REPORT

The following report consists of solutions to K-Means and K-NN Technique used in Data Mining.

A **Python** program has been developed which performs clustering and classification based on prior knowledge and skill set of NBA Players. A **.csv** sheet comprises of different attributes which define the NBA Player. Every record here can be considered as a point in the 'N' space vector, where 'N' could be as many as the attributes that can be defined. Currently we have twenty-nine attributes defined, among which twenty-six are numeric while the rest are string type.

The program reads the data from this file, normalizes the same and treats them as 'N' space vectors. Not always all 'N' space vectors define the object, thus the program also takes up an **attribute file** as a command-line parameter to gain more knowledge on the space vectors. Thus allowing **attribute redundancy features**. With the help of this, a target space vector can be visualized, and data mining techniques can be applied, generating better and faster results.

Every data mining problem requires some training data or the sampling data. The program has been implemented where a user can give the **sampling data counter** from the command line itself, and the rest would be considered as the test dataset.

No use on NumPy or other libraries have been used for normalizing the attributes.

To find the mean for each attribute the formula used is $\mu = (\sum x)/N$, where 'x' is the attribute value and 'N' is the number of records in training set.

To find the standard deviation $\sigma = \sqrt{\frac{\sum(x-\mu)^2}{N}}$

To standardize the attribute values, $normal\_attribute = \frac{x-\mu}{\sigma}$ is used.

To calculate the Euclidean distance in N-vector space,
$distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2 + \cdots}$ has been used

To run the program, we could use the following for our command line:

For K-Means Clustering:

$$\$\ python\ main.py\ kmeans\ [number - clusters]\ [attribute - file]\ [sampling - set]$$

For K-NN Classification

$$\$\ python\ main.py\ knn\ [top - k - number]\ [attribute - file]\ [sampling - set]$$

PROBLEM 1

The dataset provided for this problem NBAstats.csv is stats from NBA players. The players are indexed by their names, and they are labeled by 5 different positions: {center (C), power forward (PF), small forward (SF), shooting guard (SG), point guard (PG)} and there are 27 attributes, e.g., age, team, games, games started, minutes played and so on (that makes total of 29 columns in the data matrix). Make sure that you standardize the data (zero-mean and standard deviation = 1) before you analyze the data.

1. *Write a function cluster = mykmeans(X, k) that clusters data $X \in R^{n \times p}$ (n number of objects and p number of attributes) into k clusters*

```
function mykmeans(X, k):
  # X – Sample / Training Data
  # k – Number of clusters to be generated
  clusters = RandomInitialization(k, attributes)
  return iterateKmeans(X, clusters)

function iterateKmeans(X, clusters)
  Assign change_flag <- False
  For player in X:
  Do:
     Assign small <- MAX(FLOAT)
     Assign myCluster <- None
     For cluster in clusters:
     Do:
        Temp <- Euclidean_Distance(player, cluster)
        If Temp < small
        Then:
            Small <- Temp
            myCluster = cluster
     Endfor
     myCluster.clusterData.add(player)
  EndFor

  For cluster in clusters:
  Do:
     For player in cluster.clusterData:
     Do:
         Find Sum of all attributes
     EndFor
     C <- Find mean for the sum generated
     If C != cluster.center
        change_Flag <- True
```

```
            cluster.center = C
        endif
    endfor
    if change_Flag = True:
    then:
        iterateKmeans(X, clusters)
    else:
        return clusters

function RandomInitialization
  – generates 'k' random points as initial cluster points based on attributes
```

The following has been implemented in **main.py**.

mykmeans function initializes the cluster points. Currently, the cluster points are randomly generated. For every cluster, and for its standardized attribute value, a random number is generated in the range [0, 1] and is used as the cluster point.

To handle the iterations, iterateKmeans function is used. This helps us to find the points of the clusters and iterate until a proper boundary is not formed, contributing to the goal of clustering.

2. *For this problem, use all features except team. Use your code to group the players into k = {3, 5} clusters. Report the centers found for each clusters for each k, distribution of positions in each cluster and your brief observation.*

Here training set is 375 and k = 3

Attributes	Cluster 1	Cluster 2	Cluster 3
Total Points	164	132	79
Centers based on attributes			
Age	0.11	-0.07	-0.11
G	0.48	-0.95	0.59
GS	0.01	-0.77	1.28
MP	0.22	-1.07	1.32
FG	0.03	-0.94	1.5
FGA	0.02	-0.92	1.49
FG%	0.14	-0.28	0.17
3P	0.09	-0.56	0.75
3PA	0.08	-0.57	0.78
3P%	0.1	-0.26	0.24
2P	0	-0.87	1.46
2PA	-0.02	-0.87	1.48
2P%	0.11	-0.19	0.1
eFG%	0.04	-0.08	0.05
FT	-0.1	-0.71	1.4

FTA	-0.1	-0.71	1.39
FT%	0.14	-0.34	0.28
ORB	0.14	-0.56	0.65
DRB	0.1	-0.83	1.17
TRB	0.13	-0.79	1.07
AST	-0.01	-0.65	1.1
SLK	0.05	-0.74	1.12
BLK	0.04	-0.5	0.76
TOV	0	-0.81	1.36
PF	0.33	-0.95	0.9
PS/G	0.01	-0.91	1.5
Distribution of Positions			
C	31	25	14
PF	35	31	17
PG	29	23	15
SF	33	28	16
SG	36	25	17

Observation:

- From the 3 clusters generated, we can infer that the clusters are based on 3 spaces. Cluster 2 whose centroids are mostly in negative, Cluster 1 whose centroid has both +ve and -ve and Cluster 3 whose centroids are towards +ve.
- Players belonging to Cluster 3 are the ones above average and performers. (Note: Only if positive values indicate better performance).
- We could say that the clustering was based on overall performance of the player and not on their positions.

Here training set is 375 and k = 5

Attributes	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Total Points	89	127	58	53	48
Centers based on attributes					
Age	-0.32	0.15	-0.12	-0.04	0.39
G	-1.04	0.46	0.58	0.51	-0.55
GS	-0.82	-0.09	1.26	0.74	-0.55
MP	-1.11	0.25	1.39	0.6	-0.92
FG	-1	0.01	1.59	0.58	-0.74
FGA	-0.92	0.07	1.68	0.35	-0.88
FG%	-0.85	0.03	0.09	0.43	0.91
3P	-0.5	0.41	1.26	-0.78	-0.85
3PA	-0.48	0.42	1.31	-0.81	-0.9
3P%	-0.03	0.48	0.51	-0.74	-1.01
2P	-0.97	-0.16	1.35	1	-0.52
2PA	-0.92	-0.14	1.43	0.88	-0.64
2P%	-0.77	0.03	0.09	0.23	0.98

eFG%	-0.55	-0.02	0.01	0.18	0.85
FT	-0.73	-0.17	1.63	0.4	-0.59
FTA	-0.74	-0.21	1.48	0.66	-0.56
FT%	-0.3	0.31	0.51	-0.39	-0.44
ORB	-0.7	-0.31	0.1	1.81	0.01
DRB	-0.89	-0.14	0.89	1.31	-0.5
TRB	-0.89	-0.19	0.68	1.54	-0.36
AST	-0.65	0.12	1.5	-0.14	-0.76
SLK	-0.73	0.14	1.4	0.07	-0.78
BLK	-0.63	-0.34	0.24	1.63	-0.01
TOV	-0.85	0	1.58	0.38	-0.75
PF	-1.05	0.15	0.77	1.06	-0.55
PS/G	-0.95	0.02	1.68	0.42	-0.78
Distribution of Positions					
C	7	3	2	34	24
PF	20	23	9	15	16
PG	16	33	15	0	3
SF	22	32	15	4	4
SG	24	36	17	0	1

Observation:

- From the above table, it can be inferred that Cluster 4 and Cluster 5 are more inclined towards 'C' players.
- Cluster 2 has most players mapped to it, which could say that it would be near to center of the vector space.

3. *Some of the attributes are perhaps redundant in terms of Linear Algebra. Report which ones are redundant and explain why.*

Attributes can be defined redundant only when they can be derived by set of other attributes. We have the following attributes redundant here:

- FG and FGA:
  We can see that FG% is defined as FG/FGA
- 3P and 3PA:
  We can see that 3P% is defined as 3P/3PA
- 2P and 2PA:
  We can see that 2P% is defined as 2P/2PA
- FT and FTA:
  We can see that FT% is defined as FT/FTA

4. *For this problem, use the following set of attributes {2P%, 3P%, FT%, TRB, AST, STL, BLK} to perform k-means clustering with k = {3, 5}. Report the centers found for each clusters for each k, distribution of positions in each cluster and your brief observation.*

Here training set is 375 and k = 3

Attributes	Cluster 1	Cluster 2	Cluster 3
Total Points	93	86	196
Centers based on attributes			
3P%	0.26	-1.37	0.48
2P%	0.14	0.19	-0.15
FT%	0.16	-0.7	0.23
TRB	1.06	-0.13	-0.45
AST	1	-0.7	-0.17
SLK	1.16	-0.7	-0.24
BLK	0.79	0.22	-0.47
Distribution of Positions			
C	19	41	10
PF	19	29	35
PG	20	5	42
SF	21	7	49
SG	14	4	60

Observation:

- Cluster 3 has major portions of the players mapped and can be taken close to the center of the vector space.
- Cluster 2 has been mapped to most of the 'C' position players boundary.

Here training set is 375 and k = 5

Attributes	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Total Points	41	30	114	48	142
Centers based on attributes					
3P%	0.55	-0.7	0.51	-0.95	-0.1
2P%	0.11	-0.99	0.04	0.29	0.05
FT%	0.51	-2.4	0.21	-0.46	0.35
TRB	0.41	-0.8	0.07	1.65	-0.57
AST	2.18	-0.91	0.18	-0.1	-0.55
SLK	1.52	-0.94	0.52	0.21	-0.73
BLK	0.03	-0.49	-0.23	2.05	-0.42
Distribution of Positions					
C	1	7	4	34	24
PF	2	11	23	11	36
PG	21	2	22	1	21
SF	8	7	30	2	30
SG	9	3	35	0	31

Observation:

- Cluster 5 has major portions of the players mapped and can be taken close to the center of the vector space.
- Cluster 4 has been mapped to most of the 'C' position players boundary.

- Cluster 3 handles the other part of the cluster 5, while cluster 1 and 2 are in near corners of the vector space.
- Cluster 1 shares a portion of 'PG' positioned players' boundary.

PROBLEM 2

1. *Write a function class = myknn(X, test, k) that performs k-nearest neighbor (k-NN) classification where X ∈ R n×p (n number of objects and p number of attributes) is training data, test is testing data, and k is a user parameter.*

```
Function Distcance
function myknn(X, test, k) :
  # X    – Array of sample/training data
  # test – Array of test data
  # k    - Top K nearest data sets to compare with
  qccuracy_points <- 0
  Assign distance <- MAX(FLOAT)
  knn_nodes[k]
  foreach player in X:
  do:
      player_distance <- Euclidean_Distance(player, test_player)
      if player_distance > distance:
      then:
          if length_of(knn_nodes) >= k:
          then:
             Replace player with max distance player from knn_nodes
          else:
             Add player to knn_nodes
          endif
      endif
  endfor
  foreach knn_player in knn_nodes:
  do:
       switch knn_player.pos:
         case "C": C_counter++
                   break
         case "PF": PF_counter++
                   break
         case "PG": PG_counter++
                   break
         case "SF": SF_counter++
                   break
         case "SG": SG_counter++
                   break
  endfor
  if C_counter > MAX(PF_counter, PG_counter, SF_counter, SG_counter):
       return 'C'
  if PF_counter > MAX(C_counter, PG_counter, SF_counter, SG_counter):
       return 'PF'
```

```
if PG_counter > MAX(PF_counter, C_counter, SF_counter, SG_counter):
     return 'PG'
if SF_counter > MAX(PF_counter, PG_counter, C_counter, SG_counter):
     return 'SF'
return 'SG'
```

2. *For this problem, use all features except team. Use your k-NN code to perform classification. Set k = {1, 5, 10, 30} and report their accuracies and your observation.*

k	1	5	10	30	40
Accuracy	0.49	0.54	0.53	0.54	0.51

Observation:

- The program has a training dataset of 375 and testing dataset is of size 100.
- It is observed that with increasing value of 'k', the accuracy of the program increases. But after a certain threshold the accuracy drops drastically.
- Technique like gradient descent can be used to find the best value for 'k' to generate improved accuracy.

3. *For this problem, use the following set of attributes {2P%, 3P%, FT%, TRB, AST, STL, BLK} to perform k-NN classification with k = {1, 5, 10, 30}. Report accuracies for each k and your observation*

k	1	5	10	30	40
Accuracy	0.55	0.58	0.58	0.65	0.57

Observation:

- The program has a training dataset of 375 and testing dataset is of size 100.
- It is observed that with increasing value of 'k', the accuracy of the program increases. But after a certain threshold the accuracy drops drastically.
- Technique like gradient descent can be used to find the best value for 'k' to generate improved accuracy.
- By comparing the two tables, it can be inferred that Attribute redundancy generate better results in Data Mining algorithms.