WI(NE)TWORKS

Zach Brown, Bonnie Lin, Kelly Yang

12/12/2018

Section 1 - Introduction

In short, networks represent complex relationships. Networks are used in a variety of fields, examples of which include describing online social interactions, biological cell development, and brain connectivity. Learning the ropes of network science, or the science of connectivity, by developing the ability to group and cluster data is very helpful, particularly for dealing with complex data in the field of statistics & data science. The development of network analysis tools flourished as media companies, such as Facebook, created networks and grouping mechanisms that acted not only as a basis for social connections, but also informed targeted advertising and recommender systems. The algorithms that support these networks and clusters are technically very complex, and the goal of our project is to explore and illustrate some of the methods that are used to create the complex networks that affect our daily lives.

Section 2 - Data Description & Mining

2.1 - Overview

This project uses a dataset, made publicly available on Kaggle, that contains approximately 130,000 wine reviews (https://www.kaggle.com/zynicide/wine-reviews). The author presents two versions of the dataset, both of which are *csv* files. One with 150,000 observations of 11 variables, and the other with 130,000 observations of 14 variables. This project uses the latter version, in which the original author has added different fields for each observation. For instance, variables, like the taster's name, the taster's twitter handle, and the title of their reviews, are added to the observations.

As we can see, the data set has 129,971 different wines and 14 different fields (columns are the variables).

2.2 - Variables Explained

Below is the complete list of variables included in the data set. The descriptions are taken directly from the Kaggle page (link posted in section 2.1) from which we downloaded the data file.

X1 - [no description given, appears to be an identification variable]

country - The country that the wine is from

description - [no further description given]

designation - The vineyard within the winery where the grapes that made the wine are from

points - The number of points WineEnthusiast rated the wine on a scale of 1-100 (though they say they only post reviews for wines that score >=80)

price - The cost for a bottle of the wine

province - The province or state that the wine is from

region_1 - The wine growing area in a province or state (ie Napa)

region_2 - Sometimes there are more specific regions specified within a wine growing area (ie Rutherford inside the Napa Valley), but this value can sometimes be blank

taster_name - [no further description given]

taster_twitter_handle - [no further description given]

title - The title of the wine review, which often contains the vintage if you're interested in extracting that feature

variety - The type of grapes used to make the wine (ie Pinot Noir)

winery - The winery that made the wine

2.3 - Variables of Interest

Before moving on to our exploration of networks, we had to isolate a select group of variables that would be appropriate for creating graph objects, in preparation for creating clusters and networks. In doing this, we ensured that we had both numerical and categorical variables, with the prior understanding that the building blocks of clusters and networks treat these variables slightly different. These mechanics will be discussed in a later section.

2.3.1 - Initial Intuitions

Upon initially seeing the data set, there were a couple of variables that we were interested in analyzing for clustering. These variables were points, price, and country. In addition to directly getting these three variables from the downloaded file, we were also interested in having a variable for vintage (year). The creator of the original data set made sure to include the year within all of the wine titles, so our next step was to create another column in the data set that held the vintages by parsing the vintages from the titles with some basic text mining, as we show below:

```
wine <- wine_orig %>%
  mutate(year = str_extract(wine_orig$title, "[1|2][0|1|9][0-9][0-9]"),
         year = as.numeric(year))
```

2.3.2 - Missingness

Based on the basic descriptions listed above, we selected variables to move forward in creating graph objects for two primary reasons. The first is that these variables were not missing as many observations as others. For instance, there are only 63 wines with missing countries compared to 26244 wines with missing taster names. In the following code, we demonstrate the missingness of the variables in question.

```
sum(is.na(wine_orig$taster_name)) ##The number of missing taster names
```

```
## [1] 26244
```

```
sum(is.na(wine_orig$country)) ##The number wines with missing country names
```

```
## [1] 63
```

```
sum(is.na(wine_orig$region_2)) ## Region_1 gets removed for another reason below
```

```
## [1] 79460
```

```
sum(is.na(wine_orig$variety)) ##Almost all of the wine varieties (types of grape) are included
```

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

Due to the high number of missing data points, we decided to remove all of the variables associated with the tasters from our dataset and analysis.

2.3.3 - Uniqueness

Next, we wanted to look at how many unique observations existed for each type of variable. This would be a concern within the building of networks if there were so many different unique categories, such that the difference would cause the networks to view wines as more dissimilar than necessary. Since it is a categorical identifier, we expected to remove the *X1* variable from our dataset, but we wanted to see what other variables had high counts of unique observations, as the code performs below:

```
sapply(wine, function(x) length(unique(x)))
```

```
##                   X1              country          description
##               129971                   44               119955
##          designation               points                price
##                37980                   21                  391
##             province            region_1            region_2
##                  426                 1230                   18
##          taster_name taster_twitter_handle                title
##                   20                   16               118840
##              variety               winery                 year
##                  708                16757                   65
```

From the output, we see that province may be a variable worth keeping dependent on the size of the samples taken for creating the networks. We see that *region_1* variable has 1230 unique observations. This could prove a problem dependent on how many wines we sample to create our clusters. Say we had a sample of 10000+ observations this may not have been *that* big of an issue, but we found that we had to subset much smaller samples, due to memory/data storage issues in R.

Additionally, to maintain a reasonable scope for the project, we also expected to avoid the more complicated text mining/analysis aspects by excluding the *description* variable.

2.3.4 - Creating the Functional Data Set for Analysis

Upon deeper thought, we decided to remove *province* from our analytical exploration because it would lead to us having two of six variables (contributing to the building of our clusters) being related to geography. Instead, we decided that it would make more sense to include variety and have five variables of interest (price, points, year, country, and variety). The next step consists of filtering the data such that our functioning data set only included these variables, preparing them in the right data type for making the dissimilarity matrix, and omitting non-complete cases.

```
wine_interestedvars <- wine %>%
  select(price, year, country, points, variety) %>%
  mutate(country = as.factor(country),
         variety = as.factor(variety))
wine_interestedvars_country <- wine_interestedvars %>%
  filter(country == "US" || country == "France" || country == "Spain" ||
           country == "Portugal" || country == "Italy")%>%
  na.omit()

nrow(wine_interestedvars_country)
```

```
## [1] 116765
```

We also decided to filter the data for wines from the countries with the most frequent occurrence within the data set.

This maintains a large number of complete case data points (116,765) compared to our original set of approximately 130,000.

Section 3 - Brief Overview of Cluster & Network Analysis

3.1 - Network Analysis Overview

Network building & cluster analysis work hand in hand, particularly when it comes to data visualization. Creating networks involves a mechanical approach that connects individual data points based on similar properties (both numerical and categorical). Cluster analysis is a little bit more free as there are many different methods and approaches to aggregating different parts of these networks into discernible groups.

Our process for creating networks began with taking a sample from the original data set. Each data point sampled then becomes a **node**, an element of the network. After this, we utilize the *daisy* function (a function described in section the next section) to establish connections between these nodes. These connections, at their core, evaluate the degree to which nodes are similar based on the explanatory variables we use in our investigation.

To determine the degree to which nodes are similar, we run a function (*daisy*) that assigns a **dissimilarity** score to each pair of nodes (dissimilarity being the degree to which two nodes are different). We then determine a threshold for determining an **edge**, or a connection between nodes. In **Section 4**, we decide that if the dissimilarity score is below .35, then the two data points are similar enough to warrant the creation of an edge in our network.

3.2 - Building Clusters and Contextual Applications

After we have established these networks (a collection of edges and nodes), we can then try to aggregate nodes into specific groupings. There are several different methods used for creating these clusters which will be discussed in a later section. One of the major inferential pieces of cluster analysis is related to **modularity**, a numerical measure of the strength of the division amongst separate communities.

In the context of our project, we saw different clusterings that were based on both quantitative (*price/vintage*) and categorical (*country/variety*) factors. One of the major goals of cluster analysis is to create different groupings or communities of nodes (wines in our case) that are based on a variety of factors rather than only known variables. The application of these created clusters in the context of our project could contribute to further recognizing similarities between wines. Because our project was more exploratory in nature, we only looked at rather simplistic variables in the creation of our networks and clusters for the purposes of understanding the foundational principles. In further iterations of the project, we could look at some of the other variables (possibly including some text mining of descriptions) to determine wine similarities on a broader scale. These similarities could then be used in applications or search functions where a user inputs a wine they like and receive recommendations for other ones they may enjoy based on the created clusters.

Section 4 - Explanation of Clustering Algorithms

4.1 - Primary Statistical Packages

4.1.1 - Cluster (https://cran.r-project.org/web/packages/cluster/cluster.pdf)

Within the overall cluster package, our analysis utilizes the *daisy* function in R. This function will be the mechanism that executes the dissimilarity matrix used for creating our networks. This function creates dissimilarity scores for each pair of nodes *(0 if two nodes are exactly the same and 1 if they are extremely different)*. Based on this, we can generate a threshold that would determine if an edge should be placed between two nodes. Below is an annotated walkthrough of how this process is capable of generating the edges of a network.

```
set.seed(1234)

sample.wine <- wine_interestedvars %>%
  distinct() %>%
  sample_n(25)
```

This pulls a sample of 25 unique wines from the functional data set created in *section 2.3.4*.

```
##Creates the dissimilarity matrix
sample.wine.dis <- daisy(sample.wine, metric = "gower", stand = FALSE)
sample.wine.dis.matrix <- as.matrix(sample.wine.dis)##Converts daisy object to a matrix
##Initializes the edge creation mechanism
sample.wine.dis.edge <- matrix(rep(0, 25^2), nrow = 25, ncol = 25)

for(i in 1:25){
  for(j in 1:25){
    sample.wine.dis.edge[i,j] = ifelse(sample.wine.dis.matrix[i,j] < 0.35, 0, 1)
  }
}
```

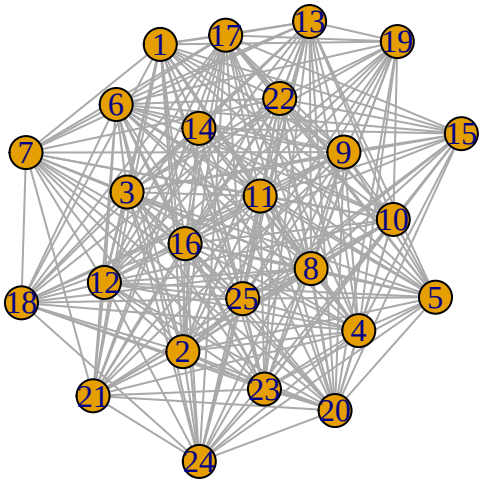
This is how we generate the edges used in all of our networks and clusters. An important argument of note within the *daisy* function is the chosen metric. Dissimilarities calculated using numeric variables often utilize Euclidean Distances (root sum-of-squares of differences) or Manhattan/City-block distances (sum of absolute differences). However, because we have non-numerical variable types (*variety* & *country*), we must utilize "Gower's distance" (which uses a standardization technique). This information can be found and further investigated by typing **?daisy** in the console and reading the R documentation.

We also establish the threshold for which we decide if a pair of nodes is similar enough to warrant an edge. We utilized a little bit of trial and error to determine that .35 was a good dissimilarity threshold for creating edges.

4.1.2 - igraph

This package is very effective in creating visual networks rather quickly and efficiently. Below is an example visual of a wine network created using the sample drawn in the previous section in conjunction with the edge matrix.

```
sample.graph <- graph_from_adjacency_matrix(sample.wine.dis.edge, mode = "upper")
plot(sample.graph)
```



4.2 - Clustering Algorithms

The *igraph* package has a variety of community detection functions that create clusters based on previously created networks. Below are six different algorithms that *igraph* has to generate clusters. This information was gathered from the following link : (https://stackoverflow.com/questions/9471906/what-are-the-differences-between-community-detection-algorithms-in-igraph).

4.2.1 - Walktrap (Random Walks)

This method generates clusters via random walks. The default setting is set to four steps, but this method essentially works by performing a large number of random walks. Over time, communities/clusters will develop as these random walks will only travel to a small grouping of nodes (i.e. the walk goes through many edges within communities and seldom travels on edges that link separate communities). This method is a good introductory approach to creating community networks because it runs rather quickly compared to other methods.

4.2.2 - Edge Betweenness

This is a top-down, hierarchical approach. It operates by asserting edges certain scores dependent on how many shorter paths exists that connect the two nodes associated with the edge of interest. Edges are removed in decreasing score order. The motivating idea behind this method is that edges with high scores connect different groups (as there are lots of smaller edges underneath them). Because of the way this method operates, it is very time consuming. The source we looked at noted that feasibility of large networks starts to diminish once you have roughly 700 nodes with approximately 3500 edges. For this reason, we will have to use smaller samples throughout the rest of the project for this method compared to some of the other algorithms we use.

4.2.3 - Fastgreedy

The Fastgreedy method is somewhat of the antithesis of *edge betweenness*. Though also a hierarchical approach, the Fastgreedy method is a "bottoms up" approach. Every node starts as its own independent community. Nodes merge locally to optimize modularity (for every new connection). This iterative process stops when local modularity can no longer increase. This method fits its name very well. It executes very quickly and is "greedy" in the sense that nodes can be added to clusters rather easily even if there is a high

degree of dissimilarity. As long as there is an increase in modularity, nodes will merge together into one community.

4.2.4 - Spinglass

This method stems from the field of statistical physics as it can best be explained by comparing nodes and edges to particles and interactions (respectively). This method begins by appropriating a given *spin state* value to all of the different nodes within the network. This notion of a *spin state* has roots in the orientation of elementary particles (for example the direction that electrons are spinning around its axis). Within the field of statistics, the concept of spin merely operates as a baseline for similarity of nodes.

This algorithm begins with randomly assigning each node a different *spin state* (the default argument in "igraph" sets *spins = 25*, but this number can go as high as 200). Then, based on the number of interactions (edges) between the nodes, the algorithm either leaves the nodes in their current states or merges them to the same state. The algorithm simulates this problem many times and then the output is a defined number of communities. One interesting property of this method is that the highest number of communities that can be output is the equal to the *spins* argument. Therefore, if you wanted to only have the network finish in four or fewer communities, you could set *spins = 4*. One of the pitfalls of this method, however, is that because the initial spin *spin state* of each node is random,two dissimilar nodes could be placed in the same community by chance. This particularly an issue for nodes that do not have any edges.

4.2.5 - Leading Eigenvector

Also a top-down hierarchical approach, this method separates nodes into to groups such that the separation optimizes the modularity function. These splits are discovered by evaluating the leading eigenvector of the modularity matrix until the groups created are very tightly knit.

4.2.6 - Label Propagation

This method can be thought of somewhat similar to the way a simulation operates. All of the nodes in the network are assigned labels. The nodes are then assigned donor labels based on their surrounding labels. This process stops when the label of the target node is one of the most frequent labels in its general "neighborhood". This method calculates very quickly, but is very dependent on the random assortment and/or placement when labels are first assigned. Therefore, it makes sense to run this method many different times (500+) and take note of how the communities are created.

Section 5 - Data Analysis

5.1 - Analytical Overview

We will now begin our analysis. We will start with pulling a random sample of 100 observations from our cleaned data. In earlier iterations of this analysis, we attempted (and successfully executed on 4 of the 6 algorithms) an $n > 1000$ sample size. However, the *edge-betweenness* & *spinglass* algorithms caused system terminations due to memory issues and excess run times. We will organize the analysis by looking at each

```
set.seed(1234)
wine.100 <- wine_interestedvars_country%>%
  sample_n(100) %>%
  na.omit()
```

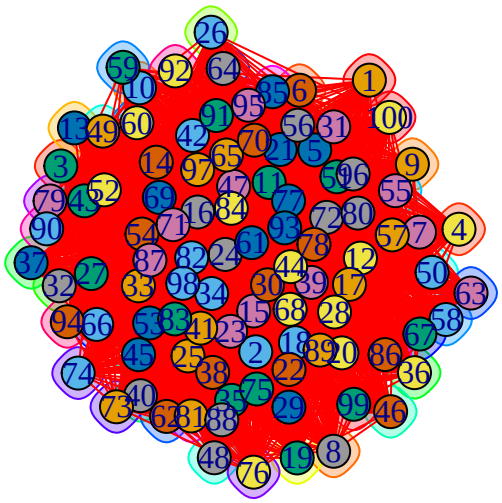
To summarize the earlier procedure laid out for the small subset of 25 observations, we go through the following steps:

1)Create a dissimilarity matrix with the `daisy()` function **2)**Define a threshold, which we tuned to generate the most conclusive groups **3)**Create group objects using the `graph_from_adjacencymatrix()` function from the *igraph* package **4)**Use the membership and modularity score generated from each algorithm to compare its performance

Below is the code used for steps 1-3. We will complete step **(4)** on a case by case basis for each algorithm.

5.2 - Walktrap

```
set.seed(100)
wt.100 <- walktrap.community(graph.100)
plot(wt.100, graph.100)
```



```
membership(wt.100)
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

This sample provides a network that does not have any defined communities. Rather, each node is representing its own cluster. In addition, creating a tally for each group does not seem appropriate as there are simply too many groups. This is shown using the membership command above. Because of this, we would not expect a high modularity score either.

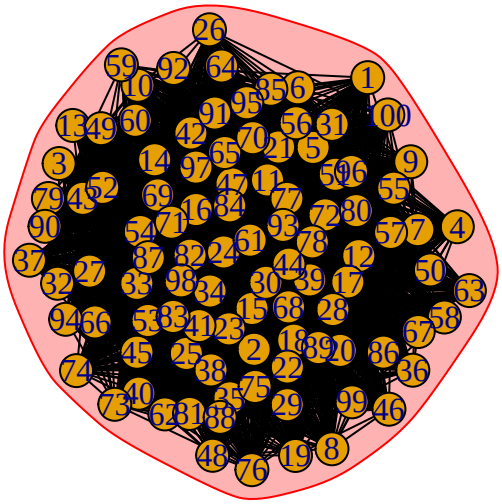
```
modularity(graph.100, membership(wt.100))
```

```
## [1] -0.01027502
```

As predicted, this method produces a low modularity score.

5.3 - Edge Betweenness

```
set.seed(100)
ebc.100 <- edge.betweenness.community(graph.100)
plot(ebc.100, graph.100)
```



```
sizes(ebc.100)
```

```
## Community sizes
##   1
## 100
```

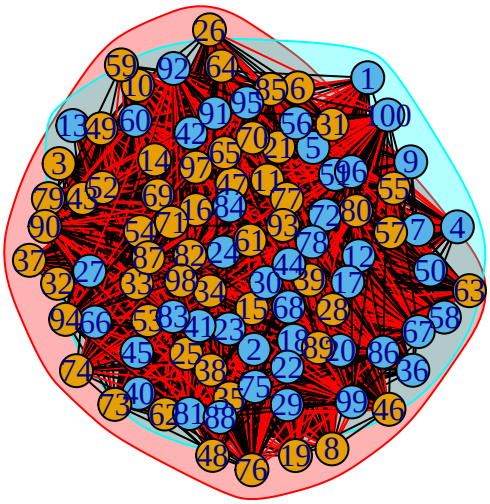
```
modularity(graph.100, membership(ebc.100))
```

```
## [1] 0
```

This hierarchical method contrasts the *walktrap*. The clearest difference is that this algorithm places the entire network in a singular community. Because of this, there is no separation of communities, thus leading to a modularity simulated sample.

5.4 - Fast-Greedy

```
set.seed(100)
fg.100 <- fastgreedy.community(graph.100)
plot(fg.100, graph.100)
```



```
sizes(fg.100)
```

```
## Community sizes
##  1  2
## 55 45
```

```
modularity(graph.100, membership(fg.100))
```

```
## [1] 0.01572409
```

This method produces a "more suitable" network compared to the first two algorithms. It makes sense that the modularity would increase for this method because it is based on local modularity optimization. We have approximately even tallies of membership in each group (55 & 45 in groups 1 & 2 respectively). The factors contribute to a fairly high modularity score in the context of our data exploration.

Let us now look at how these communities may have been clustered based means of the numerical variables attached to each node.

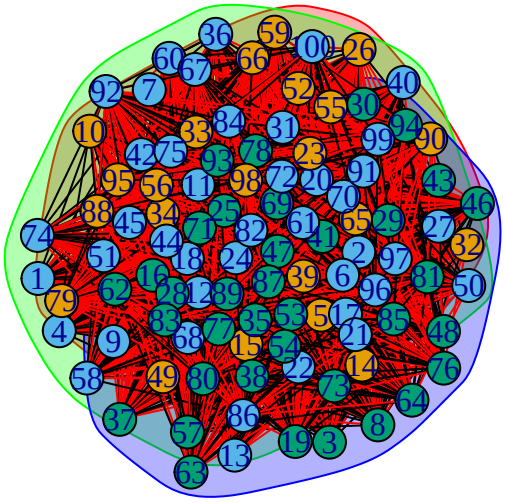
```
wine.100$cluster <- as.factor(membership(fg.100))
 wine.100 %>%
  group_by(cluster) %>%
  summarise(mean_price = mean(price),
            mean_year = mean(year),
            mean_points = mean(points))
```

```
## # A tibble: 2 x 4
##   cluster mean_price mean_year mean_points
##   <fct>        <dbl>     <dbl>       <dbl>
## 1 1             36.9     2010.        88.5
## 2 2             34.1     2010.        88.7
```

Based on this output, it appears that price is a key differentiator between the two clusters. The difference in mean price per bottle is approximately $2.78

5.5 - Spinglass

```
set.seed(100)
sg.100 <- spinglass.community(graph.100)
plot(sg.100, graph.100)
```



```
sizes(sg.100)
```

```
## Community sizes
##  1  2  3
## 22 42 36
```

```
modularity(graph.100, membership(sg.100))
```

```
## [1] 0.01826896
```

This algorithm further builds on the relative success seen in the previous one. Likely due to its significantly longer run-time, this method outperformed all other algorithms in this analysis on the basis of modularity. The three clusters each have a decent number of nodes per community as well. We can also look at the summary table (similar to the one in the previous section) to look at the possible contributing factors to cluster designation.

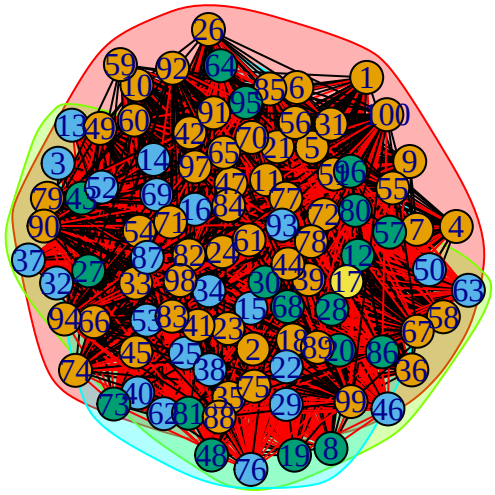
```
wine.100$cluster <- as.factor(membership(sg.100))
 wine.100 %>%
  group_by(cluster) %>%
  summarise(mean_price = mean(price),
            mean_year = mean(year),
            mean_points = mean(points))
```

```
## # A tibble: 3 x 4
##   cluster mean_price mean_year mean_points
##   <fct>        <dbl>     <dbl>       <dbl>
## 1 1             35.0     2010.        88.6
## 2 2             35.4     2011.        89.1
## 3 3             36.4     2010         88.0
```

This output illustrates that clusters relied primarily on being separated dependent on *price* and *points*. Cluster 2 differs significantly from clusters 1 & 3 with regards to mean price, and the mean points & mean year values are subtly different for each grouping.

5.6 - Leading Eigenvector

```
set.seed(100)
le.100 <- leading.eigenvector.community(graph.100)
plot(le.100, graph.100)
```



```
sizes(le.100)
```

```
## Community sizes
##  1  2  3  4
## 58 23 18  1
```

```
modularity(graph.100, membership(le.100))
```

```
## [1] 0.004564837
```

This algorithm performs better than *edge-betweenness* & *walktrap*, though not as well as the *fast-greedy* or *spinglass*. The modularity score is not very high, but this likely stems from the contrast of group sizes within the network (i.e. having one group with 58 nodes and another that is a node by itself). This method is supposed to be a "happy-medium" of sorts as it operates efficiently (based on run time) due to its eigenvector based mechanics, though we see that in this sample situation it does not optimize modularity at the same level compared to others. We will now look at the summary statistics similar to previous sections.

```
wine.100$cluster <- as.factor(membership(le.100))
 wine.100 %>%
  group_by(cluster) %>%
  summarise(mean_price = mean(price),
            mean_year = mean(year),
            mean_points = mean(points))
```

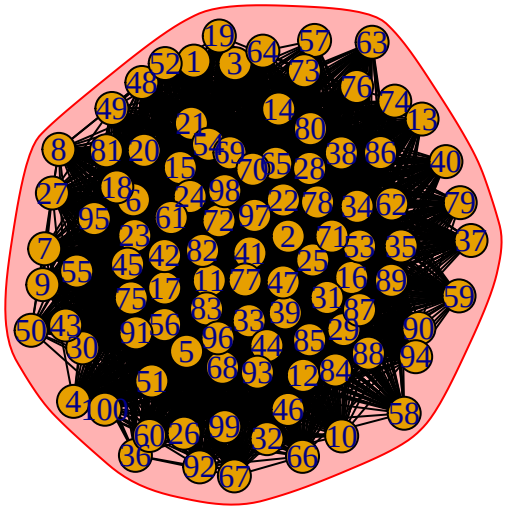
```
## # A tibble: 4 x 4
##   cluster mean_price mean_year mean_points
##   <fct>        <dbl>     <dbl>       <dbl>
## 1 1             34.1     2011.        88.7
## 2 2             35.1     2010.        87.8
## 3 3             42.6     2010.        89.5
## 4 4             14       2007         85
```

It is difficult to draw too many conclusions from this output due to the different sizes of each group, but it appears that there is a fairly even mix of *prices, years*, and *points* for our wines. This means that the networks could be drawn more significantly based on the categorical variables (*variety & country**).

5.7 - Label Propagation

```
set.seed(100)

lp.100 <- label.propagation.community(graph.100)
plot(lp.100, graph.100)
```



```
sizes(lp.100)
```

```
## Community sizes
##   1
## 100
```

```
modularity(graph.100, membership(lp.100))
```

```
## [1] 0
```

Similar to the *edge-betweenness* algorithm, the *label propagation* method placed all of the nodes into a single community. This will make it difficult for comparative analysis. This is also why there will be a modularity score of 0.

Additionally, for each algorithm, we were able to find the summary statistics for the numerical variables in the group.

The above output gives us a starting sense of how each generated cluster differ, giving us an initial comparison between the clusters within each algorithm.

5.8 - Direct Algorithm Comparison

Next, we will compare the clusters between three algorithms. Because of R's space constraint, rather than using a visualization for each clustering algorithm, we assessed each algorithm's performance via the *leading eigenvector*, *fast-greedy*, and *spinglass* algorithms. Below we have generated tables that show a breakdown of the clusters of observations by each algorithm, or, in other words, a tally of how many wines were categorized into group 1 of the two algorithms. It is important to note that we do not know how each algorithm partitioned each cluster; however, the tables displayed illustrate the exact breakdowns of each cluster and show how the sizes of each algorithm's communities overlap.

First, we will compare the *leading eigenvector* algorithm clusters with those of the *fast-greedy* algorithm.

```
ec_fg
```

```
##       FG.100
```

```
## LE.100  1  2
##      1 28 30
##      2 18  5
##      3  9  9
##      4  0  1
```

The table shows that the leading eigenvector algorithm formed four communities whereas the fast-greedy algorithm produced a total of two clusters. The first cluster the fast greedy method generated was split into four separate groups, but there were 28 items that overlapped for both of their first clusters. Eighteen of the first fast-greedy cluster was put into the eigenvector's second cluster while the nine left over were placed into the leading eigenvector's third cluster and none were placed in the fourth cluster. The second fast-greedy community was partitioned in a way that 30 overlapped with the leading eigenvector's first cluster, 5 in the second cluster, nine in the third cluster, and 1 in the fourth and last cluster.

Next, we will compare the *leading eigenvector* algorithm communities with those of the *spinglass* algorithm.

```
ec_spin
```

```
##       SPIN.100
## LE.100  1  2  3
##      1 16 31 11
##      2  5  4 14
##      3  1  6 11
##      4  0  1  0
```

The table displays the leading eigenvector algorithm still formed four communities whereas the spinglass algorithm produced three clusters. The same logic follows with the previous table in that this display shows how the two algorithms overlap and differ among their communities. Their first communities had 16 that overlapped, their second both had 4 that overlapped, and their third community had 11 that overlapped. It seems that the leading eigenvector algorithm had more overlap in general with spinglass than it had with the fast-greedy clustering algorithm. It is interesting to note that the the leading eigenvector's first cluster was split into three clusters by the spinglass method with the greatest value (31) being placed in spinglass's cluster number two.

Lastly, we will compare the *spinglass* algorithm clusters with those of the *fast-greedy* method.

```
spin_fg
```

```
##         FG.100
## SPIN.100  1  2
##        1 16  6
##        2  9 33
##        3 30  6
```

The table shows the spinglass algorithm still formed three communities whereas the fast-greedy algorithm formed two clusters. The two algorithms had 16 that overlapped in both of their first clusters and a value of 33 that overlapped in their second clusters. This suggests that perhaps spinglass and fast-greedy formed rather similar clusters as their overlap was larger than that of the previous tables. Another interesting point is at the intersection where fast-greedy's first cluster meets with spinglass's third cluster of which 30 nodes overlap.

Having visualized these, we can conclude that while the data may not be conclusive and suggestive of any substantial results because we used the data to explore the techniques of network analysis, the implications of our final project can carry us to apply our network analysis to wine recommender systems and other recommender systems in general.

Section 6 - Limitations & Other Key Findings

One of the more interesting findings of this analysis concerned the amount of time that it takes to generate these types of networks. Some of the more thorough approaches (in particular the *edge-betweenness* and *spinglass* algorithms) took hours to run when the sample size we used began approaching 1000. However, some of the less intensive models (like *walktrap* and *fast-greedy*) ran in a matter of seconds. Thus, we decided to limit the sample size to 100 for each algorithm.

If the algorithms were performing the same, the distribution of observations in each cluster group would not vary greatly across algorithms. From the generated tally tables, we can see that this is not the case, as the numbers of groups vary widely as do the numbers of observations that fall into each groups.

Section 7 - Conclusion

Through the final project, we were able to use one graph object and six algorithms to group seemingly random observations of wine reviews into clusters that made clearer the differences and similarities of the wine reviews. In doing so, the results suggested the greatly varying.

Though there were some pitfalls and roadblocks along the way, we believe that this project was successful for a variety of reasons. The first reason is that we were able to explore a new field of statistics that is not covered in the curriculum.

Moving forward, we will be able to use and better assess these clustering algorithms along with other community detection tools in exploratory data analysis. Better understanding the complex relationships underlying.

Works Cited

Belle, G. V. (2008). Statistical rules of thumb. Hoboken (N.J.): Wiley.

Maechler, M. (2018, April 9). Package 'cluster'. Retrieved December 8, 2018, from https://cran.r-project.org/web/packages/cluster/cluster.pdf. *Only first author listed*

Tamas. (2012, February 28). "What are the differences between community detection algorithms in igraph?" https://stackoverflow.com/questions/9471906/what-are-the-differences-between-co munity-detection-algorithms-in-igraph

Thout, Z. (2017). Wine Reviews. Retrieved November 29, 2018, from https://www.kaggle.com/zynicide/wine-reviews.