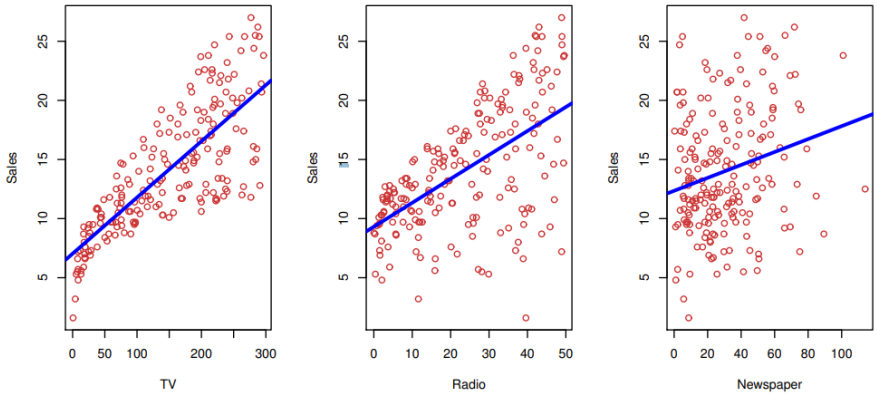
Statistical learning

Statistical learning refers to a vast set of tools for understanding data. These tools can be classified as supervised or unsupervised. Broadly speaking, supervised statistical learning involves building a statistical model for predicting, or estimating, an output based on one or more inputs. Problems of this nature occur in fields as diverse as business, medicine, astrophysics, and public policy. With unsupervised statistical learning, there are inputs but no supervising output; nevertheless we can learn relationships and structure from such data.

In order to motivate our study of statistical learning, we begin with a simple example. Suppose that we are statistical consultants hired by a client to investigate the association between advertising and sales of a particular product. The Advertising data set consists of the sales of that product in 200 different markets, along with advertising budgets for the product in each of those markets for three different media: TV, radio, and newspaper. The data are displayed in Figure. It is not possible for our client to directly increase sales of the product. On the other hand, they can control the advertising expenditure in each of the three media. Therefore, if we determine that there is an association between advertising and sales, then we can instruct our client to adjust advertising budgets, thereby indirectly increasing sales. In other words, our goal is to develop an accurate model that can be used to predict sales on the basis of the three media budgets.



In this setting, the advertising budgets are *input variables* while sales is an *output variable*. The input variables are typically denoted using the symbol X, with a subscript to distinguish them. So X1 might be the TV budget, X2 the radio budget, and X3 the newspaper budget. The inputs go by different names, such as predictors, independent variables, features or sometimes just variables. The output variable—in this case, sales—is often called the response or dependent variable, and is typically denoted using the symbol Y. Throughout this book, we will use all of these terms interchangeably.

More generally, suppose that we observe a quantitative response Y and p different predictors, X1, X2,...,Xp. We assume that there is some relationship between Y and X = (X1, X2,...,Xp), which can be written in the very general form

$$Y = f(X) + \epsilon.$$

Here f is some fixed but unknown function of X1,...,Xp, and ϵ is a random error term, which is independent of X and has mean zero. In this formulation, f represents the systematic information that X provides about Y.

Why Estimate f?

There are two main reasons that we may wish to estimate f: prediction and inference. We discuss each in turn.

Prediction

In many situations, a set of inputs X are readily available, but the output Y cannot be easily obtained. In this setting, since the error term averages to zero, we can predict Y using

$$\hat{Y} = \hat{f}\,(X)$$

Where $\hat{f}$ represents our estimate for f, and $\hat{Y}$ represents the resulting prediction for Y. In this setting, ˆf is often treated as a black box, in the sense that one is not typically concerned with the exact form of $\hat{f}$, provided that it yields accurate predictions for Y . As an example, suppose that X1,...,Xp are characteristics of a patient's blood sample that can be easily measured in a lab, and Y is a variable encoding the patient's risk for a severe adverse reaction to a particular drug. It is natural to seek to predict Y using X, since we can then avoid giving the drug in question to patients who are at high risk of an adverse reaction—that is, patients for whom the estimate of Y is high.

The accuracy of $\hat{Y}$ as a prediction for Y depends on two quantities, which we will call the reducible error and the irreducible error. In general will not be a perfect estimate for f, and this inaccuracy will introduce some error. This error is reducible because we can potentially improve the accuracy of $\hat{f}$ by using the most appropriate statistical learning technique to estimate f. However, even if it were possible to form a perfect estimate for f, so that our estimated response took the form $\hat{Y}$ = f(X), our prediction would still have some error in it! This is because Y is also a function of ϵ, which, by definition, cannot be predicted using X. Therefore, variability associated with ϵ also affects the accuracy of our predictions. This is known as the irreducible error, because no matter how well we estimate f, we cannot reduce the error introduced by ϵ.

$$\begin{aligned} \mathrm{E}(Y-\hat{Y})^2 &= \mathrm{E}[f(X)+\epsilon-\hat{f}(X)]^2 \\ &= \underbrace{[f(X)-\hat{f}(X)]^2}_{\text{Reducible}} + \underbrace{\mathrm{Var}(\epsilon)}_{\text{Irreducible}}, \end{aligned}$$

where $\mathrm{E}(Y-\hat{Y})^2$ represents the average, or expected value, of the squared difference between the predicted and actual value of Y , and Var(ϵ) represents the variance associated with the error term ϵ.

Inference

We are often interested in understanding the association between Y and X1,...,Xp. In this situation we wish to estimate f, but our goal is not necessarily to make predictions for Y. Now $\hat{f}$ cannot be treated as a black box, because we need to know its exact form. In this setting, one may be interested in answering the following questions:

- Which predictors are associated with the response? It is often the case that only a small fraction of the available predictors are substantially associated with Y. Identifying the few important predictors among a large set of possible variables can be extremely useful, depending on the application.
- What is the relationship between the response and each predictor? Some predictors may have a positive relationship with Y, in the sense that larger values of the

predictor are associated with larger values of Y. Other predictors may have the opposite relationship. Depending on the complexity of f, the relationship between the response and a given predictor may also depend on the values of the other predictors.

- Can the relationship between Y and each predictor be adequately summarized using a linear equation, or is the relationship more complicated? Historically, most methods for estimating f have taken a linear form. In some situations, such an assumption is reasonable or even desirable. But often the true relationship is more complicated, in which case a linear model may not provide an accurate representation of the relationship between the input and output variables.

How Do We Estimate f?

Our goal is to apply a statistical learning method to the training data in order to estimate the unknown function f. In other words, we want to find a function $\hat{f}$ such that $Y \approx \hat{f}(X)$ for any observation (X, Y). Broadly speaking, most statistical learning methods for this task can be characterized as either *parametric* or *non-parametric*.

Parametric Methods

Parametric methods involve a two-step model-based approach.

1. First, we make an assumption about the functional form, or shape, of f. For example, one very simple assumption is that f is linear in X:

   $$f(X) = \beta0 + \beta1X1 + \beta2X2 + \cdots + \beta pXp$$

   This is a *linear model*, which will be discussed extensively in Chapter 3. Once we have assumed that *f* is linear, the problem of estimating *f* is greatly simplified. Instead of having to estimate an entirely arbitrary p-dimensional function *f*(X), one only needs to estimate the p + 1 coefficients β0, β1,..., βp.
2. After a model has been selected, we need a procedure that uses the training data to fit or train the model. In the case of the linear model, we need to estimate the parameters β0, β1,..., βp. That is, we want to find values of these parameters such that

   $$Y \approx \beta0 + \beta1X1 + \beta2X2 + \cdots + \beta pXp.$$

   The most common approach to fitting the model is referred to as (ordinary) least squares, However, least squares is one of many possible ways to fit the linear model.

Non-Parametric Methods

Non-parametric methods do not make explicit assumptions about the functional form of f. Instead they seek an estimate of f that gets as close to the data points as possible without being too rough or wiggly.

Such approaches can have a major advantage over parametric approaches: by avoiding the assumption of a particular functional form for f, they have the potential to accurately fit a wider range of possible shapes for f.

Any parametric approach brings with it the possibility that the functional form used to estimate f is very different from the true f, in which case the resulting model will not fit the data well. In contrast, non-parametric approaches completely avoid this danger, since essentially no assumption about the form of f is made. But non-parametric approaches do suffer from a major disadvantage: since they do not reduce the problem of estimating f to a small number of parameters, a very large number of observations (far more than is typically needed for a parametric approach) is required in order to obtain an accurate estimate for f.

Assessing Model Accuracy:

A wide range of statistical learning methods that extend far beyond the standard linear regression approach. *Why is it necessary to introduce so many different statistical learning approaches, rather than just a single best method?* There is no free lunch in statistics: no one method dominates all others over all possible data sets. On a particular data set, one specific method may work best, but some other method may work better on a similar but different data set. Hence it is an important task to decide for any given set of data which method produces the best results. Selecting the best approach can be one of the most challenging parts of performing statistical learning in practice.

Measuring the Quality of Fit

In order to evaluate the performance of a statistical learning method on a given data set, we need some way to measure how well its predictions actually match the observed data. That is, we need to quantify the extent to which the predicted response value for a given observation is close to the true response value for that observation. In the regression setting, the most commonly-used measure is the *mean squared error (MSE)*, given by

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2,$$

Where $\hat{f}(x_i)$ is the prediction that $\hat{f}$ gives for the *i*th observation. The *MSE* will be small if the predicted responses are very close to the true responses, and will be large if for some of the observations, the predicted and true responses differ substantially.

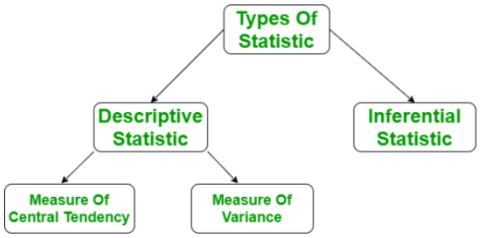
The MSE is computed using the training data that was used to fit the model, and so should more accurately be referred to as the training MSE. But in general, we do not really care how well the method works on the training data. Rather, we are interested in the accuracy of the predictions that we obtain when we apply our method to previously unseen test data.

Descriptive Statistics

In Descriptive statistics, we are describing our data with the help of various representative methods like by using charts, graphs, tables, excel files etc. In descriptive statistics, we describe our data in some manner and present it in a meaningful way so that it can be easily understood. Most of the times it is performed on small data sets and this analysis helps us a lot to predict some future trends based on the current findings. Some measures that are used to describe a data set are measures of central tendency and measures of variability or dispersion.

Types of Descriptive statistic:

1. Measure of central tendency
2. Measure of variability



1. Measure of central tendency:

It represents the whole set of data by single value.It gives us the location of central points. There are three main measures of central tendency:

a) Mean
b) Mode
c) Median

a) Mean:

It is the sum of observation divided by the total number of observations. It is also defined as average which is the sum divided by count.

$$\text{Mean } (\bar{x}) = \frac{\sum x}{n}$$

where, n = number of terms

Program:

```
import numpy as np
arr = [5, 6, 11]
mean = np.mean(arr)
print("Mean = ", mean)
```

Output :

```
Mean =  7.333333333333333
```

b) Mode:

It is the value that has the highest frequency in the given data set. The data set may have no mode if the frequency of all data points is the same. Also, we can have more than one mode if we encounter two or more data points having the same frequency.

Program:

```
from scipy import stats
arr =[1, 2, 2, 3]
mode = stats.mode(arr)
print("Mode = ", mode)
```

Output:

```
Mode =  ModeResult(mode=array([2]), count=array([2]))
```

c) Median:

It is the middle value of the data set. It splits the data into two halves. If the number of elements in the data set is odd then the centre element is median and if it is even then the median would be the average of two central elements.

Odd	Even
$\frac{n+1}{2}$	$\frac{n}{2}, \frac{n}{2}+1$

where, n=number of terms

Program:

```
import numpy as np
arr =[1, 2, 3, 4]
median = np.median(arr)
print("Median = ", median)
```

Output:

```
Median =  2.5
```

2. Measure of variability:

Measure of variability is known as the spread of data or how well is our data is distributed. The most common variability measures are:

a) Range
b) Variance
c) Standard deviation

a) Range

The range describes the difference between the largest and smallest data point in our data set. The bigger the range, the more is the spread of data and vice versa.

Range = Largest data value – smallest data value

Program:

```
import numpy as np
arr = [1, 2, 3, 4, 5]
Maximum = max(arr)
Minimum = min(arr)
Range = Maximum-Minimum
print("Maximum = ",Maximum)
print("Minimum = ",Minimum)
print("Range = ", Range)
```

Output:

```
Maximum = 5
Minimum = 1
Range = 4
```

b) Variance

It is defined as an average squared deviation from the mean. It is being calculated by finding the difference between every data point and the average which is also known as the mean, squaring them, adding all of them and then dividing by the number of data points present in our data set.

$$\sigma^2 = \frac{\sum(\chi - \mu)^2}{N}$$

where N = number of terms

u = Mean

Program:

```
import statistics
arr = [1, 2, 3, 4, 5]
print("Var = ", (statistics.variance(arr)))
```

Output:

```
Var =  2.5
```

c) Standard Deviation

It is defined as the square root of the variance. It is being calculated by finding the Mean, then subtract each number from the Mean which is also known as average and square the result. Adding all the values and then divide by the no of terms followed the square root.

$$\sigma = \sqrt{\frac{\Sigma (x - u)^2}{N}}$$

where N = number of terms

u = Mean

Program:

```
import statistics
arr = [1, 2, 3, 4, 5]
print("Std = ", (statistics.stdev(arr)))
```

Output:

```
Std = 1.5811388300841898
```

Linear Regression:

- The term regression is used when you try to find the relationship between variables.
- In Machine Learning and in statistical modeling, that relationship is used to predict the outcome of events.

Assessing the Accuracy of the Coefficient Estimates

We assume that the true relationship between X and Y takes the form $Y = f(X) + \epsilon$ for some unknown function f, where ϵ is a mean-zero random error term. If f is to be approximated by a linear function, then we can write this relationship as

$$Y = \beta 0 + \beta 1X + \epsilon$$

Here $\beta 0$ is the intercept term-that is, the expected value of Y when X = 0, and $\beta 1$ is the slope—the average increase in Y associated with a one-unit increase in X. The error term is a catch-all for what we miss with this simple model: the true relationship is probably not linear, there may be other variables that cause variation in Y, and there may be measurement error. We typically assume that the error term is independent of X.

Simple Linear Regression:

Simple Linear Regression is a type of Regression algorithms that models the relationship between a dependent variable and a single independent variable. The relationship shown by a Simple Linear Regression model is linear or a sloped straight line, hence it is called Simple Linear Regression.

The key point in Simple Linear Regression is that the ***dependent variable must be a continuous / real value***. However, the independent variable can be measured on continuous or categorical values.

Simple Linear regression algorithm has mainly two objectives:

- **Model the relationship between the two variables.** Such as the relationship between Income and expenditure, experience and Salary, etc.
- **Forecasting new observations.** Such as Weather forecasting according to temperature, Revenue of a company according to the investments in a year, etc.

Mathematically, we can write this linear relationship as

$$Y \approx \beta_0 + \beta_1 X.$$

You might read "≈" as "is approximately 8eighbou as". We will sometimes describe by saying that we are regressing Y on X (or Y onto X). For example, X may represent TV advertising and Y may represent sales. Then we can regress sales onto TV by fitting the model

$$sales \approx \beta_0 + \beta_1 \times TV.$$

Implementation of Simple Linear Regression:

Here we are taking a dataset that has two variables: Sales (dependent variable) and TV (Independent variable). The goals of this problem is:

- We want to find out if there is any correlation between these two variables
- We will find the best fit line for the dataset.
- How the dependent variable is changing by changing the independent variable.

Step-1: Data Pre-processing

The first step for creating the Simple Linear Regression model is data pre-processing. We have already done it earlier in this tutorial. But there will be some changes, which are given in the below steps:

First, we will import the three important libraries, which will help us for loading the dataset, plotting the graphs, and creating the Simple Linear Regression model.

```
Import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

Next, we will load the dataset into our code:

```
data_set= pd.read_csv('Salary_Data.csv')
```

After that, we need to extract the dependent and independent variables from the given dataset. The independent variable is years of experience, and the dependent variable is salary. Below is code for it:

```
x= data_set.iloc[:, 0].values
y= data_set.iloc[:, -1].values
```

In the above lines of code, for x variable, we have taken 0 value since we want to extract first column from the dataset. For y variable, we have taken -1 value as a parameter, since we want to extract the last column.

Next, we will split both variables into the test set and training set. We have 100% observations, so we will take 70% observations for the training set and 30% observations for the test set. We are splitting our dataset so that we can train our model using a training dataset and then test the model using a test dataset. The code for this is given below:

```
# Splitting the 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= 0.3)
```

By executing the above code, we will get x-test, x-train and y-test, y-train dataset.

For simple linear Regression, we will not use Feature Scaling. Because Python libraries take care of it for some cases, so we don't need to perform it here. Now, our dataset is well prepared to work on it and we are going to start building a Simple Linear Regression model for the given problem.

Step-2: Fitting the Simple Linear Regression to the Training Set:

Now the second step is to fit our model to the training dataset. To do so, we will import the LinearRegression class of the linear_model library from the scikit learn. After importing the class, we are going to create an object of the class named as a regressor. The code for this is given below:

```
#Fitting the Simple Linear Regression model to the training dataset
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(x_train, y_train)
```

In the above code, we have used a fit() method to fit our Simple Linear Regression object to the training set. In the fit() function, we have passed the x_train and y_train, which is our training dataset for the dependent and an independent variable. We have fitted our regressor object to the training set so that the model can easily learn the correlations between the predictor and target variables. After executing the above lines of code, we will get the below output.

Step: 3. Prediction of test set result:

Dependent (Sales) and an independent variable (TV). So, now, our model is ready to predict the output for the new observations. In this step, we will provide the test dataset (new observations) to the model to check whether it can predict the correct output or not.

We will create a prediction vector y_pred, and x_pred, which will contain predictions of test dataset, and prediction of training set respectively.

```
#Prediction of Test and Training set result
y_pred= regressor.predict(x_test)
x_pred= regressor.predict(x_train)
```

On executing the above lines of code, two variables named y_pred and x_pred will generate in the variable explorer options that contain salary predictions for the training set and test set.

Multiple Linear Regression:

Multiple Linear Regression is one of the important regression algorithms which models the linear relationship between a single dependent continuous variable and more than one independent variable.

Some key points about MLR:

- For MLR, the dependent or target variable(Y) must be the continuous/real, but the predictor or independent variable may be of continuous or categorical form.
- Each feature variable must model the linear relationship with the dependent variable.
- MLR tries to fit a regression line through a multidimensional space of data-points.

In Multiple Linear Regression, the target variable(Y) is a linear combination of multiple predictor variables x1, x2, x3, ...,xn. Since it is an enhancement of Simple Linear Regression, so the same is applied for the multiple linear regression equation, the equation becomes:

$$Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4$$

Implementation of Simple Linear Regression:

Here we are taking a dataset that has four variables: Sales (dependent variable) and TV, Radio and NewsPaper (Independent variables).

Step-1: Data Pre-processing

The first step for creating the Simple Linear Regression model is data pre-processing. We have already done it earlier in this tutorial. But there will be some changes, which are given in the below steps:

First, we will import the three important libraries, which will help us for loading the dataset, plotting the graphs, and creating the Simple Linear Regression model.

```
Import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

Next, we will load the dataset into our code:

```
data_set= pd.read_csv('Salary_Data.csv')
```

After that, we need to extract the dependent and independent variables from the given dataset. The independent variable is years of experience, and the dependent variable is salary. Below is code for it:

```
x= data_set.iloc[:, 0:3].values
y= data_set.iloc[:, -1].values
```

In the above lines of code, for x variable, we have taken 0:3 range of values since we want to extract first three columns from the dataset. For y variable, we have taken -1 value as a parameter, since we want to extract the last column.

Next, we will split both variables into the test set and training set. We have 100% observations, so we will take 70% observations for the training set and 30% observations for the test set. We are splitting our dataset so that we can train our model using a training dataset and then test the model using a test dataset. The code for this is given below:

```
# Splitting the 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= 0.3)
```

By executing the above code, we will get x-test, x-train and y-test, y-train dataset.

Step-2: Fitting the Simple Linear Regression to the Training Set:

Now the second step is to fit our model to the training dataset. To do so, we will import the LinearRegression class of the linear_model library from the scikit learn. After importing the class, we are going to create an object of the class named as a regressor. The code for this is given below:

```
#Fitting the Simple Linear Regression model to the training dataset
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(x_train, y_train)
```

In the above code, we have used a fit() method to fit our Simple Linear Regression object to the training set. In the fit() function, we have passed the x_train and y_train, which is our training dataset for the dependent and an independent variable. We have fitted our regressor object to the training set so that the model can easily learn the correlations between the predictor and target variables. After executing the above lines of code, we will get the below output.

Step: 3. Prediction of test set result:

Dependent (Sales) and an independent variable (TV). So, now, our model is ready to predict the output for the new observations. In this step, we will provide the test dataset (new observations) to the model to check whether it can predict the correct output or not.

We will create a prediction vector y_pred, and x_pred, which will contain predictions of test dataset, and prediction of training set respectively.

```
#Prediction of Test and Training set result
y_pred= model.predict(x_test)
x_pred= model.predict(x_train)
```

On executing the above lines of code, two variables named y_pred and x_pred will generate in the variable explorer options that contain salary predictions for the training set and test set.

We can also check the score for training dataset and test dataset. Below is the code for it:

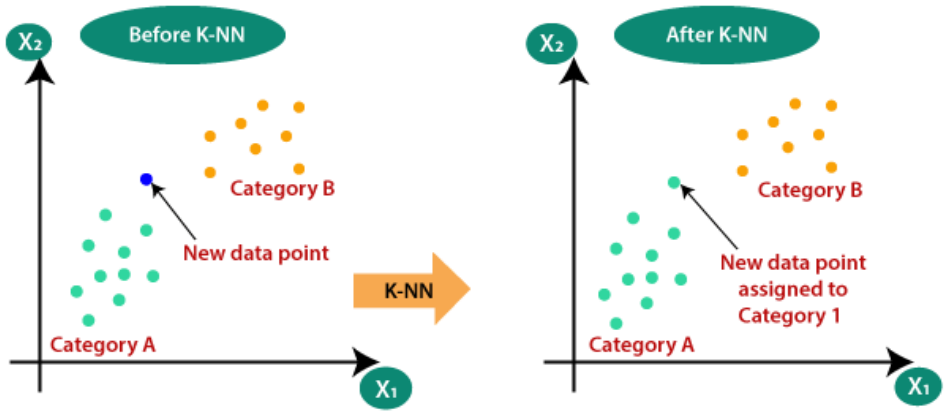
```
print('Train Score: ', model.score(x_train, y_train))
print('Test Score: ', model.score(x_test, y_test))
```

K-Nearest neighbor (KNN) Algorithm:

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
- It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

Step-1: Select the number K of the 12eighbours

Step-2: Calculate the Euclidean distance of K number of 12eighbours

Step-3: Take the K nearest 12eighbours as per the calculated Euclidean distance.

Step-4: Among these k 12eighbours, count the number of the data points in each category.

Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.

Step-6: Our model is ready.

Generating KNN Regressor Model:

```
from sklearn.neighbors import KNeighborsRegressor
scores=[]
for i in range(1,10):
  model = KNeighborsRegressor(n_neighbors=i)
  model.fit(x_train, y_train)
  scores.append(model.score(x_test,y_test))


k = scores.index(max(scores))
model = KNeighborsRegressor(n_neighbors=k)
model.fit(x_train, y_train)
print("Train Score",model.score(x_train,y_train))
print("Test Score",model.score(x_test,y_test))
```

```
Train Score 0.9556178882661865
Test Score 0.9276949091790976
```