COMP337 - CA Assignment 1
Data Clustering
Implementing the k-means and k-medians clustering algorithms

Name: Sami Idreesi
Student ID: 201348278

1. **(25 marks)** Implement the k-means clustering algorithm to cluster the instances into k clusters.

   First I defined a function for computing the squared Euclidean distance between an object and a cluster centre (centroid)

```python
#Returns the squared euclidean distance between X and Y
#For the k-means algorithm
def sqrdEuclidDistance(X,Y):
    #Return the Euclidean distance between X and Y
    return np.linalg.norm(X-Y)**2
```

   Then I defined an assign function that assigns objects in the dataset to a cluster using the squared Euclidean distance function.

```python
#Given dataset and indices of centroids, the function updates the clusters of the objects in the dataset
#based on the algorithm passed as a parameter
def assign(algorithm, centroids, dataset):
    numOfObjects = len(dataset)
    k = len(centroids)

    #Go through every object in the dataset
    for i in range(numOfObjects):
        #Current object
        X = dataset[i][2]
        centroidIndOfX = -1
        distanceToClosestCentroid = np.Inf
        #Find the closest centroid for the current object
        #by going through all the centroids and finding
        #the centroid that gives the smallest distance
        for j in range(k):
            currentCentroid = centroids[j]
            #Computed the squared euclidean distance if the algorithm
            #passed is the k-means algorithm
            if(algorithm == "k-means"):
                dist = sqrdEuclidDistance(X, currentCentroid)
            #Computed the squared l1 distance if the algorithm
            #passed is the k-medians algorithm
            elif(algorithm == "k-medians"):
                dist = l1distance(X, currentCentroid)

            #Check if the distance is smaller
            if dist < distanceToClosestCentroid:
                #Found closer centroid. Store information about it
                distanceToClosestCentroid = dist
                centroidIndOfX = j

        #assign to object its closest centroid
        dataset[i][1] = centroidIndOfX
```

I also defined a helper function to retrieve all the objects in a cluster using the clusters centre (centroid) for use when I compute new cluster centres.

```
#Function to filter the dataset given a centroid
def filterByCentroid(dataset, currCentroid):
    filteredData = []
    for x in dataset:
        if(currCentroid == x[1]):
            filteredData.append(x)

    return filteredData
```

Using the functions already defined, next I actually implemented k-means clustering. The stopping condition for the algorithm is the parameter maxIter - so the algorithm will perform the clustering for the number of iteration defined in maxIter.

```
#Implementation of both the clustering algorithms: k-means and k-medians
#dependant on the parameter: algorithm
def clusteringAlgorithm(algorithm, k, dataset, maxIter=10):
    #Generate indices for initial centroids
    seed = random.randint(0, len(dataset))
    ### Enter own seed for testing! ###
    np.random.seed(10)
    centroidInds = np.random.choice(len(dataset), k, replace=False)
    centroidInds = np.asarray(centroidInds, dtype=np.float64, order='C')

    #Use generated indices to retrieve the objects that correspond to them in the dataset
    centroids = []
    for k in range(len(centroidInds)):
        for i in range(len(dataset)):
            if(int(centroidInds[k]) == i):
                centroids.append(dataset[i][2])

    #Make initial assignment of objects to the clusters
    assign(algorithm, centroids, dataset)

    #Repeat algorithm for maxIter iterations
    for i in range(maxIter):
        #Go through every cluster and compute new centroid
        #for the cluster based on the objects in the cluster
        for y in range(len(centroids)):
            currObjsInCentroid = filterByCentroid(dataset, y)
            #Retrieve just the features from the objects
            currObjsInCentroid = [features for (fname, centroid, features) in currObjsInCentroid]

            newCentroid = 0
            #Compute new centroid based on algorithm
            if(algorithm == "k-means"):
                newCentroid = np.mean(currObjsInCentroid)
            elif(algorithm == "k-medians"):
                newCentroid = np.median(currObjsInCentroid)
            centroids[y] = newCentroid
        #Update assignment of objects to clusters
        assign(algorithm, centroids, dataset)


    return (dataset, centroids)
```

2. **(25 marks)** Implement the k-medians clustering algorithm to cluster the instances into k clusters.

   The implementation for k-medians uses the same functions described in the previous question, except for the distance function.

```
#Returns the l1/Manhattan distance between objects X and Y
#For the k-medians algorithm
def l1distance(X,Y):

    return np.linalg.norm(X-Y, ord=1)
```

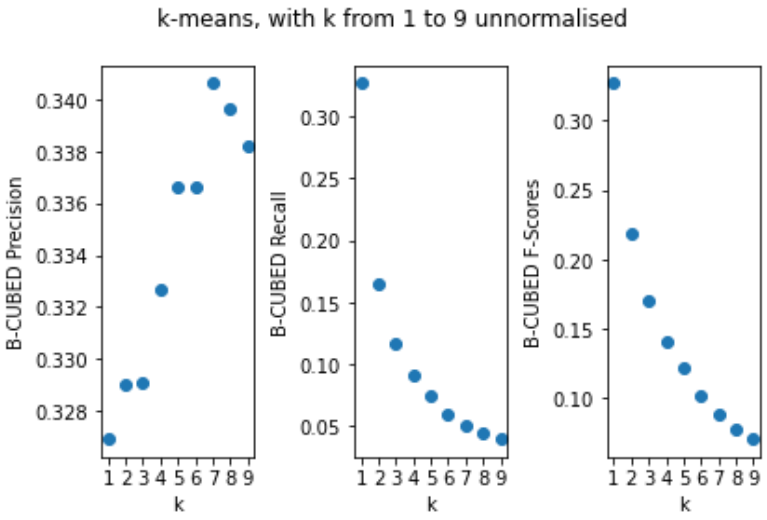
3. **(10 marks)** Run the k-means clustering algorithm you implemented in part (1) to cluster the given instances. Vary the value of k from 1 to 9 and compute the B-CUBED precision, recall, and F-score for each set of clusters. Plot k in the horizontal axis and the B-CUBED precision, recall and F-score in the vertical axis in the same plot.

For questions 3-6 outputs - I set my initial seed at 10, when randomly picking the initial centroids so the initial centroids are the same for each question and I can compare fairly as opposed to each question having different initial cluster centres.

Output from console for Question 3
Seed to generate initial centroids set as 10

```
Running:  k-means with k from 1 to 9  unnormalised
k:  1
B_CUBED Precision:  0.32690939662419977
B_CUBED Recall:  0.32690939662419977
B_CUBED F-Score:  0.32690939662419977
k:  2
B_CUBED Precision:  0.32903679510657174
B_CUBED Recall:  0.16449404569433027
B_CUBED F-Score:  0.21931414802318028
k:  3
B_CUBED Precision:  0.32907618705725933
B_CUBED Recall:  0.11643462273999686
B_CUBED F-Score:  0.17011919536181389
k:  4
B_CUBED Precision:  0.3326799909702846
B_CUBED Recall:  0.0906218530870927
B_CUBED F-Score:  0.14070951561763198
k:  5
B_CUBED Precision:  0.33659664441943926
B_CUBED Recall:  0.07537809148104692
B_CUBED F-Score:  0.1215789868147658
k:  6
B_CUBED Precision:  0.3366514282996414
B_CUBED Recall:  0.0602636708825676
B_CUBED F-Score:  0.1013334668158903
k:  7
B_CUBED Precision:  0.3406300257106015
B_CUBED Recall:  0.05071091360944559
B_CUBED F-Score:  0.08778978466822858
k:  8
B_CUBED Precision:  0.33962278185837186
B_CUBED Recall:  0.044188431370737524
B_CUBED F-Score:  0.0778159871799642 5
k:  9
B_CUBED Precision:  0.33821934422791816
B_CUBED Recall:  0.03971692796629743
B_CUBED F-Score:  0.07066695478002183
```

Resulting plot for Question 3

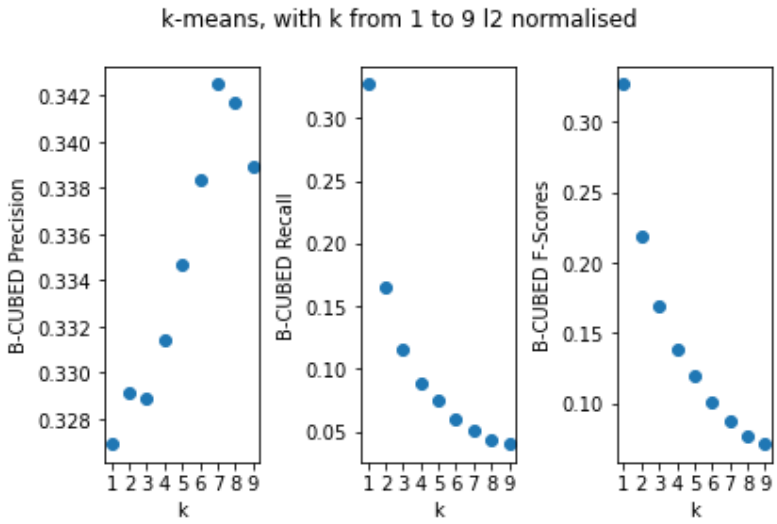


4. **(10 marks)** Now re-run the k-means clustering algorithm you implemented in part (1) but normalise each object (vector) to unit l2 length before clustering. Vary the value of k from 1 to 9 and compute the B-CUBED precision, recall, and F-score for each set of clusters. Plot k in the horizontal axis and the B-CUBED precision, recall and F-score in the vertical axis in the same plot.

Output from console for Question 4
Seed to generate initial centroids set as 10

```
Running:  k-means with k from 1 to 9  l2 normalised
k:  1
B_CUBED Precision:  0.32690939662419977
B_CUBED Recall:  0.32690939662419977
B_CUBED F-Score:  0.32690939662419977
k:  2
B_CUBED Precision:  0.32915683886083513
B_CUBED Recall:  0.16453100026792064
B_CUBED F-Score:  0.2193844780118014
k:  3
B_CUBED Precision:  0.32888567371325994
B_CUBED Recall:  0.11517816723792279
B_CUBED F-Score:  0.16906733967758836
k:  4
B_CUBED Precision:  0.33143858239453083
B_CUBED Recall:  0.08918062471706655
B_CUBED F-Score:  0.13908694097814417
k:  5
B_CUBED Precision:  0.33472333577934177
B_CUBED Recall:  0.07432488613372011
B_CUBED F-Score:  0.1201337715217828
k:  6
B_CUBED Precision:  0.33836844428899676
B_CUBED Recall:  0.059783261425892215
B_CUBED F-Score:  0.10078994854697464
k:  7
B_CUBED Precision:  0.3424979055678144
B_CUBED Recall:  0.05061852717546955
B_CUBED F-Score:  0.08781721036365234
k:  8
B_CUBED Precision:  0.34171660856446395
B_CUBED Recall:  0.04354172633290526
B_CUBED F-Score:  0.07696871100286379
k:  9
B_CUBED Precision:  0.3389594522518037
B_CUBED Recall:  0.04001256455502073
B_CUBED F-Score:  0.07112546153623485
```

Resulting plot for Question 4

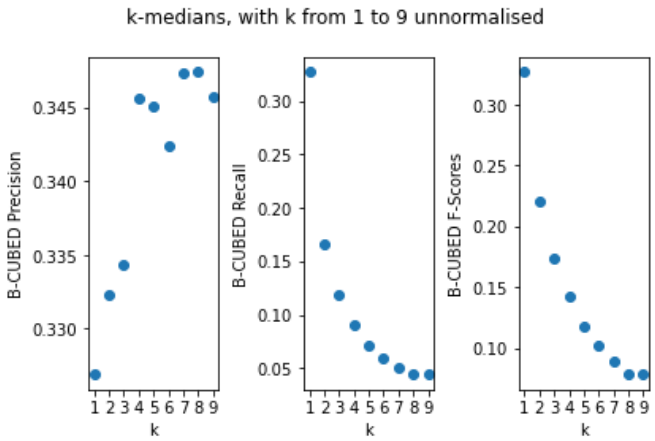


5. **(10 marks)** Run the k-medians clustering algorithm you implemented in part (2) over the unnormalised objects. Vary the value of k from 1 to 9 and compute the B-CUBED precision, recall, and F-score for each set of clusters. Plot k in the horizontal axis and the B-CUBED precision, recall and F-score in the vertical axis in the same plot.

Output from console for Question 5
Seed to generate initial centroids set as 10

```
Running:  k-medians with k from 1 to 9  unnormalised
k:  1
B_CUBED Precision:  0.32690939662419977
B_CUBED Recall:  0.32690939662419977
B_CUBED F-Score:  0.32690939662419977
k:  2
B_CUBED Precision:  0.3323289664028086
B_CUBED Recall:  0.16576897848319955
B_CUBED F-Score:  0.22096014467191025
k:  3
B_CUBED Precision:  0.33432340795077653
B_CUBED Recall:  0.11902144289132584
B_CUBED F-Score:  0.17324611336274773
k:  4
B_CUBED Precision:  0.34560518549880254
B_CUBED Recall:  0.09086205781543037
B_CUBED F-Score:  0.14177513709542683
k:  5
B_CUBED Precision:  0.34504108917090826
B_CUBED Recall:  0.07164567954841511
B_CUBED F-Score:  0.11751805951932975
k:  6
B_CUBED Precision:  0.3423851588549765
B_CUBED Recall:  0.06007889801461552
B_CUBED F-Score:  0.10143523855783412
k:  7
B_CUBED Precision:  0.3473184982777664
B_CUBED Recall:  0.051228277639711384
B_CUBED F-Score:  0.08890410482585795
k:  8
B_CUBED Precision:  0.34741657931404046
B_CUBED Recall:  0.04452102253305124
B_CUBED F-Score:  0.0786462007657068
k:  9
B_CUBED Precision:  0.3457557477260039
B_CUBED Recall:  0.04424386323112314
B_CUBED F-Score:  0.07818461182044602
```

Resulting plot for Question 4

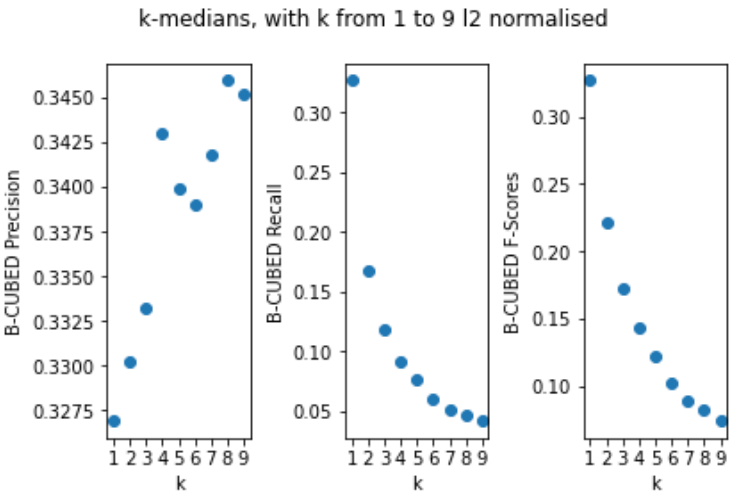


6. **(10 marks)** Now re-run the k-medians clustering algorithm you implemented in part (2) but normalise each object (vector) to unit l2 length before clustering. Vary the value of k from 1 to 9 and compute the B-CUBED precision, recall, and F-score for each set of clusters. Plot k in the horizontal axis and the B-CUBED precision, recall and F-score in the vertical axis in the same plot.

Output from console for Question 5
Seed to generate initial centroids set as 10

```
Running:  k-medians with k from 1 to 9  l2 normalised
k:  1
B_CUBED Precision:  0.32690939662419977
B_CUBED Recall:  0.32690939662419977
B_CUBED F-Score:  0.32690939662419977
k:  2
B_CUBED Precision:  0.33017642044997664
B_CUBED Recall:  0.1672286841400209
B_CUBED F-Score:  0.22093724498115913
k:  3
B_CUBED Precision:  0.3331950653319357
B_CUBED Recall:  0.11865189715542171
B_CUBED F-Score:  0.17267610808644246
k:  4
B_CUBED Precision:  0.3429231167974365
B_CUBED Recall:  0.09182287672878116
B_CUBED F-Score:  0.1426204590780853
k:  5
B_CUBED Precision:  0.3399088415805741
B_CUBED Recall:  0.07654216054914496
B_CUBED F-Score:  0.12250244696350834
k:  6
B_CUBED Precision:  0.33897736228424186
B_CUBED Recall:  0.06070712576565257
B_CUBED F-Score:  0.10214483657267441
k:  7
B_CUBED Precision:  0.3417979527730969
B_CUBED Recall:  0.051394573220868246
B_CUBED F-Score:  0.08895383878299655
k:  8
B_CUBED Precision:  0.3459180684983672
B_CUBED Recall:  0.04651656950693359
B_CUBED F-Score:  0.08150054625538083
k:  9
B_CUBED Precision:  0.3451481553022272
B_CUBED Recall:  0.04173095222697499
B_CUBED F-Score:  0.07386608147965909
```

Resulting plot for Question 4



7. **(10 marks)** Comparing the different clustering's you obtained in (3)-(6), discuss in which setting you obtained best clustering for this dataset.

All the clustering's follow a similar trend. As k increases the Precision increases, Recall decreases and the F-Score also decreases. The best value of k for the majority of clustering's would be in the range of 4-6. This is where we have the higher values of precision but it also provides a good compromise for the Recall and F-Score as with higher k's in the range 7 to 9 the Recall and F-Score become very low. Moreover, running the algorithm on objects normalised to l2 length didn't have much of an effect on the Recall and F-Score compared to the unnormalized data but only some differences in the Precision.

The best clustering setting I obtained, when testing with different seeds that gave different initial different cluster centres, was the k-medians algorithm on unnormalized objects with k as 4. On average this gave significant improvement compared to the k-means algorithm and slight improvement over the normalised k-medians algorithm.