K-Nearest Neighbors Technique

KNN is a supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point

In [1]:

```
#Importing the libraries "pandas and numpy" to my python script
#with the standard short name as "pd and np".

#Uploading the file on google colab and choosing the selected dataset by clicking "choose
files".

import pandas as pd
import numpy as np
from sklearn import metrics

from google.colab import files
uploaded = files.upload()
```

Choose File **No file selected**

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving New_Data5.csv to New_Data5.csv
```

In [2]:

```
#In the first step, importing the dataset in the project by the "read_csv" function
#and that reads the data into a pandas dataframe object.

#Data with 768 rows and 9 columns.

data = pd.read_csv('New_Data5.csv')
data
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Pedigree	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

In [3]:

```
#The shape is a tuple of the array dimensions which gives the number of rows and
# columns of a given dataframe
```

```
data.shape
```

Out[3]:

```
(768, 9)
```

In [4]:

```
#.value_counts() function returns a series containing the counts of unique values.

data.isnull().any().value_counts()
```

Out[4]:

```
False    9
dtype: int64
```

In [5]:

```
#df.columns attribute where return the column labels of the given dataframe.

data.columns
```

Out[5]:

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'Pedigree', 'Age', 'Outcome'],
      dtype='object')
```

In [6]:

```
#.drop() function is used to drop the Outcome column.

df_x = data.drop(columns='Outcome', axis=1)
df_y = data['Outcome']
```

In [7]:

```
#Importing standardscaler from sklearn.preprocessing.

#It is used to standardize the data values into a standard format.

from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
scaledX = scale.fit_transform(df_x)
```

Splitting into train and test

In [8]:

```
#Importing train_test_split function to split arrays or matrices into
#random subsets for train and test data.

#The purpose of split is to break the data into training and test or validation set.

#After that can check the validation of the model by fitting the training data and
#predicting using the test or validation by applying the technique.

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(scaledX, df_y, test_size=0.2, random
_state=42)
```

K-Nearest Neighbors Technique

In [9]:

```
#Importing KNN from sklearn.neighbors and then predict.

#It is a user-defined function which accepts an array of distances, and returns
#an array of the same shape containing the weights.
```

```
#.fit method takes the training data as arguments, which can be one array in the case
#of unsupervised learning, or two arrays in the case of supervised learning.

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(x_train, y_train)
y_pred = knn.predict(x_test)
```

Confusion Matrix

In [10]:

```
#.confusion_matrix() is a table that is used in classification problems to
#assess where errors in the model were made.

cs = metrics.confusion_matrix(y_test,y_pred)
print("Confusion matrix: \n",cs)
```

```
Confusion matrix:
 [[78 21]
 [28 27]]
```

Accuracy Score

In [11]:

```
#Now, using accuracy_score package calculates the accuracy score for a set
#of predicted labels against the true labels.

#Showing the accuracy score by prints it.

ac = metrics.accuracy_score(y_test, y_pred)
print("Accuracy score: ",ac)
```

```
Accuracy score:  0.6818181818181818
```

Error Rate

In [12]:

```
#Here showing the error rate what the percentage of our predictions are wrong.

er = 1-ac
print("Error rate: ",er)
```

```
Error rate:  0.31818181818181823
```

Precision

In [13]:

```
#The .precision_score() function is intuitively the ability of the classifier
#not to label as positive a sample that is negative.

p = metrics.precision_score(y_test,y_pred)
print("Precision: ", p)
```

```
Precision:  0.5625
```

Recall

In [14]:

```
#The .recall_score() function is intuitively the ability of the classifier to find all th
e positive samples.

r = metrics.recall_score(y_test,y_pred)
```

```
print("Recall: ", r)
```

```
Recall:  0.4909090909090909
```

Classification Report

In [15]:

```
#The .classification_report is used to show the performance evaluation metric,
#which is used to show all the precision, recall, f1-score and support.

cr = metrics.classification_report(y_test,y_pred)
print("Classification report: \n\n", cr)
```

```
Classification report:

              precision    recall  f1-score   support

           0       0.74      0.79      0.76        99
           1       0.56      0.49      0.52        55

    accuracy                           0.68       154
   macro avg       0.65      0.64      0.64       154
weighted avg       0.67      0.68      0.68       154
```