The project we implemented is Serendip, a topic modeling tool which makes exploring and discovering large text corpora easier. It provides a fluent user experience that smoothly connects different types of information across various scales and data types. Users could explore questions by intermixing texts connections, passages within texts, and sets of words that define topics at different scales in their inquiry. For example, a user might want to know what topics are included for a specific document. When exploring the topics, the user might be interested in a particular topic. He can then inquire and discover other documents that have the same topic, and he can even dive deeper into the detail of that document or topic, or vice versa. This is the “serendipity” we provide for the user, allowing them to explore all the corpus according to their interests across different scales and data types.

From the experience of constructing this project, we gained a lot of implementation skills. Before taking this course, none of us have front-end background. Even if we have done some homework and projects prior to this one, they are designed by us so that we can choose something that we can handle. But this time, since all of the papers are challenging to some extent, we are forced to grow. So most of the time we are struggling to figure out how to use CSS, JavaScript and D3 to implement some particular functions. And since this project contains a large number of interactions, now we are much more familiar with handling visualization with d3, and we also learned how to use other tools like jQuery.

The paper provides comprehensive information for reproducing the same visualization system. Basically, we can find information about 1. What corpus they are using; 2. How to preprocess data, this allows us to choose the corpus we want without struggling for data format. 3. Detailed instruction about what is implemented and why this is important. 4. And they also provide an online demo. Since this vis system has a lot of interactions, it makes it easier to examine the functionalities and decide how to structure our implementation. Therefore, we think everything is the system is reproducible.

We implementation most of the core features as the original one, and we enhanced some user experiences. For example, some interactive functions are hard to find in the original one due to lack of hints, so we add some interactions like changing the background or cursor icon when hovering the interactable elements in all three pages so that user can understand what to do easier. We also enhance the user interface of the RankViewer by enlarging the views and add a title to the right view. We skipped some minor details like annotations, history, rename, and setting. They might be helpful to explore the system, but they won’t influence the core function and correctness of the system. In the Corpus viewer, the discarded features are document/topics color annotation, hiding selections, and load/store previous selection. We also skipped some other minor functions like filtering, aggregation, and part of functions in sorting in the Corpus Viewer controller, since there are already a lot of interactions we need to do, and we don’t have enough time to finish all of them.

We used Shakespeare dataset, the same dataset as the paper. This model of 50 topics was built on a corpus of the 36 plays of William Shakespeare. The documents were divided into

chunks of 1000 words for the model and then stitched back together to create full documents. Excluded stopwords as defined by Mallet, as well as additional hand-curated stopwords specific to Shakespeare. Metadata includes title, year, and genre.

We already provide a simple user scenario in the first paragraph, let's explain it in detail. Say the user wants to know what content is inside 1 King Henry VI. He first enters CorpusViewer, sort the topics by this document, and find the document has a topic (No.2) that includes words like "battle", "war", and "brave". So now he knows 1 King Henry VI might include some content related to them. Since he is interested in this topic, he wants to know what else documents have the same topic. So he the document by this topic, and found Macbeth also contains the topic. Since he doesn't know much about Macbeth, he wants to know its content. So he goes to the TextViewer of Macbeth, and see all the paragraphs. He then wants to know where are the parts that are mostly related to the topic, so he checks the topic overview, and click the place where the topic occurs the most and see the exact paragraph. He sees the word "swords" in the paragraph, and he wants to know what else topics also have the same word. So he clicked the word, and enters RankViewer, and found that topic No. 6 also has the same word. If he wants to know more about topic 6, he can go back to CorpusViewer and explore again.

Our implementation loss the connection between documents and words. It provides a view of multiple documents and multiple topics (corpus view), view of the single document to multiple topics and words (text view), and the view of topics to words (rank view). Through those views, we build a fluent exploration from a higher level document to lower level word. However, they will be awkward if users need to find the relationship of documents to single/multiple words, i.e. how many times does one word appear among documents. The other limitation is the lack of information in ranking view. For example, the location of the color-coded bar represents the frequency of a word in the corresponding topic. However, there is no way to know which end represents a higher frequency. Besides that, the ranking view might need more interactions. Users can select a word in frequency distribution view to add it to the bar chart, but there is no way to refer a topic through selected words. An effective improvement can be made if we highlight selected words in frequency distribution view. Those limitations give us suggestions for future improvement.