Practical 8

Aim: Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

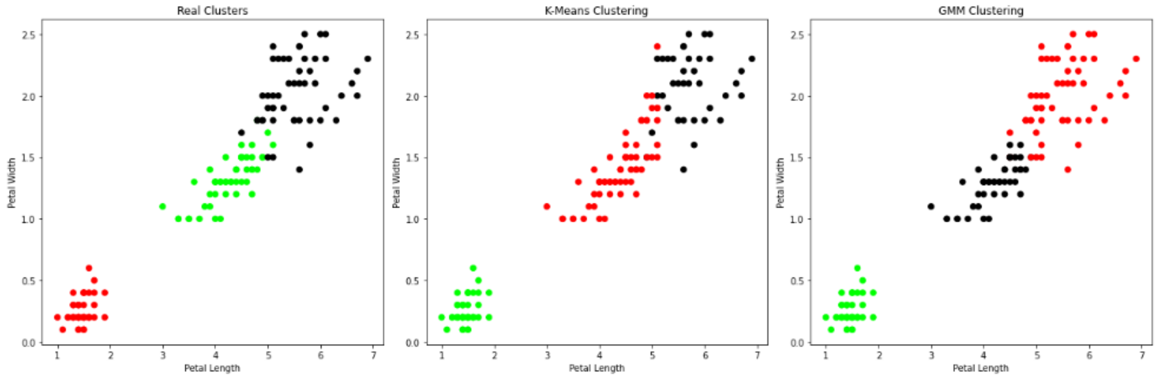
Data Set:

	A	B	C	D	E	F
1	Id	SepalLeng	SepalWidt	PetalLeng	PetalWidt	Species
2	1	5.1	3.5	1.4	0.2	Iris-setosa
3	2	4.9	3	1.4	0.2	Iris-setosa
4	3	4.7	3.2	1.3	0.2	Iris-setosa
5	4	4.6	3.1	1.5	0.2	Iris-setosa
6	5	5	3.6	1.4	0.2	Iris-setosa
7	6	5.4	3.9	1.7	0.4	Iris-setosa
8	7	4.6	3.4	1.4	0.3	Iris-setosa
9	8	5	3.4	1.5	0.2	Iris-setosa
10	9	4.4	2.9	1.4	0.2	Iris-setosa
11	10	4.9	3.1	1.5	0.1	Iris-setosa
12	11	5.4	3.7	1.5	0.2	Iris-setosa
13	12	4.8	3.4	1.6	0.2	Iris-setosa
14	13	4.8	3	1.4	0.1	Iris-setosa
15	14	4.3	3	1.1	0.1	Iris-setosa
16	15	5.8	4	1.2	0.2	Iris-setosa
17	16	5.7	4.4	1.5	0.4	Iris-setosa
18	17	5.4	3.9	1.3	0.4	Iris-setosa
19	18	5.1	3.5	1.4	0.3	Iris-setosa
20	19	5.7	3.8	1.7	0.3	Iris-setosa
21	20	5.1	3.8	1.5	0.3	Iris-setosa
22	21	5.4	3.4	1.7	0.2	Iris-setosa
23	22	5.1	3.7	1.5	0.4	Iris-setosa
24	23	4.6	3.6	1	0.2	Iris-setosa
25	24	5.1	3.3	1.7	0.5	Iris-setosa
26	25	4.8	3.4	1.9	0.2	Iris-setosa
27	26	5	3	1.6	0.2	Iris-setosa
28	27	5	3.4	1.6	0.4	Iris-setosa
29	28	5.2	3.5	1.5	0.2	Iris-setosa
30	29	5.2	3.4	1.4	0.2	Iris-setosa

iris

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn import preprocessing
import pandas as pd
import numpy as np

# Load the Iris dataset
iris = datasets.load_iris()
X = pd.DataFrame(iris.data, columns=['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(iris.target, columns=['Targets'])

# KMeans clustering
kmeans_model = KMeans(n_clusters=3, random_state=42)
kmeans_model.fit(X)

# Standardize the data
scaler = preprocessing.StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)

# Gaussian Mixture Model (GMM) clustering
gmm = GaussianMixture(n_components=3, random_state=42)  # Set n_components to match the number of true clusters
gmm.fit(X_scaled_df)

# Visualization
plt.figure(figsize=(18, 6))
colormap = np.array(['red', 'lime', 'black'])
```

```
# Real clusters
plt.subplot(1, 3, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# K-Means clustering
plt.subplot(1, 3, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[kmeans_model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# GMM clustering
plt.subplot(1, 3, 3)
gmm_labels = gmm.predict(X_scaled_df)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[gmm_labels], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

plt.tight_layout()
plt.show()

# Observation
print('Observation: The GMM using EM algorithm-based clustering matched the true labels more closely than KMeans.')
```




```
Observation: The GMM using EM algorithm-based clustering matched the true labels more closely than KM
eans.
```

Practical : 9

Aim: Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

Dataset:

	A	B	C	D	E	F
1	Id	SepalLeng	SepalWidt	PetalLeng	PetalWidt	Species
2	1	5.1	3.5	1.4	0.2	Iris-setosa
3	2	4.9	3	1.4	0.2	Iris-setosa
4	3	4.7	3.2	1.3	0.2	Iris-setosa
5	4	4.6	3.1	1.5	0.2	Iris-setosa
6	5	5	3.6	1.4	0.2	Iris-setosa
7	6	5.4	3.9	1.7	0.4	Iris-setosa
8	7	4.6	3.4	1.4	0.3	Iris-setosa
9	8	5	3.4	1.5	0.2	Iris-setosa
10	9	4.4	2.9	1.4	0.2	Iris-setosa
11	10	4.9	3.1	1.5	0.1	Iris-setosa
12	11	5.4	3.7	1.5	0.2	Iris-setosa
13	12	4.8	3.4	1.6	0.2	Iris-setosa
14	13	4.8	3	1.4	0.1	Iris-setosa
15	14	4.3	3	1.1	0.1	Iris-setosa
16	15	5.8	4	1.2	0.2	Iris-setosa
17	16	5.7	4.4	1.5	0.4	Iris-setosa
18	17	5.4	3.9	1.3	0.4	Iris-setosa
19	18	5.1	3.5	1.4	0.3	Iris-setosa
20	19	5.7	3.8	1.7	0.3	Iris-setosa
21	20	5.1	3.8	1.5	0.3	Iris-setosa
22	21	5.4	3.4	1.7	0.2	Iris-setosa
23	22	5.1	3.7	1.5	0.4	Iris-setosa
24	23	4.6	3.6	1	0.2	Iris-setosa
25	24	5.1	3.3	1.7	0.5	Iris-setosa
26	25	4.8	3.4	1.9	0.2	Iris-setosa
27	26	5	3	1.6	0.2	Iris-setosa
28	27	5	3.4	1.6	0.4	Iris-setosa
29	28	5.2	3.5	1.5	0.2	Iris-setosa
30	29	5.2	3.4	1.4	0.2	Iris-setosa

iris

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Display the features and target classes
print('Features (sepal-length, sepal-width, petal-length, petal-width):')
print(X)
print('Class: 0 - Iris-Setosa, 1 - Iris-Versicolor, 2 - Iris-Virginica')
print(y)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create and fit the K-Nearest Neighbors classifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

# Predict the labels for the test set
y_pred = classifier.predict(X_test)

# Display the confusion matrix and classification report
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))

print('Accuracy Metrics:')
print(classification_report(y_test, y_pred))
```

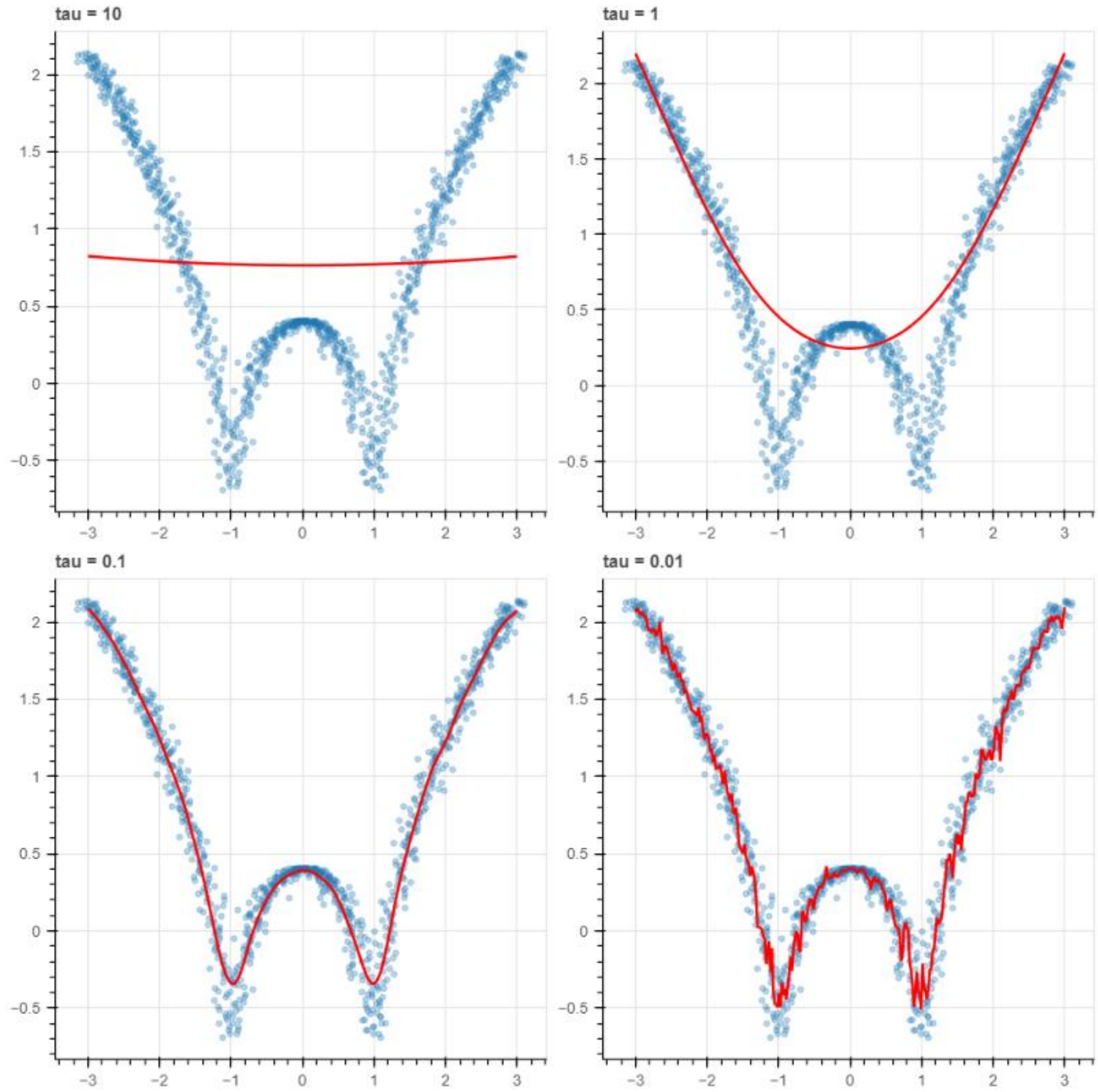
```
Features (sepal-length, sepal-width, petal-length, petal-width):
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
```

```
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
Class: 0 - Iris-Setosa, 1 - Iris-Versicolor, 2 - Iris-Virginica
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Practical : 10

Aim: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

[1]:
```
import numpy as np
from bokeh.plotting import figure, show
from bokeh.layouts import gridplot

# Local Weighted Regression (LWR) function
def local_regression(x0, X, Y, tau):
    # Add a bias term to x0
    x0 = np.r_[1, x0]

    # Add bias term to X
    X = np.c_[np.ones(len(X)), X]

    # Calculate the kernelized weight for each point
    xw = X.T * radial_kernel(x0, X, tau)  # XTranspose * Weight (W)

    # Compute beta (the weight vector)
    beta = np.linalg.pinv(xw @ X) @ xw @ Y  # Using pseudo-inverse for stability

    # Predict the value
    return x0 @ beta  # Predict the value using dot product
# Radial kernel function (Gaussian)
def radial_kernel(x0, X, tau):
    # Calculate radial kernel (Gaussian weights)
    return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))

# Generate dataset
n = 1000
X = np.linspace(-3, 3, num=n)
Y = np.log(np.abs(X ** 2 - 1) + 0.5)

# Add jitter to X to simulate noise
X += np.random.normal(scale=0.1, size=n)

# Define the domain for prediction
domain = np.linspace(-3, 3, num=300)

# Function to plot the locally weighted regression
def plot_lwr(tau):
    # Make predictions using the local regression model
    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]

    # Create plot
    plot = figure(width=400, height=400)
    plot.title.text = 'tau = %g' % tau  # Display tau value in the title

    # Scatter plot of the original data
    plot.scatter(X, Y, alpha=0.3)

    # Line plot of the predicted regression
    plot.line(domain, prediction, line_width=2, color='red')

    return plot

# Display the plots with different tau values in a grid layout
show(gridplot([[plot_lwr(10.0), plot_lwr(1.0)], [plot_lwr(0.1), plot_lwr(0.01)]]))
```


tau = 10
tau = 1
tau = 0.1
tau = 0.01

→ tips.csv

	A	B	C	D	E	F	G
1	total_bill	tip	sex	smoker	day	time	size
2	16.99	1.01	Female	No	Sun	Dinner	2
3	10.34	1.66	Male	No	Sun	Dinner	3
4	21.01	3.5	Male	No	Sun	Dinner	3
5	23.68	3.31	Male	No	Sun	Dinner	2
6	24.59	3.61	Female	No	Sun	Dinner	4
7	25.29	4.71	Male	No	Sun	Dinner	4
8	8.77	2	Male	No	Sun	Dinner	2
9	26.88	3.12	Male	No	Sun	Dinner	4
10	15.04	1.96	Male	No	Sun	Dinner	2
11	14.78	3.23	Male	No	Sun	Dinner	2
12	10.27	1.71	Male	No	Sun	Dinner	2
13	35.26	5	Female	No	Sun	Dinner	4
14	15.42	1.57	Male	No	Sun	Dinner	2
15	18.43	3	Male	No	Sun	Dinner	4
16	14.83	3.02	Female	No	Sun	Dinner	2
17	21.58	3.92	Male	No	Sun	Dinner	2
18	10.33	1.67	Female	No	Sun	Dinner	3
19	16.29	3.71	Male	No	Sun	Dinner	3

→ Output:

```
The Data Set (10 Samples) X:
 [-2.99399399 -2.98798799 -2.98198198 -2.97597598 -2.96996997 -2.96396396
 -2.95795796 -2.95195195 -2.94594595]
The Fitting Curve Data Set (10 Samples) Y:
 [2.13582188 2.13156806 2.12730467 2.12303166 2.11874898 2.11445659
 2.11015444 2.10584249 2.10152068]
Normalized (10 Samples) X:
 [-2.94973156 -2.85391642 -2.91528377 -2.72415615 -3.02368489 -2.86973494
 -2.93495497 -3.10473025 -2.84580848]
Xo Domain Space (10 Samples):
 [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
 -2.85953177 -2.83946488 -2.81939799]
```