**ICA #12: Clustering Using R (Due Friday, Apr 21 at 9:00 am)**

**What to submit:** a single word/pdf file with answers for the questions in **Part 6**.

**Before you start**

You’ll need two files to do this exercise: **Clustering.r** (the R script file) and **Census2000.csv** (the data file[[1]](#footnote-1)). Both of those files can be found on the course site. The data file contains 32,038 rows of census data for regions across the United States.

*Download both files and save them to the folder where you keep your R files.*

**Part 1: Look at the Data File**

1. Open the **Census2000.csv** data file in Excel. If it warns you, that’s ok. Just click “Yes” and/or “OK.” You’ll see something like this:



This is the raw data for our analysis. Each row represents a citizen respondent to the census.

The input file for a Cluster analysis follows this general format. Each row represents a case, and each data element describes that case. We will use this data set to create groups (clusters) of similar citizens, based on these descriptor variables as dimensions. A citizen of a cluster should be more similar to the other citizens in its cluster than citizens in any other cluster.

For the Census2000 data set, here is the complete list of variables:

|  |  |
| --- | --- |
| **Variable** | **Description** |
| RegionID | postal code of the region |
| RegionLongitude | region longitude |
| RegionLatitude | region latitude |
| RegionDensityPercentile | region population density percentile (1=lowest density, 100=highest density) |
| RegionPopulation | number of people in the region |
| MedianHouseholdIncome | median household income in the region |
| AverageHouseholdSize | average household size in the region |

1. Close the Census2000.csv file. If it asks you to save the file, choose “Don’t Save”.

**Part 2: Explore the Clustering.r Script**

1. Open the **Clustering.r** file in RStudio. This contains the R script that performs the clustering analysis.
2. Look at lines 7 through 29. These contain the parameters for the clustering analysis. Here’s a rundown:

|  |  |  |
| --- | --- | --- |
| INPUT\_FILENAME | Census2000.csv | The data is contained in Census2000.csv |
| PLOT\_FILENAME | ClusteringPlots.pdf | Various plots that describe the input variables and the resulting clusters. Output from the clustering analysis. |
| OUTPUT\_FILENAME | ClusteringOutput.txt | The output from the clustering analysis, including cluster statistics. |
| CLUSTERS\_FILENAME | ClusterContents.csv | More output from the clustering analysis. This file contains the normalized variable scores for each case along with the cluster to which it was assigned. |
| NORMALIZE | 1 | Whether to normalize the data(1 = yes, 0 = no) |
| RM\_OUTLIER | 1 | Whether to remove outliers(1 = yes, 0 = no) |
| MAX\_CLUSTER | 25 | The maximum number of clusters to generate in SSE plot |
| NUM\_CLUSTER | 5 | The number of clusters to generate for solution |
| MAX\_ITERATION | 500 | The number of times the algorithm should refine its clustering effort before stopping.  |
| VAR\_LIST | c("RegionDensityPercentile","MedianHouseholdIncome","AverageHouseholdSize"); | The variables to be included in the analysis (check the first row of the data set, or the table above, for variable names within the Census2000 data set) *By the way, c() is a function that lets you build a list of values.* |

1. Look at lines 33 through 41. These install (when needed) and load the **cluster** and **psych** packages. These perform the clustering analysis and visualization.

**Part 3: Execute the Clustering.r Script**

1. Select Code/Run Region/Run All.

It could take a few seconds to run since the first time it has to install some extra modules to do the analysis. It also takes a little while to perform the clustering analysis. Be patient!

You’ll see a lot of action in the Console window at the bottom left side of the screen, ending with this:



**Part 4: Reading Plots**

1. Now minimize RStudio and open the **ClusteringPlots.pdf** file in your working directory (the folder with your Clustering.r script).
2. On page 1 of the ClusteringPlots.pdf file you’ll see this graphic:


These are histograms for the three variables used to cluster the cases. These variables were specified in line 29 of the script using the VAR\_LIST variable:


We can see that **MedianHouseholdIncome** and **AverageHouseholdSize** have a slightly *right-skewed* distribution.

**RegionDensityPercentile** looks a lot different – that’s because the measure is percentile, so the frequency is the same for each level of x. Think of it this way – if you have 100 things ordered from lowest to highest, the top 10% will have 10 items, the next highest 10% will have 10 items, etc.



1. Now look at the line graph on page 2 of the of ClusterPlots.pdf:

This shows the total **within-cluster sum of squares error** (i.e. **within-cluster SSE**) as the number of clusters increase. As we would expect, the error decreases within a clusters when the data is split into more clusters.

We can also see that **the benefit of increasing the number of clusters decreases as the number of clusters increases**. The biggest additional benefit is from going from 2 to 3 clusters (of course you’d see a big benefit going from 1 to 2 – 1 cluster is really not clustering at all!).
We see things really start to flatten out around 10 to 12 clusters. We probably wouldn’t want to create a solution with more clusters than that.
2. From line 27 of our script we know that we specified our solution to have 5 clusters:



Now look at the pie chart on page 3 of the ClusterPlots.pdf:



This shows the relative size of those five clusters. The size is just the number of observations that were placed into each cluster. So Cluster #3 is the largest cluster with 8,726 observations.

**Part 4: Reading Output**

1. Now open the file **ClusteringOutput.txt**.

*It will also be in the folder with your Clustering.r script. You can open it in RStudio by choosing File|Open File.*

**Reading Summary Statistics**

1. The first thing you’ll see are the summary statistics for each variable (about lines 6 through 10):

> describe(inputFile[,VAR\_LIST])

 vars n mean sd median trimmed mad min max range skew

RegionDensityPercentile 1 31951 50.83 28.68 51.00 50.83 37.06 1 100.00 99.00 0.00

MedianHouseholdIncome 2 32038 39393.72 16426.57 36146.00 37525.23 11230.69 0 200001.00 200001.00 1.99

AverageHouseholdSize 3 32038 2.57 0.42 2.56 2.56 0.27 0 8.49 8.49 -0.01

1. Now look the statistics about the case count (a case is an observation; a row of data). You’ll find this by scrolling to lines 54 through 70.

> # Display some quick stats about the cleaning process

> print("Total number of original cases:")

[1] "Total number of original cases:"

> nrow(inputMatrix)

[1] 32038

> print("Total number of cases with no missing data:")

[1] "Total number of cases with no missing data:"

> numNoMissing

[1] 31951

> print("Total number of cases without missing data and without outliers:")

[1] "Total number of cases without missing data and without outliers:"

> nrow(kData)

[1] 30892

You can see that we started with 32,038 observations. When we removed the cases with missing data for one or more of the variables we were left with 31,951 observations. And when we removed the outliers we have 30,892.

By the way, if you add together the cluster sizes from that pie chart on the last page, you get…30,892! So all of our cleaned data (no missing data, no outliers) are accounted for in our set of clusters!

1. Lines 96 through 100 display the size of each cluster. Note that this matches up with the earlier pie chart:

> # Display the cluster sizes

> print("Cluster size:")

[1] "Cluster size:"

> MyKMeans$size

[1] 7257 7986 8726 2090 4833

1. Now look around lines 111 through 117. This is the first part of the summary statistics for each cluster. Specifically, these are the normalized cluster means:

> aggregate(kData,by=list(MyKMeans$cluster),FUN=mean)

 Group.1 RegionDensityPercentile MedianHouseholdIncome AverageHouseholdSize

1 1 0.8835353 -0.2681572 -0.5992532

2 2 -1.1224866 -0.5594931 -0.5055258

3 3 -0.4836836 -0.1400215 0.3502257

4 4 0.9557776 -0.3145690 1.3672892

5 5 0.8561222 1.3520755 0.2790425

**The cluster means are normalized values because the original data was normalized before it was clustered.** This was done in the R script in lines 82 and 83:

if (NORMALIZE == 1) {

 kData <- scale(kData)}

*What the scale() function does is that it will first "center" each column by subtracting the column means from the corresponding column, and then "rescale" each column by dividing each (centered) column by its standard deviation.*

|  |
| --- |
| **Why do we need to normalize the data?**It is important to normalize the data so that it is all on the same scale. This keeps data with large values from skewing the results; those variables will have larger variance and will have a greater influence on the clustering algorithm. For example, a typical value for household income is going to be much larger than a typical value for household size, and the variance will therefore be larger. By normalizing, we can be sure that each variable will have the same influence on the composition of the clusters.**How to interpret normalized values?**For normalized values, “0” is the average value for that variable in the population. Look at lines 40 through 44 of ClusteringOutput.txt. This is the summary statistics for the normalized data.> describe(kData); vars n mean sd median trimmed mad min max range skew kurtosis seRegionDensityPercentile 1 31951 0 1 0.01 0.00 1.29 -1.74 1.71 3.45 0.00 -1.20 0.01MedianHouseholdIncome 2 31951 0 1 -0.20 -0.12 0.69 -2.42 9.83 12.26 2.05 9.24 0.01AverageHouseholdSize 3 31951 0 1 -0.05 -0.04 0.67 -6.50 14.87 21.37 0.67 11.57 0.01From the summary statistics for after normalization, the population mean for each variable becomes 0. |

So now let’s look at cluster (group) 1:

 Group.1 RegionDensityPercentile MedianHouseholdIncome AverageHouseholdSize

1 1 0.8835353 -0.2681572 -0.5992532

Looking at the normalized values for cluster 1: the average MedianHouseholdIncome (-0.268) and AverageHouseholdSize (-0.599) for group 1 are negative, thus are below the population average (i.e. 0). The average RegionDenistyPercentile (0.884) is positive, thus is above the population average (i.e. 0). In other words, the regions in cluster 1 are more dense, have lower income, and fewer people in their families than the overall population.

Contrast that with cluster (group) 5:

 Group.1 RegionDensityPercentile MedianHouseholdIncome AverageHouseholdSize

5 5 0.8561222 1.3520755 0.2790425

This group has a higher than average RegionDensityPercentile (0.856), AverageHouseholdSize (0.279), and MedianHouseholdIncome (1.352) than the population average (i.e. 0). In other words, these regions are more dense, have more people in their families than the overall population average, and have higher income than the overall population.

1. Detailed descriptive statistics for each group are listed below the summary of means (around lines 123 through 152):

> describeBy(kData, MyKMeans$cluster)

INDICES: 1

 vars n mean sd median trimmed mad min max range skew kurtosis se

RegionDensityPercentile 1 7257 0.88 0.50 0.88 0.89 0.62 -0.31 1.71 2.02 -0.06 -1.13 0.01

MedianHouseholdIncome 2 7257 -0.27 0.54 -0.25 -0.26 0.53 -2.27 1.86 4.13 -0.15 0.15 0.01

AverageHouseholdSize 3 7257 -0.60 0.60 -0.51 -0.54 0.49 -3.00 0.40 3.40 -1.16 1.70 0.01

-------------------------------------------------------------------------------------

INDICES: 2

 vars n mean sd median trimmed mad min max range skew kurtosis se

RegionDensityPercentile 1 7986 -1.12 0.41 -1.18 -1.15 0.47 -1.74 0.01 1.74 0.50 -0.61 0.00

MedianHouseholdIncome 2 7986 -0.56 0.45 -0.56 -0.57 0.39 -2.27 2.49 4.75 0.48 2.53 0.01

AverageHouseholdSize 3 7986 -0.51 0.51 -0.43 -0.46 0.41 -2.97 1.03 4.00 -1.06 2.47 0.01

-------------------------------------------------------------------------------------

INDICES: 3

 vars n mean sd median trimmed mad min max range skew kurtosis se

RegionDensityPercentile 1 8726 -0.48 0.50 -0.41 -0.45 0.52 -1.74 0.56 2.30 -0.46 -0.47 0.01

MedianHouseholdIncome 2 8726 -0.14 0.50 -0.15 -0.14 0.49 -2.04 2.21 4.24 0.02 0.34 0.01

AverageHouseholdSize 3 8726 0.35 0.53 0.25 0.29 0.41 -0.83 2.99 3.83 1.44 3.27 0.01

-------------------------------------------------------------------------------------

INDICES: 4

 vars n mean sd median trimmed mad min max range skew kurtosis se

RegionDensityPercentile 1 2090 0.96 0.60 1.02 1.01 0.70 -1.35 1.71 3.07 -0.68 -0.01 0.01

MedianHouseholdIncome 2 2090 -0.31 0.65 -0.28 -0.30 0.67 -2.27 1.76 4.03 -0.21 -0.25 0.01

AverageHouseholdSize 3 2090 1.37 0.72 1.21 1.31 0.78 0.35 2.99 2.64 0.57 -0.76 0.02

-------------------------------------------------------------------------------------

INDICES: 5

 vars n mean sd median trimmed mad min max range skew kurtosis se

RegionDensityPercentile 1 4833 0.86 0.50 0.88 0.88 0.52 -1.74 1.71 3.45 -0.64 0.73 0.01

MedianHouseholdIncome 2 4833 1.35 0.63 1.26 1.30 0.66 0.26 2.99 2.73 0.66 -0.35 0.01

AverageHouseholdSize 3 4833 0.28 0.64 0.30 0.30 0.56 -2.95 2.92 5.86 -0.46 1.94 0.01

**Within-Cluster SSE (Cohesion) and Between-Cluster SSE (Separation)**

1. We want to better understand the “quality” of the clusters. Let’s look at the within-cluster sum of squares error (i.e. within-cluster SSE). In R, it is called “**withinss**.” The within-cluster SSE measures **cohesion** – how similar the observations within a cluster are to each other.

The following are the lines which contain that statistic (around lines 154 through 159):

> # Display withinss (i.e. the within-cluster SSE for each cluster)

> print("Within cluster SSE for each cluster (Cohesion):")

[1] "Within cluster SSE for each cluster (Cohesion):"

**withinss error for each cluster
(cohesion)**

> MyKMeans$withinss

[1] 6523.491 4990.183 6772.426 2707.390 5102.896

These are presented in order, so 6523.491 is the withinss for cluster 1, 4990.183 is the withinss for cluster 2, etc. We can use this to compare the cohesion of this set of clusters to another set of clusters we will create later using the same data.

**IMPORTANT:**

**Generally, we want higher cohesion; that means less error. So the smaller these withinss values are, the lower the error, the higher the cohesion, and the better the clusters.**

1. Finally, look at the sum of squares errors between clusters (i.e. between-cluster SSE). In R, it is called “**betweenss**”. The between-cluster SSE measures **separation** – how different the clusters are from each other (cluster 1 vs. cluster 2, cluster 1 vs. cluster 3, etc.).

The following are the lines which contain that statistic (around lines 161 through 173):

> # Display betweenss (i.e. the between-cluster SSE between clusters)

> print("Total between-cluster SSE (Seperation):")

[1] "Total between-cluster SSE (Seperation):"

**total betweenss error**

> MyKMeans$betweenss

[1] 45301.67

> # Compute average separation: more clusters = less separation

> print("Average between-cluster SSE:");

[1] "Average between-cluster SSE:"

**average betweenss error
(separation)**

> MyKMeans$betweenss/NUM\_CLUSTER

[1] 9060.334

We are interested in the average betweenss. That gives us the average difference between clusters. Again, we can use this to compare the separation of this set of clusters to another set of clusters we will create later using the same data.

**IMPORTANT:**

**Generally, we want higher separation; that means higher SSE between clusters. So the larger the average betweenss value is, the higher the separation, and the better the clusters.**

1. Close the ClusteringOutput.txt file in RStudio.

**Part 5: Comparing Two Sets of Clustering Results**

Now we’re going to create another set of clusters (10 clusters instead of 5) and examine the withinss and betweenss to understand the tradeoff between the number of clusters, cohesion, and separation.

1. Return to the Cluster.R file in RStudio.
2. Look at line 27:



Change this value from 5 to 10.

1. Re-run the script. Select Code/Run Region/Run All.
2. When it’s done, open **ClusteringPlots.pdf**. You’ll see a new pie chart:

Now there are 10 clusters instead of 5. Remember, this is the same data, just organized differently.

*Obviously the cluster sizes are smaller than they were before because we’re dividing up the observations into more groups.*

*For fun, you can also observe that the histograms and the box plots look the same as before. This is because we’re working with the same set of data, so the overall means, standard deviations, and distributions are the same.*

1. Close ClusteringPlots.pdf.
2. Open ClusteringOutput.txt in RStudio.
3. You’ll notice now, in the cluster means section (around line 107 of the output) there are 10 clusters:

> aggregate(kData,by=list(MyKMeans$cluster),FUN=mean);

 Group.1 RegionDensityPercentile MedianHouseholdIncome AverageHouseholdSize

1 1 -0.04084547 -0.4907062 -0.20704689

2 2 -0.21870682 0.4282854 0.34956210

3 3 -1.07772451 -0.4149886 -0.03061695

4 4 0.95228933 0.3572517 0.05807308

5 5 0.91455361 1.7790754 0.58241124

6 6 1.10265544 -0.4061357 1.52365057

7 7 1.17939105 0.8699481 -1.14659537

8 8 -1.01271967 -0.4669490 1.34135444

9 9 1.13001595 -0.6123089 -0.79911649

10 10 -1.29298695 -0.6151248 -0.93628733

You will observe that cluster 5 has the highest median household income, while cluster 6 has the highest average household size. Because these values are *normalized*, you aren’t looking at the actual values (i.e., the number of people in an average household). But it does let you compare clusters to each other.

Most importantly, we can compare the withinss and betweenss statistics for this new set of clusters to our previous configuration of 5 clusters (around line 185 of the output file):

> # Display withinss (i.e. the within-cluster SSE for each cluster)

> print("Within cluster SSE for each cluster (Cohesion):")

[1] "Within cluster SSE for each cluster (Cohesion):"

> MyKMeans$withinss

 [1] 1951.253 1770.433 1930.194 1500.782 1830.547 1621.774 1185.288 1256.886 2035.843 1805.546

> # Display betweenss (i.e. the SSE between clusters)

> print("Total between-cluster SSE (Seperation):")

[1] "Total between-cluster SSE (Seperation):"

> MyKMeans$betweenss

[1] 54509.51

> # Compute average separation: more clusters = less separation

> print("Average between-cluster SSE:")

[1] "Average between-cluster SSE:"

> MyKMeans$betweenss/NUM\_CLUSTER

[1] 5450.951

We can see that the withinss error ranges from 1185.288 (cluster 7) to 2035.843 (cluster 9). Compare this to our 5 cluster solution, where withinss ranges from 2707.390 (cluster 4) to 6772.426 (cluster 3). The withinss error is clearly lower for our 10 cluster solution; those clusters have **higher cohesion** than our 5 cluster solution. This makes sense – if we put our observations into more clusters, we’d expect those clusters to (1) be smaller and (2) more similar to each other.

However, we can see that the separation is lower (i.e., worse) in our 10 cluster solution. For the 10 cluster solution, the average betweenss error is 5450.951; for the 5 cluster solution, the average betweenss error was 9060.334. This means the clusters in our current solution have **lower separation** than our 5 cluster solution. This also makes sense – if we have more clusters using the same data, we’d expect those clusters to be closer together.

**How many clusters should I choose?**

**So our 10 cluster solution (compared to the 5 cluster solution) has (1) higher cohesion (good) but (2) lower separation (bad).** How do we decide which one is better?

As you might expect, there’s single answer, but the general principle is to obtain a solution with the fewest clusters of the highest quality. A solution with fewer clusters is appealing because it is simpler. Take our census example: It is easier to explain the composition of five segments of population regions than 10. Also when separation is lower you’ll have a more difficult time coming up with meaningful distinctions between them – the means for each variable across clusters will get more and more similar.

However, too few clusters are also meaningless. You may get higher separation but the cohesion will be lower. This means there is such variance within the cluster (withinss error) that the average variable value doesn’t really describe the observations in that cluster.

To see how that works, let’s take a hypothetical list of six exam scores: 100, 95, 90, 25, 20, 15

If these were all in a single cluster, the mean exam score would be 57.5. But none of those values are even close to that score – the closest we get is 32.5 points away (90 and 25). If we created two clusters:

100, 95, 90 AND 25, 20, 15

Then our cluster averages would be 95 (group 1) and 20 (group 2). Now the scores in each group are much closer to their group means – no more than 5 points away.

So here’s what you can do:

1. Choose solutions with the fewest possible clusters.
2. But also make sure the clusters means are describing distinct groups.
3. Make sure that the range of values on each variable within a cluster is not too large to be useful.

**Part 6: Try it yourself**

Use the Clustering.r script and the same Census2000.csv dataset to create a set of 7 clusters.

1. What is the size of the largest cluster?
2. Compare the characteristics (RegionDensityPercentile, MedianHouseholdIncome, and AverageHouseholdSize) of cluster 3 to the population as a whole?
3. What is the range of withinss error for those 7 clusters (lowest to highest)?
4. Is the cohesion generally higher or lower than the 5 cluster solution?
5. What is the average betweenss error for those 7 clusters?
6. Is the separation higher or lower than the 10 cluster solution?
7. Based on the analyses we have done so far, use your own words to summarize how the number of clusters can affect withinss error (cohesion) and betweenss error (separation).
1. Adapted from SAS Enterprise Miner sample data set. [↑](#footnote-ref-1)