COMP 527: Course work 2

Name: Orion Assefaw

Student ID: 201530497

Email: o.assefaw@liverpool.ac.uk

Date: 07/05/2021

1. The full K-Means clustering algorithm implementation was done in python, and a separate "clustering.py" file can be referred for more details on how it was implemented. The following is a copy of the code obtained from that separately submitted python file:

```
class KMeans(object):

    def __init__(self, dataset, K = 4, maximumIter = 150):
        self.X = dataset
        self.K = K
        self.maxIterations = maximumIter
        self.centroids = []
        for i in range(K):
            emptyClustersList = []
        self.clusters = [emptyClustersList]

    def fit(self):
        indices = np.random.choice(self.X.shape[0], self.K, replace=False)
        self.centroids = np.array(list([self.X[indices_index] for indices_index in indices]))


        for i in range(self.maxIterations):
            self.clusters = self.clustersMaker(self.centroids)[0]
            cprior = self.centroids
            self.centroids = self.formCenters(self.clusters)
            if self.movementChecker(self.centroids, cprior):
                break

        return self.centroids, self.clustersMaker(self.centroids)[1]

    def formCenters(self, clusters):
        centroids = np.empty((self.K, self.X.shape[1]))
        for clusterIndex, cluster in enumerate(clusters):
            cMean = np.mean(self.X[cluster], axis = 0)
```

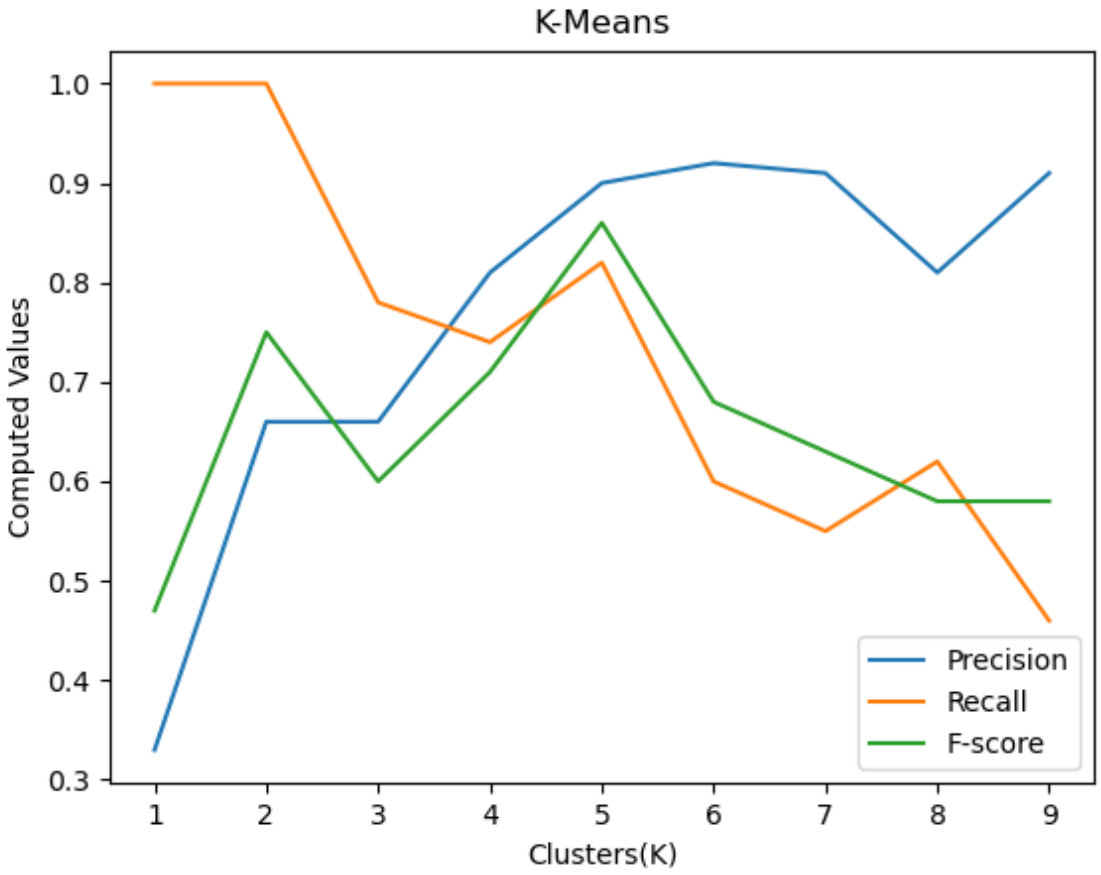
```
            centroids[clusterIndex] = cMean

        return centroids

    def clustersMaker(self,centroids):
        clusters = []
        for i in range(self.K):
            emptyClusters = []
            clusters.append(emptyClusters)

        for objIndex, objects in enumerate(self.X):
            centroidIndices = self.shortestDist(objects, centroids)
            clusters[centroidIndices].append(objIndex)
        clusterID = np.zeros(self.X.shape[0])

        for clusterIndex, cluster in enumerate(clusters):
            for objIndex in cluster:
                clusterID[objIndex] = clusterIndex

        return clusters, clusterID

    def movementChecker(self, centroids, cprior):
        distances = []
        for i in range(self.K):
           dist = self.eucDist(centroids[i],cprior[i])
           distances.append(dist)

        if sum(distances) == 0:
            return True
        else:
            return False

    def shortestDist(self, objects, centroids):

        distancesList = []
        for c in centroids:
            dist = self.eucDist(c,objects)
            distancesList.append(dist)
        indexOfNearest = np.argmin(distancesList)

        return indexOfNearest#, mydict

    def eucDist(self, X, Y):
        return np.sqrt(np.sum((X - Y)**2))
```

2. The full K-Medians clustering algorithm implementation was done in python, and a separate "clustering.py" file can be referred for more details on how it was implemented. The following is a copy of the code obtained from that separately submitted python file:

```
class KMedians(object):

    def __init__(self, dataset, K = 4, maximumIter = 150):
        self.X = dataset
        self.K = K
        self.maxIterations = maximumIter
        self.centroids = []
        for i in range(K):
            emptyClustersList = []
        self.clusters = [emptyClustersList]

    def fit(self):
        indices = np.random.choice(self.X.shape[0], self.K, replace=False)
        self.centroids = np.array(list([self.X[indices_index] for indices_index in indices]))

        for i in range(self.maxIterations):
            self.clusters = self.clustersMaker(self.centroids)[0]
            cprior = self.centroids
            self.centroids = self.formCenters(self.clusters)
            if self.movementChecker(self.centroids, cprior):
                break

        return self.centroids, self.clustersMaker(self.centroids)[1]

    def formCenters(self, clusters):
        centroids = np.empty((self.K, self.X.shape[1]))
        for clusterIndex, cluster in enumerate(clusters):
            cMedian = np.median(self.X[cluster], axis=0)
            centroids[clusterIndex] = cMedian

        return centroids

    def clustersMaker(self,centroids):
        clusters = []
        for i in range(self.K):
            emptyClusters = []
            clusters.append(emptyClusters)

        for objIndex, objects in enumerate(self.X):
            centroidIndices = self.shortestDist(objects, centroids)
            clusters[centroidIndices].append(objIndex)

        clusterID = np.zeros(self.X.shape[0])
        for clusterIndex, cluster in enumerate(clusters):
            for objIndex in cluster:
                clusterID[objIndex] = clusterIndex

        return clusters, clusterID,

    def movementChecker(self, centroids, cprior):
```

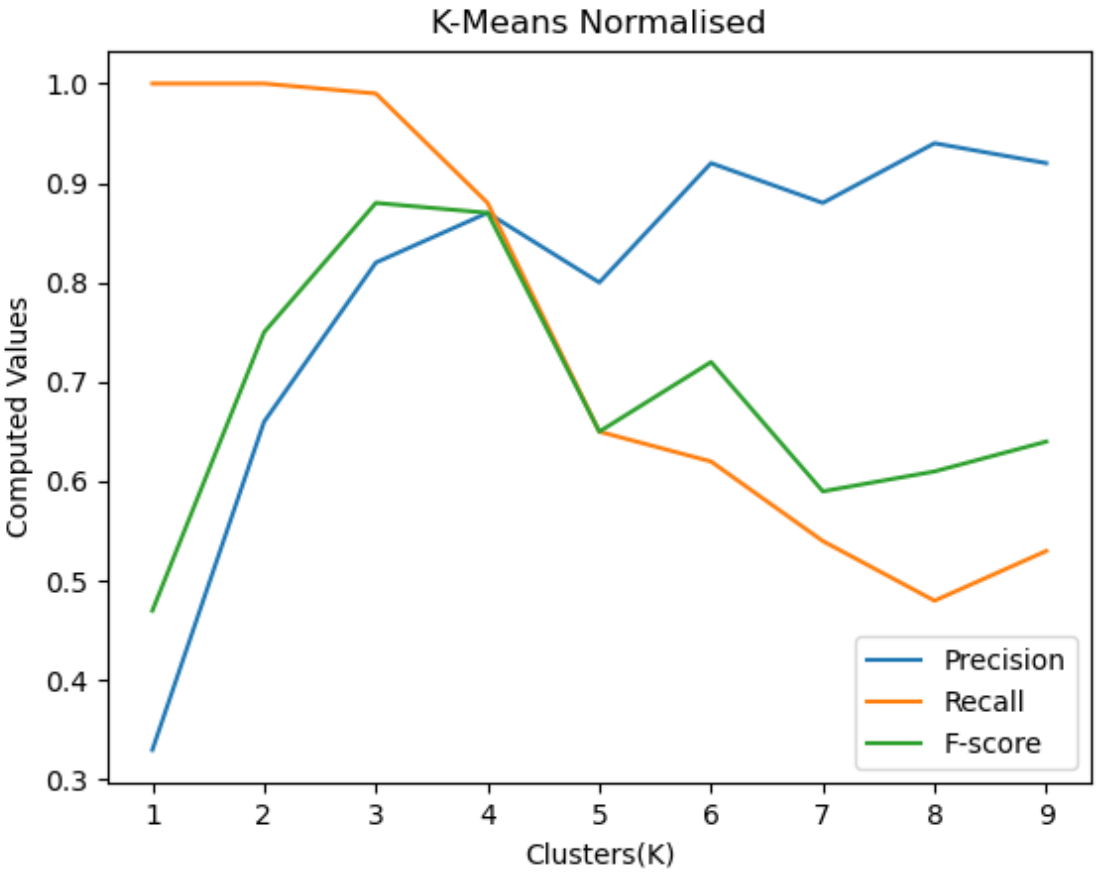
```
        distances = []
        for i in range(self.K):
           dist = self.manDist(centroids[i],cprior[i])
           distances.append(dist)

        if sum(distances) == 0:
            return True
        else:
            return False

    def shortestDist(self, objects, centroids):
        distancesList = []
        for c in centroids:
            dist = self.manDist(c,objects)
            distancesList.append(dist)
        indexOfNearest = np.argmin(distancesList)
        return indexOfNearest

    def manDist(self, X, Y):
        return np.abs(X - Y).sum()
```

3. The K-Means clustering algorithm was run on an unnormalized data and the B-cubed metric scores were computed as well. Results obtained after running the code were as follows:



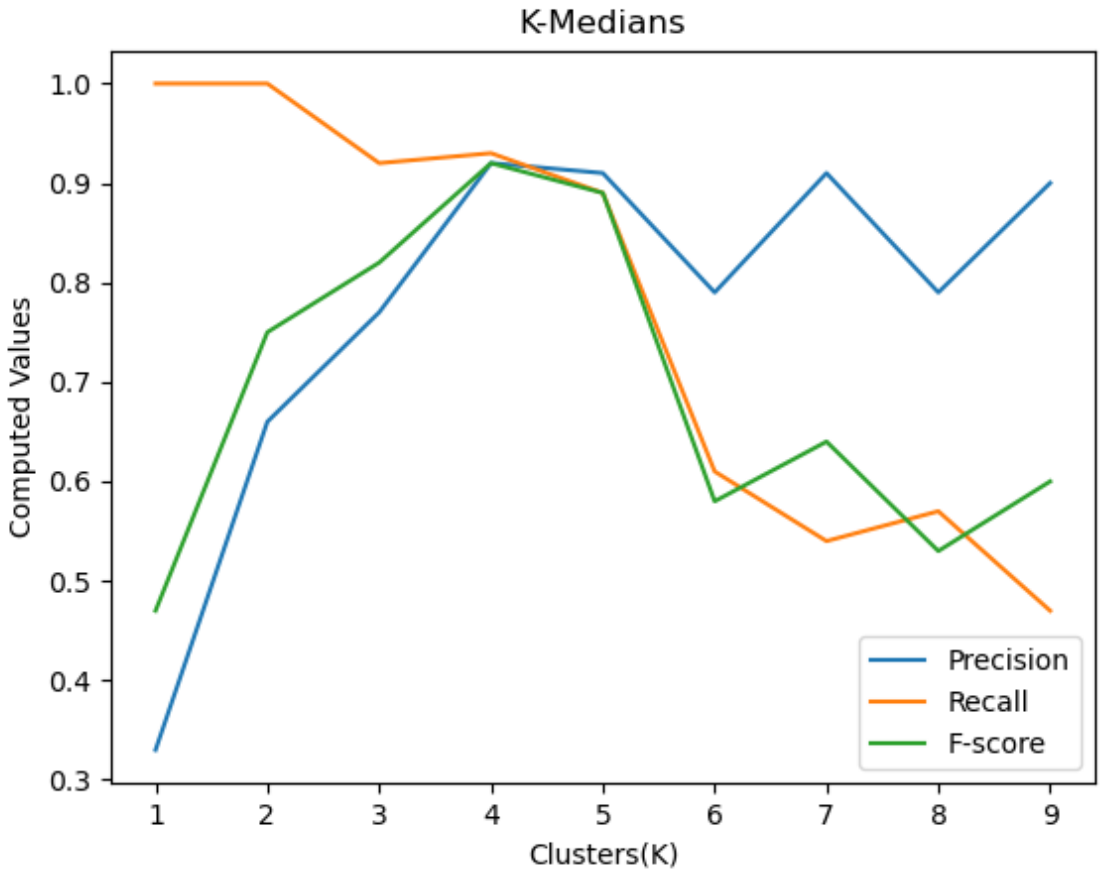

```
K-Means precision List for K = 1 to 9, respectively: [0.33, 0.66, 0.66, 0.81, 0.9, 0.92, 0.91, 0.81, 0.91]
K-Means Recall List for K = 1 to 9, respectively: [1.0, 1.0, 0.78, 0.74, 0.82, 0.6, 0.55, 0.62, 0.46]
K-Means F-score List for K = 1 to 9 respectively: [0.47, 0.75, 0.6, 0.71, 0.86, 0.68, 0.63, 0.58, 0.58]
```

4. The K-Means clustering algorithm was run on a normalized data and the B-cubed metric scores were computed as well. Results obtained after running the code were as follows:



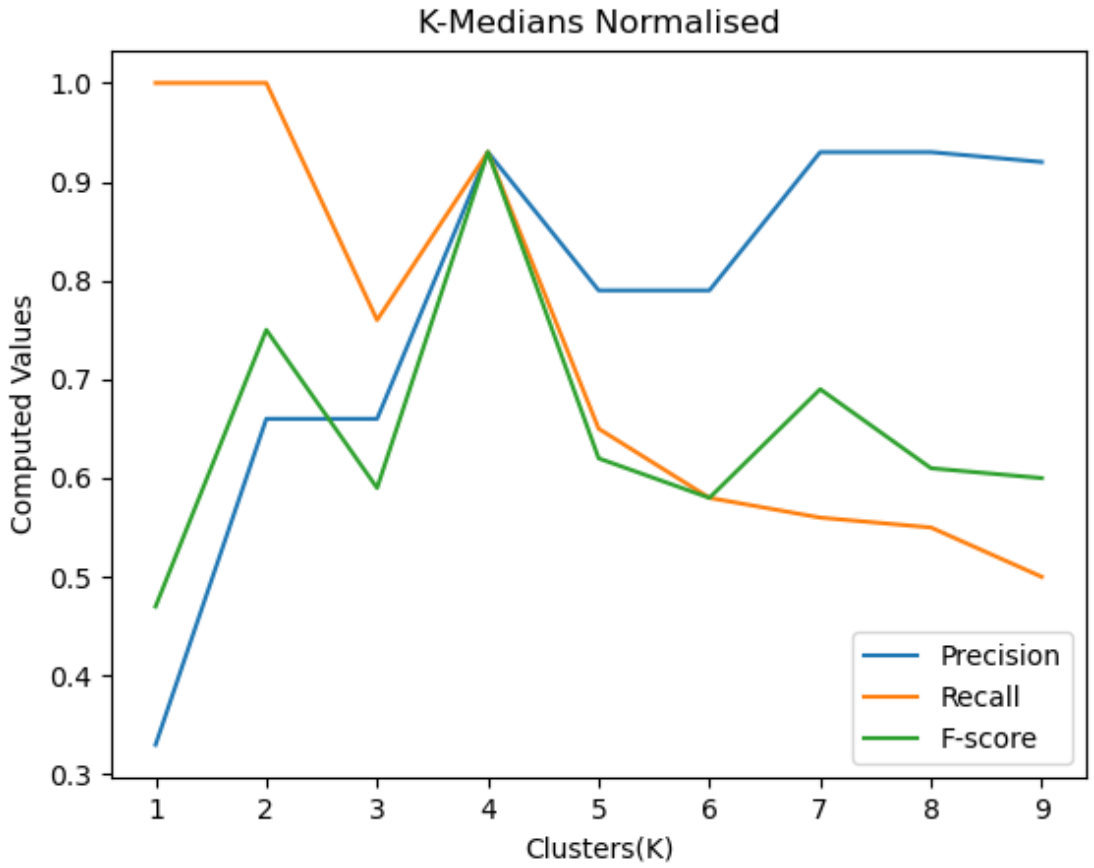

```
K-Means Normalised precision List for K = 1 to 9, respectively: [0.33, 0.66, 0.82, 0.87, 0.8, 0.92, 0.88, 0.94, 0.92]
K-Means Normalised Recall List for K = 1 to 9, respectively: [1.0, 1.0, 0.99, 0.88, 0.65, 0.62, 0.54, 0.48, 0.53]
K-Means Normalised F-score List for K = 1 to 9 respectively: [0.47, 0.75, 0.88, 0.87, 0.65, 0.72, 0.59, 0.61, 0.64]
```

5. K-Medians clustering algorithm was run on an unnormalized data and the B-cubed metric scores were computed as well. Results obtained after running the code were as follows:




```
K-Medians Precision List for K = 1 to 9, respectively: [0.33, 0.66, 0.77, 0.92, 0.91, 0.79, 0.91, 0.79, 0.9]
K-Medians Recall List for K = 1 to 9, respectively: [1.0, 1.0, 0.92, 0.93, 0.89, 0.61, 0.54, 0.57, 0.47]
K-Medians F-score List for K = 1 to 9 respectively: [0.47, 0.75, 0.82, 0.92, 0.89, 0.58, 0.64, 0.53, 0.6]
```

6. Finally, K-Medians clustering algorithm was run on a normalized data and the B-cubed metric scores were computed as well. Results obtained after running the code were as follows:




```
K-Medians Normalised Precision List for K = 1 to 9, respectively: [0.33, 0.66, 0.66, 0.93, 0.79, 0.79, 0.93, 0.93, 0.92]
K-Medians Normalised Recall List for K = 1 to 9, respectively: [1.0, 1.0, 0.76, 0.93, 0.65, 0.58, 0.56, 0.55, 0.5]
K-Medians Normalised F-score List for K = 1 to 9 respectively: [0.47, 0.75, 0.59, 0.93, 0.62, 0.58, 0.69, 0.61, 0.6]
```

7. Looking at the above results and plots (which also are plotted together in the same plot as given below), K-Medians on normalised data with K = 4 clusters seems to be the best one. This is because it is there that the highest B-cubed metric scores across the 3 measurements of Precision, Recall and F-score is

observed. With an average score of about 0.938, it is also there that the highest B-cubed F-score achieved, which is the most important metric as it is a reflection of both the Precision and Recall. Furthermore, the 3 B-cubed metrics coincide or are the closest, in terms of values, at this particular setting. And generally speaking, K = 4 clusters across the 3 out of the 4 settings (the exception being K-Means on un normalised data), seems to be the best cluster size as it is there where the highest F-scores are observed and where the 3 metric scores of precision, recall and F-score get closest to each other.