TF-IDF Vectorization and Comparison Interface of Project Gutenberg's English Literature Database

Robert Zipp
Khoury College of Computer Sciences
Northeastern University
San Jose, CA 95113 zipp.r@northeastern.edu

Abstract

The authors developed an implementation to conduct TF-IDF vectorization of Project Gutenberg English books, which has not been attempted in the public domain until now. This model successfully enables end-users to query the software to compare any two texts within the data set for similarity. The project was successful in developing a working model to compare full-length works of literature and serves as a promising proof-of-concept for use in NLP and the digital humanities.

1. Introduction

Document vectorization is a task in natural language processing, "…the main purpose of [which] is to represent the unique characteristics of the document numerically so that computers can handle such unstructured text data."

Deepset.ai succinctly describes the nature of the vector data structure:

> *"…a vector is a data structure that is similar to a list or an array. For the purpose of input representation, it is simply a succession of values, with the number of values representing the vector's "dimensionality." Vector representations contain information about the qualities of an input object."*

1.1 TF-IDF for text processing

In Term-Frequency/Inverse Document Frequency (TF-IDF) vectorization, the objective of vectorization is to represent words in a document corpus in a way that mathematically summarizes the "relevance" of each word. TF-IDF vectorization is computed where "***t*** is the term (word)" being measured for its commonness, "and ***N*** is the number of documents (d) in the corpus (D)... The denominator is simply the number of documents in which the term, ***t***, appears in."

$$idf(t, D) = \log\left(\frac{N}{count\,(d \in D : t \in d)} \right)$$

TF-IDF in mathematical notation.
Image Source: https://www.capitalone.com/tech/machine-learning/understanding-tf-idf/

In the literature and discourse on document classification and in the field of natural language processing more broadly, algorithms have been developed that are most effective at handling large sets of shorter text pieces, such as Tweets and/or news articles. Algorithms used to process a ML-viable corpus of long documents are more limited in the present day. For example, using the modern processing technique BERT limits one to 512 tokens for processing and analysis. Out of the most popular modern-day frameworks, TF-IDF vectorization is the most well-suited for application to longer documents because of its lack of token limitations and its linear, not exponential, memory and speed costs.

1.2: Overview of dataset: Project Gutenberg

Project Gutenberg is a free repository of long-form open-access eBooks available in human and machine-readable formats such as EPUB, HTML5, and Plain Text UTF-8 and includes classic literature by notable authors such as Herman Melville and Jane Austen. The project was started and led by Michael Hart, who also prototyped the modern eBook format in 1971. Likely due to its nature as a volunteer project of mostly digital publishing professionals, it lacks the modern infrastructure necessary to power a real-time API for data science / artificial intelligence projects. The only current direct query system is to set up a private mirror on one's local machine.

1.3: Project purpose

This project seeks to demonstrate a method for processing the Project Gutenberg corpus in a way that allows for use in NLP. This project accomplishes this by using R and Python to extract and process the texts of the corpus into a simple Python dictionary structure. It tests the dataset's viability for NLP work by running it through TF-IDF and offering a command-line interface for an end user to compare the document similarity of different documents in the text, which has never been done before, at least as found in the literature review conducted for this project.

---

2. Method

While Gutenberg does not provide for any direct API connections to its dataset, there exists an open-source package in R named "gutenbergr" that enables one to successfully initiate the processing of Gutenberg texts.

At the time of this project, the library was active in the CRAN repository, but was archived in October 2022 because "check issues were not corrected despite reminders." For the sake of replicability, the author notes that all versions of gutenbergr are still available for access in CRAN's archives.

2.1 Dataset and Data Pre-processing

Before developing a model in Python, it is necessary to extract and preprocess the Gutenberg texts in an R file. In addition to the gutenbergr library, the data was processed with the tidyr and diplyr packages. The output of the gutenbergr library is a set of .csv files which are imported and concatenated in a Jupyter notebook. Only English-language texts were selected for this project.

A major challenge of this project was the size of the data. Without enterprise-grade data storage, the amount of data that could be processed in a reasonable time was limited. Each .csv file was sliced to 120,000 rows in an attempt to widen the data set to more novels.

```
import sys
import csv
csv.field_size_limit(sys.maxsize)
#open subset of book limited to 120000 characters / limit size to preserve
memory
def concat_book(path):
    init_book = ""
    with open(path, newline='') as csvfile:
        book_reader = csv.reader(csvfile, delimiter=' ', quotechar='|')
        for row in book_reader:
                if sys.getsizeof(init_book) < 131072:
                    init_book = init_book + ' '.join(row)
        init_book = init_book[:120000]
        return init_book
```

Figure 1: Code snippet of data processing

After this, the texts were cleaned and tokenized using sklearn library functions. Unfortunately, the data set was still unfeasibly large at this point in the processing, which necessitated the slicing of each data object further into an arbitrary 9,500 char length string. The first 500 characters were sliced from each data object because the first 500 characters of most texts contain metadata & copyright information from Project Gutenberg that would have contributed to significant noise in the vectorization.

2.2 TF-IDF Vectorization

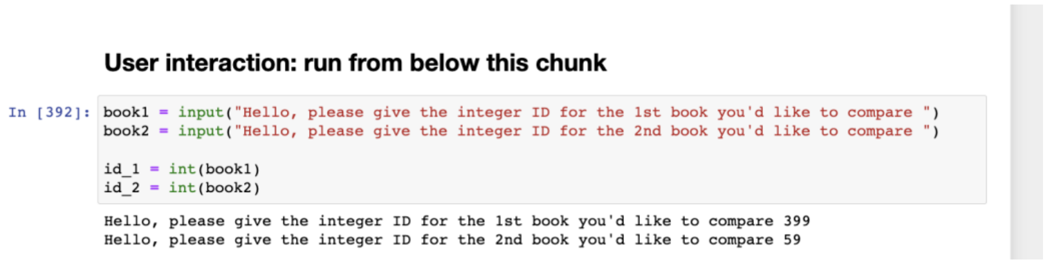
The vectorization implemented in this project was built using a number of functions in alignment with the mandates of the algorithm as described previously in this paper.

2.3 Similarity calculation

There are a number of different similarity functions that can be used to compare two data objects for similarity. Because TF-IDF vectorization is by definition in vector space, Euclidian distance is the optimal choice for similarity calculation between novels. After the TF-IDF values were computed for each data object, the authors use `scipy.spatial.distance.euclidean` to calculate similarity.

3. Results

The program successfully enables a user to calculate the similarity value of two novels in the data set by entering the ID numbers of data objects. These results are detailed below in Figure 3. The scope of this project was limited to comparisons between two data objects, but the architecture of the algorithms means this program can be extended to compare any sample size less than *n*.



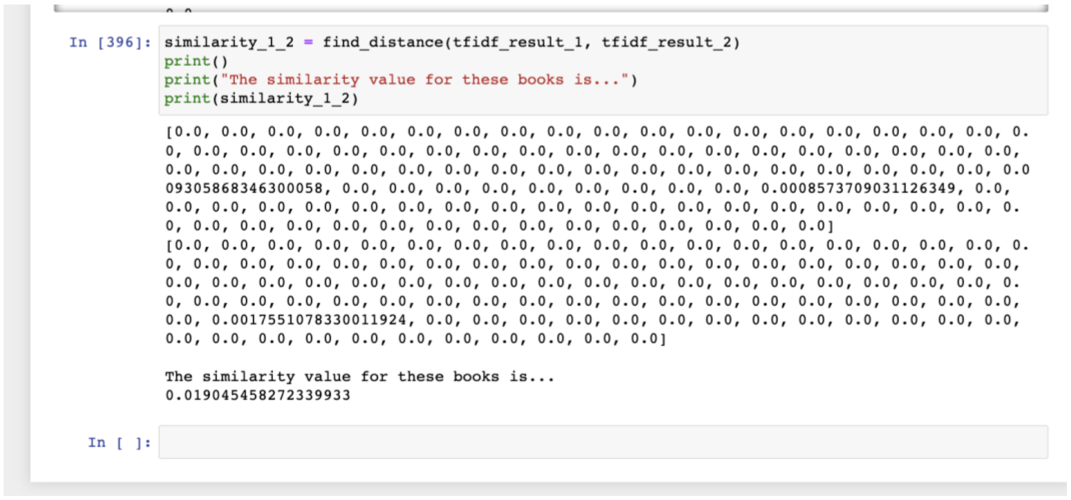
User interaction: run from below this chunk

```
In [392]: book1 = input("Hello, please give the integer ID for the 1st book you'd like to compare ")
          book2 = input("Hello, please give the integer ID for the 2nd book you'd like to compare ")

          id_1 = int(book1)
          id_2 = int(book2)

          Hello, please give the integer ID for the 1st book you'd like to compare 399
          Hello, please give the integer ID for the 2nd book you'd like to compare 59
```

Figure 2: User input of novel ID #s to be used in similarity calculation.



```
In [396]: similarity_1_2 = find_distance(tfidf_result_1, tfidf_result_2)
          print()
          print("The similarity value for these books is...")
          print(similarity_1_2)

          [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
          0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
          0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
          09305868346300058, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0008573709031126349, 0.0,
          0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
          0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
          [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
          0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
          0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
          0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
          0.0, 0.0017551078330011924, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
          0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

          The similarity value for these books is...
          0.019045458272339933

In [ ]:
```

Figure 3: Output of Euclidian distance similarity calculation for two novels in the Gutenberg data set.

4. Discussion

The goal of the project was to attempt to build an interface through which compare classic works of literature on their similarity using NLP techniques. This has not been attempted yet in the literature reviewed for the project. Based on the parameters of the project and the results demonstrated herein, we claim the implementation was a limited success.

The success of this project opens the door for new lines of academic inquiry in the field of English Literature. Most literary criticism rests on the use of "close reading", wherein several lines or pages of a work of literature are analyzed and discussed. However, the creation of tools like this one enable non-technical English scholars to begin to unravel the similarities between entire works in ways that are not feasible for analog rhetorical strategy.

The author hopes this tool will assist in the extension of projects like the one conducted by the Stanford Literary Lab on the subject of the unique characteristics of Jane Austen's vocabulary, when plotted against those of contemporaneous and more recent authors.

5. Conclusion

While the project was successful in a partial capacity, there were a number of limitations that can be addressed in future experimentation.

The use of a Jupyter Notebook environment produced several limitations. The biggest limitation encountered was a lack of data storage and processing power necessary to complete a more comprehensive analysis. The processing of thousands of large .csv files required the authors to wait for days for processing to complete. A cloud data center and/or GPU would have made this project much more reasonable. With that in mind, the authors hope to make use of Google Colab's free GPU resources for future iterations of this project instead of Jupyter Notebook exclusively. Jupyter Notebook's limited data and I/O sizes also presented challenges in the implementation of this project.

Another limitation of this project was caused by the exclusive use of native Python data structures, namely dictionaries, to access and process data. Using a library like pandas or numpy would have enabled for a more robust front-end program that could allow users to query the data by the title of the novel in plain English, instead of by ID number. This is because of the more robust indexing features available in both pandas and numpy.

Acknowledgement

This project was completed for CS6120 under the instruction of Professor Uzair Ahmad.

.

References

https://www.gutenberg.org/about/background/50years.html

https://www.nytimes.com/2017/07/06/upshot/the-word-choices-that-explain-why-jane-austen-endures.html

https://datagraphi.com/blog/post/2021/9/24/comparing-performance-of-a-modern-nlp-framework-bert-vs-a-classical-approach-tf-idf-for-document-classification-with-simple-and-easy-to-understand-code

https://towardsdatascience.com/text-classification-with-nlp-tf-idf-vs-word2vec-vs-bert-41ff868d1794

https://www.kaggle.com/code/naim99/text-classification-tf-idf-vs-word2vec-vs-bert

https://medium.com/@cmukesh8688/tf-idf-vectorizer-scikit-learn-dbc0244a911a

https://monkeylearn.com/blog/what-is-tf-idf/

https://docs.haystack.deepset.ai/docs/retriever#tf-idf