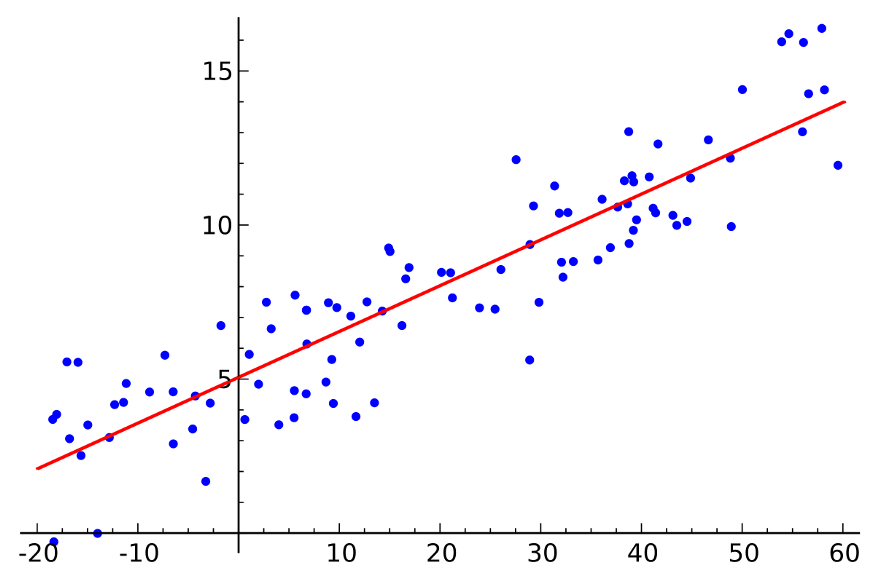
**DIFFERENT ALGORITHUMS THAT CAN BE USED**

**LINEAR REGRESSION**

Before knowing what is linear regression, let us get ourselves accustomed to regression. Regression is a method of modelling a target value based on independent predictors. This method is mostly used for forecasting and finding out cause and effect relationship between variables. Regression techniques mostly differ based on the number of independent variables and the type of relationship between the independent and dependent variables.



Linear Regression

Simple linear regression is a type of regression analysis where the number of independent variables is one and there is a linear relationship between the independent(x) and dependent(y) variable. The red line in the above graph is referred to as the best fit straight line. Based on the given data points, we try to plot a line that models the points the best. The line can be modelled based on the linear equation shown below.

y = a\_0 + a\_1 \* x Linear Equation

The motive of the linear regression algorithm is to find the best values for a\_0 and a\_1. Before moving on to the algorithm, let’s have a look at two important concepts you must know to better understand linear regression.

**CODE FOR IMPLEMENTING LINEAR REGRESSION**

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score

clf = LinearRegression(normalize=True)

clf.fit(x\_train,y\_train)

y\_pred = clf.predict(x\_test)

print(r2\_score(y\_test,y\_pred))

**NAIVES BAYES**

A classifier is a machine learning model segregating different objects on the basis of certain features of variables. It is a kind of classifier that works on the Bayes theorem. Prediction of membership probabilities is made for every class such as the probability of data points associated with a particular class.

The class having maximum probability is appraised as the most suitable class. This is also referred to as Maximum A Posteriori (MAP).

* The MAP for a hypothesis is:
  + 𝑀𝐴𝑃 (𝐻) = max 𝑃((𝐻|𝐸))
  + 𝑀𝐴𝑃 (𝐻) = max 𝑃((𝐻|𝐸)  ∗ (𝑃(𝐻)) /𝑃(𝐸))
  + 𝑀𝐴𝑃 (𝐻) = max(𝑃(𝐸|𝐻) ∗ 𝑃(𝐻))
  + 𝑃 (𝐸) is evidence probability, and it is used to normalize the result. The result will not be affected by removing (𝐸).

**CODE FOR IMPLEMENTING NAVES BAYES (GUASSIAN NAVES BAYES )**

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

X, y = load\_iris(return\_X\_y=True)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5, random\_state=0)

gnb = GaussianNB()

y\_pred = gnb.fit(X\_train, y\_train).predict(X\_test)

**KNN**

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another.The k value in the k-NN algorithm defines how many neighbors will be checked to determine the classification of a specific query point. For example, if k=1, the instance will be assigned to the same class as its single nearest neighbor. Defining k can be a balancing act as different values can lead to overfitting or underfitting. Lower values of k can have high variance, but low bias, and larger values of k may lead to high bias and lower variance. The choice of k will largely depend on the input data as data with more outliers or noise will likely perform better with higher values of k. Overall, it is recommended to have an odd number for k to avoid ties in classification, and cross-validation tactics can help you choose the optimal k for your dataset.

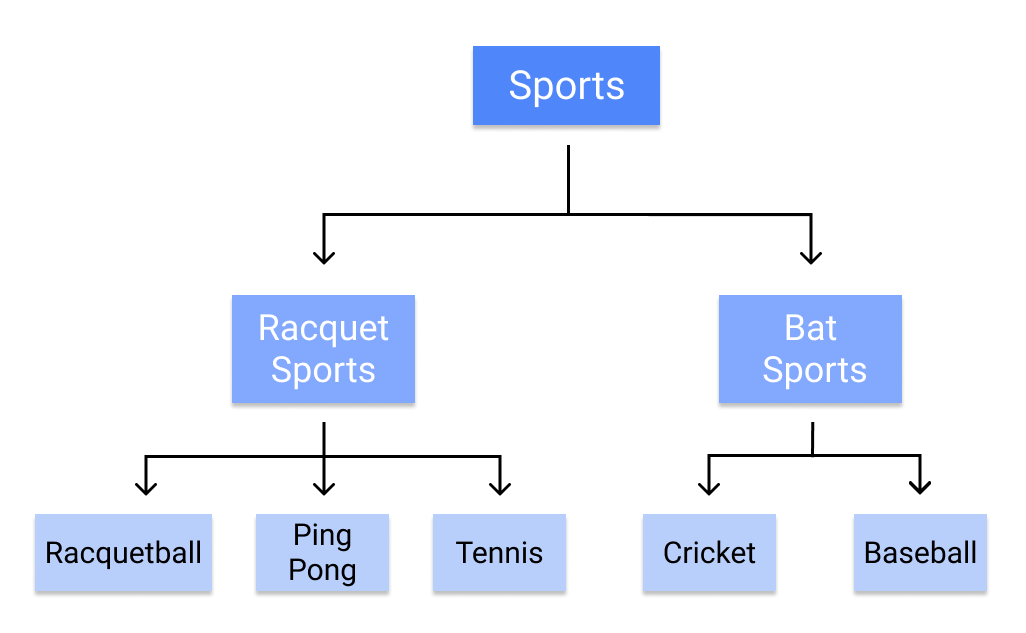
**CODE FOR IMPLEMENTING KNN**

from sklearn.neighbors import KNeighborsClassifier  
model\_name = ‘K-Nearest Neighbor Classifier’  
knnClassifier = KNeighborsClassifier(n\_neighbors = 5, metric = ‘minkowski’, p=2)  
knn\_model = Pipeline(steps=[(‘preprocessor’, preprocessorForFeatures), (‘classifier’ , knnClassifier)])  
knn\_model.fit(X\_train, y\_train)  
y\_pred = knn\_model.predict(X\_test)

**DECISION TREE**

A decision tree is a supervised learning algorithm that is perfect for classification problems, as it’s able to order classes on a precise level. It works like a flow chart, separating data points into two similar categories at a time from the “tree trunk” to “branches,” to “leaves,” where the categories become more finitely similar. This creates categories within categories, allowing for organic classification with limited human supervision.

To continue with the sports example, this is how the decision tree works:

C:\Users\ACER\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\5CF0F09B.tmp

**Random Forest**

The random forest algorithm is an expansion of decision tree, in that you first construct a multitude of decision trees with training data, then fit your new data within one of the trees as a “random forest.”

It, essentially, averages your data to connect it to the nearest tree on the data scale. Random forest models are helpful as they remedy for the decision tree’s problem of “forcing” data points within a category unnecessarily.

**CODE FOR IMPLEMENTING DECISION TREE**

from sklearn.datasets import load\_iris

from sklearn import tree

iris = load\_iris()

X, y = iris.data, iris.target

clf = tree.DecisionTreeClassifier()

clf = clf.fit(X, y)

**SVM**

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

**Example:** SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat.

**CODE FOR SVM**

#Import svm model

from sklearn import svm

#Create a svm Classifier

clf = svm.SVC(kernel='linear')

# Linear Kernel

#Train the model using the training sets

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

#Predict the response for test dataset

y\_pred = clf.predict(X\_test)

**ALGORITHUM THAT WE WILL USE**

We will use K-Nearest Neighbors algorithm because various researches prove it is one of the best algorithms to give good performance and till time along with optimized models organizations uses this algorithm in recommendation systems as support.

**K-Nearest Neighbour ~** KNN is a machine learning algorithm used for regression, and classification. It is also known as the lazy learner algorithm. It simply uses a distance-based method o find the K number of similar neighbours to new data and the class in which the majority of neighbours lies, it results in that class as an output. Now let us get our system ready for project implementation.