Book-Recommendations Report

Linh Dang

October 2025

1 Summary

This project focuses on making a similar-book recommendation system (demo only) using TF-IDF to calculate the scores of each word appearance and compare them against the topic keywords.

- We used Project Gutenberg, an open-source library, to obtain data. For this project, we made a mini database of 11 classical books and how the TF-IDF (Term Frequency-Inverse Document Frequency) calculation works on them by topic. In addition, we also implemented two nearest neighbor search algorithms and compared them in this project.
- For TF, the highest-frequency words that we discovered are commonly used English words such as "the", "and", "a", and so on. With TF only, the project is insufficient as we wish to categorize books by topics and relatability. Hence, we switched to IDF, which determined more unique words; however, the words are too specific, leading to the topics being undefined as well. Thus, we resorted back to TF-IDF as a whole to determine the sequence of words and how we could group them by topic given that we had buzzwords for each topic.
- For relatability, we used a nearest neighbor algorithm of NumPy and compared it to Faiss nearest neighbor search (Jurafsky & Martin, n.d., ch. 11, p. 13). Both worked well, yet Faiss is more catered to big datasets. Since this is just a demo version of a project, we prioritized the NumPy algorithm more to save memory and time.

2 Introduction

TF-IDF is a popular natural language processing (NLP) technique that quantifies the significance of a word by calculating the frequency and rarity of this word in a database. This project takes TF-IDF and nearest neighbor algorithms to implement a book recommendation system. Our steps include parsing and cleaning the book database, implementing TF-IDF, and finding books similar to a certain one with the aforementioned algorithms.

To avoid common words appearing, TF-IDF will calculate the most frequent words and the ratio to their frequency, resulting in words like "a" or "the" ending up with negative values.

```
tfidf.head(5)
```

term	a	aback	abaft	abaht	abandon	abandoned	abase	abased	abasement	abashed
Alice's Adventures in Wonderland	-0.002072	0.00000	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	0.000000
Beowulf- An Anglo-Saxon Epic Poem	-0.001222	0.00000	0.000000	0.0	0.000000	0.000025	0.0	0.0	0.0	0.000000
Dracula	-0.001682	0.00001	0.000025	0.0	0.000007	0.000006	0.0	0.0	0.0	0.000000
Frankenstein; Or, The Modern Prometheus	-0.001590	0.00000	0.000000	0.0	0.000000	0.000023	0.0	0.0	0.0	0.000000
Grimms' Fairy Tales	-0.001740	0.00000	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	0.000013

5 rows × 24353 columns

Figure 1: First 5 books and their TF-IDF

While TF-IDF is sufficient to use in this project, we will discuss how it might not be perfect in larger and modern databases and how training a machine to recognize and categorize words might be challenging, accompanied by limitations such as synonyms, data structure, etc. We will observe our data, analyze the findings, and suggest multiple pathways to take in the future.

3 Methodology

3.1 Parsing

Before analyzing our data, we ensure that our data will be clean, which means that we will omit the Project Gutenberg logo, introduction lines, bibliography, title, table of contents, illustration, etc., and obtain the raw text only. As we performed this with Python libraries such as Pandas and NumPy, we had to strip parts manually, which can result in mistakes such as leaving behind illustrations, tables of contents that are formatted differently in some books, and so on. However, we ensure that those will not interfere with our TF-IDF analysis, as TF-IDF will mark high-frequency words that are common negative values regardless. Nevertheless, for time and space preservation, we tried to omit unnecessary content as much as possible. In addition, we also eliminated

numbers and special punctuation, as we wish to analyze and categorize raw words only.

3.2 TF-IDF Calculation

In order to calculate TF-IDF, we need to understand the differences between TF, IDF, and TF-IDF as a whole. For TF, we can understand it as how much this document talks about the word. For IDF, it is about how rare the word is across the whole library. Finally, for TF-IDF, we combined both of them and created a metric that depicts how much this document talks about the word and how rare the word is. The differences can be found in the following tables.

```
Alice's Adventures in Wonderland — top TF terms
the        0.06135
and        0.03223
to         0.02908
a          0.02381
it         0.02148

Beowulf— An Anglo-Saxon Epic Poem — top TF terms
the        0.08074
of         0.04220
to         0.02687
and        0.02035
in         0.01765

Dracula — top TF terms
the        0.05132
and        0.03615
i          0.02947
to         0.02941
of         0.02362

Frankenstein; Or, The Modern Prometheus — top TF terms
the        0.05851
and        0.04015
i          0.03807
of         0.03649
to         0.02818
```

Figure 2: TF most common terms

```
Top 5 IDF terms:
lattice    1.70475
meseemeth  1.70475
mercies    1.70475
merciless  1.70475
mercury    1.70475

Bottom 5 IDF terms:
a          -0.08701
best       -0.08701
rising     -0.08701
rise       -0.08701
lying      -0.08701
```

Figure 3: IDF most and least common terms

```
Alice's Adventures in Wonderland — top TF-IDF terms
alice      0.02685
gryphon    0.00361
hatter     0.00352
duchess    0.00262
turtle     0.00230

Beowulf— An Anglo-Saxon Epic Poem — top TF-IDF terms
beowulf    0.01112
hrothgar   0.00588
grendel    0.00446
folk       0.00256
higelac    0.00241

Dracula — top TF-IDF terms
helsing    0.00304
mina       0.00277
lucy       0.00260
jonathan   0.00235
van        0.00234

Frankenstein; Or, The Modern Prometheus — top TF-IDF terms
clerval    0.00150
felix      0.00144
elizabeth  0.00111
justine    0.00109
geneva     0.00080
```

Figure 4: TF-IDF most common terms

These tables deepen our understanding of TF-IDF even more: TF has the most common terms, IDF has the most unique terms, and TF-IDF combines both and finds the true frequency of the terms.

Nonetheless, as is evident in Figure 4, the terms for TF-IDF are mostly first names, as they are unique and appear frequently, though not as much as common English words. We figured out a way to soften this issue by comparing two books' TF-IDF with each other, as first names will become irrelevant in the comparison process. In addition, TF-IDF also struggles with differentiating synonyms and words that have a prefix or suffix over them. In Figure 3, the terms 'rising' and 'rise' are in separate values, yet they signify the same thing, only in different tenses. From this observation, we can also anticipate that TF-IDF can have problems with synonyms as well, as they can mean the same thing, yet in different words.

3.3 Nearest Neighbor Algorithms

Our goal is to find similar books to recommend to readers who like a certain book. To achieve this, we used a cosine algorithm in NumPy to find similar TF-IDF terms in two or more books and compare them. We also created a list of buzzwords (unigrams for simplicity) for each topic.

```
Top of topic: romance
Alice's Adventures in Wonderland            0.000091
Beowulf- An Anglo-Saxon Epic Poem           0.000042
Sense and Sensibility                       0.000023
Little Women; Or, Meg, Jo, Beth, and Amy    0.000021
Pride and Prejudice                         0.000018
Frankenstein; Or, The Modern Prometheus     0.000008
dtype: float64

Top of topic: monster
Moby Dick; Or, The Whale                    0.000053
Beowulf- An Anglo-Saxon Epic Poem           0.000038
Frankenstein; Or, The Modern Prometheus     0.000029
Grimms' Fairy Tales                         0.000028
Dracula                                     0.000019
Wuthering Heights                           0.000017
dtype: float64

Top of topic: sea
Moby Dick; Or, The Whale                    0.000427
Beowulf- An Anglo-Saxon Epic Poem           0.000029
Nora's twin sister                          0.000021
Dracula                                     0.000016
Frankenstein; Or, The Modern Prometheus     0.000009
Little Women; Or, Meg, Jo, Beth, and Amy    0.000002
dtype: float64

Top of topic: family
Beowulf- An Anglo-Saxon Epic Poem           0.000038
Pride and Prejudice                         0.000015
Little Women; Or, Meg, Jo, Beth, and Amy    0.000011
Wuthering Heights                           0.000005
Nora's twin sister                          0.000004
Frankenstein; Or, The Modern Prometheus     0.000001
dtype: float64
```

Figure 5: Books by topic

From this figure, we can analyze that the TF-IDF does decently well in analyzing words and matching them with the buzzwords under each topic. However, a limitation we can point out here is it has trouble against poems, as "Beowulf" has appeared under romance when it only shows words under the romance category, such as "love" and "affection," to prove how determined a hero is for his ideal instead of relationships. A reasonable explanation would be that poems have a different structure compared to traditional books. Thus, we can conclude that TF-IDF can only read words as they are regardless of the context and structure.

```
recommend("Frankenstein; Or, The Modern Prometheus", k=5)
```

```
Because you liked: Frankenstein; Or, The Modern Prometheus
Grimms' Fairy Tales         0.715644
Dracula                     0.683694
Pride and Prejudice         0.598882
Wuthering Heights           0.586172
Moby Dick; Or, The Whale    0.511576
Name: Frankenstein; Or, The Modern Prometheus, dtype: float64
```

Figure 6: Recommended books by NumPy

Furthermore, to test out another algorithm that does the same thing, we used Faiss Nearest Neighbor Search algorithm to enhance our understanding.

```
faiss_neighbors("Frankenstein; Or, The Modern Prometheus", k=5)
```

```
[(np.str_("Grimms' Fairy Tales"), np.float32(0.71563566)),
 (np.str_('Dracula'), np.float32(0.6836748)),
 (np.str_('Pride and Prejudice'), np.float32(0.5988808)),
 (np.str_('Wuthering Heights'), np.float32(0.5861689)),
 (np.str_('Moby Dick; Or, The Whale'), np.float32(0.51157194))]
```

Figure 7: Recommended books by Faiss

The books being recommended are exactly the same. Upon researching further, we found out that Faiss is appropriate for larger and more complicated databases, as it saves time and space. Thus, we concluded that the NumPy methodology is still an efficient tool for this project.

4 Results

After researching and exploring what TF-IDF can do, we can conclude that TF-IDF did a sufficient job on analyzing the data, as it can give out relevant and accurate books to match under a topic, though not without some limitations.

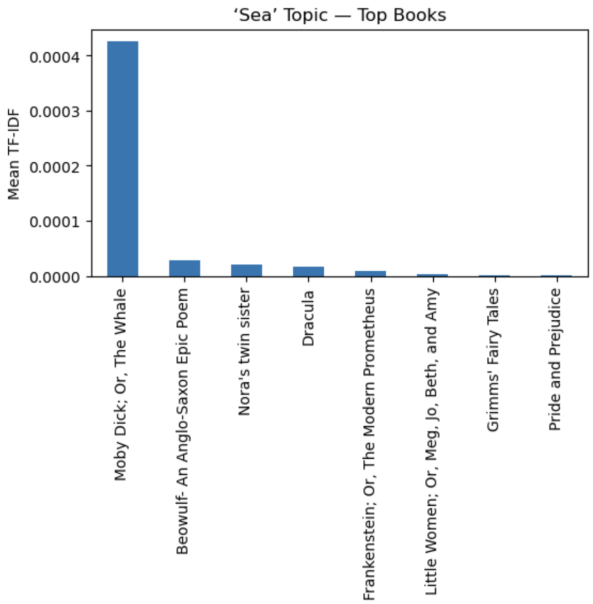

Figure 8: Ranking by topic sample

With the results above, we can trust TF-IDF to recommend us classical

books that are similar to each other or even by topic. Albeit it might not be perfect or ideal, this project is made to show that we can have more potential with TF-IDF or similar tools if we train them to differentiate between first names, context, structure, synonyms, and more.

5 Conclusion

TF-IDF is still a relevant tool to recognize words and their frequency in the modern age, with some limitations being solvable with time and dedication. Some of its limitations include first names, context, structure, synonyms, etc. However, with more modern tools such as BM25, we can anticipate a bright future for Information Retrieval (IR) in the future.

6 References

1. Jurafsky, D., & Martin, J. H. (n.d.). *Speech and Language Processing* (3rd ed., draft), ch. 11, p. 13.
2. CS242 Staff. (n.d.). *[Project 1: TFIDF Handout]*. Course handout.