**Final Project**

For this project will discuss an implementation of the Text Rank algorithm to summarize blocks of text in as few sentences as possible. It is typically used to find keywords or to reduce the size of long passages by omitting less relevant information. The text rank algorithm essentially evaluates all words or sentences in a corpus of text and determines what abstract concepts are emphasized the most in the text. It then ranks the words or sentences by how closely they align with said concepts.

**Overview of the process**

To accomplish a text ranking, the algorithm first encodes all preprocessed words in a corpus into word vectors and uses these vector representations to find how closely related each word or sentence is to each other word or sentence. When the goal is to omit irrelevant sentences from text, each sentence is summarized as a single vector by some aggregate value of the word vectors representing each word them. Usually, the “cosine similarity” between both words’ representative vectors is used to determine how closely each word or sentence is related. These similarity scores are put into matrix under rows and columns corresponding to the vectors being compared. From there it is a simple matter of generating a complete graph that connects each word or sentence to each other word or sentence with weights corresponding to their similarity scores and then running the page rank algorithm find which sentences tie into the rest of the text to most, that is, which words or sentences are most central to the text.

The rest of the paper will, firstly, explain the steps taken to follow the process outlined in the original paper that introduced the Text Rank algorithm (Mihalcea and Tarau) and, secondly, attempt to explain the reasoning behind said process.

**Preprocessing**

As with any process that is intended for use on a heterogenous set of data, a step of preprocessing must be done to eliminate the elements of the data that are irrelevant or reduce the power of the conclusions. Depending on the goals of a study in natural language processing, punctuation, citations, stop words, etc. may be excluded from a corpus as they do not contribute any meaning to a text and act only to connect the substantial content of the text in a purely mechanical way. These words or tokens can reduce the efficiency of processing and introduce slight bias in a study. Once the raw text has been cleaned it is tokenized to allow for easy processing. When the words of the text are tokenized, they can easily be matched with a corresponding vector in a file and substituted in analysis.

The Python Library, NLTK or Natural Language Tool Kit, and Python’s re library were used to clean and tokenize the text examined in this paper. Links, emojis, and all ascii symbols besides numbers and letters were excised from the original text before tokenization. The end of sentence punctuation marks (?!.) were used to detect the limits of sentences, therefore there was no need to address them in clean\_text as they were not going to be in the body of the sentence.

Text

Description automatically generated

**Word Vectorizing**

The word vector model assumes that all words are made of a collection of attributes represented by dimensions in Euclidean space. When researchers seek to evaluate these dimensions, they typically use one of a range of techniques that relies on the assumption that words that are related or have similar meanings are typically found in proximity with one another in text. With this assumption in mind, machine learning algorithms are used to compile these vectors by crawling through of a relevant set of data and iteratively adjust each words dimensions until they converge.

An analogy often used to explain word vectors is *queen is to king as woman is to man* or rather *queen* - *king = woman – man* and its implication *queen – woman + man = king.* Both words can intuitively be broken into attributes corresponding to gender and political power. As the above example explains, if one takes the woman out of queen they are left with monarch. If one further adds man to monarch, they are left with king. One can find interesting results by taking the implications of this model of classifying words by comparing a set of word vectors to common analogies.

The Glove machine learning algorithm is one of the most popular, contemporary methods for training word vectors. In service of finding interesting results in limited time and with limited resources, a pretrained set of word vectors, provided by the Standford’s website for the Glove machine learning website, was used for this project.

For each body of text examined, a nested for loop was used to compare each word in a vector file with each word in a text. Each corpus was given a separate vector file containing its limited vocabulary to make vectorizing the sentences more efficient.

Text

Description automatically generated

vectorLoader.loadVocab() from Preprocess.py

Once each body of text was given its own vector file, a dictionary containing the mappings of each word to its vector was generated. Then, the text would be broken into separate sentences and each sentence would be evaluated by taking the sum of each word in the sentences. Since cosine similarity was to be used to evaluate the relationships between the sentences, the magnitudes of the sentence-vector sums would be factored out meaning that the varying lengths of the sentences would not affect the results.

Text

Description automatically generated

**Building the Graph**

Once the sentences were vectorized, their similarity scores had to be put in a relationship matrix. The similarity score used was “cosine similarity” as it was presented by Mihalcea and Tarau as a viable score for similarity between two vectors. It was calculated using dot product of the compared vectors divided by the product of both vectors’ magnitudes.

Text

Description automatically generated

Formula for Cosine Similarity retrieved from: https://www.sciencedirect.com/topics/computer-science/cosine-similarity

First, the content of each sentence was loaded in as an attribute of each sentence node. Then, a csv file was written to contain a representation of the sentence relationship matrix. Only the spaces below the top-left bottom-right diagonal were used as the rest of the measurements would be copies of their reflections in the 2-dimensional set of combinations. To make importing to Neo4j as simple as possible, the csv only had two columns: one to contain the index of their sentences and the other to contain a string with the comma-delimited mappings of each other sentence to its similarity relative to the sentence in question.

Text

Description automatically generated

Tabulate Function from “Preprocess.py”

Table

Description automatically generated with medium confidence

Example of a Resultant Table

After the matrix was encoded into the csv format, it was imported into Neo4j using the following merge script:

Graphical user interface, text, application, email

Description automatically generated

**Page Ranking the Text**

The page rank algorithm is famous for assisting the google search engine for ranking web pages by importance, allowing it to prioritize the most credible and important information for the luxury of their consumers. Its success was surely not overlooked as variations of the said algorithm, including ones for undirected and weighted graphs, have been used for a wide range of machine learning. Text Rank happens to use a variation for weighted graphs that operates on the invariant below: A picture containing text

Description automatically generated

Page rank graphs with weighted relations. (Mihalcea and Tarau)

Though the above formula has functions for counting in and out relations, an undirected graph can blend in by counting its directions twice: once for in and once for out.

The special ingredient that makes page rank useful is its emphasis on not only accounting for the power of nodes in themselves but also for factoring in the power of the nodes it is closest to. As can be seen above, the influence of node is determined by taking the sum of the product of each node each that flows into it and its preference for (calculated by taking the proportion of the weight flowing from edge to ). Once the sum of the terms is found decimal damping ratio is applied to that sum and its compliment is added after that.

A variation of PageRank is intuitively reasonable. As all words, by themselves, cannot be said to be more important than others in the context of all use in language; their importance is always relative to their context not some inherent value. Page rank, on the other hand, treats all nodes as inherently equal and only ranks them by their connections to other nodes within a context. And, of course, the central concepts emerge within contexts because the words are evaluated relative to one another. The entire idea of summarization is finding commonality, something that has meaning in the relationships between words. Because PageRank is a recursive algorithm that reaches to all nodes indirectly connected to each node. This allows for a recommendation effect to occur that makes it so that words that are like many other words to have a greater influence on ones that are not as similar, making connections indirectly forming stark centrality. As summarized by Mihalcea and Tarau: “The sentences that are highly recommended by other sentences in the text are likely to be more informative for the given text and will be therefore given a higher score.”

Fortunately, for the purposes of this project, ranking the vectorized sentences was as simple as projecting a graph and running a single algorithm. The example below was the result for a passage written specifically to test the algorithm. It is important to note that when projecting the graph, the orientation needs to be specified as undirected or else the results will not be congruent with what should be expected from the type of text ranking done in this paper.

Graphical user interface, text, application, email

Description automatically generated

Project statement from neo4j\_script.txt

**Graphical user interface, text, application, email

Description automatically generated**

Example output of the entire process

**Results**

After exporting the results of the page ranking into export.csv and running “results(5)”. Here are the top five sentences (out of 24) from volcano.txt (text retrieved from What if we Drilled into a Supervolcano?)

1. Experts predict that if the supervolcano lurking below the surface of Yellowstone National Park in Wyoming were to fully erupt, the explosion would kill tens of thousands of people immediately and spread a 10-foot (3-meter) layer of molten ash as far as 1,000 miles (1,609 kilometers) [sources: USGS, Lemas].
2. Campi Flegrei sits under a large swath of Naples, the southern Italian city that's nestled against a bay and the supervolcano's better-known cousin, Mount Vesuvius.
3. The lava flow and avalanche of ash from a full explosion could endanger millions of people [sources: ICDP, Starr].
4. The International Continental Scientific Drilling Program, the folks behind the project to drill into Campi Flegrei, claim the project would have been pretty safe and incredibly useful.
5. Opponents say that drilling into a supervolcano is kind of like getting into a cage with a sleeping great white shark and prodding it with a sharp stick.

The result is unexpected. Instead of answering the question: What is a Supervolcano? The top sentences emphasize concerns about the safety of the project and the sheer danger of Campi Flegrei. Even the sentence that mentions Mount Vesuvius’ relation to Campi Flegrei, is right before a description of the death toll of Mount Vesuvius’ eruption in the year 79. This sentence must have been chosen because it contained Campi Flegrei (the subject of the article) and information that is directly related to Mount Flegrei. By the logic of PageRank sentence about Vesuvius’ eruption must have also contributed to the sentence containing it.

Text

Description automatically generated

“results” from Preprocess.py

**References**

TextRank: Bringing Order into Texts

<https://aclanthology.org/W04-3252.pdf>

GloVe: Global Vectors for Word Representation

<https://aclanthology.org/D14-1162.pdf>

GloVe word Vectors dataset:

<https://nlp.stanford.edu/data/glove.6B.zip>

What if we Drilled into a Supervolcano?

https://science.howstuffworks.com/science-vs-myth/what-if/what-if-we-drilled-into-supervolcano.htm#:~:text=A%20supervolcano%20is%20a%20potentially%20explosive%20mountain%20that,like%20a%20bad%20thing%2C%20that%27s%20because%20it%20is.