**Mathematical explanation of K-Nearest Neighbour**

## **Introduction**

K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. The following two properties would define KNN well −

* **Lazy learning algorithm** − KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.
* **Non-parametric learning algorithm** − KNN is also a non-parametric learning algorithm because it doesn’t assume anything about the underlying data.

## **Working of KNN Algorithm**

K-nearest neighbors (KNN) algorithm uses ‘feature similarity’ to predict the values of new datapoints which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. We can understand its working with the help of following steps −

**Step 1** − For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.

**Step 2** − Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

**Step 3** − For each point in the test data do the following −

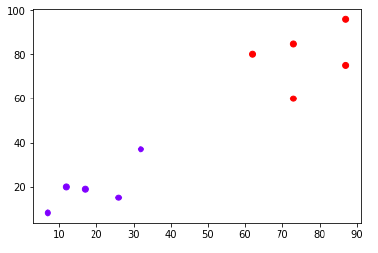
* **3.1** − Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.
* **3.2** − Now, based on the distance value, sort them in ascending order.
* **3.3** − Next, it will choose the top K rows from the sorted array.
* **3.4** − Now, it will assign a class to the test point based on most frequent class of these rows.

**Step 4** − End

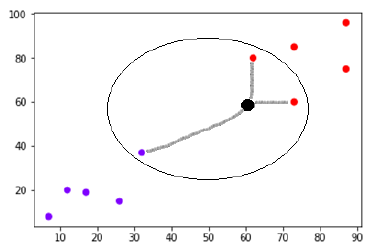
### **Example**

The following is an example to understand the concept of K and working of KNN algorithm −

Suppose we have a dataset which can be plotted as follows −



Now, we need to classify new data point with black dot (at point 60,60) into blue or red class. We are assuming K = 3 i.e. it would find three nearest data points. It is shown in the next diagram −



We can see in the above diagram the three nearest neighbors of the data point with black dot. Among those three, two of them lies in Red class hence the black dot will also be assigned in red class.

## **Implementation in Python**

As we know K-nearest neighbors (KNN) algorithm can be used for both classification as well as regression. The following are the recipes in Python to use KNN as classifier as well as regressor −

## **KNN as Classifier**

First, start with importing necessary python packages −

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

Next, download the iris dataset from its weblink as follows −

path = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

Next, we need to assign column names to the dataset as follows −

headernames = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

Now, we need to read dataset to pandas dataframe as follows −

dataset = pd.read\_csv(path, names = headernames)

dataset.head()

sepal-length sepal-width petal-length petal-width Class

0 5.1 3.5 1.4 0.2 Iris-setosa

1 4.9 3.0 1.4 0.2 Iris-setosa

2 4.7 3.2 1.3 0.2 Iris-setosa

3 4.6 3.1 1.5 0.2 Iris-setosa

4 5.0 3.6 1.4 0.2 Iris-setosa

Data Preprocessing will be done with the help of following script lines.

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

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

Next, we will divide the data into train and test split. Following code will split the dataset into 60% training data and 40% of testing data −

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.40)

Next, data scaling will be done as follows −

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(X\_train)

X\_train = scaler.transform(X\_train)

X\_test = scaler.transform(X\_test)

Next, train the model with the help of KNeighborsClassifier class of sklearn as follows −

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors = 8)

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

At last we need to make prediction. It can be done with the help of following script −

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

Next, print the results as follows −

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

result = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(result)

result1 = classification\_report(y\_test, y\_pred)

print("Classification Report:",)

print (result1)

result2 = accuracy\_score(y\_test,y\_pred)

print("Accuracy:",result2)

### **Output**

Confusion Matrix:

[[21 0 0]

[ 0 16 0]

[ 0 7 16]]

Classification Report:

precision recall f1-score support

Iris-setosa 1.00 1.00 1.00 21

Iris-versicolor 0.70 1.00 0.82 16

Iris-virginica 1.00 0.70 0.82 23

micro avg 0.88 0.88 0.88 60

macro avg 0.90 0.90 0.88 60

weighted avg 0.92 0.88 0.88 60

Accuracy: 0.8833333333333333

## **KNN as Regressor**

First, start with importing necessary Python packages −

import numpy as np

import pandas as pd

Next, download the iris dataset from its weblink as follows −

path = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

Next, we need to assign column names to the dataset as follows −

headernames = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

Now, we need to read dataset to pandas dataframe as follows −

data = pd.read\_csv(url, names = headernames)

array = data.values

X = array[:,:2]

Y = array[:,2]

data.shape

output:(150, 5)

Next, import *KNeighborsRegressor* from *sklearn* to fit the model −

from sklearn.neighbors import KNeighborsRegressor

knnr = KNeighborsRegressor(n\_neighbors = 10)

knnr.fit(X, y)

At last, we can find the MSE as follows −

print ("The MSE is:",format(np.power(y-knnr.predict(X),2).mean()))

### **Output**

The MSE is: 0.12226666666666669

## **Pros and Cons of KNN**

### **Pros**

* It is very simple algorithm to understand and interpret.
* It is very useful for nonlinear data because there is no assumption about data in this algorithm.
* It is a versatile algorithm as we can use it for classification as well as regression.
* It has relatively high accuracy but there are much better supervised learning models than KNN.

### **Cons**

* It is computationally a bit expensive algorithm because it stores all the training data.
* High memory storage required as compared to other supervised learning algorithms.
* Prediction is slow in case of big N.
* It is very sensitive to the scale of data as well as irrelevant features.

## **Applications of KNN**

The following are some of the areas in which KNN can be applied successfully −

### **Banking System**

KNN can be used in banking system to predict weather an individual is fit for loan approval? Does that individual have the characteristics similar to the defaulters one?

### **Calculating Credit Ratings**

KNN algorithms can be used to find an individual’s credit rating by comparing with the persons having similar traits.

### **Politics**

|  |  |  |  |
| --- | --- | --- | --- |
| AME | AGE | GENDER | CLASS OF SPORTS |
| Ajay | 32 | 0 | Football |
| Mark | 40 | 0 | Neither |
| Sara | 16 | 1 | Cricket |
| Zaira | 34 | 1 | Cricket |
| Sachin | 55 | 0 | Neither |
| Rahul | 40 | 0 | Cricket |
| Pooja | 20 | 1 | Neither |
| Smith | 15 | 0 | Cricket |
| Laxmi | 55 | 1 | Football |
| Michael | 15 | 0 | Football |
| Rahul | 14 | 0 | ? |

# Logistic Regression in Machine Learning

* Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
* Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**.
* Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**.
* In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
* The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
* Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.
* Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:



#### Note:**Logistic regression uses the concept of predictive modeling as regression; therefore, it is called logistic regression, but is used to classify samples; Therefore, it falls under the classification algorithm.**

## **Logistic Function (Sigmoid Function):**

* The sigmoid function is a mathematical function used to map the predicted values to probabilities.
* It maps any real value into another value within a range of 0 and 1.
* The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
* In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

## **Assumptions for Logistic Regression:**

* The dependent variable must be categorical in nature.
* The independent variable should not have multi-collinearity.

## **Logistic Regression Equation:**

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

* We know the equation of the straight line can be written as:

Logistic Regression in Machine Learning

* In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

Logistic Regression in Machine Learning

* But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

Logistic Regression in Machine Learning

The above equation is the final equation for Logistic Regression.

## **Type of Logistic Regression:**

On the basis of the categories, Logistic Regression can be classified into three types:

* **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
* **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
* **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

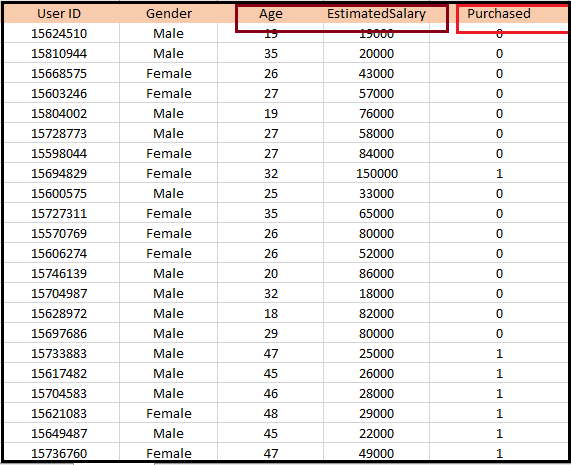
## **Python Implementation of Logistic Regression (Binomial)**

To understand the implementation of Logistic Regression in Python, we will use the below example:

Play Video

**Example:** There is a dataset given which contains the information of various users obtained from the social networking sites. There is a car making company that has recently launched a new SUV car. So the company wanted to check how many users from the dataset, wants to purchase the car.

For this problem, we will build a Machine Learning model using the Logistic regression algorithm. The dataset is shown in the below image. In this problem, we will predict the **purchased variable (Dependent Variable)** by using **age and salary (Independent variables)**.



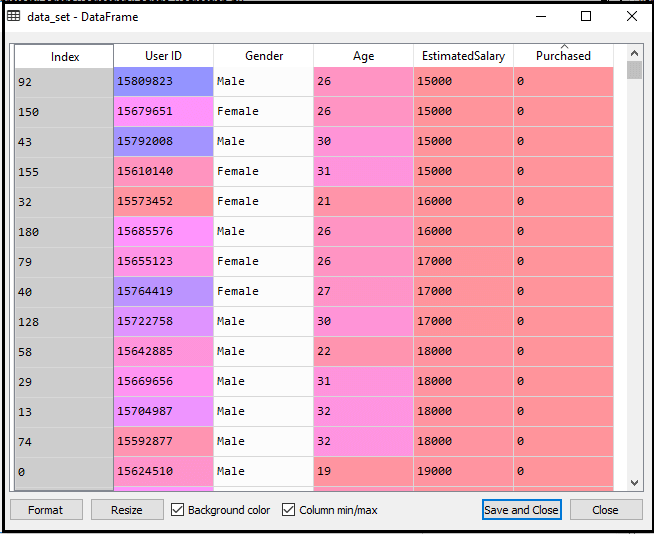
**Steps in Logistic Regression:** To implement the Logistic Regression using Python, we will use the same steps as we have done in previous topics of Regression. Below are the steps:

* Data Pre-processing step
* Fitting Logistic Regression to the Training set
* Predicting the test result
* Test accuracy of the result(Creation of Confusion matrix)
* Visualizing the test set result.

**1. Data Pre-processing step:** In this step, we will pre-process/prepare the data so that we can use it in our code efficiently. It will be the same as we have done in Data pre-processing topic. The code for this is given below:

1. #Data Pre-procesing Step
2. # importing libraries
3. **import** numpy as nm
4. **import** matplotlib.pyplot as mtp
5. **import** pandas as pd
7. #importing datasets
8. data\_set= pd.read\_csv('user\_data.csv')

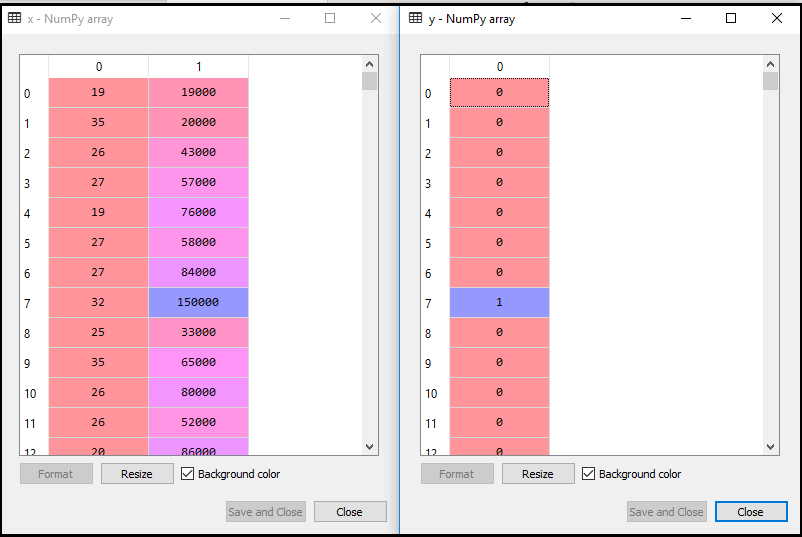
By executing the above lines of code, we will get the dataset as the output. Consider the given image:



Now, we will extract the dependent and independent variables from the given dataset. Below is the code for it:

1. #Extracting Independent and dependent Variable
2. x= data\_set.iloc[:, [2,3]].values
3. y= data\_set.iloc[:, 4].values

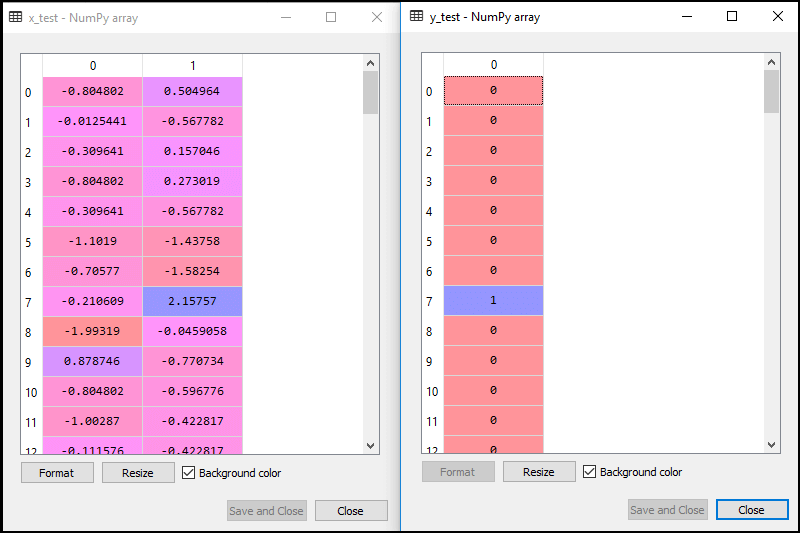
In the above code, we have taken [2, 3] for x because our independent variables are age and salary, which are at index 2, 3. And we have taken 4 for y variable because our dependent variable is at index 4. The output will be:



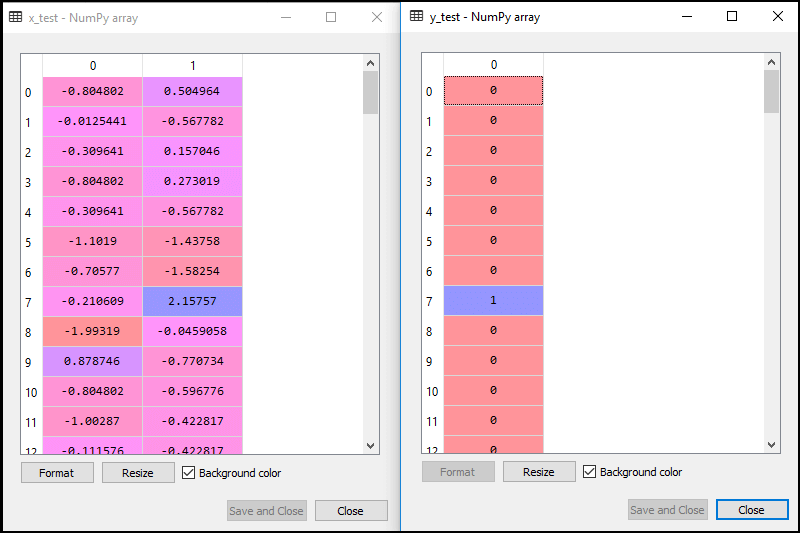
Now we will split the dataset into a training set and test set. Below is the code for it:

1. # Splitting the dataset into training and test set.
2. from sklearn.model\_selection **import** train\_test\_split
3. x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.25, random\_state=0)

The output for this is given below:

**For test set:** 

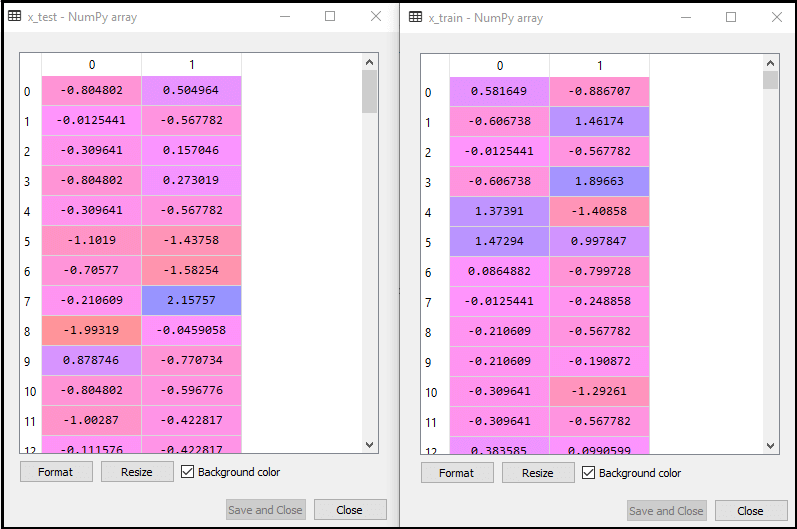
**For training set:**



In logistic regression, we will do feature scaling because we want accurate result of predictions. Here we will only scale the independent variable because dependent variable have only 0 and 1 values. Below is the code for it:

1. #feature Scaling
2. from sklearn.preprocessing **import** StandardScaler
3. st\_x= StandardScaler()
4. x\_train= st\_x.fit\_transform(x\_train)
5. x\_test= st\_x.transform(x\_test)

The scaled output is given below:



**2. Fitting Logistic Regression to the Training set:**

We have well prepared our dataset, and now we will train the dataset using the training set. For providing training or fitting the model to the training set, we will import the **LogisticRegression** class of the **sklearn** library.

After importing the class, we will create a classifier object and use it to fit the model to the logistic regression. Below is the code for it:

1. #Fitting Logistic Regression to the training set
2. from sklearn.linear\_model **import** LogisticRegression
3. classifier= LogisticRegression(random\_state=0)
4. classifier.fit(x\_train, y\_train)

**Output:** By executing the above code, we will get the below output:

**Out[5]:**

1. LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,
2. intercept\_scaling=1, l1\_ratio=None, max\_iter=100,
3. multi\_class='warn', n\_jobs=None, penalty='l2',
4. random\_state=0, solver='warn', tol=0.0001, verbose=0,
5. warm\_start=False)

Hence our model is well fitted to the training set.

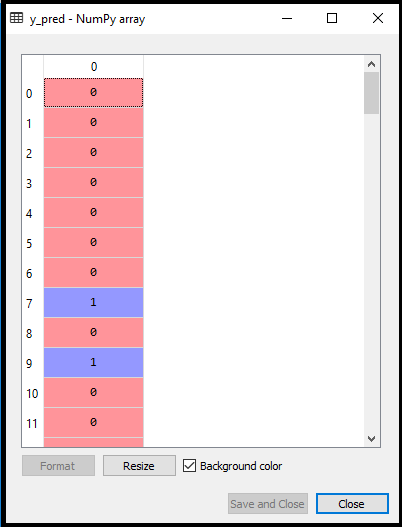
**3. Predicting the Test Result**

Our model is well trained on the training set, so we will now predict the result by using test set data. Below is the code for it:

1. #Predicting the test set result
2. y\_pred= classifier.predict(x\_test)

In the above code, we have created a y\_pred vector to predict the test set result.

**Output:** By executing the above code, a new vector (y\_pred) will be created under the variable explorer option. It can be seen as:



The above output image shows the corresponding predicted users who want to purchase or not purchase the car.

**4. Test Accuracy of the result**

Now we will create the confusion matrix here to check the accuracy of the classification. To create it, we need to import the **confusion\_matrix** function of the sklearn library. After importing the function, we will call it using a new variable **cm**. The function takes two parameters, mainly **y\_true**( the actual values) and **y\_pred** (the targeted value return by the classifier). Below is the code for it:

1. #Creating the Confusion matrix
2. from sklearn.metrics **import** confusion\_matrix
3. cm= confusion\_matrix()

**Output:**

By executing the above code, a new confusion matrix will be created. Consider the below image:

