Lab 4 - Machine Learning for Social Science

To be handed in no later than October 21, 10:00. The submission should include code, relevant output, as well as answers to questions. We recommend the use of RMarkdown to create the report.

Part 1: Topic modeling

In this lab, we will use a data set containing a random sample of public Facebook posts by members of the U.S. Congress from 2017.[1] Our broad objective in this first part of the lab is to explore what topics were discussed, and possible variation by party membership.

1. Begin by importing `fb-congress-data3.csv`. Report basic information about the data set; how many rows and column it has, as well as the name of the variables.
2. As you may have noticed from your inspection in #1, this data set has yet to be pre-processed (it contains punctuation, etc.). Hence, that is what you shall do now. More specifically, perform the following steps:
   i. Use `quanteda`'s `corpus()` function to create a corpus of your data set. Hint: For the argument `x` select your data set, for the argument `text` select the column name which stores the text, for the argument `docid_field` select the id variable, and finally, add the names of remaining variables to the `meta` argument (in a list).
   ii. Tokenize your corpus using the `tokens()` function. This splits each document into a vector of so-called tokens. Make the following specifications (which will remove punctuation, numbers, non-alpha-numeric symbols, and urls):
      - `remove_punct = TRUE`
      - `remove_numbers = TRUE`
      - `remove_symbols = TRUE`
      - `remove_url = TRUE`
      - `padding = FALSE`
   iii. Exclude english stopwords using the `tokens_remove()` function. Setting `x` to the output from the previous step, setting the second argument to `stopwords("en")`, and setting `padding=FALSE`.
   iv. To get a feel of how your data looks like now, print the first 3 texts by simple subsetting of the output from iii.
   v. As mentioned in the lecture, topic models expect the data to be in a *document-term-matrix* form. Transform your tokens into a document-term-matrix using the `quanteda`'s function `dfm()`.
   vi. As a last pre-processing step, we want to exclude (a) words which are very infrequent (below 5). and (b) documents which have very few words (below 10). When you have done a–b, report the dimensionality of your resulting document-term-matrix. Hint: To do trim infrequent words, use `quanteda`'s function `dfm_trim()`. To exclude documents with too few words, you may use the following code (where `dtm` is the object in which you have stored your document-term-matrix):

[1] Obtained from https://lse-my459.github.io/

```
rowsums <- rowSums(dtm)
keep_ids <- which(rowsums>=10)
dtm <- dtm[keep_ids,]
```

3. Now we are ready to do some topic modeling! To do so, we will use the `topicmodels` package, and the function `LDA()`. Set `x` to your document-term-matrix and specify `method="Gibbs"` (note: Gibbs is the name of a particular estimation procedure; see the Appendix of the lecture for more details). Set the number of iterations to 1000, and specify a seed number to ensure replicability (hint: to specify iterations and seed number, use the `control` argument). Finally, set the number of topics, `K=50`.[2] With these settings specified, start the estimation. This could take a minute or two.

4. Once the estimation is finished, use the `get_terms()` function to extract the 15 words with the highest probability in each topic. In a real research setting, we would carefully examine each of the topics. Here, I only ask you to briefly skim them, and then focus on 5 that (*i*) you think are interesting, (*ii*) has a clear theme, and (*iii*) are clearly distinct from the other topics. Provide a label to each of those based on the top 15 words. Complementing your label, please also provide a bar chart displaying on the y-axis the top 15 words, and on the x-axis their topic probabilities. Hint: you can retrieve each topic's distribution over words using `topicmodels`'s function `posterior`.[3] Lastly, please also report a general assessment—based on your skim—about the general quality of the topics; do most of them appear clearly themed and distinct, or are there a lot of "junk" topics?

5. Out of the 5 topics that you labeled, select *two* which you think are particularly interesting. For these two, identify the three documents which have the highest proportion assigned of this topic (hint 1: use `topicmodels`'s `posterior()` to extract documents' distribution over topics | hint 2: to identify the document ids which correspond to each row of what you extract from `posterior()`, you can use `ldaobject@documents`. See help file for more details.), and do a qualitative inspection ($= 2 \times 3$ documents to read). Does your readings corroborate your labels? Are they about what you expected?

6. Now, estimate a topic model—as in #3—but with `K=3` instead. Extract the top 15 words from each topic, (try to) label each, and then make an assessment of the overall quality of them. To further explore the quality of this topic model, reconsider the documents you read in #5: extract the distribution over topics for these documents (from your new `K=3` model). How well does this topic model capture the theme of these documents? Based on your analysis, which of the two `K`'s do you prefer? Motivate.

7. Continuing with the topic model you concluded the most appropriate, perform the following sets of analyses:

    i. Compute the prevalence of each topic, across all documents. Report which is the most prevalent topic, overall, and then report—in the form of a single plot; e.g., a bar chart—the prevalence of the topics you labeled.

    ii. Compare the prevalence on your labeled topics between *democrats* and *republicans*. You can for example fit a fractional regression model using `glm(family="quasibinomial")`[4] or using t-tests of difference in means. Interpret.

8. **BONUS** (not obligatory; suggestion; do after you have completed the rest of the lab). As a bonus exercise—to expose you to the traditional computer sciency way of selecting the number of topics, K—you shall consider a data-driven approach, relying on the measure of *hold-out likelihood* (or, *perplexity* as its also called). To do so, do the following:

    i. Split your document term matrix into two (a training and test set); 80/20 division.

[2] Note: As we discussed in the lecture, in real research settings, where we usually have a clear research question, we would likely explore a range of K and select K based on how well it enables us to address the research question.

[3] It provides both (a) each document's distribution over topics, and (b) each topic's distribution over words.

[4] If you go this route, note that you then also need to compute robust standard errors, which you can do using the `sandwich` package: `coeftest(myglm, vcov. = vcovHC(myglm, type = 'HC1'))`

ii. Write a loop which in each iteration (a) estimates a topic model using a particular K, and then (b) computes (and stores) its perplexity using the `topicmodels` function `perplexity()`, which takes as input the model object and the test document-term-matrix (note: the document-term-matrix needs to be transformed into a particular format: use `...` for this).

iii. Consider the following range of `K`: ${3, 10, 25, 50, 75, 100, 200}$, and run the loop. This may take a few minutes. Once the loop has finished, plot your results (x-axis: `K`, y-axis: `perplexity`). Interpret. Based on this, what is a reasonable `K`?

Part 2: Word embeddings

In this second part of the lab, we will continue with the data *U.S. Congress–Facebook posts* data set. However, now with a different focus: a focus on the word-level, using word embeddings instead of topic models.

1. Because word embeddings are not negatively affected by *stop words* or other highly frequent terms, your first task is to re-import the `fb-congress-data3.csv` file, and re-process the data; performing step *i–ii* in task #2, but skipping #3. Here, we also *do not* want to transform our documents into a document-term matrix. Instead, after having tokenized and cleaned the documents, paste each back into a *single string* per document. Hint: for this, you could for example write: `sapply(mytokens,function(x)paste(x,collapse = " "))`. As a last pre-processing step, transform all your text into lowercase (hint: you can use the function `tolower()` for this).

2. Now we are set to fit word embeddings! To begin, let us fit one word embedding model to all documents—not separating posts by democrats and republicans. Use `word2vec`'s `word2vec()` function to fit a *cbow* model (`type="cbow"`) using 15 negative samples per real context/observation (`negative=15`), and setting `dim=50`, the number of dimensions of the word vetors/embeddings. This will take a minute or two.

3. When the estimation in #2 is finished, identify the 10 nearest terms to 3 focal words of your choice/interest. Make sure to select words which occur frequently in your data. Hint: to retrieve the closest words in embedding/word vector space, you may use the following code: `predict(w2v,c("word2","word2","word3"),type="nearest",top_n = 10)`, where `wv2` is the object storing the *fitted model* of the `word2vec` function. Does the results you find makes sense? Why/why not?

4. What initially made people so excited about word embeddings was their surprising ability to solve seemingly complex analogy tasks. Your task now is to attempt to replicate one such classical analogy result, first with the embedding vectors that you have already estimated, and second using a pre-trained embedding model. To do so, please perform the following steps:

   i. Extract the whole embedding matrix: `embedding <- as.matrix(w2v)`.

   ii. Identify the rows in the embedding matrix which correspond to `king`, `man`, `woman`, and create a new R object `kingtowoman` which is equal to the vector for king, minus the vector for man, plus the vector for woman. Hint: to extract the row corresponding to a particular word (e.g., "king"), you may use `w2v[rownames(w2v)=="king",]`.

   iii. Use `word2vec`'s function `word2vec_similarity()` to identify the 20 most similar words to `kingtowoman`. Do you find "queen" in the top 20? Why do you think you get the result you do?

   iv. Next, we will consider a *pre-trained embedding model* (trained on all Wikipedia articles that existed in 2014 and about 5 million news articles). The embedding vectors from this model are stored in the file "glove6B200d.rds".[5] Note: this file is large; more than 300MB. Use `readRDS()` to import it, and stored it in an `R` object called `pretrained`. Each row stores the embedding vector for a particular word. With this info in mind, report how many embedding dimensions were used for this model, and how many words we have embedding vectors for.

[5] Original source for this file: https://nlp.stanford.edu/projects/glove/

v. Repeat steps ii–iii for `pretrained`. Does "queen" appear in the top 20 here? What do you think explains this difference/similarity to the self-trained result?

vi. Given the result in (v), what do you expect, if you were to construct a measure of *occupational gender bias* along the lines of Garg et al. (2018), that is by comparing the distance between different occupations and gendered words, for example: $\overrightarrow{occupationalbias} = \text{dist}(\overrightarrow{statistician}, \overrightarrow{man}) - \text{dist}(\overrightarrow{statistician}, \overrightarrow{woman})$, would this score be "more correct" than the one you would obtain from the same calculation on your facebook/congress model? Why/why not?

5. Now we shall make a comparison between democrats and republicans. Split the data from step #1 into two based on party affiliation. Then, repeat 2–3, but now separately for republicans and democrats. For #3, select words which you expect might be used differently between the two political camps (but still are frequently used by both; for example "abortion", "obamacare"). Do you find any differences? Do they align with your expectations?

6. **BONUS** (not obligatory). Continuing with the comparison between democrats and republicans, your next task is to construct a *sentiment dimension* for each party, and then—by projecting the rest of the words onto this dimension—explore which words that have the strongest negative association in each party. To do so, please follow these steps:

   i. Import the two text files `positive.txt` and `negative.txt`. They contain a large number of positively and negatively laiden words, respectively.

   ii. Separately for each party:

      a. Identify the rows (in the corresponding party embedding matrix) which correspond to the words in the `positive` and `negative` document, extract their vectors into two separate matrices (positive, negative) and calculate the column averages.

      b. Compute the difference between the two average vectors from a (negative – positive). This is the party-specific "negative sentiment" dimension.

      c. Use `word2vec`'s function `word2vec_similarity()` to calculate the similarity to the sentiment dimension for all words except those included in the negative file.

   iii. Using the results from ii, identify the 50 words with the strongest negative association (most similarity with the negative sentiment dimension) for both republicans and democrats. Report what you find. Are there substantive differences in the composition of words? Additionally, you shall pick a set of words which you hypothesize may be used in a more/less positive light between the parties; are they? Note: absolute differences cannot be compared across models; instead you shall report relative differences (i.e., their ranking).

7. **BONUS** (not obligatory). How can we know if our results from #6 are statistically significant / robust? The non-parametric boostrap! Repeat step 6 (the latter part, where you selected a specific set of words to compare) but this time across $\times 20$ bootstrapped samples. Report the upper and lower bound of comparisons you made in #6.