Data Science for Public Policy

Alena Stern - Georgetown University

PPOL 670 | Assignment 08
Cluster Analysis, Text Analysis, Adv. Data Viz

Due Date: Tuesday, April 26th at 11:59 PM.

Deliverable:

1. A .Rmd file with your R code.
2. The resulting project .html file.
3. The URL to your private GitHub repository from assignment 08.

Grading Rubric

- [0.5 point] Create a private, well-managed GitHub repository, including an appropriate `.gitignore`, and an informative `README.md` file. You should add at least one commit for each question. Points will be reduced for infrequent commits or unclear commit messages.
- [0.5 point] Write a clean and well-composed `.Rmd file`, including separate named code chunks for each task. The resulting `.html` file should show code and results, but hide unnecessary warnings and messages.
- [1 point] Exercise 01
- [1 point] Exercise 02
- [1.5 points] Exercise 03
- [1.5 points] Exercise 04

Points: 6 points

Learning and data science are both collaborative practices. We encourage you to discuss class topics and homework topics with each other. However, the work you submit must be your own. A student should never see another student's code or receive explicit coding instructions for a homework problem. Please attend office hours or contact one of the instructors if you need help or clarification.

Plagiarism on homework or projects will be dealt with to the full extent allowed by Georgetown policy (see http://honorcouncil.georgetown.edu).

Setup

Create a new folder with a new R project (`.Rproj`) and R Markdown file (`.Rmd`). Then create a new **private GitHub repository**. Add awunderground and ncstabile17 (section 01) or alenastern and joshrrosen (section 02) and your partner (if working with a partner) to your repository.

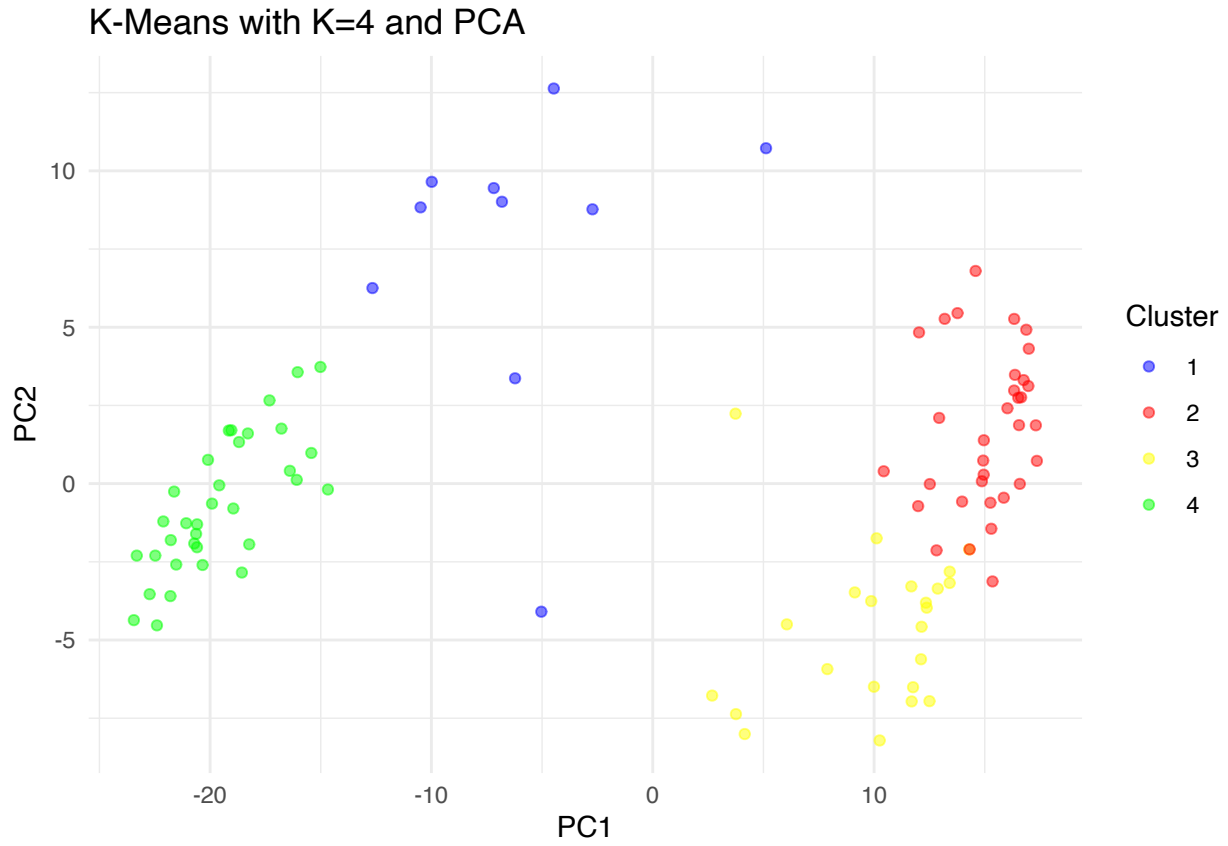
Exercise 01 (1 point)

This exercise uses data on U.S. Senate Votes from Session 103 (1993) to Session 114 (2016) from Brad Robinson via data.world. In the `votes_time_series.csv` data on Canvas, each row represents a senator-Session pair.

a. Read in the `votes_time_series.csv` file and replace all missing values with 0 (the value signifying an absent vote). Filter the dataframe to **only include the votes for Session 103** and save the result in a new dataframe called `votes_103`.
b. Create a dataframe called `votes_numeric` that contains only the votes columns (they all start with "v") and run Principal Components Analysis on `votes_numeric`, saving the output to `votes_pca`.
c. How much variance does the first principal component explain? How much cumulative variance is explained by the first 5 principal components?
d. Extract the principal components from `votes_pca` and column bind the first two principal components with the `votes_103` data to create a new dataframe called `votes_pcs`. Use `votes_pcs` to create two scatterplots visualizing the data with PC1 on the x-axis and PC2 on the y-axis: The first plot should use the aesthetic mapping `color = party` and the second plot should use `color = region`, where region is equal to one of the four Census regions (West, Midwest, Northeast, South). You can append region to `votes_pcs` by using the `states_regions.csv` file on Canvas. Make sure to include good titles and labels in your graph. Include the graphs side by side in your markdown document. **Hint:** `library(patchwork)` will be useful for displaying the graphs side by side.

Exercise 02 (1 point)

a. Next, we will conduct cluster analysis on `votes_numeric`. Use `library(factoextra)` to calculate the within sum of squares, silhouette distance, and gap statistic. Make sure to `set.seed(20220412)` first.
b. Create a function that takes a number of clusters `k` as one argument and the `votes_pcs` dataframe as another argument and performs kmeans clustering where `k` is set as the number of clusters for kmeans and the votes columns are used as the data. The function should then produce a scatterplot with PC1 on the x-axis and PC2 on the y-axis and the color encoding the cluster assignment. The graph should have good labels and a title that includes the number of clusters being used (see the example below). The function should return a ggplot object.
c. Run your function with each of the different potential optimal numbers of clusters suggested by the different metrics calculated in part a and display the graphs side by side.



Exercise 03 (1.5 points)

a. Read in the executive orders dataset used in the text analysis lab and filter to exclude rows where the `text` column is missing. Use the `unnest_tokens()` function to create bigrams (two-word tokens) from the text. In the lab we used this function to create single-word tokens. Read the documentation to learn how to modify that function call to create bigrams.The n-grams chapter of Text Mining with R may also be helpful.

b. Split the bigram column into two separate columns (word1 and word2) with each word of the bigram (**Hint:** the `separate()` function may be helpful). Filter the dataframe to rows that do not have any stop words for either word of the bigram. Count the number of appearances of each bigram (unique combinations of word1 and word2) and filter to rows with more than 150 appearances. Assign the result to `bigram_150` and use the code below to visualize the data.

```
# plot the bigrams that exist more than 150 times
bigram_graph <- bigram_150 %>%
  graph_from_data_frame()

# plot the relationships (you may want to make the plot window bigger)
set.seed(2017)
ggraph(bigram_graph, layout = "fr") +
  geom_edge_link() +
  geom_node_point() +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1)
```

c. Calculate the tf-idf for each president and bigram. To do this, you will first want to count the number of appearances of each bigram-president pair (remembering to filter out stop words). After separating the bigrams into separate words for filtering out stop words, you'll want to merge the individual word columns back into a single bigram column before calculating tf-idf (**Hint:** check out the `paste()` function).
d. Plot the bigrams with the 15 largest tf-idf values for each president.

Exercise 04 (1.5 points)

This exercise uses a dataset of the descriptions and voting outcomes of bills from the 114th Senate from legiscan.com. Your goal will be to create a supervised machine learning model using `library(tidymodels)` to classify whether each bill passed (`passed` = 1) or did not pass (`passed` = 0).

a. Read `senate_bills_114.csv` into R and save as an object called `bills`. You will need to modify passed to be a factor with "1" as the first level and "0" as the second as shown below. We do this because tidymodels by default treats the first factor level as the positive case. How many bills in the dataset passed?

```
bills <- read_csv("senate_bills_114.csv") %>%
  mutate(passed = factor(passed, labels = c("1", "0"), levels = c("1", "0")))
```

b. Split `bills` into a training and testing dataset setting `strata = "passed"` and `prop = .75`. Note that you will not need the `bill_number` column and can drop it from the dataframe before creating the split. Don't forget to `set.seed(20220414) first`.
c. We will perform our text pre-processing steps using `library(textrecipes)`, a package built for tidymodels with additional recipe steps for text analysis. Read the textrecipes website and function reference to identify the right steps to create a recipe that performs the following steps: 1) tokenizes the `description` column, 2) removes stopwords (note, you'll have to `install.packages("stopwords")` first to use `step_stopwords`), 3) stems the words, 4) filters to the 200 most common tokens, 5) performs tf-idf.
d. `prep()` and `bake()` your recipe on the training data to take a look at the words used for tf-idf (they will be the end of the column names). Are there any words that seem like domain-specific stopwords? Update `step_stopwords()` in your recipe to remove these stopwords using the `custom_stopword_source` argument and re-create your recipe.
e. Create a model object using logistic regression (`logistic_reg()`) and the `"glm"` package for the engine. Create a workflow with your recipe and model specification and then pass that workflow to `fit()` to train the model on the training data. (Note that we aren't asking you to do cross-validation or model tuning here)
f. Use the fitted model to `predict()` the class membership **and** class probability using the test data. Calculate the accuracy, precision, recall, and the roc curve for your model. Using these metric results, how did your model do? What are at least three things you might add to your approach to improve your model results?

Stretch (5 points)

Part 1

a. Create a function that takes as an argument a session number and the `votes` dataframe and does the following tasks:

1. Filters the `votes` data to the given session and assigns the results to an object
2. Runs PCA on the votes columns of the filtered data and binds PC1 and PC2 to the dataframe created in step 1
3. Conducts kmeans cluster analysis on the votes columns with 4 clusters
4. Appends the cluster assignments to the dataframe created in step 2 and returns the updated dataframe

b. Use `library(purrr)` to map over a vector of the unique session numbers present in the `votes` data applying the function created in part with each session. The results should be saved in a single dataframe. Make sure to `set.seed(20220413)` before mapping over the vector of session numbers.

Part 2

`library(gganimate)` extends the grammar of graphics as implemented by `library(ggplot2)` to include the description of animation. See the `gganimate` website for more detail and some useful examples that can serve as models for this task.

a. Use gganimate to create an animated scatter plot that shows the voting behavior for each senator in Sessions 103-116. Your plot should have PC1 and PC2 on the x and y axis and encode the state using color. Save the output of your code to an object called `clus_plot`.
b. Use the lines below to animate your graph and save the animation as a gif. Note that to use `gifski_renderer()` you will need to install and load `library(gifski)`.
c. Embed clust_gif.gif in your html output.

```
animate(clus_plot, duration = 10, fps = 20, width = 400, height = 400, renderer = gifski_renderer())
anim_save("clust_gif.gif")
```

Part 3

In this exercise, we will use agglomerative hierarchical clustering on the votes dataset.

a. Using the votes data from exercises 1 and 2, filter the votes data to Sesssion 113 and create a dataframe with just the votes columns.
b. Using `fviz_nbclust()`, use agglomerative hierarchical clustering to identify the optimal number of clusters for the data based on within sum of squares and silhouette distance. You will need to use the `factoextra::hcut` package for hierarchical clustering. Note that `fviz_nbclust()` enables you to set values for the parameters of of the function used for clustering. Here is an example function call for within sum of squares:

```
set.seed(20220414)
fviz_nbclust(votes_numeric,
             FUN = hcut,
             k.max = 10,
             method = "wss",
             hc_func = "hclust",
             hc_method = "ward.D2",
             hc_metric = "euclidean")
```

You will need to paste this code into your markdown document (including the set.seed() call!) and create a second call modifing the code to calculate the silhouette distance.

c. Use `stats::hclust()` to plot a dendogram that shows the hierarchical relationship of the observations as they're formed into fewer and fewer clusters. See this tutorial for reference. Make sure to use the `ward.D2` agglomeration method in `hclust()`. Assign the output of your `hclust()` call to an object called `hclust_euc` and use it to create a dendogram. Use `rect.hclust()` to draw rectangles around each cluster for the optimal number of clusters identified in part b.

d. Use `cutree()` on the `hclust_euc` to extract the cluster assignments, setting the parameter `k` equal to the optimal number of clusters identified earlier. Append this column to the Session 113 dataframe you created in part a.