Building An Interactive Visualization Tool

Henry Tran, David Vu, Peri Venetis, Kevin Nguyen, Polat Tarim

Toronto Metropolitan University

Abstract

This study investigates the adaptivity of an approach to search engine data visualization. This approach or baseline model builds an interactive visualization tool to visualize the interconnections between entities and gain insight into similarity clusters. Secondary research was conducted from credible sources through academics such as a Ph.D. student and peer-reviewed academic journals on such an approach and model. Our contribution is to an already existing model of visualizing data with an interactive network graph. Our analysis showed that we could show data distributions in multiple ways based on this approach but also add more information. We conclude that network graphs are an effective and adaptable way to visualize data.

Keywords: Network Graph, Visualization, Information Retrieval, Search Engine, Similarity Clusters, BM25, Text Classification, Naive Bayes, K-nearest Neighbors, Interactive

Building An Interactive Visualization Tool

It can often be difficult to have a great or instinctual understanding of large data distributions. You can attain exactly this understanding by interactively visualizing this distribution. We were motivated to solve this difficulty with a tool that helps present results easily and with high comprehensibility. We contributed to this problem by showing how an interactive visualization tool, namely, a network graph, is an effective and adaptive way to visualize data distributions. We suggest its adaptability by visualizing different data distributions while building on an existing implementation of an interactive network graph by Diogo A.P. Nunes.

Formal Description

The analysis shows the adaptability and effectiveness of a baseline model provided by Nunes by further improving the baseline model with additional implementations. The methods, techniques, and procedures used in this study are done accordingly to fit what Nunes had done earlier. The means that this research has used mainly consisted of utilizing a BM25, K's Nearest Neighbour (KNN), and Naive Bayes algorithms alongside the quality of life improvements for the baseline model from Nunes. The main objective is to show how the model is adaptable and effective, as it can be visible through the additions made in the research.

Baseline Model

Nunes' implementation involves multiple functions such as "create_node_trace(G)" and "create_edge_trace(G)" which essentially create a trace for each node and edge for the purpose of using its positions to plot the graph. This positional information is used later in combination with the interactive slider function "get_interactive_slider_similarity_graph()" to plot the graph accordingly. These functions are consistent with our implementation and serve as part of the baseline model for our new model.

New model and Implementation

Building onto the baseline model, it is observed through Figures 4-6, that the new model contains KNN, Naive Bayes, and BM25 algorithms for text classification and ranking which the baseline model did not include.

BM25 Algorithm

The BM25 python file takes in the constructed index with regard to the corpus, the document list, and 2 constants b and k used for calculations later. For the sake of simplicity, b and k are set to 0.75 and 1.5 respectively whenever an instance of this class is initialized. BM25 ranking uses the RSV function to calculate the similarity score between a query and a document as seen in Figure 7.

Naive Bayes Algorithm

NaiveBayes.py takes in the dictionary generated by the inverted index and the document list. For training purposes, function fit(self, X_train, y_train) is passed the two training helper files to store all relevant information in a dictionary with the category class as a key and the relevant document IDs and keywords as the values. The probability that a term in the query is relevant given each category class is computed according to the formula as seen in Figure 8.

As seen in Figure 9, testing is then implemented by computing the probability of the category. Finally, we multiply this with the sum of the probabilities of all query terms of a single category. Picking the most probable category will give the representative category text class for the query as seen in Figure 10.

KNN Algorithm

The KNN python file takes in the constructed index with regards to the corpus, our k-value, the number of nearest neighbors identified by ‘k’ for later use, and “info” representing the list of documents obtained from our collection. The model is trained by loading the relevant documents from the training helper file to a dictionary “train_dict” in the function fit(X_train). “predict” then is used for making predictions based on the trained model while using “scores”, we list the scores that indicate how similar each item in the training set is to the item that the prediction is being made for. The method also uses these scores to identify the k-nearest neighbors and then uses the classes of those neighbors to make a prediction. Additionally, the BM25 ranking function is reused as a similarity score helper function to find out the set of ‘k’ nearest neighbors or most relevant documents to the input query. The majority of the category class in the nearest neighbor set will be chosen as the representative category class of the query.

Related Work

Diogo A.P. Nunes, a student pursuing a Ph.D. in NLP shows through his article how to build an interactive visualization tool namely a network graph. His implementation uses two python packages Plotly and NetworkX. Nunes separates his implementation into four steps. He first gets the sample similarity by calculating the similarity each sample has with every other sample using word embeddings. Next, he uses the NetworkX python package to visualize this data into a graph with N nodes. He then uses the Plotly python package to loop over all the nodes of the NetworkX graph collecting their positions which will be used to plot a new graph. He also creates a trace over the edges of the first graph. Now that he knows how to plot the network graph using the positional information of the nodes and edges, he is able to create an interactive slider where the user can move the similarity limit and plot its respective network graph. Finally, hovering over the nodes shows the terms that belong to each cluster.

Our approach differs from his first step as our sample similarity is different. We use a program to find the similarity scores using the BM25 model. Secondly, the way we visualize our information is different from Nunes. For the nodes of our network graph, while Nunes shows the words as shown in Figure 1, our implementation shows the documents in a corpus related to a query. Furthermore, each node is a different color based on how high the BM25 score is with respect to the query. Also, documents that are relevant to each other are connected based on cosine similarity. Lastly, we use algorithms of KNN, Naive Bayes, and BM25 for text classification and ranking. Our contribution to this approach shows how this interactive tool can be implemented to visualize different data distributions and similarities between entities.

Closely related papers also indicate the usefulness and reduced visual complexity of using two-dimensional graphs to show similarity clusters. Using a variety of two-dimensional maps, each of which encapsulates a distinct feature of the similarity structure, demonstrates how to display a collection of pairwise similarities between objects.[1] Moreover, when a group of nodes in a cluster are temporarily replaced with abstract nodes, a clustered graph can significantly decrease the visual complexity.[2] Thus, it is strongly suggested that using a variety of network graphs to show similarity clusters is a comprehensible and useful way to visualize data distributions.

---

[1] Cook, J., Sutskever, I., Mnih, A. &amp; Hinton, G.. (2007). Visualizing Similarity Data with a Mixture of Maps. <i>Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics</i>, in <i>Proceedings of Machine Learning Research</i> 2:67-74 Available from https://proceedings.mlr.press/v2/cook07a.html.

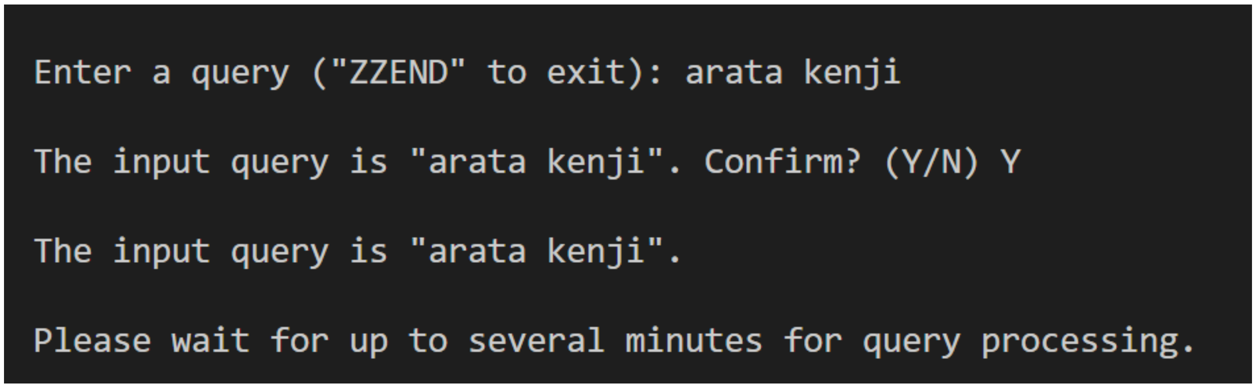
[2] Huang, Xiaodi & Lai, Wei. (2006). Clustering graphs for visualization via node similarities. Journal of Visual Languages & Computing. 225-253. 10.1016/j.jvlc.2005.10.003.

Demonstration

Starting the program is easy, you just need to run the file **“final_project.py”**

You may enter a search query for example: “**arata kenji”**

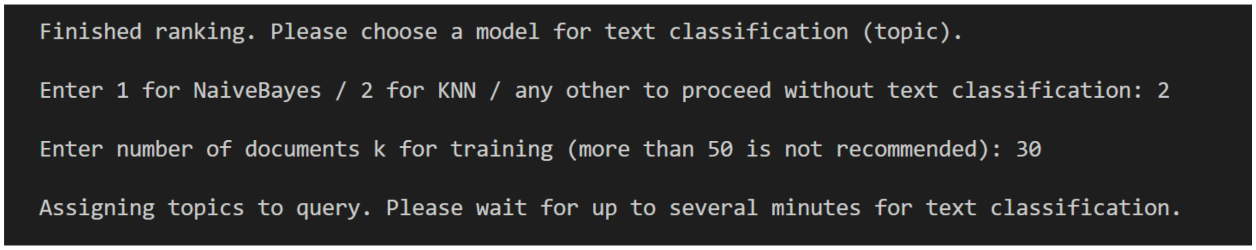
1. Enter the search query and wait for it to be processed



```
Enter a query ("ZZEND" to exit): arata kenji

The input query is "arata kenji". Confirm? (Y/N) Y

The input query is "arata kenji".

Please wait for up to several minutes for query processing.
```

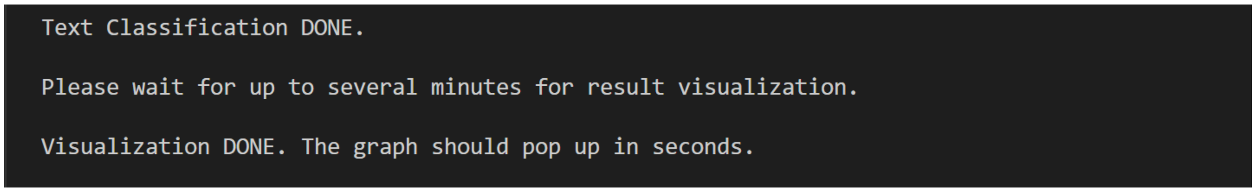
2. Select the ranking method and the number of documents ‘k’ for training.

   We selected KNN with k=30 for training.



```
Finished ranking. Please choose a model for text classification (topic).

Enter 1 for NaiveBayes / 2 for KNN / any other to proceed without text classification: 2

Enter number of documents k for training (more than 50 is not recommended): 30

Assigning topics to query. Please wait for up to several minutes for text classification.
```

3. You will get a confirmation when the classification is done.



```
Text Classification DONE.

Please wait for up to several minutes for result visualization.

Visualization DONE. The graph should pop up in seconds.
```

4. As seen in Figure 2, a new window will open to show an interactive graph with a similarity score slider and BM25 color legend. Notice that in Figure 3, when the

similarity score threshold changes, only documents with the same or higher similarity score are connected.

Comparison

The implementation of our graph has properties that Nunes' baseline model does not have. If you refer to Figure 2, you can identify three main properties that the baseline model, Nunes, does not have. Firstly, we implemented the property for nodes to have different colors with respect to their BM25 score shown on the right-hand side of Figure 2. Secondly, the baseline model only shows similarity clusters for word embeddings while our model shows similarity clusters for the documents. You may hover over nodes similar to the baseline model, however, our model not only shows the document, but the context in which the query is relevant in the document as you can see in Figure 2. For example, in the query, "arata kenji" you can see in Figure 2 the sentence in which either both or one of the terms occur. This further shows the adaptability and effectiveness of this approach through the addition of more information and properties to the baseline model.

Limitations

Visualizing data with similarity clusters poses some limitations. Clustering outliers is a potential difficulty that can cause outliers to get their own clusters or possibly drag other nodes to them. It might be best to remove outliers when visualizing data with many outliers for this implementation or rather visualize them in a separate graph. Another limitation is the time to produce a network graph based on large data distributions can take possibly up to hours as you collect information about the data. One way to address this limitation is to make sure the data and context of your query are accurate and reliable. For example, if your data set contains a corpus of library books, it would not make sense to search for an infographic but rather specify your search to a topic. In addition to the overall efficiency of the program, the training data for text classification models are not guaranteed to be in line with the original collection. This would cause uncertainty and inconsistency as the program tries to search for a query term that does exist in the database but is not listed as a keyword. The result would return "None" in that case so that the entire process is not terminated without one crashed component. Lastly, a network graph can be complex which could lead to a misinterpretation of the data. For example, we recommend not to train over fifty documents because then the graph would be filled with many nodes and connecting lines which could make it hard to read and interpret. Addressing this would include ensuring the reliability and accuracy of the data so that showing similarity between entities is perhaps more accurate for fewer results. Therefore, with the limitations of network graphs, it is observed that they may be prevented by addressing them accordingly.

Conclusion

Through investigating the effectiveness and adaptability of using network graphs to visualize data distributions, it is observed that using this visualization tool can be adapted to use many different types of data distributions to plot this data. For further adaptability, we are able to implement new properties to this visualization tool such as more information when hovering over nodes, and differing colors for nodes to show more or fewer similarities. Furthermore, we are able to implement new algorithms to the baseline implementation. This adaptability of the baseline model shows that you can effectively visualize data of many types to show different similarities in a comprehensive way.

References

Cook, J., Sutskever, I., Mnih, A. &amp; Hinton, G.. (2007). Visualizing Similarity Data with a Mixture of Maps. <i>Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics</i>, in <i>Proceedings of Machine Learning Research</i> 2:67-74 Available from https://proceedings.mlr.press/v2/cook07a.html.

Huang, Xiaodi & Lai, Wei. (2006). Clustering graphs for visualization via node similarities. Journal of Visual Languages & Computing. 225-253. 10.1016/j.jvlc.2005.10.003.

Nunes, D. A. P. (2021, December 15). Visualising similarity clusters with interactive graphs. Medium. Retrieved December 18, 2022, from https://towardsdatascience.com/visualising-similarity-clusters-with-interactive-graphs-20a4b2a18534.

Footnotes

[1]Cook, J., Sutskever, I., Mnih, A. &amp; Hinton, G.. (2007). Visualizing Similarity Data with a Mixture of Maps. <i>Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics</i>, in <i>Proceedings of Machine Learning Research</i> 2:67-74 Available from https://proceedings.mlr.press/v2/cook07a.html.

[2]Huang, Xiaodi & Lai, Wei. (2006). Clustering graphs for visualization via node similarities. Journal of Visual Languages & Computing. 225-253. 10.1016/j.jvlc.2005.10.003.

Figures

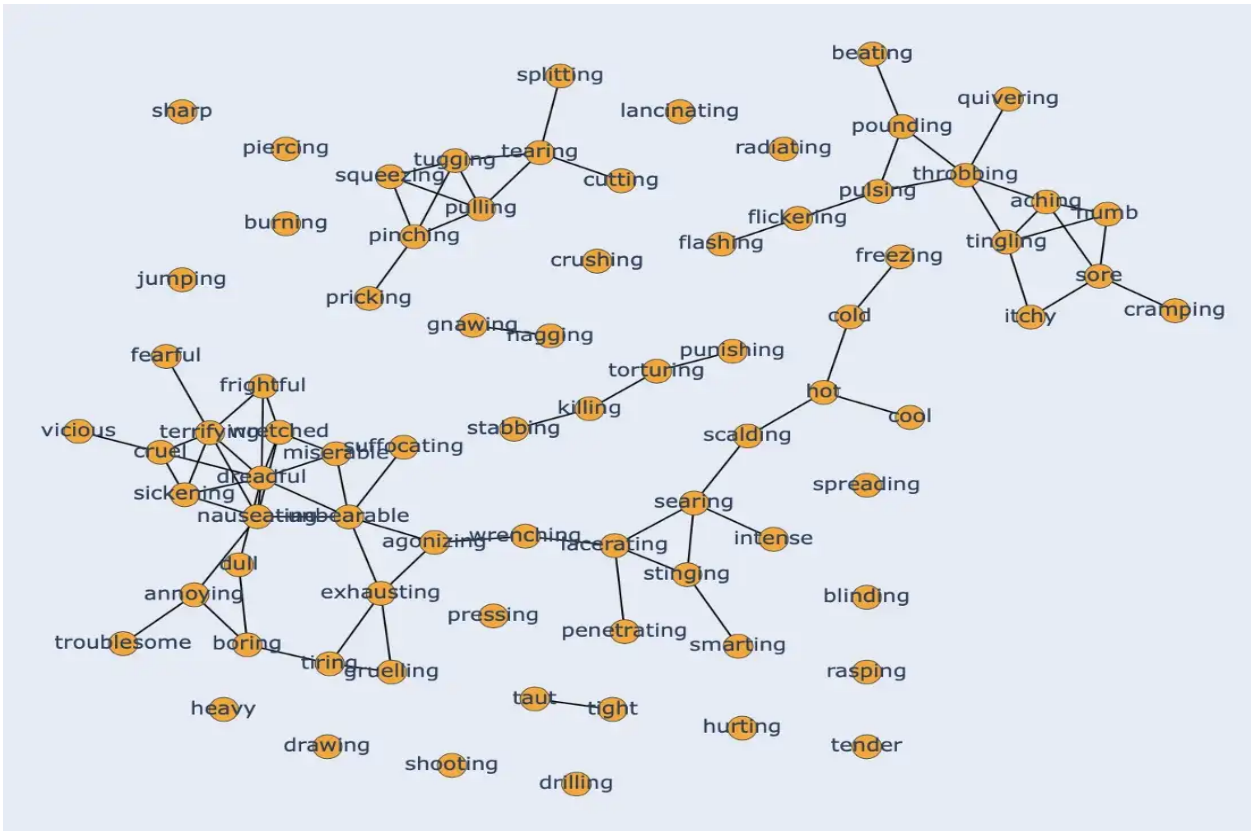


Figure 1. Graph plotting with word names in each node. Hovering over the plot will show word names for each cluster.

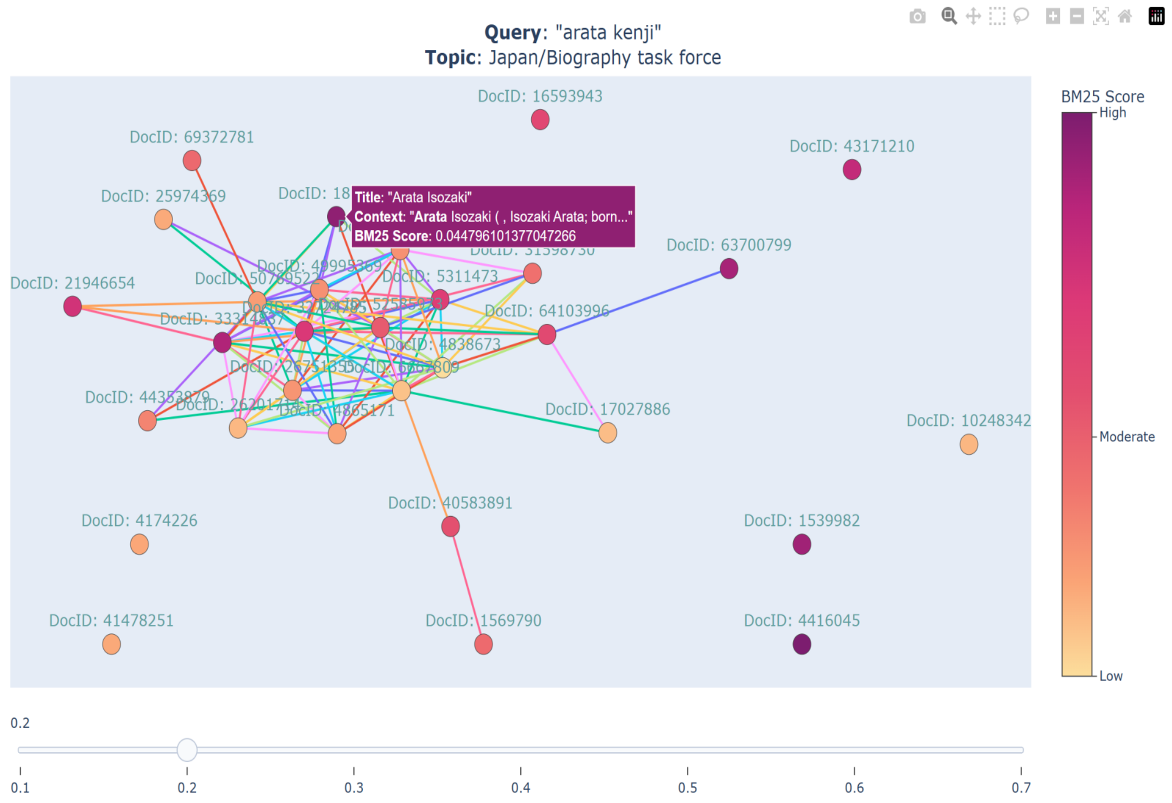
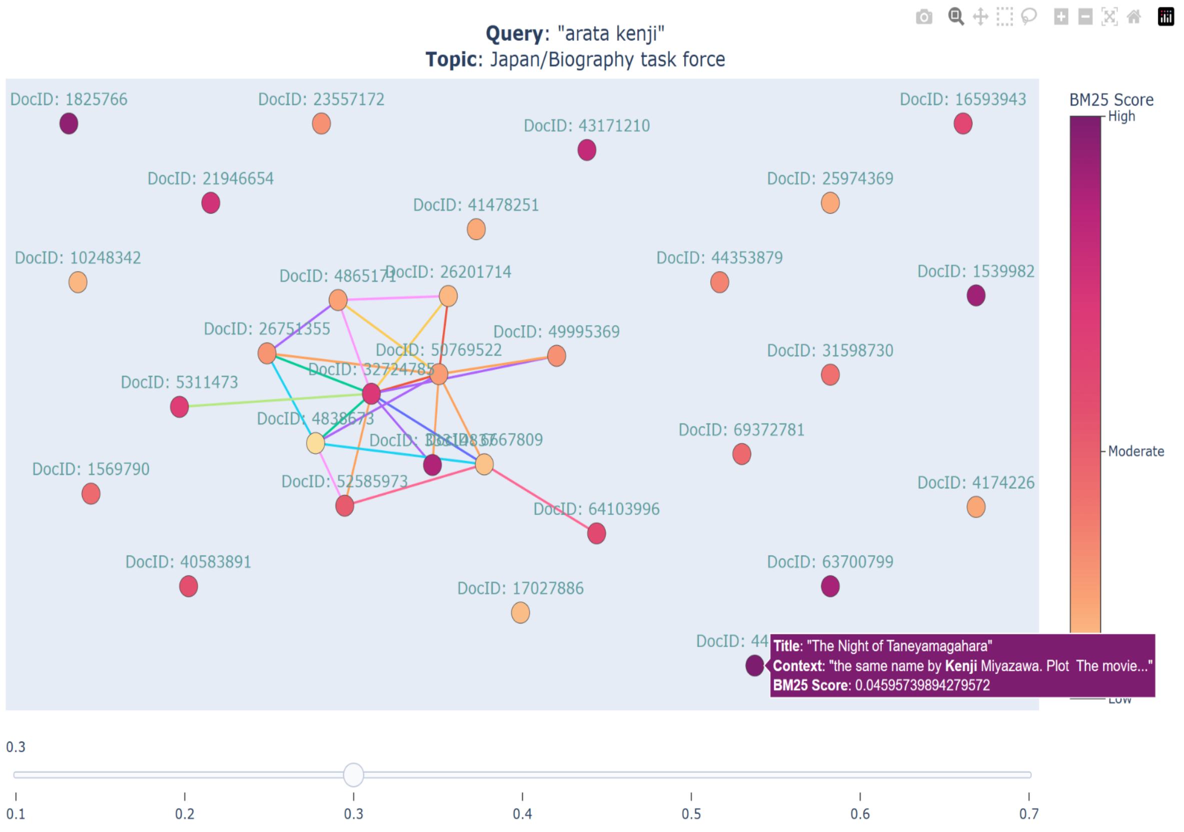
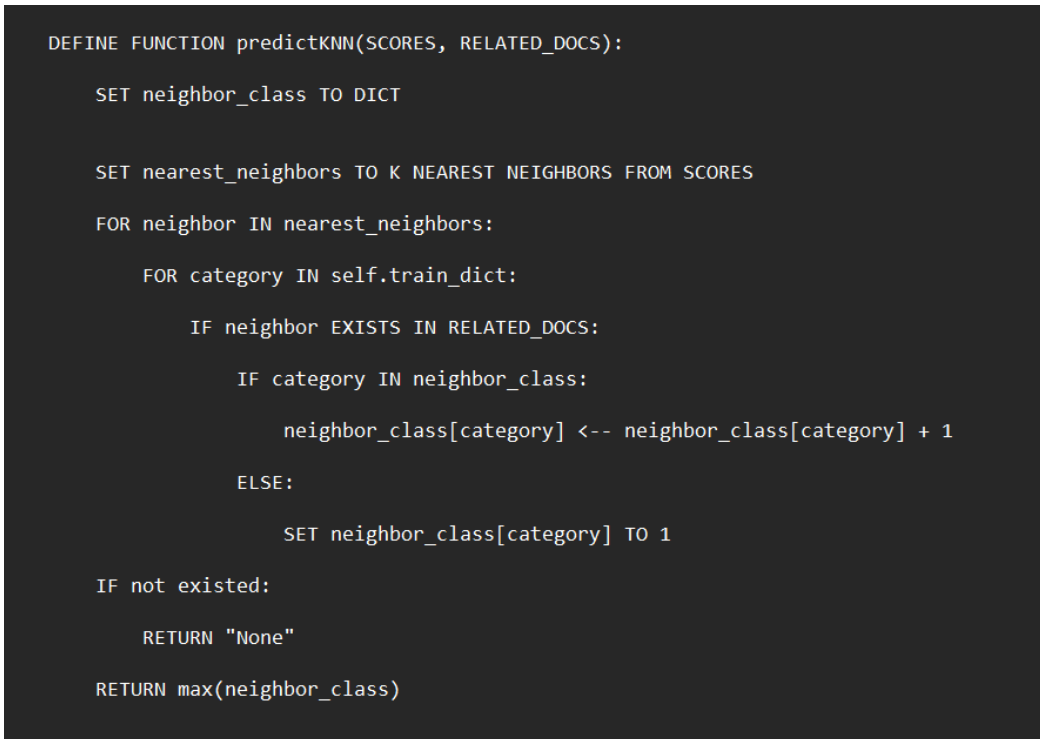


Figure 2. Example of our implementation's results. Contains the graph, the BM25 score legend on the right, the interactive slider which changes the similarity score limit below the graph, and also hovering over the nodes shows the documents in the cluster, the title of the document, the document id, the similarity score, and the context in which the query is relevant in the document.



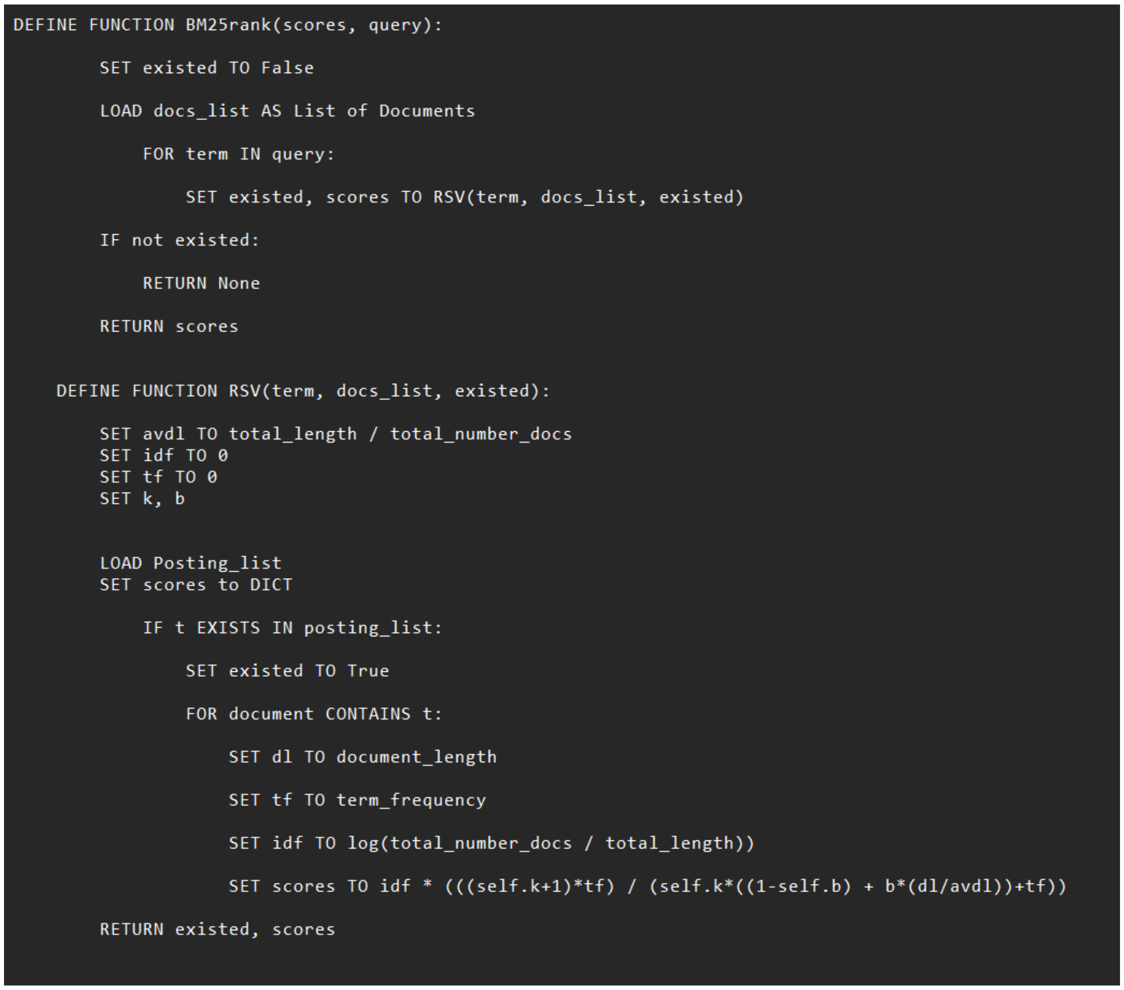
Title: "The Night of Taneyamagahara"
Context: "the same name by **Kenji** Miyazawa. Plot The movie..."
BM25 Score: 0.045957398942795 72

Figure 3. Example of our implementation's results when documents similarity threshold changes. As compared to Figure 2, as we raise the similarity threshold, fewer document nodes are plotted as a result of not meeting the similarity limit.



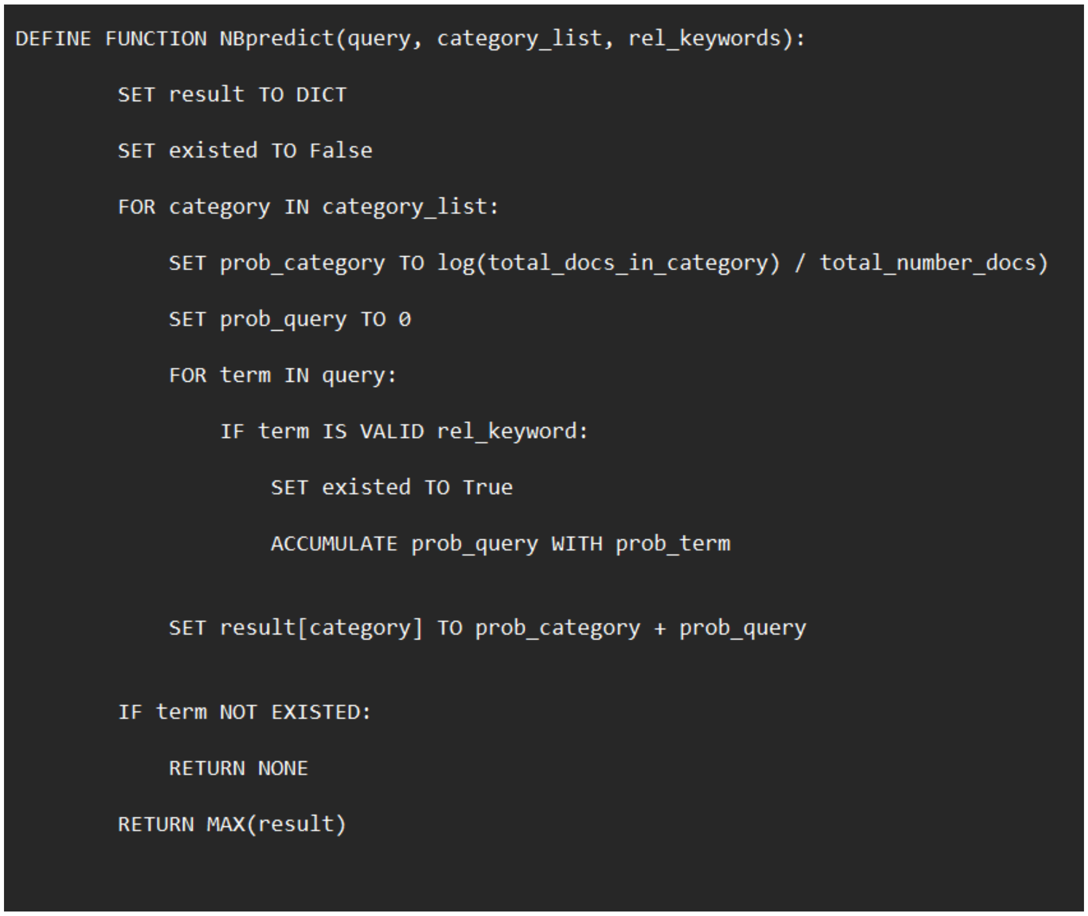
```
DEFINE FUNCTION predictKNN(SCORES, RELATED_DOCS):
    SET neighbor_class TO DICT

    SET nearest_neighbors TO K NEAREST NEIGHBORS FROM SCORES
    FOR neighbor IN nearest_neighbors:
        FOR category IN self.train_dict:
            IF neighbor EXISTS IN RELATED_DOCS:
                IF category IN neighbor_class:
                    neighbor_class[category] <-- neighbor_class[category] + 1
                ELSE:
                    SET neighbor_class[category] TO 1
    IF not existed:
        RETURN "None"
    RETURN max(neighbor_class)
```

Figure 4. Flowchart for KNN algorithm



```
DEFINE FUNCTION BM25rank(scores, query):

        SET existed TO False

        LOAD docs_list AS List of Documents

            FOR term IN query:

                SET existed, scores TO RSV(term, docs_list, existed)

        IF not existed:

            RETURN None

        RETURN scores


    DEFINE FUNCTION RSV(term, docs_list, existed):

        SET avdl TO total_length / total_number_docs
        SET idf TO 0
        SET tf TO 0
        SET k, b


        LOAD Posting_list
        SET scores to DICT

            IF t EXISTS IN posting_list:

                SET existed TO True

                FOR document CONTAINS t:

                    SET dl TO document_length

                    SET tf TO term_frequency

                    SET idf TO log(total_number_docs / total_length))

                    SET scores TO idf * (((self.k+1)*tf) / (self.k*((1-self.b) + b*(dl/avdl))+tf))

        RETURN existed, scores
```

Figure 5. Flowchart for BM25 algorithm



```
DEFINE FUNCTION NBpredict(query, category_list, rel_keywords):
        SET result TO DICT
        SET existed TO False
        FOR category IN category_list:
            SET prob_category TO log(total_docs_in_category) / total_number_docs)
            SET prob_query TO 0
            FOR term IN query:
                IF term IS VALID rel_keyword:
                    SET existed TO True
                    ACCUMULATE prob_query WITH prob_term

            SET result[category] TO prob_category + prob_query

        IF term NOT EXISTED:
            RETURN NONE
        RETURN MAX(result)
```

Figure 6. Flowchart for NaiveBayes algorithm

$$RSV^{BM25} = \sum_{i \in q} \log \frac{N}{df_i} \cdot \frac{(k_1 + 1) tf_i}{k_1((1-b) + b\frac{dl}{avdl}) + tf_i}$$

N/df(i) is the IDF for term i
tf(i) is the term frequency of term i in a document
dl is the length of the current document
avdl is the average document length over the collection

Figure 7. BM25 Algorithm

$$P(x_k \mid c_j) \leftarrow \frac{T_{c_j x_k} + \alpha}{\sum_{x_i \in V} [T_{c_j x_i} + \alpha]}$$

where x(k) is one of the query terms
c(j) is one of the category classes
T(c(j)x(i)) is the total term frequency of term x(i) with respect to category c(j)
ά is a constant, which is 1 in this implementation

Figure 8. Naive Bayes Algorithm

$$P(c_j) \leftarrow \frac{N_{c_j}}{N}$$

N(c(j)) is the number of documents relevant to the category c(j)
N is the total number of documents in the collection (corpus)

Figure 9. Naive Bayes Testing

$$c_{NB} = \underset{c_j \in C}{\operatorname{argmax}} \left[\log P(c_j) + \sum_{i \in positions} \log P(x_i \mid c_j) \right]$$

Figure 10. Naive Bayes. Picking the most probable category.