Daniel O’Brien

Multi-tiered text analysis

Text analysis is an increasingly researched field that has just a few barricades holding back its progress. One of these barricades is the intensive computational costs used by its processes. Enterprise-level systems are required to perform any sort of meaningful text-based analysis in a timely fashion. This means that useful text-analysis projects get pushed aside because they lack cost efficiency. The goal for this project is to explore potential efficiency gains that can be found by implementing a multi-tiered check-based analysis system.

Searching for efficiency gains in computer programs is not a matter of saving a few seconds for convenience sake. Companies have realized the economic value of data, and are scrambling to acquire, store, and analyze it to increase profits. The term “Big Data” was coined to represent this new focus on treating data as a vital economic input [1]. With companies buying mass amounts of data, even through economic uncertainty [2], data is being stored and shared at a rate never seen before.

“Big data” is a major tech buzzword that gets defined in many different ways. Here, 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 seen as any data that are too large for traditional storing and analytical processes [4]. Next, many definitions include velocity of analysis as an essential Big Data attribute. This ties back into the large volumes of data, as the analytical programs used 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 that line becomes. By having larger sets of data, this line becomes more accurately defined as it is less effected by biases and outlier data. As such, Companies who can afford it are purchasing incredibly large data sets and handing them off to data scientists to analyze. These Data Scientists are then left with the problem of efficiency. However, there are many data analytics tools that do not make us of optimal data transformation or efficient analysis [3]. With such large data sources, these inefficiencies render a program useless. While the program used in this research is not necessarily a part of “Big Data”, the multi-tiered approach used still offers the potential efficiency gains that Big Data applications are reliant on.

Increasing efficiency of computer programs has always been a fundamental part of the Computer Science discipline. Optimizations are made through clever decisions about how data gets processed. Oftentimes, this involves the pre-processing, concurrent processing, and cleaning of data sets. Pre-processing is useful for 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, but it can also be used for programs that are run on large scale servers with plenty of resources available. Trimming data (the basis of this research) is best utilized when doing analysis against large sets of data.

This project explores potential efficiency gains for text-analysis tools by utilizing multi-tiered systems to trim data sets. Here, article titles are compared before any further analysis is done. The program then decides, based on the title, if the articles are similar enough to proceed with the analysis. If they are similar enough, the program then uses the articles’ sub-titles for the next tier of analysis. Finally, passing all of the preliminary tiers of analysis, the full articles are analyzed for similarity. This allows the program to identify dissimilar articles using minimal computational resources.

Boosting the computational efficiency of text-analysis modules has strong implications for where these modules can be leveraged. One possible application of this system can be found in University credit transfers. When a student transfers colleges, the school they are transferring into must decide which of the student’s credits will transfer. This process involves manually comparing course titles, sub-titles, and descriptions to decide if the proper material was covered. Using a multi-tiered approach as discussed in this paper would allow universities to q efficiently compare the course descriptions submitted by its transfer students. The results can then be used as a guide to help improve speed and accuracy for comparing course descriptions.

Another possible application for a multi-tiered text-analysis module is looking for patent infringements. Many companies encourage their employees to identify and file patents. These companies are then left with large amounts of patents that they must make sure are not being infringed upon. This is a tough task, especially with the tricky legalese that protect certain patents and doom others. By using a multi-tiered approach when comparing patents to product descriptions, companies can sift through tons of information without an unnecessary waste of man power. The product descriptions that make it past the analysis tiers can be handed off to lawyers to be manually inspected, while the dissimilar products are discarded without further need for analysis. This creates a huge boost in both computational and organizational efficiency, and would allow a company to be more vigilant when enforcing their patents.

The module created to conduct this research was made using Python. Python was chosen for its immense data science community and countless open-source packages that can be leveraged to perform data analysis. Also, python is a very versatile programming language. This allowed the program to be structured in a way that is easy to modify and maintain. This versatility was priceless for data gathering. This module utilizes packages for the acquisition, storage, manipulation, and analysis of data, as well as the handling of inputs and outputs from the program. These packages include BeautifulSoup, pandas, Doc2Vec, nltk, and sklearn.

The module was created in a Jupyter notebook development environment. For the unaware, Jupyter notebook is a web-based Integrated Development Environment (IDE) that is most popularly used for creating analytical applications. The most critical component of the IDE is its use of cells to structure a program. These cells allow code to be partitioned into sections that are written, executed, and tested separately. However, the global scope of the program is still maintained. With this methodology in place, certain functions only need to be ran once per session, and their outputs can be used over and over again in different contexts without having to worry about crazy file management schemes. Development time was shortened considerably as less time was spent waiting for computationally expensive functions to run.

Using Jupyter Notebook to lower development time was essential to keep this project in scope. As discussed earlier, analysis on large sets of data is incredibly resource intensive. This tool was written and tested on a personal computer with a 1.6 GHz intel i5 processor. One example that illustrates how the IDE can be leveraged is found in the function responsible for fulling the corpora. There are two factors that affect the time it takes to fill the corpora. The first is computational resources, which is rather obvious. The second factor, network speed, is far more important here. With the limited computationally resources available when designing and testing the tool, wasting time by unnecessarily re-executing functions would have been fatal to the project’s success.

One of the most important packages used in this module is Doc2Vec. Doc2Vec is an algorithm that is used to generate vectors for phrases, paragraphs, and documents. These vectors can then be compared against vectors created from other documents, and the difference between the two can be used to determine the similarity of the documents. Since the module 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.

Doc2Vec was chosen for its focus on semantics, as well as its flexibility. The vectors are generated by considering the context in which they are used, and not just the specific words used. This means the vectors are semantic representations of the document they were generated from. 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 speaks heavily on financial institutions, these documents should be considered similar. However, if we only take the specific words used into account, then the analysis may show that they have little in common. This is unacceptable, as many false negatives would arise. Doc2Vec helps alleviate this problem by considering the context of the entire document. It can recognize that bank and financial institutions are often used in the same context, even if only a few words are shared between the documents. However, this relies heavily on the model built for the doc2vec analysis, and it can only perform accurately in contexts it was properly prepared for.

With the document vectors prepared, the module utilizes cosine analysis to determine their similarity. This means that 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 a bag-of-words approach that gives more significance to the specific words that appear. False-negatives and false-positives are avoided as a result of the program prioritizing context over content. This helps make the program far more robust.

Another package that was considered for performing the text-analysis was word2vec. Word2vec works very similarly to doc2vec, but is built to be used on short phrases. This made word2vec seem like the best option for analyzing the title and subtitles of each document. Even though word2vec would have been more accurate than doc2vec in analyzing article titles, there were a few considerations that held word2vec back. One of these considerations was the variable length of the titles. Some titles were too long to be accurately summed into a vector, and the variety of title lengths made it hard to account for all possible cases. The second consideration found with word2vec was that it could not handle full document length analysis. This meant that doc2vec model was still necessary, and training, storing, and using two models is incredibly resource heavy. As a result, word2vec did not provide enough value for the resources it required to run.

The doc2vec model for this module 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 meant that training the model for a more general data set was not feasible. Also, for most applications of this module, a general model would not be needed. 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. By focusing on one topic to build the model off of, we can maintain the accuracy of the tool while keeping a reasonable size and runtime for the module.

The decision to use Wikipedia articles for building the corpus was made because of its widespread availability and use. There are countless Wikipedia articles available online, each with a “See Also” section filled with related articles that made building topic-specialized corpora simple and easy. It also simplifies building random corpora, as almost everything in existence today has its own wiki article. Another reason Wikipedia was chosen is because anyone with internet access can gain immediate access to the articles being used. This gives the reader of this paper the option to look up articles used and see if they are really as similar (or dissimilar) as the tool says. This increases the transparency of the experiment by accounting for inaccuracies that may result from the similarity analysis portion of the tool.

The corpora that are analyzed for this research were designed to mimic potential real world use-cases while also accounting for possible biases. To do this, three types of corpora were defined. First, one of three starting seeds is chosen. These seeds, all related to the Information Technology Industry, are then used to build a corpus using related articles. This is done by grabbing the 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 called the *all related* corpus. The next type of corpus is entirely *random*, and gets built using Wikipedia’s random article function until the specified sample size is reached. Finally, a third corpus, named *fifty-fifty* is built using both of these methodologies. Half of the corpus is filled by using the seed and its related articles, and the other half is filled using the random article function.

These three corpora are core to the analysis being done, so they were designed to offset each other’s biases. The *all related* corpus is incredibly dense with articles relating to the information technology industry. By using this corpus to build the model, the program becomes optimized for dealing with articles related to information technology. This decision was made with real world use cases in mind. Using the patent infringement identification example from before, this program would work best under the assumption that it will be used for a specific industry that the model was built to understand.

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 robustness of the program, and serves to test the validity of the data found using the *all related* corpus.