INCREASING THE EFFICIENCY OF TEXT ANALYSIS PROGRAMS USING A MULTI-TIERED APPROACH

Daniel O'Brien

Prof. Michael Gildein, Faculty Advisor

Spring 2020

Submitted to the Honors Council in partial fulfillment of the requirements for the degree of Honors in Liberal Arts

Table of Contents

Abstract

Natural Language Processing is an increasingly researched field that has a few major barricades holding back its progress. One of these barricades is the intensive computational costs used by its processes. The goal for this project is to explore potential efficiencies that can be gained by implementing a multi-tiered filter based analysis system. A text-analysis script was written and designed in order to prove this hypothesis. The resulting data indicates that a multi-tier approach can be successfully implemented for an increase in efficiency.

I. Introduction

i. Why Computational Efficiency is Essential

Natural Language Processing (NLP) is an increasingly researched field that has a few major barricades holding back its progress. One of these barricades is the intensive memory and computational costs used by its processes. Enterprise-level systems designed to perform any sort of meaningful text-based analysis must do so in a resource-efficient fashion. Useful text-analysis projects get pushed aside because they lack cost efficiency. The goal for this project is to explore potential efficiencies that can be gained by implementing a multi-tiered filter based analysis system. This research aims to explore the possibility of trimming data sets using a multi-tiered filtering system to find efficiency gains.

Companies have realized the economic value of data, and are scrambling to acquire, store, and analyze it in large quantities. With companies buying mass amounts of data, even through economic uncertainty, data is being stored and shared at a rate never seen before. This can be seen in companies such as Facebook and Google who have leveraged the collection of data to be their main source of revenue. The term “Big Data” was coined to represent this new focus on data as a vital economic input [1].

"Big data" is a major buzzword in the information technology industry that gets defined in many different ways. We will define it as an umbrella term that covers all technologies and methodologies used for storing and utilizing large volumes of data. The inclusion of large volumes of data is an essential, widely accepted attribute of Big Data. The definition of "large volumes of data" is debated as well, but can be characterized as any data that are too large for traditional storing and analytical processes [2]. Next, many definitions include velocity of analysis as an essential Big Data attribute. This ties back into the large volumes of data, as analytical programs must be efficient enough to provide results in a reasonable time frame.

Companies are leveraging Big Data analysis to predict future events based on past occurrences. This is done by gathering data points, determining their relationship, and using that relationship to infer target results. This can be thought of as taking two or more data points and drawing a line through them that best fits their relationship. The less data points present, the more ambiguous the line becomes. By having larger sets of data, this line becomes more accurately defined and less effected by biases and outlier data. As such, companies who can afford it are purchasing incredibly large data sets. These data sets are then handed off to data scientists, who are left to solve the problem of efficiency. However, plenty of analytics tools get developed without the consideration of optimal data transformation and/or analysis [3]. With such large data sources, these inefficiencies render a program ineffective. This multi-tiered approach used offers the potential efficiency gains that Big Data applications are reliant on.

ii. The Value of a Multi-Tiered Approach

Increasing the efficiency of computer programs has always been a fundamental part of the information technology industry. These boosts in efficiencies have been found through clever decisions about how data gets processed. Oftentimes, this involves the pre-processing, concurrent

processing, and even the trimming of data sets. Pre-processing is useful for finding efficiency boosts in applications that face large usage spikes but otherwise spend large amounts of time idle, such as electronic voting systems. Concurrent processing is useful in situations where shared data is important, such as when many people are trying to buy a single concert ticket. As explored by this research, trimming data is best utilized for doing analysis against large sets of data or when computation procedures require significant resources.

The multi-tiered filtering process analyzes articles' titles first, and uses the results to decide if the articles are similar enough to justify further analysis. If they are, the program performs a similar check on the articles' sub-titles. Finally, if all of the preliminary tests are passed, the full articles are analyzed. This allows the program to identify dissimilar articles without unnecessarily wasting computational resources. This naturally makes the content of the article titles' and subtitles' the most significant piece of information used in the process. Titles are dense with information, and analyzing them first is the most cost-efficient way to identify dissimilar documents. This is the core process being studied in this research.

iii. Potential Real World Applications

Boosting the computational efficiency of text analysis modules has strong implications for where these modules can be leveraged. One possible application of this system is in University Admission offices. When a student transfers into a college, the college must decide which of the student's credits will transfer. This process is usually done manually, comparing course titles, sub-titles, and descriptions to determine if the course material learned previously was sufficient. A multi-tiered text-analysis approach, as discussed in this paper, would allow universities to efficiently compare the course descriptions and make a guided decision as to whether or not the credits should transfer. This would improve both the speed and accuracy of the decisions made.

Another application for a multi-tiered text-analysis module is in protecting patents. Many companies encourage their employees to identify and file patents. They are then left with the problem of protecting a large volume of patents by identifying and taking action against any possible infringements. This is a tough task, especially with the tricky legalese that surround patent law. By using a multi-tiered approach, patents can be compared to product descriptions allowing companies to sift through huge volumes of information quickly and efficiently. Any product descriptions that are identified as similar to a patent can be handed off to lawyers, while dissimilar products are discarded without need for further analysis. This creates a huge boost in both computational and organizational efficiency, which allows a company to be more vigilant when enforcing their patents.

II. The Script

i. Development Environment

The script created to conduct this research was written in Python. This decision was made due to Python's lively data science community, and the countless open-source Python packages that can be used to perform data analysis. Also, Python is a versatile programming language that allowed the script to be structured in an easily modifiable way. This versatility was priceless for gathering data and shifting focus throughout development. The packages used in this script for data acquisition, storage, manipulation, and analysis include BeautifulSoup, pandas, Doc2Vec, nltk, and sklearn.

The script was developed in a Jupyter notebook development environment. Jupyter notebook is a web-based Integrated Development Environment (IDE) that is most popularly used for creating analytical applications. The most useful component of the IDE is its cell-based execution. These cells are partitions of code that are written, executed, and tested separately, but

still share a global scope. With this methodology in place, certain functions only needed to be ran once per session, and their outputs were used multiple times in different contexts. This shortened the development time of the script considerably as less time was spent waiting for computationally expensive functions to run.

Leveraging Jupyter Notebook decreased development time and was essential to keeping this project in scope. As discussed earlier, analysis on large sets of data is incredibly resource intensive. This script was written on a personal computer with a 1.6 GHz intel i5 processor and 8 GB of Random Access Memory (RAM), which is obviously not ideal. With the limited computationally resources available when designing the tool, any time wasted re-executing functions would have been fatal to the project's success. An example of how the IDE was leveraged to reduce development time was the creation of the corpora. This function takes rather long to run and is bottlenecked by network speed. Using Jupyter Notebook, it only needs to be run once per session. The analysis algorithm could then be designed around the corpora without having to repeatedly refill them.

The source code manager of choice for this project was GitHub. This decision was made to maximize convenience and visibility. Leveraging version control is crucial in writing clean functional code. It allows for code to be written without fear of breaking anything essential, since any past version can be referenced and reverted to. Another advantage found with GitHub was an increase in transparency. Bite sized commits enabled discussion between my mentor and I throughout the development of the script. Code commits for each specific feature could be used as a reference, promoting a greater level of collaboration between us. Also, Since GitHub is the most popular version control platform with about 14 million users as of March 2017 [4], anyone interested in inspecting the script itself is likely familiar enough with the platform to do so.

ii. Python Packages

One of the most important packages used in this module is Doc2Vec. Doc2Vec is an algorithm that generates vectors for phrases, paragraphs, and documents. These vectors can then be compared against other vectors, and the difference between them can be used to determine the similarity of the documents they were created from. Since the script only uses this one method for its analysis, the entirety of the research relies on the accuracy of Doc2Vec. Using a different analysis algorithm would greatly change the outcome of the results, even if the same dataset was used. Therefore, all results found by this research should be considered in the context of Doc2Vec based analysis.

Another package that was considered for performing text-analysis in the script was Word2Vec. Word2Vec works very similarly to Doc2Vec, except is built to be used for analyzing short phrases. It is more accurate in doing so, and thus was enticing for the analysis of the documents' titles and subtitles. However, there were a couple of considerations that held back its implementation. One of these considerations was the variable length of the titles. Some titles were too long to be accurately summed into a Word2Vec vector, and the variety of title lengths made it hard to account for all possible cases. The second consideration made was Word2Vec ability (or lack thereof) to handle full length document analysis. This meant that the use of Doc2Vec would still have been necessary, and two separate models would have had to have been trained and stored. Since training and storing another whole model is incredibly resource intensive, the decision was made that the costs of using Word2Vec outweighed its benefits.

Doc2Vec was chosen primarily for its ability to recognize semantics, as well as its flexibility. By generating word vectors, the context becomes more important than the specific words used. This is important because of the complexities found in natural language. For example,

if one document speaks heavily on banks and stock markets, while another describes the board game Monopoly, these documents are probably not very similar. However, if we only take the specific words used into account, then the analysis may show that they have a lot in common. This is unacceptable, and many such false positives would arise with this approach. Doc2Vec helps alleviate this problem by considering the context of the entire document. It can recognize that the banks, mortgages, and loans are often used in different contexts, even though they show up often in both documents. This ability to handle context relies heavily on the model built for the analysis, and so the goals of which a model is built for greatly effects how accurately the context of a document can be understood.

iii. Building the Corpora and Doc2Vec Model

The model used in this script was built using Wikipedia articles in the Technology industry. This was done for the purpose of maintaining scope and ensuring accuracy in the data gathered. Building a model is computationally expensive, and the resulting file size is always quite large. This means that training the model for a more general or complete data set was not feasible. More importantly, training the model on a more general data set would have negatively affected the accuracy of its analysis. For example, if this module were to be used to identify patent infringements in the technology sector, it would only need to be semantically aware of Technology based terminology. However, if it was trained on a broader set of data, its understanding of the information technology industry would suffer. This would likely cause an increase in the number of false negatives produced by the analysis. By focusing on one topic to build the model off of, we can maintain analytical accuracy for the multi-tiered system while maintaining a reasonable size and runtime for the script.

The decision to use Wikipedia articles for building the corpus and training the model was made due to its widespread availability and abundance of data. There are countless Wikipedia articles available online, each with a "See Also" section filled with related articles. This made building topic-specialized corpora simple and easy, as I could place my trust in the similar article algorithms used by Wikipedia. It also simplified building *random* corpora, as they have their own "random page" function. Since almost everything in existence today has its own wiki article, the *random* corpora built contain a vast range of articles. Another reason Wikipedia was chosen is because anyone with internet access can gain immediate access to the articles being used. This gives readers of this paper the option to look up articles used and see if they are really as similar (or dissimilar) as the script says. This increases the transparency of the experiment while acknowledging the potential existence of inaccuracies that may come as result of the script's analysis.

Once the model is built and the document vectors are prepared, the script uses the cosine similarity analysis formula to compare the documents. Simply put, the similarity of the documents is determined by taking the cosine of the angle between the vectors. This approach allows documents to be analyzed according to their semantic orientation, as opposed to approaches that overstate the significance of specific words that appear (like the Monopoly example provided earlier). This prioritization of context helps reduce the number of false-negatives and positives produced by the script, increasing the accuracy and validity of its results.

Each article undergoes pre-processing before being analyzed. This process is necessary for performing the analysis and increases the accuracy of results as well. The first component of pre-processing a document is tokenizing its words. This is the process of splitting words by whitespace and removing punctuation, and is done here with the python package nltk (natural language

tokenizer). The second component of pre-processing the data is filtering out frequently found words. First, “stop words” such as “the”, “and”, “but”, etc. are removed. These words provide little semantic value and are found frequently in every article, so considering them in the analysis would result in an increase of false-positives. The next step is filtering out the most frequently found words in the corpus. In this script, the top three most frequently found words are removed from the corpus entirely. This ensures that the value of other words isn’t diminished by one or two extremely common words. Finally, the resulting text is stored in a list variable to be handed off to the analysis portion of the script.

The corpora that are populated for this research were designed to mimic potential real world use-cases while accounting for some possible biases of the multi-tiered system. To do this, three types of corpora were defined. The First is built using only articles related to the Information Technology industry, starting with one of three possible article seeds. Whichever seed gets chosen is then used to build a corpus of related articles. This is done by grabbing the Uniform Resource Locators (URLs) located in the seed article’s “See Also” page. The process repeats itself on each article added until the specified sample size is reached. This is the *related* corpus. The next type of corpus gets built using Wikipedia’s random article function, and is called the *random* corpus. Finally, a third corpus is built using both of these methodologies, and is called the *fifty fifty* corpus. Half of the corpus is filled by using one of the three seeds along with its related articles, and the other half is filled using Wikipedia’s random article function. These corpora can be used to represent the script being run in best, average, and worst case scenarios.

These three corpora are core to the analysis being done, so they were designed to offset each other’s biases. The *related* corpus is the main focus, and is dense with articles relating to information technology. This corpus is what we use to build the model. This decision was made

with real world use cases in mind. Using the patent infringement example from before, it makes sense to train the script to be used for a specific industry. However, the program should still be reasonably capable of analyzing documents it was directly trained for. Otherwise, the multi-tiered approach will not be able to appropriately filter all of the articles it encounters. Any inconsistencies resulting from this directly affects the effectiveness of the multi-tiered approach, and threatens the validity of the data gathered in this research. The *fifty fifty* and *random* corpora are used to test against this bias to see if the program can operate successfully with a broad variety of data. The data gained from analyzing these corpora will determine the consistency and validity of the program, while providing greater context for data found using the *related* corpus.

iv. Recording the Results

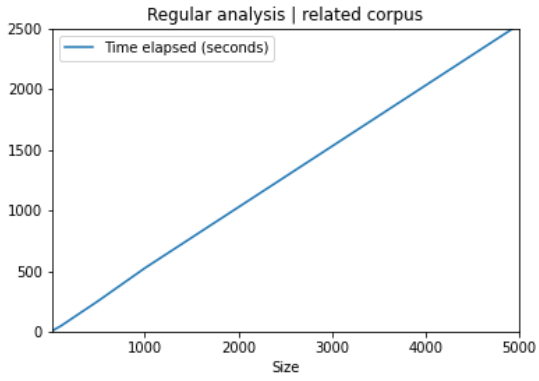
While there are many data points that get recorded by this script, the most important two are sample size and time elapsed. The time taken to perform a regular text-analysis process is recorded for each corpus, as well as the time taken to perform the analysis with the multi-tiered system enabled. This data is then exported to a comma-separated value (CSV), and is the main dataset used for finding patterns of efficiency boosts within the script. After each corpus is analyzed, another CSV is exported containing the similarity results for each article, including whether or not it passed each tier. This data was recorded with the hope of identifying any possible biases of the similarity analysis algorithm. Analysis is done on this data to see how many articles were filtered out by the tier system, and how that effected the total run time of the method. This data can also be found in the project's GitHub repository and can be inspected by anyone interested in verifying the accuracy of the similarity analysis portion of the script.

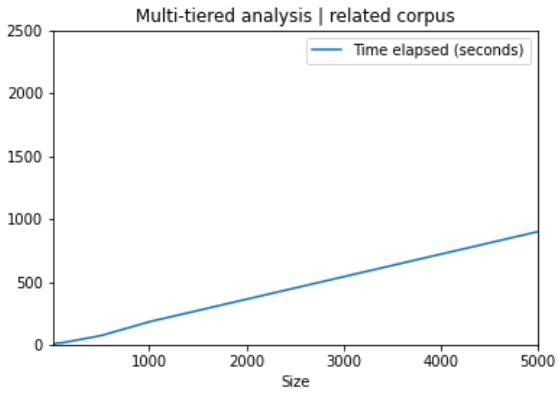
The Python package Pandas was used for formatting, storing, and manipulating data in the script. Pandas is an incredibly popular data analysis package used for its speed and flexibility.

Here, it is used to neatly record the text-analysis results output them to CSV files. The CSV files are then uploaded to the GitHub repository, increasing the transparency of the results found in this project. Pandas was also used plot the output data produced by the script. This was where Pandas assisted with the rapid development to this project. It's methods for filtering, analyzing, and visualizing data allowed me to explore the data at hand and find its most valuable insights. These methods helped create a sense of context for which the data can be discussed.

III. Results

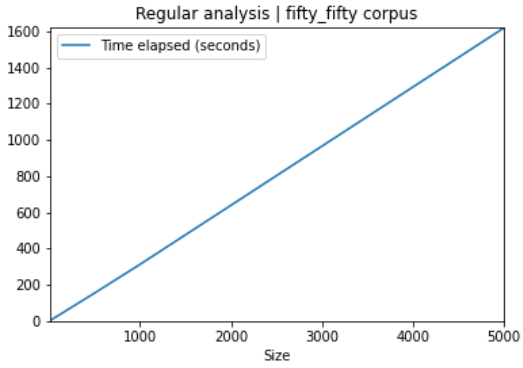
First we will look at the results of running the script on an *all related* corpus.

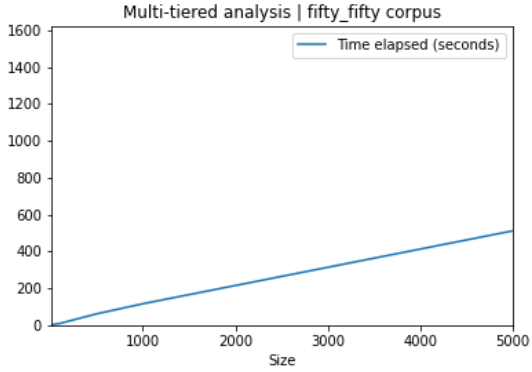




The two line graphs shown above depict the amount of time the analysis takes to run for each sample size. The left graph shows the script run time without the multi-tiered system, and the right graph shows the run time with the Multi-tiered system enabled. We can see that the graph depicting the multi-tiered system is flattened by a decent margin. With a sample size of 5000 points, the multi-tiered system runs 2.8x faster than the regular analysis. This can be considered a success for the multi-tiered approach, especially since many of the articles in the *related* corpus make it through the filtering process.

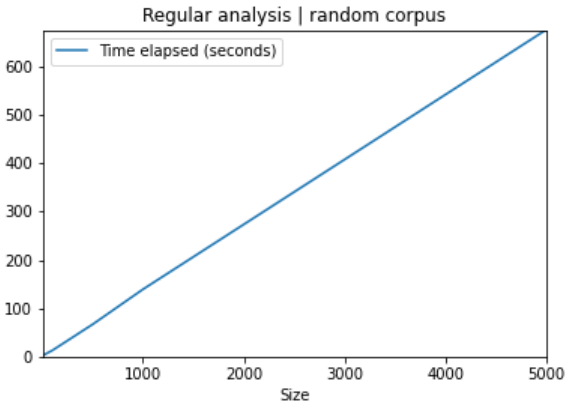
Next we will look at the results of running the script on a *fifty fifty* corpus.

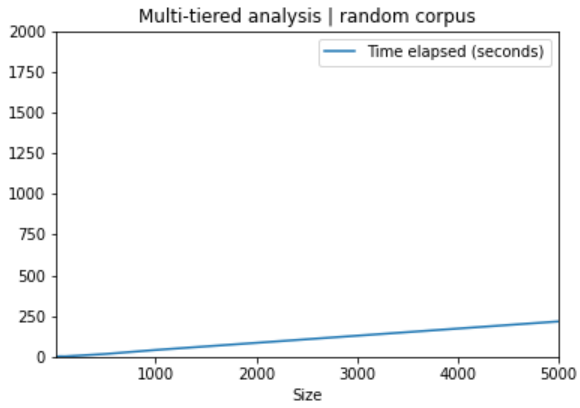




Again, the two line graphs above depict the amount of time it took the analysis to run for each sample size, with the left graph illustrating a regular approach and the right graph illustrating the multi-tiered approach. The results are very similar to the results of the *related* corpus. Here, the disparity between the two graphs is slightly larger. With a sample size of 5000, the multi-tiered approach runs 3.5x faster than the regular analysis. The slight increase in efficiency found using this corpus as opposed to the *related* corpus is a result of more articles being filtered out by the multi-tiered system. This further validates the success of the approach.

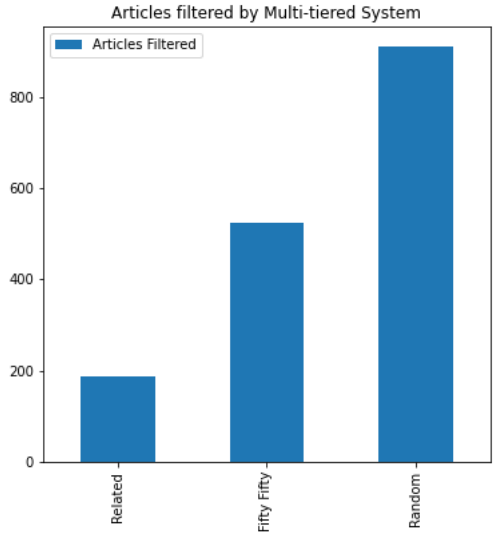
Finally, we will look at the results of running the script on a *random* corpus.





The average *random* article is shorter than the articles found in the other corpora. This causes the total run time to be lower for both types of analysis. Regardless, the difference in the time taken to run each script is significant. At a sample size of 5000, the multi-tiered system runs 3.1x faster than the regular analysis. This is to be expected, as a random corpus would result in more articles are identified by the multi-tiered system as dissimilar. With less articles passing the filtering process, less articles are fully processed and more time is saved. This is a testament to the system's ability to gain efficiency by ignoring valueless documents.

In an effort to further validate these findings, we will take a look at the number of articles that were filtered out in each corpus.



The bar graph above depicts the number of articles filtered out by the multi-tiered system for each corpus at a sample size of 1000. The results found here are as hypothesized. The small but meaningful number of filtered articles for the *related* corpus were responsible for the small boosts in efficiency that was found. Predictably, almost half of the *fifty fifty* corpus was filtered out by the script. This resulted in a more noticeable increase in efficiency being found. Finally, the random corpus was almost entirely filtered out. This makes sense, since most articles in the corpus

were in no way related to the information technology-related seed articles. This incredibly large number of filtered articles led to the considerable efficiency boost found when running the script against the *random* corpus.

IV. The Conclusion

This data shows that it is possible to boost efficiency in natural language processing applications using a multi-tiered approach. It also shows that this approach works on broad sets of documents that span multiple different topics. The process proves to be effective in the best, average, and worst case scenario. This means that the approach has a wide range of potential applications in the Data Science Industry. Considerations should be made to implement a similar system in any application involved in natural language processing and text similarity analysis.

IV. Appendix

i. Works Cited

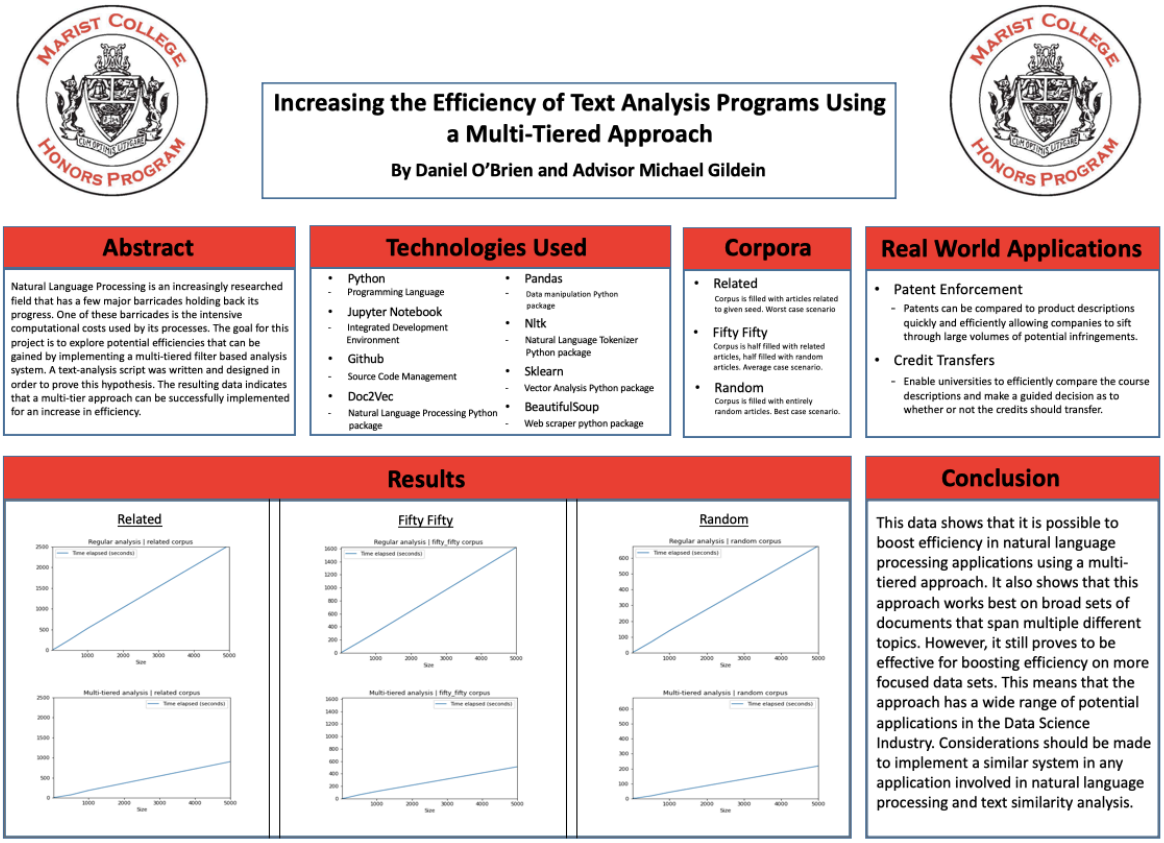
[1] C. Pentzold, C. Brantner, and L. Fölsche, "Imagining big data: Illustrations of 'big data' in US news articles", *New Media & Society, 2010-2016.* [Online]. Available: https://doi.org/10.1177/1461444818791326. [Accessed April 20, 2020]

[2] C. Pentzold, C. Brantner, and L. Fölsche, "Imagining big data: Illustrations of 'big data' in US news articles", *New Media & Society, 2010-2016.* [Online] . Available: https://doi.org/10.1177/1461444818791326. [Accessed April 20, 2020]

[3] R. Venkatraman, and S. Venkatraman, "Big Data Infrastructure, Data Visualisation and Challenges", *Association for Computing Machinery*, 2019. [Online]. Available: https://doi-org.online.library.marist.edu/10.1145/3361758.3361768. [Accessed April 23, 2020]

[4] A. Sharma, F. Ferdian, P. S. Kochhar, A. Sulistya, and D. Lo, "Cataloging GitHub Repositories", *Association for Computing Machinery*, 2017. [Online]. Available: https://doi-org.online.library.marist.edu/10.1145/3084226.3084287. [Accessed April 23, 2020]

ii. Poster Board



iii. GitHub Repository

https://github.com/danielobrien3/HBC-Text-Similarity-Analysis

iv. Resources Used

Pandas: https://pypi.org/project/pandas/

Doc2Vec (Gensim): https://pypi.org/project/gensim/

NLTK: https://pypi.org/project/nltk/

BeautifulSoup: https://pypi.org/project/beautifulsoup4/

argparse: https://pypi.org/project/argparse/

Jupyter Notebook: https://jupyter.org