**6. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.**

**NaiveBayes.ipynb**

**import** pandas **as** pd

**from** sklearn **import** tree

**from** sklearn.preprocessing **import** LabelEncoder

**from** sklearn.naive\_bayes **import** GaussianNB

*# Load Data from CSV*

data **=** pd**.**read\_csv('p-tennis.csv')

print("The first 5 Values of data is :\n", data**.**head())

*# obtain train data and train output*

X **=** data**.**iloc[:, :**-**1]

print("\nThe First 5 values of the train data is\n", X**.**head())

y **=** data**.**iloc[:, **-**1]

print("\nThe First 5 values of train output is\n", y**.**head())

*# convert them in numbers*

le\_outlook **=** LabelEncoder()

X**.**Outlook **=** le\_outlook**.**fit\_transform(X**.**Outlook)

le\_Temperature **=** LabelEncoder()

X**.**Temperature **=** le\_Temperature**.**fit\_transform(X**.**Temperature)

le\_Humidity **=** LabelEncoder()

X**.**Humidity **=** le\_Humidity**.**fit\_transform(X**.**Humidity)

le\_Windy **=** LabelEncoder()

X**.**Windy **=** le\_Windy**.**fit\_transform(X**.**Windy)

print("\nNow the Train output is\n", X**.**head())

le\_PlayTennis **=** LabelEncoder()

y **=** le\_PlayTennis**.**fit\_transform(y)

print("\nNow the Train output is\n",y)

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

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X,y, test\_size **=** 0.20)

classifier **=** GaussianNB()

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

**from** sklearn.metrics **import** accuracy\_score

print("Accuracy is:", accuracy\_score(classifier**.**predict(X\_test), y\_test))

**NaiveBayes Output:**

The first 5 Values of data is :

Outlook Temperature Humidity Windy PlayTennis

0 Sunny Hot High False No

1 Sunny Hot High True No

2 Overcast Hot High False Yes

3 Rainy Mild High False Yes

4 Rainy Cool Normal False Yes

The First 5 values of the train data is

Outlook Temperature Humidity Windy

0 Sunny Hot High False

1 Sunny Hot High True

2 Overcast Hot High False

3 Rainy Mild High False

4 Rainy Cool Normal False

The First 5 values of train output is

0 No

1 No

2 Yes

3 Yes

4 Yes

Name: PlayTennis, dtype: object

Now the Train output is

Outlook Temperature Humidity Windy

0 2 1 0 0

1 2 1 0 1

2 0 1 0 0

3 1 2 0 0

4 1 0 1 0

Now the Train output is

[0 0 1 1 1 0 1 0 1 1 1 1 1 0]

Accuracy is: 0.6666666666666666

**7. 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.**

**S-EM-Kmeans.ipynb**

*# Kmeans*

**from** sklearn **import** datasets

**from** sklearn **import** metrics

**from** sklearn.cluster **import** KMeans

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

iris **=** datasets**.**load\_iris()

print(iris)

X\_train,X\_test,y\_train,y\_test **=** train\_test\_split(iris**.**data,iris**.**target)

model **=**KMeans(n\_clusters**=**3)

model**.**fit(X\_train,y\_train)

model**.**score

print('K-Mean: ',metrics**.**accuracy\_score(y\_test,model**.**predict(X\_test)))

*#-------Expectation and Maximization----------*

**from** sklearn.mixture **import** GaussianMixture

model2 **=** GaussianMixture(n\_components**=**3)

model2**.**fit(X\_train,y\_train)

model2**.**score

print('EM Algorithm:',metrics**.**accuracy\_score(y\_test,model2**.**predict(X\_test)))

**Output:**

{'data': array([[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],

[5.7, 3.8, 1.7, 0.3],

[5.1, 3.8, 1.5, 0.3],

[5.4, 3.4, 1.7, 0.2],

[5.1, 3.7, 1.5, 0.4],

[4.6, 3.6, 1. , 0.2],

[5.1, 3.3, 1.7, 0.5],

[4.8, 3.4, 1.9, 0.2],

[5. , 3. , 1.6, 0.2],

[5. , 3.4, 1.6, 0.4],

[5.2, 3.5, 1.5, 0.2],

[5.2, 3.4, 1.4, 0.2],

[4.7, 3.2, 1.6, 0.2],

[4.8, 3.1, 1.6, 0.2],

[5.4, 3.4, 1.5, 0.4],

[5.2, 4.1, 1.5, 0.1],

[5.5, 4.2, 1.4, 0.2],

[4.9, 3.1, 1.5, 0.2],

[5. , 3.2, 1.2, 0.2],

[5.5, 3.5, 1.3, 0.2],

[4.9, 3.6, 1.4, 0.1],

[4.4, 3. , 1.3, 0.2],

[5.1, 3.4, 1.5, 0.2],

[5. , 3.5, 1.3, 0.3],

[4.5, 2.3, 1.3, 0.3],

[4.4, 3.2, 1.3, 0.2],

[5. , 3.5, 1.6, 0.6],

[5.1, 3.8, 1.9, 0.4],

[4.8, 3. , 1.4, 0.3],

[5.1, 3.8, 1.6, 0.2],

[4.6, 3.2, 1.4, 0.2],

[5.3, 3.7, 1.5, 0.2],

[5. , 3.3, 1.4, 0.2],

[7. , 3.2, 4.7, 1.4],

[6.4, 3.2, 4.5, 1.5],

[6.9, 3.1, 4.9, 1.5],

[5.5, 2.3, 4. , 1.3],

[6.5, 2.8, 4.6, 1.5],

[5.7, 2.8, 4.5, 1.3],

[6.3, 3.3, 4.7, 1.6],

[4.9, 2.4, 3.3, 1. ],

[6.6, 2.9, 4.6, 1.3],

[5.2, 2.7, 3.9, 1.4],

[5. , 2. , 3.5, 1. ],

[5.9, 3. , 4.2, 1.5],

[6. , 2.2, 4. , 1. ],

[6.1, 2.9, 4.7, 1.4],

[5.6, 2.9, 3.6, 1.3],

[6.7, 3.1, 4.4, 1.4],

[5.6, 3. , 4.5, 1.5],

[5.8, 2.7, 4.1, 1. ],

[6.2, 2.2, 4.5, 1.5],

[5.6, 2.5, 3.9, 1.1],

[5.9, 3.2, 4.8, 1.8],

[6.1, 2.8, 4. , 1.3],

[6.3, 2.5, 4.9, 1.5],

[6.1, 2.8, 4.7, 1.2],

[6.4, 2.9, 4.3, 1.3],

[6.6, 3. , 4.4, 1.4],

[6.8, 2.8, 4.8, 1.4],

[6.7, 3. , 5. , 1.7],

[6. , 2.9, 4.5, 1.5],

[5.7, 2.6, 3.5, 1. ],

[5.5, 2.4, 3.8, 1.1],

[5.5, 2.4, 3.7, 1. ],

[5.8, 2.7, 3.9, 1.2],

[6. , 2.7, 5.1, 1.6],

[5.4, 3. , 4.5, 1.5],

[6. , 3.4, 4.5, 1.6],

[6.7, 3.1, 4.7, 1.5],

[6.3, 2.3, 4.4, 1.3],

[5.6, 3. , 4.1, 1.3],

[5.5, 2.5, 4. , 1.3],

[5.5, 2.6, 4.4, 1.2],

[6.1, 3. , 4.6, 1.4],

[5.8, 2.6, 4. , 1.2],

[5. , 2.3, 3.3, 1. ],

[5.6, 2.7, 4.2, 1.3],

[5.7, 3. , 4.2, 1.2],

[5.7, 2.9, 4.2, 1.3],

[6.2, 2.9, 4.3, 1.3],

[5.1, 2.5, 3. , 1.1],

[5.7, 2.8, 4.1, 1.3],

[6.3, 3.3, 6. , 2.5],

[5.8, 2.7, 5.1, 1.9],

[7.1, 3. , 5.9, 2.1],

[6.3, 2.9, 5.6, 1.8],

[6.5, 3. , 5.8, 2.2],

[7.6, 3. , 6.6, 2.1],

[4.9, 2.5, 4.5, 1.7],

[7.3, 2.9, 6.3, 1.8],

[6.7, 2.5, 5.8, 1.8],

[7.2, 3.6, 6.1, 2.5],

[6.5, 3.2, 5.1, 2. ],

[6.4, 2.7, 5.3, 1.9],

[6.8, 3. , 5.5, 2.1],

[5.7, 2.5, 5. , 2. ],

[5.8, 2.8, 5.1, 2.4],

[6.4, 3.2, 5.3, 2.3],

[6.5, 3. , 5.5, 1.8],

[7.7, 3.8, 6.7, 2.2],

[7.7, 2.6, 6.9, 2.3],

[6. , 2.2, 5. , 1.5],

[6.9, 3.2, 5.7, 2.3],

[5.6, 2.8, 4.9, 2. ],

[7.7, 2.8, 6.7, 2. ],

[6.3, 2.7, 4.9, 1.8],

[6.7, 3.3, 5.7, 2.1],

[7.2, 3.2, 6. , 1.8],

[6.2, 2.8, 4.8, 1.8],

[6.1, 3. , 4.9, 1.8],

[6.4, 2.8, 5.6, 2.1],

[7.2, 3. , 5.8, 1.6],

[7.4, 2.8, 6.1, 1.9],

[7.9, 3.8, 6.4, 2. ],

[6.4, 2.8, 5.6, 2.2],

[6.3, 2.8, 5.1, 1.5],

[6.1, 2.6, 5.6, 1.4],

[7.7, 3. , 6.1, 2.3],

[6.3, 3.4, 5.6, 2.4],

[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]]), 'target': array([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]), 'frame': None, 'target\_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'), 'DESCR': '.. \_iris\_dataset:\n\nIris plants dataset\n--------------------\n\n\*\*Data Set Characteristics:\*\*\n\n :Number of Instances: 150 (50 in each of three classes)\n :Number of Attributes: 4 numeric, predictive attributes and the class\n :Attribute Information:\n - sepal length in cm\n - sepal width in cm\n - petal length in cm\n - petal width in cm\n - class:\n - Iris-Setosa\n - Iris-Versicolour\n - Iris-Virginica\n \n :Summary Statistics:\n\n ============== ==== ==== ======= ===== ====================\n Min Max Mean SD Class Correlation\n ============== ==== ==== ======= ===== ====================\n sepal length: 4.3 7.9 5.84 0.83 0.7826\n sepal width: 2.0 4.4 3.05 0.43 -0.4194\n petal length: 1.0 6.9 3.76 1.76 0.9490 (high!)\n petal width: 0.1 2.5 1.20 0.76 0.9565 (high!)\n ============== ==== ==== ======= ===== ====================\n\n :Missing Attribute Values: None\n :Class Distribution: 33.3% for each of 3 classes.\n :Creator: R.A. Fisher\n :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n :Date: July, 1988\n\nThe famous Iris database, first used by Sir R.A. Fisher. The dataset is taken\nfrom Fisher\'s paper. Note that it\'s the same as in R, but not as in the UCI\nMachine Learning Repository, which has two wrong data points.\n\nThis is perhaps the best known database to be found in the\npattern recognition literature. Fisher\'s paper is a classic in the field and\nis referenced frequently to this day. (See Duda & Hart, for example.) The\ndata set contains 3 classes of 50 instances each, where each class refers to a\ntype of iris plant. One class is linearly separable from the other 2; the\nlatter are NOT linearly separable from each other.\n\n.. topic:: References\n\n - Fisher, R.A. "The use of multiple measurements in taxonomic problems"\n Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n Mathematical Statistics" (John Wiley, NY, 1950).\n - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.\n (Q327.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.\n - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System\n Structure and Classification Rule for Recognition in Partially Exposed\n Environments". IEEE Transactions on Pattern Analysis and Machine\n Intelligence, Vol. PAMI-2, No. 1, 67-71.\n - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule". IEEE Transactions\n on Information Theory, May 1972, 431-433.\n - See also: 1988 MLC Proceedings, 54-64. Cheeseman et al"s AUTOCLASS II\n conceptual clustering system finds 3 classes in the data.\n - Many, many more ...', 'feature\_names': ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'], 'filename': 'C:\\Users\\HP\\anaconda3\\lib\\site-packages\\sklearn\\datasets\\data\\iris.csv'}

K-Mean: 0.8421052631578947

EM Algorithm: 0.9210526315789473

**8. 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.**

**S-KNN.ipynb**

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

**from** sklearn.neighbors **import** KNeighborsClassifier

**from** sklearn **import** datasets

iris**=**datasets**.**load\_iris()

print("Iris Data set loaded...")

x\_train, x\_test, y\_train, y\_test **=** train\_test\_split(iris**.**data,iris**.**target,test\_size**=**0.1)

*#random\_state=0*

**for** i **in** range(len(iris**.**target\_names)):

print("Label", i , "-",str(iris**.**target\_names[i]))

classifier **=** KNeighborsClassifier(n\_neighbors**=**5)

classifier**.**fit(x\_train, y\_train)

y\_pred**=**classifier**.**predict(x\_test)

print("Results of Classification using K-nn with K=5 ")

**for** r **in** range(0,len(x\_test)):

print(" Sample:", str(x\_test[r]), " Actual-label:", str(y\_test[r])," Predicted-label:", str(y\_pred[r]))

print("Classification Accuracy :" , classifier**.**score(x\_test,y\_test));

**Output:**

Iris Data set loaded...

Label 0 - setosa

Label 1 - versicolor

Label 2 - virginica

Results of Classification using K-nn with K=5

Sample: [6.8 3.2 5.9 2.3] Actual-label: 2 Predicted-label: 2

Classification Accuracy : 0.9333333333333333

Sample: [5.8 2.6 4. 1.2] Actual-label: 1 Predicted-label: 1

Classification Accuracy : 0.9333333333333333

Sample: [6.3 3.3 4.7 1.6] Actual-label: 1 Predicted-label: 1

Classification Accuracy : 0.9333333333333333

Sample: [6.2 3.4 5.4 2.3] Actual-label: 2 Predicted-label: 2

Classification Accuracy : 0.9333333333333333

Sample: [7.1 3. 5.9 2.1] Actual-label: 2 Predicted-label: 2

Classification Accuracy : 0.9333333333333333

Sample: [5.4 3.7 1.5 0.2] Actual-label: 0 Predicted-label: 0

Classification Accuracy : 0.9333333333333333

Sample: [6.3 2.5 4.9 1.5] Actual-label: 1 Predicted-label: 2

Classification Accuracy : 0.9333333333333333

Sample: [6.8 3. 5.5 2.1] Actual-label: 2 Predicted-label: 2

Classification Accuracy : 0.9333333333333333

Sample: [5.6 2.9 3.6 1.3] Actual-label: 1 Predicted-label: 1

Classification Accuracy : 0.9333333333333333

Sample: [5.5 2.3 4. 1.3] Actual-label: 1 Predicted-label: 1

Classification Accuracy : 0.9333333333333333

Sample: [5.6 3. 4.5 1.5] Actual-label: 1 Predicted-label: 1

Classification Accuracy : 0.9333333333333333

Sample: [5.8 2.7 5.1 1.9] Actual-label: 2 Predicted-label: 2

Classification Accuracy : 0.9333333333333333

Sample: [7.3 2.9 6.3 1.8] Actual-label: 2 Predicted-label: 2

Classification Accuracy : 0.9333333333333333

Sample: [5. 3.5 1.6 0.6] Actual-label: 0 Predicted-label: 0

Classification Accuracy : 0.9333333333333333

Sample: [6.4 3.2 5.3 2.3] Actual-label: 2 Predicted-label: 2

Classification Accuracy : 0.9333333333333333

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

**S-LinearRegression.ipynb**

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

x **=** np**.**linspace(**-**5, 5, 1000)

y **=** np**.**log(np**.**abs((x **\*\*** 2) **-** 1) **+** 0.5)

x **=** x **+** np**.**random**.**normal(scale**=**0.05, size**=**1000)

plt**.**scatter(x, y, alpha**=**0.3)

**def** local\_regression(x0, x, y, tau):

x0 **=** np**.**r\_[1, x0]

x **=** np**.**c\_[np**.**ones(len(x)), x]

xw **=**x**.**T **\*** radial\_kernel(x0, x, tau)

beta **=** np**.**linalg**.**pinv(xw **@** x) **@** xw **@** y

**return** x0 **@** beta

**def** radial\_kernel(x0, x, tau):

**return** np**.**exp(np**.**sum((x **-** x0) **\*\*** 2, axis**=**1) **/** (**-**2 **\*** tau **\*\*** 2))

**def** plot\_lr(tau):

domain **=** np**.**linspace(**-**5, 5, num**=**500)

pred **=** [local\_regression(x0, x, y, tau) **for** x0 **in** domain]

plt**.**scatter(x, y, alpha**=**0.3)

plt**.**plot(domain, pred, color**=**"red")

**return** plt

plot\_lr(1)**.**show()

**Output:**

