17BCE0136 R.S.Rahul Sai

K-Neighbors

In [1]:
```
import numpy as np
import pandas as pd
```

Dataset Description

Electrical Grid Stability Simulated Data Set

The local stability analysis of the 4-node star system (electricity producer in the center) implementing Decentralised Smart Grid Control concept.

In [2]:
```
data=pd.read_csv("data.csv")
```

In [3]:
```
data.head()
```

Out[3]:

	tau1	tau2	tau3	tau4	p1	p2	p3	p4	g1	g2	g3	
0	2.959060	3.079885	8.381025	9.780754	3.763085	-0.782604	-1.257395	-1.723086	0.650456	0.859578	0.887445	0.958(
1	9.304097	4.902524	3.047541	1.369357	5.067812	-1.940058	-1.872742	-1.255012	0.413441	0.862414	0.562139	0.7817
2	8.971707	8.848428	3.046479	1.214518	3.405158	-1.207456	-1.277210	-0.920492	0.163041	0.766689	0.839444	0.1098
3	0.716415	7.669600	4.486641	2.340563	3.963791	-1.027473	-1.938944	-0.997374	0.446209	0.976744	0.929381	0.3627
4	3.134112	7.608772	4.943759	9.857573	3.525811	-1.125531	-1.845975	-0.554305	0.797110	0.455450	0.656947	0.8209

In [4]:
```
from sklearn.preprocessing import StandardScaler,LabelBinarizer
from sklearn.model_selection import train_test_split
lb=LabelBinarizer()
sc=StandardScaler()
```

In [5]:
```
X=data.iloc[:,:-1]
Y=data.iloc[:,-1]
```

In [6]:
```
X=sc.fit_transform(X)
Y=lb.fit_transform(Y)
```

In [7]:
```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=0)
```

K-Neighbours (Library)

In [8]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
knn=KNeighborsClassifier(n_neighbors=3,metric='euclidean')
```

In [9]:
```
knn.fit(X_train,Y_train)
```

```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:1: DataConversionWarning: A column-vector y wa
s passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example
using ravel().
  """Entry point for launching an IPython kernel.
```

Out[9]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='euclidean',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

In [10]:
```
Y_pred=knn.predict(X_test)
```

In [11]:
```
print(classification_report(Y_test,Y_pred))
print("Accuracy: {0:.2f} %".format(knn.score(X_test,Y_test)*100))
```

```
              precision    recall  f1-score   support

           0       0.91      0.86      0.88       727
           1       0.92      0.95      0.94      1273

    accuracy                           0.92      2000
   macro avg       0.91      0.90      0.91      2000
weighted avg       0.92      0.92      0.92      2000

Accuracy: 91.65 %
```

K-Neighbours (Custom)

In [12]:
```
class KNeigbours(object):
    def __init__(self, k):
        self.k = k

    @staticmethod
    def euclid_dist(v1, v2):
        v1, v2 = np.array(v1), np.array(v2)
        distance = 0
        for i in range(len(v1) - 1):
            distance += (v1[i] - v2[i]) ** 2
        return np.sqrt(distance)

    def predict(self, train_set, test_inst):
        distances = []
        for i in range(len(train_set)):
            dist = self.euclid_dist(train_set[i][:-1], test_inst)
            distances.append((train_set[i], dist))
        distances.sort(key=lambda x: x[1])

        neighbours = []
        for i in range(self.k):
            neighbours.append(distances[i][0])

        classes = {}
        for i in range(len(neighbours)):
            response = neighbours[i][-1]
            if response in classes:
                classes[response] += 1
            else:
                classes[response] = 1

        sorted_classes = sorted(classes.items(), key=lambda x: x[1], reverse=True)
        return sorted_classes[0][0]

    @staticmethod
    def evaluate(y_true, y_pred):
        n_correct = 0
        for act, pred in zip(y_true, y_pred):
            if act == pred:
                n_correct += 1
        return n_correct / len(y_true)
```

In [13]:
```
knn=KNeigbours(k=3)
preds=[]
```

In [14]:
```
train_set=pd.concat([pd.DataFrame(X_train),pd.DataFrame(Y_train)],axis=1)
test_set=pd.concat([pd.DataFrame(X_test),pd.DataFrame(Y_test)],axis=1)
train_set=train_set.astype(float).values.tolist()
test_set=test_set.astype(float).values.tolist()
```

In [15]:
```
for row in test_set:
  predictors = row[:-1]
  pred=knn.predict(train_set,predictors)
  preds.append(pred)
```

In [16]:
```
actual = np.array(test_set)[:, -1]
print("Accuracy: {} %".format(knn.evaluate(actual, preds)*100))
```

```
Accuracy: 84.3 %
```

In [17]:
```
print(classification_report(Y_test,preds))
```

```
              precision    recall  f1-score   support

           0       0.83      0.72      0.77       727
           1       0.85      0.91      0.88      1273

    accuracy                           0.84      2000
   macro avg       0.84      0.82      0.82      2000
weighted avg       0.84      0.84      0.84      2000
```

Inference

```
Classification Accuracy of K-Neigbours algorithm (Library) : 91.65 % with k=3
Classification Accuracy of K-Neigbours algorithm (Custom)  : 84.30 % with k=3
```

Inference : The library function is better optimised in terms of the prediction subroutine than the custom written function as the custom written function.