Homework 4

Megan Robertson

September 26, 2015

Problem 1

```
load("hw1prob1.Rdata") #loading the data for problem 1
```

a. First normalize the documents by their total word count, and then apply IDF weighting to the words. Save this matrix as dtm1. Now reverse the order; first apply IDF weighting to the words, then normalize the documents by their total word count, and save this matrix as dtm2. Are dtm1 and dtm2 different? Explain why you think they should or shouldn't be different.

```
IDFweight = function(dtm){
  D=nrow(dtm)
  dtm = scale(dtm, center = FALSE, scale = 1/log(D/colSums(dtm != 0)))
}
```

```
#normalizing by word count and then IDF weighting
dtm1 = dtm/rowSums(dtm) #normalizing by word count
dtm1 = IDFweight(dtm1) #IDF weighting

#IDF weighting than normalizing the word count
dtm2 = IDFweight(dtm) #IDF weighiting
dtm2 = dtm2/rowSums(dtm2) #normalizing by word count
```

dtm1 and dtm2 should be different. IDF weighting is a strategy used to deal with the issue of common words. IDF weighting essentially gives common words a weight of zero and thus these words are basically removed from the representation of words in the document. Thus, if IDF weighting is carried out first, the documents will then have a smaller number of words and each cell of the dtm will be normalized by a smaller number. On the other hand, if IDF weighting is carried out second, each cell in the dtm will be normalized by a larger number before IDF weighting. Therefore, dtm1 and dtm2 should be different.

b. We're going to forgo IDF weighting with such a small collection. Normalize the documents by their total word count, and call this matrix dtm3. According to the document-term matrix, which document is closest (measured in Euclidean distance) to the document named "tmnt mike"?

```
#this code is from class sources, which I was directed to by my peers
dtm3 = dtm/rowSums(dtm) #normalizing by the word count
Euc.dist = sqrt(rowSums((scale(dtm3, center=dtm3[3,], scale=F)^2))) #finding the Euclidean
  #distances between documents
Euc.dist = sort(Euc.dist); smallest = (Euc.dist[2]) #sorting the distances and finding the
  #second smallest (smalles is 0 because that is between document and itself)
print(smallest)
```

```
##  tmnt raph
## 0.03782949
```

The document that is closest is tmnt raph.

c. Sticking with the normalized matrix dtm3, compute the distance between each pair of documents using the function dist. Now run hierarchical agglomerative clustering, both with single linkage and complete linkage, using the function hclust. Plot the resulting dendograms

for both linkages.If you had to split the documents into 2 groups, which linkage do you think gives a more reasonable clustering?

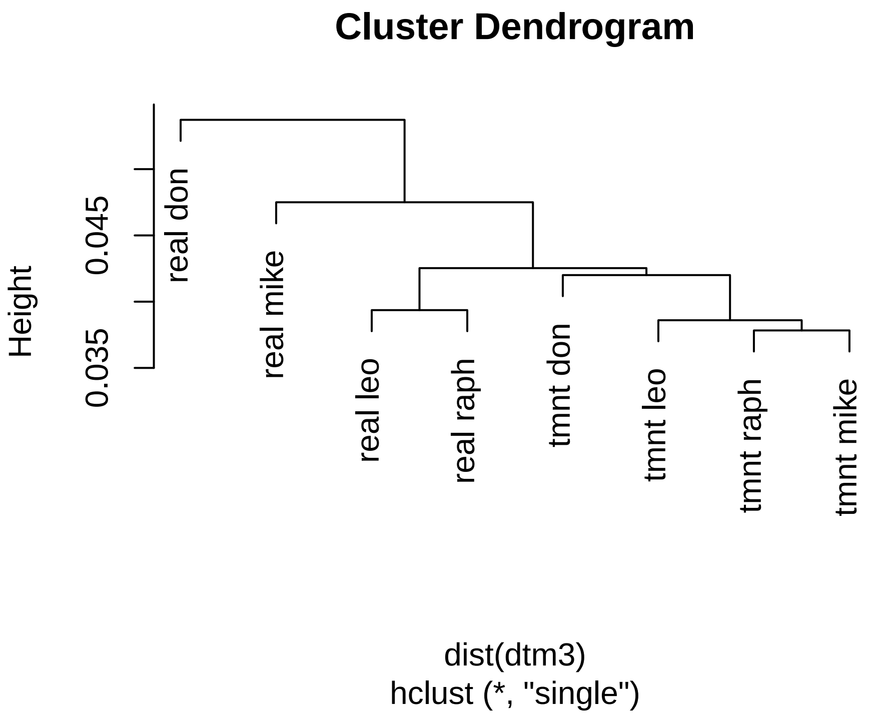
```
hclust(dist(dtm3), method="single") #finding clusters with single linkage
```

```
##
## Call:
## hclust(d = dist(dtm3), method = "single")
##
## Cluster method   : single
## Distance         : euclidean
## Number of objects: 8
```

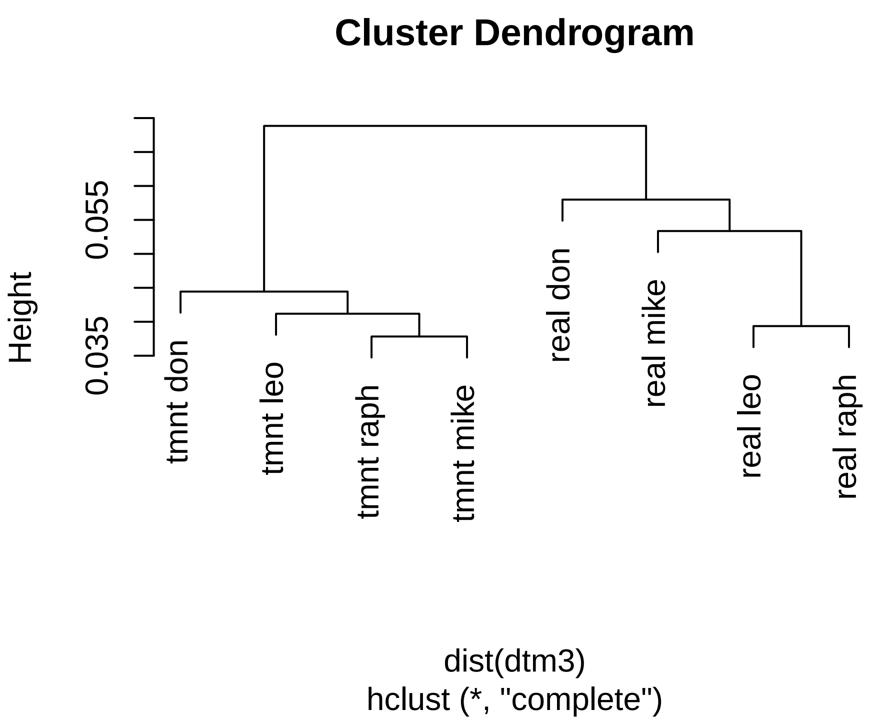
```
hclust(dist(dtm3), method="complete") #finding clusters with single linkage
```

```
##
## Call:
## hclust(d = dist(dtm3), method = "complete")
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 8
```

```
plot(hclust(dist(dtm3), method="single")) #making single linkage dendrogram
```



```
plot(hclust(dist(dtm3), method="complete")) #plotting complete linkage dendrogram
```



The complete linkage is more reasonable since the dendrogram demonstrates that the tmnt documents and the real documents can be separated into two distinct clusters.

d. Combine the word counts from all of the documents into one cumulative word count vector. I.e., for each word, you should now have a count for the number of times it appears across all 8 documents. List the top 20 most common words, and how many times they appear. What percentage of total occurrences do these top 20 words represent? How many of the top words account for 50% of total occurrences?

```
#top twenty
word.counts = colSums(dtm) #finding the word counts for each word
top.twenty = sort(word.counts)[6801:6820] #making a vector of the top twenty words
(sum(top.twenty)/sum(word.counts)) #dividing occurences of to twenty by total occurences
```

```
## [1] 0.2434249
```

```
#how many words to get 50% of occurrences
sorted = sort(word.counts) #sorting the vector of word counts

empty.column = rep(0, times=6820) #making columns of 0 to make a data frame
percentage.df = as.data.frame(cbind(empty.column, empty.column, empty.column))
colnames(percentage.df) = c("num.words", "total.count", "percentages")
total.count=0
for (i in 1:nrow(percentage.df)){ #creating a data frame
  percentage.df$num.words[i] = i #filling in the number of words being considered
  total.count = total.count + sorted[i] #finding the total word counts
  #for number of words being considered
  percentage.df$total.count[i] = total.count #storing this total in the data frame
  percentage.df$percentages[i] = total.count/sum(sorted) #finding percentage by
  #dividing by the total number of words
}
greater.than.fifty = c(which(percentage.df$percentage>=0.5)) #making a vector of the number of
#words that account for 50% or more of the total words
```

```
greater.than.fifty[1] #finding the index of first word that accounts for 50% or more
```

```
## [1] 6567
```

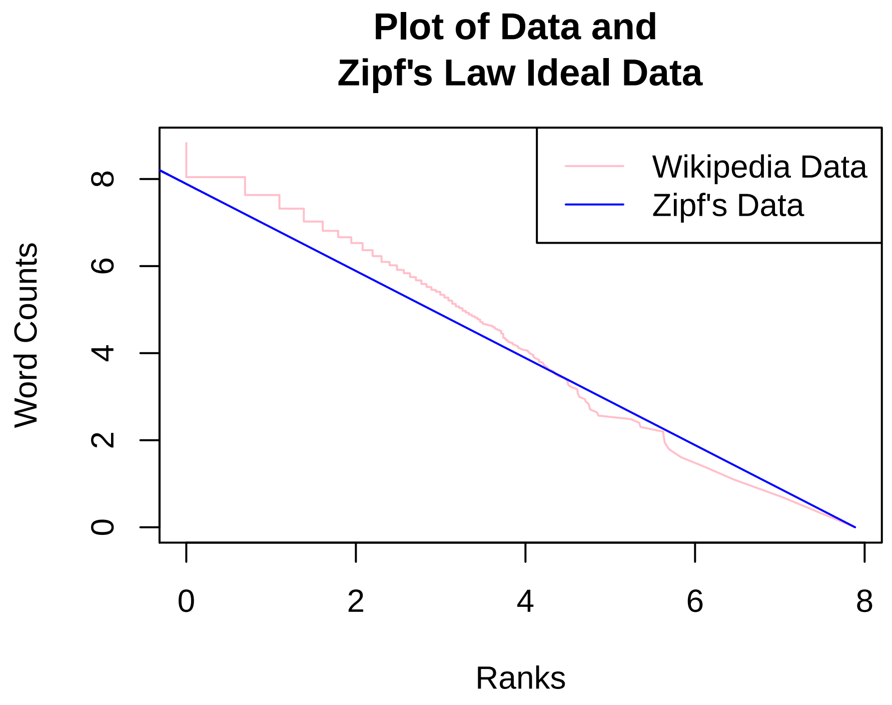
```
6820-greater.than.fifty[1] + 1 #find the number of words that accounts for 50%+
```

```
## [1] 254
```

The top 20 words represents 24.3% of total occurrences. The number of words that account for 50% of occurrences is 254.

e. Zipf's law states that, given a collection of documents like the one we have, the number times a word appears is inversely proportional to its rank (the words being ranked by how common they are). In other words, the second most common word appears half as often as the most commonword, the third most common word appears a third as often as the most common word, etc. This "law" is one that has been empirically confirmed, and aside from word counts, these kind of "power" laws have also been observed in a wide variety of different problems.1 Does our collection of 8 wikipedia articles appear to follow Zipf's law? Can you give a plot to provide visual evidence for or against this claim?

```
ranks=c(1:6820) #making a vector for the ranks of counts of words
plot(log(sort(word.counts, decreasing=T)), log((ranks)), type="l", ylab="Word Counts",
     xlab="Ranks", col="pink", main="Plot of Data and \nZipf's Law Ideal Data") #plotting the actual
  #data with a log transformation
lines(log(2664/ranks), log(ranks), type="l", col="blue") #plotting what the
#data would look like if Zipf's law held
legend("topright", c("Wikipedia Data", "Zipf's Data"), lty=c(1,1), col=c("pink", "blue"))
```



Our collection of 8 Wikipedia articles appears to follow Zipf's law. The blue line in the plot above represents what the word count would look like for data that exactly followed Zipf's law, and the pink line represents the actual word count data from our Wikipedia articles (both have been log transformed in order to visualize a linear relationship). Thus, the plot demonstrates that our data closely resembles the data that was modeled based on Zipf's law.

2. Compute the PageRank vector for the following graph, with d = 0.85. Repeat the calcu-

lation for d = 1(BrokenRank). What's the difference? Explain. You should be computing PageRank by explicitly constructing A and finding its leading eigenvector (i.e., you can't use the page.rank function in the igraph package).

```
L = matrix(0, nrow=10, ncol=10, byrow=TRUE) #making a 10x10 matrix of zeros
#next four lines put a 1 where it should be L based on the graph
L[2,3] <- 1; L[3,4] <- 1; L[1,10] <- 1
L[5,4] <-1; L[6,4] <- 1; L[10,4] <- 1; L[7,5] <- 1; L[9,8] <- 1
L[8,4] <- 1; L[10, 4] <- 1; L[7,5] <- 1; L[2,6] <- 1; L[7,8] <- 1
L[4,9] <- 1; L[3,10] <- 1; L[5,10] <- 1; L[7,10] <- 1
```

```
#function for PageRank from Lab 4
myPageRank = function(data, damp.par){ #function inputs are data and a dampening parameter
  num.links = rowSums(data) #finding the number of links pointing toward each page
  M = diag(num.links) #making a diagonal matrix
  M.inverse = solve(M) #finding the inverse of this diagonal matrix
  n = nrow(data) #storing the number of rows in the data
  onesVector = c(rep(1, times=nrow(data)*ncol(data))) #making a vector with 1s
    #to use to make the matrix
  E = matrix(onesVector, nrow=nrow(data), ncol=ncol(data)) #making matrix of 1s
  A = ((1-damp.par)/n)*E + damp.par*t(data)%*%M.inverse #defining A for PageRank
  eigenvectors = eigen(A) #finding the eigenvectors of A
  eigen.df = data.frame(eigenvectors$vectors) #making the eigenvectors into a data frame
  indices = which(signif(eigenvectors$values) == 1) #storing index of
  #eigenvectors with eigenvalue 1
  vec = matrix(eigen.df[,indices[1]]) #extracting eigenvector
  #with eigenvalue 1 from the data frame
  rank = scale(vec, center = FALSE, scale=as.numeric(sum(vec))) #scaling the eigenvector
  #to get the vector of page ranks
  return(rank) #returning the vector of PageRanks, this is the output of the function
}

PageRank = myPageRank(L, 0.85) #calling myPageRank on link matrix and d=0.85

#broken PageRank (code taken from Lab 4 with minor adjustments)
num.links = rowSums(L) #finding the number of links for each page
M = diag(num.links) #making a diagonal matrix with the number of the links
Minv = solve(M) #finding the inverse of the matrix M
L.t = t(L) #getting the transpose of the link matrix
A = (L.t%*%Minv) #defining A
eigenvectors = eigen(A) #calculating the eigenvectors
eigen.df = data.frame(eigenvectors$vectors) #makes eigenvectors into a data frame
indices = which(signif(eigenvectors$values) == 1) #getting indices of eigenvalues 1
vec.1 = matrix(eigen.df[,indices[1]]) #one eigenvector with eigenvalues of 1
brokenPageRank = scale(vec.1, center = FALSE, scale=as.numeric(sum(vec.1))) #normalizing
#to get page ranks
```

The PageRank vector is (0.015+0i, 0.015+0i, 0.021375+0i, 0.3097121+0i, 0.01925+0i, 0.021375+0i, 0.015+0i, 0.255767+0i, 0.2782553+0i, 0.04926563+0i) and the BrokenRank vector is (0+0i, 0+0i, 0+0i, 0.3333333+0i, 0+0i, 0+0i, 0+0i, 0.3333333+0i, 0.3333333+0i, 0+0i). The difference in the ranks results because of the web graph. The Markov chain is not strongly connected because the graph has a dangling link (1) as well as at least one loop (8-4-9). Thus, the stationary distribution for the Markov chain is non-unique, and the BrokenRank vector is ambiguously defined. The BrokenRank vector contains ranks of zeros for some pages as a result of this ambiguous definition and the issues with the graph mentioned above.

Problem 3

a. Let X_1, . . . $X_p \in R^p$, and C be a function assigning points to clusters 1, . . . , K. Let n_k be the number of points assigned to the kth cluster. Prove that the within-cluster scatter here is exactly the within-cluster variation: $\frac{1}{2}\sum_{k=1}^{K}\frac{1}{n_k}\sum_{c(i)=k}\sum_{c(j)=k}||x_i - x_j||_2^2$ where $\bar{X}_k$ is the average of points in the group k. That is $\bar{X}_k = \frac{1}{n_k} = \sum_{c(i)=k} X_i$.

$\frac{1}{2}\sum_{k=1}^{K}\frac{1}{n_k}\sum_{c(i)=k}\sum_{c(j)=k}||x_i - x_j||_2^2$

$= \frac{1}{2}\sum_{k=1}^{K}\frac{1}{n_k}\sum_{c(i)=k}\sum_{c(j)=k} x_i^2 - 2x_i\ x_j + x_j^2$

$= \frac{1}{2}\sum_{k=1}^{K}\frac{1}{n_k}\sum_{c(i)=k}\sum_{c(j)=k} x_i^2 - 2\sum_{c(i)=k}\sum_{c(j)=k} x_i\ x_j + \sum_{c(i)=k}\sum_{c(j)=k} x_j^2$

$= \frac{1}{2}\sum_{k=1}^{K}\sum_{c(i)=k} x_i^2 - \frac{2}{n_k}\sum_{c(i)=k}\sum_{c(j)=k} x_i\ x_j + \sum_{c(i)=k} x_j^2$

$= \frac{1}{2}\sum_{k=1}^{K} 2\sum_{c(i)=k} x_i^2 - \frac{2}{n_k}\sum_{c(i)=k}\sum_{c(j)=k} x_i\ x_j$

$= \sum_{k=1}^{K}\sum_{c(i)=k} x_i^2 - \frac{1}{n_k}\sum_{c(i)=k}\sum_{c(j)=k} x_i\ x_j$

$= \sum_{k=1}^{K}\sum_{c(i)=k} x_i^2 - \frac{1}{n_k}\sum_{c(i)=k}\sum_{c(j)=k} x_i\ x_j + \frac{1}{n_k}\sum_{c(i)=k}\sum_{c(j)=k} x_i\ x_j - \frac{1}{n_k}\sum_{c(i)=k}\sum_{c(j)=k} x_i\ x_j$

$= \sum_{k=1}^{K}\sum_{c(i)=k} x_i^2 - \frac{2}{n_k}\sum_{c(i)=k}\sum_{c(j)=k} x_i\ x_j + \frac{1}{n_k}\sum_{c(i)=k}\sum_{c(j)=k} x_i\ x_j$

$= \sum_{k=1}^{K}\sum_{c(i)=k} x_i^2 - \frac{2}{n_k}\sum_{c(i)=k}\sum_{c(j)=k} x_i\ x_j + \frac{1}{n_k^2}\sum_{c(i)=k}\sum_{c(i)=k}\sum_{c(j)=k} x_i\ x_j$

The previous substitution is possible since $\frac{1}{n_k}\sum_{c(i)=k} 1 = 1$. Thus, this is the same as multiplying by one.

$= \sum_{k=1}^{K}\sum_{c(i)=k}[x_i^2 - 2\sum_{c(j)=k}\frac{x_i}{n_k}\sum_{c(j)=k} x_j + (\frac{1}{n_k}\sum_{c(j)=k} x_i)^2]$

b. Let Z_i, . . . , $Z_m \in R^p$. Prove $\sum_{i}^{m}||Z_i$ - c$||\ _2^2$ is minimized by taking the average of the points.

$\sum_{i}^{m}||Z_i$ - c$||\ _2^2$

$= \sum_{i}^{m}(\sqrt{(z_1-c)^2+...+(z_i-c)^2})^2$

$= \sum_{i}^{m}(z_1-c)^2 + \ldots + z_i - c)^2$

$= \sum_{i}^{m}\frac{d}{dc}(z_1^2 - 2z_i\ \mathrm{c} + c^2) + \ldots + (z_m^2\ -2z_m\ \mathrm{c} + c^2) = \sum_{i}^{m}(-2z_i + 2\mathrm{c}) + \ldots + (-2z_m + 2\mathrm{c})$

$= \sum_{i}^{m}(-2z_i + 2\mathrm{c})$

$\sum_{i}^{m}$ (-2z_i + 2c) = 0
(-2z_1 + 2c) + . . . + (-2z_m + 2c) = 0
-2(z_1 + . . . + z_m) = -2mc
(z_1 + . . . + z_m) = mc
c = $\frac{(z_1+...+z_m)}{m}$

$\frac{d}{dc} \sum_{i}^{m}$ (-2z_i + c) = -2c < 0. Thus c is a minimum.

Therefore, $\sum_{i}^{m} ||Z_i$ - c$|| \frac{2}{2}$ is minimized by c = $\bar{Z}$.

c. Let W_t denote the within-cluster variation at the start of iteration t of K-mean clustering. Prove that for any t, $W_{t+1} < W_t$.

There are two steps to the K-means algorithm. These two steps are to assign a point to the closest center and then recalculate the center of the cluster.

When assigning a data point to a cluster, the algorithm chooses the assignment that minimizes the distance. As show in part a, this reduces the within-cluster variation since the within-cluster variation is the same as the within-cluster scatter. The second step also reduces the variation since the center is chosen to be the new mean. As shown in part b, $\sum_{1^m}$ = || Z_i - c || $\frac{2}{2}$ is minimized by taking the mean, so the second step also reduces the variation.

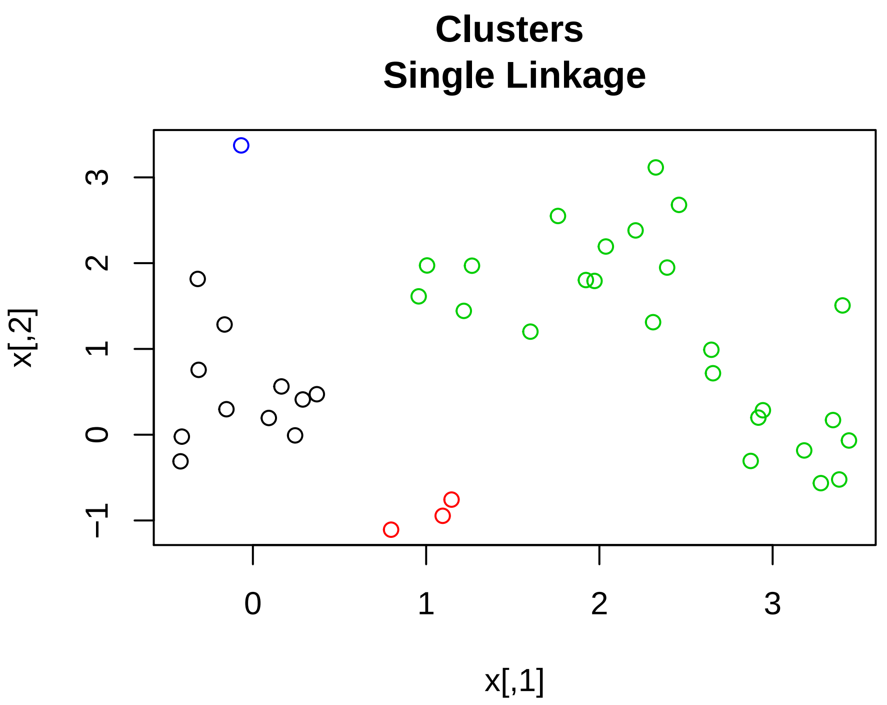
Therefore, $W_{t+1} < W_t$.

Problem 4

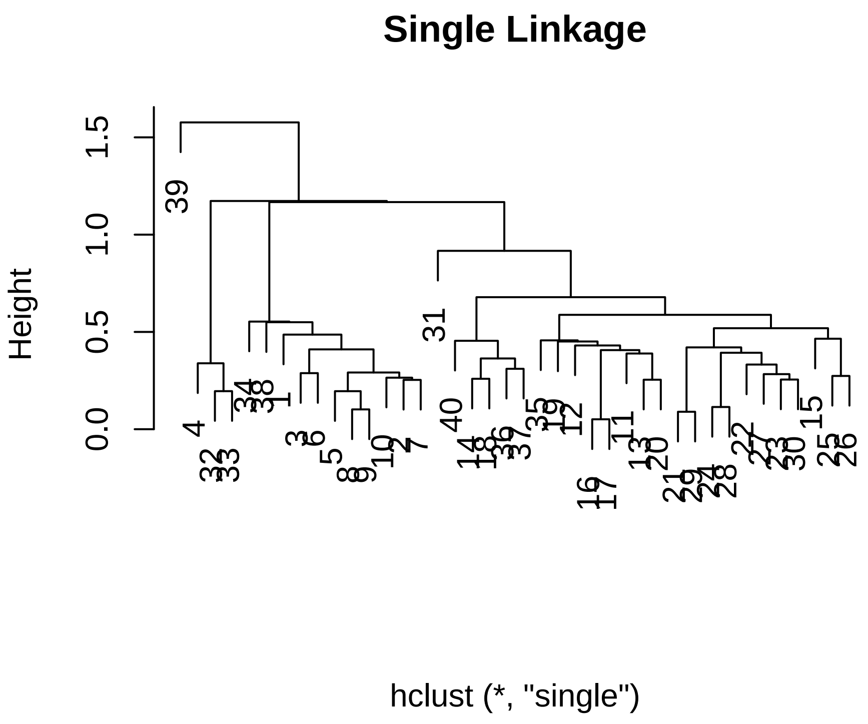
```r
load("hw1prob3.Rdata") #loading the data
```

a. Run hierarchical agglomerative clustering with single linkage, using the function hclust. Cut the tree at K = 4 clusters using the function cutree, which returns a vector of cluster assignments. Plot the points in x with different colors (or different pch values) indicating the cluster assignments. Also plot the dendogram. (Note: if you want your dendograms to look like the ones in lecture, choose a really small negative value for the hang parameter, e.g.,hang=-1e-10).

```r
hc.single = hclust(d, method="single") #hierarchical clustering for d
  #with single linkage
plot(x, col = cutree(hc.single, k = 4), main="Clusters \nSingle Linkage") #plotting
```

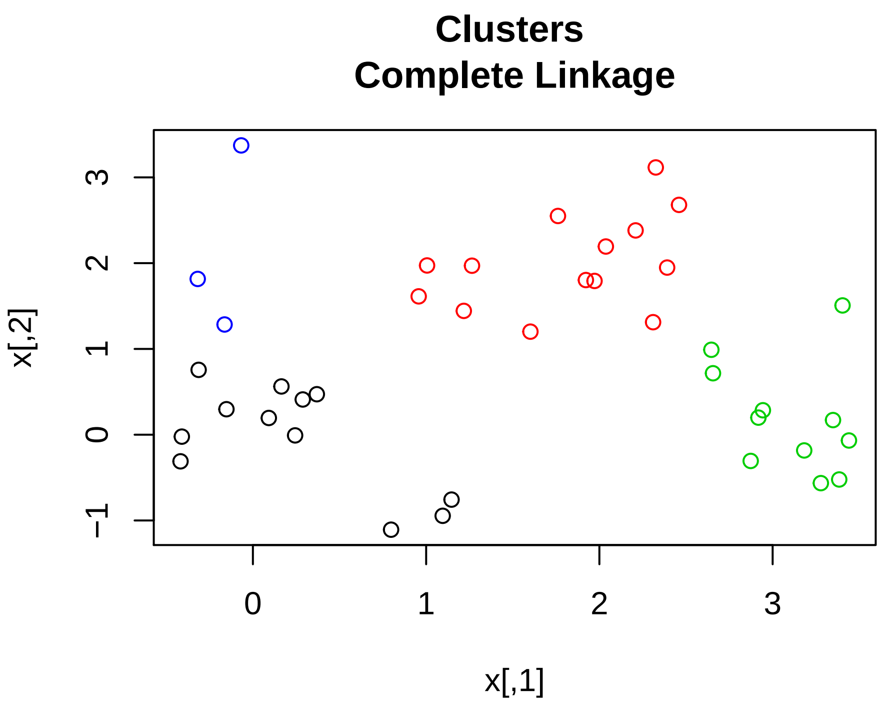


```
  #the data color coded by cluster
plot(hc.single, main="Single Linkage", xlab="") #plotting the dendogram
```

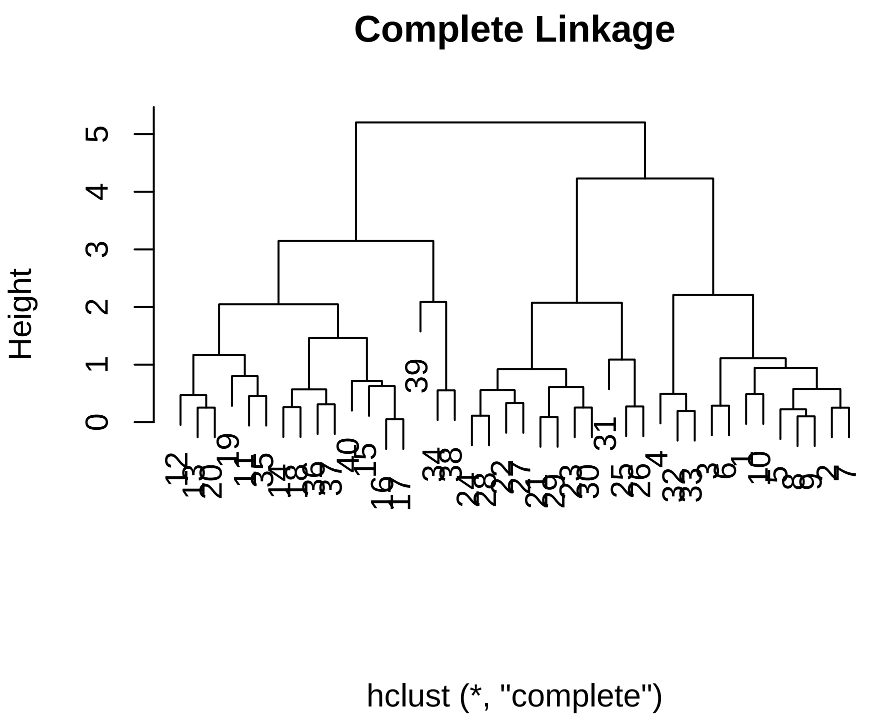


b. Repeat part (a) for complete linkage.

```
hc.complete = hclust(d, method="complete") #hierarchical clustering for d
  #with complete linkage
plot(x, col = cutree(hc.complete, k = 4), main="Clusters \nComplete Linkage") #plotting the
```

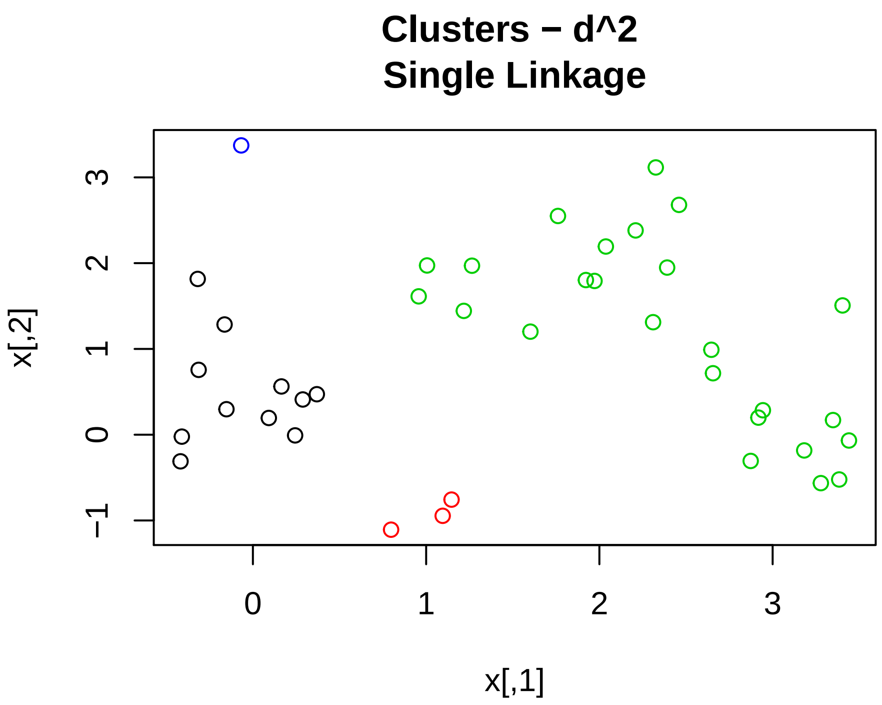


```
  #data color coded by cluster
plot(hc.complete, main="Complete Linkage", xlab="") #plotting the dendogram
```

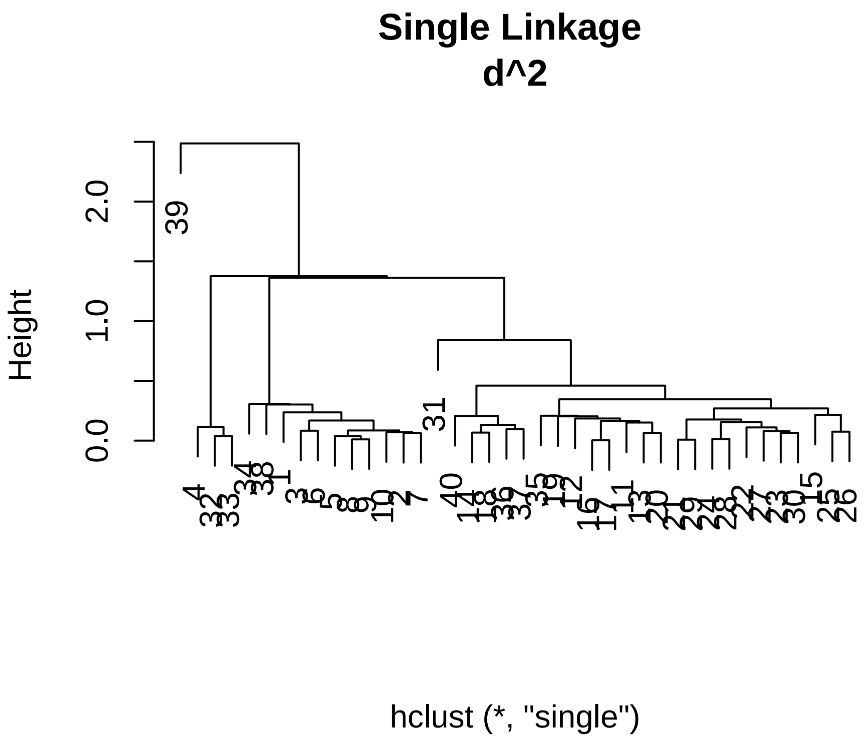


c. Repeat parts (a) and (b), but passing d^2 to the function hclust instead of d: Did the clustering assignments change? Did the dendograms change?

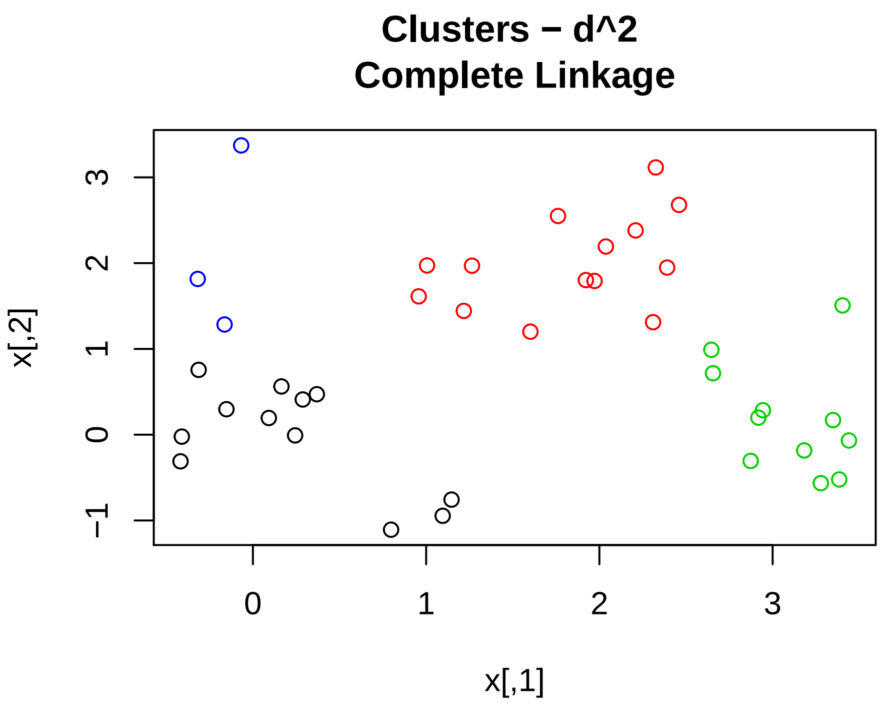
```
#single linkage
hc.single.c = hclust(d^2, method="single") #hierarchical clustering for d
  #squared with single linkage
plot(x, col = cutree(hc.single.c, k = 4), main="Clusters - d^2 \nSingle Linkage") #plotting the data
```



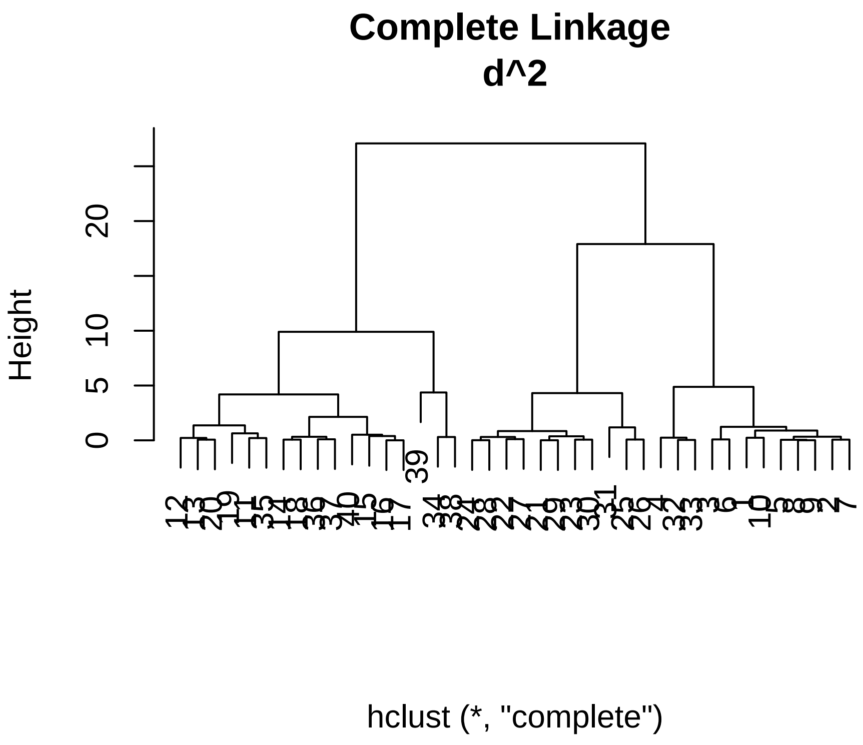
```
  #color coded by cluster
plot(hc.single.c, main="Single Linkage \nd^2", xlab="") #plotting the dendogram
```



```
#complete linkage
hc.complete.c = hclust(d^2, method="complete") #hierarchical clustering for d
  #squared with complete linkage
plot(x, col = cutree(hc.complete.c, k = 4), main="Clusters - d^2 \nComplete Linkage") #plotting the
```



```r
  #data color coded by cluster
plot(hc.complete.c, main="Complete Linkage \nd^2", xlab="") #plotting the dendogram
```



The clustering assigments did not change, but the dendrograms did change.

d. Prove that for single linkage, running agglomerative clustering with dissimilarities d_{ij} and running it again with dissimilarities h(d_{ij}) produces the same sequence of clustering assignments, provided that h is a monotone increasing function. Recall that a monotone increasing function his one such that x <= x' implies h(x) <= h(x'). Prove the same thing for complete

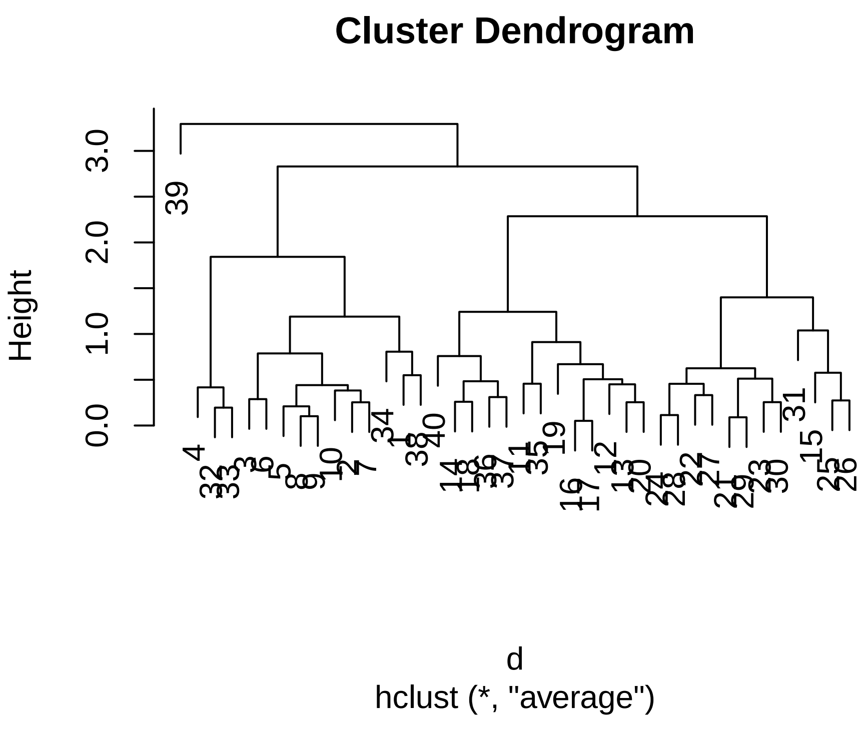
linkage.

Let G be a group of points and H be cluster of points such that $g_i \in$ G and $h_j \in$ H. Let g_j and h_j be points in G and H such that the distance between g_i and h_j is minimized. Thus, G and H will be combined by the single linkage agglomerative clustering algorithm. Monotonic increasing functions preserves ordering, so the distance between h(g_i) and h(h_j) will be still be the minimum distance, and clusters G and H will be combined. Thus running agglomerative clustering with dissimilarities d_{ij} and running it again with dissimilarities hd_{ij} produces the same sequence of clustering assignments for single linkage.

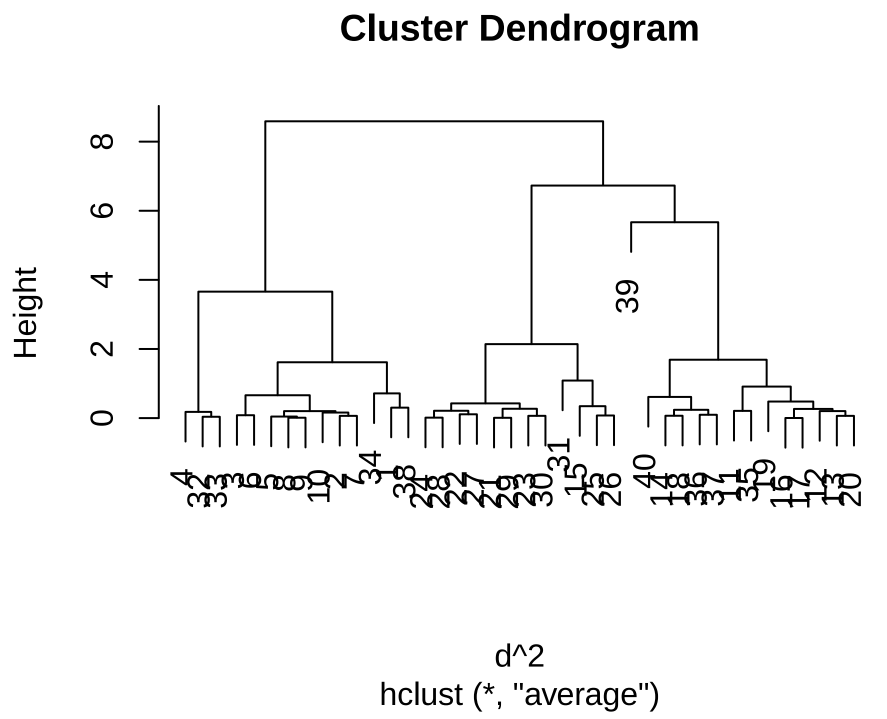
On the other hand, complete linkage utilizes the maximum distance between two points to determine which groups should be merged. Thus, if the distance between g_i and h_j is the largest of all possible points, then the distance between h(g_i) and h(h_j) will also be the largest. This occurs because monotonic increasing functions preserves ordering. Therefore running agglomerative clustering with dissimilarities d_{ij} and running it again with dissimilarities hd_{ij} produces the same sequence of clustering assignments for complete linkage.

e. Run agglomerative clustering with average linkage on each of d and d^2. Cut both trees at K = 4. Are the clustering assignments the same? How about for K = 3? (You should produce plots of x, like the ones you made above, with different colors or pch values in order to display the clustering results.)

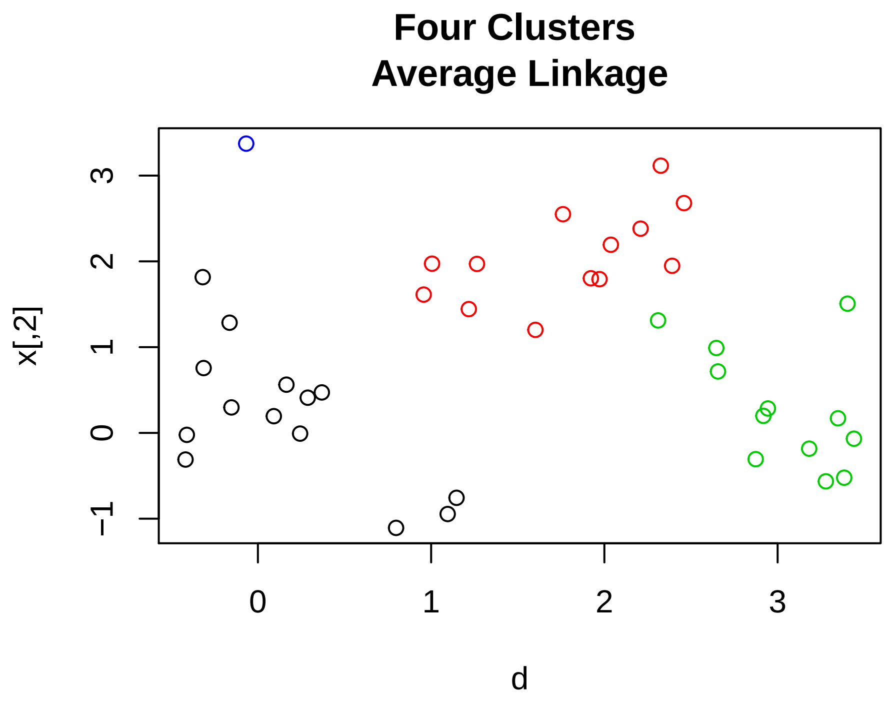
```
hc.average1 = hclust(d, method="average") #clustering with average linkage for d
hc.average2 = hclust(d^2, method="average") #clustering with average linkage for d^2

plot(hc.average1)
```



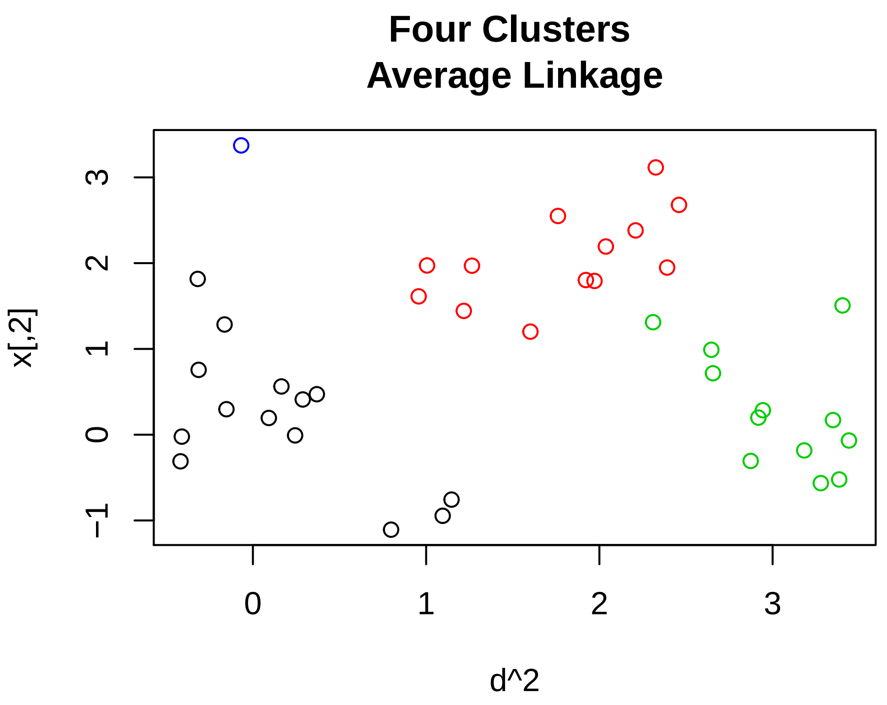
```
plot(hc.average2)
```



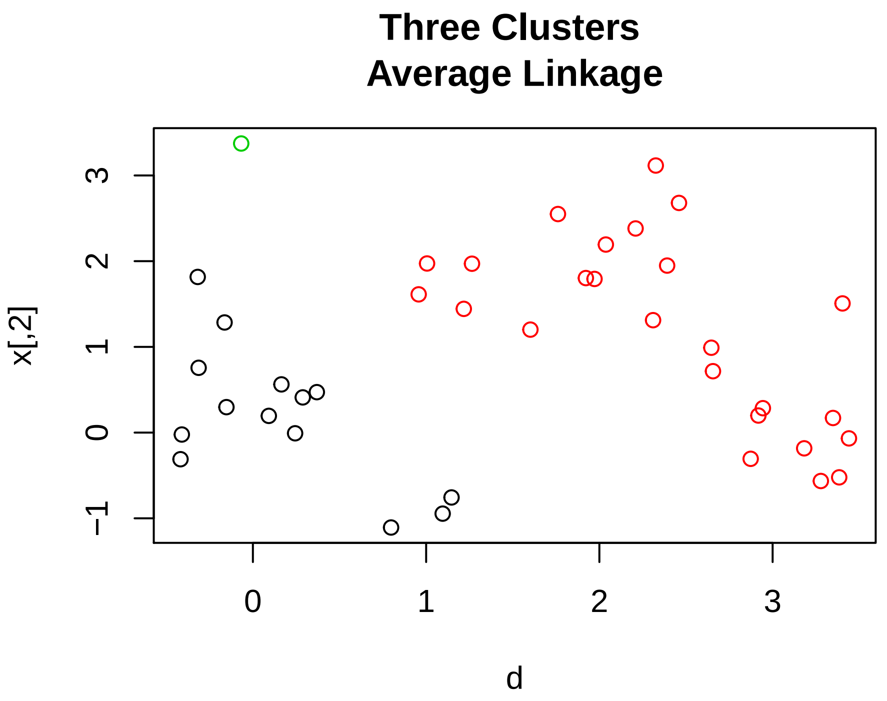
```
four.cut= cutree(hc.average1, k=4); four.cut.2=cutree(hc.average2, k=4)
plot(x, col = four.cut, main="Four Clusters \nAverage Linkage", xlab="d")
```



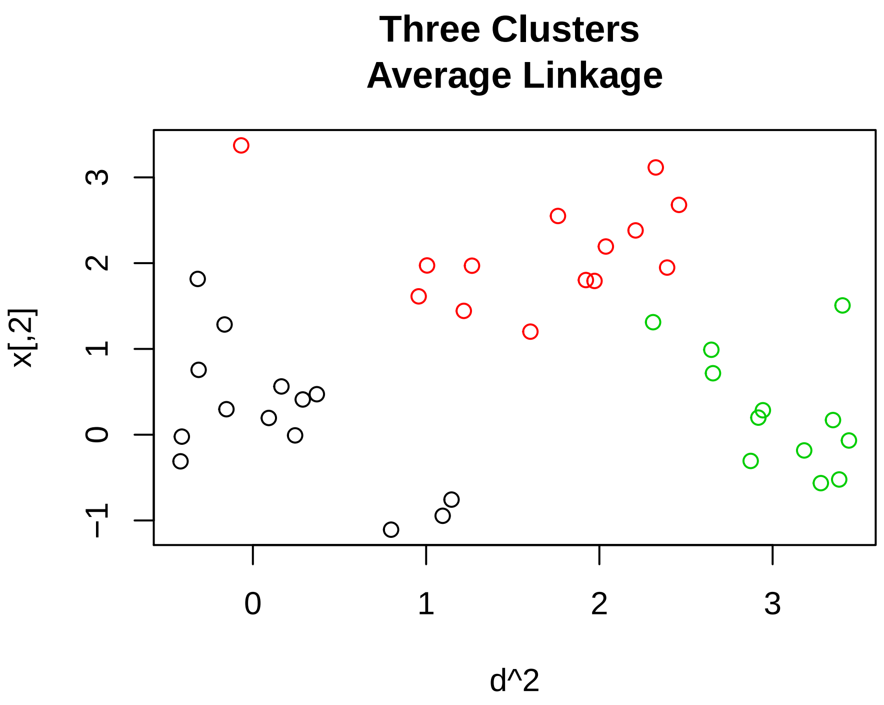
```
plot(x, col = four.cut.2, main="Four Clusters \nAverage Linkage", xlab="d^2")
```



```
three.cut= cutree(hc.average1, k=3); three.cut.2=cutree(hc.average2, k=3)
plot(x, col = three.cut, main="Three Clusters \nAverage Linkage", xlab="d")
```



```
plot(x, col = three.cut.2, main="Three Clusters \nAverage Linkage", xlab="d^2")
```



The clustering assignments are the same for K=4. The assignments are not the same for K=3.