Assignment 4

Information Retrieval

CS 834

Fall 2017

Mohammed Nauman Sididque

November 26, 2017

Contents

Chapter 1

Problem 8.3

1.1 Problem Statement

For one query in the CACM collection (provided at the book website), generate a ranking

using Galago, and then calculate average precision, NDCG at 5 and 10, precision at 10, and

the reciprocal rank by hand.

1.2 Solution

This problem uses CACM corpus provided in the test collection of the book. The queries

ran against the CACM collection have been used from the processed queries section of the

CACM test collection.

1.2.1 Query:Interested in articles on robotics motion planning particularly the geometric and combinatorial aspects we are not interested in the dynamics of arm motion

The below xml file shows the query which was used in galago for batch-search.

```
1  <parameters>
2  <query>
3    <number>6</number>
4    <text>  interested  in  articles  on  robotics   motion  planning  particularly  the
          geometric  and  combinatorial  aspects    we  are  not  interested  in  the
          dynamics  of  arm  motion    </text>
5  </query>
6  </parameters>
```

The below commands were run on command prompt for generating the ranked list of query with 10 results.

```
1  F:\Fall2017\InformationRetreival\Assignment3\Galgo\galagosearch-1.04-bin\galagosearch-1.04\
      bin>galago.bat batch-search --index=F:\Fall2017\InformationRetreival\Assignment4\index.
      idx --count=10 F:\Fall2017\InformationRetreival\Assignment4\onequery.xml > F:\Fall2017\
      InformationRetreival\Assignment4\cacm_onequery.list
```

The below list file was generated by batch search from galago.

6 Q0 CACM-0695 1 -164.66490173 galago

6 Q0 CACM-2826 2 -166.82664490 galago

6 Q0 CACM-2828 3 -167.13014221 galago

```
6 Q0 CACM-1664 4 -167.29315186 galago

6 Q0 CACM-1543 5 -167.67100525 galago

6 Q0 CACM-2078 6 -168.12095642 galago

6 Q0 CACM-2176 7 -168.47441101 galago

6 Q0 CACM-1113 8 -169.13491821 galago

6 Q0 CACM-0605 9 -169.25828552 galago

6 Q0 CACM-1517 10 -169.68901062 galago
```

The below commands were run on galago to evaluate the ranked list for query generated from the previous step.

```
1  F:\Fall2017\InformationRetreival\Assignment3\Galgo\galagosearch-1.04-bin\galagosearch-1.04\
     bin> galago.bat eval F:\Fall2017\InformationRetreival\Assignment4\cacm_onequery.list  F
     :\Fall2017\InformationRetreival\Assignment4\cacm.rel >F:\Fall2017\InformationRetreiva\
     Assignment4\onequeryoutput.txt
```

The below text file is the result of eval command by using the relevance judgments file provided in cacm test collections.

```
num_ret          6 10

num_rel          6 3

num_rel_ret      6 3

map              6 0.4111

ndcg             6 0.5833

ndcg15           6 0.5833

R-prec           6 0.3333
```

```
bpref              6 0.0000

recip_rank         6 0.3333

P5                 6 0.4000

P10                6 0.3000

P15                6 0.2000

P20                6 0.1500

P30                6 0.1000

P100               6 0.0300

P200               6 0.0150

P500               6 0.0060

P1000              6 0.0030

num_ret          all 10

num_rel          all 3

num_rel_ret      all 3

map              all 0.4111

ndcg             all 0.5833

ndcg15           all 0.5833

R-prec           all 0.3333

bpref            all 0.0000

recip_rank       all 0.3333

P5               all 0.4000
```

```
P10           all 0.3000

P15           all 0.2000

P20           all 0.1500

P30           all 0.1000

P100          all 0.0300

P200          all 0.0150

P500          all 0.0060

P1000         all 0.0030
```

The calculation of precision for the ranking was done with the help of relevance judgements provided with the test collections. It contains list of all the documents relevant to a collection. Below is the list of relevant documents for query 6.

```
6 Q0 CACM-1543 1

6 Q0 CACM-2078 1

6 Q0 CACM-2828 1
```

Precision at 10

$Precision(1) = 0/1 = 0$

$Precision(2) = 0/2 = 0$

$Precision(3) = 1/3 = 0.33$

$Precision(4) = 1/4 = 0.25$

$Precision(5) = 2/5 = 0.4$

$Precision(6) = 3/6 = 0.5$

$Precision(7) = 3/7 = 0.429$

$Precision(8) = 3/8 = 0.375$

$Precision(9) = 3/9 = 0.33$

$Precision(10) = 3/10 = 0.3$

where Precision(6) denotes precision at search result 6.

The Precision(10) for this query is 0.3.

Mean Average Precision

$MAP = 1/3(1/3 + 2/5 + 3/6)$

$MAP = 0.41$

Reciprocal Rank

The first relevant document is at position 3.

$RR = 1/3$

$RR = 0.333$

Normalized Discounted Cummulative Gain at 5

Relevance Score: 0, 0, 1, 0, 1

$CG_5 = 0 + 0 + 1 + 0 + 1$

Table 1.1: Discounted Cummulative Gain

i	rel_i	$log_2(i+1)$	$rel_i/log_2(i+1)$
1	0	1	0
2	0	1.585	0
3	1	2	0.5
4	0	2.322	0
5	1	2.585	0.387

$CG_5 = 2$

$DCG = \sum\limits_{i=1}^{5} rel_i/log_2(i+1)$

$DCG = 0 + 0 + 0.5 + 0 + 0.387 = 0.887$

Ideal Discounted Cummulative Gain has the document and relevance score as below:

Relevance Score: 1, 1, 0, 0, 0

$CG_5 = 1 + 1 + 0 + 0 + 0$

$CG_5 = 2$

$IDCG = \sum\limits_{i=1}^{5} rel_i/log_2(i+1)$

$IDCG = 1 + 0.631 + 0 + 0 + 0 = 1.631$

Normalized Discounted Cummulative Gain (NDCG)= DCG / IDCG

$NDCG = 0.887/1.631 = 0.544$

Table 1.2: Ideal Discounted Cummulative Gain

i	rel_i	$log_2(i+1)$	$rel_i/log_2(i+1)$
1	1	1	1
2	1	1.585	0.631
3	0	2	0
4	0	2.322	0
5	0	2.585	0

Normalized Discounted Cummulative Gain at 10

Relevance Score: 0, 0, 1, 0, 1, 1, 0, 0, 0, 0

$CG_{10} = 0 + 0 + 1 + 0 + 1 + 1 + 0 + 0 + 0 + 0$

$CG_{10} = 3$

$$DCG = \sum_{i=1}^{10} rel_i/log_2(i+1)$$

$DCG = 0 + 0 + 0.5 + 0 + 0.387 + 0.356 + 0 + 0 + 0 + 0 = 1.243$

Ideal Discounted Cummulative Gain has the document and relevance score as below:

Relevance Score: 1, 1, 1, 0, 0, 0, 0, 0 , 0 , 0

$CG_{10} = 1 + 1 + 1 + 0 + 0 + 0 + 0 + 0 + 0 + 0$

$CG_{10} = 3$

$$IDCG = \sum_{i=1}^{10} rel_i/log_2(i+1)$$

$IDCG = 1 + 0.61 + 1 + 0.5 + 0 + 0 + 0 + 0 + 0 + 0 = 2.11$

Normalized Discounted Cummulative Gain (NDCG)= DCG / IDCG

$NDCG = 1.243/2.11 = 0.589$

Table 1.3: Discounted Cummulative Gain

i	rel_i	$log_2(i+1)$	$rel_i/log_2(i+1)$
1	0	1	0
2	0	1.585	0
3	1	2	0.5
4	0	2.322	0
5	1	2.585	0.387
6	1	2.807	0.356
7	0	3	1
8	0	3.17	0
9	0	3.22	0
10	0	3.459	0

Table 1.4: Ideal Discounted Cummulative Gain

i	rel_i	$log_2(i+1)$	$rel_i/log_2(i+1)$
1	1	1	1
2	1	1.585	0.61
3	1	2	0.5
4	0	2.322	0
5	0	2.585	0
6	0	2.807	0
7	0	3	0
8	0	3.17	0
9	0	3.22	0
10	0	3.459	0

Chapter 2

Problem 8.4

2.1 Problem Statement

For two queries in the CACM collection, generate two uninterpolated recall precision graphs,

a table of interpolated precision values at standard recall levels, and the average interpolated

recall-precision graph.

2.2 Solution

This problem uses CACM corpus provided in the test collection of the book. The queries

ran against the CACM collection have been used from the processed queries section of the

CACM test collection.

Table 2.1: Precision and Recall Table for Query 6

Index	1	2	3	4	5	6	7	8	9	10
Relevant	no	no	yes	no	yes	yes	no	no	no	no
Recall	0	0	0.33	0.33	0.67	1	1	1	1	1
Precision	0	0	0.33	0.25	0.4	0.5	0.429	0.375	0.33	0.3

2.2.1 Query: Interested in articles on robotics motion planning particularly the geometric and combinatorial aspects we are not interested in the dynamics of arm motion

The below list file generated from batch search of galago has ranked result of 10 for the query.

```
6 Q0 CACM-0695 1 -164.66490173 galago

6 Q0 CACM-2826 2 -166.82664490 galago

6 Q0 CACM-2828 3 -167.13014221 galago

6 Q0 CACM-1664 4 -167.29315186 galago

6 Q0 CACM-1543 5 -167.67100525 galago

6 Q0 CACM-2078 6 -168.12095642 galago

6 Q0 CACM-2176 7 -168.47441101 galago

6 Q0 CACM-1113 8 -169.13491821 galago

6 Q0 CACM-0605 9 -169.25828552 galago

6 Q0 CACM-1517 10 -169.68901062 galago
```

Table 2.2: Precision and Recall Table for Query: Parallel algorithms

Index	1	2	3	4	5	6	7	8	9	10
Relevant	no	no	yes	no	no	yes	no	yes	yes	yes
Recall	0	0	0.2	0.2	0.2	0.4	0.4	0.6	0.8	1
Precision	0	0	0.33	0.25	0.2	0.33	0.286	0.375	0.44	0.5

2.2.2 Query: Parallel algorithms

The below list file generated from batch search of galago has ranked result of 10 for the query.

```
19 Q0 CACM-2433 1 -10.82248116 galago

19 Q0 CACM-1811 2 -10.83561993 galago

19 Q0 CACM-2973 3 -10.86207390 galago

19 Q0 CACM-2289 4 -10.93358135 galago

19 Q0 CACM-0950 5 -10.96933460 galago

19 Q0 CACM-2266 6 -11.00977135 galago

19 Q0 CACM-2785 7 -11.07456684 galago

19 Q0 CACM-3075 8 -11.07782173 galago

19 Q0 CACM-3156 9 -11.13305855 galago

19 Q0 CACM-2664 10 -11.15161610 galago
```

Both the queries show their highest precision of 0.5 at recall value of 1. This results in a interpolated recall-precision graph to be a straight horizontal line at precision 0.5. Interpolated recall-precision graph is drawn with curves that are descending in manner,
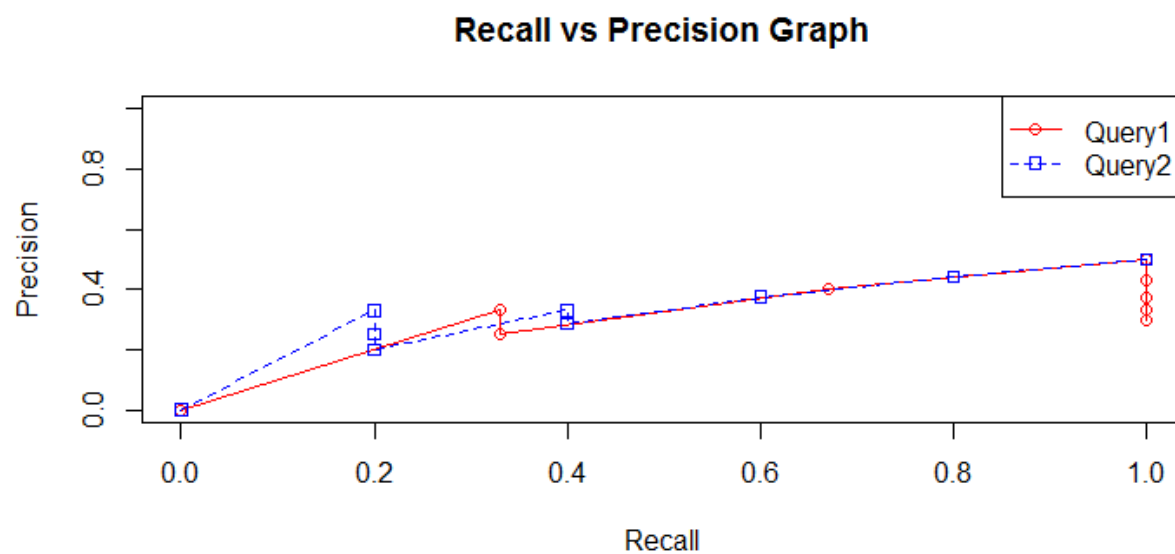
Recall vs Precision Graph

Figure 2.1: Recall vs Precision Graph

starting from highest point towards to lowest. But, for both the queries highest precision

was reached at recall 1leading to a straight line.

Table 2.3: Precision values at standard recall levels calculated using interpolation

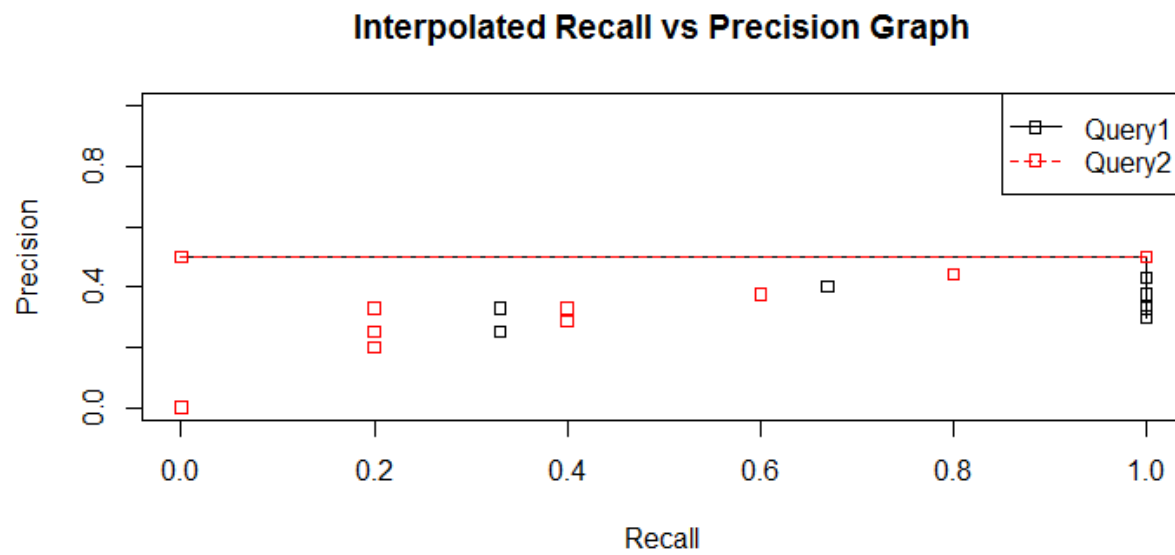| Recall	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Ranking 1	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
Ranking 2	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
Average Ranking	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

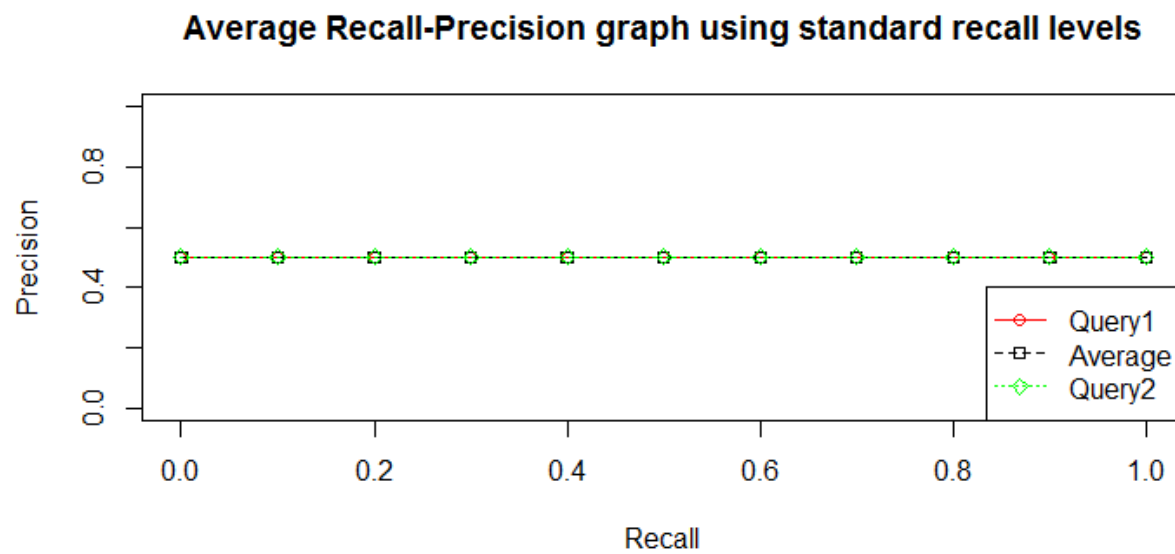Figure 2.2: Interpolated Recall vs Precision Graph

Figure 2.3: Average recall-precision graph using standard recall levels

Chapter 3

Problem 8.5

3.1   Problem

Generate the mean average precision, recall-precision graph, average NDCG at 5 and 10, and precision at 10 for the entire CACM query set.

3.2   Solution

The results have been calculated by galago batch-search for rank result of 10. Galago provides NDCG 5 and 15 but not NDCG10. So, by making a request for 10 results per query the NDCG15 is same as NDCG10 beacause all the calculations are based on 10 results for each query.

The values for MAP is lesser than Precision at 10 for the CACM query set. Th query sets having most of their relevant results at top indexes (between 1 and 5) have higher or

equal NCDG5 and NCDG10, but for queries with most of their relevant documents lower in

the rank (between 6 and 10) have higher NCDG10 than NCDG5.
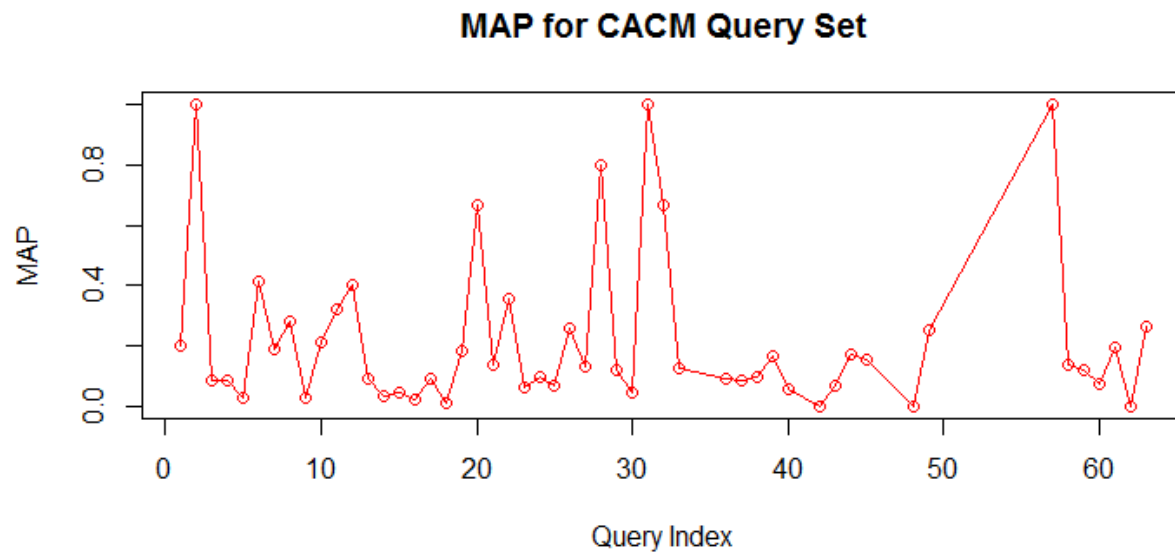
MAP for CACM Query Set
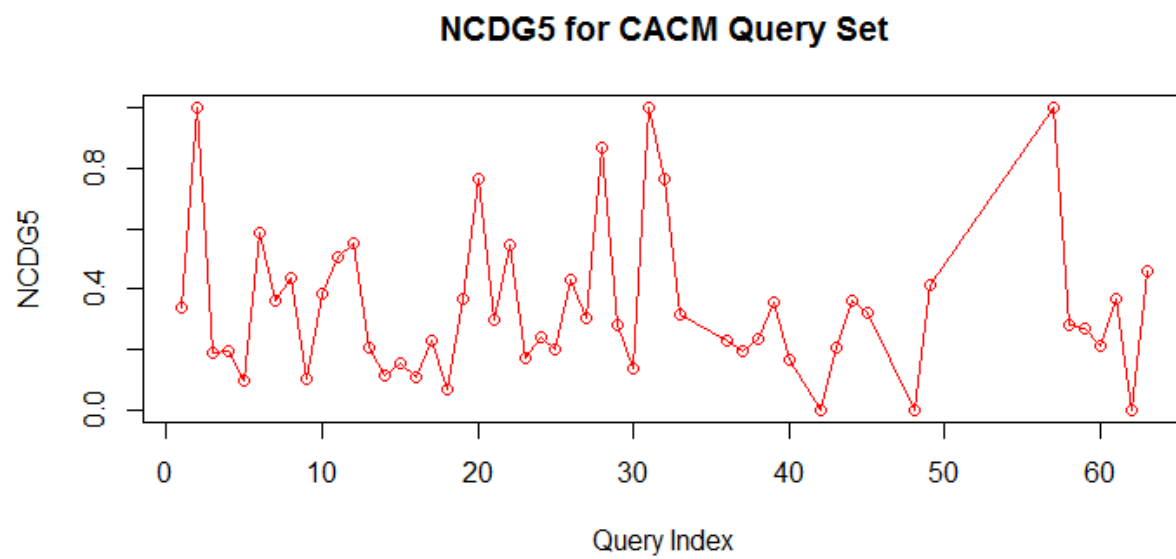

Figure 3.1: MAP values for CACM Query Set
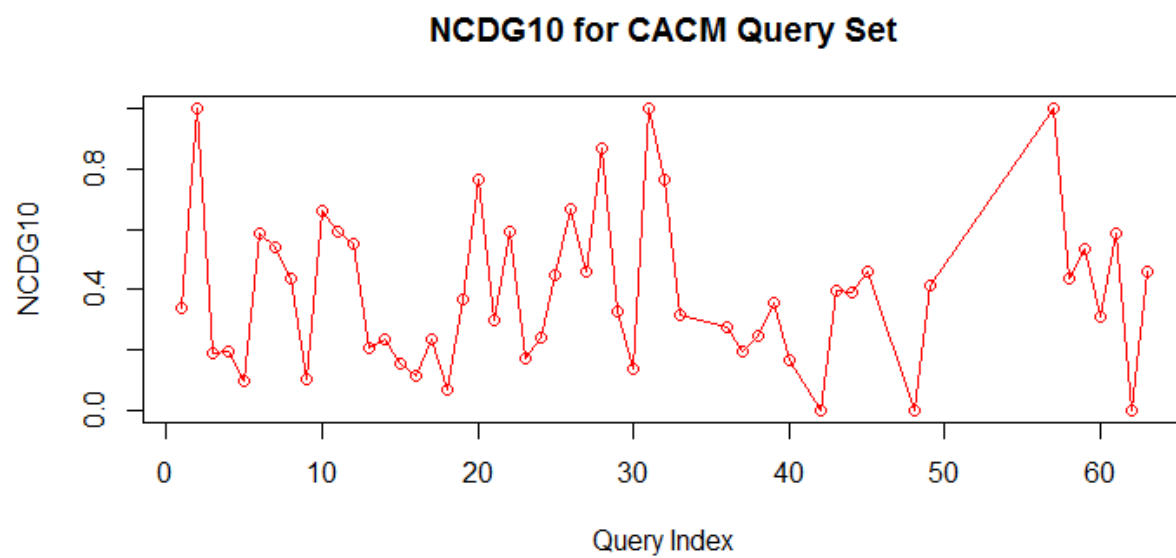
Figure 3.2: NCDG5 values for CACM Query Set

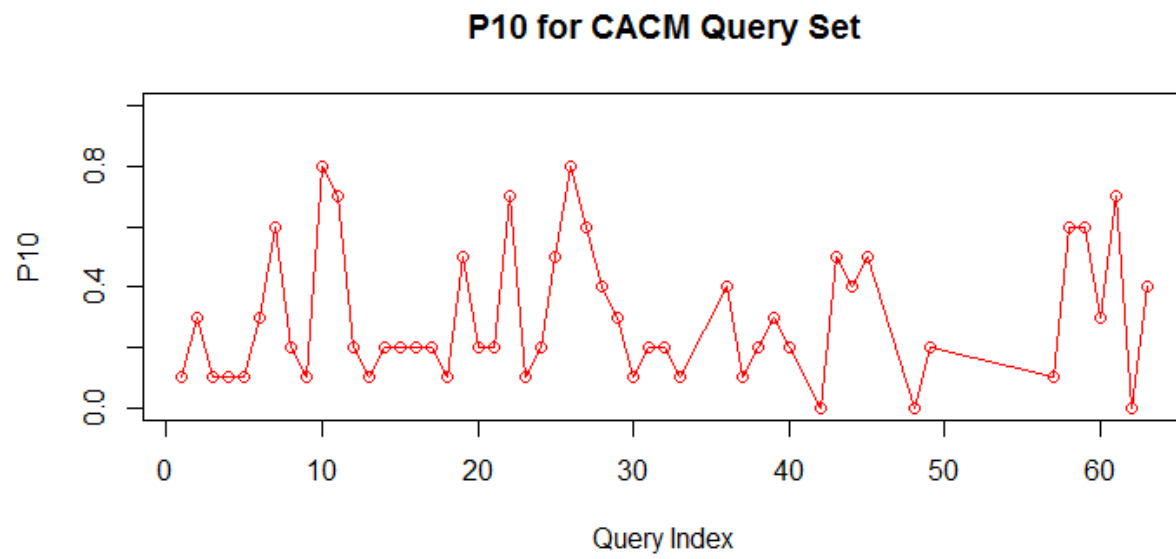Figure 3.3: NCDG10 values for CACM Query Set
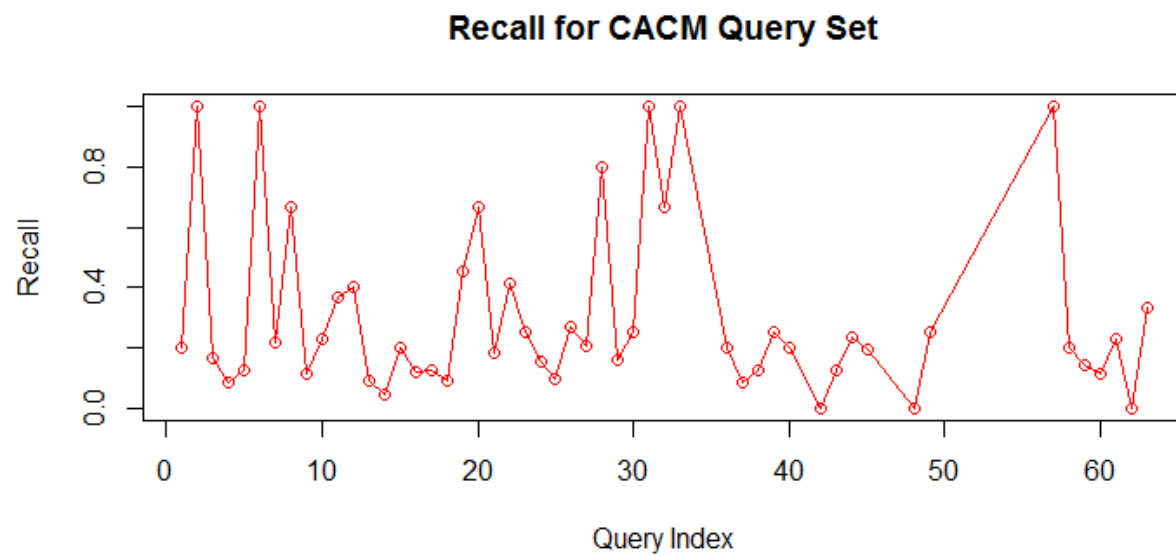
Figure 3.4: Precision at 10 values for CACM Query Set
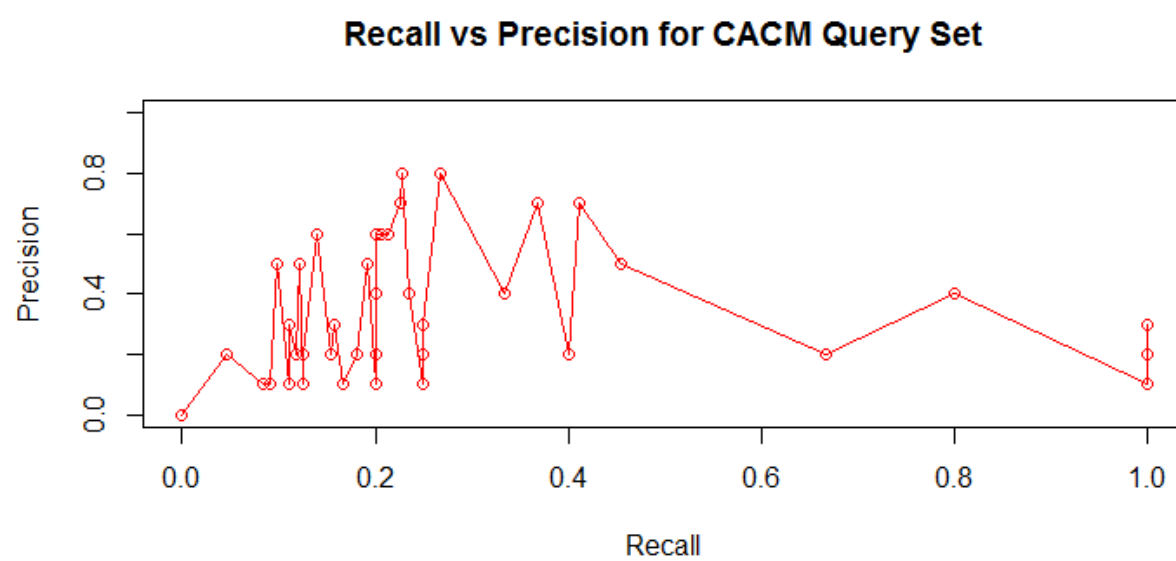

Figure 3.5: Recall values for CACM Query Set

22

Figure 3.6: Recall vs Precision for CACM Query Set

Chapter 4

Problem 8.7

4.1   Problem

Another measure that has been used in a number of evaluations is R-precision. This is defined

as the precision at R documents, where R is the number of relevant documents for a query.

It is used in situations where there is a large variation in the number of relevant documents

per query. Calculate the average R-precision for the CACM query set and compare it to the

other measures.

4.2   Solution

R-precision is defined as precision at R documents, where R is the number of relevant docu-

ments for a query. In context to the CACM query set, the relevance judgements file suggests

that all the queries have varied number of relevant documents. It ranges from 0 to 51. So,

Table 4.1: CACM Query Set measured against different measures

map	0.2840
ndcg	0.4419
ndcg15	0.4734
R-prec	0.3192
bpref	0.0000
recip_rank	0.7296
P5	0.3922
P10	0.2980
P15	0.2614
P20	0.2373
P30	0.1843

for calculating R-precision number of relevant documents per query is set to 51 and evaluated in the galago. The results for the different measures inclusive of R-precision at relevant document 51 are ahown in the table:

Chapter 5

Problem 8.9

5.1   Problem

For one query in the CACM collection, generate a ranking and calculate BPREF. Show that the two formulations of BPREF give the same value.

5.2   Solution

The solution is based on relevance results for query, interested in articles on robotics motion planning particularly the geometric and combinatorial aspects we are not interested in the dynamics of arm motion . The Table 5.1 shows the relevance results for the query. The relevance results show the search results had 3 relevant documents and 7 non- relevant documents.

```
6 Q0 CACM-0695 1 -164.66490173 galago
```

```
6 Q0 CACM-2826 2 -166.82664490 galago

6 Q0 CACM-2828 3 -167.13014221 galago

6 Q0 CACM-1664 4 -167.29315186 galago

6 Q0 CACM-1543 5 -167.67100525 galago

6 Q0 CACM-2078 6 -168.12095642 galago

6 Q0 CACM-2176 7 -168.47441101 galago

6 Q0 CACM-1113 8 -169.13491821 galago

6 Q0 CACM-0605 9 -169.25828552 galago

6 Q0 CACM-1517 10 -169.68901062 galago
```

5.2.1 Calculating BPREF

$$BPREF = 1/R \sum_{d_r}(1 - (N_{d_r}/min(R, N)))$$

where R is the number of relevant documents that are considered

N is the number of non-relevant documents that are considered

d_r is the number of relevant documents

For a query result with 3 relevant documents R is 3 implying that first 3 non-relevant documents are considered. The relevance table for the query is manipulated for BPREF as,

$$BPREF = 1/R \sum_{d_r}(1 - (N_{d_r}/min(R, N)))$$

$$BPREF = 1/3[(1 - 2/3) + (1 - 3/3) + (1 - 3/3)]$$

Table 5.1: Relevance table showing R relevant and non-relevant documents

Index	Relevance
1	No
2	No
3	Yes
4	No
5	Yes
6	Yes

$BPREF = 1/3[(1/3) + 0 + 0] = 3/8 = 0.11$

5.2.2 Calculating BPREF based on preference

$BPREF = P/(P + Q)$

where P is the number of relevant documents

Q is the number of non-relevant documents

For query: Code optimization for space efficiency

P = 3

Q = 3

$BPREF = 3/(3 + 3)$

$BPREF = 0.5$

The value of BPREF is 0.11 and 0.3 by computing respectively with relevant documents

formula and the clickthrough preference formula. The values computed are not equal because the bpref for clickthough formula does not take into account the ranking of relevant results and the value for BPREF with relevant documents formula dips due to the fist relevant result showing aftr two non-relevant result and the rest two relevant result appear after all the non-relevant documents taken in consideration. Thus, the last wo relevant documents do not add any value to the BPREF.

Chapter 6

Problem 9.8

6.1 Problem

Cluster the following set of two-dimensional instances into three clusters using each of the five agglomerative clustering methods: (4, 2), (3, 2), (2, 2), (1, 2), (1, 1), (1, 1), (2, 3), (3, 2), (3, 4), (4, 3) Discuss the differences in the clusters across methods. Which methods produce the same clusters? How do these clusters compare to how you would manually cluster the points?

6.2 Solution

The code for agglomerate clustering methods generates 3 clusters for the data set.

6.2.1 Discussion of the Methods

Single Linkage: It uses the minimum distance between between elements of two clusters to merge them as one cluster. For two clusters A and B, it finds the minimum euclidean distance between the points of each cluster and compares them against the minimum threshold distance of all the other clusters to merge them in one cluster. The disadvantage of this approach is that it does not consider how far spread each cluster is and focusing only on the minimum distance between the clusters to merge them.

Complete Linkage: It uses the maximum distance between elements of two clusters to merge them as one cluster. For two clusters A and B, it finds the maximum euclidean distance between the points of each cluster and compares them against the minimum distance of all the other clusters to merge them in one cluster. This approach creates a more comapact and less spread cluster than single linkage clustering technique.

Average Clustering: It uses average distance of all the elements between two clusters to merge them as one cluster. For two clusters A and B, it finds the average distance by calculating the euclidean distance between all the points in each cluster and dividing them by the number of elements in each cluster. The average distance calculated is compared against average distance of other clusters to merge the minimum average distance clusters in to one cluster. The type of cluster formed by average linkage depend heavily on the structure of clusters, since it is based on the average distance between all the elements in the two clusters.

Average Group Clustering: It uses centroid distance between teo clusters to merge them as one cluster. For two clusters A and B, it finds the centroid of the two clusters and merges them together by comparing against the centroid distances of other clusters. It forms similar clusters to the average linkage clusters.

Ward's Method: It uses sum of variance between two clusters to merge them as one cluster.It forms minimum spread clusters around the centroid of the cluster.

```python
1  '''
2  Created on Nov 22, 2017
3
4  @author: nauman
5  '''
6  import math
7
8  def singleLinkage(clusterPointA, clusterPointB):
9      minDistance = math.inf
10     for i in range(0, len(clusterPointA)):
11         for j in range(0,len(clusterPointB)):
12             distance = math.sqrt(math.pow((clusterPointA[i][0] - clusterPointB[j][0]),2) + math.pow((clusterPointA[i][1] - clusterPointB[j][1]),2))
13             if minDistance > distance:
14                 minDistance = distance
```

```python
15    return minDistance

16

17 def agglomerativeSingleLinkageCluster(clusterValue, clusterPoints):

18

19    distanceMatrix = []

20    finalClusters = []

21

22    for i in range(0, len(clusterPoints)):

23        finalClusters.append([i])

24

25    for c in range(len(clusterPoints), clusterValue, -1):

26        bestClusterA = []

27        bestClusterB  = []

28        bestCost = math.inf

29        for i in range(0, len(clusterPoints)):

30            temp = []

31            for j in range(0, len(clusterPoints)):

32                temp.append(0)

33            distanceMatrix.append(temp)

34

35        for i in range(0, len(clusterPoints)):

36            for j in range((i+1), len(clusterPoints)):

37                distance = singleLinkage(clusterPoints[i], clusterPoints[j])

38                distanceMatrix[i][j] = (distance)

39                if bestCost > distance:

40                    bestCost = distance
```

33

```python
                    if bestClusterA and bestClusterB:
                        bestClusterA.pop()
                        bestClusterB.pop()
                    bestClusterA.append(clusterPoints[i])
                    bestClusterB.append(clusterPoints[j])
        clusterPoints.remove(bestClusterB[0])
        index = clusterPoints.index(bestClusterA[0])
        for x in range(0,len(bestClusterB[0])):
            clusterPoints[index].append(bestClusterB[0][x])
    return (clusterPoints)


def completeLinkage(clusterPointA, clusterPointB):
    maxDistance = 0
    for i in range(0, len(clusterPointA)):
        for j in range(0,len(clusterPointB)):
            distance = math.sqrt(math.pow((clusterPointA[i][0]-clusterPointB[j
    ][0]),2) + math.pow((clusterPointA[i][1]-clusterPointB[j][1]),2))
            if maxDistance < distance:
                maxDistance = distance
    return maxDistance


def agglomerativeCompleteLinkageCluster(clusterValue, input):

    distanceMatrix = []
    finalClusters = []
```

```python
66        clusterPoints = input

67        for i in range(0, len(clusterPoints)):

68            finalClusters.append([i])

69

70        for c in range(len(clusterPoints), clusterValue, -1):

71            bestClusterA = []

72            bestClusterB  = []

73            bestCost = 0

74            for i in range(0, len(clusterPoints)):

75                temp = []

76                for j in range(0, len(clusterPoints)):

77                    temp.append(0)

78                distanceMatrix.append(temp)

79

80            for i in range(0, len(clusterPoints)):

81                for j in range((i+1), len(clusterPoints)):

82                    distance = completeLinkage(clusterPoints[i], clusterPoints[j])

83                    distanceMatrix[i][j] = (distance)

84                    if bestCost < distance:

85                        bestCost = distance

86                        if bestClusterA and bestClusterB:

87                            bestClusterA.pop()

88                            bestClusterB.pop()

89                        bestClusterA.append(clusterPoints[i])

90                        bestClusterB.append(clusterPoints[j])

91            clusterPoints.remove(bestClusterB[0])
```

```
92          index = clusterPoints.index(bestClusterA[0])

93          for x in range(0,len(bestClusterB[0])):

94              clusterPoints[index].append(bestClusterB[0][x])

95      return (clusterPoints)

96

97  def averageLinkage(clusterPointA, clusterPointB):

98      averageDistance = 0

99      for i in range(0, len(clusterPointA)):

100         for j in range(0,len(clusterPointB)):

101             averageDistance = averageDistance +  math.sqrt(math.pow((

     clusterPointA[i][0]-clusterPointB[j][0]),2) + math.pow((clusterPointA[i

     ][1]-clusterPointB[j][1]),2))

102     averageDistance = averageDistance/(len(clusterPointA) +len(clusterPointB))

103     return averageDistance

104

105

106 def agglomerativeAverageLinkageCluster(clusterValue, clusterPoints):

107

108     distanceMatrix = []

109     finalClusters = []

110

111     for i in range(0,len(clusterPoints)):

112         finalClusters.append([i])

113

114     for c in range(len(clusterPoints), clusterValue, -1):

115         bestClusterA = []
```

```python
116        bestClusterB   = []

117        bestCost = math.inf

118        for i in range(0,len(clusterPoints)):

119            temp = []

120            for j in range(0,len(clusterPoints)):

121                temp.append(0)

122            distanceMatrix.append(temp)


124        for i in range(0,len(clusterPoints)):

125            for j in range((i+1),len(clusterPoints)):

126                distance = averageLinkage(clusterPoints[i], clusterPoints[j])

127                distanceMatrix[i][j] = (distance)

128                if bestCost > distance:

129                    bestCost = distance

130                    if bestClusterA and bestClusterB:

131                        bestClusterA.pop()

132                        bestClusterB.pop()

133                    bestClusterA.append(clusterPoints[i])

134                    bestClusterB.append(clusterPoints[j])

135        clusterPoints.remove(bestClusterB[0])

136        index = clusterPoints.index(bestClusterA[0])

137        for x in range(0,len(bestClusterB[0])):

138            clusterPoints[index].append(bestClusterB[0][x])

139    return (clusterPoints)


141 def averageGroupLinkage(clusterPointA, clusterPointB):
```

```python
142     centroidX_A = 0

143     centroidY_A = 0

144     centroidX_B = 0

145     centroidY_B = 0

146     centroidA = []

147     centroidB = []

148     for i in range(0, len(clusterPointA)):

149         centroidX_A = centroidX_A + clusterPointA[i][0]

150         centroidY_A = centroidY_A + clusterPointA[i][1]

151     centroidA.append(centroidX_A/(len(clusterPointA)))

152     centroidA.append(centroidY_A/(len(clusterPointA)))

153     for j in range(0, len(clusterPointB)):

154         centroidX_B = centroidX_B + clusterPointB[j][0]

155         centroidY_B = centroidY_B + clusterPointB[j][1]

156     centroidB.append(centroidX_B/(len(clusterPointB)))

157     centroidB.append(centroidY_B/(len(clusterPointB)))

158     clusterDistance =    math.sqrt(math.pow((centroidA[0]-centroidB[0]),2) +
    math.pow((centroidA[1]-centroidB[1]),2))

159     return clusterDistance

160

161

162 def agglomerativeAverageGroupLinkageCluster(clusterValue, clusterPoints):

163

164     distanceMatrix = []

165     finalClusters = []

166
```

```python
167        for i in range(0,len(clusterPoints)):
168            finalClusters.append([i])

169

170        for c in range(len(clusterPoints), clusterValue, -1):
171            bestClusterA = []
172            bestClusterB  = []
173            bestCost = math.inf
174            for i in range(0,len(clusterPoints)):
175                temp = []
176                for j in range(0,len(clusterPoints)):
177                    temp.append(0)
178                distanceMatrix.append(temp)

179

180            for i in range(0,len(clusterPoints)):
181                for j in range((i+1),len(clusterPoints)):
182                    distance = averageGroupLinkage(clusterPoints[i], clusterPoints
     [j])
183                    distanceMatrix[i][j] = (distance)
184                    if bestCost > distance:
185                        bestCost = distance
186                        if bestClusterA and bestClusterB:
187                            bestClusterA.pop()
188                            bestClusterB.pop()
189                        bestClusterA.append(clusterPoints[i])
190                        bestClusterB.append(clusterPoints[j])
191            clusterPoints.remove(bestClusterB[0])
```

```python
192            index = clusterPoints.index(bestClusterA[0])

193            for x in range(0,len(bestClusterB[0])):

194                clusterPoints[index].append(bestClusterB[0][x])

195        return (clusterPoints)

196

197 def wardMethod(clusterPointA, clusterPointB):

198        centroidX_A = 0

199        centroidY_A = 0

200        centroidX_B = 0

201        centroidY_B = 0

202        centroidA = []

203        centroidB = []

204        for i in range(0, len(clusterPointA)):

205            centroidX_A = centroidX_A + clusterPointA[i][0]

206            centroidY_A = centroidY_A + clusterPointA[i][1]

207        centroidA.append(centroidX_A/(len(clusterPointA)))

208        centroidA.append(centroidY_A/(len(clusterPointA)))

209        for j in range(0,len(clusterPointB)):

210            centroidX_B = centroidX_B + clusterPointB[j][0]

211            centroidY_B = centroidY_B + clusterPointB[j][1]

212        centroidB.append(centroidX_B/(len(clusterPointB)))

213        centroidB.append(centroidY_B/(len(clusterPointB)))

214        clusterDistance =   math.sqrt(math.pow((centroidA[0]-centroidB[0]),2) +
       math.pow((centroidA[1]-centroidB[1]),2))

215        variance = len(clusterPointA)* len(clusterPointB)* clusterDistance/ len(
       clusterPointA)+ len(clusterPointB)
```

```python
216     return variance
217
218
219 def agglomerativeWardMethod(clusterValue, clusterPoints):
220
221     distanceMatrix = []
222     finalClusters = []
223
224     for i in range(0, len(clusterPoints)):
225         finalClusters.append([i])
226
227     for c in range(len(clusterPoints), clusterValue, -1):
228         bestClusterA = []
229         bestClusterB  = []
230         bestCost = math.inf
231         for i in range(0, len(clusterPoints)):
232             temp = []
233             for j in range(0, len(clusterPoints)):
234                 temp.append(0)
235             distanceMatrix.append(temp)
236
237         for i in range(0, len(clusterPoints)):
238             for j in range((i+1), len(clusterPoints)):
239                 distance = wardMethod(clusterPoints[i], clusterPoints[j])
240                 distanceMatrix[i][j] = (distance)
241                 if bestCost > distance:
```

41

```
242                      bestCost = distance
243                    if bestClusterA and bestClusterB:
244                        bestClusterA.pop()
245                        bestClusterB.pop()
246                    bestClusterA.append(clusterPoints[i])
247                    bestClusterB.append(clusterPoints[j])
248        clusterPoints.remove(bestClusterB[0])
249        index = clusterPoints.index(bestClusterA[0])
250        for x in range(0,len(bestClusterB[0])):
251            clusterPoints[index].append(bestClusterB[0][x])
252    return (clusterPoints)
253
254 def main():
255    clusters = 3
256    input = [[[-4, -2]], [[-3, -2]], [[-2, -2]], [[-1, -2]], [[1, -1]], [[1,
       1]], [[2, 3]], [[3, 2]], [[3, 4]], [[4,3]]]
257    file = open("ClusteringOutput.txt", "w")
258    clusterPointsSingle = agglomerativeSingleLinkageCluster(clusters,input)
259    file.write("Single Linkage" + "\n")
260    for i in range(0, len(clusterPointsSingle)):
261        file.write("Cluster " + str(i+1) + ": " +str(clusterPointsSingle[i]) +
       "\n")
262    inputComplete =[[[-4, -2]], [[-3, -2]], [[-2, -2]], [[-1, -2]], [[1, -1]],
        [[1, 1]], [[2, 3]], [[3, 2]], [[3, 4]], [[4,3]]]
263    clusterPointsComplete = agglomerativeCompleteLinkageCluster(clusters,
       inputComplete)
```

```python
264     file.write("\nComplete Linkage\n")

265     for i in range(0, len(clusterPointsComplete)):

266         file.write("Cluster " + str(i+1) + ": " +str(clusterPointsComplete[i])
        + "\n")

267     inputAverage = [[[-4, -2]], [[-3, -2]], [[-2, -2]], [[-1, -2]], [[1, -1]],
        [[1, 1]], [[2, 3]], [[3, 2]], [[3, 4]], [[4,3]]]

268     clusterPointsAverage = agglomerativeAverageLinkageCluster(clusters,
        inputAverage)

269     file.write("\nAverage Linkage\n")

270     for i in range(0, len(clusterPointsAverage)):

271         file.write("Cluster " + str(i+1) + ": " +str(clusterPointsAverage[i])+
        "\n")

272     inputAverageGroup = [[[-4, -2]], [[-3, -2]], [[-2, -2]], [[-1, -2]], [[1,
        -1]], [[1, 1]], [[2, 3]], [[3, 2]], [[3, 4]], [[4,3]]]

273     clusterPointsAverageGroup = agglomerativeAverageGroupLinkageCluster(
        clusters, inputAverageGroup)

274     file.write("\nAverage Group Linkage\n")

275     for i in range(0, len(clusterPointsAverageGroup)):

276         file.write("Cluster " + str(i+1) + ": " +str(clusterPointsAverageGroup
        [i])+ "\n")

277     inputWard = [[[-4, -2]], [[-3, -2]], [[-2, -2]], [[-1, -2]], [[1, -1]],
        [[1, 1]], [[2, 3]], [[3, 2]], [[3, 4]], [[4,3]]]

278     clusterPointsWard = agglomerativeWardMethod(clusters, inputWard)

279     file.write("\nWard's Algorithm Linkage\n")

280     for i in range(0, len(clusterPointsWard)):
```

```
281        file.write("Cluster " + str(i+1) + ": " +str(clusterPointsWard[i]) + "
      \n")
282    file.close()
283
284 main()
```

Single Linkage

Cluster 1: [[-4, -2], [-3, -2], [-2, -2], [-1, -2]]

Cluster 2: [[1, -1], [1, 1]]

Cluster 3: [[2, 3], [3, 2], [3, 4], [4, 3]]

Complete Linkage

Cluster 1: [[-4, -2], [4, 3], [3, 4], [-3, -2], [3, 2], [-2, -2], [2, 3], [-1, -2]]

Cluster 2: [[1, -1]]

Cluster 3: [[1, 1]]

Average Linkage

Cluster 1: [[-4, -2], [-3, -2], [-2, -2], [-1, -2]]

Cluster 2: [[1, -1], [1, 1]]

Cluster 3: [[2, 3], [3, 2], [3, 4], [4, 3]]

Average Group Linkage

Cluster 1: [[-4, -2], [-3, -2], [-2, -2], [-1, -2]]

```
Cluster 2: [[1, -1], [1, 1]]

Cluster 3: [[2, 3], [3, 2], [3, 4], [4, 3]]


Ward's Algorithm Linkage

Cluster 1: [[-4, -2], [-3, -2], [-2, -2], [-1, -2]]

Cluster 2: [[1, -1], [1, 1]]

Cluster 3: [[2, 3], [3, 2], [3, 4], [4, 3]]
```

6.2.2 Clustering Results vs Mannual Clustering

All the agglomerate clustering techniques except for complete linkage technique produced the same result for the provided dataset. The results of the clusters formed by all the clustering methods has been shown above.

Mannual Clustering of the data points on euclidean distance results in the same clusters generated by agglomerative clustering.

Cluster 1: (-4,-2), (-3,-2), (−2,-2), (-1,-2)

Cluster 2: (1,-1), (1,1)

Cluster 3: (2,3), (3,2), (3,4), (4,3)

Cluster 1 is easy to create due to it being far away from other data points. Cluster 2 and 3 have a close margin where the distance between point (1,1) and (-1,1) is equal to 2 and the distance between point (1,1) and (2,3) is $\sqrt{5}$. Therefore the points (1,1) and (-1,1) have been clustered together as Cluster 2.

If the data points are clustered on the basis of the quadrants they lie in:

Cluster 1: 1st Quadrant$->$ (-4,-2), (-3,-2), (-2,-2), (-1,-2)

Cluster 2: 3rd Quadrant$->$ (1,-1)

Cluster 3: 4th Quadrant$->$ (1,1), (2,3), (3,2), (3,4), (4,3)

Chapter 7

9.9

7.1 Problem

Use K-means and spherical K-means to cluster the data points in Exercise 9.8. How do the clusterings differ?

7.2 Solution

```python
'''
Created on Nov 24, 2017

@author: nauman
'''
import numpy as np
from sklearn.cluster import KMeans
```

```python
8  from spherecluster import SphericalKMeans

9

10 def kMeans(num):

11

12     file = open("Kmeans.txt", "a+")
13     input = np.array ([[-4, -2], [-3, -2], [-2, -2], [-1, -2], [1, -1], [1,
       1], [2, 3], [3, 2], [3, 4], [4,3]])
14     kmeans = KMeans(n_clusters=num, random_state=0).fit(input)
15     file.write("K means output for cluster size : "+ str(num) + "\n")
16     file.write("Clusters index of points" + "\n")
17     file.write(str(kmeans.labels_) + "\n")
18     file.write("Center of Clusters\n")
19     file.write(str(kmeans.cluster_centers_) + "\n")
20     file.close()

21

22 def sphericalKMeans(num):
23     num = 4
24     file = open("Kmeans.txt", "a+")
25     input = np.array ([[-4, -2], [-3, -2], [-2, -2], [-1, -2], [1, -1], [1,
       1], [2, 3], [3, 2], [3, 4], [4,3]])
26     kmeans = SphericalKMeans(n_clusters=num).fit(input)
27     file.write("Spherical K means output for cluster size : "+ str(num) + "\n"
       )
28     file.write("Clusters index of points" + "\n")
29     file.write(str(kmeans.labels_) + "\n")
30     file.write("Center of Clusters\n")
```

```
31      file.write(str(kmeans.cluster_centers_) + "\n")

32      file.close()

33

34

35  kMeans(4)

36  sphericalKMeans(4)
```

K means output for cluster size : 3

Clusters index of points

[2 2 2 2 0 0 1 1 1 1]

Center of Clusters

[[1.00000000e+00   5.55111512e-17]

 [3.00000000e+00   3.00000000e+00]

 [-2.50000000e+00   -2.00000000e+00]]

Spherical K means output for cluster size : 4

Clusters index of points

[3 3 3 1 2 0 0 0 0 0]

Center of Clusters

[[0.70710678 0.70710678]

 [-0.4472136 -0.89442719]

 [0.70710678 -0.70710678]

 [-0.81836024 -0.57470559]]

K means output for cluster size : 2

Clusters index of points

[0 0 0 0 0 1 1 1 1 1]

Center of Clusters

[[-1.8 -1.8]

 [2.6 2.6]]

Spherical K means output for cluster size : 4

Clusters index of points

[3 3 3 0 2 1 1 1 1 1]

Center of Clusters

[[-0.4472136 -0.89442719]

 [0.70710678 0.70710678]

 [0.70710678 -0.70710678]

 [-0.81836024 -0.57470559]]

K means output for cluster size : 4

Clusters index of points

[2 2 0 0 3 3 1 1 1 1]

Center of Clusters

[[-1.50000000e+00 -2.00000000e+00]

 [3.00000000e+00   3.00000000e+00]

 [-3.50000000e+00 -2.00000000e+00]

 [1.00000000e+00   5.55111512e-17]]

Table 7.1: Index to Data Point relation

Index	0	1	2	3	4	5	6	7	8	9
Data Point	(-4, -2)	(-3,-2)	(-2,-2)	(-1,-2)	(1,-1)	(1,1)	(2,3)	(3,2)	(3,4)	(4,3)

```
Spherical K means output for cluster size : 4

Clusters index of points

[3 3 3 1 2 0 0 0 0 0]

Center of Clusters

[[ 0.70710678  0.70710678]

 [-0.4472136  -0.89442719]

 [ 0.70710678 -0.70710678]

 [-0.81836024 -0.57470559]]
```

The output of the code shows clusters on the basis of the index of data points and center of each cluster.Table 5.1 shows the index to data point relationship.The results for cluster size 2,3 and 4 are show for both the clustering techniques.

 The clusters formed by K-means for cluster size 3 are same as the clusters formed by Single Linkage, Average Linkage and Average Group Linkage from the problem 9.8. The clusters formed by spherical K-means for cluster 3 is different from K-means due to the difference between finding similarity between points for clustering. K-means uses euclidean distance to find similarity while spherical K-mean uses cosine similarity to cluster items together. Spherical K-means clusters items on which fall in the ame quadrant approach discussed in the previous problem.

Chapter 8

Problem 9.11

8.1 Problem

The K nearest neighbors of a document could be represented by links to those documents.
Describe two ways this representation could be used in a search application.

8.2 Solution

The primary advantage of K nearest neighbor comapared to K means and agglomerative
clustering techniques is that a document can be present in multiple clusters unlike the K
means and agglomerative clustering technique where each documents is in a single cluster.

The K nearest neighbors of a documents can be used for tell-text search in search ap-
plication. All the documents in the corpus can be clustered by K nearest neighbour. The

document can be in multiple clusters based on the features extracted from the cluster. For a corpus of documents containing sports news from USA, it can be classified into multiple clusters as football, baseball, basketball, athletics, Olympics, CONCAF Cup etc. A document that is in cluster soccer can simultaneously be clustered with Olympics and CONCAF Cup. Similarly the queries to the documents can also be clustered on the basis of its features. It presents all the documents that are in the cluster soccer for a query of feature soccer. The same set of documents can also be recalled if the query is about CONCAF Cup. It allows for relevant search results.

The K nearest neighbors can also be used for showing clustered search results. It can be helped to refine a very generic query to a more precise query to return relevant result. For example, a system that employs K nearest neighbor clustering on its queries. For a user input query apple, it will show all the possible clusters where the query term apple appears. For apple, it can be a fruit, it can be the company Apple, it can be products of Apple Company etc. A user on the basis of suggestions can select the particular suggestion to find search results that fall under the specified feature cluster.