Project 15: Web search with Linguistic Expansions from Term-Frequency Histograms

Yu Hang Lee (920000520) , Sean Ellis (920737841)

Department of Computer Science, San Francisco State University

CSC 664-01: Multimedia System

Dr. Rahul Singh

May 16, 2021

Abstract

Query definition is crucial to the success of web search. In this project, we are exploring how starting from a given query, alternative query formulations can be devised that can help improve information retrieval on the web. The basic idea of this project will involve the use of WordBars. We want to implement the WordBars methodology. In this modern era, most of us have access to the Internet and we seek answers from the web. So trying to build a query search is important to us in order to make others' life easier. We do have Google, Yahoo, Bing, and others web search engines in the world but this project we are trying to build it with high frequency terms and synsets.

Proposed problem formulation

The specific thing that we are trying to do is building a query search on a link that can be accessed by anyone and can show the result to them while they can choose the synsets or other features that are provided in the link. We are building a query that can show the users that the top 10 result from the search and then can see the data showing how many times the words are counted in the query search bar. We want to build a TF/IDF (Term Frequency / Inverse Document Frequency) while using the mathematics calculations from the lecture note of the class. This formulation of the project fits into solving the general problem?

Prior work in the area

In this paper, Cost and Benefit of Using Wordnet Senses for sentiment analysis, it has shown that Wordnet senses are better features compared to lexeme based features. Even though NLP (natural language processing) applications heavily depend on annotated resources, such a study, to the best of our knowledge, has never been done. This paper shows us that the cost assumption if we want to build the whole project up running in the future, there are two types of costs associated with each of the machines: a one-time setup cost or fixed cost and a variable cost or running cost. The setup cost is the cost of training the model. The running cost is the cost of obtaining polarity prediction for a document using the model trained. It is important for us to know the cost because while we are using Google API, if we search more than 100 websites, Google will start to ask us to pay for using their API. Other costs if we want to keep this project running and open to public users to use it.

In the article, An advanced guide to NLP analysis with Python and NLTK, WordNet is a large lexical database corpus in NLTK. WordNet maintains cognitive synonyms or synsets of words correlated by nouns, verbs, adjectives, adverbs, synonyms, antonyms, and more. There are codes provided in this article about similarity comparison, tree and treebank and named entity recognition. We implement and make some changes for similarity comparison to help us for our backend coding. Since we are not building everything ourselves, we found some open source code that is available on the web in order to help us implement the query search analysis that we created.

High-level Description

In order to evaluate and analyze the proposed problem, we need to implement a responsive User Interface that can display the findings of our computational results.

The calculations that we conduct when given data need to be performed Server-Side. This will allow us to create and manipulate any formula or method necessary to analyze and achieve our query formulation/refinement. However, without a responsive interface, we will not be able to conduct any Research Analysis as to what method/implementation is most effective. This is why we need to build a Client-Side Website that allows us to conduct tests and Case Studies. This Client-Side Application needs to be able to request and send necessary Data to the Server. This Data will be used to initiate the Search Query, as well as help narrow down refining of the results. It is important that the Client-Side Application helps identify certain terms or synsets that the User is asking to retrieve. In addition, the Application may be used to formulate new Queries and Search Terms if the User chooses to utilize.

Lastly and more importantly, we either need to use manually imported Websites to search Query terms through, or implement a Web Search Engine that will be able to retrieve URLs on the World Wide Web. If we use a local database that has a specific/restricted amount of websites to search through, we are limiting ourselves to those topics and search queries used. However, if we implement a Web Search Engine, we can test any distinct query against the entire Web, and not have to worry about storing Website Data locally.

Low-Level Implementation

Back-End.

In order to achieve the Math-based calculations and query formulation, we decided to write our program in the Python Scripting Language. This is used to complete all our Term Frequency calculations, our Web Retrieval, and formulating our data for Server-Side retrieval.

Within Python, we found that the Google Custom Search API will achieve our Web Retrieval. This API is implemented using an API Key provided by Google. One limitation to this, however, is that we can only conduct up to 100 Queries per day. This limitation restricts our daily Search Analysis as we can only retrieve up to 10 documents per query.

Python is also used to calculate our Inverse Document Frequency. This is calculated by using the following formula:

Inverse Document Frequency (IDF)

$idf_j = \log_2 (N/df_j)$

- N : total number of documents
- df_j : document frequency of term j (number of docs in which j appears)

When calculating each terms document frequency, we tokenize and compare the query against each Websites: Page Title, Snippet (Preview provided by Custom Search API), and the first 25,000 HTML Characters (if able to be read). Retrieving the pages HTML Source Code is achieved by using Python's Built-in Library called ***requests***.

Lastly, Python is the main source for importing and conducting our Wordnet Query Analysis and refinement. This is achieved by importing the NLTK Corpus API. NLTK allows us to

create a list of Stop Words, conduct stemming on both the HTML Page and User Query, and to find different Synsets for comparison and analysis.

Server-Side.

Implementing a Client-Server Application was vital for our experiment. We needed something to server and retrieve data from both the User, and Python. We chose to implement an ExpressJS Server. This allows the User to conduct real-time Web Queries, and retrieve real-time results from the Google Custom Search API. Without implementing the Express Server, we would have needed to implement a Console-based Application. ExpressJS allows us to serve a pretty and responsive UI for testing purposes.

In order to retrieve and serve the formulated data back to the User, we needed to implement a way to store and retrieve that data. Because our Python and Javascript Programming skills are very limited, we were only able to create Server-side JSON files to store and read the data from. These JSON files are created based on the Remote IP Address of the User, and distinctly hashed and named accordingly. This allows the Application to store and retrieve the data based on the specific User, and the Server doesn't confuse or return mixed results.

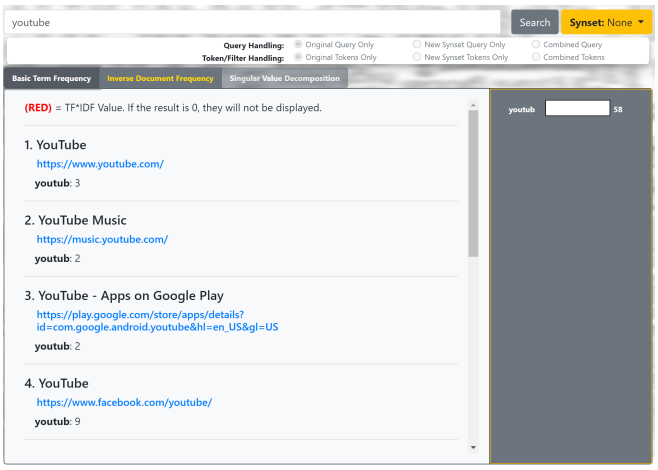
Client-Side.

As a final requirement for requesting and serving queries and data, we needed to create our User Interface. We chose to create a Client-Side HTML Website using EJS, an Embedded JavaScript Template for our Node Application. EJS allows us to easily handle all the data that is served to the Client from the Server. Lastly, we created our basic layout using Bootstrap elements for quick HTML/CSS deployment.

Experimental Evaluations

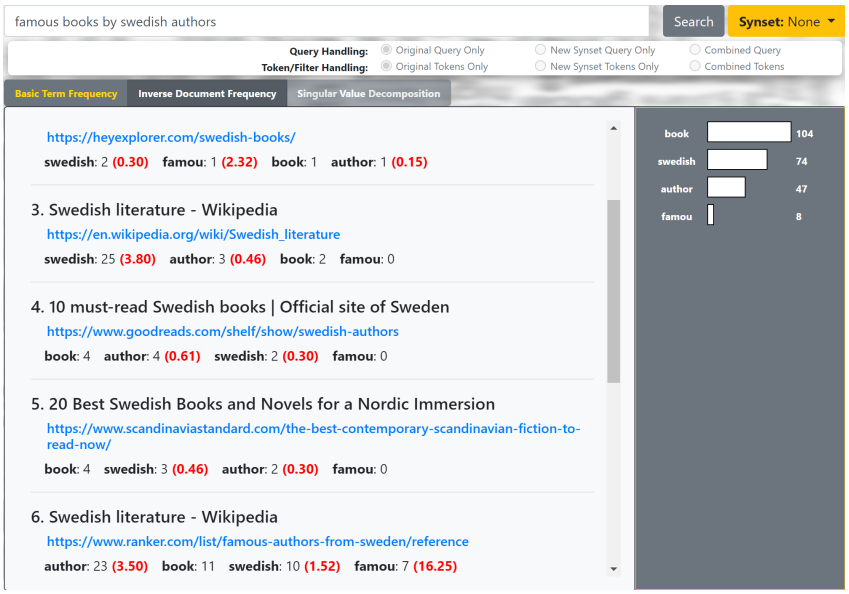
Navigational/Transactional Search Query

Our first study began with Navigational/Transactional Search Queries. This type of Query is used when a User is looking for a specific website/source. The first example we use is searching for the term: *Youtube*. When conducting a Basic Search Query, that is, not adding or subtracting any Search Terms to the Web Search Engine or Term Frequency Calculation, we end up with these results (to the left).



In addition, we did not provide any Screenshot for the other Synset/Query Handling scenarios because Wordnet is not able to formulate variations of Youtube, so we retrieved the exact same results.

For our next Navigational Search Query, we tested the Query: *famous books by swedish authors*. Although this search query wasn't as specific as the last, it still tests how the program can narrow down the retrieval pages. As speculated from the image (to the right), we notice that we retrieve TF-IDF values. Entries 3 and 6 show how the



values ‘swedish’ and ‘famou’ are more valuable within the 10 documents retrieved. However, this is only searching the Web when tokenizing the original search query. When we conduct the same search query, but specify that we also want to “Combine” Hypernyms to the Token/Filter Handling list, we don’t find much of a difference in the TF-IDF calculations. We observe that the words “scandinavi, public, and commun” are added to the list of words that are searched within each Website. The only notable difference that this refinement makes is for Entry 5.

5. 20 Best Swedish Books and Novels for a Nordic Immersion

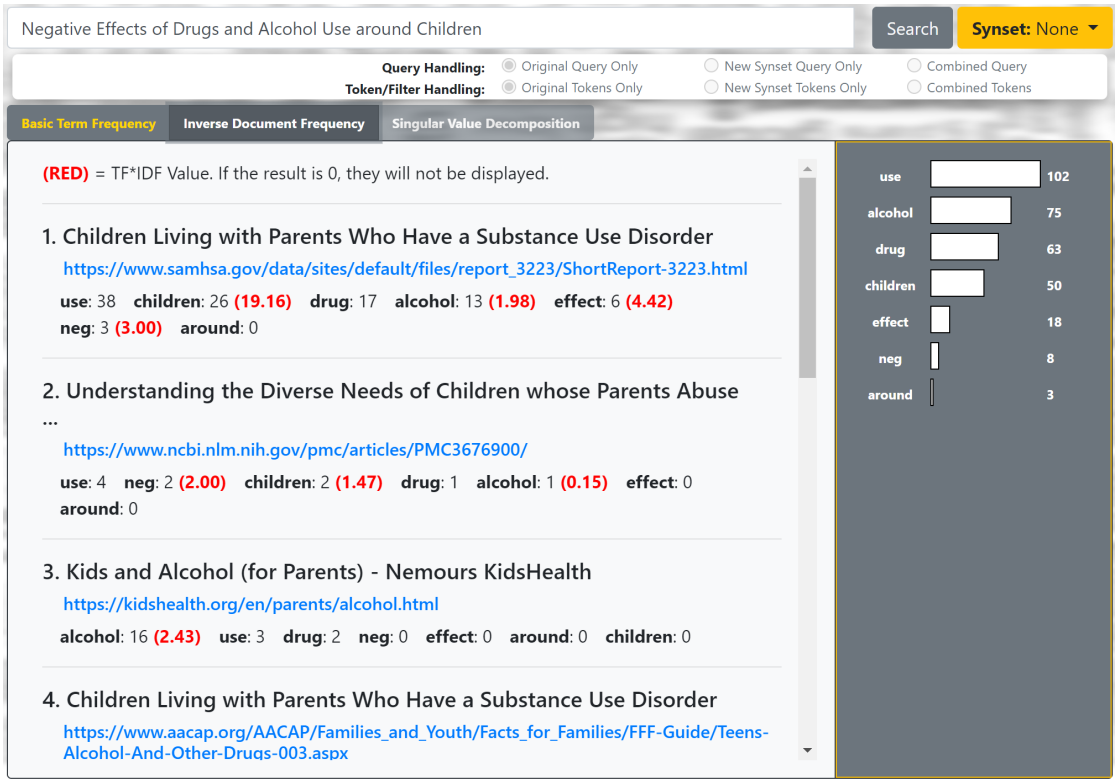
https://www.scandinaviastandard.com/the-best-contemporary-scandinavian-fiction-to-read-now/

scandinavian: 5 **(8.68)** **book**: 4 **swedish**: 3 **(0.46)** **author**: 2 **(0.30)** **famou**: 0 **public**: 0 **commun**: 0

When we added ***Hypernyms*** to the list of words that we conduct our search with, Entry 5 suddenly became much more valuable than before because it contains the word scandinavian, which is in-line with searching for swedish literature.

Informational/Scholarly Search Query

After conducting random Transactional and Navigational searches, we decided to broaden our search terms. We wanted to see how our Application handles and displays information regarding Factual and Logical Articles. One of the tests that we conducted was: *Negative Effects of Drugs and Alcohol on Children.* The purpose of this type of query is to test how our Synset choices might be able to find additional Term Frequencies that can help narrow our results. When we use the Default/Basic Searching, we obtain the following results:



This shows us that the terms ‘children, effect, and alcohol’ are high-valued terms throughout the 10 documents. As identified within the first Website, the term Children is highly present.

Our next test with the same query is conducted by including the *Hyponyms and Hypernyms* found for each Query Term. This resulted in no changes as the Hyponyms and Hypernyms were oddly different terms that did not match any words within the documents.

Conclusions

Query Refinement is no simple task.

When attempting to formulate a new query, there are so many different options to choose from. We discovered that you can either, (1) add Synset terms to the query, or (2) populate Search Engine results with the original query, and formulate differing Term Frequency analysis against the original findings to sort/sift through each returned result.

Adding Synset Terms.

When using this approach, it seems as though using different Synsets finds widely varying results, and cannot reliably refine queries to get exact matches. This is shown when a user searches for Transactional/Navigational Websites. Synsets start to branch away from the specific terms that the User is looking for.

Comparing Results Against Additional Synset Terms

This type of filtering was interesting to analyze. Although most Websites didn't necessarily include all the Synset variations that the Python Wordnet was able to populate, it still was able to add a rare element to each website. If a website included a Synset term within its description or title, it showed slight increased occurrences of the abstract terms that the User might be looking for.

Web Scraping Limitations

One thing we discovered was that when calculating the IDF for each Search Term, it was very difficult to get reliable calculations. For instance, our Custom Search API only allows us to retrieve up to 10 documents at a time. This limits our IDF calculation to comparing each individual document with a total of 10 documents. So when the User searches for a basic,

specific term like “Kittens”, Google is going to return 10 documents that all include the word Kittens. This means that our IDF always calculated to 0, because we used the following calculation:

$$\log_2\left(\frac{\textit{Total Documents}}{\textit{Total Documents Term is Present}}\right)$$

It seems as though Transactional and Navigational Search Queries never needed to be refined or changed because the original query returned exactly what the User was looking for.

Final Remarks

We had a very hard time analyzing Web Sites that hide their HTML Source Code. This meant that some Websites showed little to none TF-IDF calculations because they had very little words to compare and analyze. Because of our Web Scraping limitations and Synset refinement weariness, I believe that our Application did not successfully refine Web Search Queries.

References

1. O. Hoeber and X. D. Yang, Evaluating the effectiveness of term frequency histograms for supporting interactive Web search tasks, In Proceedings of the ACM Conference on Designing Interactive Systems, pp. 360-368, 2008

2. O. Hoeber and X. D. Yang, Evaluating the effectiveness of term frequency histograms for supporting interactive Web search tasks, In Proceedings of the ACM Conference on Designing Interactive Systems, pp. 360-368, 2008

3. NLTK Toolkit. *Natural Language Toolkit.* https://www.nltk.org/. Accessed on 4/1/2021.

4. Rahul, S. (2021). MULTIMEDIA INFORMATION SYSTEMS. *MULTIMEDIA INFORMATION SYSTEMS*, 29–80. https://ilearn.sfsu.edu/ay2021/pluginfile.php/1068666/mod_resource/content/1/MULTIMEDIA-LectureNotes-Class.pdf

5. *Create a Python Web Server - Python Tutorial*. (2020). Python. https://pythonbasics.org/webserver/

6. Otto, M. J. T. (2021). *Bootstrap*. Bootstrap. https://getbootstrap.com/

7. A R, B., Joshi, A., & Bhattacharyya, P. (2012). *Cost and Benefit of Using WordNet Senses for Sentiment Analysis*. Cost and Benefit of Using WordNet Senses for Sentiment Analysis. https://www.cse.iitb.ac.in/~pb/papers/lrec12-cost-benefit.pdf.

8. Managoli, G. (2020, August 7). *An advanced guide to NLP analysis with Python and NLTK*. Opensource.com. https://opensource.com/article/20/8/nlp-python-nltk.