Enhancing Personalized Recommendations: A Spacy-Based Hybrid Approach for Book Recommendation Systems

JAYANK TYAGI[1], VANDANA CHOUDHARY[2], NAMITA GOYAL[3]
[1, 2, 3] Maharaja Agrasen Institute of Technology, Delhi, India

Abstract- Recommendation systems have become ubiquitous in our digital age, with various platforms utilizing algorithms to predict what items users might be interested in and offer personalized recommendations. To build more effective recommendation models, it is essential to analyse and understand natural language, such as product descriptions or user reviews. Tokenization, part-of-speech tagging, named entity recognition, dependency parsing, and other features offered by Spacy, an open-source NLP toolkit, are effective tools that may be used to extract meaning from natural language text. Using Spacy, it is possible to identify the key topics or themes of a product description or review and match them to the interests of individual users. Spacy's speed and scalability make it well-suited for real-time recommendation systems that need to process large volumes of data quickly. As such, Spacy has become increasingly popular among researchers and practitioners working in this area, due to its unique set of NLP features and capabilities.

I. INTRODUCTION

1.1 About Spacy

Recommendation systems have become ubiquitous in our digital age, with various platforms utilizing algorithms to predict what items users might be interested in and offer personalized recommendations. To build more effective recommendation models, it is essential to analyse and understand natural language, such as product descriptions or user reviews. Tokenization, part-of-speech tagging, named entity recognition, dependency parsing, and other features offered by Spacy, an open-source NLP toolkit, are effective tools that may be used to extract meaning from natural language text. Using Spacy, it is possible to identify the key topics or themes of a product description or review and match them to the interests of individual users. Spacy's speed and scalability make it well-suited for real-time recommendation systems that need to process large volumes of data quickly. As such, Spacy has become increasingly popular among researchers and practitioners working in this area, due to its unique set of NLP features and capabilities.

1.2 Types of recommendation systems

Recommendation systems can have various variants, including collaborative filtering, content-based filtering, and hybrid approaches that merge the two. Here's a brief overview of each type:

1. Collaborative filtering: based on the idea that people who have previously expressed preferences in a similar way are likely to do so in the future. Based on historical user behaviours, such as ratings or purchase histories, collaborative filtering algorithms produce tailored recommendations. The two most common types are user-based and item-based collaborative filtering. While user-based collaborative filtering makes suggestions based on the preferences of people who are similar to the user, item-based collaborative filtering makes suggestions that are similar to products the user has previously enjoyed [1].

2. Content-based filtering: This strategy suggests goods that are similar to those that a user has previously appreciated based on the properties of the things themselves. If a user likes a particular kind of music, an algorithm using content-based filtering can, for example, suggest other songs or performers. Content-based filtering may be especially useful when there is limited data available regarding user activities [2].

3. Hybrid approaches: These approaches combine content-based and cooperative filtering to benefit from each approach's strengths. Hybrid approaches can

improve the accuracy and effectiveness of suggestions by accounting for both user behaviour and item properties. [3].

There are further recommendation systems that consider additional aspects including user demographics and geography, such as knowledge-based systems and context-aware systems. However, in research and business, collaborative filtering, content-based filtering, and hybrid techniques are most frequently employed.

II. LITERATURE REVIEW

Recommendation systems have become an essential part of our digital life, with numerous platforms utilising algorithms to present consumers with personalised suggestions. Natural Language Processing (NLP) plays a crucial role in building effective recommendation models, by analyzing and understanding natural language, such as product descriptions or user reviews. Spacy, an open-source NLP library, has become an increasingly popular tool among researchers and practitioners working on recommendation systems, due to its unique set of NLP features and capabilities.

A literature review was conducted to identify existing research on Spacy-based recommendation systems. The review included academic papers, conference proceedings, and industry reports. The following key findings and research gaps emerged from the review:
Spacy has been used successfully in several recommendation systems, including those for music, movies, and restaurants. For instance, Spacy has been used to extract keywords and entities from restaurant reviews and trained a collaborative filtering model to make personalized recommendations [5].

Spacy's NLP capabilities, such as named entity recognition and sentiment analysis, can be used to improve the accuracy of recommendation systems. For example, Spacy was to extract sentiment scores from user reviews and incorporate them into a content-based filtering model for movie recommendations [5].
Spacy can be used in conjunction with deep learning and other machine learning methods to create more sophisticated recommendation systems. For example, a combination of Spacy and a deep neural network was used to recommend personalized YouTube videos based on user watch histories [4].

However, there is a lack of research on the scalability and efficiency of Spacy-based recommendation systems, especially in comparison to other NLP libraries. There is also a need for more research on Spacy-based recommendation systems in niche domains, such as healthcare or finance.

Finally, there is a need for more research on the ethical implications of using Spacy-based recommendation systems, such as the potential for bias or the impact on user privacy.

Finally, while there is existing research on Spacy-based recommendation systems, there are significant gaps in the literature that must be filled. Scalability and efficiency, domain-specific applications, and ethical considerations are among the gaps. This work seeks to fill these gaps by offering a thorough analysis of the existing state-of-the-art in Spacy-based recommendation systems and recommending areas for future research.

III. SPACY-BASED RECOMMENDATION MODELS

Spacy is a free and open-source natural language processing (NLP) package with a variety of capabilities that can be used to create successful recommendation systems. We will explore three major Spacy-based recommendation models in this section: collaborative filtering, content-based filtering, and hybrid techniques. We will detail the precise Spacy features employed in each model and how they contribute to the model's effectiveness.

1. Collaborative Filtering: Collaborative filtering is a popular recommendation model that relies on user behavior, such as ratings and interactions, to recommend items to users with similar tastes. Spacy can be used to preprocess and analyze user-generated content, such as reviews and comments, to better understand user preferences.
2. Content-Based Filtering: Content-based filtering is another common recommendation model that relies on the features of items to recommend similar items to users. Spacy can be used to extract

features from item descriptions and user-generated content to build effective content-based filtering models.

3. Hybrid Approaches: To give more accurate and diversified recommendations, hybrid recommendation models combine the characteristics of collaborative filtering with content-based filtering. Spacy may be used to develop successful hybrid models by extracting characteristics from both user-generated content and item descriptions.

IV. PROJECT DESCRIPTION

To get a better grasp and understanding of Spacy based recommendation systems, a project on book recommendation systems based on Spacy employ NLP and generate content-based recommendations was built.

For the project data we used a data set scrapped by researchers at USCD as good reads did not have an API after 2016. The books metadata data set consisted of the metadata of 2.36 million books which contain data like the name, author, year of release, description, number of ratings etc. To generate accurate recommendations, we plan to employ several strategies like content based and collaborative filtering.

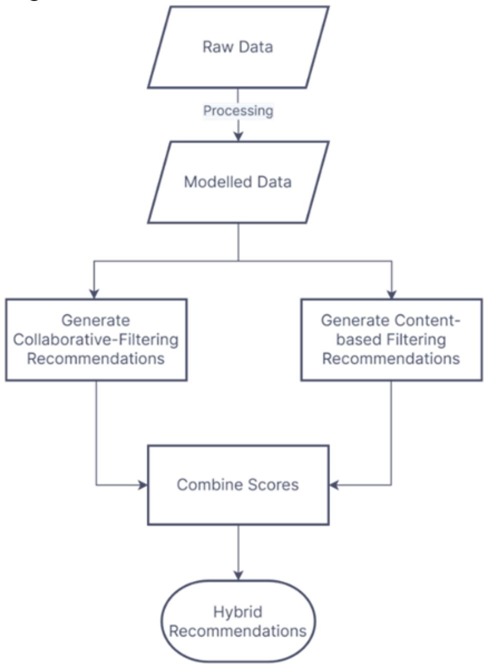
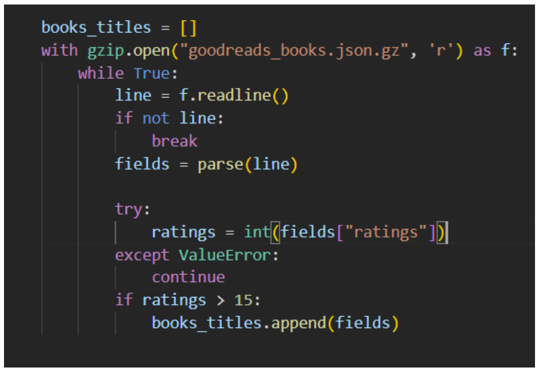

Raw Data
Processing
Modelled Data
Generate Collaborative-Filtering Recommendations
Generate Content-based Filtering Recommendations
Combine Scores
Hybrid Recommendations

Figure 1: Steps to create a recommendation system

V. DATA

We used the dataset scrapped by researcher's at USCD. the books metadata dataset which contains the metadata of 2.36 million books which contains data like the name, author, year of release, description, number of ratings etc. and the user book-interactions dataset which has 229 million user-book interactions, which is a set of extracted '.csv' files, where each user-book interaction is represented by several integers [6-7]. This dataset records the user-ID and the interaction of that user given by a unique "user_id" with each book given by its unique "book_id" and it also records what the user has rated that book out of five.

Before we could begin to even generate any recommendations, the data had to be modelled to isolate the fields that are essential to building a recommendation system. Firstly, To work with such a large dataset (~ 4GB), so a streaming technique of reading the data line by line to overcome memory limitations was employed. The dataset contained multiple versions of the same book so to overcome duplicity and generally improve the generated recommendations, only books with more than 15 ratings were considered for the dataset which returned 1.3 million books out of the 2.36 million.



```
books_titles = []
with gzip.open("goodreads_books.json.gz", 'r') as f:
    while True:
        line = f.readline()
        if not line:
            break
        fields = parse(line)

        try:
            ratings = int(fields["ratings"])
        except ValueError:
            continue
        if ratings > 15:
            books_titles.append(fields)
```

Figure 2: code filtering books with ratings > 15

VI. BUILDING A MAKESHIFT SEARCH ENGINE

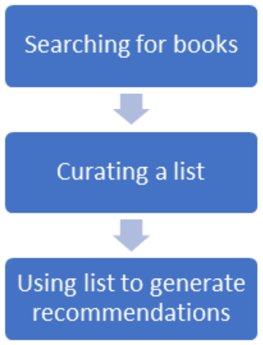


Figure 3: Steps for a recommendation system

To begin, a rudimentary search engine was needed to identify the book IDs of the books we wanted. The TF-IDF approach was used to reflect how important a word is to a document in a collection or corpus.

The term frequency-inverse document frequency (TF-IDF) is a statistical metric used to assess the significance of a word in a document or set of documents. It is often utilised in tasks like as information retrieval and natural language processing. The term frequency (TF) and the inverse document frequency (IDF) are combined to form TF-IDF.

The term frequency stands for the count of times a specific word appears in a document. A higher term frequency indicates that the word is more important in the document.

A word's usage frequency across all of the documents in a collection is calculated using the inverse document frequency. While words that are uncommon or unique to a certain document will have a high inverse document frequency, words that are relatively prevalent in multiple texts will have a low inverse document frequency.

The sum of a word's term frequency and inverse document frequency determines its final TF-IDF score within a document. Words with high TF-IDF scores are considered to be more important or relevant to the content of the document.

Now, to finally match our given search query with the given book titles cosine similarity was used. Cosine similarity is a measure used to compare two sequences of numbers by taking the cosine of the angle between them. These sequences are treated as vectors in a mathematical space, and the cosine similarity is determined by the dot product of the vectors divided by the product of their lengths. The magnitude of the vectors is not important in this calculation, only the angle between them. This metric is often used in data analysis to compare numerical sequences. [8].

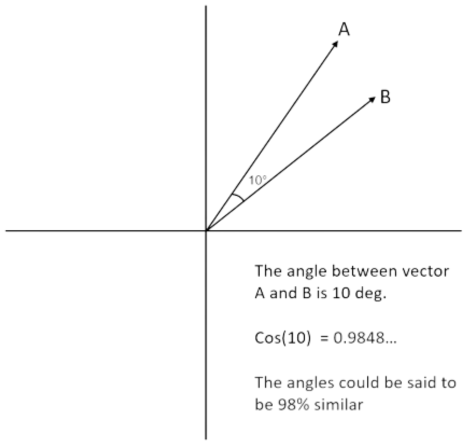

A
B
10°
The angle between vector A and B is 10 deg.
Cos(10) = 0.9848...
The angles could be said to be 98% similar

Figure 4: Cosine similarity visualized

To make sure that the books returned are an exact match we made the results clickable and showed image by writing two functions that return our results as hypertext.

Now, a list of “book_id” of liked books can be created by exactly matching our search results.

VII. GENERATING RECOMMENDATIONS

To generate recommendations, we need to map book_id in the dataset to the book_ids we we used the “book_id_map.csv” to map all of the book_ids.

Now, we need to find the list of all the users that have read and reviewed the same books as us and rated them highly (> 4 stars), as these are the users that like similar books as us. To find such users we used the “goodreads_interactions.csv” that contains all of the reviews and ratings books by each user. We create a list of all books that users who like the same books as us like. We will use this list of books to generate the recommendations.

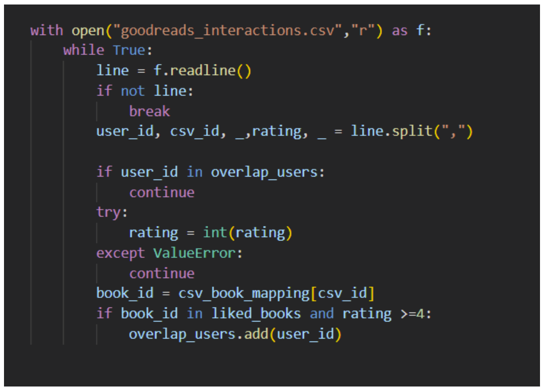

```
with open("goodreads_interactions.csv","r") as f:
    while True:
        line = f.readline()
        if not line:
            break
        user_id, csv_id, _,rating, _ = line.split(",")

        if user_id in overlap_users:
            continue
        try:
            rating = int(rating)
        except ValueError:
            continue
        book_id = csv_book_mapping[csv_id]
        if book_id in liked_books and rating >=4:
            overlap_users.add(user_id)
```

Figure 5: Code showing a list of similar books

a. Collaborative Filtering Technique

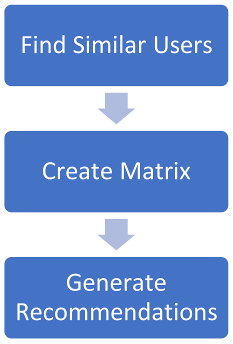


Figure 6: Steps for collaborative filtering

In a collaborative filtering approach, we first identify a list of similar users and the books they enjoyed, and then, using the scipy and sklearn libraries in Python, we calculate the cosine similarity between a specific user and all other users. Finally, we identify the 15 users who are most similar to that user based on their ratings data. In order to generate book recommendations, the code then combines the ratings information for these comparable individuals.

The first step is to create a sparse matrix called "ratings_mat_coo" from the "interactions" DataFrame using the "coo_matrix" function from scipy, as described in a previous answer. The "ratings_mat_coo" matrix is then converted to a Compressed Sparse Row (CSR) format matrix called "ratings_mat" using the "tocsr" method.

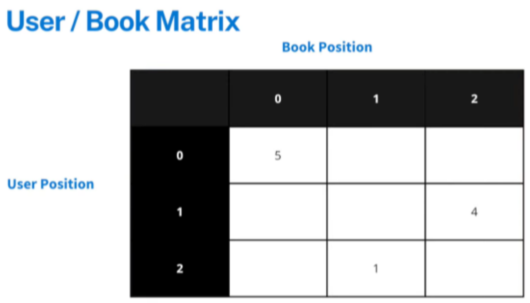


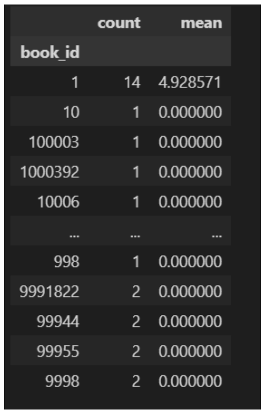
Figure 7: Illustrating a user/book matrix

Next, the code uses the "cosine_similarity" function from sklearn's "metrics.pairwise" module to compute the cosine similarity between the ratings of the end user (specified by "my_index" variable) and the ratings of all other users in the dataset. The resulting similarity matrix is flattened into a 1D array using the "flatten" method.

The indices of the 15 users with the highest resemblance to the target user are then found using the "argpartition" function from the numpy library. These indices are stored in the "indices" variable.

Next, the code filters the "interactions" DataFrame to only include interactions from the similar users, using the "isin" method to select rows with user indices in the "indices" array. The resulting DataFrame is stored in the "similar_users" variable.

Finally, using the "groupby" and "agg" methods, the code groups the "similar_users" DataFrame by book ID and aggregates the rating data. The "count" and "mean" functions are used to get the number of ratings for each book and the average rating. The generated DataFrame, "book_recs," includes suggestions for the target user based on the ratings of other users.



	count	mean
book_id		
1	14	4.928571
10	1	0.000000
100003	1	0.000000
1000392	1	0.000000
10006	1	0.000000
...	...	...
998	1	0.000000
9991822	2	0.000000
99944	2	0.000000
99955	2	0.000000
9998	2	0.000000

Figure 8: Generated collaborative filtering recommendations

To normalize the generated ratings between 0-1 which would be essential in order to generate an effective hybrid recommendation as we don't want a larger numerical rating to dominate other smaller ratings that we will generate with the content-based recommendations so we used the Min-Max normalization technique, also known as feature scaling, is a technique used to rescale numerical values in a dataset to a specific range. It transforms the original data so that it falls within a predefined range, typically between 0 and 1.

The process involves the following steps:

1. Determine the dataset's minimum and maximum values for the feature you want to normalize.
2. Subtract the minimum value from each value in the feature, then divide the result by the range (the range being the range between the minimum and maximum values).
3. Now, the values will fall into the range of 0 and 1, with 0 denoting the smallest value and 1 denoting the maximum value. Values in between are linearly scaled based on their relative position within the range.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Figure 9: Min-Max Normalization

Collaborative filtering is the name given to this approach of generating recommendations. Collaborative filtering is a machine learning technique that is often utilised in the development of book recommendation systems. It works by analysing previous user behaviour (for example, what books they have purchased or reviewed) and discovering trends in this data. Based on these patterns, the algorithm can then offer things to users that are similar to those in which they have previously expressed interest.

The accuracy of a collaborative filtering-based book recommendation system is affected by the quality of the training data, model complexity, and implementation-specific aspects of the recommendation algorithm. In general, collaborative filtering systems can be extremely good at providing suggestions, but there is always a trade-off between recommendation accuracy and model computational complexity.

One way to enhance the accuracy of a collaborative filtering-based recommendation system is to incorporate additional information about the users and the items being recommended. For example, incorporating metadata about the books (e.g., genre, author, publication date) can help the system make more accurate recommendations by allowing it to take into account the characteristics of the items being recommended.

Overall, collaborative filtering is a potent approach for developing book recommendation systems, and when combined with the correct data and model design, it can yield very accurate suggestions.

A study published in the journal ACM Transactions on Information Systems found that a matrix factorization technique called singular value decomposition (SVD) achieved an accuracy rate of 80% on a dataset of movie ratings.

b. Content-based Recommendations

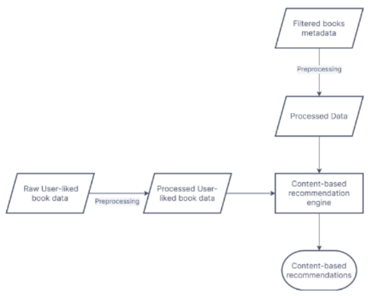
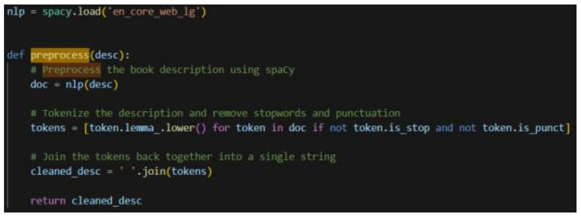

Filtered books metadata
Preprocessing
Processed Data
Raw User-liked book data
Preprocessing
Processed User-liked book data
Content-based recommendation engine
Content-based recommendations

Figure 10: Flowchart to show content-based recommendation generation.

Before generating content-based recommendations, we needed to preprocess the book descriptions in our book metadata in order to generate a book-vector graph using NLP, so that we can easily find similar books based on cosine similarity between the user-liked booked generated vector and the book metadata book-vector.

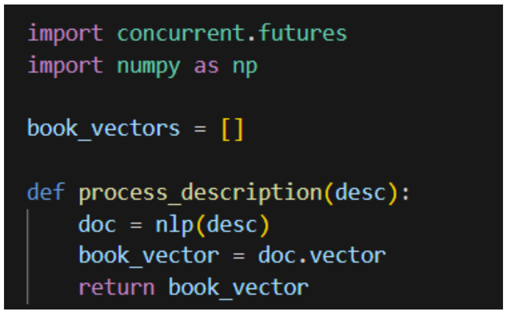
So, first we clean and preprocess the data to ensure consistency and remove noise. This involved handling missing values, standardizing formats, removing duplicates, and performing other data cleaning techniques. We used Spacy's inbuilt library to tokenize the book description, remove stop words and convert it all to lower case [5].



```
nlp = spacy.load('en_core_web_lg')


def preprocess(desc):
    # Preprocess the book description using spaCy
    doc = nlp(desc)

    # Tokenize the description and remove stopwords and punctuation
    tokens = [token.lemma_.lower() for token in doc if not token.is_stop and not token.is_punct]

    # Join the tokens back together into a single string
    cleaned_desc = ' '.join(tokens)

    return cleaned_desc
```

Figure 11: Code for preprocessing book descriptions

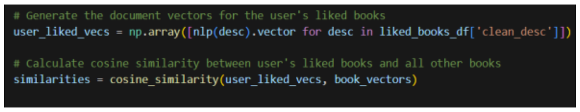
Now, the next step to come up with descriptions was to vectorize the description of each book with an NLP model so that the numerical values hence generated can be compared to the vector similarly generated of user-liked books. Vectorization is performed to convert textual data (book descriptions) into numerical representations that can be understood and processed by machine learning algorithms. Vectorization enables us to measure the similarity between book descriptions using mathematical operations. By representing the descriptions as vectors, we can calculate distances or similarities between vectors to identify similar books.



```
import concurrent.futures
import numpy as np

book_vectors = []

def process_description(desc):
    doc = nlp(desc)
    book_vector = doc.vector
    return book_vector
```

Figure 12: Code showing generated book-vector

Similarly, from our user-liked book data we can generate a vector of user liked books and compare the two vectors to find such similar books which the user may like with the help of cosine similarity. We aggregate the first 100k most similar books into our final recommendation dataframe.



```
# Generate the document vectors for the user's liked books
user_liked_vecs = np.array([nlp(desc).vector for desc in liked_books_df['clean_desc']])

# Calculate cosine similarity between user's liked books and all other books
similarities = cosine_similarity(user_liked_vecs, book_vectors)
```

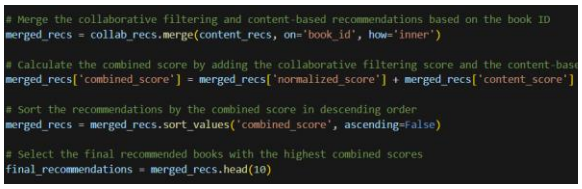
Figure 13: Calculating cosine similarity between user-liked and book vectors.

We then input the cosine similarity scores in a new column 'content_score' to make combining scores with our earlier generated scores from collaborative filtering recommendations easier.

Spacy was used for this part of our system because of its comprehensive set of tools and functionalities for NLP tasks, making it a preferred choice for working with text data. Its ease of use, efficiency, language support, and integration with deep learning frameworks make it a valuable tool in the NLP ecosystem.

c. Combining scores to generate hybrid-recommendations

The recommendations are merged based on the book ID, and a combined score is calculated by adding the collaborative filtering score and the content-based score. The recommendations are then sorted based on the combined score in descending order, and the top n-recommendations books are selected as the final recommendations.



```
# Merge the collaborative filtering and content-based recommendations based on the book ID
merged_recs = collab_recs.merge(content_recs, on='book_id', how='inner')

# Calculate the combined score by adding the collaborative filtering score and the content-base
merged_recs['combined_score'] = merged_recs['normalized_score'] + merged_recs['content_score']

# Sort the recommendations by the combined score in descending order
merged_recs = merged_recs.sort_values('combined_score', ascending=False)

# Select the final recommended books with the highest combined scores
final_recommendations = merged_recs.head(10)
```

Figure 14: Merging two scores to generate combined score and provide recommendations.

CONCLUSION

Finally, this work has investigated the field of recommendation systems with a focus on Spacy-based techniques. We began by introducing Spacy, a sophisticated natural language processing (NLP) package, and emphasising its importance in a variety of NLP applications. We then looked at other types of recommendation systems, such as collaborative filtering, content-based filtering, and hybrid approaches that combine the strengths of numerous strategies.

The literature review presented in this paper shed light on the existing body of work surrounding Spacy-based recommendation systems. It highlighted the advancements made in this field, showcasing the effectiveness of leveraging Spacy's NLP capabilities in generating accurate and personalized recommendations.

Furthermore, this study presented a unique Spacy-based hybrid book recommendation engine. The engine generates individual recommendation ratings by leveraging collaborative filtering and content-based filtering approaches, as well as Spacy and NLP. These scores are then merged to generate hybrid recommendations, which provide users with a more comprehensive and refined set of ideas.

To summarize, this research paper demonstrates the value and potential of Spacy-based recommendation systems in providing enhanced and personalized recommendations. The integration of Spacy's NLP capabilities with collaborative and content-based filtering approaches has the potential to revolutionize the recommendation landscape, enabling more accurate and contextually relevant suggestions for users. As the field of recommendation systems continues to evolve, further exploration and research in this area will undoubtedly lead to even more sophisticated and effective Spacy-based recommendation engines.

REFERENCES

[1] Su, Xiaoyuan, and Taghi M. Khoshgoftaar. "A survey of collaborative filtering techniques." *Advances in artificial intelligence* 2009 (2009).

[2] Lops, Pasquale, Marco De Gemmis, and Giovanni Semeraro. "Content-based recommender systems: State of the art and trends." *Recommender systems handbook* (2011): 73-105.

[3] Burke, R. (2002). "Hybrid Recommender Systems: Survey and Experiments." User Modeling and User-Adapted Interaction, 12(4), 331-370.

[4] Covington, Paul, Jay Adams, and Emre Sargin. "Deep neural networks for youtube recommendations." Proceedings of the 10th ACM conference on recommender systems. 2016

[5] Vasiliev, Yuli. *Natural language processing with Python and spaCy: A practical introduction*. No Starch Press, 2020.

[6] Wan, Mengting, et al. “Fine-Grained Spoiler Detection from Large-Scale Review Corpora.” *ACL Anthology*, 31 May 2019, aclanthology.org/P19-1248.

[7] Mengting Wan and Julian McAuley. 2018. Item recommendation on monotonic behavior chains. In Proceedings of the 12th ACM Conference on Recommender Systems (RecSys '18). Association for Computing Machinery, New York, NY, USA, 86–94. https://doi.org/10.1145/3240323.3240369

[8] Singhal, Amit. *Modern Information Retrieval: A Brief Overview*. singhal.info/ieee2001.pdf.