Comparing Amazon books similarity in a scalable way through Spark NLP

Luca Sangiovanni

Abstract

This project aims at measuring the similarity of Amazon books by looking at users' reviews and books descriptions, in an efficient way, so that it can be replicated on larger datasets. Two different pipelines have been used for analyzing the texts, together with two similarity measures: Jaccard similarity and Cosine similarity.

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work, and including any code produced using generative AI systems. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

1 Data and Preprocessing

1.1 Introduction to the Data

The data used for the project is provided by Kaggle, and is made of two datasets: *books_rating.csv* contains users reviews of the books, and it's made of around 3 million rows (1 row per review), while *books_data.csv* contains a short plot description for each book, with around 200,000 rows.
Since I used my personal laptop CPU, in order to decrease the code running time I used a smaller subset of the datasets, but the results can be easily replicated on larger portions of data, since everything has been done in a way that prioritizes maximum efficiency, by using the PySpark and Spark NLP packages provided by Python, which allow for distibuted computing, increasing the efficiency on larger datasets.
Regarding the reviews dataset, I decided to create two filters to subset the

data, and both of them contain only reviews that are between 300 and 600 characters long (which I considered to be an adequate length to capture key information from a review) of books that have a number of reviews above a certain threshold (which varies between the two filters); I decided to keep only 50 reviews per book. Neither filter contains NA values.
The two filters that can be used are:

- big_only_fiction: It is the larger dataset, and it has been filtered to only contain books that fall under the *fiction* genre. It contains 135 unique books, each with exactly 50 reviews, for a total of 6750 unique reviews (rows).

- small_full: It is the smaller one; it contains books of all genres, and it has been filtered to only contain books which have at least 260 reviews. The final dataset has 16 unique books, each with exactly 50 reviews, for a total of 800 unique reviews.

The datasets resulting from the cleaning process are the **df_reviews** (which comes from *books_rating.csv*) and the **df_descriptions** (which instead comes from *books_data.csv*). They both have a simple structure, with only two columns: the book title together with either the review (*Table 1*) or the book description (*Table 2*).

book_title	**review_text**
The Lies of Locke Lamora	From very early on, this book grabbed me and wouldn't let go. Great world building. Believable and likable characters. Lots of fun.The only thing I didn't care for in this book is the frequent [...].

Table 1: Structure of the dataset *df_reviews* after being cleaned.

book_title	**description**
House of Mirth	A black comedy of manners about vast wealth and a woman who can define herself only through the perceptions of others. The beautiful Lily Bart lives among the nouveaux riches of [...]

Table 2: Structure of the dataset *df_descriptions* after being cleaned.

1.2 Reviews aggregation methods

Since each book in the dataset was associated with multiple reviews, it was necessary to aggregate the information at the book level in order to make mean-

ingful comparisons between books. Two separate aggregation strategies were applied, each tailored to a specific type of similarity measure that I used:

- **Vector-based (for Cosine Similarity)**: To compute cosine similarity between books based on the semantic content of their reviews, I first transformed each review into an embedding vector. However, since each book had multiple reviews (and therefore multiple vectors), I needed a way to reduce these to a single vector per book. To do this, I grouped all vectors by book title and calculated their average. The averaging was performed element-wise across all vectors, resulting in a single representative embedding for each book. This average vector captures the overall meaning of the book reviews. After averaging, I normalized each vector to ensure fair and consistent computations.
- **Text-based (for Jaccard Similarity)**: For Jaccard similarity, which is based on comparing sets of words rather than numerical vectors, I needed a single block of text per book. To create this, I grouped the dataset by book title and concatenated all review texts for each book into one combined document. This aggregated text was later passed to a custom pipeline and converted into sets to compute the similarity between books.

2 Measures of similarity

2.1 Similarities

From the point of view of information theory, similarity is defined as the commonness between two text snippets. The greater the commonness, the higher the similarity, and vice versa [5].
To compare the similarity of the books, I looked at both the *books reviews* provided by the users and the *book plots descriptions* provided directly by Amazon. I wanted to use two different approaches, and compare their results, therefore I employed two of the most used similarity metrics: Cosine Similarity and Jaccard Similarity.

2.2 Cosine Similarity

Cosine similarity is a measure of semantic similarity, which calculates the cosine of the angle between two vectors that are projected in a multi-dimensional space (embeddings), regardless of their size; its value is bounded between -1 and 1, and it is often used to measure document similarity in text analysis [7].
This metric operates on two real-valued vectors of equal length, and the vectors typically represent documents from models like dense semantic embeddings (the one I used, as we will see in the next chapters). These *word embeddings* provide

vector representations of words in which these vectors retain the underlying linguistic relationship between the words [1].

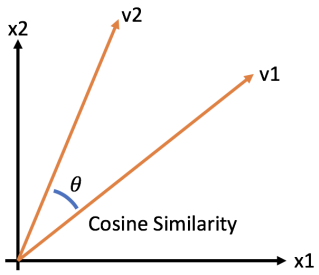


Figure 1: Cosine similarity.

$$\text{Cosine Similarity}(\vec{A}, \vec{B}) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \cdot \|\vec{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$

2.3 Jaccard Similarity

On the other hand, **Jaccard similarity** treats data elements as a set. It is a string-based similarity measure which works on string chains and characters organization using a term-based similarity procedure [3]. By dividing the size of the union by the size of the intersection of the two sets, this coefficient can be calculated [2]. Therefore, as we said in the previous chapter, it takes as input two sets of elements, usually sets of tokens, and not word embeddings.

$$\text{Jaccard Similarity}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Since the Jaccard similarity is limited to assessing the lexical similarity of texts, the cosine similarity might be better suited for comparing the semantic content of texts represented by vectors. I tried both metrics, and compared the results.

3 NLP Pipelines

Since the two similarity measures I used take two different inputs (the cosine similarity expects *numerical input* in the form of vectors, while the Jaccard ex-

pects *set of tokens*), I decided to implement two different NLP pipelines when processing text data, for both the reviews and the books descriptions.

3.1 Pretrained BERT

This first pipeline uses a pretrained **BERT** (Bidirectional Encoder Representations from Transformers) model to generate sentence embeddings, necessary for cosine similarity. Since this metric computes the angle between two vectors in high-dimensional space, the input must be real-valued vectors that represent the semantic content of the text.
BERT is a state-of-the-art language model developed by Google that has been widely adopted for various NLP tasks, and it is a *corpus-based* model, which means that it uses the information obtained from a corpus to to measure the similarity between terms [6]. Its strength lies in its ability to understand the context of words in both directions, left and right, within a sentence. Unlike traditional models that read text in a fixed direction (left-to-right or right-to-left), BERT reads the entire sentence at once, allowing it to capture deeper and more nuanced meanings of words depending on their context. Notably, BERT produces context-dependent embeddings: while other models like *word2vec* duce a fixed embedding for each word, BERT can produce different embeddings for the same word depending on the context [4], which contributes at making this architecture very powerful.
I used the *sent_small_bert_L2_128* model from Spark NLP, which is a lightweight BERT variant optimized for sentence-level embeddings. This model takes an input sentence (book review or description) and outputs a fixed-length dense vector (**embedding**) that captures the semantic content of the sentence.

3.2 Custom Pipeline

The second pipeline, on the other hand, is designed for Jaccard similarity, which works on sets of tokens rather than vectors. Since Jaccard similarity measures the overlap between two sets, I needed a pipeline that produces clean, tokenized text. This **custom pipeline** includes document assembly, tokenization, stopword removal, and lemmatization. The output is a list of lemmatized tokens for each document. These token sets are then compared using Jaccard similarity, which calculates the ratio between the number of shared tokens and the total number of unique tokens.

3.3 Execution Times

While, as we will see in the next chapters, the BERT pipeline, combined with the cosine similarity tends to have very good results in terms of ability of computing similarities between books, it is also true the fact that its code execution time is

way higher than its Jaccard counterpart, with the custom pipeline. By looking at *Figure 2*, which shows the computing time for the *big_only_fiction* subset of the data, the difference is clear.

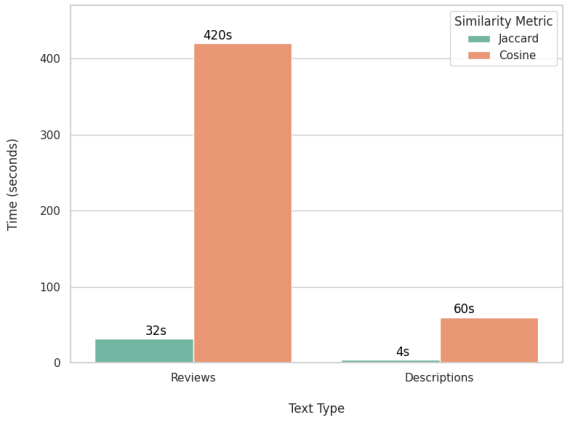


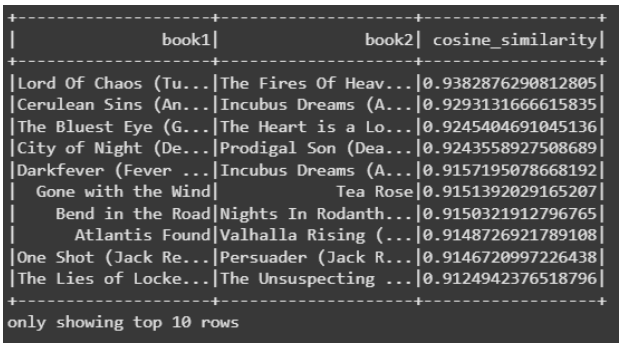
Figure 2: Cosine similarity and Jaccard similarity code execution times, for the *big_only_fiction* dataset.

4 Experimental Results

After defining the methods, we can go on and apply them to our data; I used the bigger dataset, so that the displayed results will be more informative.

4.1 BERT Pipeline & Cosine Similarity

If we look at the reviews data, after using the BERT pipeline we can see which books have the most similar reviews, by looking at the averaged embeddings for each book.



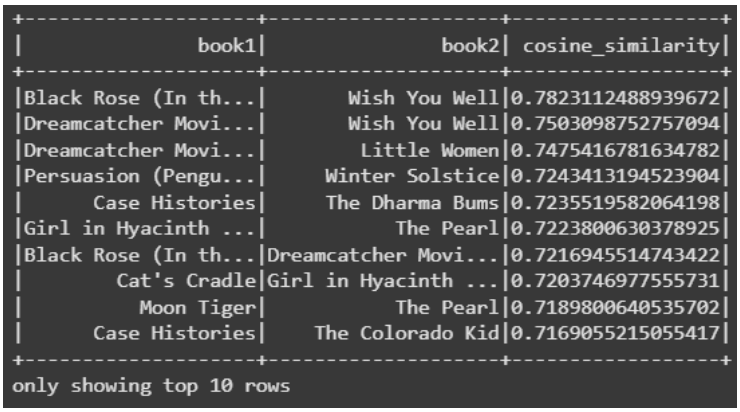
book1	book2	cosine_similarity
Lord Of Chaos (Tu...	The Fires Of Heav...	0.9382876290812805
Cerulean Sins (An...	Incubus Dreams (A...	0.9293131666615835
The Bluest Eye (G...	The Heart is a Lo...	0.9245404691045136
City of Night (De...	Prodigal Son (Dea...	0.9243558927508689
Darkfever (Fever ...	Incubus Dreams (A...	0.9157195078668192
Gone with the Wind	Tea Rose	0.9151392029165207
Bend in the Road	Nights In Rodanth...	0.9150321912796765
Atlantis Found	Valhalla Rising (...	0.9148726921789108
One Shot (Jack Re...	Persuader (Jack R...	0.9146720997226438
The Lies of Locke...	The Unsuspecting ...	0.9124942376518796

only showing top 10 rows

Figure 3: Top 10 most similar books, based on reviews, according to Cosine Similarity.

As we can see from *Figure 3*, the top 10 most similar books have a high value of cosine similarity (which ranges from -1 to 1), and therefore can be considered to have strong similarities. The validity of our results can be easily confirmed just by looking at the books names: the most similar books are *Lord of Chaos* and *The Fires of Heaven*, which are both part of a series of novels called *The Wheel of Time*, by the American author Robert Jordan, and they are sequels. Going on to the second couple of most similar books, we can find *Cerulean Sins* and *Incubus Dreams*, that are also sequels, part of the *Anita Blake: Vampire Hunter* saga.
If we had to go down the list, we would notice that the similarities are not random (they consistently share themes, genres, or narrative structures), which can lead us to think that our model based on BERT embeddings can be considered valid.

Moving on to the books descriptions, we can do the same thing. In this case, however, the cosine similarity is, on average, way lower than the one we just saw in the case of users reviews; the most similar books, in fact, have a cosine similarity of 0.78. (*Black Rose* and *Wish You Well*). This cause of this result can easily be found in the fact that the amount of reviews is way higher, while books descriptions are only one per book. Despite the "low" amount of text provided by each review, our model was still able to provide very high values of similarity, proving the robustness of the pipeline and the model itself. The top 10 most similar books can be seen below, in *Figure 4*.



book1	book2	cosine_similarity
Black Rose (In th...	Wish You Well	0.7823112488939672
Dreamcatcher Movi...	Wish You Well	0.7503098752757094
Dreamcatcher Movi...	Little Women	0.7475416781634782
Persuasion (Pengu...	Winter Solstice	0.7243413194523904
Case Histories	The Dharma Bums	0.7235519582064198
Girl in Hyacinth ...	The Pearl	0.7223800630378925
Black Rose (In th...	Dreamcatcher Movi...	0.7216945514743422
Cat's Cradle	Girl in Hyacinth ...	0.7203746977555731
Moon Tiger	The Pearl	0.7189800640535702
Case Histories	The Colorado Kid	0.716905215055417

only showing top 10 rows

Figure 4: Top 10 most similar books, based on descriptions, according to Cosine Similarity.

We can also specify a particular book we are interested in, and find the top 3 most similar books, according to both user reviews (*Figure 5*) and to books descriptions (*Figure 6*).

book1	book2	cosine_similarity
Red Tent	The Bluest Eye (G K Hall Large Print Book Series)	0.8634628167803026
Red Tent	The Heart is a Lonely Hunter	0.8630728676725744
Red Tent	The Other Boleyn Girl	0.8611876205898488

only showing top 3 rows

Figure 5: Top 3 most similar books to *Red Tent*, according to user reviews (Cosine Similarity).

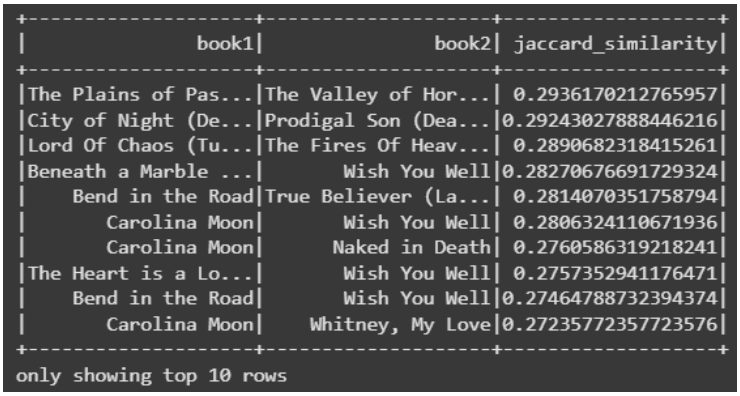
book1	book2	cosine_similarity
Moon Tiger	The Pearl	0.7189800640535702
Moon Tiger	Whitney, My Love	0.6969640680856999
Moon Tiger	The Reader	0.689377755480393

only showing top 3 rows

Figure 6: Top 3 most similar books to *Moon Tiger*, according to books descriptions (Cosine Similarity).

4.2 Custom Pipeline & Jaccard Similarity

If we do the same analysis, but follow the custom pipeline and use the Jaccard Similarity as similarity metric, we notice that the results are quite different. As we saw in Chapter 2, pretrained BERT is better for our task, since it is able to understand the underlying context and meaning of a text, even when two reviews use different wording but mean similar things. This is critical when dealing with recommendation, where semantics can be considered more important than surface overlap.
On the other hand, although our custom pipeline is more easily interpretable, simple and fast to implement, it's not as effective in comparing similar texts, since it doesn't have an understanding of context or semantics inside a written text. As a consequence, as we can see in *Figure 7*, the value of Jaccard Similarity (which instead ranges from 0 to 1), is generally low or moderate, indicating limited overlap.



book1	book2	jaccard_similarity
The Plains of Pas...	The Valley of Hor...	0.2936170212765957
City of Night (De...	Prodigal Son (Dea...	0.29243027888446216
Lord Of Chaos (Tu...	The Fires Of Heav...	0.2890682318415261
Beneath a Marble ...	Wish You Well	0.28270676691729324
Bend in the Road	True Believer (La...	0.2814070351758794
Carolina Moon	Wish You Well	0.2806324110671936
Carolina Moon	Naked in Death	0.2760586319218241
The Heart is a Lo...	Wish You Well	0.2757352941176471
Bend in the Road	Wish You Well	0.27464788732394374
Carolina Moon	Whitney, My Love	0.27235772357723576

only showing top 10 rows

Figure 7: Top 10 most similar books, based on reviews, according to Jaccard Similarity.

The performance difference can be seen in a better way if we look at the similarity of the books based on their plot description (*Figure 8*). In this case the only couples which have a significant similarity are *City of Night* with *Prodigal Son* (both part of the *Frankenstein* series), and *House of Mirth* with *Sense and Sensibility*. Again, this difference is both due to the fact that books descriptions have way less characters compared to users reviews, but also because of the similarity metric used, which is less powerful than the Cosine Similarity.

book1	book2	jaccard_similarity
City of Night (De...	Prodigal Son (Dea...	0.5789473684210527
House of Mirth	Sense and Sensibi...	0.28301886792452824
Memory in Death (...	True Believer (La...	0.12269938650306744
Dreamcatcher Movi...	Wish You Well	0.11895910780669139
Dreamcatcher Movi...	Little Women	0.11811023622047245
Bridget Jones's D...	Pride and Prejudice	0.11764705882352944
Little Women	Wish You Well	0.11267605633802813
Case Histories	The Good German	0.10227272727272729

Figure 8: Top 10 most similar books, based on descriptions, according to Jaccard Similarity.

5 Visualizing the similarities

In order to understand better the similarities between books, I created a plot, after having processed the data through a *UMAP* algorithm. UMAP (Uniform Manifold Approximation and Projection) is a dimensionality reduction technique used to represent high-dimensional data in a lower-dimensional space, often two or three dimensions, for visualization and analysis. The result can be seen in the image below (*Figure 9*).

Figure 9: 2D visualization of book embeddings using a UMAP model for dimensionality reduction. N.B.: I used the *small_full* dataset, since the bigger dataset had too many books, therefore the visualization would have been messy.

By looking at the plot, we can see that the similarities computed with the Cosine Similarity visible:

- In the top left corner we can see some classic literature books (like *Persuasion*, *Pride and Prejudice* or *Little Women*).
- On the right side we can see books which can be considered more modern (like *The Notebook*, *The Happiest Baby on the Block* or *Me Talk Pretty*).
- If we look on the bottom right, we see books like *Dune*, *Fahrenheit 451*, *Wizard's First* or *Hannibal*, likely sci-fi/fantasy/thriller books.

As written in the plot caption, in this case I preferred to use the smaller subset of the data, so that the results inside the image don't override with each other, and the result is more easily understandable. Of course, since this smaller dataset has a very limited amount of books and reviews, the result is not as significant as the results shown in the other chapters (where I used the bigger dataset), but this technique can be applied to any dataset.

6 Final Remarks

This project set out to compare Amazon books by analyzing their user reviews and descriptions in a way that can easily scale to larger datasets, and the usage of PySpark and Spark NLP allowed me to build efficient pipelines. Considering the two approaches tested, the BERT-based pipeline combined with Cosine Similarity clearly produced the most meaningful results. It was able to pick up on semantic similarities between books, even when they used different words. In some cases, it correctly identified sequels or books from the same series as highly similar, which confirms the strength of this method. The custom pipeline using Jaccard Similarity was faster and easier to interpret, but it struggled with capturing deeper meaning and context, returning lower similarity scores. The use of UMAP for visualizing the book embeddings also helped to show how books cluster based on content, reinforcing what we observed through the similarity scores. In the end, this project showed that with the right tools and models, it's possible to measure text-based similarity at scale in a meaningful way. There's room to expand this work further by using larger datasets, experimenting with other models, or including more features like author, genre, or publication year to refine the similarity even more.

References

[1] Dhivya Chandrasekaran. Evolution of semantic similarity — a survey. *CoRR*, 2021.

[2] Joyinee Dasgupta. A survey of numerous text similarity approach. *International Journal of Scientific Research in Computer Science Engineering and Information Technology*, 2023.

[3] Nova Eka Diana. Measuring performance of n-gram and jaccard similarity metrics in document plagiarism application. *Journal of Physics Conference Series*, 2019.

[4] Qixiang Fang. Evaluating the construct validity of text embeddings with application to survey questions. *EPJ Data Science*, 2022.

[5] Dekang Lin. An information-theoretic definition of similarity. *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, 1998.

[6] Jiapeng Wang. Measurement of text similarity: A survey. *Information*, 2020.

[7] Ramadan Zebari. Evaluating of efficacy semantic similarity methods for comparison of academic thesis and dissertation texts. *Science Journal of University of Zakho*, 2023.