Assignment 4

CS 734: Introduction to Information Retrieval

Fall 2017

Grant Atkins

Finished on December 4, 2017

1

Question

8.10. Consider a test collection that contains judgments for a large number of time-sensitive queries, such as "olympics" and "miss universe". Suppose that the judgments for these queries were made in 2002. Why is this a problem? How can online testing be used to alleviate the problem?

Answer

An obvious problem is that the judgements may be stale by the time new queries have been made at a later part of the day, or even a later date. If new stories are being created and the judgements are pointing to old documents that start to become irrelevant its apparent that the judgements have to be reevaluated. There could also be issues with bias based on the past queries. Queries based on "olympics" and "miss universe" will have a higher relevance factor introducing bias into the results retrieved in the current state.

This is where online testing comes in. Online testing is the idea that it is possible to test, or even train, data using live traffic. In this problem, online testing can be used in a variety of ways based on time-sensitive queries. Assuming there is live tracking of what queries are going out, we can track user: queries, click data, and timestamps.

There are a few ways the current system could be improved. We can use user query timestamps to provide live updates to the relevance models that at X time this seems like a likely query a user would ask. The most beneficial item we can use in online testing is of course the clickthrough data based on users. The clickthrough data will have the documents for which the user was looking for. Although this can be difficult to integrate in live tracking, the clickthrough rate for ads or URIs provided from SERPs can help live update the documents relevance for all users. Of course there could also be noise in the live data if everyone's click data is considered, which is unreasonable so it would probably be efficient to limit it to 10% of the users.

2

Question

Answer

3

Question

```
9.8. 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?
```

Answer

To solve this problem I took I wrote a program in R named **agg_clusters.R**, as shown in Listing 1 to calculate the clusters and use the five agglomerative methods: single-linkage, average-linkage, complete-linkage, centroid-linkages, and Ward's minimum variance method. I first created a csv file that had all the values of the 2D instances named **cluster-data.csv** located in my repository [1]. A hard lesson learned when doing this problem was that when copying and pasting from a textbook, make sure the encodings on the negative symbol are ASCII compliant or you will be left clueless. I then read in this csv file and used the dist() method to create a matrix with the euclidean distances between each point. From there I used the hclust() method to cluster the distance matrix for each of the agglomerative methods and created plots for each method as shown in Figures 1, 1, 3, 4 and 5.

To my surprise, each of the clusters/plots created were exactly the same. I again tried using the same code, this time creating dendrograms and I was in fact getting the same clusters as shown in Figures 6 and 7, showing dendrograms of single-linkage and complete-linkage respectively. Compared to how I would manually cluster the points not much would change. I would still have to find a distance metric between all of them, using euclidean, and then also I would have to compute a cost to cluster each points by.

```
1 library(cluster)
2 library(ggplot2)
3 library(factoextra)
4
5 # data
6 df <- read.csv("./data/cluster-data.csv", header = TRUE)
7 # Make NxN dimensions with euclidean distance
8 dist <- dist(df, method = "euclidean")
```

```
# select methods to run against
agg_methods <- c("single", "average", "complete", "centroid", "
    ward.D2")

# iterate through each method and make a plot
for(method in agg_methods){
  p_title <- paste("Using '", method,"' Cluster Method", sep =
      "")
  print(p_title)
  filename <- paste("../docs/", method, ".pdf", sep = "")
  pdf(filename)

  avg <- hclust(dist, method = method)
  sub_grp <- cutree(avg, k = 3)
  cluster <- fviz_cluster(list(data = df, cluster = sub_grp),
                          main = p_title, xlab = "2D Instance",
                              ylab = "Distance Value")
  print(cluster)
  # dendrogram vis
  # filename <- paste("../docs/", method, "-dendro.pdf", sep =
      "")
  # pdf(filename)
  # plot(avg, cex = 0.6)
  # rect.hclust(avg, k = 4, border = 2:5)
  dev.off()
}
```

Listing 1: R script to cluster data and create cluster plots

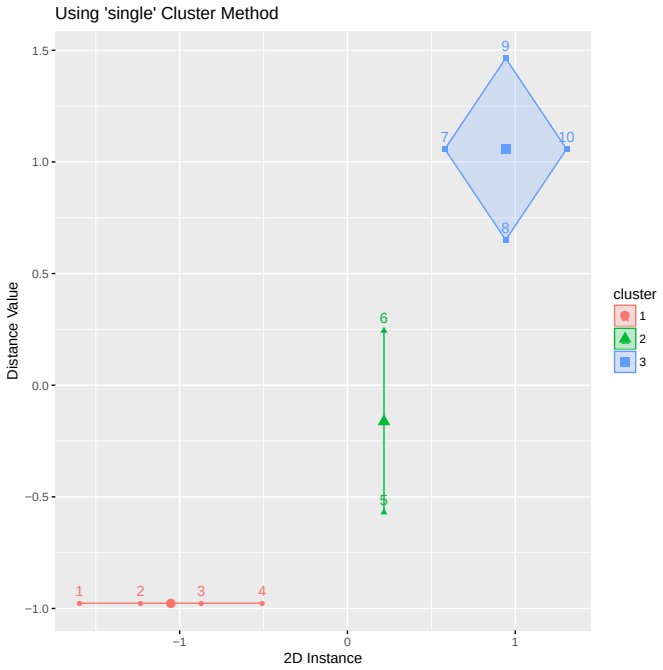

Figure 1: Single-linkage

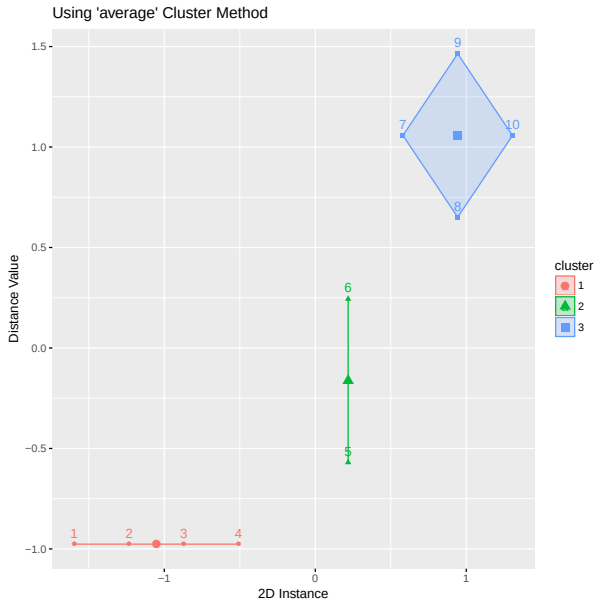

Figure 2: Average-linkage

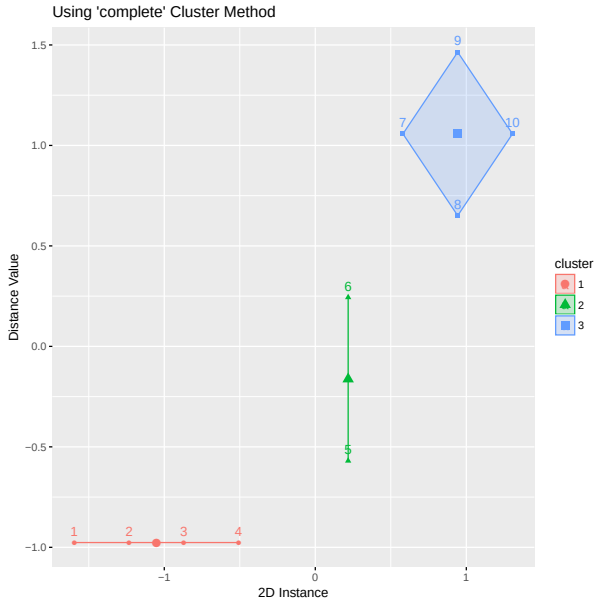

Figure 3: Complete-linkage

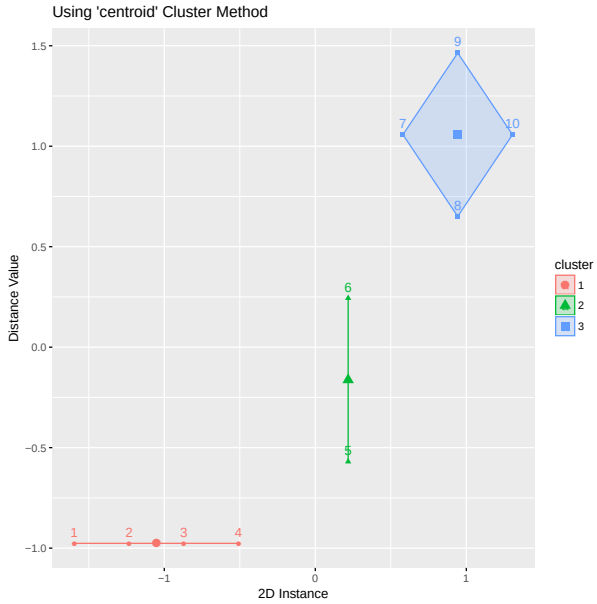

Figure 4: Centroid-linkage

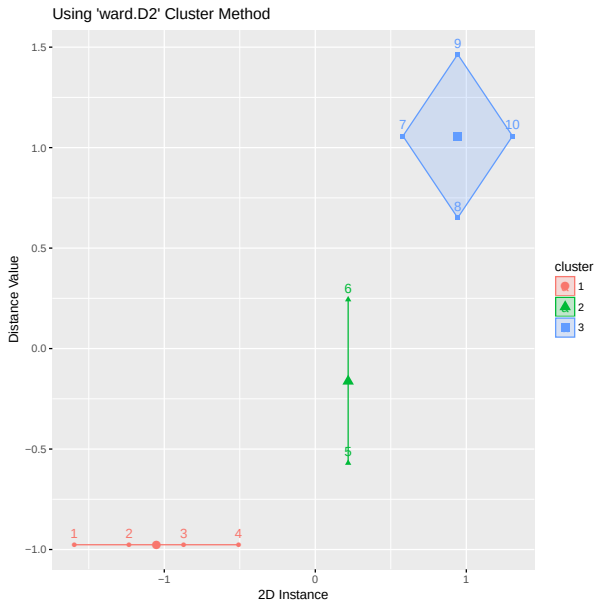

Figure 5: Ward's minimum variance method

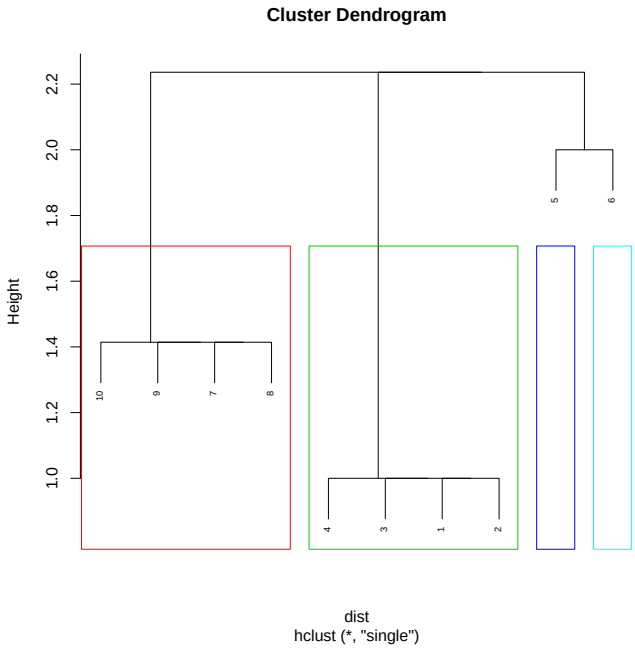

Figure 6: Single-linkage dendrogram

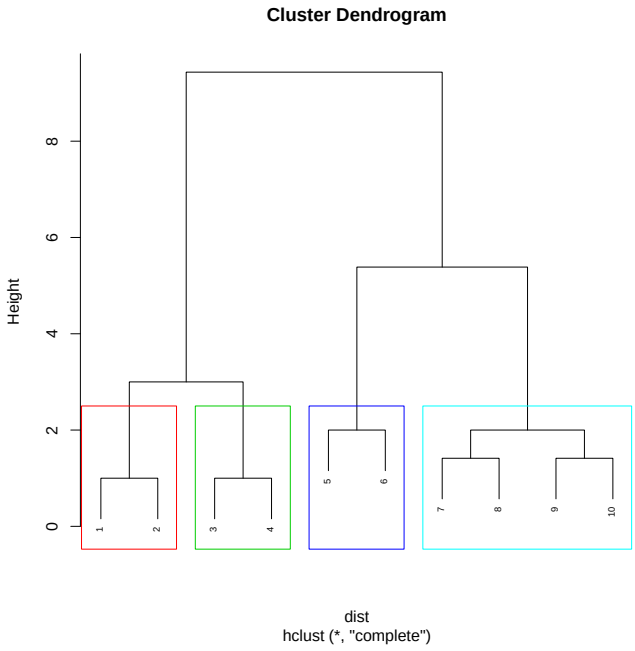

Figure 7: Complete-linkage dendrogram

4

Question

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

Answer

As described in the book [2], spherical K-means is the same as K-means except that for the distance measurement between nodes is based on cosine similarity, not euclidean. Spherical K-means seeks to get somewhat of an equal space distance between nodes.

To solve this problem I wrote an R program, as shown in Listing 2. This script utilizes the k-means function provided by the cluster library. I first took the euclidean distance of the 2D points from 9.8, and then I also wrote a function to compute the cosine similarity in a method named *cosineDist*. After getting these two distances matrices I applied k-means to them. The k-means clusters based on euclidean distance is shown in Figure 8. The k-means clusters based on cosine distance is shown in Figure 9.

For both regular k-means and spherical k-means I specified two centers because this seemed to work out well. Regular k-means showed to have 5 nodes in each cluster while spherical k-means showed to have a 4 to 6 split.

```
1  library(cluster)
2  library(ggplot2)
3  library(factoextra)
4
5  # data
6  df <- read.csv("./data/cluster-data.csv", header = TRUE)
7
8  # Make NxN dimensions with euclidean distance
9  euc_dis <- dist(df, method = "euclidean")
10
11 cosineDist <- function(x){
12   as.dist(1 - x%*%t(x)/(sqrt(rowSums(x^2) %*% t(rowSums(x^2)))))
13 }
14 cos_sim <- cosineDist(as.matrix(df))
15
16 pdf("../docs/kmeans-euc.pdf")
17 k2 <- kmeans(euc_dis, centers = 2, nstart = 25)
18 clust <- fviz_cluster(k2, data = df,
19                       main = "Euclidean Distance K-Means",
```

```
                                xlab = "2D Instance", ylab = "Cluster
                                    Height")
print(clust)
dev.off()

pdf("../docs/kmeans-cos.pdf")
cos_k2 <- kmeans(cos_sim, centers = 2, nstart = 25)
clust <- fviz_cluster(cos_k2, data = df,
                                main = "Cosine Similarity K-Means",
                                xlab = "2D Instance", ylab = "Cluster
                                    Height")
print(clust)
dev.off()
```

Listing 2: R script that computes k-means for euclidean distance and cosine similarity

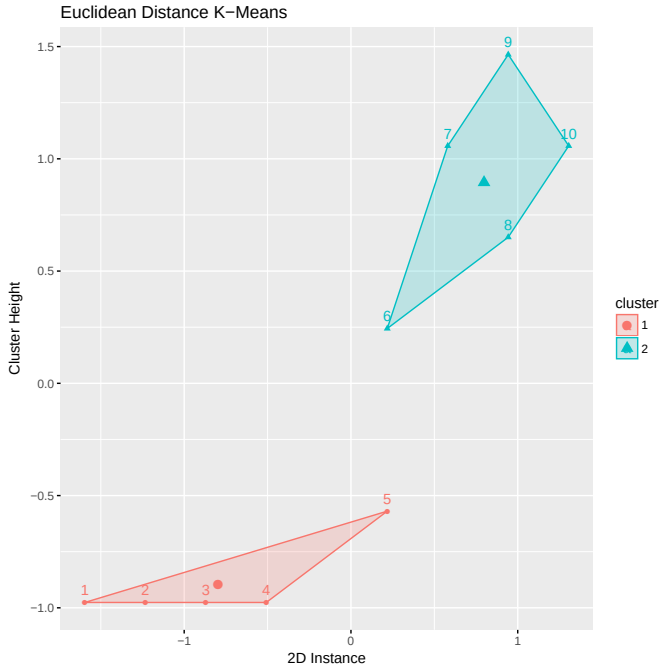

Figure 8: K-means with euclidean distance

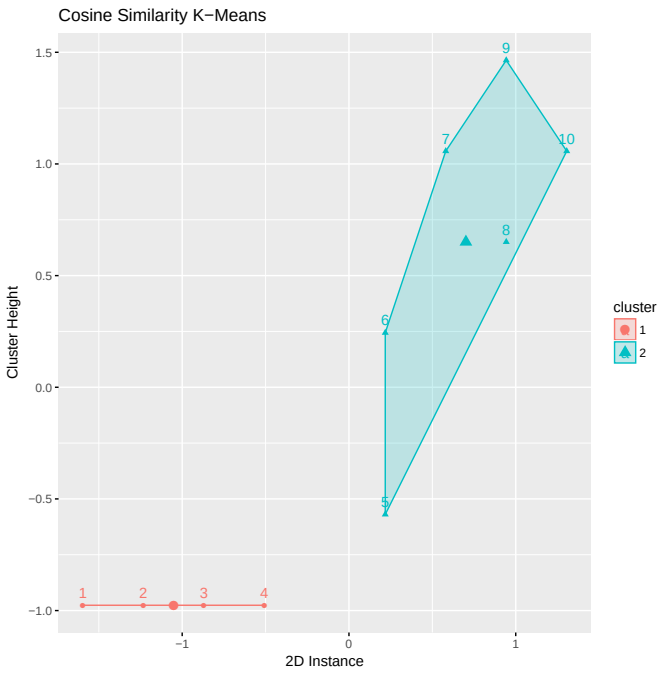

Figure 9: K-means with cosine distance

5

Question

9.11. 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.

Answer

I think using this approach in a search engine application is an interesting approach. Often search engines are known to have a cache to quick lookup results. Documents in these search are also often stored in inverted indexes with popularity weight.

I believe one way K nearest neighbors can be used is to represent a document based on the links in the document itself. If a link points to an outside document that shows a relationship. I think if we grouped top-K links found in the document to the document itself, this way could be used to augment SERP results. What I mean is that if a document had 5 links as nearest neighbors it would be represented as follows:

```
{
d_1: [link_1,link_2, ..., link_k],
d_2: [link_1,link_2, ..., link_k],
.
.
.
d_k: [link_1,link_2, ..., link_k]
}
```

When a search engine retrieves results whichever N number of links have a high weight association they can be shown on the SERP results as a successive answer. This is assuming there is a threshold specified somewhere for the weights.

Another Idea is that the overlap determined by k nearest neighbors could also be used in a pseudo relevance feedback way. Pseudo relevance feedback is an automated analysis provided locally to users in a way the user doesn't have to interact with the system. I think a good example a way this could work would be based on clickthrough data provided by a user. One example is a user goes to a website like Pinterest where users are shown a collection

of images, and for the sake convenience lets say the images are grouped in categories. If a user clicks a link in a section, we can take the K nearest neighbor links and say they are closely associated. A pseudo relevance feedback model can pick do some sort of pre-fetching locally and then get the associated terms of the documents. The next time the user visits the Pinterest website a user should have some sort of model that says they like the documents closely associated with their clickthrough data.

References

[1] Atkins, Grant. "CS734 Assignment 4 Repository" Github. N.p., 4 December 2017. Web. 4 December 2017.`https://github.com/grantat/cs834-f17/tree/master/assignments/A4`.

[2] B. Croft, D. Metzler, and T. Strohman. "Search Engines: Information Retrieval in Practice." Pearson, 2009. Web. 14 October 2017. ISBN 9780136072249.