Assignment 2
Data Mining and Visulisation
Robert Johnson
200962268

Contents

1 Introduction **3**
- 1.1 The data . . . 3
  - 1.1.1 Issues with the data provided . . . 3

2 Types of K-means **3**
- 2.1 K-means . . . 3
- 2.2 K-means ++ . . . 3

3 Question 1 **4**

4 Question 2 **5**
- 4.1 K-means . . . 5
- 4.2 K-means++ . . . 6

5 Question 3 **7**
- 5.1 K-means . . . 7
- 5.2 K-means++ . . . 8

6 Question 4 **9**
- 6.1 K-means . . . 9
- 6.2 K-means++ . . . 10

7 Question 5 **11**
- 7.1 K-means . . . 11
- 7.2 K-means++ . . . 12

8 Question 6 **13**
- 8.1 K-means . . . 13
- 8.2 K-means++ . . . 14

9 Question 6 normalised **15**
- 9.1 K-means . . . 15
- 9.2 K-means++ . . . 16

10 Question 7 **17**
  - 10.0.1 Euclidian . . . 18
  - 10.0.2 Manhatten . . . 18
  - 10.0.3 Cosine similarity . . . 18

11 Final verdict **19**

1 Introduction

This assignment focus's around understanding and implementing k-means clustering. Looking at the impact of using different measuring methods (Euclidian, Manhatten, Cosine similarity) as well as the benefits / drawbacks of using l2 normalisation.

1.1 The data

The assignment provided us with 4 different types of items: countries, animals, fruits and veggies. Each type consists of many different records, and each record had 300 features.

1.1.1 Issues with the data provided

When running the k-means algorithm I encountered issues relating to choosing starting points, after examining the cases causing these issues I found some interesting characteristics of the data provided. Initially I assumed that an item could only be part of one specific group of items e.g fruit or veg, this apparently is not the case. "cucumber" was part of not only fruit but also veg, because of this it would be impossible to correctly classify this item into both groups at the same time. If I were to carry out this assignment in industry it would be important to either correct this case before inserting it into the classifyer, or remove it completely and instead insert it after classifying to determine its actual class.

2 Types of K-means

2.1 K-means

K-means works by choosing K random starting nodes, where K is the amount of clusters we wish to find it then assigns each point to a cluster based on minimum distance. Each cluster center is then averaged from the points that belong to it. This repeatidly happens until each center no longer moves in which case the algorithm has converged to a **local optimum** or until a certain number of rounds has been completed.

Because of the fact that each run only finds a **local optimum** it is important to repeat the process for multiple different starting clusters to find the **global optimum**.

2.2 K-means ++

K-means++ works the same as K-means after the initial selection of clusters, however choosing these initial clusters is important to find a global optimum quickly, as such David Arthur and Sergei devised a way to help select these points smartly. Rather than choosing each point completely randomly he instead proposed choosing each point with probability D, where D is the minimum distance from a point already chosen initial clusters.

3 Question 1

Implement the k-means clustering algorithm with Euclidean distance to cluster the instances into k clusters (30 marks)

```
def runKMeansOnInput(listOfItems, currentStartingPoint, whatDistanceMeasure, itemnumbers, howManyDifferentItems):
    arrayOfMethods = [DM.euclidianDistance,DM.euclidianDistance,DM.manhattenDistance,
                      DM.manhattenDistance,DM.cosineDistance,DM.cosineDistance]
    for p in range (100):
        closestPoint = arrayOfMethods[whatDistanceMeasure](listOfItems, currentStartingPoint)
        previousRound = currentStartingPoint[:]
        currentStartingPoint = np.zeros((k,300))
        countOfHowMany = np.zeros(k)
        for i in range(len(closestPoint)):
            currentStartingPoint[closestPoint[i]] = np.add(currentStartingPoint[closestPoint[i]],listOfItems[i][0])
            countOfHowMany[closestPoint[i]] += 1
        for i in range(k):
            currentStartingPoint[i] = currentStartingPoint[i]/countOfHowMany[i]
            if(countOfHowMany[i] == 0):
                print(itemnumbers)
        if(np.array_equal(previousRound,currentStartingPoint)):
            break
    F,P,R,TP,FN,FP = MP.workOutMetrics(currentStartingPoint, closestPoint, listOfItems, k, howManyDifferentItems)
    return F,R,P,TP,FP,FN
```

(a) Code to run k-means.

```
def euclidianDistance(listOfItems,currentStartingPoint):
    closestPoint = []
    for item in listOfItems:
        minValue = math.sqrt(np.sum(np.subtract(currentStartingPoint[0],item[0])**2))
        minIndex = 0
        for second in range(1,len(currentStartingPoint)):
            currentDistance = math.sqrt(np.sum(np.subtract(currentStartingPoint[second],item[0])**2))
            if (currentDistance < minValue):
                minValue = currentDistance
                minIndex = second
        closestPoint.append(minIndex)
    return closestPoint
```

(b) Code to calculate the euclidianDistance between centroids and points.

Figure 1: Pictures of k-means code

Figure (1a) takes perameters:

- listOfItems - List of datapoints
- currentStartingPoint - The current cluster locations
- whatDistanceMeasure - What measuring technique to use
- itemnumbers - The item numbers chosen initially
- howManyDifferentItems - The amount of different types of items

And returns: f-score, recall, TP, FP, FN of the current clusters

Figure (1b) takes perameters:

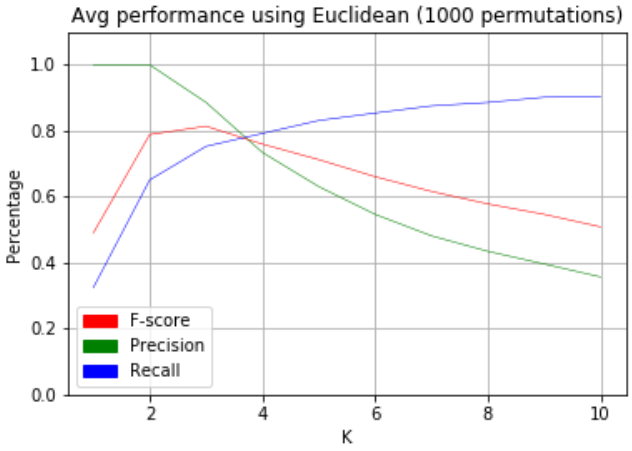
- listOfItems - List of datapoints
- currentStartingPoint - The current cluster locations

And returns: An array containing which centroid each datapoint is nearest to.

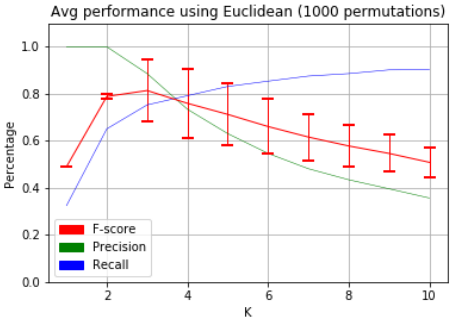
4 Question 2

Vary the value of k from 1 to 10 and compute the precision, recall, and F-score for each set of clusters. Plot k in the horizontal axis and precision, recall and F-score in the vertical axis in the same plot. (10 marks)

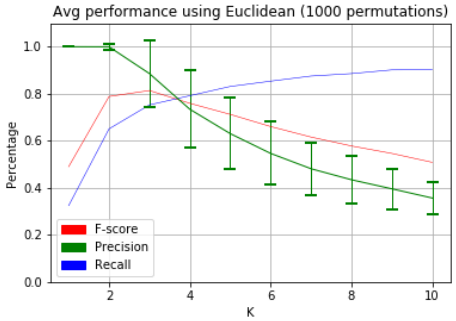
4.1 K-means



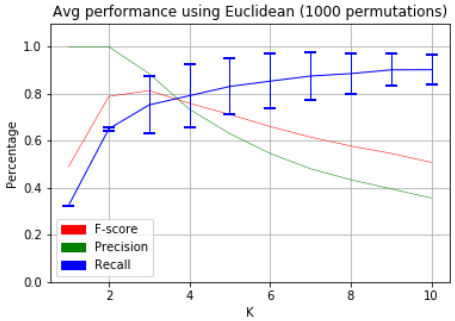
(a) Average



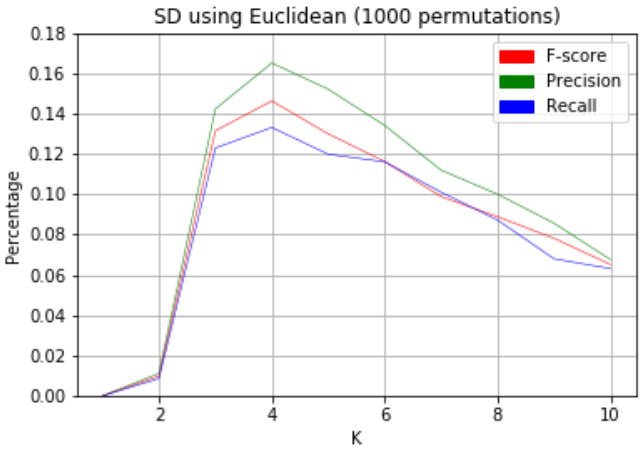
(b) F-score SD



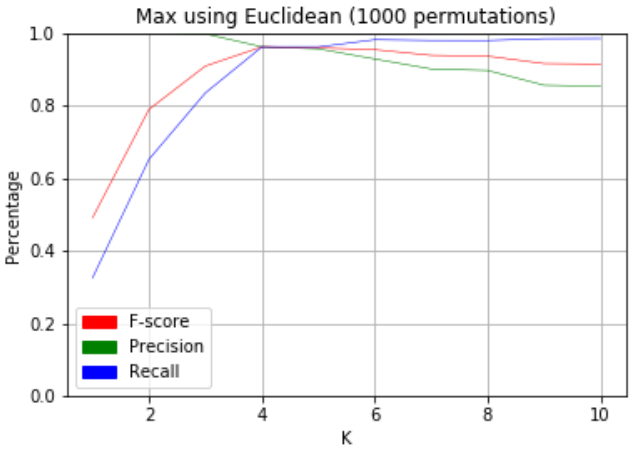
(c) Precision SD



(d) Recall SD



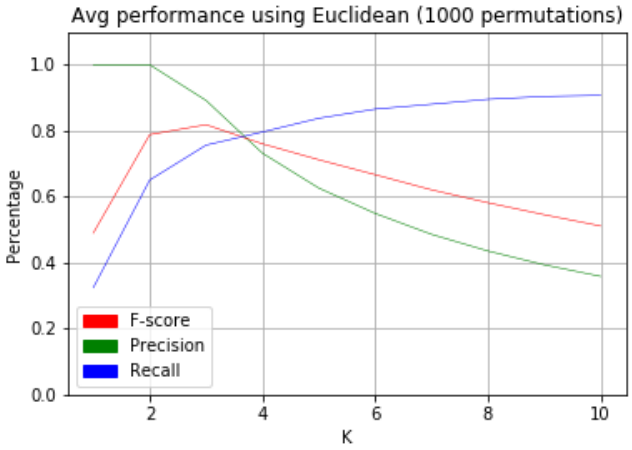
(e) All SD's



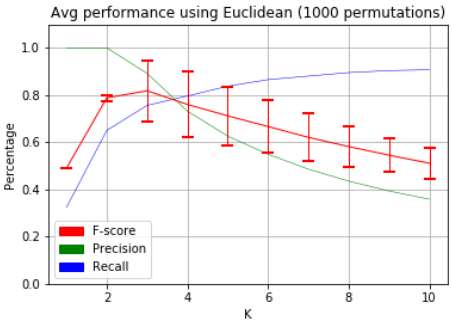
(f) Max f-score found.

Figure 2: K-means 1000 iterations using Euclidean

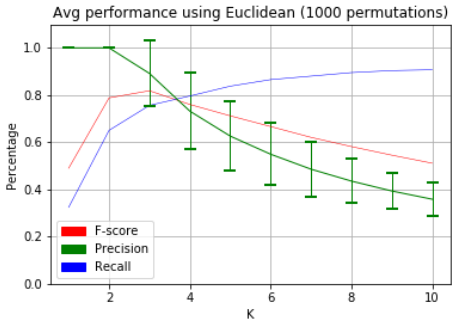
4.2 K-means++



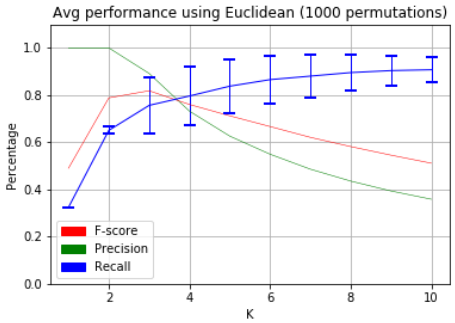
(a) Average



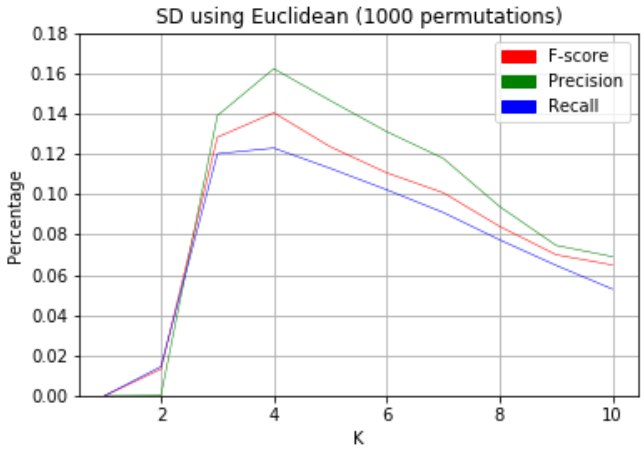
(b) F-score SD



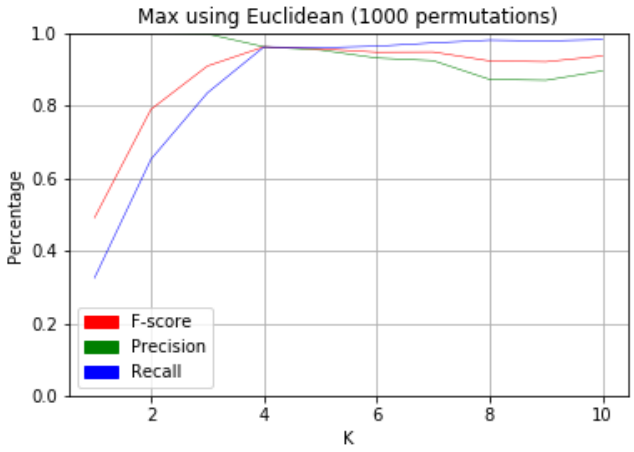
(c) Precision SD



(d) Recall SD



(e) All SD's



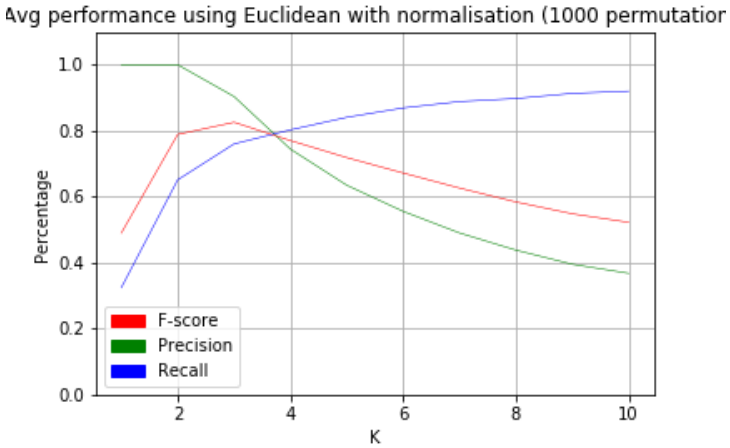
(f) Max f-score found.

Figure 3: K-means++ 1000 iterations using Euclidean

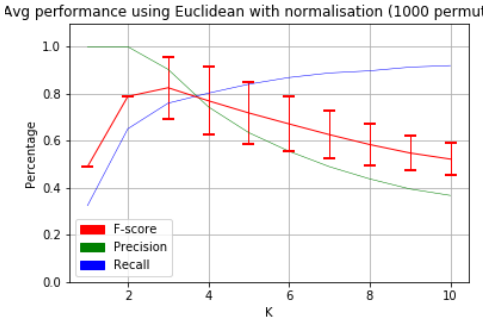
5 Question 3

Now re-run the k-means clustering algorithm you implemented in part (1) but normalise each feature vector to unit L2 length before computing Euclidean distances. Vary the value of k from 1 to 10 and compute the precision, recall, and F-score for each set of clusters. Plot k in the horizontal axis and precision, recall and F-score in the vertical axis in the same plot. (10 marks)

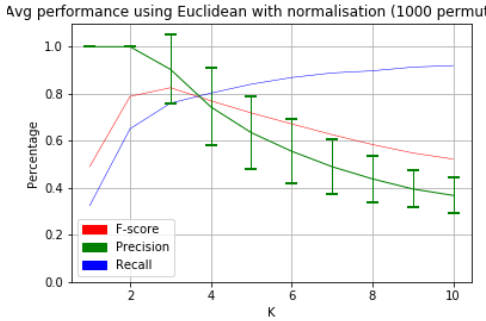
5.1 K-means



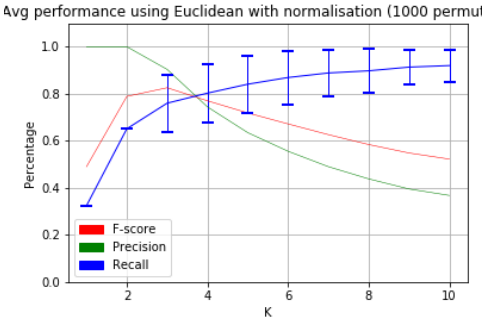
(a) Average



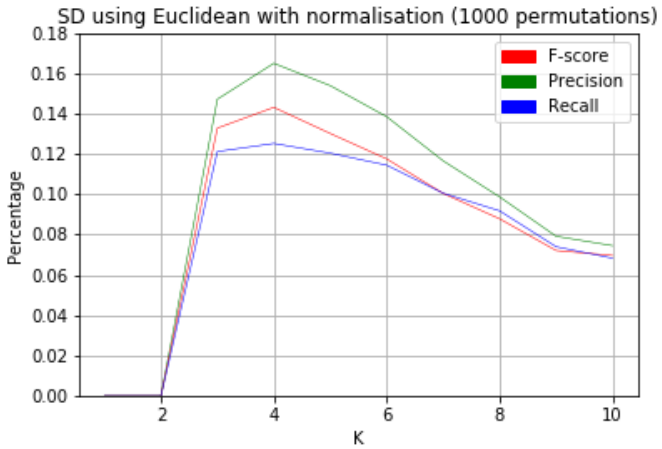
(b) F-score SD



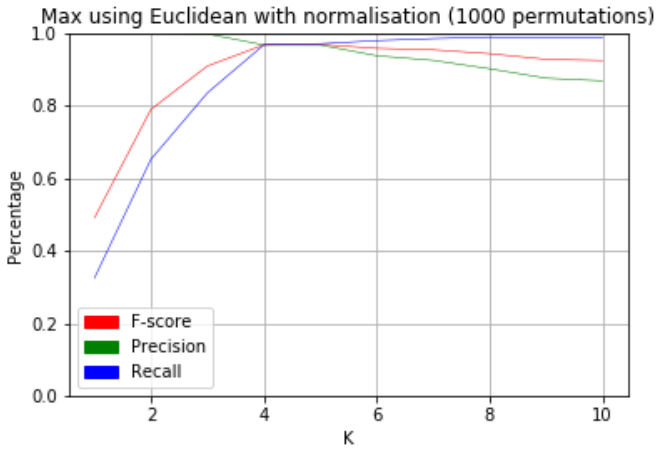
(c) Precision SD



(d) Recall SD



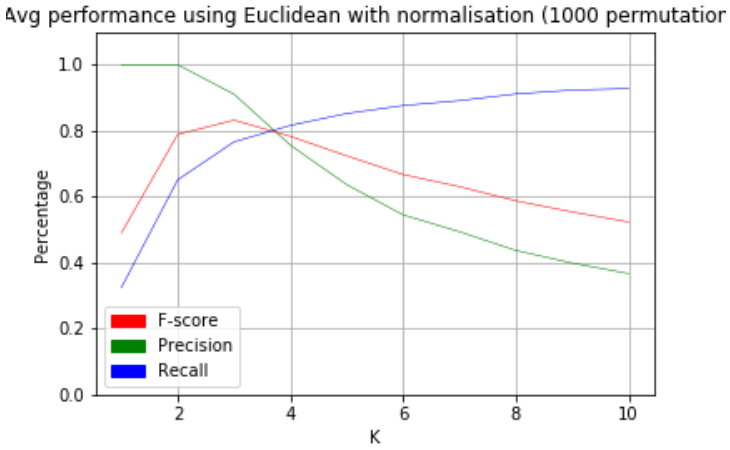
(e) All SD's



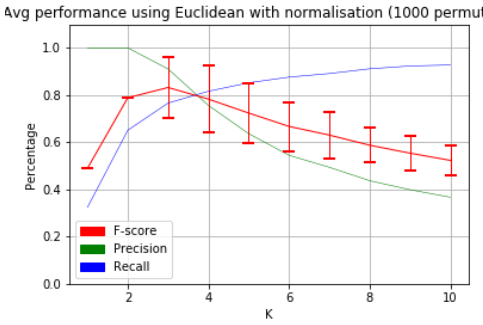
(f) Max f-score found.

Figure 4: K-means 1000 iterations using Euclidean with normalisation

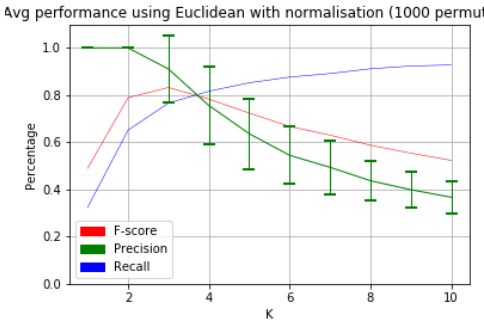
5.2 K-means++



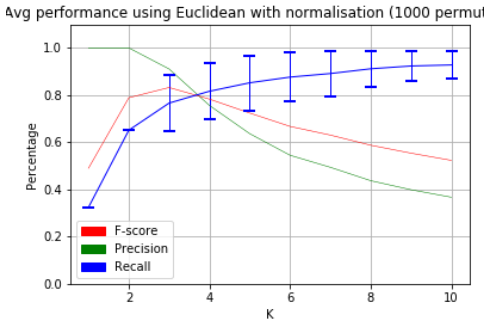
(a) Average



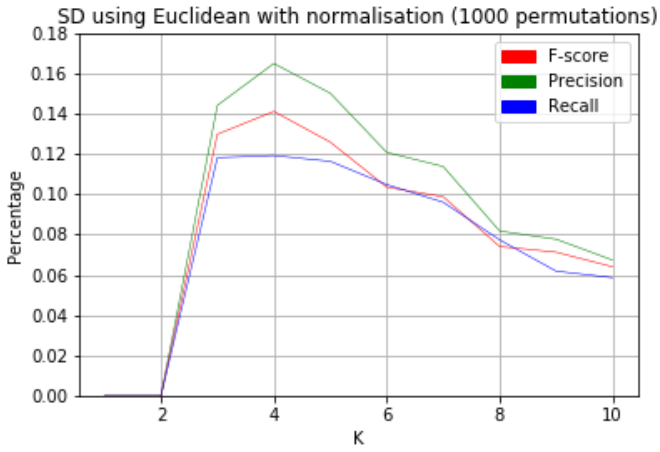
(b) F-score SD



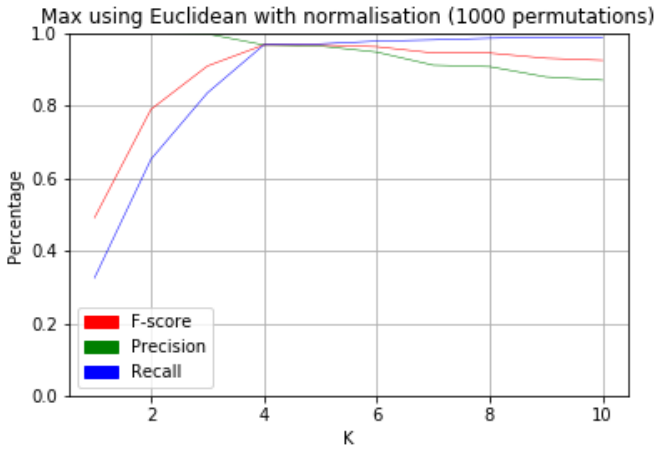
(c) Precision SD



(d) Recall SD



(e) All SD's



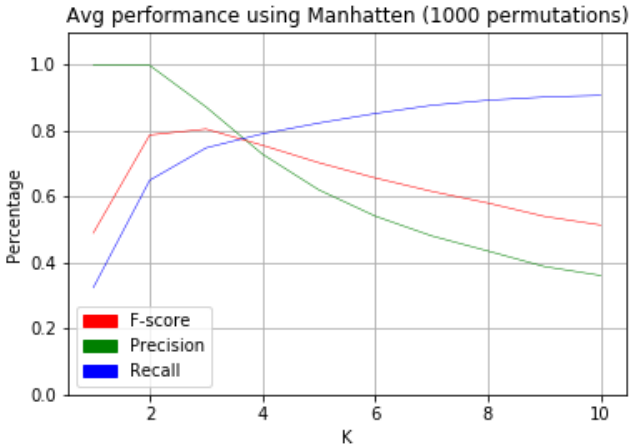
(f) Max f-score found.

Figure 5: K-means++ 1000 iterations using Euclidean with normalisation

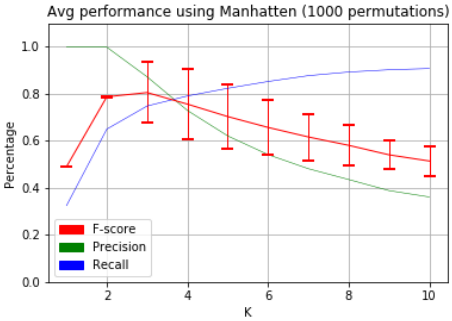
6 Question 4

Now re-run the k-means clustering algorithm you implemented in part (1) but this time use Manhattan distance over the unnormalised feature vectors. Vary the value of k from 1 to 10 and compute the precision, recall, and F-score for each set of clusters. Plot k in the horizontal axis and precision, recall and F-score in the vertical axis in the same plot. (10 marks)

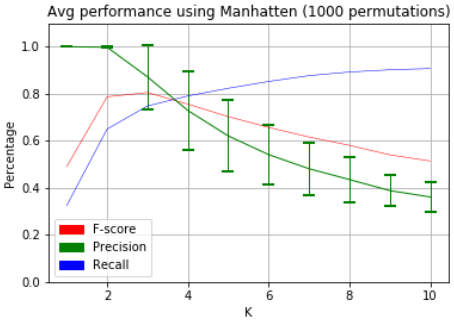
6.1 K-means



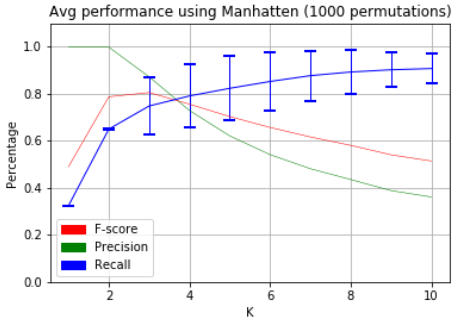
(a) Average



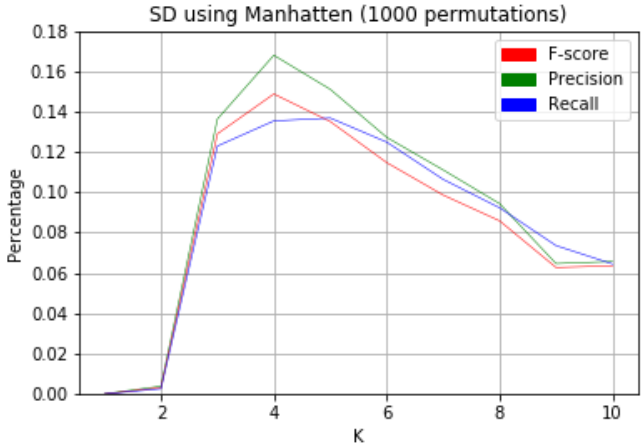
(b) F-score SD



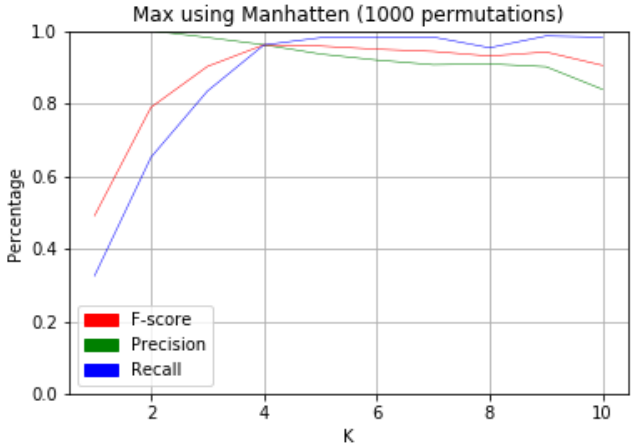
(c) Precision SD



(d) Recall SD



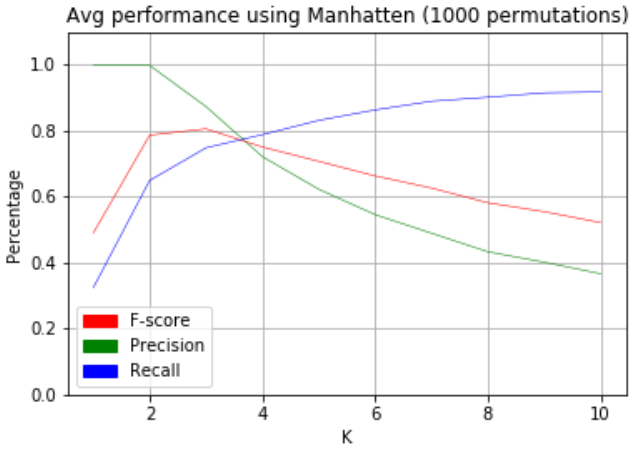
(e) All SD's



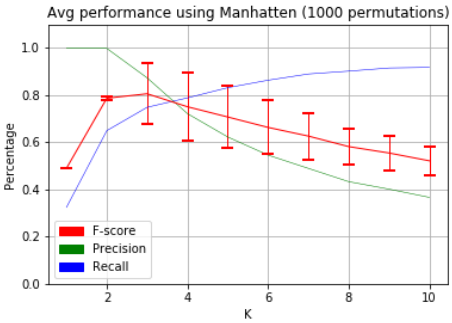
(f) Max f-score found.

Figure 6: K-means 1000 iterations using Manhatten

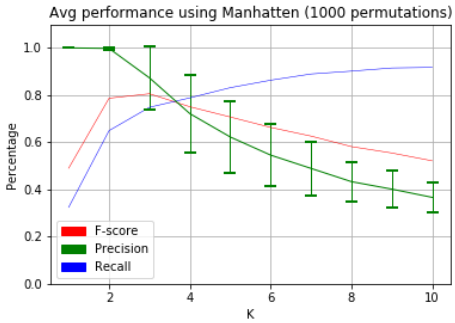
6.2 K-means++



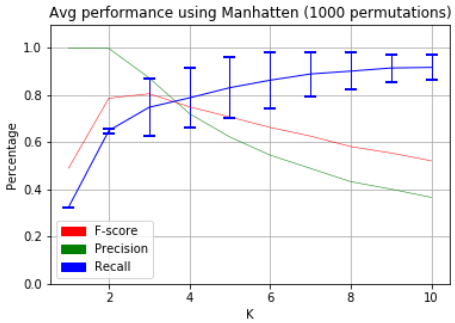
(a) Average



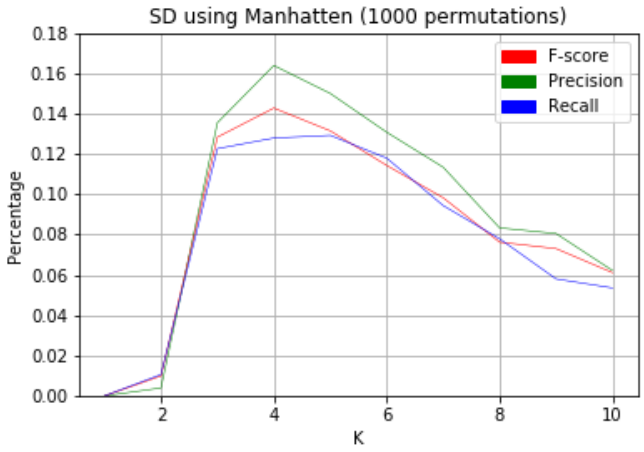
(b) F-score SD



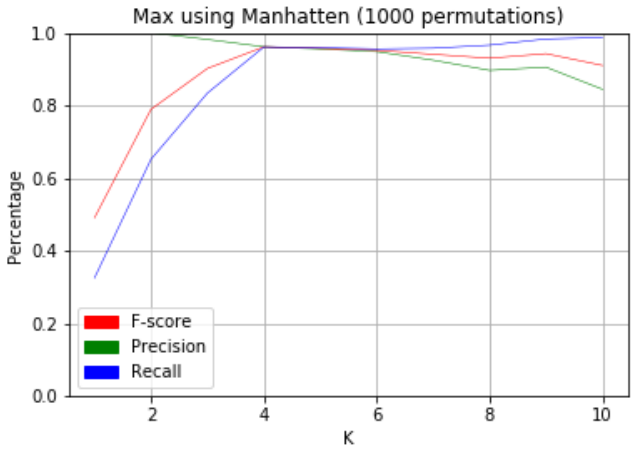
(c) Precision SD



(d) Recall SD



(e) All SD's



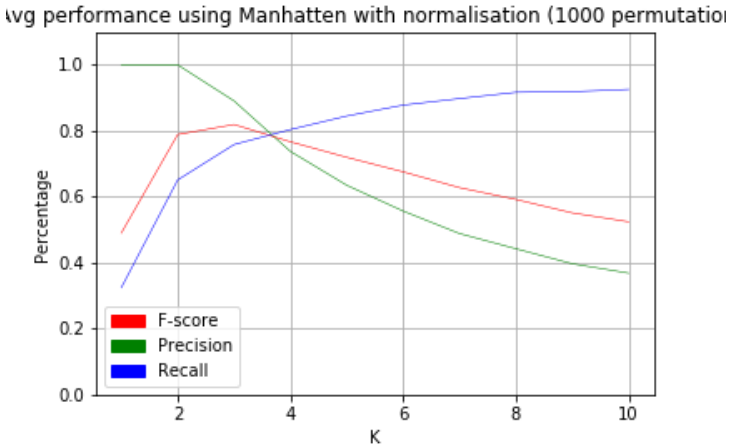
(f) Max f-score found.

Figure 7: K-means++ 1000 iterations using Manhatten

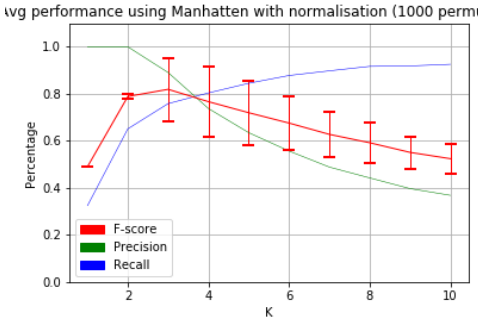
7 Question 5

Now re-run the k-means clustering algorithm you implemented in part (1) but this time use Manhattan distance with L2 normalised feature vectors. Vary the value of k from 1 to 10 and compute the precision, recall, and F-score for each set of clusters. Plot k in the horizontal axis and precision, recall and F-score in the vertical axis in the same plot. (10 marks)

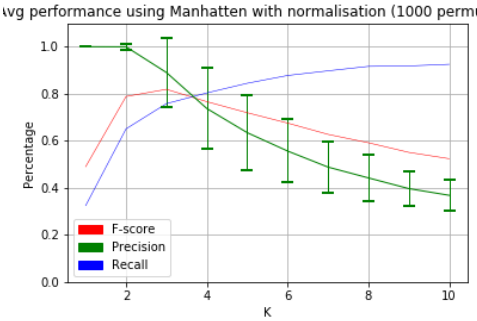
7.1 K-means



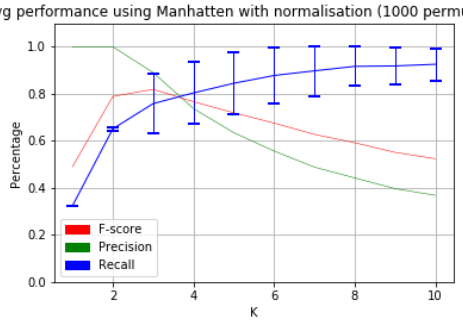
(a) Average



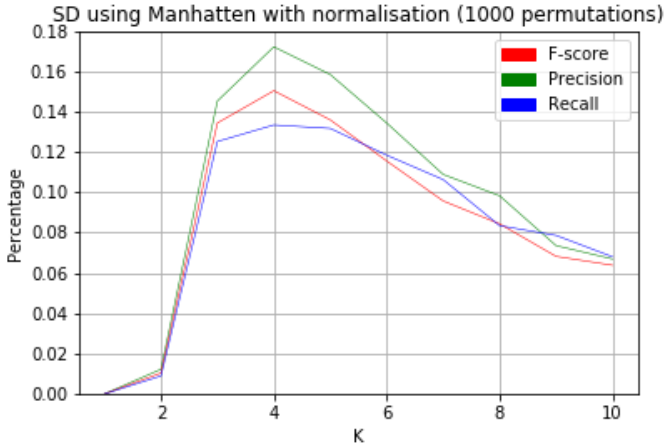
(b) F-score SD



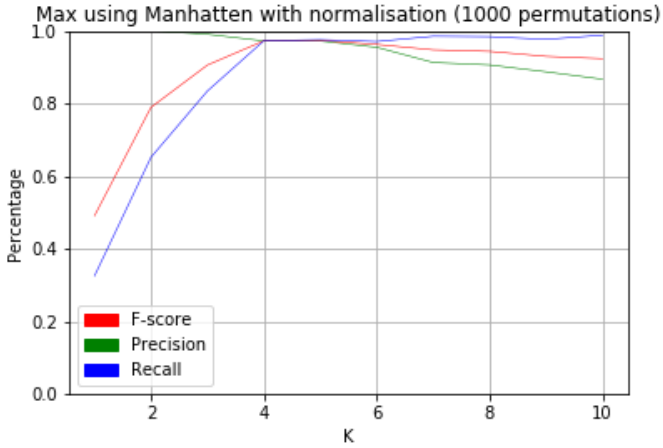
(c) Precision SD



(d) Recall SD



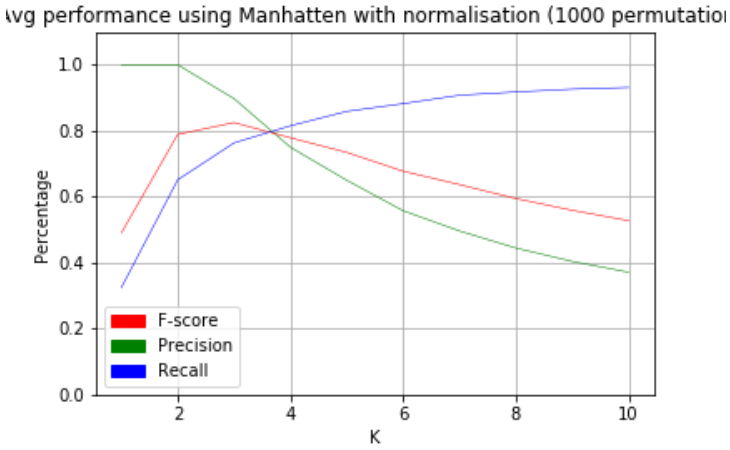
(e) All SD's



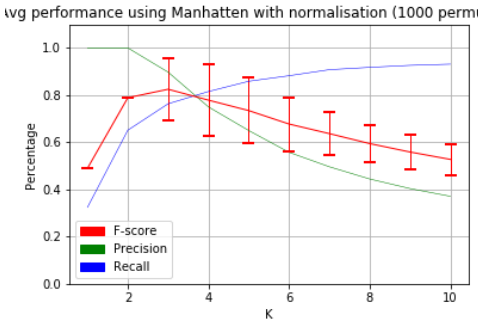
(f) Max f-score found.

Figure 8: K-means 1000 iterations using Manhatten with normalisation

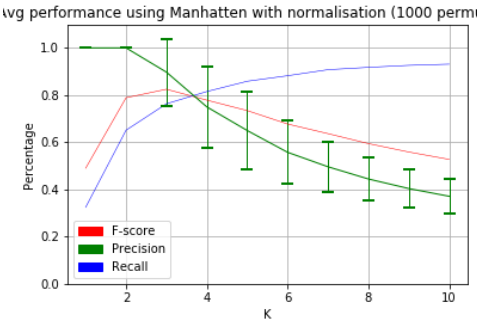
7.2 K-means++



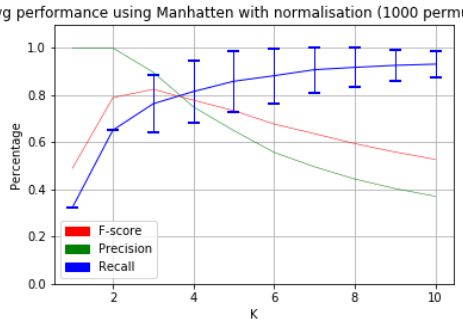
(a) Average



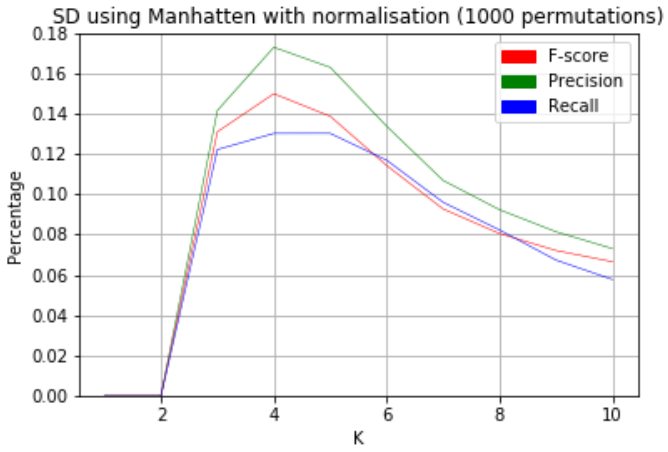
(b) F-score SD



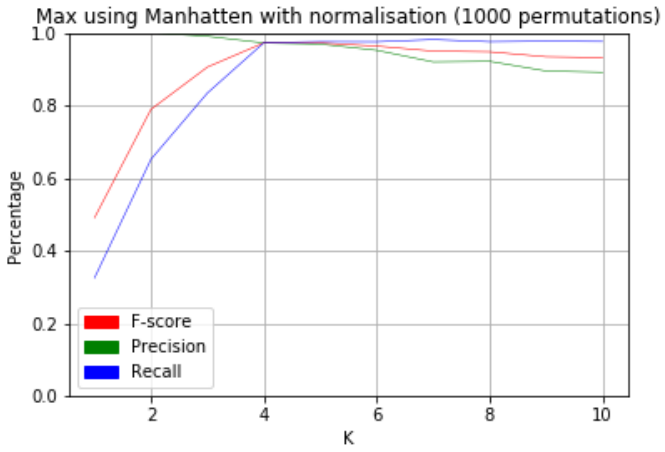
(c) Precision SD



(d) Recall SD



(e) All SD's



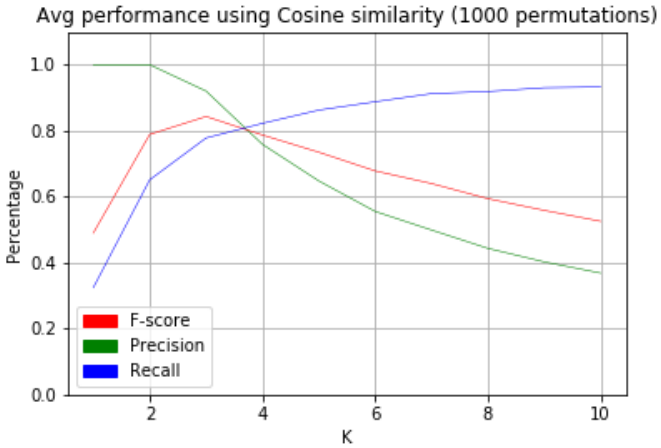
(f) Max f-score found.

Figure 9: K-means++ 1000 iterations using Manhatten with normalisation

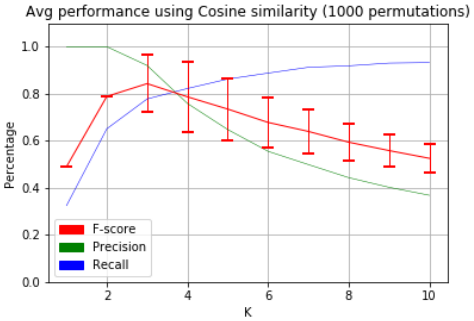
8 Question 6

Now re-run the k-means clustering algorithm you implemented in part (1) but this time use cosine similarity as the distance (similarity) measure.Vary the value of k from 1 to 10 and 2 compute the precision, recall, and F-score for each set of clusters. Plot k in the horizontal axis and precision, recall and F-score in the vertical axis in the same plot (10 marks)

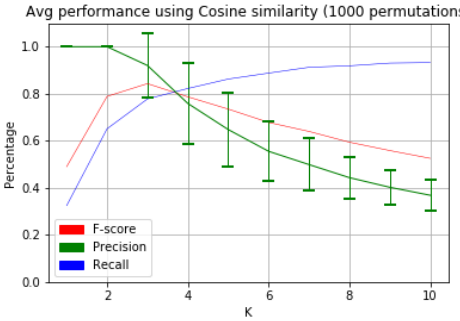
8.1 K-means



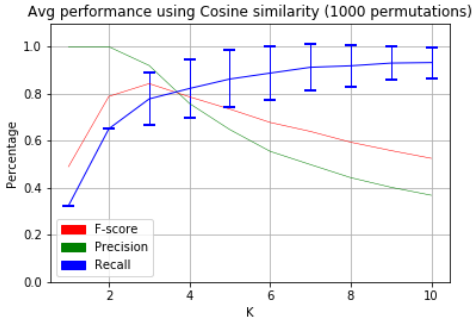
(a) Average



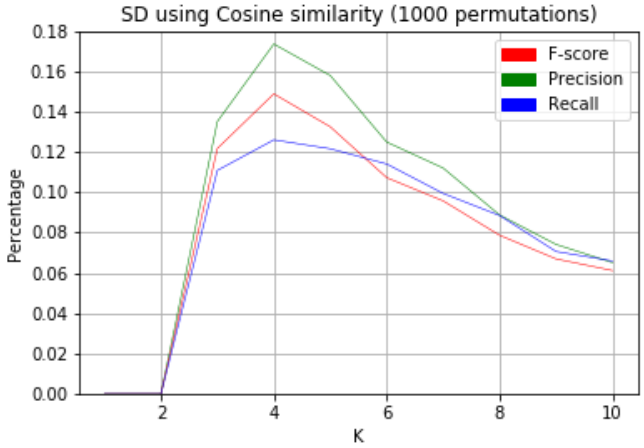
(b) F-score SD



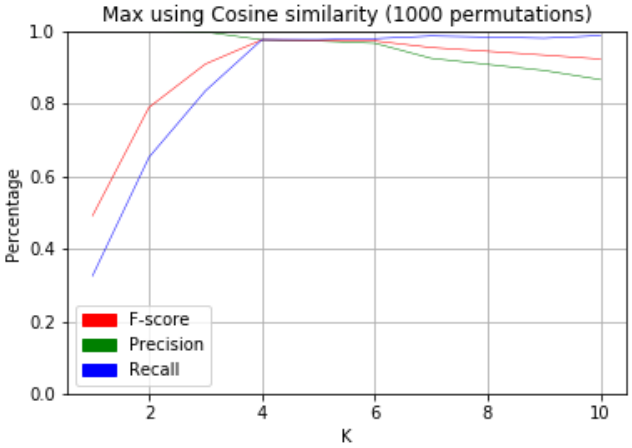
(c) Precision SD



(d) Recall SD



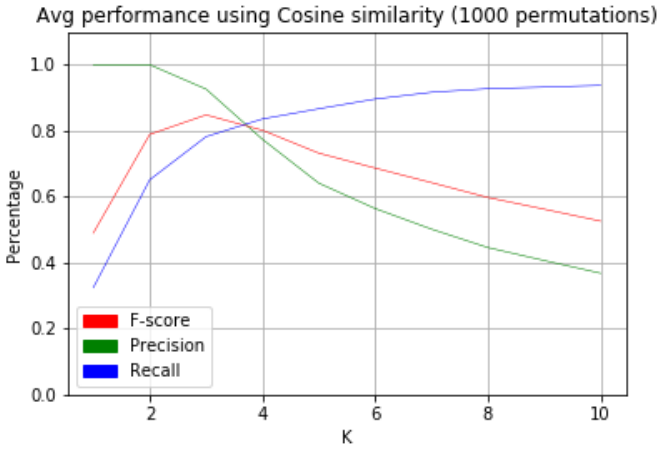
(e) All SD's



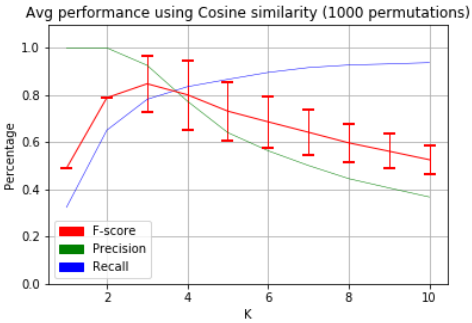
(f) Max f-score found.

Figure 10: K-means 1000 iterations using Cosine similarity

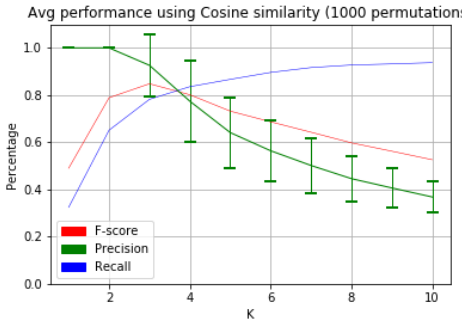
8.2 K-means++



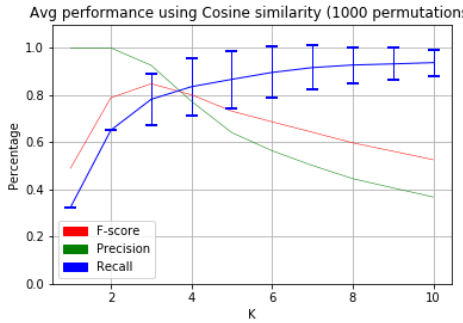
(a) Average



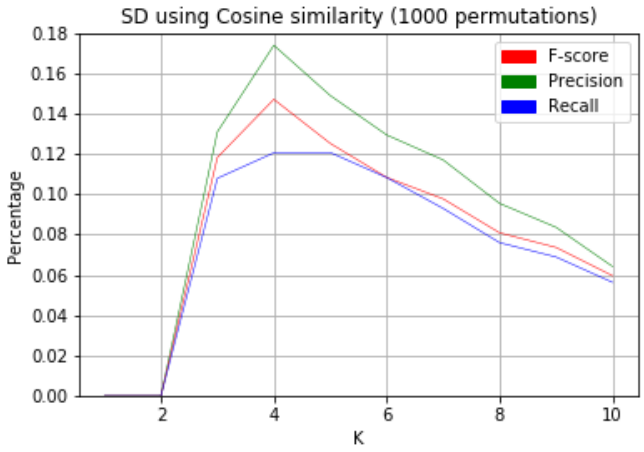
(b) F-score SD



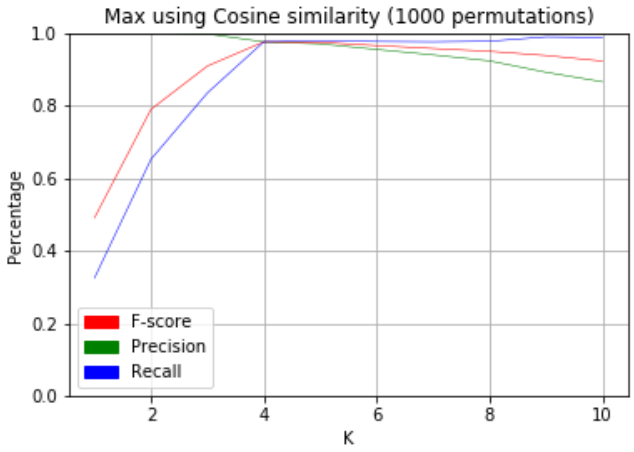
(c) Precision SD



(d) Recall SD



(e) All SD's

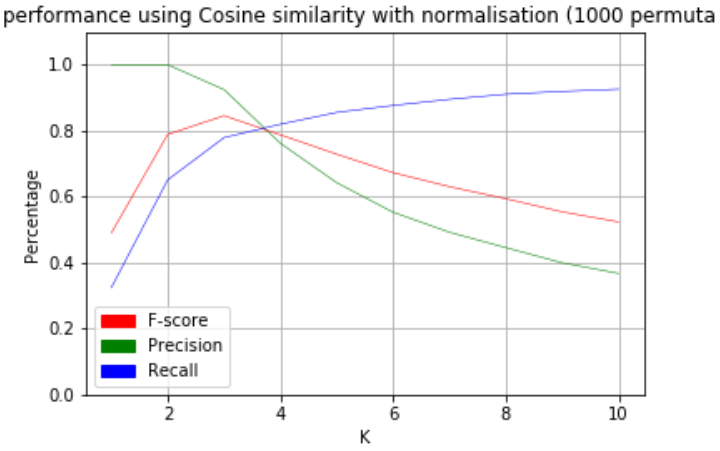


(f) Max f-score found.

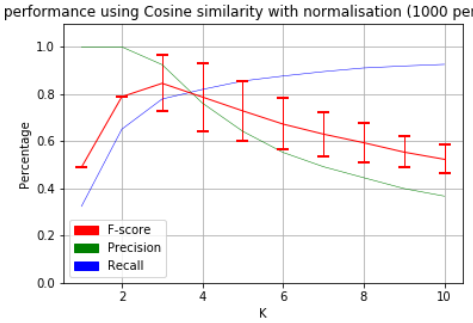
Figure 11: K-means++ 1000 iterations using Cosine similarity

9 Question 6 normalised

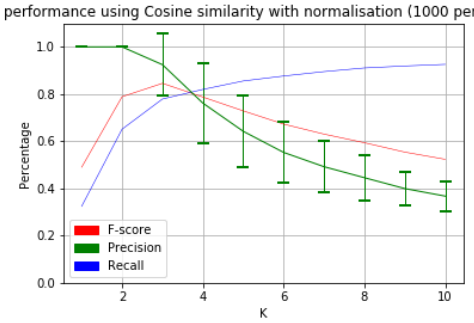
9.1 K-means



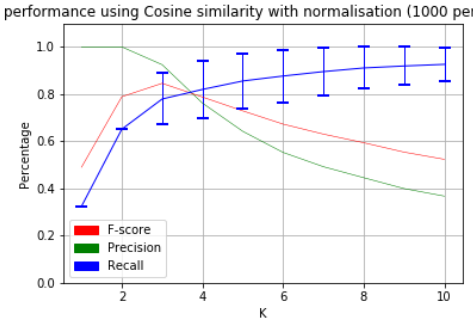
(a) Average



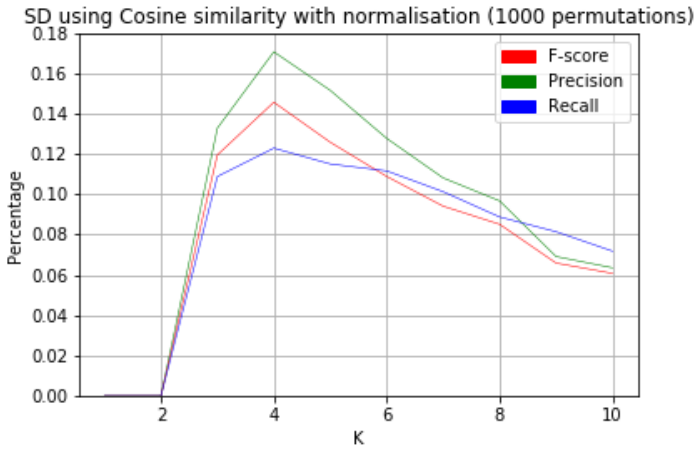
(b) F-score SD



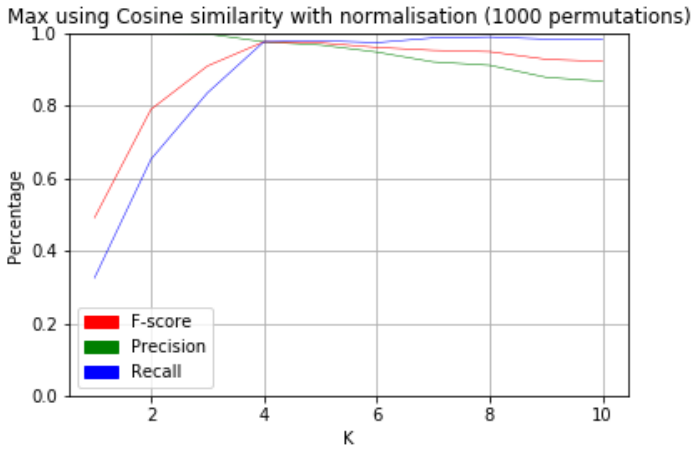
(c) Precision SD



(d) Recall SD



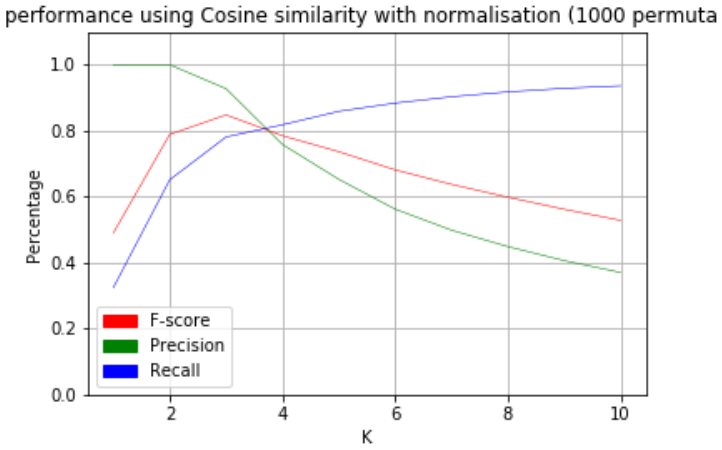
(e) All SD's



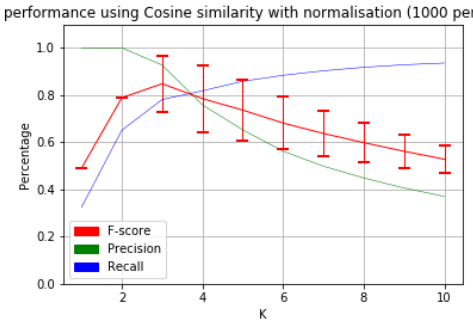
(f) Max f-score found.

Figure 12: K-means 1000 iterations using Cosine similarity normalised

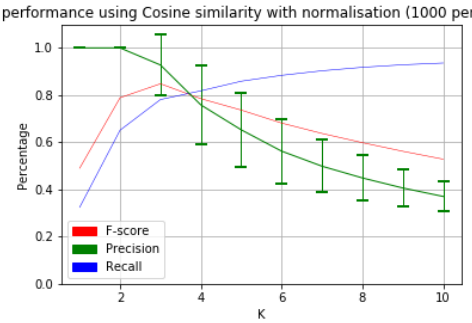
9.2 K-means++



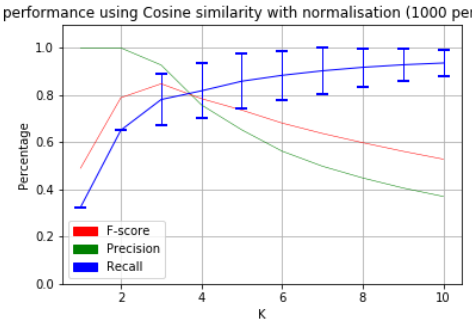
(a) Average



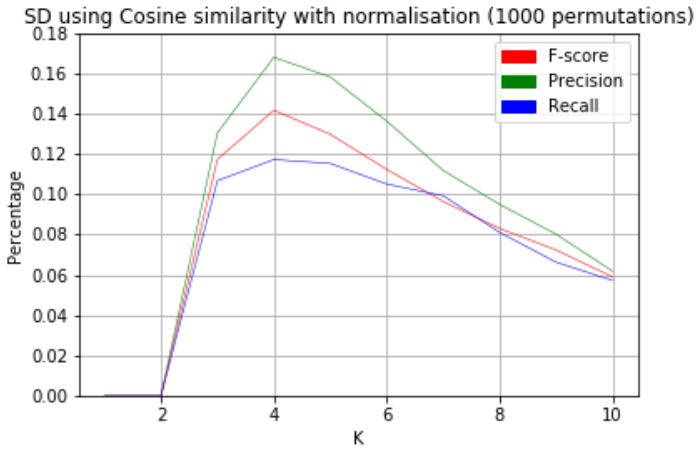
(b) F-score SD



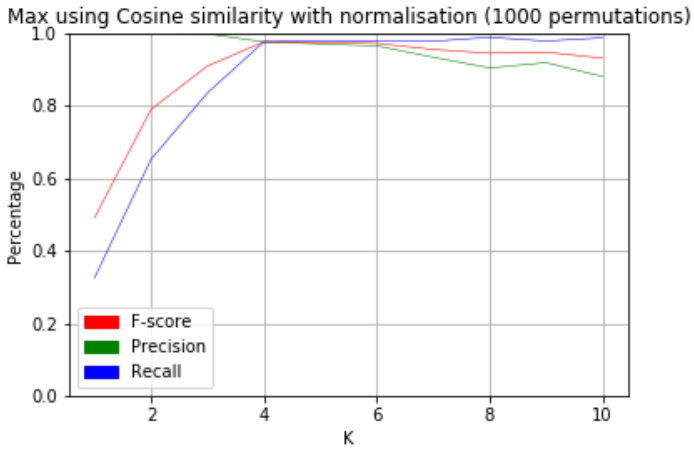
(c) Precision SD



(d) Recall SD



(e) All SD's



(f) Max f-score found.

Figure 13: K-means++ 1000 iterations using Cosine similarity normalised

10 Question 7

Comparing the different clusterings you obtained in (2)-(6) discuss what is the best setting for k-means clustering for this dataset. (20 marks)

Statistics	K	k-means Euclidean	k-means++ Euclidean	k-means Euclidean with normalisation	k-means++ Euclidean with normalisation	k-means Manhatten	k-means++ Manhatten	k-means Manhatten with normalisation	k-means++ Manhatten with normalisation	k-means Cosine similarity	k-means++ Cosine similarity	k-means Cosine Similarity with normalisation	k-means++ Cosine similarity with normalisation
f-scores avg	1	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%
	2	78.86%	78.83%	78.89%	78.89%	78.73%	78.68%	78.86%	78.89%	78.89%	78.89%	78.89%	78.89%
	3	81.38%	81.84%	82.57%	83.22%	80.52%	80.58%	81.90%	82.48%	84.32%	84.81%	84.57%	84.81%
	4	75.95%	76.04%	77.02%	78.30%	75.60%	75.07%	76.65%	77.87%	78.71%	80.15%	78.76%	78.50%
	5	71.25%	71.22%	71.87%	72.39%	70.32%	70.73%	71.95%	73.44%	73.50%	73.21%	72.90%	73.65%
	6	66.09%	66.69%	67.20%	66.74%	65.71%	66.32%	67.55%	67.75%	67.83%	68.69%	67.30%	68.16%
	7	61.62%	62.11%	62.65%	63.06%	61.66%	62.64%	62.76%	63.69%	64.01%	64.28%	63.06%	63.76%
	8	57.84%	58.20%	58.44%	58.77%	58.10%	58.19%	59.19%	59.47%	59.43%	59.79%	59.41%	59.87%
	9	54.67%	54.56%	54.85%	55.41%	54.07%	55.48%	55.12%	55.90%	55.88%	56.24%	55.43%	56.21%
	10	50.89%	51.16%	52.26%	52.36%	51.44%	52.17%	52.43%	52.77%	52.59%	52.63%	52.39%	52.86%
Avg distance from Best		2.09%	1.88%	1.37%	1.03%	2.33%	1.96%	1.30%	0.72%	0.43%	0.08%	0.67%	0.27%

(a) Average f-score

Statistics	K	k-means Euclidean	k-means++ Euclidean	k-means Euclidean with normalisation	k-means++ Euclidean with normalisation	k-means Manhatten	k-means++ Manhatten	k-means Manhatten with normalisation	k-means++ Manhatten with normalisation	k-means Cosine similarity	k-means++ Cosine similarity	k-means Cosine Similarity with normalisation	k-means++ Cosine similarity with normalisation
f-scores max	1	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%
	2	78.89%	78.89%	78.89%	78.89%	78.89%	78.89%	78.89%	78.89%	78.89%	78.89%	78.89%	78.89%
	3	90.83%	90.83%	90.83%	90.83%	90.16%	90.16%	90.54%	90.54%	90.83%	90.83%	90.83%	90.83%
	4	96.10%	96.10%	96.64%	96.64%	96.10%	96.10%	97.21%	97.21%	97.50%	97.50%	97.50%	97.50%
	5	95.83%	95.56%	96.78%	96.63%	95.78%	95.71%	97.35%	97.20%	97.35%	97.35%	97.21%	97.36%
	6	95.35%	94.63%	95.70%	96.14%	94.91%	95.16%	96.22%	96.32%	97.19%	96.47%	95.95%	97.06%
	7	93.78%	94.69%	95.30%	94.42%	94.29%	94.06%	94.73%	94.94%	95.37%	95.66%	95.15%	95.39%
	8	93.56%	92.24%	94.23%	94.40%	93.03%	92.98%	94.31%	94.74%	94.34%	94.87%	94.77%	94.35%
	9	91.51%	92.01%	92.69%	92.97%	94.13%	94.23%	92.94%	93.38%	93.29%	93.73%	92.69%	94.64%
	10	91.37%	93.62%	92.34%	92.41%	90.47%	91.00%	92.31%	93.15%	92.20%	92.19%	92.09%	92.98%
Avg distance from Best		1.33%	1.20%	0.72%	0.72%	1.28%	1.23%	0.61%	0.42%	0.36%	0.31%	0.55%	0.16%

(b) Max f-score

Statistics	K	k-means Euclidean	k-means++ Euclidean	k-means Euclidean with normalisation	k-means++ Euclidean with normalisation	k-means Manhatten	k-means++ Manhatten	k-means Manhatten with normalisation	k-means++ Manhatten with normalisation	k-means Cosine similarity	k-means++ Cosine similarity	k-means Cosine Similarity with normalisation	k-means++ Cosine similarity with normalisation
f-score - SD	1	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%
	2	77.88%	77.50%	78.89%	78.89%	78.42%	77.69%	77.83%	78.89%	78.89%	78.89%	78.89%	78.89%
	3	68.24%	69.02%	69.29%	70.25%	67.63%	67.75%	68.47%	69.39%	72.15%	72.99%	72.63%	73.09%
	4	61.33%	62.01%	62.72%	64.20%	60.74%	60.80%	61.63%	62.89%	63.84%	65.45%	64.20%	64.34%
	5	58.26%	58.87%	58.86%	59.81%	56.82%	57.60%	58.37%	59.58%	60.26%	60.70%	60.33%	60.68%
	6	54.47%	55.63%	55.45%	56.39%	54.24%	54.90%	56.01%	56.33%	57.12%	57.87%	56.42%	56.93%
	7	51.75%	52.04%	52.61%	53.19%	51.80%	52.82%	53.20%	54.42%	54.44%	54.52%	53.66%	54.12%
	8	48.96%	49.80%	49.66%	51.37%	49.53%	50.57%	50.77%	51.42%	51.57%	51.70%	50.91%	51.56%
	9	46.86%	47.57%	47.64%	48.29%	47.81%	48.18%	48.30%	48.70%	49.19%	48.88%	48.85%	48.98%
	10	44.38%	44.67%	45.30%	45.96%	45.09%	46.07%	46.06%	46.12%	46.47%	46.68%	46.32%	46.97%
Avg distance from Best		2.63%	2.13%	1.79%	1.00%	2.63%	2.20%	1.78%	1.06%	0.44%	0.07%	0.62%	0.28%

(c) f-score - SD

Statistics	K	k-means Euclidean	k-means++ Euclidean	k-means Euclidean with normalisation	k-means++ Euclidean with normalisation	k-means Manhatten	k-means++ Manhatten	k-means Manhatten with normalisation	k-means++ Manhatten with normalisation	k-means Cosine similarity	k-means++ Cosine similarity	k-means Cosine Similarity with normalisation	k-means++ Cosine similarity with normalisation
f-score + SD	1	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%	49.04%
	2	79.84%	80.16%	78.89%	78.89%	79.03%	79.67%	79.89%	78.89%	78.89%	78.89%	78.89%	78.89%
	3	94.52%	94.66%	95.84%	96.19%	93.41%	93.40%	95.32%	95.57%	96.48%	96.62%	96.51%	96.53%
	4	90.57%	90.07%	91.32%	92.40%	90.47%	89.33%	91.68%	92.85%	93.58%	94.85%	93.31%	92.65%
	5	84.23%	83.57%	84.87%	84.97%	83.82%	83.87%	85.53%	87.29%	86.74%	85.72%	85.47%	86.62%
	6	77.70%	77.74%	78.94%	77.09%	77.17%	77.74%	79.10%	79.17%	78.55%	79.52%	78.17%	79.38%
	7	71.49%	72.18%	72.68%	72.93%	71.51%	72.46%	72.33%	72.95%	73.57%	74.05%	72.46%	73.39%
	8	66.72%	66.60%	67.21%	66.18%	66.68%	65.81%	67.62%	67.51%	67.29%	67.87%	67.92%	68.17%
	9	62.48%	61.56%	62.06%	62.54%	60.34%	62.78%	61.93%	63.11%	62.57%	63.60%	62.02%	63.44%
	10	57.40%	57.65%	59.22%	58.76%	57.79%	58.28%	58.81%	59.42%	58.70%	58.59%	58.46%	58.76%
Avg distance from Best		1.87%	1.95%	1.26%	1.37%	2.35%	2.03%	1.15%	0.69%	0.73%	0.40%	1.05%	0.58%

(d) f-score + SD

Figure 14: f-score measures

The above tables shows the performance of each setting for every K (1-10), the final row representing the avg value for the max value found for each k - the value found for the current setting.

10.0.1 Euclidian

Averaging perforamance over all k-values for Euclidian we notice that the normalised counterparts of k-means and k-means++ perform better in all aspects. However comparing k-means and k-means++ within this catagory we notice that k-means beats k-means++ in the terms of avg-SD this is due to the small difference in average performance and the large standard deviation of k-means. k-means++ performing better in all other aspects. Therefore the best performing Euclidian settings on average is k-means and k-means++ when normalised.

10.0.2 Manhatten

Again much like Euclidian k-means and k-means++ perform worse than their counterparts with normalisation in all aspects. Comparing the average performance of k-means and k-means++ actually shows us that k-means++ outperforms k-means when normalised, therefore the best setting for Manhatten on average is k-means++ with normalisation.

10.0.3 Cosine similarity

Comparing cosine simarity settings we notice that k-means++ out perfrom their k-means counterparts both normalised and un-normalised in all aspects. Comparing the normalised and un-normalised settings we notice that without normalisation results in the best average f-score however the normalised setting finds the best max f-score on average.

11 Final verdict

Removing the values which are dominant settings within each catagory, Euclidian, Manhatten and cosine similarity returns the following tables.

Statistics	K	k-means Euclidean with normalisation	k-means++ Euclidean with normalisation	k-means++ Manhatten with normalisation	k-means++ Cosine similarity	k-means++ Cosine similarity with normalisation
f-scores avg	1	49.04%	49.04%	49.04%	49.04%	49.04%
	2	78.89%	78.89%	78.89%	78.89%	78.89%
	3	82.57%	83.22%	82.48%	84.81%	84.81%
	4	77.02%	78.30%	77.87%	80.15%	78.50%
	5	71.87%	72.39%	73.44%	73.21%	73.65%
	6	67.20%	66.74%	67.75%	68.69%	68.16%
	7	62.65%	63.06%	63.69%	64.28%	63.76%
	8	58.44%	58.77%	59.47%	59.79%	59.87%
	9	54.85%	55.41%	55.90%	56.24%	56.21%
	10	52.26%	52.36%	52.77%	52.63%	52.86%
Avg distance from Best		1.37%	1.03%	0.72%	0.08%	0.27%

(a) Average f-score

Statistics	K	k-means Euclidean with normalisation	k-means++ Euclidean with normalisation	k-means++ Manhatten with normalisation	k-means++ Cosine similarity	k-means++ Cosine similarity with normalisation
f-scores max	1	49.04%	49.04%	49.04%	49.04%	49.04%
	2	78.89%	78.89%	78.89%	78.89%	78.89%
	3	90.83%	90.83%	90.54%	90.83%	90.83%
	4	96.64%	96.64%	97.21%	97.50%	97.50%
	5	96.78%	96.63%	97.20%	97.35%	97.36%
	6	95.70%	96.14%	96.32%	96.47%	97.06%
	7	95.30%	94.42%	94.94%	95.66%	95.39%
	8	94.23%	94.40%	94.74%	94.87%	94.35%
	9	92.69%	92.97%	93.38%	93.73%	94.64%
	10	92.34%	92.41%	93.15%	92.19%	92.98%
Avg distance from Best		0.72%	0.72%	0.42%	0.31%	0.16%

(b) Max f-score

Statistics	K	k-means Euclidean with normalisation	k-means++ Euclidean with normalisation	k-means++ Manhatten with normalisation	k-means++ Cosine similarity	k-means++ Cosine similarity with normalisation
f-score - SD	1	49.04%	49.04%	49.04%	49.04%	49.04%
	2	78.89%	78.89%	78.89%	78.89%	78.89%
	3	69.29%	70.25%	69.39%	72.99%	73.09%
	4	62.72%	64.20%	62.89%	65.45%	64.34%
	5	58.86%	59.81%	59.58%	60.70%	60.68%
	6	55.45%	56.39%	56.33%	57.87%	56.93%
	7	52.61%	53.19%	54.42%	54.52%	54.12%
	8	49.66%	51.37%	51.42%	51.70%	51.56%
	9	47.64%	48.29%	48.70%	48.88%	48.98%
	10	45.30%	45.96%	46.12%	46.68%	46.97%
Avg distance from Best		1.79%	1.00%	1.06%	0.07%	0.28%

(c) f-score - SD

Statistics	K	k-means Euclidean with normalisation	k-means++ Euclidean with normalisation	k-means++ Manhatten with normalisation	k-means++ Cosine similarity	k-means++ Cosine similarity with normalisation
f-score + SD	1	49.04%	49.04%	49.04%	49.04%	49.04%
	2	78.89%	78.89%	78.89%	78.89%	78.89%
	3	95.84%	96.19%	95.57%	96.62%	96.53%
	4	91.32%	92.40%	92.85%	94.85%	92.65%
	5	84.87%	84.97%	87.29%	85.72%	86.62%
	6	78.94%	77.09%	79.17%	79.52%	79.38%
	7	72.68%	72.93%	72.95%	74.05%	73.39%
	8	67.21%	66.18%	67.51%	67.87%	68.17%
	9	62.06%	62.54%	63.11%	63.60%	63.44%
	10	59.22%	58.76%	59.42%	58.59%	58.76%
Avg distance from Best		1.26%	1.37%	0.69%	0.40%	0.58%

(d) f-score + SD

Figure 15: f-score measures Reduced

Looking at cosine similarity we notice that all other settings are dominated. Therefore when determining the setting we wish to use we should decide what are the most important aspects for our application, for example if we wish to run the algorithm for many iterations to find the best local optimum it would be most beneficial to use cosine similarity with normalisation using k-means++, however if we wish to run for few iterations and find an average cluster setup it would be better to use k-means++ without normalisation.

Finally if we allow the user to use different settings for each k-value it would beneficial to reference figure 14 for example the settings to find the best k-value on average would be k-means++ using cosine similarity and a k-value of 3. If we wanted to find the best local optimum of many runs we would be better off using a k value of 4 with any setting involving cosine similarity.