Assignment 4

Information Retrieval

CS 834

Fall 2017

Mohammed Nauman Sididque

November 24, 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.

Table 1.1: Relevance Table for Query:Code optimization for space efficiency

Index	Relevant	Relevance Score
1	Yes	3
2	No	1
3	No	1
4	Yes	2
5	No	0
6	No	0
7	Yes	3
8	No	0
9	No	0
10	Yes	2

1.2.1 Query: Code optimization for space efficiency

The table below has been created manuallyby looking at the search results of the query. The first column in the table is the position in the search result, the second column is whether the document is relevant or not and last column is the relevance score in range of 0 to 3 where 0 is for no relevance and 3 is assigned to document with high relevance.

Precision

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

$Precision(2) = 1/2 = 0.5$

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

$Precision(4) = 2/4 = 0.5$

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

$Precision(6) = 2/6 = 0.33$

$Precision(7) = 3/7 = 0.43$

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

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

$Precision(10) = 4/10 = 0.4$

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

The precision for this query is 0.4.

Reciprocal Rank

$MRR = 1/1 + 1/4 + 1/7 + 1/10$

$MRR = 1 + 0.25 + 0.143 + 0.1$

$MRR = 1.493$

MRR is the mean reciprocal rank for the query. For this query, the relevant results are present at position 1, 4, 7 and 10.

The MRR for the result is calculated to be 1.493.

Table 1.2: Discounted Cummulative Gain

i	rel_i	$log_2(i+1)$	$rel_i/log_2(i+1)$
1	3	1	3
2	1	1.585	0.63
3	1	2	0.5
4	2	2.322	0.86
5	0	2.585	0

Normalized Discounted Cummulative Gain at 5

Below is the list of documents D1 through D5 and their relevance scores. This data has been reused from table 1.1.

Documents: D1, D2, D3, D4, D5

Relevance Score: 3, 1, 1, 2, 0

$CG_5 = 3 + 1 + 1 + 2 + 0$

$CG_5 = 7$

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

$DCG = 3 + 0.63 + 0.5 + 0.86 + 0 = 4.99$

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

Documents: D1, D4, D2, D3, D5

Relevance Score: 3, 2, 1, 1, 0

$CG_5 = 3 + 2 + 1 + 1 + 0$

$CG_5 = 7$

Table 1.3: Ideal Discounted Cummulative Gain

i	rel_i	$log_2(i+1)$	$rel_i/log_2(i+1)$
1	3	1	3
2	2	1.585	1.26
3	1	2	0.5
4	1	2.322	0.43
5	0	2.585	0

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

$IDCG = 3 + 1.26 + 0.5 + 0.43 + 0 = 5.19$

Normalized Discounted Cummulative Gain (NDCG)= DCG / IDCG

$NDCG = 4.99/5.19 = 0.96$

Normalized Discounted Cummulative Gain at 10

Below is the list of documents D1 through D10 and their relevance scores. This data has been reused from table 1.1.

Documents: D1, D2, D3, D4, D5, D6, D7, D8, D9, D10

Relevance Score: 3, 1, 1, 2, 0, 0, 3, 0, 0, 2

$CG_{10} = 3 + 1 + 1 + 2 + 0 + 0 + 3 + 0 + 0 + 2$

$CG_{10} = 12$

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

$DCG = 3 + 0.63 + 0.5 + 0.86 + 0 + 0 + 1 + 0 + 0 + 0.578 = 6.568$

Table 1.4: Discounted Cummulative Gain

i	rel_i	$log_2(i+1)$	$rel_i/log_2(i+1)$
1	3	1	3
2	1	1.585	0.63
3	1	2	0.5
4	2	2.322	0.86
5	0	2.585	0
6	0	2.807	0
7	3	3	1
8	0	3.17	0
9	0	3.22	0
10	2	3.459	0.578

Table 1.5: Ideal Discounted Cummulative Gain

i	rel_i	$log_2(i+1)$	$rel_i/log_2(i+1)$
1	3	1	3
2	3	1.585	1.89
3	2	2	1
4	2	2.322	0.86
5	1	2.585	0.387
6	1	2.807	0.356
7	0	3	0
8	0	3.17	0
9	0	3.22	0
10	0	3.459	0

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

Documents: D1, D7, D4, D10, D2, D3, D5, D6, D8, D9

Relevance Score: 3, 3, 2, 2, 1, 1, 0, 0 , 0 , 0

$CG_{10} = 3 + 3 + 2 + 2 + 1 + 1 + 0 + 0 + 0 + 0$

$CG_{10} = 12$

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

$IDCG = 3 + 1.89 + 1 + 0.86 + 0.387 + 0.356 + 0 + 0 + 0 + 0 = 7.493$

Normalized Discounted Cummulative Gain (NDCG)= DCG / IDCG

$NDCG = 6.568/7.493 = 0.977$

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: Relevance table for Query: Code optimization for space efficiency

index	relevant
1	yes
2	no
3	no
4	yes
5	no
6	no
7	yes
8	no
9	no
10	yes

2.2.1 Query: Code optimization for space efficiency

2.2.2 Query: Parallel algorithms

Table 2.2: Precision and Recall Table for Query: Code optimization for space efficiency

Index	1	2	3	4	5	6	7	8	9	10
Relevant	yes	no	no	yes	no	no	yes	no	no	yes
Recall	0.25	0.25	0.25	0.5	0.5	0.5	0.75	0.75	0.75	1
Precision	1.0	0.5	0.33	0.5	0.4	0.33	0.429	0.375	0.33	0.4

Table 2.3: Relevance table for Query: Parallel algorithms

index	relevant
1	no
2	yes
3	yes
4	yes
5	yes
6	yes
7	no
8	yes
9	yes
10	no

Table 2.4: Precision and Recall Table for Query: Parallel algorithms

Index	1	2	3	4	5	6	7	8	9	10
Relevant	no	yes	yes	yes	yes	yes	no	yes	yes	no
Recall	0.0	0.143	0.286	0.429	0.571	0.714	0.714	0.857	1	1
Precision	0.0	0.5	0.67	0.75	0.8	0.833	0.714	0.75	0.78	0.7

Table 2.5: 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	1	1	1	0.5	0.5	0.5	0.429	0.429	0.4	0.4	0.4
Ranking 2	0.833	0.833	0.833	0.833	0.833	0.833	0.833	0.833	0.78	0.78	0.78
Average Ranking	0.917	0.917	0.917	0.667	0.667	0.667	0.631	0.631	0.59	0.59	0.59

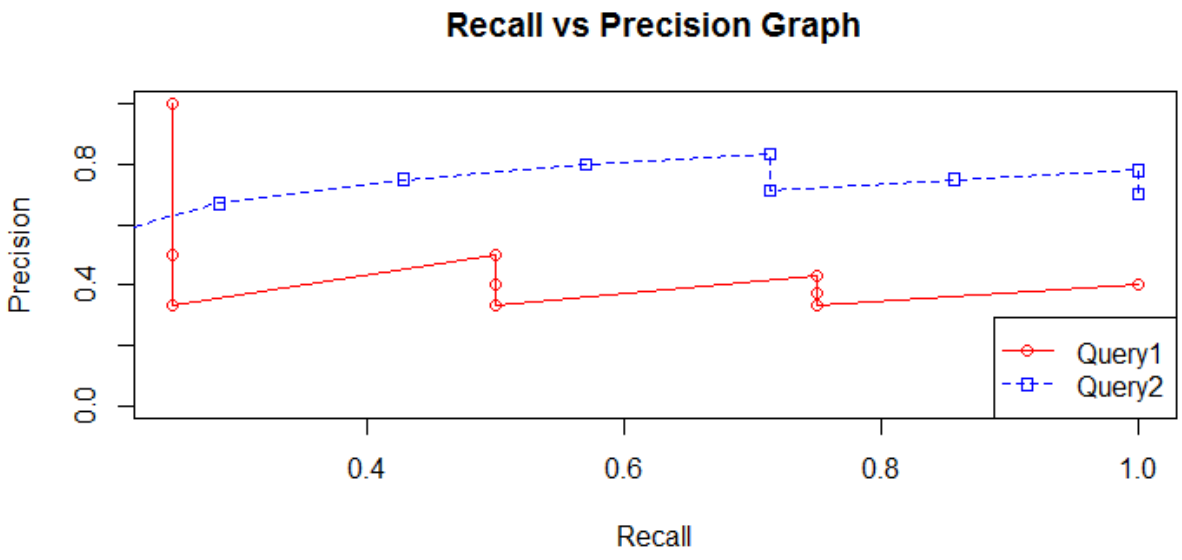

Figure 2.1: Recall vs Precision Graph

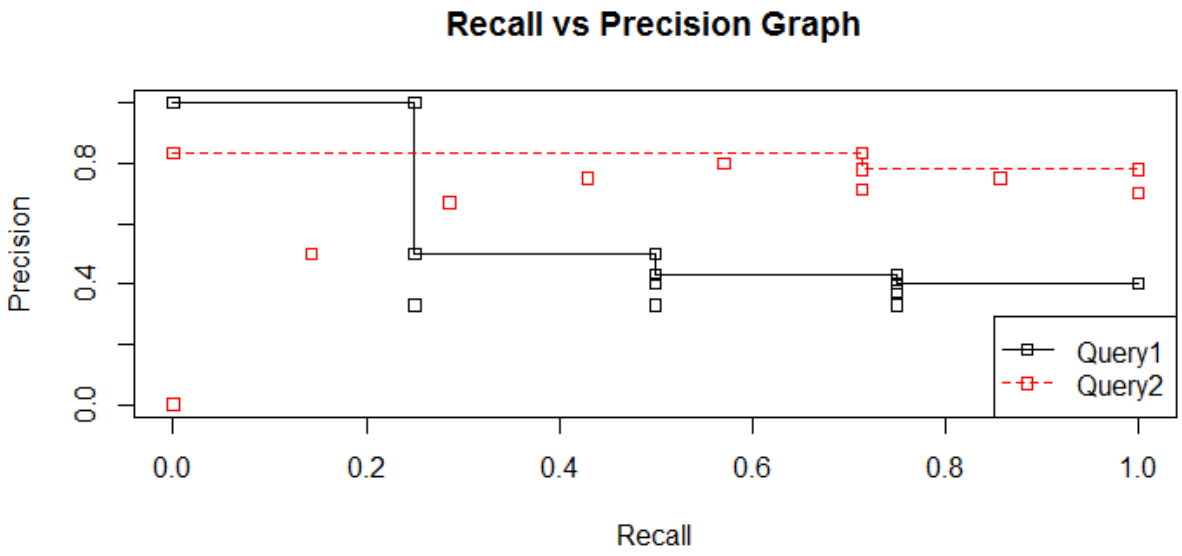

Figure 2.2: Interpolated Recall vs Precision Graph

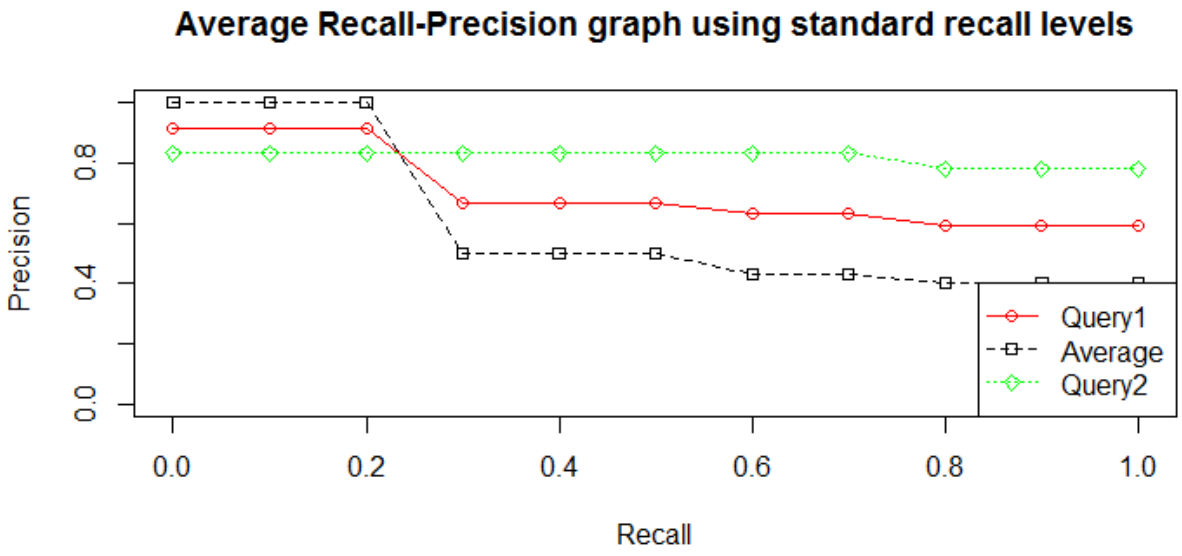

Figure 2.3: Average recall-precision graph using standard recall levels

Chapter 3

Problem 8.9

3.1 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.

3.2 Solution

The solution is based on relevance results for query, code optimization for space efficiency. The Table 4.1 shows the relevance results for the query. The relevance results show the search results had 4 relevant documents and 6 non- relevant documents.

3.2.1 Calculating BPREF

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

Table 3.1: Relevance table for Query: Code optimization for space efficiency

index	relevant
1	yes
2	no
3	no
4	yes
5	no
6	no
7	yes
8	no
9	no
10	yes

Table 3.2: Relevance table showing R relevant and non-relevant documents

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

where R is the number of non-relevant documents that are considered

d_r is the number of relevant documents

For a query result with 4 relevant documents R is 4 implying that first 4 non-relevant documents are considered. The relevance table for the query is manipulated for BPREF as,

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

$$BPREF = 1/4[(1 - 0/4) + (1 - 2/4) + (1 - 4/4) + (1 - 4/4)]$$

$$BPREF = 1/4[1 + (1/2) + 0 + 0] = 3/8 = 0.375$$

3.2.2 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 = 4

Q = 6

$BPREF = 4/(4 + 6)$

$BPREF = 0.4$

The value of BPREF is 0.375 and 0.4 by computing respectively with relevant documents formula and the clickthrough preference formula.

Chapter 4

Problem 9.8

4.1 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?

4.2 Solution

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

4.2.1 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.

```
'''
Created on Nov 22, 2017

@author: nauman
'''
import math

def singleLinkage(clusterPointA, clusterPointB):
    minDistance = math.inf
    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 minDistance > distance:
                minDistance = distance
```

```
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
```

```
41                      if bestClusterA and bestClusterB:
42                          bestClusterA.pop()
43                          bestClusterB.pop()
44                      bestClusterA.append(clusterPoints[i])
45                      bestClusterB.append(clusterPoints[j])
46          clusterPoints.remove(bestClusterB[0])
47          index = clusterPoints.index(bestClusterA[0])
48          for x in range(0,len(bestClusterB[0])):
49              clusterPoints[index].append(bestClusterB[0][x])
50      return (clusterPoints)
51
52  def completeLinkage(clusterPointA, clusterPointB):
53      maxDistance = 0
54      for i in range(0, len(clusterPointA)):
55          for j in range(0,len(clusterPointB)):
56              distance = math.sqrt(math.pow((clusterPointA[i][0]-clusterPointB[j
    ][0]),2) + math.pow((clusterPointA[i][1]-clusterPointB[j][1]),2))
57              if maxDistance < distance:
58                  maxDistance = distance
59      return maxDistance
60
61  def agglomerativeCompleteLinkageCluster(clusterValue, input):
62
63      distanceMatrix = []
64      finalClusters = []
65
```

```
    clusterPoints = input
    for i in range(0,len(clusterPoints)):
        finalClusters.append([i])

    for c in range(len(clusterPoints), clusterValue, -1):
        bestClusterA = []
        bestClusterB  = []
        bestCost = 0
        for i in range(0,len(clusterPoints)):
            temp = []
            for j in range(0,len(clusterPoints)):
                temp.append(0)
            distanceMatrix.append(temp)

        for i in range(0,len(clusterPoints)):
            for j in range((i+1),len(clusterPoints)):
                distance = completeLinkage(clusterPoints[i], clusterPoints[j])
                distanceMatrix[i][j] = (distance)
                if bestCost < distance:
                    bestCost = distance
                    if bestClusterA and bestClusterB:
                        bestClusterA.pop()
                        bestClusterB.pop()
                    bestClusterA.append(clusterPoints[i])
                    bestClusterB.append(clusterPoints[j])
        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 = []
```

```
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)
123 
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)
140 
141 def averageGroupLinkage(clusterPointA, clusterPointB):
```

```
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
```

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

    for c in range(len(clusterPoints), clusterValue, -1):
        bestClusterA = []
        bestClusterB  = []
        bestCost = math.inf
        for i in range(0,len(clusterPoints)):
            temp = []
            for j in range(0,len(clusterPoints)):
                temp.append(0)
            distanceMatrix.append(temp)

        for i in range(0,len(clusterPoints)):
            for j in range((i+1),len(clusterPoints)):
                distance = averageGroupLinkage(clusterPoints[i], clusterPoints
    [j])
                distanceMatrix[i][j] = (distance)
                if bestCost > distance:
                    bestCost = distance
                    if bestClusterA and bestClusterB:
                        bestClusterA.pop()
                        bestClusterB.pop()
                    bestClusterA.append(clusterPoints[i])
                    bestClusterB.append(clusterPoints[j])
        clusterPoints.remove(bestClusterB[0])
```

```
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)
```

```
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:
```

```
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)
```

```
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]]
```

4.2.2 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 5

9.9

5.1 Problem

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

5.2 Solution

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

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

```
from spherecluster import SphericalKMeans

def kMeans(num):

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

def sphericalKMeans(num):
    num = 4
    file = open("Kmeans.txt", "a+")
    input = np.array ([[-4, -2], [-3, -2], [-2, -2], [-1, -2], [1, -1], [1,
    1], [2, 3], [3, 2], [3, 4], [4,3]])
    kmeans = SphericalKMeans(n_clusters=num).fit(input)
    file.write("Spherical K means output for cluster size : "+ str(num) + "\n"
    )
    file.write("Clusters index of points" + "\n")
    file.write(str(kmeans.labels_) + "\n")
    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 5.1: 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 6

Problem 9.11

6.1 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.

6.2 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 application. 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 relavant 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.