ML- Lab programs:

1. K-Means:

**def** getSquaredDistance(point1, point2):

**return** round(((point2[0] **-** point1[0])**\*\***2 **+** (point2[1] **-** point1[1])**\*\***2),4)

**def** getDistanceFromPoints(centroids, datapoints):

distance\_from\_cluster **=** [] *# [cluster1\_distances, cluster2\_distances,...]*

intermediate\_result **=** []

**for** i **in** centroids:

**for** j **in** datapoints:

intermediate\_result**.**append(getSquaredDistance(i,j))

distance\_from\_cluster**.**append(intermediate\_result)

intermediate\_result **=** [] *# reset intermediate\_result as empty list*

**return** distance\_from\_cluster

**def** printResult(centroids, point\_to\_cluster\_mapping):

**for** i **in** range(len(centroids)):

print("Centroid",i,centroids[i])

**for** i **in** point\_to\_cluster\_mapping:

print("Point: ",i,"Cluster:",point\_to\_cluster\_mapping[i])

**def** kmeansclustering(centroids, datapoints):

'''

Driver code for K-Means clustering

'''

k **=** len(centroids)

distance\_from\_cluster **=** getDistanceFromPoints(centroids, datapoints) *# [cluster1\_distances, cluster2\_distances,...]*

*# assign each datapoint to the nearest cluster*

point\_to\_cluster\_mapping **=** {} *# point -> cluster*

max\_valued\_cluster **=** 0

**for** i **in** range(len(datapoints)):

point\_to\_cluster\_mapping[i] **=** **None** *# initial mapping as None*

**for** cluster **in** range(len(centroids)):

**if** distance\_from\_cluster[cluster][i] **<** distance\_from\_cluster[max\_valued\_cluster][i]:

max\_valued\_cluster **=** cluster

point\_to\_cluster\_mapping[i] **=** max\_valued\_cluster

*# compute new centroids by averaging with new points*

cluster\_counter **=** 0 *# [cluster1\_new\_elements\_added, cluster2\_new\_elements\_added...]*

**for** i **in** range(len(centroids)):

**for** j **in** point\_to\_cluster\_mapping:

**if** point\_to\_cluster\_mapping[j] **==** i:

centroids[i][0] **+=** datapoints[j][0] *# x-coordinate adding*

centroids[i][1] **+=** datapoints[j][1] *# y-coordinate adding*

cluster\_counter **+=** 1

**if** cluster\_counter **!=** 0:

centroids[i][0] **=** round(centroids[i][0]**/**cluster\_counter,4)

centroids[i][1] **=** round(centroids[i][1]**/**cluster\_counter,4)

cluster\_counter **=** 0

printResult(centroids, point\_to\_cluster\_mapping)

**return** centroids

**def** kmeans\_iterator(centroids, datapoints):

old\_centroids **=** centroids

new\_centroids **=** centroids

iteration **=** 0

**while** iteration **!=** 15:

iteration **+=** 1

print("\nIteration ", iteration)

old\_centroids **=** new\_centroids

new\_centroids **=** kmeansclustering(new\_centroids, datapoints)

centroids **=** [[2,10], [5,8], [1,2]]

datapoints **=** [[2,10], [2,5], [8,4], [5,8], [7,5], [6,4], [1,2], [4,9]]

kmeans\_iterator(centroids, datapoints)

1. **Decision tree:**

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

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

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

df**=** pd**.**read\_csv("/home/admn/Downloads/zoo1.csv")

df**.**head()

class\_type\_output **=** df["class\_type"]

df **=** df**.**drop("class\_type", axis**=**1)**.**drop("animal\_name",axis**=**1)

print(df)

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

x\_train, x\_test, y\_train, y\_test **=** train\_test\_split(df, class\_type\_output, test\_size**=**0.20)

**from** sklearn.tree **import** DecisionTreeClassifier

classifier **=** DecisionTreeClassifier()

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

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

y\_prediction

**from** sklearn.metrics **import** classification\_report, confusion\_matrix, accuracy\_score

confusion\_matrix(y\_test,y\_prediction)

print(classification\_report(y\_test, y\_prediction))

print(accuracy\_score(y\_test, y\_prediction))

predicted\_class **=** list(y\_prediction)

actual\_class **=** list(y\_test)

**for** i **in** range(len(predicted\_class)):

print("Predicted class =", predicted\_class[i],"\tActual class =",actual\_class[i])

1. **Linear regression:**

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

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

df **=** pd**.**read\_csv("/home/admn/Downloads/student\_scores.csv")

df

df**.**plot(x**=**"Hours", y**=**"Scores", style**=**"o")

plt**.**show()

x\_mean **=** df["Hours"]**.**mean()

y\_mean **=** df["Scores"]**.**mean()

print(x\_mean, y\_mean)

df["x"] **=** df["Hours"] **-** x\_mean

df["y"] **=** df["Scores"] **-** y\_mean

df["x\*y"] **=** df["x"] **\*** df["y"]

df["x^2"] **=** df["x"]**\*\***2

df["y^2"] **=** df["y"]**\*\***2

df

summation\_x\_y **=** df["x\*y"]**.**sum()

summation\_x\_squared **=** df["x^2"]**.**sum()

summation\_y\_squared **=** df["y^2"]**.**sum()

print(summation\_x\_y, summation\_x\_squared, summation\_y\_squared)

correlation **=** summation\_x\_y **/** (summation\_x\_squared **\*** summation\_y\_squared)**\*\***0.5

correlation

**def** getMean(numbers):

**if** len(numbers) **==** 0:

**return** **None**

**else**:

current\_sum **=** 0

**for** i **in** numbers:

current\_sum **+=** i

current\_avg **=** current\_sum**/**len(numbers)

**return** current\_avg

**def** getStandardDeviation(numbers):

**if** len(numbers) **==** 0:

**return** 0

**else**:

mean **=** getMean(numbers)

std\_deviation **=** 0

**for** i **in** numbers:

std\_deviation **+=** (i **-** mean)**\*\***2

**return** (std\_deviation**/**len(numbers))**\*\***0.5

std\_deviation\_x **=** getStandardDeviation(df["x"]**.**tolist())

std\_deviation\_y **=** getStandardDeviation(df["y"]**.**tolist())

print(std\_deviation\_x, std\_deviation\_y)

m **=** correlation **\*** (std\_deviation\_y **/** std\_deviation\_x)

m

c **=** df["Scores"]**.**mean() **-** m **\*** df["Hours"]**.**mean()

c

df["y\_prediction"] **=** m **\*** df["Hours"] **+** c

df

plot1 **=** plt**.**scatter(df["Hours"], df["Scores"])

plot2 **=** plt**.**scatter(df["Hours"], df["y\_prediction"])

plt**.**show()

**4.mean, median, mode, standard variation and normalization:**

**def** getMode(numbers):

max\_occur **=** **-**1

**if** len(numbers) **==** 0:

**return** **None**

**else**:

occurences **=** {}

**for** i **in** numbers:

**if** occurences**.**get(i) **==** **None**:

occurences[i] **=** 1

**else**:

occurences[i] **+=** 1

**if** occurences[i] **>** max\_occur:

max\_occur **=** occurences[i]

*# get max occurence number*

**for** i **in** occurences:

**if** occurences[i] **==** max\_occur:

**return** i

**return** **None**

**def** getMean(numbers):

**if** len(numbers) **==** 0:

**return** **None**

**else**:

current\_sum **=** 0

**for** i **in** numbers:

current\_sum **+=** i

current\_avg **=** current\_sum**/**len(numbers)

**return** current\_avg

**def** getMedian():

numbers **=** []

inp **=** 0

**while** **True**:

inp **=** int(input("Enter a number OR type 'exit'"))

**if** inp **==** 'exit':

**break**

**else**:

numbers**.**append(inp)

**if** len(numbers) **==** 0:

**return** **None**

**else**:

middle\_index **=** len(numbers)**//**2

**return** numbers[middle\_index]

**def** getStandardDeviation(numbers):

**if** len(numbers) **==** 0:

**return** 0

**else**:

mean **=** getMean(numbers)

std\_deviation **=** 0

**for** i **in** numbers:

std\_deviation **+=** (i **-** mean)**\*\***2

**return** (std\_deviation**/**len(numbers))**\*\***0.5

**def** getVariance(numbers):

**return** getStandardDeviation(numbers)**\*\***2

**def** getNormalization(features):

x\_min **=** min(features)

x\_max **=** max(features)

normalized\_vals **=** []

**for** i **in** features:

normalized\_vals**.**append((i **-** x\_min)**/**(x\_max **-** x\_min))

**return** normalized\_vals

getNormalization([10,20,30,40])

**def** getStandardization(features):

mean **=** getMean(features)

std\_deviation **=** getStandardDeviation(features)

standardized\_vals **=** []

**for** i **in** features:

standardized\_vals**.**append((i **-** mean)**/**std\_deviation)

**return** standardized\_vals

getStandardization([10,20,30,40])

*# MinMax Normalization*

**def** doMinMaxNormalization(numbers):

result **=** []

**if** len(numbers) **==** 0:

**return** result

**else**:

min\_value **=** min(numbers)

max\_value **=** max(numbers)

**for** i **in** numbers:

result**.**append((i **-** min\_value)**/**(max\_value **-** min\_value))

**return** result

features **=** [100000,**-**2,50,12,700,9000]

print(doMinMaxNormalization(features))