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	0.109
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	0.820

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.

Clustering

Dataset Description

Anuran Calls (MFCCs) Data Set

Acoustic features extracted from syllables of anuran (frogs) calls, including the family, the genus, and the species labels (multilabel).

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

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

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

Out[3]:

	MFCCs_1	MFCCs_2	MFCCs_3	MFCCs_4	MFCCs_5	MFCCs_6	MFCCs_7	MFCCs_8	MFCCs_9	MFCCs_10	...	MFCCs_17	MFCCs_18
0	1.0	0.152936	-0.105586	0.200722	0.317201	0.260764	0.100945	-0.150063	-0.171128	0.124676	...	-0.108351	-0.077623
1	1.0	0.171534	-0.098975	0.268425	0.338672	0.268353	0.060835	-0.222475	-0.207693	0.170883	...	-0.090974	-0.056510
2	1.0	0.152317	-0.082973	0.287128	0.276014	0.189867	0.008714	-0.242234	-0.219153	0.232538	...	-0.050691	-0.023590
3	1.0	0.224392	0.118985	0.329432	0.372088	0.361005	0.015501	-0.194347	-0.098181	0.270375	...	-0.136009	-0.177037
4	1.0	0.087817	-0.068345	0.306967	0.330923	0.249144	0.006884	-0.265423	-0.172700	0.266434	...	-0.048885	-0.053074

5 rows × 26 columns

In [4]:
```
X=data.iloc[:,1:-4]
Y=data.iloc[:,-4]
```

In [5]:
```
Y.nunique()
```

Out[5]: 4

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

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

K-Means Clustering

K-means Clustering (Library)

In [8]:
```
from sklearn.cluster import KMeans
```

In [9]:
```
from sklearn.metrics import silhouette_score

x=[]
sil = []
kmax = 10

for k in range(2, kmax+1):
  kmeans = KMeans(n_clusters = k).fit(X_train)
  labels = kmeans.labels_
  sil.append(silhouette_score(X_train, labels, metric = 'euclidean'))
  x.append(k)
```

In [10]:
```
for i in zip(x,sil):
  print(i)
```

```
(2, 0.33615042568302983)
(3, 0.3586470357055605)
(4, 0.36090644028848684)
(5, 0.36469802624333525)
(6, 0.2825279468442282)
(7, 0.28995488336593667)
(8, 0.2932980160754407)
(9, 0.29943154667052946)
(10, 0.23809084635969954)
```

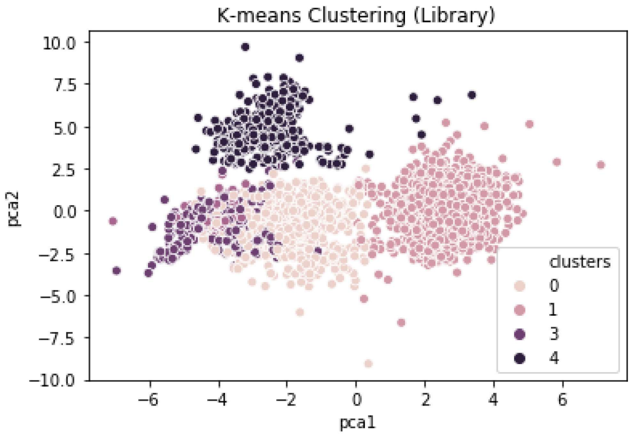
In [11]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
km = KMeans(n_clusters = 5)
```

In [12]:
```
data=pd.DataFrame(X_train)
data['clusters'] = km.fit_predict(data)
reduced_data = PCA(2).fit_transform(data)
```

In [13]:
```
reduced_data.shape
```

Out[13]:
```
(5756, 2)
```

In [14]:
```
results = pd.DataFrame(reduced_data,columns=['pca1','pca2'])
sns.scatterplot(x="pca1", y="pca2", hue=data['clusters'], data=results)
plt.title('K-means Clustering (Library)')
plt.show()
```



In [15]:
```
from sklearn.metrics import silhouette_score
silhouette_score(X_train,data['clusters'])
```

Out[15]:
```
0.3647438484624485
```

K-Means Clustering (Custom)

(continued on next page)

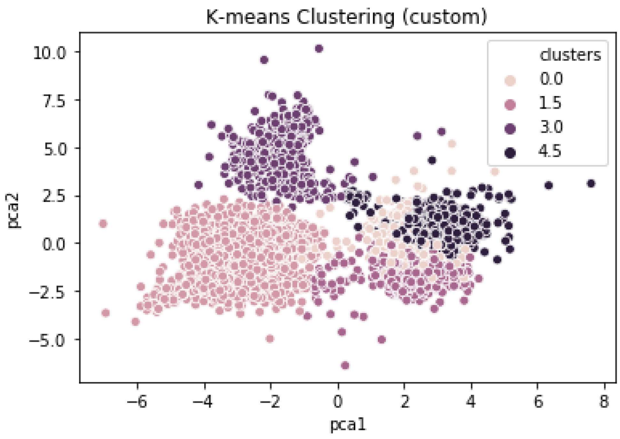
In [16]:
```python
class KMeans_custom:
    def __init__(self, n_clusters):
        self.data = pd.DataFrame()
        self.n_clusters = n_clusters
        self.centroids = pd.DataFrame()
        self.clusters = np.ndarray(1)
        self.old_centroids = pd.DataFrame()
        self.verbose = False
        self.predictions = list()

    def train(self, df, verbose):
        self.verbose = verbose
        self.data = df.copy(deep=True)
        self.clusters = np.zeros(len(self.data))

        if 'species' in self.data.columns:
            self.data.drop('species', axis=1, inplace=True)

        unique_rows = self.data.drop_duplicates()
        unique_rows.reset_index(drop=True, inplace=True)
        self.centroids = unique_rows.sample(n=self.n_clusters)
        self.centroids.reset_index(drop=True, inplace=True)

        if self.verbose:
            print("\nRandomly initiated centroids:")
            print(self.centroids)

        self.old_centroids = pd.DataFrame(np.zeros(shape=(self.n_clusters, self.data.shape[1])),
                                          columns=self.data.columns)

        while not self.old_centroids.equals(self.centroids):

            if self.verbose:
                time.sleep(3)

            self.old_centroids = self.centroids.copy(deep=True)

            for row_i in range(0, len(self.data)):
                distances = list()
                point = self.data.iloc[row_i]

                for row_c in range(0, len(self.centroids)):
                    centroid = self.centroids.iloc[row_c]
                    distances.append(np.linalg.norm(point - centroid))

                self.clusters[row_i] = np.argmin(distances)

            for cls in range(0, self.n_clusters):

                cls_idx = np.where(self.clusters == cls)[0]

                if len(cls_idx) == 0:
                    self.centroids.loc[cls] = self.old_centroids.loc[cls]
                else:
                    # Set the new k-mean to the mean value of the data points within this cluster
                    self.centroids.loc[cls] = self.data.iloc[cls_idx].mean()

                if self.verbose:
                    print("\nRow indices belonging to cluster {}: [n={}]".format(cls, len(cls_idx)))
                    print(cls_idx)

            if self.verbose:
                print("\nOld centroids:")
                print(self.old_centroids)
                print("New centroids:")
                print(self.centroids)
```

In [17]:
```python
km = KMeans_custom(n_clusters=5)
```

In [18]:
```python
data=pd.DataFrame(X_train)
km.train(data,verbose=False)
```

In [19]:
```python
data['clusters'] = km.clusters
reduced_data = PCA(2).fit_transform(data)
```

In [20]:
```
results = pd.DataFrame(reduced_data,columns=['pca1','pca2'])
sns.scatterplot(x="pca1", y="pca2", hue=data['clusters'], data=results)
plt.title('K-means Clustering (custom)')
plt.show()
```



In [21]:
```
silhouette_score(X_train,data['clusters'])
```

Out[21]: 0.19872144546341883

Inference

```
K-Means Clustering (Library) Silhouette Score : 0.36 where n=5
K-Means Clustering (Custom)  Silhouette Score : 0.19 where n=5
```

Lower silhouette score of the custom algorithm indicates more overlapping of the clusters, thereby the library function gets a better score due to it's optimised techniques.

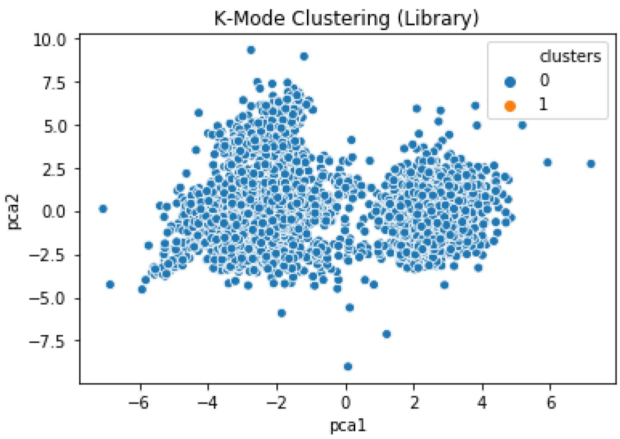
K-Mode Clustering

K-Mode Clustering (Library)

In [22]:
```
from kmodes.kmodes import KModes
kmo = KModes(n_clusters = 2)
```

In [23]:
```
data=pd.DataFrame(X_train)
data['clusters'] = kmo.fit_predict(data)
reduced_data = PCA(2).fit_transform(data)
```

In [24]:
```
results = pd.DataFrame(reduced_data,columns=['pca1','pca2'])
sns.scatterplot(x="pca1", y="pca2", hue=data['clusters'], data=results)
plt.title('K-Mode Clustering (Library)')
plt.show()
```



In [25]:
```
silhouette_score(X_train,data['clusters'])
```

Out[25]: 0.18575828514490955

K-Mode Clustering (Custom)

In [26]:
```
from sklearn.base import BaseEstimator, ClusterMixin
from sklearn.utils import check_random_state
from sklearn.utils.validation import check_array
```

In [27]:
```
from util import get_max_value_key, encode_features, get_unique_rows, decode_centroids, pandas_to_numpy
from util.dissim import matching_dissim, ng_dissim
```

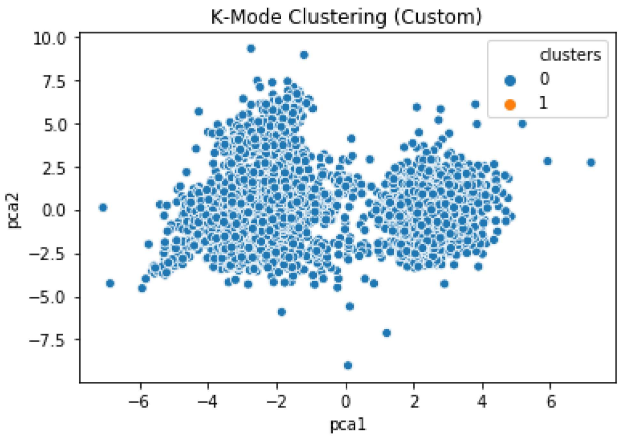
In [28]:
```
from helper import init_huang,init_cao,move_point_cat,_labels_cost,_k_modes_iter,k_modes,k_modes_single
```

In [29]:
```
class KModes_Custom(BaseEstimator, ClusterMixin):
    def __init__(self, n_clusters=8, max_iter=100, cat_dissim=matching_dissim,
                 init='Cao', n_init=1, verbose=0, random_state=None, n_jobs=1):

        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.cat_dissim = cat_dissim
        self.init = init
        self.n_init = n_init
        self.verbose = verbose
        self.random_state = random_state
        self.n_jobs = n_jobs
        if ((isinstance(self.init, str) and self.init == 'Cao') or
                hasattr(self.init, '__array__')) and self.n_init > 1:
            if self.verbose:
                print("Initialization method and algorithm are deterministic. "
                      "Setting n_init to 1.")
            self.n_init = 1

    def fit(self, X, y=None, **kwargs):
        X = pandas_to_numpy(X)

        random_state = check_random_state(self.random_state)
        self._enc_cluster_centroids, self._enc_map, self.labels_, self.cost_, \
        self.n_iter_, self.epoch_costs_ = k_modes(
            X,
            self.n_clusters,
            self.max_iter,
            self.cat_dissim,
            self.init,
            self.n_init,
            self.verbose,
            random_state,
            self.n_jobs,
        )
        return self

    def fit_predict(self, X, y=None, **kwargs):
        return self.fit(X, **kwargs).predict(X, **kwargs)

    def predict(self, X, **kwargs):
        assert hasattr(self, '_enc_cluster_centroids'), "Model not yet fitted."
        X = pandas_to_numpy(X)
        X = check_array(X, dtype=None)
        X, _ = encode_features(X, enc_map=self._enc_map)
        return _labels_cost(X, self._enc_cluster_centroids, self.cat_dissim)[0]

    @property
    def cluster_centroids_(self):
        if hasattr(self, '_enc_cluster_centroids'):
            return decode_centroids(self._enc_cluster_centroids, self._enc_map)
        else:
            raise AttributeError("'{}' object has no attribute 'cluster_centroids_' "
                                 "because the model is not yet fitted.")
```

In [30]:
```
kmo = KModes_Custom(n_clusters = 2)
```

In [31]:
```
data=pd.DataFrame(X_train)
data['clusters'] = kmo.fit_predict(data)
reduced_data = PCA(2).fit_transform(data)
```

In [32]:
```
results = pd.DataFrame(reduced_data,columns=['pca1','pca2'])
sns.scatterplot(x="pca1", y="pca2", hue=data['clusters'], data=results)
plt.title('K-Mode Clustering (Custom)')
plt.show()
```



In [34]:
```
silhouette_score(X_train,data['clusters'])
```

Out[34]: 0.11330828514490955

Inference

```
K-Modes Clustering (Library) Silhouette Score : 0.18 where n=2
K-Modes Clustering (Custom)  Silhouette Score : 0.11 where n=2
```

- The low performance of k-mode clustering algorithms is due to the fact that k-mode is more suited for categorical variables and this dataset lacks categorical variables.
- Lower silhouette score of the custom algorithm indicates more overlapping of the clusters, thereby the library function gets a better score due to it's optimised techniques.