Assessment: Natural Language Processing with Python

Instructions

To earn a micro-badge for this workshop, write code for/answer the prompts on the next page. You should do your work in a code environment (like Jupyter); include all code, code outputs, and short answers directly in your notebook. Submit this notebook to GradPathways by exporting it to an `html` file (in Jupyter Notebooks, you can do this by going to `File > Save Page As` in your browser).

Links

- GradPathways Badge
- Event page
- Workshop reader

Environment and File Setup

For this assessment, you will extract linguistic information about a corpus of texts and use this to model the texts. We intend this both to reinforce the materials covered in the workshop sessions and to serve as an occasion with which you can practice NLP methods for your own research. To wit: you are invited to use your *own* corpus of text files to complete the assessment; by the time you've finished, you'll have built a foundation for a research project with NLP.

If you don't want to use your own corpus, or if you don't have one ready to hand, we've also provided you one. Under `data/sherlock` you'll find 56 Sherlock Holmes short stories. You can find a corresponding file manifest at `data/manifest.csv`, which you should use as a reference when doing your work. If you choose to use your own corpus, we suggest using somewhere between 50-100 documents. But note: 100 novels is far more material than 100 short stories. Try to aim for a corpus that has **at least 150,000 total words.**

This assessment does not require you to generate and store new files, but you are welcome to do so as you see fit.

While we ask that you do this work in a code notebook, you may use a local environment on your own computer or Google Colab. Regardless of which environment you use, you will need to make sure that you have all required packages installed. The requirements file for these packages is under `data/requirements.txt`.

The directory structure for this assessment is:

```
requirements.txt                A list of required packages
text_mining_assessment.pdf      These instructions
data/                           The data directory
|-- manifest.csv                A file manifest
`-- sherlock/                   Plaintext files of 56 Sherlock Holmes stories
```

Rubric

Readers at GradPathways will be looking for a few things in this assessment:

1. Working code: were you able to successfully implement code for each prompt?
2. Understanding the code: can you explain what your code does and why you implemented it?
3. Supported examples and materials: have you used graphs and other results to produce evidence for your findings?
4. Critical reflection: do your short answers provide context (conceptual, domain-specific, etc.) for your findings and observations? Can you use your results to reason about your corpus, or even provide preliminary hypotheses?

Prompts

1. Processing
    a. Load the following data into your environment, assigning each to their respective variables:
        - `nlp`: `spaCy`'s `en_core_web_md` model
        - `manifest`: file manifest for the corpus files
        - `paths`: a list of paths to all the corpus files; be sure to sort these!
    b. Write a function to load corpus files from `paths`. This function should `yield` opened documents. Use it in concert with the model's `.pipe()` method to load and process the corpus. Assign the output to a variable named `corpus`
    c. As you wait for the documents to process, explain in a sentence or two why someone might want to use a generator to handle text data
    d. By default, `spaCy` assigns word embedding vectors to both individual tokens and to the document as a whole. You can access these with the `.vector` attribute. Use a list comprehension to get the **document** vectors from each file in the corpus and store the output in a `vecs` variable. Wrap this list in a `numpy` array
2. Clustering documents with word embeddings
    a. With the embedding vectors extracted, you can cluster and visualize the corpus. Use the `AgglomerativeClustering` object from `scikit-learn` to do the clustering on the vectors. Then, run a dimensionality reduction with `TSNE` (also in `scikit-learn`) to create two-dimensional (XY) representations of the vectors. Use the following values for `TSNE`:
        - `init`: pca
        - `learning_rate`: auto
        - `angle`: any float under 0.3
        - **Hint:** if you're unsure about the syntax here, refer to the final section in chapter three of the workshop reader
    b. Convert your XY data into a `pandas` dataframe called `vis_data`. Assign two new columns to this dataframe: 1) `title`: title of each story (available in `manifest`); 2) `label`: output clusters from `AgglomerativeClustering` (accessible via the `.labels_` attribute). Use `altair` to make a scatterplot of this data. The plot's `color` argument should take in the `label` column (use `label:N` to get the coloring right); `tooltip` should take in `name`. Don't forget to set the plot to its interactive mode!
    c. `AgglomerativeClustering` defaults to two clusters. Based on a visual inspection of the plot you've made, do you think this number of clusters is adequate for your corpus? Why not or why not?
    d. Re-cluster your data a few times with a different number of clusters. Find what you think is the optimal number of clusters for the corpus. Explain your reasoning for doing so, using supporting visualizations as you see fit
        - **Hint:** If you're having trouble dividing up your documents into groups in the visualization, try adjusting the parameters of `TSNE` to better reflect what the `AgglomerativeClustering` object analyzes. A good place to start is `angle`
        - **Note:** Clustering almost always involves some element of interpretation, but, in your own work, you can also use empirical measures to help you set an appropriate number of clusters. Silhouette scoring is a common strategy for doing so
    e. Once you've picked your optimal number of clusters, assign the cluster labels to a `CLUSTER` column in `manifest`. Use a `groupby` to count the number of documents in each cluster
3. Unique words
    a. It's now to time to explore the actual words in your corpus. First, you'll look at intersecting words across the clusters you've created. To do so, create a list of sets, where each set corresponds to the unique tokens in a corpus text. Be sure to use the `.text` attribute when compiling these sets from `spaCy` documents. Assign this list to a new column in `manifest` called `TYPES`
    b. In a for loop, step through each unique cluster in `manifest`. Use

`.groupby('CLUSTER').get_group(<CLUSTER_NUM>)['TYPES']` to extract all the sets for a given cluster. Perform a **set intersection** on the list, add the result to a dictionary called `doc_intersections` (use the cluster number as a key), and print the following to screen:
   - Cluster number
   - Number of elements in a set
   - **Hint:** not sure how to do a set intersection? Take a look at this link

c. Get the intersection of `doc_intersections` and store it in a variable named `corpus_intersections` (the result should be relatively short). This is the overlap between all unique words in all clusters

d. Return to your dictionary of sets and build one more for loop to step through each one. Within the for loop, use `<CURRENT_SET>.difference(corpus_intersections)` to get the difference between the current set of words and the intersection across the corpus. Print the result of `.difference()` to screen along with the cluster number and inspect the contents. What do you see? Can you discern any patterns among these words that might help you understand why they've been grouped together?

4. Linguistic features

a. In 3a, you created sets from the corpus texts using `spaCy`'s `.text` attribute. But as you know from our series, `spaCy` provides several other document and token annotations, ranging from entities to part-of-speech tags and syntatic dependencies. Select three such attributes and for each one, write a function that will gather information about this attribute from the corpus texts
   - **Hint:** You're probably going to be counting attributes. If so, it's best to normalize those counts, usually by dividing them against the document as a whole (refer to the advanced feature engineering section of the second chapter in the corpus reader)

b. Before running these functions on the corpus text, explain your reasoning for selecting each attribute in a sentence or two. Make a prediction: do you think your attribute will help you discern differences between your clusters?

c. Run the functions and assign the result of each to columns in `manifest`. Then group `manifest` by `CLUSTER` and make a histogram of each attribute you created. Based on what you see, do you think these attributes successfully partition the corpus along the lines your clustering has defined? Explain why or why not
   - **Hint:** The syntax to create a histogram should look like the following: `manifest.groupby('CLUSTER')[<ATTRIBUTE_COLUMN>].hist();`