Report

Data

I chose this dataset from **Project Gutenberg** because I was already familiar with their platform since I used it for homework 1, but at the same time I am very fond of the murder mystery genre. I enjoy watching documentaries or fictional tv shows based on it and have also read some books by Agatha Christie and Arthur Conan Doyle before. For that reason, I chose to collect multiple documents for different **murder mystery genre** based books as my **first category**. In order to make this interesting, I decided to choose a completely opposite themed genre, which was romantic novels as my **second category**. I manually divided each book into chapters and created a separate text file for each chapter.

Methodology

For this assignment, I reused and modified the preprocessor I had created for my first homework. I made a few changes to adapt it to this task. Instead of taking input from the command line, I turned it into a function that accepts specific inputs. I also added two new parameters: `word_count` and `binary`. If `binary` is set to `True`, the function creates a bag of words in binary format (i.e., it only checks if a word is present or not). Otherwise, it counts the frequency of each word and creates a standard bag of words. I used `defaultdict` to store the word counts, which made it easy to handle the data.

To open and process multiple text files, I didn't want to do it manually, so I asked ChatGPT for help in automating the process. This saved me a lot of time and effort. Once the files were imported, I organized them into two categories based on their content.

For the probability calculations, I referred to the tutorial notebooks provided in class. I created `defaultdicts` to store word counts for both categories. I used a probability function from the tutorial, which included Laplace smoothing to avoid zero probabilities. To handle the vocabulary, I used Python's `set` data structure because it automatically removes duplicates, ensuring that each word is counted only once. After calculating the probabilities, I stored them so I could use them later for computing the log-likelihood ratio (LLR).

Calculating the LLR was straightforward. I used the probabilities I had already computed and applied the `math.log` function to them. This gave me a measure of how strongly each word was associated with one category compared to the other.

For topic modeling, I followed the instructions in the assignment PDF and used the `gensim` library, as suggested. I also used `pyLDAvis` to create visualizations of the topics. Since I wasn't working in a Jupyter notebook, I saved the visualization as an HTML file so I could view it easily.

Additionally, to analyze the topic distributions, I wrote a function that calculates the average topic distribution for each category. This made it easy to unpack and compare the results for both categories.

Finally, I created another file called `binary.py` in which I just copied over my code from the normal python file and just tweaked it according to the output from my normalize_text module. As I have previously mentioned, this file takes an input binary which creates a different way of representing the bag of words for analysis.

Result and Analysis

Before beginning my research I would like to say that I had thought analyzing novels would be interesting and while it was, each chapter didn't have enough words on an average for me to derive some meaningful analysis from it. Nevertheless, it was a learning experience for me.

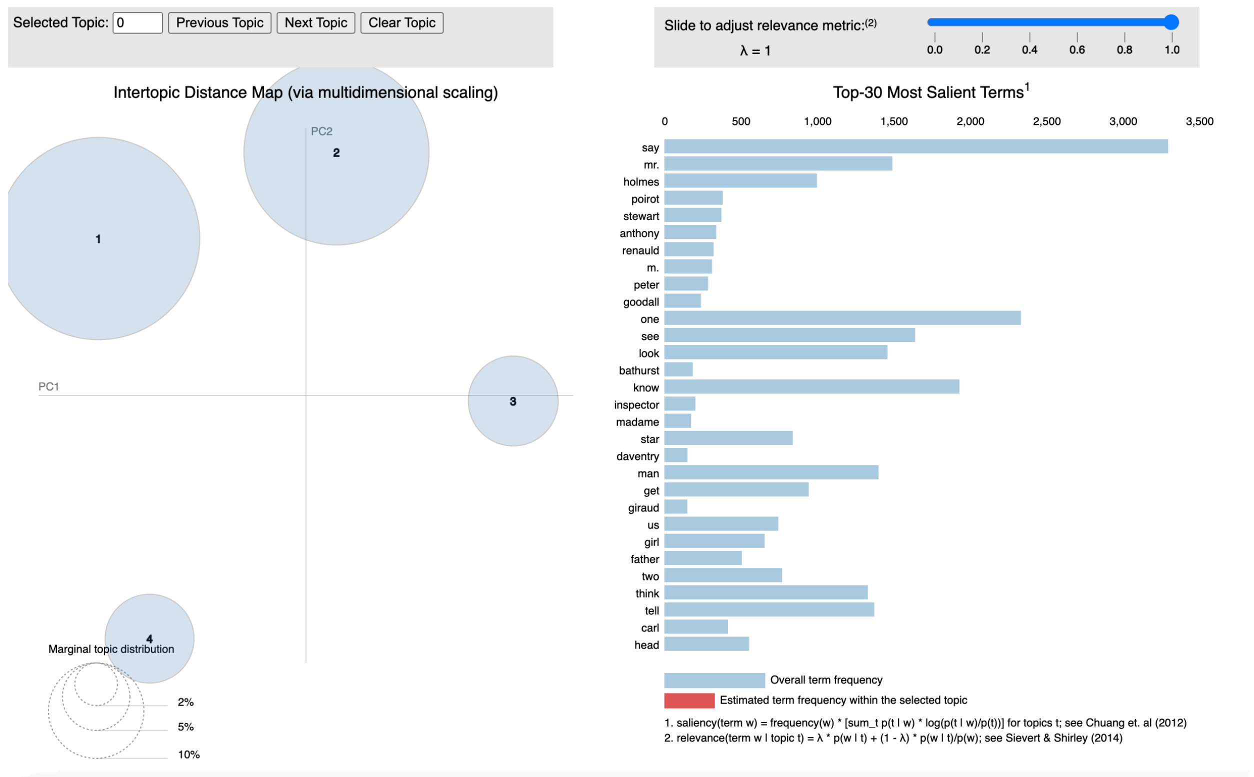
In the probability analysis, I noticed that the most common words in romance novels were mostly context-based, with character names popping up frequently. The same was true for crime novels where contextual words were common, along with a lot of character names. For example, in the crime genre, it was easy to recognize names like Holmes, Watson, and M as characters from Sherlock Holmes, and the word clue is an obvious connection to the mystery theme. However, for the romance novels, I didn't immediately recognize the names. It was only after skimming through the stories that I realized they were the protagonists.

The text output for section 2.3 (LLR) based on which I have concluded the above statements is given below:

```
Romance:
[('toward', 4.352299738971371), ('Ralph', 3.896282351700375), ('Star', 3.8738094958483162), ('Rich

Crime:
[('afterwards', 3.5212980507141696), ('grey', 3.4657281995593587), ('Holmes', 3.3761160408696713
```

The LDA vis output for section 2.4 is given below:



1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Additionally, for **2.5** I created the bag-of-words using the binary approach and didn't add any preprocessing to see if I could get any different results.

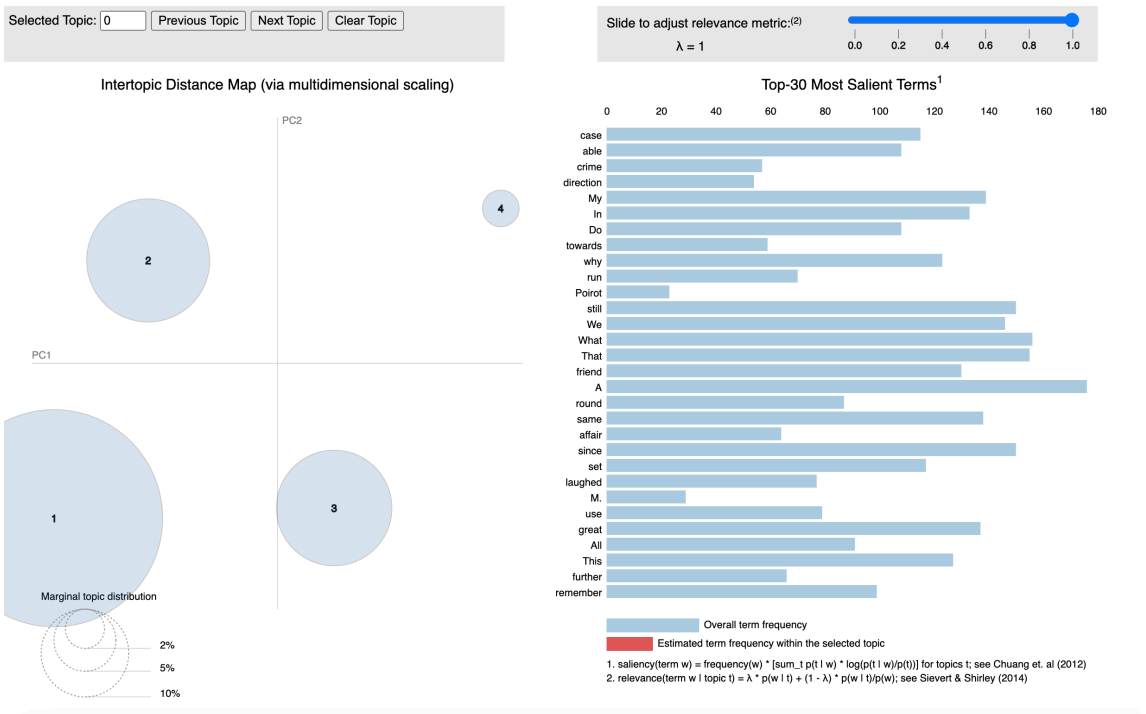
Below is the text data that I got from my `binary.py` file

```
Romance:
[('toward', 4.352299738971371), ('Ralph', 3.896282351700375), ('Star', 3.8738094958483162), ('Rich

Crime:
[('afterwards', 3.5212980507141696), ('grey', 3.4657281995593587), ('Holmes', 3.3761160408696713
```

I noticed that the data was more or less same, with just certain words being replaced with others. Overall, the output was still extremely context based.

The LDA vis output for 2.5 is given below:-



Selected Topic: 0

Slide to adjust relevance metric:(2)

λ = 1

1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

The LDA vis was a bit different than the previous version and arguably became worse due to the fact that there was no pre-processing. I thought removing those steps would make it context-free, but the result was to my disappointment, more or less the same.

Discussions