number users watching movie m divided by the total number users from the data
- Confidence: it is number of people Who have seen movie M1 ,M2 both divided by number of people watching movie M1.

Movie Recommendation:
$$\text{support}(\mathbf{M}) = \frac{\# \text{ user watchlists containing } \mathbf{M}}{\# \text{ user watchlists}}$$

Movie Recommendation:
$$\text{confidence}(\mathbf{M}_1 \rightarrow \mathbf{M}_2) = \frac{\# \text{ user watchlists containing } \mathbf{M}_1 \text{ and } \mathbf{M}_2}{\# \text{ user watchlists containing } \mathbf{M}_1}$$

- Lift : It is confidence divided by support, It is basically the likelihood of the person watching the movie if you suggest.

Movie Recommendation:

$$\text{lift}(\mathbf{M_1} \rightarrow \mathbf{M_2}) = \frac{\text{confidence}(\mathbf{M_1} \rightarrow \mathbf{M_2})}{\text{support}(\mathbf{M_2})}$$

Implementing on colab by using python:

- First step will be to download apyori package on colab to use it. By using !pip install apyori.
- Second step is data preprocessing.

```
1 dataset = pd.read_csv('Market_Basket_Optimization.csv', header = None)
2 transactions = []
3 for i in range(0, 7501):
4     transactions.append([str(dataset.values[i,j]) for j in range(0, 20)])
```

- Training the Apriori model.

```
1 from apyori import apriori
2 rules = apriori(transactions = transactions, min_support = 0.003, min_confidence = 0.2, min_lift = 3, min_length = 2, rs
```

- Visualising the result.
For visualising we will put the data into panda data frames

```
1 def inspect(results):
2     lhs = [tuple(result[2][0][0])[0] for result in results]
3     rhs = [tuple(result[2][0][1])[0] for result in results]
4     supports = [result[1] for result in results]
5     confidences = [result[2][0][2] for result in results]
6     lifts = [result[2][0][3] for result in results]
7     return list(zip(lhs, rhs, supports, confidences, lifts))
8 resultsinDataFrame = pd.DataFrame(inspect(results), columns = ['Left Hand Side', 'Right H
```

```
1 resultsinDataFrame.nlargest(n = 10, columns = 'Lift')
```

1. Displaying in unsorted way
2. Displaying it in ordered way.

2. Eclat:

- Eclat stands for Equivalence Class transformation.
- It is another association rule learning algorithm.
- For this algorithm we generally only calculate support unlike apriori.
- Support : while calculating support it uses the set of the items instead of considering only one element.

Movie Recommendation:
$$\text{support}(M) = \frac{\# \text{ user watchlists containing } M}{\# \text{ user watchlists}}$$

- Eclat algorithm:

Step 1: Set a minimum support



Step 2: Take all the subsets in transactions having higher support than minimum support



Step 3: Sort these subsets by decreasing support

- Implementation of Eclat is same as that of apriori but for eclat we will only consider support

6. Reinforcement learning

- Reinforcement learning is a powerful branch in machine learning .
- It is used to solve interacting problems where data is observed up to time t is considered to decide which action to take at time $t+1$!
- In this course we will see 2 reinforcement learning models:
 1. Upper confidence bound
 2. Thompson sampling

Multi-Armed Bandit problem

- Multi armed bandit problem is a classic problem in ml and reinforcement learning that model the trade-off between exploration and exploitation.
- Goals is to figure out which out of the machine has best distribution.
- You need to explore the machines in order to figure out which machine is best in order to avoid exploitation.
- In the algorithm :
 1. You have k arms
 2. Each arm i gives a reward drawn from a probability distribution R_i .
 3. At each step t , you choose an arm and receive some reward.
 4. The objective is to maximise the expected total reward over time:

1. Upper Confidence Bound

- Upper confidence bound algorithm is a powerful approach used in multi armed bandit problem in machine learning particularly in reinforcement learning,
- Strategy used:
 1. Average reward of each arm (exploitation)
 2. Uncertainty or variance (exploration)
- Algorithm:

Step 1. At each round n , we consider two numbers for each ad i :

- $N_i(n)$ - the number of times the ad i was selected up to round n ,
- $R_i(n)$ - the sum of rewards of the ad i up to round n .

Step 2. From these two numbers we compute:

- the average reward of ad i up to round n

$$\bar{r}_i(n) = \frac{R_i(n)}{N_i(n)}$$

- the confidence interval $[\bar{r}_i(n) - \Delta_i(n), \bar{r}_i(n) + \Delta_i(n)]$ at round n with

$$\Delta_i(n) = \sqrt{\frac{3 \log(n)}{2 N_i(n)}}$$

Step 3. We select the ad i that has the maximum UCB $\bar{r}_i(n) + \Delta_i(n)$.

Implementation of the algorithm:

- First step is to import the libraries to be used
 - Second step is to import the dataset.
 - Third step will be to implement the ucb algorithm step by step.
- ```
1 import math
2 N = 10000
3 d = 10
4 ads_selected = []
5 numbers_of_selections = {0} * d
6 sums_of_rewards = {0} * d
7 total_reward = 0
8 for n in range(0, N):
9 ad = 0
10 max_upper_bound = 0
11 for i in range(0, d):
12 if (numbers_of_selections[i] > 0):
13 average_reward = sums_of_rewards[i] / numbers_of_selections[i]
14 delta_i = math.sqrt(3/2 * math.log(n + 1) / numbers_of_selections[i])
15 upper_bound = average_reward + delta_i
16 else:
17 upper_bound = 1e400
18 if (upper_bound > max_upper_bound):
19 max_upper_bound = upper_bound
20 ad = i
21 ads_selected.append(ad)
22 numbers_of_selections[ad] = numbers_of_selections[ad] + 1
23 reward = dataset.values[n, ad]
24 sums_of_rewards[ad] = sums_of_rewards[ad] + average_reward
25 total_reward = total_reward + reward
```
- Last step is to visualise the graph to see the ad selected for maximum number of times.

```
1 plt.hist(ads_selected)
2 plt.title('Histogram of ads selections')
3 plt.xlabel('Ads')
4 plt.ylabel('Number of times each ad was selected')
5 plt.show()
```

## 2. Thompson sampling Algorithm

- Thompson sampling is an reinforcement learning algorithm generally used for solving exploitation-exploration trade-off .It is an probabilistic model.
- This algorithm tries balancing exploration and exploitation by sampling actions according to their probability of being optimal.
- Goal is to maximise the total reward.
- Algorithm used:

**Step 1.** At each round  $n$ , we consider two numbers for each ad  $i$ :

- $N_i^1(n)$  - the number of times the ad  $i$  got reward 1 up to round  $n$ ,
- $N_i^0(n)$  - the number of times the ad  $i$  got reward 0 up to round  $n$ .

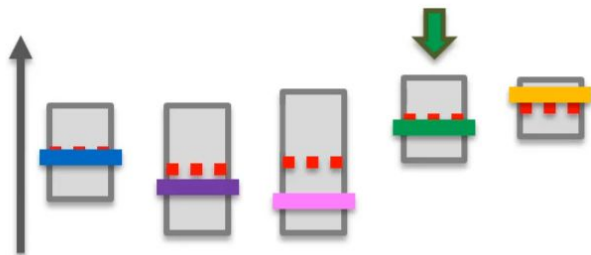
**Step 2.** For each ad  $i$ , we take a random draw from the distribution below:

$$\theta_i(n) = \beta(N_i^1(n) + 1, N_i^0(n) + 1)$$

**Step 3.** We select the ad that has the highest  $\theta_i(n)$ .

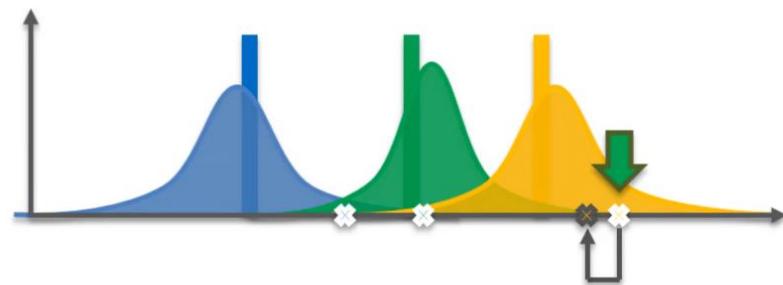
- Difference between UCB and thompson algorithm

## UCB



- Deterministic
- Requires update at every round

## Thompson Sampling



- Probabilistic
- Can accommodate delayed feedback
- Better empirical evidence

## Implementing the thompson algorithm:

- Start with importing the libraries .
- Then import the dataset.
- Then third step is to implement the algorithm.

```
[] 1 import random
 2 N = 10000
 3 d = 10
 4 ads_selected = []
 5 numbers_of_rewards_1 = [0] * d
 6 numbers_of_rewards_0 = [0] * d
 7 total_reward = 0
 8 for n in range(0, N):
 9 ad = 0
 10 max_random = 0
 11 for i in range(0, d):
 12 random_beta = random.betavariate(numbers_of_rewards_1[i] + 1, numbers_of_rewards_0[i] + 1)
 13 if random_beta > max_random:
 14 max_random = random_beta
 15 ad = i
 16 ads_selected.append(ad)
 17 reward = dataset.values[n, ad]
 18 if reward == 1:
 19 numbers_of_rewards_1[ad] = numbers_of_rewards_1[ad] + 1
 20 else:
 21 numbers_of_rewards_0[ad] = numbers_of_rewards_0[ad] + 1
 22 total_reward = total_reward + reward
```

Visualising the results - Histogram

- Visualising the histogram to see the ad viewed for the most number of times.

# NATURAL LANGUAGE PROCESSING

- Natural language processing is a subfield of artificial intelligence that focuses on enabling computers to understand, interpret and generate human language.
- Focuses on extracting meaning and information from text and speech such as entities, and relationships and sentiments.
- Applications:
  1. Chatbots and virtual assistant.
  2. Search engines
  3. Machine translations
  4. Sentiments analysis.

# Bag of words:

- Bag of words model represent text as a collection(bag\_ of its words,disregarding grammar and word order but keeping multiplicity.
- Example:

Suppose you have two sentences:

1. "I love NLP"
2. "I love machine learning"

## Step 1: Build Vocabulary

Create a list of unique words from all documents:

CSS

Copy

Edit

```
["I", "love", "NLP", "machine", "learning"]
```

## Step 2: Vectorize Each Sentence

Count the number of times each word appears in a sentence.

- Sentence 1 → [1, 1, 1, 0, 0]
- Sentence 2 → [1, 1, 0, 1, 1]

Each sentence is now a **vector of word counts** based on the vocabulary.

- Cons: Ingones word order and context,vocabulary size can get large, Sparse representation.
- Pros : Simple and easy to implement.

# Implementation of python for sentiment analysis.

- First step is importing all the necessary libraries.
- Second is to import the dataset file.
- Third step is data cleaning that is a very important step

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

```
1 import re
2 import nltk
3 nltk.download('stopwords')
4 from nltk.corpus import stopwords
5 from nltk.stem.porter import PorterStemmer
6 corpus = []
7 for i in range(0, 1000):
8 review = re.sub('[^a-zA-Z]', ' ', dataset['Review'][i])
9 review = review.lower()
10 review = review.split()
11 ps = PorterStemmer()
12 review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]
13 review = ' '.join(review)
14 corpus.append(review)
```

[nltk\_data] Downloading package stopwords to /root/nltk\_data...  
[nltk\_data] Unzipping corpora/stopwords.zip.

- Fourth step is to construct bag of words model
- Next is to use classification variable for classifying.

```
5] 1 from sklearn.feature_extraction.text import CountVectorizer
2 cv = CountVectorizer()
3 x = cv.fit_transform(corpus).toarray()
4 y = dataset.iloc[:, -1].values
```

```
1 len(X[0])
```

# DEEP LEARNING:

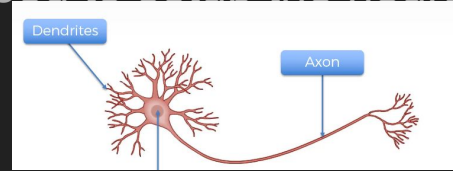
- The concept of deep learning and neural network was there for quite a long time but they started making impact just now.
- This happened because back then people knew the concept but we didn't had right standards to facilitates the neural networks.
- In order to let neural network and deep learning work properly we need to two things: a lot of data and processing power.
- **Deep learning is a type of artificial intelligence that teaches computers to learn and make decisions like a human brain using ayers of neurons**



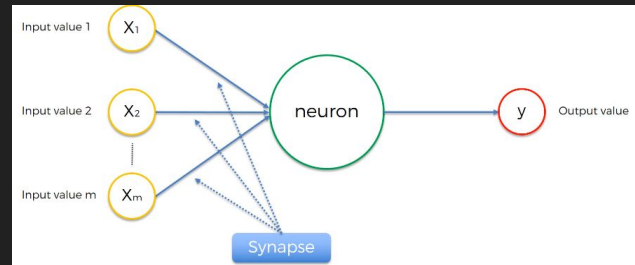
# Artificial neural network:

## Neurons:

- Since the topic of deep learning is related to the concept of mimicking human brain so for setting up this system we need to setup the neural system as same as that of human body.
- This is how a human neuron looks like:

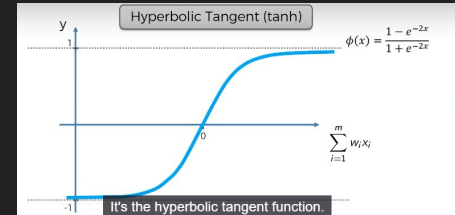
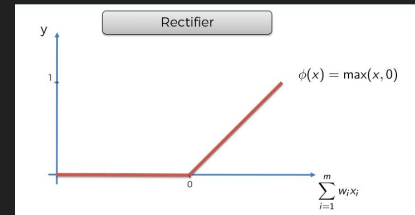
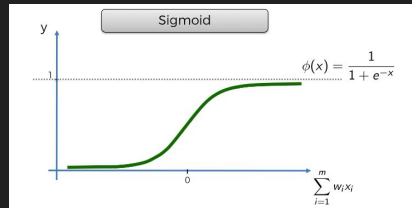
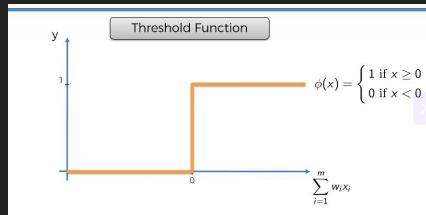


- as you see the parts of neuron it has: dendrites, axon.
- So dendrites of the neuron works as the receiver of the signals and axon works as transmitter that transmits the signals.
- This connection between that neurons forms between each other to form the neural system is called as synapsis.
- Neurons in ai is known as node .
- Inputs are independent variable.
- Output is dependant variable.



# Activation Function:

- Activation function is a decision maker for each neuron in a neural network .It decides whether the neuron should be activated or not ,based on the input it receives.
- Types of activation function:



- If you dependent variable is binary then we generally us threshold function and sigmoid function.

## Gradient Descent:

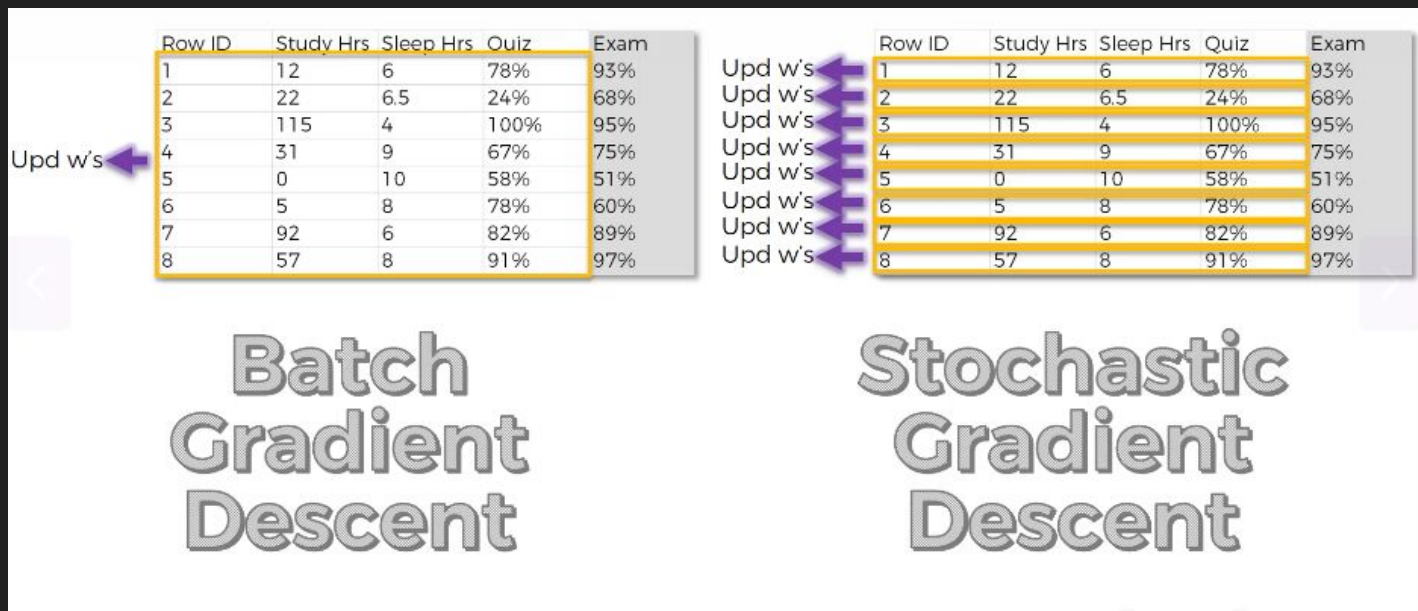
- Gradient descent is a method for solving the optimization problem for minimising the cost function value.
- It's an efficient methods as it works well even with millions of parameters.
- This method is like a trial-and-error method

## Brute Force:

- This method is used by checking all the coordinates on a map to find the lowest points.
- This method generally do not have any prior knowledge of the functions shape or slope.
- This method is like trying every option blindly.

# Stochastic Gradient Descent:

- Stochastic gradient Descent is a variation of gradient descend .
- In stochastic gradient descent,you update the model using only one data point at a time.
- It is specially used for large datasets.



# Training the ANN with Stochastic Gradient Descent

**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.

# Step by step implementation of ANN:

- First step is to import all the libraries and in this implementation of ann we will use tensorflow library .
- Second step is data preprocessing that is importing the dataset,encoding the categorical data,then splitting the dataset and feature scaling.
- Third step to build the ann model.that includes building the ann model instance and then constructing first layer the second hidden layer and output layer by using keras of tensorflow library.
- Training the model on the dataset.
- Then predict the output .
- Last step is to construct the confusion matrix and calculating the accuracy.

```
1 import numpy as np
2 import pandas as pd
3 import tensorflow as tf
```

```
[] 1 ann = tf.keras.models.Sequential()
Adding the input layer and the first hidden layer
[] 1 ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
Adding the second hidden layer
[] 1 ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
Adding the output layer
[] 1 ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

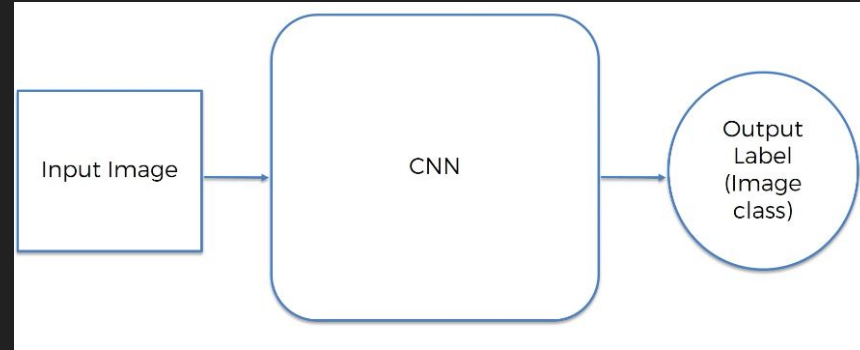
```
[] 1 ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Training the ANN on the Training set

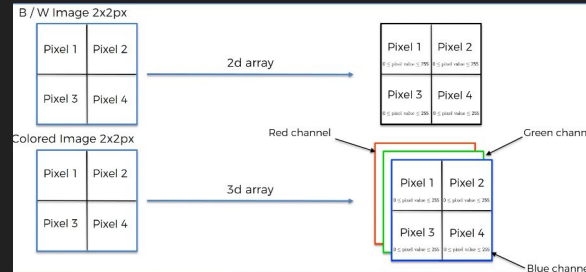
```
1 ann.fit(X_train, y_train, batch_size = 32, epochs = 100)
```

# Convolutional neural network:

- Convolutional neural network is a neural network that works by identifying and understanding images.
- Convolutional neural network is used for classifying images.



- This method treats black and white as a 2-d array .
- And colored images to a 3-d array on the basis of the primary color.

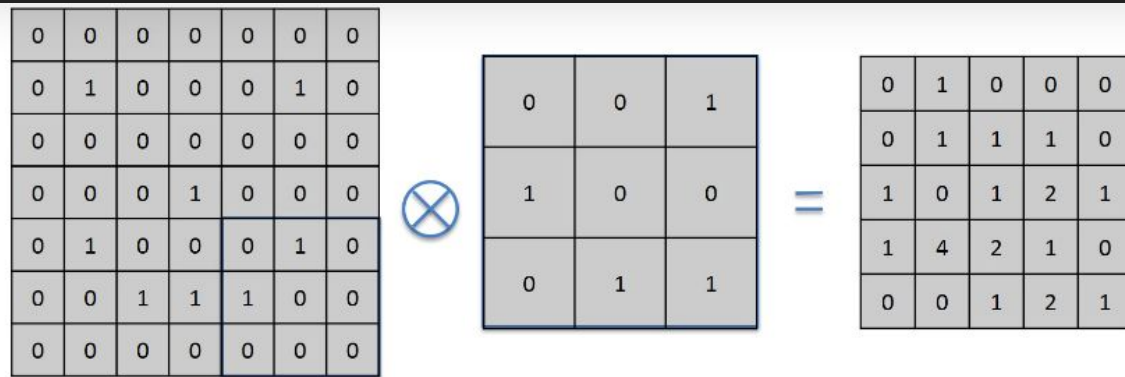


# 1. Convolutional:

- Formula:

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau$$

- Convolutional nn works by identifying the features of the images as we have some feature detector that detects the feature.
- for example : we are creating 5X5 feature map by multiplying input images and feature detectors.



Input Image

Feature  
Detector

Feature Map



## 2. Pooling:

- Pooling in convolutional neural network is a technique used to reduce the spatial dimensions of the input features maps while retaining the most important information.
- It helps make more efficient and reduce overfitting.
- Reduces the number of parameters and computations.
- Helps the network become invariant to small translations or distortions in the input.
- It helps extract dominant features that are used for classification.

### Types of pooling:

1. Max-pooling: take maximum value out of the patch of the feature.
2. Average pooling: Takes the average of the values from the patch.
3. Global pooling: it is of two types -globalmax pooling,global min pooling.

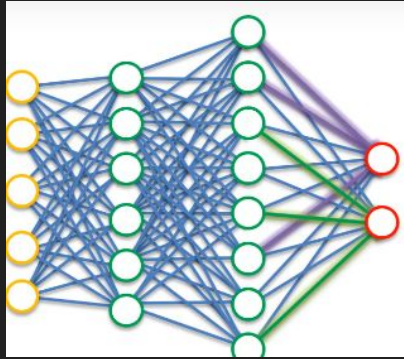
### 3. Flattening:

- After applying pooling we can apply flattening the that is going to convert the pooled feature map matrix into an array.



## 4. Full connection:

- After flattening that column of element is passed as an input layer to the neural network that are connected to the hidden layers.
- The full connection adds up the hidden layers to the input layers that we get from flattening step.



# Implementation of cnn:

- First step is to import tensor and keras library ,and also important to import imagedatagenerator class from keras library.
- Data preprocessing both dataset that is training and testing dataset.

```
Preprocessing the Training set

[] 1 train_datagen = ImageDataGenerator(
2 rescale=1./255,
3 shear_range=0.2,
4 zoom_range=0.2,
5 horizontal_flip=True)
6 training_set = train_datagen.flow_from_directory(
7 'dataset/training_set',
8 target_size=(64, 64),
9 batch_size=32,
10 class_mode='binary')

Preprocessing the Test set

[] 1 test_datagen = ImageDataGenerator(rescale=1./255)
2 test_set = test_datagen.flow_from_directory(
3 'dataset/test_set',
4 target_size=(64, 64),
5 batch_size=32,
6 class_mode='binary')
```

- Creating instance of the CNN model then passing the data through all the steps for model building.

```
Initialising the CNN

[] 1 cnn = tf.keras.models.Sequential()

Step 1 - Convolution

[] 1 cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu', input_shape=(64, 64, 3)))

Step 2 - Pooling

[] 1 cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

Adding a second convolutional layer

[] 1 cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
2 cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

Step 3 - Flattening

[] 1 cnn.add(tf.keras.layers.Flatten())
```

```
Step 3 - Flattening

[] 1 cnn.add(tf.keras.layers.Flatten())

Step 4 - Full Connection

[] 1 cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))

Step 5 - Output Layer

[] 1 cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Part 2 - Training the CNN

- Training the CNN model by compiling the cnn and training the cnn model on training set and evaluating it on test set.

▼ Compiling the CNN

```
[] 1 cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

▼ Training the CNN on the Training set and evaluating it on the Test set

```
▶ 1 cnn.fit(x = training_set, validation_data = test_set, epochs = 25)
```

Part 4 - Making a single prediction

- Making the single prediction.

```
▶ 1 import numpy as np
2 from keras.preprocessing import image
3 test_image = image.load_img('dataset/single_prediction/cat_or_dog_1.jpg', target_size = (64, 64))
4 test_image = image.img_to_array(test_image)
5 test_image = np.expand_dims(test_image, axis = 0)
6 result = cnn.predict(test_image)
7 training_set.
```

# Dimensionality Reduction:

- Dimensionality Reduction is a crucial technique in machine learning and data preprocessing.
- It involves reducing the number of input variables(independent variable) in a dataset while preserving the relevant information.
- There are two types of dimensionality reduction techniques:
  1. Feature Selection.
  2. Feature extraction
- Feature selection techniques are backward elimination,forward selection , bidirectional elimination,score comparison,etc.
- Feature extractions techniques are Principal component analysis(PCA),Kiear discriminant analysis(LDA),and kernel PCA.
- We use dimensionality reduction technique is to solve curse of dimensionality problem.

# 1. PCA

- PCA is a linear dimensionality reduction technique is feature extraction techniques of dimensionality reduction method.
- It helps reduce the number of variables in your dataset while preserving as much information
- We use PCA when:
  1. Curse of dimensionality
  2. Features are redundant
  3. You want to visualize high-dimensional data.'
  4. To speed up machine learning algorithms .
- Algorithm:

- Standardize the data.
- Obtain the Eigenvectors and Eigenvalues from the covariance matrix or correlation matrix, or perform Singular Vector Decomposition.
- Sort eigenvalues in descending order and choose the  $k$  eigenvectors that correspond to the  $k$  largest eigenvalues where  $k$  is the number of dimensions of the new feature subspace ( $k \leq d$ ).
- Construct the projection matrix  $\mathbf{W}$  from the selected  $k$  eigenvectors.
- Transform the original dataset  $\mathbf{X}$  via  $\mathbf{W}$  to obtain a  $k$ -dimensional feature subspace  $\mathbf{Y}$

## Steps to implement PCA:

- First step is to import all the necessary libraries.
- Second step is to import dataset
- Then splitting the dataset and feature scaling.
- Then the next step is to apply pca on the dataset.

```
1 from sklearn.decomposition import PCA
2 pca = PCA(n_components = 2)
3 X_train = pca.fit_transform(X_train)
4 X_test = pca.transform(X_test)
```

- Then the next step is to train any model on the dataset.
- Then constructing confusion matrix and accuracy of the model
- Visualising the mode.



## 2. LDA(LINEAR DISCRIMINANT ANALYSIS)

- LDA is an supervised learning algorithm .
- Unlike PCA it works but maximizing the class separability.
- Generally used for classification task.

### LDA vs PCA:

| Feature           | PCA                                | LDA                         |
|-------------------|------------------------------------|-----------------------------|
| Type              | Unsupervised                       | Supervised                  |
| Goal              | Maximize variance                  | Maximize class separability |
| Uses Labels       | ✗                                  | ✓                           |
| Best for          | Feature extraction                 | Classification tasks        |
| Output Dimensions | $\leq$ number of original features | $\leq$ number of classes    |

## Steps to implement PCA:

- First step is to import all the necessary libraries.
- Second step is to import dataset
- Then splitting the dataset and feature scaling.
- Then the next step is to apply LDA on the dataset.

```
1 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
2 lda = LDA(n_components = 2)
3 X_train = lda.fit_transform(X_train, y_train)
4 X_test = lda.transform(X_test)
```

- Then the next step is to train any model on the dataset.
- Then constructing confusion matrix and accuracy of the model
- Visualising the mode.

### 3. KERNEL PCA

- Kernel pca is an extension of standard pca.
- This method uses the kernel trick to perform dimensionality reduction.
- This is suitable for data that is not linearly separable.



#### Steps in Kernel PCA:

1. Choose a kernel function  $K(x_i, x_j)$ .
2. Compute the **kernel (Gram) matrix**  $K$ .
3. Center the kernel matrix.
4. Perform **eigen-decomposition** on the centered matrix.
5. Select top  $k$  eigenvectors  $\rightarrow$  reduced dimension.

## Steps to implement kernel PCA:

- First step is to import all the necessary libraries.
  - Second step is to import dataset
  - Then splitting the dataset and feature scaling.
  - Then the next step is to apply kernel pca on the dataset.
- 
- Then the next step is to train any model on the dataset.
  - Then constructing confusion matrix and accuracy of the model
  - Visualising the mode.

```
1 from sklearn.decomposition import KernelPCA
2 kpca = KernelPCA(n_components = 2, kernel = 'rbf')
3 x_train = kpca.fit_transform(x_train)
4 x_test = kpca.transform(X_test)
```

# XGBOOST:

- XGBOOST stands for extreme gradient boosting .
- It is the most powerful and widely used machine learning model
- It is based on gradient boosting that is an ensemble learning technique that builds model sequentially.
- It is used for both regression and classification task.

Code for Xgboost :

```
[4] 1 from xgboost import XGBClassifier
 2 classifier = XGBClassifier()
 3 classifier.fit(X_train, y_train)
```