

People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
Mohamed El Bachir El Ibrahimi University of Borj Bou Arréridj
Faculty of Mathematics and Informatics
Informatics Department



DISSERTATION

Presented in fulfillment of the requirements of obtaining the degree

Master in Informatics

Specialty: Networking And Multimedia

THEME

The design of a book recommender system

Presented by:

Yakoub SAADI

Aymen CHRAF

Publicly defended on: 10/06/2023

In front of the jury composed of:

President:

Examiner:

Supervisor: Dr. Nadjib BENAOUA

2022/2023

Dedication

To our parents, siblings, and friends.

We dedicate our heartfelt gratitude.

Acknowledgment

I am immensely grateful for the help and support I received throughout this thesis. Thank you, Allah, for your blessings. To my father, your hard work inspired me. Mother, your love fueled my determination. Sisters, your support kept me going. Friends, our joyful moments were priceless. To all who aided my studies during the past years, thank you. And to my teacher Dr. Nadjib BENAOUA, your guidance was invaluable during the preparation of this thesis.

To each one of you, Your presence in my life has been a true blessing, and I am honored to have had you by my side.

AYMEN CHRAF

I would like to express my sincere gratitude and appreciation to Allah for the blessings and guidance throughout this journey. I am also deeply thankful to my family for their unwavering support and encouragement. My loving father, mother, and brothers have always been my pillars of strength, providing me with the necessary motivation and belief in my abilities. I am immensely grateful to my teacher, Dr. Nadjib BENAOUA, for his invaluable guidance and mentorship. His expertise and dedication have played a pivotal role in shaping the success of my project. Additionally, I would like to acknowledge the contribution of my dear friends. Their constant presence, joy, and shared moments have made this journey truly memorable and enjoyable.

YAKOUB SAADI

abstract

Recommender systems have revolutionized the way we navigate through an abundance of options in various domains. Book recommendation systems, in particular, have gained significant attention due to their potential to assist avid readers in discovering relevant and personalized book suggestions. This thesis focuses on designing a hybrid book recommendation system by merging three sub-recommendation systems: memory-based collaborative filtering, content-based filtering, and model-based collaborative filtering approaches. The objective is to improve the accuracy and quality of book recommendations. A comprehensive methodology was employed, including an overview, system design, implementation, and evaluation using a famous dataset. The proposed hybrid model was rigorously validated and compared against other commonly used algorithms. Results revealed the superiority of the hybrid system in terms of recommendation quality and relevance. This research contributes to the field of recommender systems and highlights the potential of hybrid approaches for enhancing personalized book recommendations.

Keywords: Book Recommender systems, Content-based filtering approach, Collaborative-based filtering approach, Recommender system evaluation.

Résumé

Les systèmes de recommandation ont révolutionné la façon dont nous naviguons parmi une multitude d'options dans divers domaines. Les systèmes de recommandation de livres en particulier, ont attiré une attention considérable en raison de leur capacité à aider les lecteurs avides à découvrir des suggestions de livres pertinentes et personnalisées. Cette thèse porte sur la conception d'un système hybride de recommandation de livres en fusionnant trois sous-systèmes de recommandation: le filtrage collaboratif, le filtrage basé sur le contenu et les approches de recommandation basées sur des modèles. L'objectif est d'améliorer la précision et la qualité des recommandations de livres. Une méthodologie complète a été utilisée, comprenant une vue d'ensemble, la conception du système, la mise en œuvre et l'évaluation à l'aide d'un ensemble de données célèbres. Le modèle hybride proposé a été rigoureusement validé et comparé à d'autres algorithmes couramment utilisés. Les résultats ont révélé la supériorité du système hybride en termes de qualité et de pertinence des recommandations. Cette recherche contribue au domaine des systèmes de recommandation et met en évidence le potentiel des approches hybrides pour améliorer les recommandations de livres personnalisées.

Mots clés: systèmes de recommandation de livres, filtrage basé-contenu, filtrage collaboratif, évaluation des systèmes de recommandation.

ملخص

أحدثت أنظمة التوصية ثورة في الطريقة التي نختار بها عند وجود وفرة من الخيارات المتاحة في مختلف المجالات. اكتسبت أنظمة التوصية بالكتب على وجه الخصوص، اهتماما كبيرا نظرا لقدرتها على مساعدة القراء المتعطشين في اكتشاف الكتب التي توافق اهتماماتهم. تركز هذه الأطروحة على تصميم نظام توصية هجين للكتب عن طريق دمج ثلاث أنظمة توصية فرعية: التصفية التعاونية المستندة إلى الذاكرة، والتصفية القائمة على المحتوى، والتصفية التعاونية المستندة إلى النموذج. الهدف هو تحسين دقة وجودة توصيات الكتب الموصى بها. لذلك، تم استخدام منهجية شاملة تشمل نظرة عامة، تصميم النظام، تجسيده، و تقييمه باستخدام مجموعة بيانات مشهورة. تم التحقق من صحة النموذج الهجين المقترح بدقة ومقارنته بخوارزميات أخرى شائعة الاستخدام. أظهرت النتائج تفوق النظام الهجين من حيث جودة التوصيات ومدى ملاءمتها. يندرج هذا البحث ضمن مجال أنظمة التوصية ويسلط الضوء على إمكانات الأنظمة الهجينة لتعزيز توصيات الكتب الشخصية.

كلمات مفتاحية: أنظمة توصية الكتب، نهج التصفية المبني على المحتوى، نهج التصفية المبني على التعاون، تقييم نظام التوصية

Table of contents

List of Figures	xi
List of Tables	xii
List of Algorithms	xiii
1 General Introduction	1
1.1 Context	1
1.2 Objectives	1
1.3 Methodology and results	2
1.4 Report outline	2
2 Overview of Recommender Systems	3
2.1 Introduction	3
2.2 Definition	3
2.3 Implicit and Explicit Feedback	5
2.3.1 Explicit Feedback	5
2.3.2 Implicit Feedback	5
2.4 Basic Models of Recommender Systems	6
2.4.1 Content-based Recommender Systems	6
2.4.2 Collaborative Filtering Models	7
2.4.2.1 Model-Based Filtering Technique	7
2.4.2.2 Memory-Based Filtering Technique	9
2.4.3 Hybrid Recommender Systems	10
2.5 Similarity Measurement Approaches	11

2.5.1	Cosine similarity	11
2.5.2	Jaccard Similarity	12
2.6	Evaluation of Recommender Systems	12
2.7	Issues and Challenges	16
2.7.1	The cold start problem	16
2.7.1.1	The item cold start	17
2.7.1.2	The user cold start	17
2.7.2	The gray sheep	18
2.7.3	The sparsity problem	18
2.8	Conclusion	19
3	Book Recommender Systems: State of the Art	20
3.1	Introduction	20
3.2	Importance of book recommender systems	21
3.3	A classification of book recommender systems	21
3.3.1	Content-Based Recommender Systems	22
3.3.2	Collaborative Filtering Recommender Systems	22
3.3.3	Knowledge-Based Recommender Systems	22
3.3.4	Hybrid Recommender Systems	23
3.3.5	Demographic-Based Recommender Systems	23
3.3.6	Context-Aware Recommender Systems	24
3.3.7	Community-Based Recommender Systems	25
3.3.8	Stylometry-based Recommender Systems	25
3.4	Examples of online book recommender systems	26
3.4.1	Goodreads	27
3.4.2	LibraryThing	27
3.4.3	BookBub	28
3.4.4	What Should I Read Next	29
3.5	Datasets for evaluating book recommender systems	29
3.5.1	Goodreads	30
3.5.2	Amazon Reviews	30
3.5.3	Book-Crossing Dataset	30
3.5.4	LibraryThing	31

3.5.5	University of California San Diego Book Graph:	31
3.5.6	Project Gutenberg Dataset	31
3.5.7	Open Library Dataset	31
3.6	Conclusion	31
4	Our Recommendation System Design	33
4.1	Introduction	33
4.2	The proposed recommendation approach	33
4.3	Content-based approach	34
4.3.1	Genre similarity	35
4.3.2	Author similarity	37
4.3.3	Release year difference similarity	39
4.3.4	Overall similarity	40
4.3.5	Rating prediction approach	42
4.3.6	Advantages and limitations of our content-based approach	44
4.4	KNNm-based approach	45
4.4.1	Similarity matrix calculation	46
4.4.2	Rating prediction approach	46
4.4.3	Advantages and limitations of our KNNm-based approach	48
4.5	SGD-based approach	48
4.5.1	Rating prediction and error calculation approach	49
4.5.2	Matrixes training approach	50
4.5.3	Advantages and limitations of our SGD-based approach	51
4.6	Hybridization approach	52
4.6.1	Hybrid prediction approach	52
4.6.2	Top N Recommendation list creation	52
4.7	Conclusion	53
5	Experimental Evaluation	54
5.1	Introduction	54
5.2	Experimental environment	55
5.2.1	Google Colab	56
5.2.2	Python	56

5.2.3	Surprise Library	56
5.3	Validation methodology	57
5.4	Recommendation algorithms implementation	58
5.4.1	The DataProvider Class	58
5.4.2	The CBAAlgorithm Class	59
5.4.3	The CoAlgorithm Class	60
5.4.4	The SGDAAlgorithm Class	61
5.4.5	HybridAlgorithm Class	62
5.5	The used dataset	63
5.6	Evaluation Implementation	64
5.6.1	EvaluationData class	64
5.6.2	RecommenderMetrics class	65
5.6.3	EvaluatedAlgorithm class	67
5.6.4	Evaluator class	68
5.6.5	Evaluation	69
5.7	Results discussion	70
5.7.1	The sub-recommender systems vs the hybrid recommender system . . .	70
5.7.2	The hybrid recommender system vs other known recommender systems.	72
5.8	Conclusion	75
6	General Conclusion	76
6.1	Contributions	76
6.2	Limitations	76
6.3	Future Work and Perspectives	77
	References	77

List of Figures

2.1	Logical process of a recommendation system.	4
2.2	Popular strategies used by recommender systems.	6
2.3	Content-based filtering recommender system logical process.	7
2.4	Matrix Factorization Example.	8
2.5	Collaborative filtering recommender system logical process.	9
2.6	Hybrid recommender system logical process.	10
2.7	The Cold Start Problem.	16
2.8	The Gray Sheep Problem.	18
4.1	Converting genres list into vectors.	35
4.2	A curve showing the decreasing similarity with the increase in years difference.	40
4.3	Rating prediction in SGD-based algorithm.	49
5.1	Our recommender system repository link.	55

List of Tables

5.1	Performance Metrics results of our sub-algorithms and the hybrid algorithm. . .	70
5.2	results of Performance Metrics of the hybrid recommender system and other known recommender systems.	73

List of Algorithms

1	Genre similarity calculations.	36
2	Author similarity calculations.	38
3	Year similarity calculations.	39
4	Compute similarities between all pairs of items in content-based.	42
5	Rating prediction for content based.	44
6	Rating prediction for KNNm.	47
7	Matrix factorization training with SGD.	51
8	Predict all ratings for a reader in each algorithm.	52
9	Combine predictions from multiple recommenders.	53

Chapter 1

General Introduction

1.1 Context

In the era of excessive information and abundant choices, recommender systems play a crucial role in our everyday lives. They have become essential tools for easing the burden of choice by offering personalized recommendations that enhance our overall experience. One particular domain that greatly benefits from recommender systems is book recommendations. These systems cater to avid readers by providing tailored suggestions that align with their interests, fostering engagement and enriching their reading experiences.

1.2 Objectives

The objective of this thesis is to **design a book recommendation system** that enhances the user experience by accurately suggesting books based on their individual preferences. The primary problem addressed is the challenge of designing an appropriate recommendation algorithm that takes into account the uniqueness of book recommendations while avoiding the individual weaknesses of different types of recommendation systems. We aim to develop a recommendation algorithm that outperforms existing algorithms in terms of accuracy and quality.

1.3 Methodology and results

To achieve our objectives, we propose a **hybrid recommendation system** that merges three sub-recommendation systems tailored specifically for book recommendations. These sub-systems incorporate collaborative filtering, content-based filtering, and model-based approaches to capture different aspects of reader preferences and book characteristics. By combining these techniques, we aim to leverage the strengths of each approach and overcome their limitations, resulting in a more robust and accurate recommendation system.

To validate the effectiveness of our proposed solution, we conducted a comprehensive evaluation using a famous dataset of user book ratings. We compared the performance of our hybrid algorithm against the sub-recommendation algorithms. Then, we compared it with several commonly used algorithms, including **KNN** and **NMF**.

Remarkably, our hybrid recommendation system demonstrated superior performance, outperforming the alternative algorithms in terms of recommendation quality and relevance. These promising results indicate the potential of hybrid approaches in the realm of book recommendation systems.

1.4 Report outline

This report contains, in addition to this current chapter, the following five other chapters:

- **Chapter two** : An introduction to recommendation systems in general, their types, data analysis method in each of them, and methods of evaluating their performance.
- **Chapter three** : In this chapter, we delve deeper into book recommendation systems, the studies conducted in this field, and the progress made in it.
- **Chapter four** : This chapter details our design of the sub-recommendation systems, and how they were combined to obtain the hybrid recommendation system solution.
- **Chapter five** : Here, we apply our algorithm, then test it and discuss the results and how it outperforms other algorithms.
- **Chapter six** : A general conclusion to this report.

Chapter 2

Overview of Recommender Systems

2.1 Introduction

In our modern world, we notice that Recommendation systems are everywhere, starting from social media, marketing, education, and more. The types of applications used for recommendation are the ones that might perform content detection tasks or filter the content to provide relevant information[1][2]. Recommender systems find **relationships** between users and items based on actions. There are three commonly used recommendation system types: **content-based**, **collaborative filtering**, and **hybrid**[2][3][4]. We will talk about every type later, but first, let us get some definitions of a Recommender System.

2.2 Definition

To simplify things as much as possible, we can say that a recommender system is an algorithm that proposes ratings based on pre-rated content and the similarity between users behavior. This content could be books, music, movies ... etc. And the behavior could be past purchases, search history, demographic information, ratings, and other factors as shown in figure 2.1. Here are some more advanced definitions:

- Éric Gaussier and Esther Pacitti mentioned that recommender systems handle the problem of **overload of information** that users find by providing them with personalized and targeted content [5].

- Frank Kane defines a recommendation engine as a system that **predicts ratings or preferences** that a user might give to an item. So, a recommender system is recommending things to people based on their past behavior and the behavior of other people. It's about recommending stuff to people based on their preferences that we learn over time [2].
- Kim Falk defines a recommendation system as a system that calculates and provides relevant content to the user based on knowledge of the user, content, and interactions between the user and the item [1].

Frank Kane focused on **rating prediction**, while Kim Falk focused on **relevant content** directly. All of the past definitions focused on:

- Helping the user decide.
- Helping the user compare different items with different properties.
- Helping the user discover new categories and items.
- Helping the user explore items similar to previously purchased items.

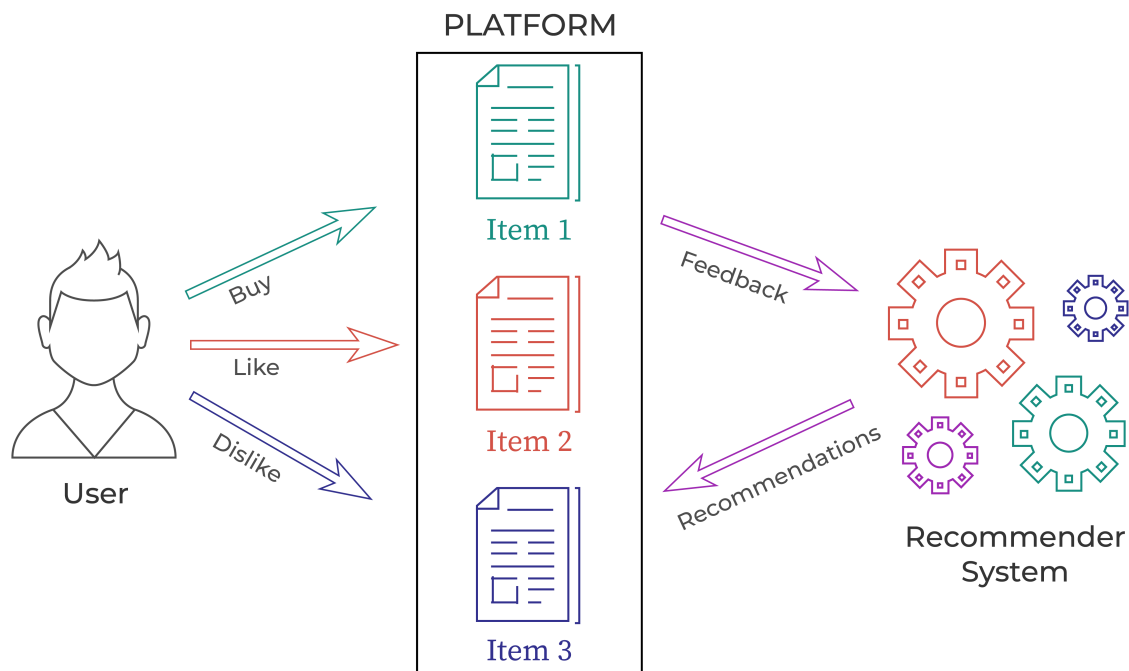


Figure 2.1: Logical process of a recommendation system.

Mostly we find recommender systems on e-commerce sites, as they generate billions of profits for companies annually. That is because the users have access to the items they want and there-

fore are more likely to purchase or watch more content and ads[6]. According to **NVIDIA**, an intelligent recommender system delivers a **22.66%** lift in conversion rates for web products[7]. this considerable number gives us a vision of how important recommender systems are in the modern commercial field.

2.3 Implicit and Explicit Feedback

Implicit and **explicit** feedback are two types of data commonly used in recommender systems to understand user preferences and make accurate recommendations[2][1][8]:

2.3.1 Explicit Feedback

Explicit feedback represents **user input** about their interest in a product. This is the most useful information because it comes straight from the user and demonstrates their direct interest in the item. To get explicit input from users, the system must ask them to rate items. After gathering input, the system determines how relevant or similar an item is to the tastes of the users. Even though this allows the recommender to understand the user's actual opinion, it is typically difficult to gather.

As a result, there are several methods for gathering user input. Implementing a like and dislike functionality allows people to simply evaluate the material. Alternatively, the system can prompt users to enter ratings, on which a scale shows how the user liked or disliked the material.

2.3.2 Implicit Feedback

Implicit feedback requires no user participation and can be generated by observing **user behavior** such as which item they viewed, where they clicked, which things they purchased, or how long they spent viewing a specific type of items. The right activities to track must be determined based on the domain in which the recommender system operates.

Recommender systems often leverage a combination of implicit and explicit feedback to gain a more comprehensive understanding of user preferences. By analyzing both types of feedback, recommender systems can generate more accurate and personalized recommendations, improving user satisfaction and overall system performance.

2.4 Basic Models of Recommender Systems

Recommender Systems can be **Personalized (PRS)** or **Group-Based (GRS)**[9]. In the real world, we frequently encounter Personalized Recommender Systems, particularly on e-commerce sites because they anticipate things for an individual while taking that individual's preferences into account. Meanwhile, Group Recommender Systems make recommendations for a group of users, with each member of the group having their preferences, which makes it hard to manage since we must please every member of the group. There are three popular strategies used by recommender systems to generate recommendations[1][2][6] as shown in the following figure:

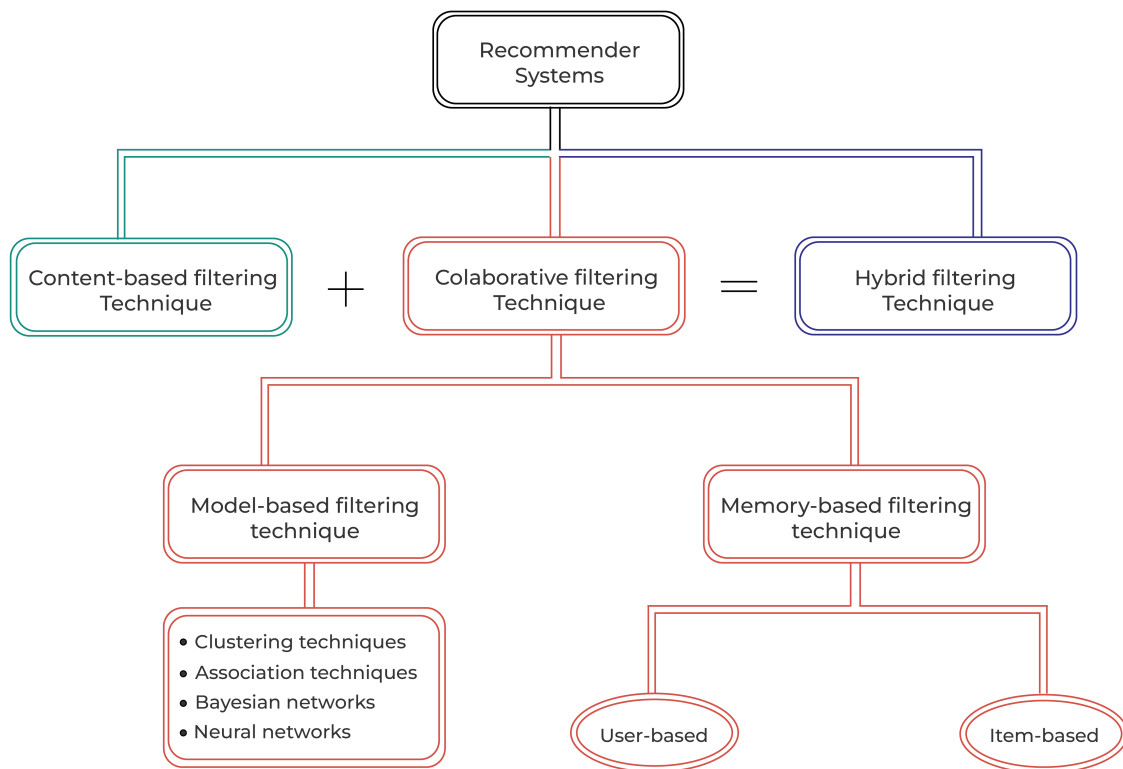


Figure 2.2: Popular strategies used by recommender systems.

2.4.1 Content-based Recommender Systems

The fundamental premise of Content-based filtering is that **if the user enjoys one item, they'll likely want a related one** as shown in figure 2.3. The recommender system will use the user's previous item metadata to produce recommendations. Content-Based filtering techniques leverage content information about individuals and items. Features, textual descriptions, and tags are just a few examples of how this item might be presented. The key concept is to match

user profiles with item profiles based on user preferences for item qualities to offer appropriate things.

As a result, users are given recommendations for products that are similar to those they have interacted with. Extracting pertinent data from the content of items, creating user and item profiles, and filtering items based on profile similarities are all necessary steps in deploying a **Content Based Filtering** method. However, in order to use this strategy, previous user activity is required.

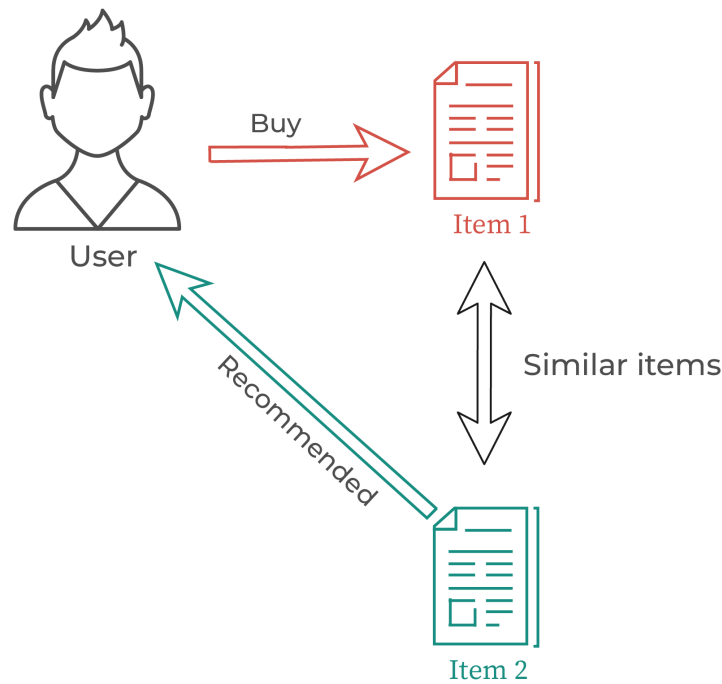


Figure 2.3: Content-based filtering recommender system logical process.

2.4.2 Collaborative Filtering Models

Collaborative filtering can be of two Categories[6]:

2.4.2.1 Model-Based Filtering Technique

Model-based filtering techniques revolutionize the way recommendations are generated. Instead of relying on explicit knowledge about item characteristics or user profiles, these techniques depend on clustering techniques and neural networks. They leverage mathematical models and computational algorithms to discover hidden patterns and relationships within user-item interaction data. By uncovering latent factors or features that represent user preferences

and item characteristics, these models can make accurate predictions and recommendations. Through the power of clustering techniques and neural networks, model-based filtering techniques transform raw data into valuable insights, enabling personalized and tailored recommendations that enhance user satisfaction. With their ability to uncover nuanced preferences and discover new item associations, these techniques pave the way for improved recommendation systems in various domains.

One popular type of model-based filtering technique is **matrix factorization**. Matrix factorization models aim to decompose the user-item interaction matrix into lower-dimensional latent factor matrices. The following figure represents an example of a matrix factorization method, where the rating's matrix is converted into two matrixes that contain three factors[10].

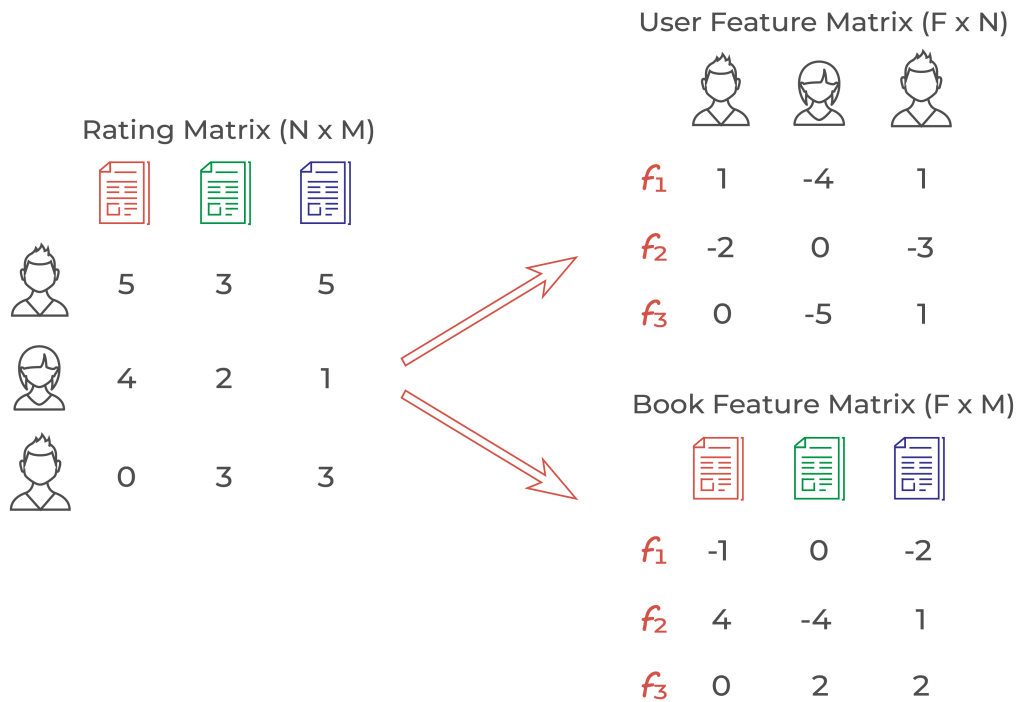


Figure 2.4: Matrix Factorization Example.

One of these matrixes is for users and the other is for items. This decomposition allows the models to capture the underlying patterns and relationships in the data. **Singular Value Decomposition (SVD)** and **Alternating Least Squares (ALS)** are commonly used matrix factorization techniques[11].

2.4.2.2 Memory-Based Filtering Technique

Unlike model-based filtering techniques that rely on mathematical models and computational algorithms, memory-based filtering techniques directly use the **available user-item interaction data** to make recommendations. As shown in the following figure, these techniques store and analyze the historical interactions between users and items to identify patterns and similarities.

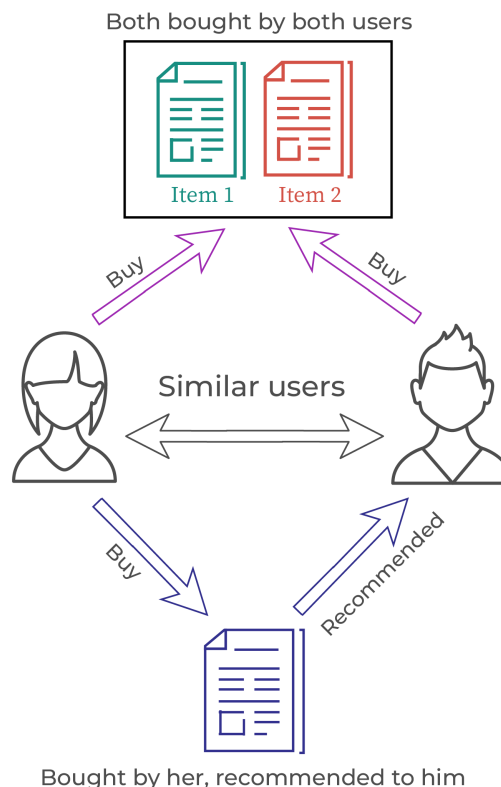


Figure 2.5: Collaborative filtering recommender system logical process.

There are two main types of memory-based filtering techniques:[1]:

- **User-Based Collaborative filtering**

It is a technique that can filter out items that a user may be interested in **based on the reactions of other users**. It operates by searching a huge group of individuals for users who have similar interests to a specific user and considers the items they like and combines them to get a ranked list of recommendations.

- **Item-Based Collaborative filtering**

Which suggests an item based on items the user has **previously bought or liked**. It recommends items similar to the items the user has already liked. In simple terms, Item-Based Collaboration is concerned with other users' **actions on the item** you are considering or purchasing, and the product's attributes are unimportant in recommending. Amazon invented item-based collaborative filtering. It is faster and more stable than user-based filtering in systems with more people than objects. It is effective because the average rating given to an item usually does not change as quickly as the average rating given by a user to other items[12].

2.4.3 Hybrid Recommender Systems

A Hybrid Recommender System is a blend of content and collaborative filtering methods. Combining them may help overcome the shortcomings of utilizing them independently, and may also be more efficient in some situations[1][2][13]. Hybrid Recommender Systems can be implemented in a variety of ways, like using content and collaborative-based methods to provide predictions separately and then combining the predictions, or simply adding method capabilities to the other as shown in the following figure.

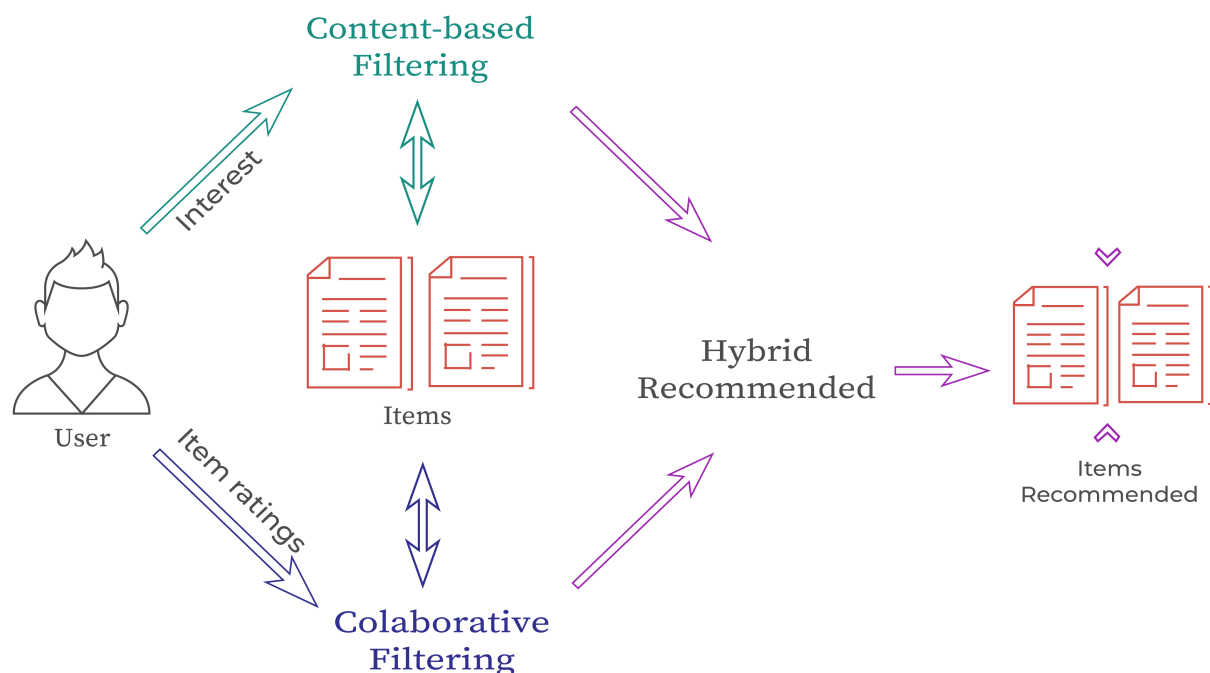


Figure 2.6: Hybrid recommender system logical process.

Several studies compare the performance of conventional procedures to hybrid methods and conclude that utilizing hybrid methods results in better accurate recommendations[4].

2.5 Similarity Measurement Approaches

A recommendation system's ability to learn is primarily based on its ability to analyze data. In memory-based systems, the important criterion that a recommendation system must measure and analyze is **similarity**. The recommendation system will not be able to give accurate suggestions if it is not sufficiently aware of the similarity between both the items and the users.

To simplify things, we can say that users are the same if they rate the same items with a similar rating, or watch similar content for the same time amount. Whereas we can say that two items are similar if they share similar characteristics or features as two books that tell the same historical story, two products that do the same job, or even two movies from the same series.

When designing a recommendation system, we deal with many types of data. For example, there are **unary data**, such as a previously purchased product or a previously visited location, and there is **binary data**, such as likes and dislikes for an item[1]. Other kinds of data include the values of ratings for an item, the characteristics of an item, and information generated from implicit reactions, such as minutes of watching. We will also deal with **data sets** of various designs. As a result, there is no fixed or optimal equation for calculating similarity.

Choosing a similarity equation is determined by the nature of the recommendation system and its work, as well as the data it will be dealing with. Based on this, we'll go over the most well-known similarity equations and their applications[1][3].

2.5.1 Cosine similarity

We start with Cosine similarity, which is defined as **the cosine of the angle between two vectors** and is used to determine the distance between two locations in space. The cosine similarity metric is based on cosine principles, and as the distance between two points increases, the similarity of those points decreases[3][4].

Cosine similarity is utilized in recommender systems with the same idea as cosine angles, where even though the similarity of the content is low, it is regarded as the least recommended

content, and for greater similarity of contents, the recommendations created are at the top.

2.5.2 Jaccard Similarity

This similarity equation was created to determine how similar two sets are. It is also known as the "**Jaccard index**" and was invented by **Paul Jaccard**. It is used on data sets that contain unary data, such as viewed movies, purchased items, liked articles, and so on[2][1].

It is simple to compute Jaccard's similarity. To determine how similar two things are, just multiply the number of users who purchased both goods by the number of users who purchased either one or both. Using this method, we may determine the likelihood of purchasing one item if the other was purchased.

2.6 Evaluation of Recommender Systems

We mentioned before that a good recommender system generates good predictions and gives the user a good experience and sales numbers for companies. But the biggest question is how to know if the recommender system generates appropriate recommendations.

There are popular metrics for recommender systems, and they all have their uses. To use these metrics, we must define a methodology for testing our recommender systems, and it can be **online** or **offline**. We are only going to talk about three offline testing methodologies[2][1]:

- **Train Test splits:** First, we split the rating data into a training set and a testing set: usually, **80%** of the data is used as a training set and the rest as a testing set.

After that, the recommender system will be ready to learn the necessary links between things or between users from the training set and can be tested by asking it to make predictions about how a new user in the testing set might rate some item they have never seen before. After performing these operations, we can extract meaningful numbers that help us measure the accuracy of the recommender system.

- **K-Fold Cross Validation:** First, we generate numerous randomly assigned training sets. Each training set is called a **fold** and is utilized to separately train your recommender system.

After that, the accuracy of the resultant systems is measured against your test set. So we receive a score based on how effectively each fold predicts user ratings, and we can average them. This guarantees that you build a recommender system that works for any collection of ratings, not simply the ones in your training set.

- **Leave One Out Cross Validation:**, Here we use the full data set without splitting it. First, we calculate the **top-N recommendation list** for every user and then remove an item from it. After that, during the training process, we let our systems build a top-N recommendation list for each user and check if it contains the removed item. That is why it is called leave one out because it checks the ability of our system to recommend the item that was removed from the top-N list for every user.

And now for the metrics[2][1]. The first metric we are going to talk about is **Mean Absolute Error (MAE)**, where the recommender system proposes ratings of elements that the user has rated before, we calculate the difference between the rating given by the user and the rating produced by the system, and we calculate the sum of these results, and then we divide this total by the total number of ratings. According to the following mathematical equation[14]:

$$\text{MAE} = \frac{1}{|\hat{R}|} \sum_{\hat{r}_{ui} \in \hat{R}} |r_{ui} - \hat{r}_{ui}| \quad (2.1)$$

Where: $\hat{R}$ is the list of all predicted ratings. $\hat{r}$ is a predicted rating. r is the actual rating.

To do that, we use Train Test splits methodology. A small **MAE** number means that the system suggests a rating close to the actual assessment. A lower **MAE** means a better result.

There is a similar metric named **Root Means Square Error (RMSE)**. This time we square the difference, calculate the sum and then calculate its square root. According to the following mathematical equation[14]:

$$\text{RMSE} = \sqrt{\frac{1}{|\hat{R}|} \sum_{\hat{r}_{ui} \in \hat{R}} (r_{ui} - \hat{r}_{ui})^2} \quad (2.2)$$

Where: $\hat{R}$ is the list of all predicted ratings. $\hat{r}$ is a predicted rating. r is the actual rating.

RMSE is more popular than MAE because it penalizes you more for big mistakes, which

makes it more reliable. A lower **RMSE** means a better result.

Getting accurate assessments doesn't always mean that recommendation systems are doing their job as they should. In the end, companies discovered that RMSE doesn't mean much in the actual world. The expected ratings of certain elements are not so much of interest as the items you provide to users in a **top-N recommender list**, and how users react to those items when proposed.

Now we will talk about a few metrics focused on top-N recommender lists, based on Leave One Out Cross Validation, starting with the one named **Hit Rate**. For each user, the recommender system generates a list containing several top recommendations, the list is called top-N recommendations. If one item on that list is already rated by the user, we consider that a hit. Finally, we sum up the number of hits for each user and divide it by the number of users to get **the hit rate**[2].

$$HR = \frac{Hits}{Users} \quad (2.3)$$

This metric can be used to create more advanced metrics including:

- **Average Reciprocal Hit Rate (ARHR):** We don't just focus on the number of hits but also on their quality since it accounts for where your hits show up in the top-N list. As a result, recommending an item in the top slot earns you more points than recommending an item in the bottom slot. Then we calculate the sum of the ranks of hits instead of just the number of hits, and we divide it by the number of users. A higher **ARHR** means a better result.

$$ARHR = \frac{\sum_{i=1}^n \frac{1}{rank_i}}{Users} \quad (2.4)$$

Where: *rank* is the rank of the item.

- **Cumulative Hit Rate (CHR):** The only difference from the hit rate is that we subtract the ratings below a given threshold. A higher **CHR** means a better result.

$$CHR = \frac{Hits_{r < threshold}}{Users} \quad (2.5)$$

Where: *r* is the rating of the removed item.

So far, we've discussed methods for measuring our recommender system's accuracy, but accuracy is not everything when measuring the performance of a recommendation system. There is also[2][1]:

- **Diversity.** It is the average similarity between recommendation pairs subtracted from one. We get the average similarity by calculating the similarity of each potential pair in a top-N recommendations list and then averaging them. We cannot say that we need a high or low Diversity value, as it changes depending on the nature of the recommender system and what we are trying to accomplish.

$$\text{Diversity} = 1 - S \quad (2.6)$$

Where: S is the average similarity between recommendation pairs

- **Coverage:** Which is the system's ability to recommend good items to users. It indicates how many users get appropriate items for them. It is calculated by dividing the number of users who get recommendations that exceed a certain value by the total number of users.

$$\text{Coverage} = \frac{U\hat{s}ers}{Users} \quad (2.7)$$

Where: $U\hat{s}ers$ is the users who get at least one good recommendation.

- **Novelty:** It is the average value of how popular the proposed items are. It aims to find the point of balance between items the user has never seen before and items that are familiar and popular since they establish **User Trust**. Since most sales originate from a small number of products, Novelty also helps expose items in what is known as the **Long Tail**, which creates a large number of sales when combined.

$$\text{Novelty} = \frac{\sum_{i \in R} rank_i}{R} \quad (2.8)$$

Where: R is the recommendations. $rank$ is the rank of an item.

- **Churn :** Over time, the user's behavior and desires may change. We measure Churn because it allows us to measure the rate at which suggestions change for that user, in other words, how **sensitive** our recommender system is to user behavior. If the suggestions

change dramatically after the user has evaluated new items, our churn score will be high. Again, we can't say that the rise or fall of this value is a good or a bad thing. It depends on the nature of the recommender system and its objectives.

2.7 Issues and Challenges

The recommendation system industry is complex work, so it is natural to face problems when we try to design one. Recommendation systems in general suffer from problems that vary in severity and type according to the nature of the recommendation system and the goals manufactured by it, but there are some common problems between them all.

2.7.1 The cold start problem

One of the most common problems is the **cold start** problem[8] as shown in figure 2.7. We know that personalized recommendations are based on information that binds content with users, and recommendation systems require previous evaluations, whether for the user himself or the product to suggest new recommendations. But how will our system deal with new items that have not received any reviews yet or which ratings are few, and also how will our system deal with new users who have not done any behavior or evaluation that allows us to know what they like and do not, this is exactly what the cold start means.

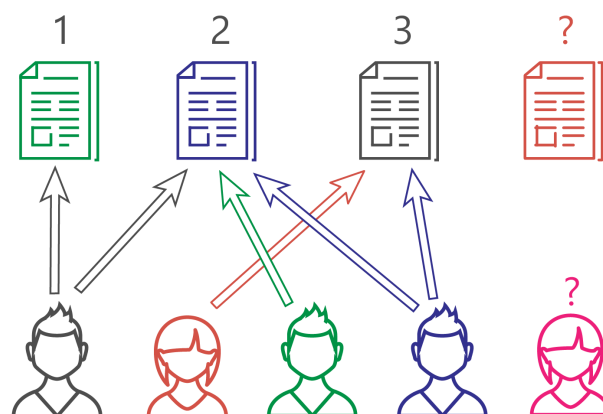


Figure 2.7: The Cold Start Problem.

The cold start is divided into two types[2][1]:

2.7.1.1 The item cold start

As we mentioned before, when you have a new product in the market, the number of users who have already tried it and given it ratings is zero, meaning that we do not have actual and ready reviews from current users.

The cold start can also infect items that have not been adequately evaluated over time. This makes the system unable to create links between this element and the previous elements, as well as between user behaviors. There is no perfect solution to this problem, but there are some actions that can reduce it.

One of them is the use of **content-based recommendation systems**. As we mentioned before, this type of recommendation system is based on suggesting elements by measuring the similarity between them and based on that we will not need the ratings of the former users in the first place. That's why it's a good idea to reserve some slots in the Top N recommendations list only for the suggested items with this type of recommender system.

There is another solution to this problem, which is to put new elements at the **top of the search pages** and **the main pages of the sites**. This will attract some users to test these elements and then evaluate them. Once the users do this, we will get rid of this problem.

Using the appropriate novelty values will contribute significantly to reducing this problem.

2.7.1.2 The user cold start

The recommender system cannot know what to suggest to a user whose behaviors and desires are unknown. In general, this problem is less severe than the previous one, because we can guess the desires and behaviors of the new user in many ways[15].

One of the common solutions is to **ask new users** about their favorite genres and products they like, or even ask them directly to give reviews of products they may have seen before. Tracking new users and focusing on their implicit reactions such as their search history and time spent viewing content, an advertisement, or reading details of a product also gives us insight into their interest in these elements.

Large companies also use **demographic information** to suggest content to new users based on content that is popular in their area, whether that area is a country, a city, or even a town.

These companies may also find ready-made information about new users by reading cookies in their browsers or purchasing it from other companies.

The cold start is not the only problem that you may face when designing a recommender system.

2.7.2 The gray sheep

There is another problem called the **gray sheep**[\[1\]](#) as shown in figure 2.8. Kim Falk defined the gray sheep as a user who has such an individual taste that even if there's data, there are likely no other consumers - or few ones- who've bought any of the products the gray sheep has. In this case, we will face a problem similar to the problem we faced in the cold start, which is that the recommendation system cannot build good relation pairs.

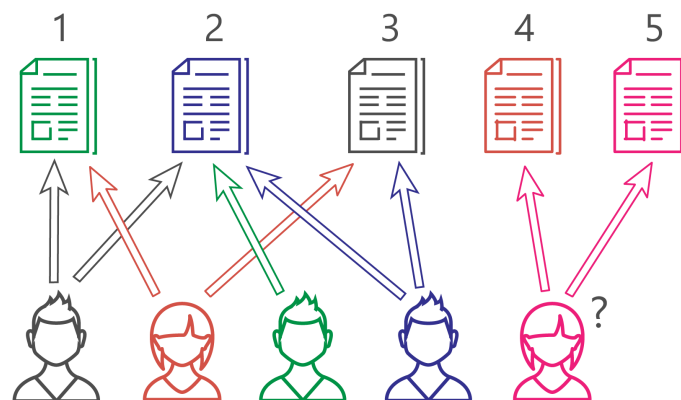


Figure 2.8: The Gray Sheep Problem.

Also, there is no perfect solution for the gray sheep, but the common one is to **use categories for the items**, the category contains items that share some features. That will allow us to get non-accurate but useful information about the desires of the gray sheep.

2.7.3 The sparsity problem

The lack of data is not the only fault in the prediction process. **The sparsity problem**[\[16\]](#) occurs when data dispersion and unpredictability cause difficulty in identifying the user's wishes. This issue arises as a result of consumers failing to rate things they are interested in or have purchased.

These users are unconcerned about rating products they are interested in, and may even post fictitious or false evaluations of items they have not seen or are not interested in.

This behavior scatters information among a variety of individuals and objects, and it can have unintended consequences, as it usually arises with **collaborative recommender systems**. There is no one-size-fits-all solution to the issue, but it is believed that educating users to conduct proper evaluations, making the system adaptable, and not depending primarily on this sort of recommendation system would all help to drastically reduce the problem.

2.8 Conclusion

In summary, we previously discussed recommender systems, including where can they be found, their functionality, and common design challenges. We acknowledged that creating a successful recommender system is a complex task that requires careful consideration of various factors. For this reason, designing such systems can be likened to an art form. It is important to note that each use case of a recommender system is unique, and there are numerous critical factors that must be considered to achieve optimal results. This thesis will focus specifically on book recommender systems, with the aim of developing a recommender system that meets the unique needs and preferences of our target audience.

Chapter 3

Book Recommender Systems: State of the Art

3.1 Introduction

Over the past few years, recommender systems have become increasingly popular across various industries and applications, from e-commerce and entertainment to social media and more. One particularly well-known application of these systems is for **book recommendations**.

With so many titles available to choose from, selecting the next book to read can be a daunting task. However, recommendation systems are designed to make this easier by suggesting books based on the reader's interests. As a result, readers can now easily find books they'll enjoy from an extensive collection of books online.

In this section, we'll delve into the many applications of recommendation systems in the book industry. We'll explore the different types of recommendation algorithms, their uses, and the benefits and drawbacks they offer, as well as the most popular book datasets available. Additionally, we'll examine some of the challenges associated with utilizing recommendation systems in the book industry. But before that, let's discuss why book recommender systems are so important.

3.2 Importance of book recommender systems

Finding the right book to read can be overwhelming for those who love reading. Numerous books exist on the market, and there are various factors to consider, such as genre, author, and format. Many readers may spend hours browsing through bookstores or online libraries without the ability to choose the right book, mainly if they are unfamiliar with the author or the genre they want to read.

Additionally, readers may feel that they do not have enough time to read a lengthy book, or they may be hesitant to invest their time in a book they may not enjoy. This indecision can lead to a reluctance to commit to a book.

To overcome the challenges of choosing the right book, we turned to **book recommender systems**, as they are an exceptional tool for readers to embark on a literary journey that best aligns with their past preferences, reading history, and other salient factors. By meticulously scrutinizing various aspects of books such as genre, author, publication date, ratings, and reviews, these systems can offer personalized recommendations that cater to each user's **unique interests**. The primary objective of book recommender systems is to enhance the overall user experience and enable them to discover books that resonate with their literary preferences.

Moreover, book recommender systems have become increasingly popular among online bookstores, libraries, and other book-related services, as studies showed that they effectively augment engagement, customer loyalty, and sales[17].

In summary, the importance of book recommender systems lies in their ability to help readers discover new books that align with their interests, while also providing publishers and authors with valuable insights into their audience's preferences.

3.3 A classification of book recommender systems

Book recommender systems can be grouped into categories based on their **methods to generate book recommendations**. Knowing the classification of book recommender systems is important for developers, researchers, and users who want to choose the best system for their particular needs. By understanding the different categories of book recommender systems, stakeholders can make informed decisions when selecting or designing a system. In general,

book recommender systems can be classified into a variety of categories that are explained in the following sections[18][19].

3.3.1 Content-Based Recommender Systems

These systems analyze the **characteristics of books**, such as genre, author, and keywords, to recommend books that are similar to the ones that the user has liked in the past. Content-based filtering systems are especially useful when a user has a particular preference for a **specific genre or author**.

A related work to this system is **Goodreads**[20]. Goodreads uses a user's reading history and preferences to recommend books with similar themes, genres, and writing styles. The system also takes into account a user's ratings and reviews of books to provide personalized recommendations.

3.3.2 Collaborative Filtering Recommender Systems

This is the most popular type of book recommender systems, which relies on the preferences of users to generate book recommendations. Collaborative filtering systems analyze **user behavior patterns**, such as ratings and reviews, to identify books that are similar to the ones that the user has enjoyed in the past.

A related work to this system is the book recommendation system used by **Amazon**[21]. It uses collaborative filtering to recommend books to users based on their past purchases and browsing behavior, as well as the behavior of other users who have similar interests. The system analyzes the data from millions of transactions and user interactions to identify patterns and recommend books that other users with similar interests have enjoyed.

3.3.3 Knowledge-Based Recommender Systems

These systems use the user's **explicit preferences** to generate book recommendations, therefore solving the **cold start** problem. They can be useful when dealing with specialized domains or topics since they rely on an expert knowledge base that contains information about the books and their characteristics. This explicit knowledge is used to generate recommendations, instead of relying solely on user behavior data like collaborative filtering or content-based

recommendation systems.

A related work to this system is the **Book Genome Project** by **Bookish**[22]. It recommends books based on a user's preferences, such as favorite authors, genres, and subjects, as well as more specific criteria like the writing style, pacing, and tone. The system uses a database of book characteristics, such as themes, settings, and character types, to match books to a user's preferences.

The **Book Genome Project** also allows users to explore books based on various categories and criteria, such as "books with strong female protagonists" or "books set in a small town."

3.3.4 Hybrid Recommender Systems

These systems combine two or more of the above techniques to generate book recommendations. For example, a hybrid system may use both user preferences and book characteristics to generate more accurate and diverse book recommendations. Hybrid systems are becoming **more popular** in recent years, as they can overcome some of the limitations of individual techniques.

A related work to this system is the book recommendation system used by **Barnes & Noble**[23]. It combines both collaborative filtering and content-based filtering approaches to make personalized book recommendations to users.

The system analyzes a user's purchase history, browsing behavior, and other user interactions to identify patterns and recommend books that other users with similar interests have enjoyed. It also uses content-based filtering to recommend books based on characteristics such as genre, author, and writing style. In addition, the system also takes into account expert reviews and curated lists to make recommendations, making it a hybrid recommender system.

3.3.5 Demographic-Based Recommender Systems

These systems provide book recommendations based on demographic information about the user, such as age, gender, location, and education level. This information is used to infer the user's interests and preferences and to match them with books that are likely to be of interest to them.

Libraries and bookstores commonly use demographic-based book recommender systems to provide personalized recommendations to users. However, this approach has limitations, as demographic information alone may not accurately reflect an individual's preferences and interests.

A related work to this system is the book recommendation system used by **Epic Reads**[24]. This system recommends books to users based on their age and reading preferences. Epic Reads provides book recommendations for three different age groups: **young adults**, **middle grade**, and **children's books**. Within each age group, the system makes recommendations based on a user's preferred genres and topics, as well as their reading level and interests.

For example, the system may recommend books that are popular among teenage girls who enjoy romance and fantasy, or books that are appropriate for middle-grade boys who are interested in science fiction and adventure.

3.3.6 Context-Aware Recommender Systems

These systems consider the **contextual information** surrounding the user's book-related preferences and the book items themselves. This information can include factors such as the user's location, time of day, weather, mood, and social context, as well as book-specific features such as genre, author, and publication date. Context-aware book recommender systems aim to provide more personalized and relevant book recommendations by taking into account the user's current situation and preferences[25].

This type of recommender system is particularly useful in mobile or location-based book recommendation scenarios, where the user's context can significantly impact their reading interests.

A related work to this system is the **BookSeer website**[26]. The website recommends books to users based on their current mood and reading preferences. The system takes into account factors such as the time of day, the weather, and the user's current location to make personalized book recommendations.

For example, on a rainy day, the system may recommend a cozy mystery novel or a romantic comedy, while on a sunny day, it may recommend a light-hearted beach read. The system also

asks users to rate their mood on a scale of 1-10 and to select their preferred reading format, such as e-books or audiobooks, to make more accurate recommendations.

3.3.7 Community-Based Recommender Systems

These systems use information about a user's **social network** to make personalized book recommendations. These systems typically rely on the assumption that people tend to share similar tastes and interests with their friends and acquaintances. Therefore, the system can use information about the books that a user's friends have read, liked, or reviewed to make recommendations that are likely to be of interest to the user.

A related work to this system is the book recommendation system used by **LibraryThing**[\[27\]](#). LibraryThing is a social networking site for book lovers that allows users to catalog their personal libraries, connect with other readers, and share book recommendations. The system recommends books to users based on their reading history, ratings, and reviews, as well as the behavior of other users who have similar interests.

Users can also join groups on the site based on shared interests, such as a group for fans of science fiction or a group for readers who enjoy historical fiction. These groups allow users to connect with other readers who share their interests and discover new books through group discussions and recommendations. Overall, the system uses the **collective knowledge** and **expertise** of the community to make personalized book recommendations.

3.3.8 Stylometry-based Recommender Systems

These systems analyze the **writing style** of a user's previously read content and recommend similar content based on similarities in the writing style. One advantage of stylometry-based recommender systems is that they can provide personalized recommendations based on a user's individual writing style preferences, rather than relying solely on the popularity or rating of content. Additionally, they can be useful in recommending content that is difficult to categorize or has a unique writing style.

A related work to this system is the book recommendation system developed by researchers at the **University of Jaén in Spain**[\[28\]](#). This system recommends books to users based on similarities in the writing styles of different authors. The system uses stylometric analysis,

which involves analyzing the patterns of word usage, sentence structure, and other linguistic features in a text, to identify the unique writing style of each author. The system then compares the writing styles of different authors to identify those who have similar styles.

For example, the system may recommend a book by a relatively unknown author to a reader who enjoys the writing style of a more popular author. This approach is particularly useful for recommending books by **lesser-known** authors who have similar writing styles to more popular authors.

Overall, each type of recommender system has its strengths and limitations. Choosing the best approach depends on various factors, such as the available data, user requirements, the accuracy and coverage of recommendations, and the desired level of personalization. Therefore, no classification can be considered better than another in general. The effectiveness of a recommender system is highly dependent on the specific context in which it is used, and what works for one situation may not be suitable for another.

3.4 Examples of online book recommender systems

Recommendation systems play a pivotal role in the growth of the book market around the world, whether these are e-books or not. The global e-book market was valued at approximately **18.13 billion U.S.** dollars in 2020 and is projected to reach **23.12 billion U.S.** dollars by 2025[29].

These systems not only help in suggesting books, but they do so without the need to go to the library and consult people, as it is enough to connect to the Internet and access one of these services online. This feature has become especially evident after the global Corona epidemic, which, according to a report by the **American Booksellers Association (ABA)**[30], the COVID-19 pandemic has significantly increased online book sales in the United States. In 2020, online book sales saw a **243.7%** increase compared to the previous year.

As a result of the great importance of book recommendation systems from a scientific, pedagogical, and even commercial point of view, many of these systems already exist and help millions of people around the world every day to choose the books they might want. Some of these systems are integrated into corporate business websites for profit purposes.

The most prominent example of this is **Amazon.com**, which is one of the largest on-line booksellers, with around **60%** self-published e-books offered on **Kindle Unlimited** in 2015[31]. AbeBooks is also an online marketplace for books that connects independent book-sellers with buyers.

There are also **Barnes & Noble**, **Book Depository**, and others, all of which use different book recommendation systems to make as many sales as possible. On the other hand, many sites are not intended for profit purposes or are indirectly profitable, where their role is simply to suggest books. We will talk about some of the most famous sites that use book recommendation systems to suggest books and not sell them[32]:

3.4.1 Goodreads

With over **125 million** members, over **2.8 billion** books, and over **1.1 billion reviews**[33], Goodreads is one of the most popular book recommendation websites in the world. It was founded in December 2006 by **Otis Chandler** and **Elizabeth Chandler** and was later acquired by **Amazon** in 2013.

The book recommendations provided by Goodreads are powered by a robust recommendation system that incorporates various architectures, including **demographic**, **content-based**, and **collaborative filtering**. This system is capable of handling massive amounts of data and generating precise predictions. It's worth noting that Goodreads is more than just a website but a whole community for book lovers, users can also join reading challenges and groups, participate in book-related discussions, and follow their favorite authors.

It has a user-friendly design that allows users to easily discover new books and connect with other book lovers. The site has a search function that allows users to find books by title, author, or ISBN, and users can also browse books by genre, lists, and recommendations. That's why this site is one of the most loved sites by book lovers and even those who do not read regularly.

3.4.2 LibraryThing

This is another online service designed for book lovers to catalog their books, connect with other readers, and discover new books based on their interests. It was launched in August 2005 by **Tim Spalding**[34].

LibraryThing allows users to create a virtual bookshelf by adding books to their library. Users can enter book details manually or use the site's barcode scanner to scan books' ISBNs to add them to their library. LibraryThing's catalog contains over **200 million** books and is powered by data from more than **1,000 libraries** worldwide.

This data is used by the book recommendation system. This system analyzes users' book collections and suggests books that they may like based on their genre preferences, author interests, and other factors. Users can also explore recommendations from other users with similar reading tastes and join groups focused on specific genres, themes, or topics.

LibraryThing is a powerful tool for book lovers to catalog their collections, discover new books, and connect with other readers worldwide. That's why it has over **2 million** visits every month worldwide, and the platform continues to grow in popularity among book lovers.

3.4.3 BookBub

This one was founded in 2012 by **Josh Schanker, Nicholas Ciarelli, and Maura Curtin**. Their main goal is to help readers discover great books while also helping authors promote their work to a wider audience. Today, BookBub has millions of subscribers worldwide[35].

BookBub's primary feature that distinguishes it from Goodreads and LibraryThing is its daily email newsletter, which provides personalized book recommendations to subscribers based on their reading preferences.

These emails feature limited-time deals and freebies on e-books, so subscribers can take advantage of book promotions and save money while discovering new authors. To do this, BookBub uses algorithms that analyze each subscriber's reading history, book ratings, and other data points to determine which books they are most likely to enjoy. The system is designed to learn from each subscriber's behavior, so the book recommendations become more accurate and personalized over time.

Overall, BookBub's powerful recommendation system and email service are key features that have helped the platform become a popular destination for readers looking to discover new books and authors.

3.4.4 What Should I Read Next

Here is a popular online platform intended for those who ask **What should I read next ?**. In 2005, after facing the problem of not finding the books she might want easily, **Anne Bogel** decided to create a platform to help other readers overcome this problem[36].

Like **LibraryThing**, **Goodreads**, and **BookBub**, What Should I Read Next's primary feature is its book recommendation service, which allows users to enter the title or author of a book they have enjoyed and receive personalized recommendations based on their reading preferences. The site's algorithms analyze each user's reading history and book ratings to determine which books they are most likely to enjoy. The site also allows users to browse curated book lists and read reviews from other readers.

In addition to its book recommendation service, What Should I Read Next features a **podcast hosted by Anne Bogel**, the site's creator. The podcast features interviews with authors and book industry professionals, as well as discussions of different books and genres. The podcast has become a popular destination for book lovers. These podcasts are also suggested by a recommendation system.

Because of all this, What Should I Read Next is one of the most suggestive book services that have been growing in recent years.

Goodreads, **LibraryThing**, **BookBub**, **What Should I Read Next**, and other online book suggestion services all have a common goal of helping users discover new books to read. All these platforms have designed different systems for recommending books, to suggest books based on a reader's reading history, interests, and ratings. In addition to the ability to create personalized reading lists and track reading progress.

In the next chapters, we will try to design our book recommendation system that can be included in a book recommendation website similar to those we talked about. But first, let's talk about some popular book databases that recommendation systems can use.

3.5 Datasets for evaluating book recommender systems

The accuracy and effectiveness of recommendations produced by recommender systems are directly correlated with the **quality** and **quantity** of available data. To ensure superior

suggestions, the data utilized must be comprehensive, accurate, reliable, and regularly updated. Thus, it is crucial to select appropriate datasets for these systems.

For book recommendation systems, the necessary data included in the dataset largely depends on the system's architecture. User ratings are a crucial attribute of datasets, as collaborative filtering systems are reliant on them. Conversely, datasets that encompass rich metadata of books, such as the author, publisher, publication year, and book description, increase the accuracy of content-based systems. Demographic data found in some datasets can be utilized by demographic recommenders. Additionally, utilizing the complete text of the books and the textual reviews of users would enhance prediction accuracy when using semantic analysis.

Numerous free datasets are available for usage in book recommendation systems, facilitating the creation of robust recommendation systems. Here is an overview of popular open-source book databases[18][19]:

3.5.1 Goodreads

The Goodreads dataset includes a collection of book ratings and reviews, along with metadata such as title, author, and genre. The dataset contains over **200 million** ratings of more than **6 million** books[37].

3.5.2 Amazon Reviews

The Amazon Reviews dataset includes a collection of book reviews and ratings, along with metadata such as title, author, and publication date. The dataset contains over **8 million** book reviews and ratings[38].

3.5.3 Book-Crossing Dataset

The Book-Crossing dataset includes a collection of book ratings and reviews from the Book-Crossing community, along with metadata such as title, author, and publication date. The dataset contains over **1 million** ratings of more than **270,000 books**[39].

3.5.4 LibraryThing

The LibraryThing dataset includes a collection of book ratings and reviews, along with metadata such as title, author, and genre. The dataset contains over **1 million** ratings of more than **100,000 books**[40].

3.5.5 University of California San Diego Book Graph:

The UCSD Book Graph is a large-scale dataset of book ratings and reviews, along with metadata such as title, author, and genre. The dataset contains over **13.5 million** ratings of more than **1.1 million books**[41].

3.5.6 Project Gutenberg Dataset

Project Gutenberg is a collection of over **60,000 free eBooks** in the public domain, which can be downloaded in various formats. The dataset includes metadata such as the title, author, and publication date for each book[42].

3.5.7 Open Library Dataset

Open Library is a collaborative initiative that strives to create a webpage for every published book. Their dataset consists of valuable information like book title, author, publication date, and cover image. Moreover, the dataset contains user-generated reviews and ratings for each book, which can be used for building recommendation systems[43].

These datasets can be used to build recommendation systems based on collaborative filtering or other techniques, and can help improve the accuracy and effectiveness of book recommendations.

3.6 Conclusion

In conclusion, book recommender systems are valuable for readers and book industry stakeholders. With the continued advancement of technology and Artificial Intelligence, book recommendation systems are expected to become even **more accurate** and personalized in the future, further enhancing the reading experience for book lovers around the world.

This chapter has highlighted the importance of book recommender systems in the book industry, with studies showing that these systems can significantly improve the user's book discovery experience and increase book sales.

Furthermore, this chapter has discussed the different classifications of book recommender systems, their mechanisms, and how popular websites use them to generate recommendations online. We have also mentioned the most important databases that book recommender systems can use and looked at previous studies that discuss the different and varied architectures of these systems.

As we seek to leverage the insights and knowledge gained from these studies, we will delve into the topic of building our book recommender system in the next chapter.

Chapter 4

Our Recommendation System Design

4.1 Introduction

This chapter focuses on the system design for the proposed recommendation approach. The chapter provides a comprehensive overview of various techniques and methods employed in our recommendation system, highlighting their advantages and limitations. The aim is to present a **well-rounded understanding** of the system design process and the considerations involved in developing our recommendation system.

We begin by introducing the proposed recommendation approach, which serves as the foundation for the subsequent discussions. We then proceed to explain the different techniques that can be employed within this approach. These techniques include the **content-based technique**, **KNNm-based technique**, **SGD-based technique**, and **hybridization technique**. Each technique is examined in detail, outlining its fundamental principles, methodologies, and the unique benefits it offers in the realm of recommendation. First, let's start with the proposed recommendation approach

4.2 The proposed recommendation approach

The design of recommendation systems varies according to the available data and the conditions in which the system operates. That is why not all recommendation systems are created equal. Some are based on simple algorithms that don't take into account the complexity of

human preferences, while others rely on overly complex models that may not be practical to implement or use. Designing a recommendation system for books requires a deep understanding of machine learning techniques, data analysis, and user behavior. Therefore, after research, we came up with a suitable design for our recommendation system.

There is no perfect design for book recommendation systems. Books have much metadata that may determine whether a reader will like them or not. Furthermore, the behavior of readers who read these books may differ, where the meanings and the ways they evaluate them differ.

Each recommendation system has advantages when it uses certain data, and when it works under certain circumstances. However, it will have weaknesses in other circumstances. That's why we come to a hybrid recommendation system that combines different recommendation algorithms to provide better recommendations. We will explore the different types of recommendation algorithms that will be used, including **content-based**, **KNN with mean based**, and **SGD-based** algorithms. Likewise, we will also discuss the advantages and disadvantages of each algorithm and how they can be combined to create a more effective system.

4.3 Content-based approach

Content-based recommendation systems are a suitable choice for items that contain **rich metadata**. That is why in the field of books, content-based recommendation systems are a good choice. This type of recommendation system performs well when there is not much data available. Therefore, this type will help us a lot in solving problems related to the **lack of data**. It also gives high flexibility in terms of designing similarity equations because books have rich metadata. As this gives us the freedom to choose the attributes that will interfere in calculating the similarity between two books. Not only that, but this also gives us the freedom to determine the coefficients for the effect of the similarity ratio of an attribute on the overall similarity between the two books. This enormous flexibility gives us the ability to design similarity equations as our system requires.

There are many attributes that can be used to measure similarity. But not all of them really reflect the similarity felt by readers, such as page count, title, and book language. The use of inappropriate attributes may lead to inaccurate results in measuring similarity, and this will eventually lead to bad quality recommendations. In our system, we chose three attributes

to calculate the overall similarity between two books, which are book **genres**, **authors**, and **release year difference**.

4.3.1 Genre similarity

Analyzing the **similarity** between book genres is one of the key factors in building a successful content-based recommendation algorithm. Books within the same genre tend to share similar themes, writing styles, and target readers.

The similarity between the genres can be considered the main criterion for determining whether two books are similar. Therefore, By using **genre similarity**, the content recommendation algorithm can provide more accurate and relevant recommendations to readers. Theoretically, it is unlikely that two books are similar if the value of genre's similarity between them is equal to **zero**, no matter how similar they are in other attributes. We can consider genre similarity as the main criterion on which we will build the rest of the other similarities.

There are many common equations for measuring genre similarity, but the most prominent of these equations are **distance metrics** such as cosine similarity or Jaccard similarity. Both of these methods are widely used in the field of recommendation systems. However, we chose **cosine similarity**, because it gives better results in the field of genre similarity.

To use cosine similarity in calculating genre similarity, we deal with **genres** on the basis that they are **vectors** whose number of dimensions is the number of book genres in the dataset. Each of these dimensions represents a specific genre. In this vector, the coordinate value in each factor is zero or one. If the coordinate of one of these dimensions is zero, this means that the book does not belong to the genre that this dimension expresses. And the book belongs to this category if the corresponding coordinate is one. The following figure simplifies the method of converting genres list into vectors:

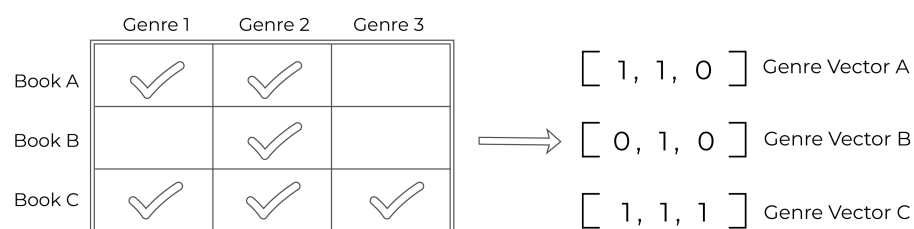


Figure 4.1: Converting genres list into vectors.

After we have expressed the genres as vectors, the similarity can now be calculated using the **cosine similarity**. Cosine similarity is defined as the dot product of two vectors divided by the product of their magnitudes. This can be expressed by the following mathematical equation:

$$similarity_{genres}(x, y) = \frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n x_i^2} \sqrt{\sum_{i=1}^n y_i^2}} \quad (4.1)$$

Where: x_i and y_i are the genre weights for $genre_i$ in genre $vector_x$ and genre $vector_y$, respectively. The sums are over all n genres.

We can translate this mathematical equation into the following algorithm:

Algorithm 1: Genre similarity calculations.

Input: book1, book2, genres, genreCount

Output: genreSimilarity

```

1 genres1 ← genres[book1];
2 genres2 ← genres[book2];
3 sumxx ← 0;
4 sumyy ← 0;
5 sumxy ← 0;
6 i ← 0;
7 while i < genreCount do
8     x ← genres1[i];
9     y ← genres2[i];
10    sumxx ← sumxx + x × x;
11    sumyy ← sumyy + y × y;
12    sumxy ← sumxy + x × y;
13    i ← i + 1;
14 genreSimilarity ← sumxy / sqrt(sumxx * sumyy);
```

The idea of this algorithm is simple. The algorithm retrieves the genre lists for *book1* and *book2* from the genres' dictionary. It then initializes sum_{xx} , sum_{yy} , and sum_{xy} as 0. These variables are used to compute the **cosine similarity** between the genre vectors of the two books.

The algorithm then iterates through each genre in the genre lists of *book1* and *book2*. For

each genre, it retrieves the genre scores x and y from the genre lists and updates the sum_{xx} , sum_{yy} , and sum_{xy} variables accordingly. After computing the sum_{xx} , sum_{yy} , and sum_{xy} variables for all genres, the algorithm computes the genre similarity as the cosine similarity between the two genre vectors. It divides the sum_{xy} variable by the square root of the multiple of sum_{xx} and sum_{yy} to obtain the cosine similarity.

The results of the cosine similarity equation are confined between 1 and -1. But because we use the values 1 and 0 only in vector factors, the results of this equation will be limited to zero in the case that the two books are not similar at all, and one in the case that the two books are identical. Later, the similarity value of the genres will play a pivotal role in measuring the total similarity between books.

4.3.2 Author similarity

Computing author similarity between books is another important factor in building a recommendation system for books. Usually, people tend to read more of a certain author's books if they like other books by the same author. That's because authors tend to use similar writing styles, tones, and themes in the books they write. It is significant to suggest more books written by an author if a reader liked some other books from that author.

By incorporating author similarity into the recommendation algorithm, we can provide users with a more holistic, accurate, and personalized recommendation.

In our system, we will measure author similarity by comparing the names of the authors. There are many ways to calculate author similarity between two books. Since we will be measuring the similarity between two lists of names, there are several similarity equations that can be used. In our case, we will use the Jaccard equation. We define the full names of the authors of the two books as two different lists, then we define a third list containing the common names of these two lists.

Jaccard similarity is defined as the size of the intersection of two sets divided by the size of the union of two sets. When used with books, it can be expressed by the following mathematical equation:

$$similarity_{authors}(x,y) = \frac{x \cap y}{x \cup y} + 1 \quad (4.2)$$

Where: x is the authors of the first book, y is the authors of the second book, and $x \cap y$ is the shared authors between the two books.

This equation can be transformed into the following algorithm:

Algorithm 2: Author similarity calculations.

Input: $book1, book2, authors$

Output: $authorsSimilarity$

```

1  $total\_similarities \leftarrow 0$ ;
2  $authors1 \leftarrow authors[book1]$ ;
3  $authors2 \leftarrow authors[book2]$ ;
4  $count \leftarrow 0$ ;
5 for  $author1$  in  $authors1$  do
6     for  $author2$  in  $authors2$  do
7         if  $author1 == author2$  then
8              $count \leftarrow count + 1$ ;
9  $union \leftarrow \text{length}(authors1) + \text{length}(authors2) - count$ ;
10  $authorsSimilarity \leftarrow count / union + 1$ ;
```

The algorithm initializes $total_similarities$ as 0. It then retrieves the author lists for $book1$ and $book2$ from the authors' dictionary. The variable $count$ is initialized as 0, which is used to count the number of authors in common between the two books.

The algorithm then iterates through each author in the author list of $book1$. For each author, it checks if the author is also in the author list of $book2$. If the author is in both author lists, it increments the $count$ variable by 1. After computing the number of authors in common, the algorithm computes the size of the union of the author lists. It then computes the similarity percentage as $count$ divided by the union.

The results of the Jaccard equations are always between zero in the case of complete dissimilarity. And one in the case of perfect similarity. Just because the authors are not the same does not necessarily mean that the books are completely different. That's why after we finish calculating the similarity of Jaccard, we add one to the total result.

In the end, the value of the similarity of the author of the two books will be between one in

the case of complete lack of similarity and two in the case of complete similarity.

4.3.3 Release year difference similarity

Another important factor in determining the value of similarity between two books is the time difference between their release date. By computing release year difference similarity, we can suggest books based on the time period. Books written over a similar period of time tend to be more similar in terms of writing style, themes, and events. The difference between different books chronologically is clear in some genres, such as science books and science fiction books. While it is not observed in some other categories such as philosophical books and religious books.

The design of the similarity equation between the two years of publication of two books responds carefully according to the requirements of the system. Our system will suggest a wide range of different books. We also do not have a specific group of target readers, so we target all readers with different preferences. All of this makes choosing a release year similarity equation complicated.

Out of all the possible equations, we chose a regression exponential function based on the difference between the two years multiplied by **-0.007** to calculate the similarity between the two years. Mathematically, it is :

$$similarity_{years}(x,y) = e^{-0.007 \cdot |x-y|} \quad (4.3)$$

Where: x is the release date of the first book, y is the release date of the second book.

It is easy to convert this equation into an algorithm. The corresponding algorithm is :

Algorithm 3: Year similarity calculations.

Input: $book1, book2, years$

Output: $yearsSimilarity$

```

1  $year1 \leftarrow years[book1];$ 
2  $year2 \leftarrow years[book2];$ 
3  $diff \leftarrow abs(years1 - years2);$ 
4  $yearsSimilarity \leftarrow exp(-0.007 \times diff);$ 

```

The algorithm gets the publication years for *book1* and *book2*. It then computes the absolute difference between the two years. The function $\exp$ is used to calculate the exponential function e raised to the power of -0.007 multiple diffs . This ensures that the similarity score decreases exponentially as the difference between the publication years of the two books increases. The following curve represents the decreasing similarity of the years according to the previous equation:

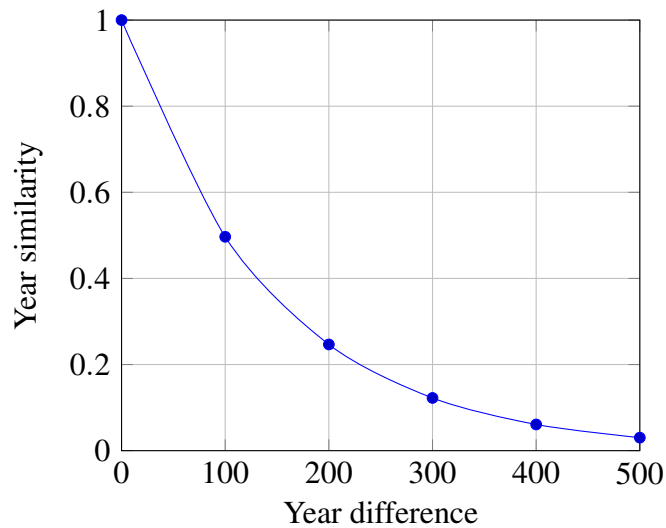


Figure 4.2: A curve showing the decreasing similarity with the increase in years difference.

The values resulting from this equation are one in the case that the two books were published in the same year, and the value is close to zero if the difference is too big. Books with a time difference of less than 10 years are nearly identical, while the similarity is halved with a difference of about 99 years.

4.3.4 Overall similarity

After calculating all the sub-similarities, we can now calculate the overall similarity. We designed the sub-similarity equations in a way that gives us a logical overall similarity when multiplying all these sub-similarities. The value of the genre similarity, which is confined between zero and one, represents the main criterion that will determine the total similarity. It's the only value that can go down to zero. If the value of the genre similarity is zero, this means that the similarity between the two books will be zero, whatever the result of the similarity of the years, and the result of the similarity of the authors.

Correspondingly, the least value of the similarity of the authors will be one at least. For that, the similarity of the book cannot negate the similarity or detract from its value. Where we consider that the existence of author's similarity between two books will be a supporting criterion for the overall similarity, and it will not be the opposite if it does not exist.

As for the similarities between authors, it cannot be said that the two books are similar just because they were published in the same year. Therefore, the highest value of years similarity is one. A large release year difference similarity will not support the overall similarity, but will do the opposite if the year difference is large. However, this value will never decrease to zero, so it will not make the total similarity zero.

The highest total similarity value between two books will be two if the two books belong to the same genres, were written by the same authors, and were published in the same year. While the similarity is zero if there is no common genre between the two books. The way to use sub-similarities to calculate the overall similarity between two books, x and y can be expressed by the following mathematical equation:

$$\begin{aligned} similarity(x,y) = & similarity_{genres}(x_{genres},y_{genres}) \times similarity_{authors}(x_{authors},y_{authors}) \\ & \times similarity_{years}(x_{year},y_{year}) \end{aligned} \quad (4.4)$$

After defining the overall and sub-similarity equations, we can construct the similarity matrix. To store similarity values between books, we create a two-dimensional array. The dimension of this matrix is the number of books. Each column or line in this matrix represents a specific book.

Initially, all values within this array are zero by default. Then we replace the similarity value between two books at the intersection of the first book line and the second book column, as well as at the intersection of the first book column and the second book line. We also leave the similarity value between the book and itself equal to zero so that this does not affect the recommendation process later. After this, the list of similarities will be ready to be used later in ratings prediction. This process can be done via the following algorithm.

Algorithm 4: Compute similarities between all pairs of items in content-based.

Input: *bookCount***Output:** *similarities*

```
1 similarities  $\leftarrow$  zeros((bookCount, bookCount));
2 for thisBook  $\in$  range(bookCount) do
3   for otherBook  $\in$  range(thisBook + 1, bookCount) do
4     genreSimilarity  $\leftarrow$  computeGenreSimilarity(thisBook, otherBook);
5     if genreSimilarity == 0 then
6       similarity  $\leftarrow$  0;
7     else
8       yearSimilarity  $\leftarrow$  computeYearSimilarity(thisBook, otherBook);
9       authorSimilarity  $\leftarrow$  computeAuthorSimilarity(thisBook, otherBook);
10      similarity  $\leftarrow$  genreSimilarity  $\times$  authorSimilarity  $\times$  yearSimilarity;
11      similarities[thisBook, otherBook]  $\leftarrow$  similarity;
12      similarities[otherBook, thisBook]  $\leftarrow$  similarity;
```

The working principle of this algorithm is as follows:

It initializes an array called *similarities*, it contains zeros with default, with size *books_count* $\times$ *books_count*. It loops through each book, and for each book, it loops through all other books after it. For each pair of books, it computes the genre similarity between them, if the genre similarity is not equal to 0, it computes year similarity, and author similarity as well. Then it combines the similarities into a single similarity score by multiplying them together.

Finally, it sets the similarity score in the *similarities* array at indices (*thisBook*, *otherBook*) and (*otherBook*, *thisBook*) in order not to waste processing power calculating this value twice. With this, we have been able to calculate the similarity between every two books in the database with the least possible processing power, and we have also stored them.

4.3.5 Rating prediction approach

After completing the calculation of the similarity between the books, we now take the second step, which is no less important in the process of recommending books. It is the process of calculating estimated ratings based on the similarity matrix.

The weighted similarity is one of the most popular methods for calculating estimated ratings based on similarity matrices. Weighted similarity in ratings prediction involves using a similarity metric to get the similarity between an item and the other rated items by a user, and then weighting the similarity scores based on the ratings of that items. This means that highly rated items are given more weight in the similarity calculation, while low-rated items are given less weight[2].

To get an estimated rating for a reader and a book using weighted similarity, we first obtain the similarity values of the books that were rated by the reader and that book. Using these similarity values, we calculate the overall similarity and the weighted similarity. then, we pick K top-rated books by that reader. Finally, we will obtain an estimated rating by dividing the weighted similarity totals by the total similarity totals. The process of calculating an estimated rating between a reader u and a book i is expressed by the following mathematical equation: 1

$$\hat{r}_{ui} = \frac{\sum_{j \in N_u^k(i)} \text{sim}(i, j) \cdot r_{uj}}{\sum_{j \in N_u^k(i)} \text{sim}(i, j)} \quad (4.5)$$

Where: $\hat{r}$ is the estimated rating, N is the list of book Neighbors, and K is the number of rated books we chose.

This equation can be translated into an algorithm that creates a list of neighbors, where each neighbor is an item that has been rated by the user u and has a similarity score with item i , computed using the pre-calculated similarity matrix. Then it Picks K neighbors with the largest similarity scores. For each neighbor, it computes the weighted average of their ratings, where the weights are the corresponding similarity scores. Finally, it divides the weighted similarity by the total similarity to give us an expected rating. This process is implemented by the following algorithm.

Algorithm 5: Rating prediction for content based.

Input: *reader, book, Ratings, similarities, meanRating***Output:** *predictedRating*

```
1 neighbors  $\leftarrow$  [];  
2 for rating  $\in$  Ratings[reader] do  
3   Similarity  $\leftarrow$  similarities[book, rating[book_id]];  
4   neighbors.append((Similarity, rating[rating_value]));  
5 simTotal  $\leftarrow$  0;  
6 weightedSum  $\leftarrow$  0;  
7 for (simScore, rating)  $\in$  k_neighbors do  
8   if simScore > 0 then  
9     simTotal  $\leftarrow$  simScore;  
10    weightedSum  $\leftarrow$  simScore  $\times$  rating;  
11 if simTotal == 0 then  
12   predictedRating  $\leftarrow$  meanRating;  
13 else  
14   predictedRating  $\leftarrow$  weightedSum / simTotal;
```

After designing the similarity equations and also designing a method for calculating the expected rating, we have completed the basic steps to create a content-based recommendation system. This system will give us several advantages when integrated into the hybrid recommendation system.

4.3.6 Advantages and limitations of our content-based approach

Our content-based recommendations system is less sensitive to data scarcity. This means that the system can predict acceptable quality ratings when there are not many ratings made by the reader. On the other hand, the content-based system can provide recommendations for new or unpopular books. This is because they analyze the content of the items and recommend similar items, even if they have not been rated or reviewed by other readers. This will be effective in a cold start situation. Not only that, but it can also suggest when the reader's desires are abnormal, so the gray sheep problem can also be reduced.

However, the system has some drawbacks. Perhaps the most notable of these drawbacks are some problems with weighted similarity. This approach can be problematic because outliers in the data, such as extremely high or low ratings, can disproportionately influence the similarity scores, leading to biased recommendations. Also, suggesting books that are very similar to books that the reader has read before only, can lead to preventing him from exploring new types of books. These problems will be mitigated when the content-based recommendation system is integrated with other recommendation systems.

4.4 KNNm-based approach

K-nearest neighbors algorithms are one of the most popular algorithms used in collaborative filtering recommendation systems. It is a memory-based algorithm. The principle of KNN is to find the items that tend to have similar users or the users that tend to like similar items[1]. That is, it does not care about the metadata of books or the data of readers, only the relationships between them. KNN algorithm can adapt to new data in real time, making it suitable for our online recommendation system.

In basic KNN, the algorithm identifies the k-nearest neighbors based on the similarity between their ratings and the user's ratings for a given item. This then leads us to the problem that readers do not always have the same method of evaluation. Some readers may give a five-star rating even if they did not like the book that much, in order to respect the writer's effort or for other reasons. While some other readers may be more strict in the evaluation, as they do not give five stars unless they liked the book very much, and they never hesitate to evaluate the book negatively if they do not like it. The difference between the behavior of readers may later affect measures of similarity between the two writers or readers, even if they are really similar.

In order to avoid this problem, we will use a modified KNN algorithm, which takes into account the average ratings made by the reader. This reader's average rating determines their rating behavior. The ratings that are further away from the average rating will have a greater effect on rating prediction. This will eventually lead to more accurate recommendations.

4.4.1 Similarity matrix calculation

There is no specific way to calculate the similarity matrix in this type of algorithm, any of the famous equations can be used to calculate the similarity. Again, we chose the cosine similarity to calculate the similarity between readers, because it is flexible in use, and gives more accurate results.

To determine the similarity between two readers, we calculate the sum of the products of their shared ratings. Then, we square each of their shared ratings, sum the squared ratings for each user, calculate the square root of each sum, and multiply those square roots. Finally, we divide the sum of products by the result of the previous calculation to get the similarity between the two readers. This mathematical equation provided by Surprise Library will make it easier to understand this method [44]:

$$[H]\text{similarity}(u, v) = \frac{\sum_{i \in I_{uv}} r_{ui} \cdot r_{vi}}{\sqrt{\sum_{i \in I_{uv}} r_{ui}^2} \cdot \sqrt{\sum_{i \in I_{uv}} r_{vi}^2}} \quad (4.6)$$

Where: u is a reader. r is the actual rating. And I_{uv} is the list of the ratings that u and v have done for the same books.

4.4.2 Rating prediction approach

After completing the design of the similarity equation, a similarity matrix can be created between users, and the values in it are calculated through the following algorithm: The algorithm initializes an empty similarity matrix of shape (n_users, n_users) to store the pairwise similarity scores between readers. It then loops over each reader, and for each reader, loops over all the readers that come after him in the matrix. This avoids redundancy since the similarity between reader i and reader j is the same as the similarity between reader j and reader i .

After designing the similarity equation, it remains for us to define a method for calculating the expected rating between a book and a reader. We'll use weighted similarity again, but this time we'll use the average value of reader ratings to make the assessment more accurate.

The rating prediction process works according to the following stages. First, we select the

readers who have previously rated this book and have the highest similarity with the target reader. Second, we calculate average readers ratings. Then, we calculate the estimated rating between a reader u and a book i according to the following mathematical formula [44]:

$$\hat{r}_{ui} = \mu_u + \frac{\sum_{v \in N_i^k(u)} \text{sim}(u, v) \cdot (r_{vi} - \mu_v)}{\sum_{v \in N_i^k(u)} \text{sim}(u, v)} \quad (4.7)$$

Where: $\hat{r}$ is the estimated rating. N is the list of reader neighbors. And μ is the mean value of ratings of the reader. The process can be translated into the following algorithm:

Algorithm 6: Rating prediction for KNNm.

Input: *reader, book, similarities, min_k, means*

Output: *predictedRating*

```

1  neighbors ← [];
2  for (otherReader, rating) ∈ readers[reader] do
3      sim ← similarities[reader, otherReader];
4      neighbors.append((otherReader, sim, rating));
5  sum_sim ← 0;
6  sum_ratings ← 0;
7  actual_k ← 0;
8  est ← means[reader];
9  for (otherReader, sim, r) ∈ k_neighbors do
10     if sim > 0 then
11         sum_sim ← sum_sim + sim;
12         sum_ratings ← sum_ratings + sim × (r − means[otherReader]);
13         actual_k ← actual_k + 1;
14  if actual_k < min_k then
15     sum_ratings ← 0;
16  if sum_sim == 0 then
17     predictedRating ← est;
18  else
19     predictedRating ← est + sum_ratings / sum_sim;

```

The algorithm creates a list of neighbors consisting of tuples containing other readers who have rated the same book as the input reader, the similarity between the input reader and those readers, and their ratings for that book. Then, it selects the k most similar neighbors from the neighbors list based on their similarity values and computes an initial estimate est for the rating as the mean rating given by the input user.

It then computes the weighted average of the ratings given by the k neighbors, where the weight for each rating is the similarity between the input reader and the neighbor, and subtracts the neighbor's mean rating from their rating. The weighted average is added to the initial estimate to get the final estimated rating. If there are fewer than min_k neighbors, the final estimated rating will be zero.

4.4.3 Advantages and limitations of our KNNm-based approach

This KNNm algorithm will give greater accuracy in identifying books that will be added later to the list of suggested books for the reader because the similarity between readers means greater accuracy. As the similarity of the desires of the readers means greatly the similarity of the books they like to read. The algorithm will give more variety in terms of books that the reader may want to read, because other readers may have positive interactions with them.

However, this algorithm clearly suffers from issues related to lack of data. This algorithm requires a sufficient amount of evaluations in order to find readers who are close to the reader's desires. Not having that many ratings means that the system will suffer from a cold start problem. The novelty and diversity of the suggestions cannot be controlled because they are subject to the behavior of the readers only. It is also difficult to find users similar to some users whose desires are extreme, which leads us to run into the problem of the gray sheep. These issues will be minimized when this algorithm is combined with other systems.

4.5 SGD-based approach

The development of the machine learning field has had a significant impact on the design and performance of recommendation systems. Traditional recommendation systems relied on rule-based approaches or simple statistical techniques to generate recommendations. However, with the development of machine learning techniques, recommendation systems have become

more sophisticated and accurate. Among many machine learning algorithms that can be used in the field of recommendation systems, such as Neural Collaborative Filtering (NCF), DeepFM, Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs), we decided to use the Stochastic Gradient Descent algorithm (SGD).

SGD is a popular optimization algorithm used in machine learning to find the best values of the model parameters that minimize the cost function. The principle of SGD is to update the model parameters iteratively based on the gradient of the cost function with respect to the parameters[45].

In the context of recommendation systems, SGD can be used to train matrix factorization models, which aim to decompose the reader-book rating matrix into two lower-dimensional matrices representing user and item features. This means that instead of calculating and recording similarity values between books or readers. Each reader and each book will have a vector that contains a certain number of factors.

4.5.1 Rating prediction and error calculation approach

When multiplying a book vector and a reader vector, the result will be an estimated rating from the reader for this book. As shown in the following document:

$$\begin{array}{ccc} \text{book}_i & [1, -5, 3.1, 0.2, 12.1, -1] & \\ \times & & \\ \text{reader}_u & [1.1, 3, 10, 8.2, -7, 34, -1] & \end{array} \Longrightarrow \text{Estimated Rating}$$

Figure 4.3: Rating prediction in SGD-based algorithm.

We can describe this by the following mathematical equation :

$$\hat{r}_{ui} = \sum_{k=1}^k p_{uk}q_{ik} \quad (4.8)$$

Where: $\hat{r}$ is the estimated rating, u is a reader, i is a book, p the matrix of readers, q the matrix of books and K is the number of factors in each vector.

The algorithm calculates the estimated rating as the dot product of the reader factors stored

in the p matrix, and the book factors that stored in the q matrix. The dot product operation calculates the sum of the element-wise product of two vectors. The result of this operation is the expected rating.

The prediction error can be calculated via the following equation :

$$err_{ui} = |\hat{r}_{ui} - r_{ui}| \quad (4.9)$$

Where: $\hat{r}$ is the estimated rating that is generated by the algorithm, u is a reader and i is a book, and r is the actual rating.

4.5.2 Matrixes training approach

After calculating the prediction error, the SGD algorithm can be used to find the optimal values of the factors in these matrices by minimizing the mean squared error between the predicted and actual ratings of the reader-book pairs according to these two equations:

$$p_u \leftarrow p_u + \gamma \cdot (err_{ui} \cdot q_i - \lambda \cdot p_u) \quad (4.10)$$

$$q_i \leftarrow q_i + \gamma \cdot (err_{ui} \cdot p_u - \lambda \cdot q_i) \quad (4.11)$$

Where: p is the matrix of readers, q is a matrix of books, γ is the learning rate and λ is the regression rate.

These equations can be used in an algorithm to train Matrixes of books and readers vectors according to the following steps. First, the algorithm initializes the readers vectors matrix, *readers_f*, and the books vectors matrix *books_f*, with random values drawn from a normal distribution. The size of these matrices is based on the number of readers and books in the training set, respectively, and the number of latent factors specified.

Second, it loops over each epoch in the range of the number of epochs specified. The number of epochs must be chosen carefully, choosing too little will result in the arrays not being trained enough, and choosing too many will result in wasting processing power without

achieving any additional precision.

Then, it loops over each rating and calculates the error between the predicted rating and the actual rating using the dot product of the user feature vector and the item feature vector. It uses this error value to update the reader vector and the book vector using stochastic gradient descent. The following Algorithm represents these steps:

Algorithm 7: Matrix factorization training with SGD.

Input: *learning_rate* , *epochs* , *reg* , *n_factors*, *n_readers* , *n_books*

Output: *Ratings*[]

```

1 readers_f  $\leftarrow$  random(n_readers,n_factors);
2 books_f  $\leftarrow$  random(n_books,n_factors);
3 for epoch  $\in$  range(epochs) do
4   for (reader,book,rating)  $\in$  Ratings[] do
5     prediction  $\leftarrow$  dot(readers_f[reader],books_f[book]);
6     error  $\leftarrow$  rating - prediction;
7     readers_f[reader]  $\leftarrow$  readers_f[reader] + learning_rate  $\times$  (error  $\times$ 
      books_f[book] - reg  $\times$  readers_f[reader]);
8     books_f[book]  $\leftarrow$  books_f[book] + learning_rate(error  $\times$ 
      readers_f[reader] - reg  $\times$  books_f[book]);

```

4.5.3 Advantages and limitations of our SGD-based approach

This algorithm will give a good addition to our hybrid system. SGD can handle sparse data, which is common in book recommendation systems, by using regularization techniques to prevent overfitting and improve the performance of the model. SGD also allows for online learning, which means that the matrixes can be updated in real-time as new data becomes available. The recommendations when using matrix factorization combine accuracy and diversity more than similarity-based systems.

On the other hand, the SGD algorithm has some drawbacks. SGD can be sensitive to the initial values of the model parameters, which may require careful initialization. Moreover, the SGD algorithm is sensitive to scarcity of data. Books and readers vectors cannot be trained if sufficient ratings are not available.

4.6 Hybridization approach

Having finished designing these three systems, we can now design a hybrid System that uses all the previous systems in order to give a Top-N- recommendation list to each reader.

4.6.1 Hybrid prediction approach

In order to generate a hybrid suggestion list. Lists of expected ratings for all books by a reader must first be generated for each of the three systems. This list can be generated from each system with the following algorithm:

For each book, we call a function that returns the value of the expected rating between the reader and the book. We store these values in a list called "**predictions**". After forming the three prediction lists, we form a hybrid prediction list according to the algorithm:

Algorithm 8: Predict all ratings for a reader in each algorithm.

Input: *reader, booksCount*

Output: *predictedRatings*

```
1 predictions  $\leftarrow$  [];  
2 for book  $\in$  range(booksCount) : do  
3   rating  $\leftarrow$  predict(reader, book);  
4   predictions.append(book, rating);
```

For each book, we collect the three predictions that produced by the three prediction systems. We get the estimated hybrid rating by dividing this sum by three.

4.6.2 Top N Recommendation list creation

A list of Top N recommendations can be obtained from the list of hybrid predictions, after removing books that already have an actual rating from the reader. The N remaining books with the highest expected ratings would be on the Top N recommendations list. This algorithm explains the steps to get the hybrid predictions :

Algorithm 9: Combine predictions from multiple recommenders.

Input: *reader, CoRecommender, CBRecommender, SGDRecommender, booksCount***Output:** *hybridPredictions*

```
1 CoPredictions  $\leftarrow$  CoRecommender.EstimatedRatingsForUser(reader);  
2 CBPredictions  $\leftarrow$  CBRecommender.EstimatedRatingsForUser(reader);  
3 SGDPredictions  $\leftarrow$  SGDRecommender.EstimatedRatingsForUser(reader);  
4 hybridPredictions  $\leftarrow$  [];  
5 for book  $\in$  range(booksCount) do  
6     estimatedRating  $\leftarrow$   
        (CoPredictions[book] + CBPredictions[book] + SGDPredictions[book])/3;  
7     hybridPredictions.append([book, estimatedRating]);
```

By mapping out how all the recommendation subsystems work, how to train them, and generating a list of suggestions for a reader, we have finished designing our hybrid recommendation system.

4.7 Conclusion

In conclusion, this chapter presented a comprehensive overview of the system design for the proposed recommendation approach. It explores various techniques and methods for recommendation, highlighting their advantages and limitations. The proposed approaches offer different perspectives on recommendation, catering to diverse reader preferences and addressing the challenges of accuracy and quality. By combining the approaches and subjecting the system to validation in the next chapter, we aim to create a system that can provide effective and personalized recommendations to readers and enhance their overall experience.

Chapter 5

Experimental Evaluation

5.1 Introduction

This chapter focuses on the **experimental evaluation** of our recommendation system. This chapter aims to assess the performance and effectiveness of the recommendation algorithms through a systematic validation process. By conducting experiments and analyzing the results, we gain valuable insights into the capabilities and limitations of the implemented algorithms.

The experimental evaluation is carried out in a controlled environment using appropriate tools and methodologies. We leverage **Google Colab** as the experimental platform, providing us with the necessary computational resources and collaborative features. The experiments are implemented using **Python**, a versatile programming language known for its efficiency in data processing and algorithm implementation.

To facilitate the evaluation process, we employ the **Surprise Library**, a powerful toolkit for building and evaluating recommender systems. This library offers a wide range of algorithms and evaluation metrics, allowing us to assess the performance of the recommendation algorithms.

In this chapter, we also provide a comprehensive description of the dataset used in the experiments. This dataset contains user-item interaction data, which serve as the basis for generating recommendations. Understanding the characteristics and properties of this dataset is crucial for interpreting the experimental results accurately.

Furthermore, we introduce the validation methodology employed to assess the recommendation algorithms. This includes data splitting strategies, evaluation protocols, and performance metrics used to measure the accuracy and quality of the recommendations.

All the codes that will be used during this chapter can be obtained from a GitHub repository we created. The repository contains a file that includes all the necessary steps to configure the project. The repository can be accessed through this [link](#), or by scanning the following QR code:



Figure 5.1: Our recommender system repository link.

The chapter concludes with a thorough analysis and discussion of the experimental results. We examine the performance of each recommendation algorithm based on the validation metrics and provide insights into their strengths and weaknesses. The findings from the experimental evaluation contribute to a deeper understanding of the recommendation system's capabilities and pave the way for potential improvements and future research.

5.2 Experimental environment

The experimental environment plays a crucial role in ensuring the **accuracy**, **reproducibility**, and **efficiency** of the experiments. We provide an overview of the key components of the experimental environment, including **Google Colab**, **Python** programming language, and the **Surprise library**.

5.2.1 Google Colab

Google Colab serves as the experimental platform, offering a cloud-based computing environment with powerful resources and collaboration features. It provides the necessary computational capabilities to handle large-scale datasets and computationally intensive tasks involved in the evaluation. Additionally, the collaborative nature of Google Colab allows for easy sharing and collaboration among researchers[46].

5.2.2 Python

Python is the programming language used for implementing the experiments. Python is widely known for its simplicity, readability, and extensive library ecosystem, making it a popular choice for data analysis and machine learning tasks. Its rich set of libraries and frameworks enables efficient data processing, algorithm implementation, and result visualization[47].

5.2.3 Surprise Library

To facilitate the evaluation process, we leverage the **Surprise Library**, a comprehensive toolkit specifically designed for building and evaluating recommender systems. The Surprise Library provides a collection of state-of-the-art recommendation algorithms, evaluation metrics, and data manipulation utilities. It simplifies the implementation of recommendation algorithms and provides a standardized framework for evaluating their performance[48].

By utilizing **Google Colab**, **Python**, and the **Surprise library**, we establish a robust experimental environment that ensures reliable and efficient evaluation of the recommendation system. The combination of these tools and technologies enables us to conduct experiments in a controlled and reproducible manner, facilitating the analysis and interpretation of the experimental results.

In the next section, we move into the validation methodology employed to assess the performance of the recommendation algorithms.

5.3 Validation methodology

Previously, we highlighted the importance of accurate recommendations in a recommender system, and now we will dive into the evaluation metrics we utilized. These metrics provide insights into the system's performance and its ability to deliver satisfactory predictions and user experiences.

To evaluate recommender systems, various metrics are commonly employed. Before discussing these metrics, it is necessary to establish a **testing methodology**, which can be either **online** or **offline**. In this context, we will focus on one offline testing methodology: Train Test Splits. We will calculate the accuracy of the generated suggestions using the **MAE** and **RMSE** scales. We will also measure **coverage**, **diversity**, and **novelty**. All of those metrics have been discussed in the second chapter.

To use these methods and evaluate the performance of the recommendation system, we took the following steps:

- Using the Python programming language, we have programmed the recommendation subsystems and the classes responsible for reading and organizing data from the dataset, with the help of the **Surprise library**, which contains many classes that help in this matter.
- We have programmed a hybrid recommendation system that uses three sub-recommendation systems.
- We have programmed classes that evaluate recommendation systems.
- We have uploaded the codes and dataset files to Google Colab to run it.
- We compared the performance of the sub-recommendation systems with the hybrid recommendation system.
- We compared our hybrid recommendation system with other recommendation systems, to compare its performance compared to other systems. We have compared it with the **basic KNN** Recommendation System, the **NMF** Recommendation System which uses the matrix factorization principle, and a **random** Recommendation System which provides random suggestions. These three recommendation systems are found by default on

the Surprise library.

5.4 Recommendation algorithms implementation

Within our recommendation system, we have implemented different algorithms that serve as the backbone of our personalized recommendation engine. These algorithms play a crucial role in providing tailored recommendations to our users.

In this section, we will take an in-depth look at five key classes that form the foundation of our recommendation system. These classes include `theDataProvider`, `CBAAlgorithm`, `CoAlgorithm`, `SGDAlgorithm`, and `HybridAlgorithm`. Each class contributes unique capabilities to the recommendation process.

5.4.1 The DataProvider Class

The `DataProvider` class handles book and rating data. It includes two dictionaries, *bookID_to_name* and *name_to_BookID*, for mapping book IDs to names and vice versa. The class also stores file paths for the ratings dataset and books dataset.

Here are the different methods that it includes:

- The `loadData` method loads the dataset. It uses the `ratingsPath` to load the ratings dataset into the `ratingsDataset` variable using the `Dataset.load_from_file` method. The book data from the `booksPath` CSV file is loaded into the *bookID_to_name* and *name_to_BookID* dictionaries. The method returns the `ratingsDataset` object.
- The `getUserRatings` method retrieves a user's ratings as a list of tuples. It reads the ratings data from the CSV file specified in `ratingsPath` and extracts the ratings for the given user ID.
- The `getPopularityRanks` method calculates the popularity ranks of books based on their rating count. It reads the ratings data from the CSV file specified in `ratingsPath`, counts the ratings for each book, and assigns ranks based on the count. The ranks are returned as a dictionary with book IDs as keys and ranks as values.
- The `getGenres` method retrieves the genre information for each book in the dataset.

It reads the books file and parses the genre data for each book. The method creates a dictionary with book IDs as keys and binary bitfields as values, representing the genres.

- The `getYears` method retrieves the publication year for each book in the dataset. It reads the books file and extracts the year information for each book. The method creates a dictionary with book IDs as keys and publication years as values.
- The `getStandardGenres` method retrieves the standard genres associated with each book in the dataset. It reads the books file and parses the genre data for each book. The method creates a dictionary with book IDs as keys and lists of standard genres as values.
- The `getAuthors` method retrieves the author information for each book in the dataset. It reads the books file and parses the author data for each book. The method creates a dictionary with book IDs as keys and lists of authors as values.
- The `getBookName` method returns the book name for a given book ID. It looks up the book ID in the *bookID_to_name* dictionary and returns the corresponding name if found; otherwise, it returns an empty string.
- The `getBookID` method returns the book ID for a given book name. It looks up the book name in the *name_to_BookID* dictionary and returns the corresponding ID if found; otherwise, it returns 0.

5.4.2 The CBAAlgorithm Class

The `CBAAlgorithm` class is a content-based recommendation algorithm that computes item similarities based on content attributes like genres, years, and authors. It is a subclass of `AlgoBase`, the parent class that provides a basic recommender template.

Here are the different methods that it includes:

- The `CBAAlgorithm`'s constructor initializes the algorithm, including the "k" parameter that determines the number of neighbors considered for similarity calculations.
- The `fit` method trains the algorithm on a provided training set. It loads metadata for each book, such as genres, years, and authors. If instructed, it imports precomputed

similarities; otherwise, it initializes a similarity matrix. It computes the content-based similarity matrix by comparing each item with every other item, calculating genre, year, and author similarities. The similarities are stored in the matrix.

- The `computeGenreSimilarity` method calculates genre similarity between two books using the cosine similarity formula and genre metadata.
- The `computeYearSimilarity` method calculates year similarity between two books using the difference in years and a simple decay function.
- The `computeAuthorSimilarity` method calculates author similarity between two books based on the number of matching authors.
- The `exportSimilarities` method saves the computed similarities matrix to a file, while `importSimilarities` loads precomputed similarities from a file.
- The `estimate` method predicts the rating between a user and an item. It checks if the user and item exist, creates a list of rated items by the user with their similarity scores, selects the *k* nearest neighbors, and computes the weighted average similarity score considering user ratings.
- The `EstamedRatingsForUser` method generates estimated ratings for a given user by iterating over all items, calling the `predict` function, and collecting the predicted ratings.

5.4.3 The CoAlgorithm Class

The `CoAlgorithm` class is defined as a subclass of `KNNWithMeans`, a collaborative filtering algorithm in `Surprise`. It adds additional functionalities to the KNN algorithm.

Here are the different methods that it includes:

- The `init` method serves as the constructor of the `CoAlgorithm` class. It initializes the algorithm by calling the constructor of the parent class, `KNNWithMeans`, and passing the desired similarity options (cosine similarity and user-based filtering). The *k* parameter determines the number of nearest neighbors to consider.
- The `fit` method overrides the `fit` method of the parent class. It takes a trainset as input and fits the algorithm on the provided training set by calling the `fit` method of

KNNWithMeans.

- The `EstamedRatingsForUser` method is a custom method that generates estimated ratings for a given user. It takes a user ID (`u`) as input. Inside the method, a list called `prid` is created to store the predicted ratings for all items in the training set.

The method then iterates over all items in the training set using the `range` function and `self.trainset.n_items` as the upper bound. For each item, it calls the `predict` function of the algorithm to generate a predicted rating for the user-item pair (`u`, `item`). The predicted rating is appended to the `prid` list. Once all predicted ratings are generated, the `prid` list is returned.

5.4.4 The SGDAAlgorithm Class

The `SGDAAlgorithm` class is defined with the `init` method serving as the constructor. It accepts hyperparameters such as `learning_rate`, `n_epochs`, `n_factors`, and `reg`, and assigns them to the corresponding instance variables.

Here are the different methods that it includes:

- The `fit` method trains the algorithm on a given training set by overriding the `fit` method of the parent class, `AlgoBase`. It first calls the parent's `fit` method to initialize the algorithm with the training set.

Next, it initializes user and item factors matrices with random values. The algorithm then enters a loop for the specified number of epochs and iterates over each user-item rating pair in the training set. For each rating, it calculates the prediction error and updates the user and item factors using the stochastic gradient descent (SGD) update rule.

- The `estimate` method calculates the estimated rating between a user and an item. It takes a user ID (`u`) and an item ID (`i`) as input. It first checks if the user and item IDs are known in the training set. If either is unknown, it raises a `PredictionImpossible` exception. The estimated rating is then calculated as the dot product between the user factors and item factors for the given user and item.
- The `EstamedRatingsForUser` method generates the estimated ratings for a given user. It takes a user ID as input and initializes an empty list to store the predicted ratings. It

iterates over all items in the training set and calls the predict function of the algorithm to generate a predicted rating for the user-item pair. The predicted rating is appended to the list. Once all predicted ratings are generated, the list is returned as the output.

5.4.5 HybridAlgorithm Class

The HybridAlgorithm class is initialized with a list of recommender algorithms, `recommendersList`. It has an instance variable, `recommenders`, to store the list of recommender algorithms.

Here are the different methods that it includes:

- The `fit` method is responsible for training the hybrid algorithm using a given training set. It iterates over each algorithm in the `recommenders` list using a for loop. For each algorithm, it calls the `fit` method of the algorithm and passes the training set as a parameter. This ensures that each individual recommender in the hybrid algorithm is trained on the same data, maintaining consistency.
- The `predict` method generates predictions for a user-item pair. It takes parameters such as `uid` (user ID), `iid` (item ID), `rui` (true rating - optional), `clip` (clipping the rating - optional), and `verbose` (verbosity - optional). An empty list called `predictions` is initialized to store the predictions generated by each recommender algorithm.

The `predict` method then iterates over each algorithm in the `recommenders` list using a for loop. For each algorithm, it calls the `predict` method of the algorithm and passes the user ID, item ID, and other optional parameters. The prediction is appended to the `predictions` list. To calculate the mean prediction, the method sums up the estimated ratings (`est`) from all the predictions in the `predictions` list and divides it by the number of predictions.

Finally, the `predict` method constructs a prediction list by combining the user ID, item ID, true rating, mean prediction, and other details from the first prediction in the `predictions` list. This prediction list is returned as the output.

- The `test` method evaluates the performance of the hybrid algorithm on a test set. It takes the `testset`, which contains user-item pairs for evaluation, as input. An empty list

called `predictions` is initialized to store the predictions for each user-item pair.

The `test` method iterates over each user-item pair in the testset using a `for` loop. For each pair, it calls the `predict` method, passing the `user ID`, `item ID`, and other optional parameters. The prediction is appended to the `predictions` list. After iterating through all user-item pairs, the `predictions` list is returned as the output of the `test` method.

5.5 The used dataset

In order to develop and evaluate recommender systems, it is crucial to have access to relevant datasets. In our project, we utilize the "goodbooks-10k" dataset.

The "goodbooks-10k" dataset, which is publicly available on GitHub. This dataset was created by Zygmunt Zajac specifically for recommender system research and evaluation purposes.

The "goodbooks-10k" dataset comprises comprehensive information about books, user ratings, and various associated metadata. It was created by collecting ratings from the Goodreads website, a popular online platform for book enthusiasts. With a diverse range of books spanning various genres and categories, this dataset offers valuable insights for recommender system development.

Key features of the "goodbooks-10k" dataset include:

- **Book Information:** Each book in the dataset is represented by a unique book ID, along with details such as title, author, publication year and 39 genres. This information provides a comprehensive understanding of the characteristics and attributes of the books included in the dataset.
- **User Ratings:** Users' ratings for individual books are a fundamental component of the dataset. Ratings are typically provided on a numerical scale, allowing users to express their preferences and satisfaction levels for different books.
- **User IDs:** Unique user IDs are assigned to individual users who have provided ratings. These user IDs enable the tracking of user behavior and preferences across multiple books, facilitating personalized recommendations.

- **Additional Metadata:** The "goodbooks-10k" dataset also offers supplementary meta-data for the books, such as tags and even book cover images. This metadata enhances the recommendation algorithms and provides additional features for analysis.

This dataset contains approximately 50K users with their ratings. Testing recommendation systems on this number of users requires high processing capacity and a long time, so we will suffice with 5K users.

To access the "goodbooks-10k" dataset and its contents, you can visit this GitHub repository [49]. The repository provides instructions on how to download and utilize the dataset effectively in your recommender system projects.

By using the goodbooks-10k", we aim to develop a more comprehensive and accurate recommender system.

5.6 Evaluation Implementation

Designing the code responsible for measuring the performance of the recommendation system is as important as designing the recommendation system itself. Through meticulous evaluation code, we can assess the performance, accuracy, and overall quality of our recommendation system, enabling us to refine and optimize its design. In this section, we will talk about the method of applying the measurements and the codes used for that.

In order for the evaluation code to be understandable and effective, we have divided it into four main classes, they can be found in Evaluation folder in our GitHub repository.

5.6.1 EvaluationData class

The objective of this class is to provide the necessary data for the evaluation process. So, it provides the different types of training sets for different tests, as well as providing a similarity matrix to calculate diversity. The code representing this class is in the file `EvaluationData.py`.

The `EvaluationData` class has several functions that serve different purposes in the code. Here's a short explanation of each function:

- `init(self, data, popularityRankings)`: The constructor function initializes the

EvaluationData object and sets its attributes. It takes data (a dataset object from the Surprise library) and popularityRankings as parameters. The function builds a full training set for evaluating overall properties, creates an anti-test set, performs a 75/25 train/test split for measuring accuracy, initializes a content-based recommender algorithm, and fits it to the training set to compute the similarity matrix between items.

- `GetFullTrainSet(self)`: This function returns the full training set that was built during the initialization of the object. It provides access to the complete set of data for evaluating overall properties.
- `GetFullAntiTestSet(self)`: This function returns the anti-test set that was created during the initialization of the object. The anti-test set contains all the user-item pairs that are not present in the training set. It can be used to evaluate how well the recommendation system can predict unseen user-item interactions.
- `GetTrainSet(self)`: This function returns the training set that was generated during the initialization of the object. It provides access to the subset of data used for training the recommendation system.
- `GetTestSet(self)`: This function returns the test set that was created during the initialization of the object. The test set is used to measure the accuracy of the recommendation system. It contains a portion of the data that was not used for training.
- `GetSimilarities(self)`: This function returns the content-based recommender algorithm `CBAAlgorithm` object that was created and fitted to the training set during the initialization of the object. The algorithm computes the similarity matrix between items, which can be used to measure diversity in the recommendations.
- `GetPopularityRankings(self)`: This function returns the popularity rankings that were passed to the object during initialization. The popularity rankings represent the popularity of items in the dataset and can be used for various evaluation purposes.

5.6.2 RecommenderMetrics class

This class contains the functions that embody the performance measures that we have chosen to evaluate the performance of the recommendation systems, it can be found in `Recommender-`

Let's explain each function:

- `MAE(predictions)`: This function calculates the **Mean Absolute Error (MAE)** metric for the given predictions. It uses the `accuracy.mae()` function from the Surprise library to compute the **MAE**.
- `RMSE(predictions)`: This function calculates the **Root Mean Squared Error (RMSE)** metric for the given predictions. It uses the `accuracy.rmse()` function from the Surprise library to compute the **RMSE**.
- `additionalMetrics(self, rankings, TestData, simsAlgo, ratingThreshold=4.0)`: This function calculates additional evaluation metrics related to *coverage*, *diversity*, and *novelty*. It takes several parameters: `rankings` (popularity rankings of items), `TestData` (test data containing user-item interactions), `simsAlgo` (content-based recommender algorithm), and an optional `ratingThreshold` (the threshold for considering a prediction as a coverage hit).

Inside the function, the following variables are initialized:

- `coverage_hits` : Keeps track of the number of hits for coverage calculation.
- `usersCount` : Stores the number of users in the test data.
- `n_diversity` : Counts the number of diversity pairs.
- `total_diversity` : Stores the sum of similarities between diverse pairs.
- `n_novelty` : Counts the number of novelty items.
- `total_novelty` : Stores the sum of ranks for novelty calculation.

The function iterates over each user in the `TestData` and performs the following steps:

- It generates predictions using the recommendation algorithm `self.algorithm` for each user.
- It extracts the top N predictions and appends them to the `topN` list.

- It sorts the `topN` list based on estimated ratings in descending order.
- It checks if the highest-rated item in `topN` exceeds the `ratingThreshold` and updates `coverage_hits` accordingly.
- It calculates diversity by iterating through combinations of items in `topN` and computing the similarity between them using the `simsAlgo` object.
- It calculates novelty by summing the ranks of items in `topN` based on the rankings.
- It updates `usersCount`.

Finally, the function calculates the *coverage*, *diversity*, and *novelty* metrics and returns them as a tuple (`coverage`, `diversity`, `novelty`).

5.6.3 EvaluatedAlgorithm class

This class is responsible for training and testing recommendation algorithms. This class calls an object from `EvaluationData` class and an object from `RecommenderMetrics` class. This class trains the algorithm with the appropriate training rule from the `EvaluationData` object and then tests it with one of the metrics from the `RecommenderMetrics`. The code is in `EvaluatedAlgorithm.py`

Here's a short explanation of each function:

- `init(self, algorithm, name)`: The constructor function initializes the `EvaluatedAlgorithm` object. It takes two parameters: `algorithm`, representing the recommendation algorithm, and `name`, representing the name or identifier of the algorithm. The function sets the `algorithm` and `name` attributes of the object.
- `Evaluate(self, evaluationData, verbose=True)`: This function evaluates the performance of the algorithm using various metrics. It takes an `evaluationData` object, which contains the necessary data for evaluation, and optional parameters `verbose`, specifying whether to print evaluation progress. Inside the function, the following steps are performed:

The algorithm is fitted to the training set obtained from `evaluationData`. Predictions are generated using the algorithm on the test set obtained from `evaluationData`.

The RMSE and MAE metrics are calculated using the `RecommenderMetrics.RMSE()` and `RecommenderMetrics.MAE()` functions, respectively. The algorithm is fitted to the full training set obtained from `evaluationData`. The anti-test set is obtained from `evaluationData`. The `additionalMetrics()` function from the `RecommenderMetrics` class is called to calculate *coverage*, *diversity*, and *novelty* metrics using the anti-test set, popularity rankings, and similarity matrix. The metrics are stored in a dictionary called `metrics`. If `verbose` is `True`, progress messages are printed during the evaluation. Finally, the function returns the metrics dictionary containing the evaluation results.

- `GetName(self)`: This function returns the name of the algorithm.
- `GetAlgorithm(self)`: This function returns the algorithm object.

5.6.4 Evaluator class

This is the class responsible for measuring and displaying the results. This class organizes the selected recommendation systems according to a list, then calls an object from `EvaluatedAlgorithm` class, this object every time tests one of the algorithms and stores the test results. Upon completion of the test, the final results of the test are displayed. The correspondent code is in `Evaluator.py`.

Let's explain each function:

- `init(self, dataset, rankings)`: The constructor function initializes the `Evaluator` object. It takes a `dataset` object and `rankings` as parameters. Inside the constructor, an `EvaluationData` object is created using the `dataset` and `rankings`, which is stored as `self.dataset`.
- `AddAlgorithm(self, algorithm, name)`: This function adds an algorithm to the list of algorithms to be evaluated. It takes an `algorithm` object and a `name` parameter. Inside the function, an `EvaluatedAlgorithm` object is created using the `algorithm` and `name`, which is then appended to the `algorithms` list of the `Evaluator` object.
- `Evaluate(self)`: This function performs the evaluation of each algorithm added to the `Evaluator` object. It iterates over the algorithms in the `algorithms` list and calls the `Evaluate()` function of each `EvaluatedAlgorithm` object. The evaluation results

are stored in a dictionary called `results`, with the algorithm name as the key and the evaluation metrics as the value.

After evaluating all the algorithms, the function prints the results in a tabular format, showing the *RMSE*, *MAE*, *coverage*, *diversity*, and *novelty* metrics for each algorithm.

5.6.5 Evaluation

After completing the programming of the algorithms, and completing the testing algorithms, the results can be obtained by running `TestAlgorithm.py` in the main folder.

Let's go through the code step by step:

- `LoadBooksAndRatingsData()`: This function loads the book ratings data using a `DataProvider` object (BD) and computes the popularity ranks for books. The function returns a tuple containing the `DataProvider` object, the loaded data, and the rankings.
- `(BD, data, rankings) = LoadBooksAndRatingsData()`: This line calls the `LoadBooksAndRatingsData()` function and assigns the returned values to variables `BD`, `data`, and `rankings`.
- `evaluator = Evaluator(data, rankings)`: An `Evaluator` object is created using the data and rankings obtained from the previous step.
- **Algorithm definitions and addition to the evaluator:** Several recommendation algorithms are instantiated, including our `CoAlgorithm`, our `CBAAlgorithm`, our `SGDAlgorithm`, and our `HybridAlgorithm`. `NMF`, `KNNBasic`, and `Random` which are found in `Surprise` library also tested. We initialize `NMF` recommender with the same parameters as our `SGD`-based recommender. Each algorithm object is added to the evaluator using the `AddAlgorithm()` function, with a name identifying the algorithm.
- `evaluator.Evaluate()`: This line invokes the evaluation process using the `Evaluate()` function of the `Evaluator` object. The evaluation is performed for each algorithm added to the evaluator. The evaluation results are printed, showing the *RMSE*, *MAE*, *coverage*, *diversity*, and *novelty* metrics for each algorithm.

5.7 Results discussion

After evaluating multiple recommendation algorithms on the book ratings dataset, we now turn our attention to discussing the results. The evaluation process provided us with valuable insights into the performance of different algorithms in terms of various metrics such as *RMSE*, *MAE*, *coverage*, *diversity*, and *novelty*.

In this discussion section, we will delve into the evaluation results and analyze the strengths and weaknesses of each of our algorithms, and compare them with other algorithms. By examining the metrics and comparing the performance of the algorithms, we aim to gain a deeper understanding of their effectiveness in providing accurate and diverse recommendations.

5.7.1 The sub-recommender systems vs the hybrid recommender system

In this evaluation of recommendation algorithms, we compared the performance of CoRecommender, CBRecommender, SGDR recommender, and HybridRecommender. We got the following results:

Algorithm	RMSE	MAE	Coverage	Diversity	Novelty
CoRecommender	0.8712	0.6879	0.9992	0.7580	7783.3509
CBRecommender	0.8803	0.6952	0.8068	0.3361	4440.6937
SGDR recommender	0.8693	0.6815	0.9916	0.6671	2401.5264
HybridRecommender	0.8444	0.6684	0.9538	0.6930	5261.7557

Table 5.1: Performance Metrics results of our sub-algorithms and the hybrid algorithm.

Let's analyze the results and discuss how the HybridRecommender outperformed the other algorithms in terms of accuracy, coverage, diversity, and novelty.

- **CoRecommender:**

- RMSE: 0.8712
- MAE: 0.6879
- Coverage: 0.9992
- Diversity: 0.7580
- Novelty: 7783.3509

The CoRecommender, the KNNm-based algorithm, achieved high accuracy with a low *RMSE* and *MAE*, indicating its ability to provide accurate recommendations to users. It also exhibited excellent coverage, suggesting that the algorithm recommended books to a large majority of users. The diversity score was a little high, but acceptable considering that we have 39 genres of books. However, the novelty scores were very high, indicating that the algorithm may not have recommended sufficiently famous suggestions.

- **CBRecommender:**

- RMSE: 0.8803
- MAE: 0.6952
- Coverage: 0.8068
- Diversity: 0.3361
- Novelty: 4440.6937

CBRecommender, the content-based recommendation algorithm, achieved moderate accuracy but relatively lower *coverage*, and *diversity* scores. While its recommendations may have been accurate, they were limited in terms of *coverage* and *diversity*. While the *novelty* was significantly moderate and appropriate. The algorithm may have favored certain types of books, resulting in a narrower range of recommendations.

- **SGDRecommender:**

- RMSE: 0.8693
- MAE: 0.6815
- Coverage: 0.9916
- Diversity: 0.6671
- Novelty: 2401.5264

SGDRecommender demonstrated good accuracy with low RMSE and MAE values. It achieved high coverage, indicating that a significant number of users received recommendations. The diversity score was moderate. However, the novelty score was low,

indicating that the algorithm may not have recommended a wide variety of books or sufficiently novel suggestions.

- **HybridRecommender:**

- RMSE: 0.8444
- MAE: 0.6684
- Coverage: 0.9538
- Diversity: 0.6930
- Novelty: 5261.7557

The HybridRecommender algorithm, which combines CoRecommender, CBRecommender, and SGDRecommender, outperformed the other algorithms in terms of accuracy, coverage, diversity, and novelty. With the lowest RMSE and MAE, it provided highly accurate recommendations. The algorithm also demonstrated good coverage, diversity, and novelty, suggesting that it recommended a wide range of books and introduced novel suggestions to users, while retaining the reliability that comes from suggesting popular books.

Overall, the HybridRecommender algorithm offered the best balance between accuracy and other evaluation metrics. By leveraging the strengths of multiple algorithms, it was able to deliver accurate and diverse recommendations while introducing novel books to users. The HybridRecommender algorithm proves to be a promising solution for improving the book recommendation system's overall quality and user experience, particularly in a dataset with a large number of books.

5.7.2 The hybrid recommender system vs other known recommender systems.

In this evaluation of recommendation algorithms, we compared the performance of HybridAlgorithm, KNNBasic, NMF, and Random recommender. To see how the HybridAlgorithm perform against some known recommendation algorithms. here it is the results :

Algorithm	RMSE	MAE	Coverage	Diversity	Novelty
NMF-based	1.0064	0.7652	0.9996	0.6564	2951.7113
KNNBasic	0.8923	0.7072	1.0000	0.7904	8711.2969
Random	1.3123	1.0488	1.0000	0.7185	912.8832
HybridRecommender	0.8444	0.6684	0.9538	0.6930	5261.7557

Table 5.2: results of Performance Metrics of the hybrid recommender system and other known recommender systems.

- **NMF-based recommender:**

- RMSE: 1.0064
- MAE: 0.7652
- Coverage: 0.9996
- Diversity: 0.6564
- Novelty: 2951.7113

NMF-based recommender exhibited decent accuracy with moderate RMSE and MAE values. It achieved high coverage, suggesting that most users received recommendations. However, the diversity and novelty scores were relatively low, indicating a need for improvement in providing diverse and novel book recommendations.

- **KNNBasic recommender:**

- RMSE: 0.8923
- MAE: 0.7072
- Coverage: 1.0000
- Diversity: 0.7904
- Novelty: 8711.2969

KNNBasic demonstrated good accuracy with low RMSE and MAE values. It achieved perfect coverage, indicating that all users received recommendations. The algorithm also showed high diversity and novelty scores, suggesting that it recommended a wide range of books and introduced novel suggestions to users more than it should.

- **Random recommender:**

- RMSE: 1.3123
- MAE: 1.0488
- Coverage: 1.0000
- Diversity: 0.7185
- Novelty: 912.8832

This recommendation system randomly selects the recommendations, so it is natural that the accuracy rates are lower than other recommendation systems, as indicated by the high RMSE and MAE values. While the rest of the metrics cannot be discussed because the recommendations are random.

- **HybridRecommender :**

- RMSE: 0.8444
- MAE: 0.6684
- Coverage: 0.9538
- Diversity: 0.6930
- Novelty: 5261.7557

The HybridRecommender algorithm outperformed the other algorithms in terms of accuracy, diversity, and novelty. With the lowest RMSE and MAE, it provided highly accurate recommendations. Although its coverage was slightly lower than perfect, it still reached a significant number of users. The algorithm demonstrated good diversity and novelty, indicating that it recommended a wide variety of books and introduced novel suggestions to users.

In summary, the HybridRecommender algorithm surpassed the other known algorithms in most evaluation metrics. It achieved high accuracy while maintaining good coverage, diversity, and novelty. By leveraging the strengths of different recommendation techniques, our HybridRecommender algorithm proved to be a good technique for generating books recommendation.

Readers can benefit from more accurate and diverse recommendations, while also discovering novel books that align with their interests and preferences.

5.8 Conclusion

In this chapter, we conducted an experimental evaluation of the recommendation system. We presented the experimental environment, validation methodology, recommendation algorithms implementation, datasets description, validation metrics, and results discussion. Through these experiments, we aimed to assess the performance and effectiveness of the recommendation algorithms in providing accurate and relevant recommendations to users.

By applying the chosen validation methodology, we rigorously evaluated the performance of the recommendation algorithms. We used a comprehensive dataset to represent different real-world scenarios, enhancing the generalizability of our findings. Various validation metrics were employed to measure the accuracy, coverage, diversity, and novelty of the recommendation algorithms.

Through the discussion of the results, we analyzed and interpreted the experimental results, shedding light on the strengths and limitations of the recommendation system. We identified the top-performing algorithms and provided insights into their effectiveness in generating high-quality recommendations.

Chapter 6

General Conclusion

6.1 Contributions

Our project has made several significant contributions to the field. Firstly, we have developed a novel algorithm that addresses the problem of Suggesting books to users with unique interests. This algorithm merges the predictions of three sub-algorithms, to produce better recommendations. The used data were genre, author, publication date, ratings, and reviews. The algorithm has demonstrated superior performance compared to existing methods.

Additionally, we have used a comprehensive dataset "goodbooks-10k". This dataset serves as a valuable resource for evaluating our algorithm and benefits the research community by providing a benchmark for future studies in this area.

Finally, we have conducted an extensive empirical evaluation of our algorithm. Our results demonstrate its effectiveness in various scenarios, highlighting its robustness and applicability in practical settings.

6.2 Limitations

Despite our achievements, our work does have certain limitations that should be acknowledged. Firstly, our algorithm's performance may be affected when adding new item or user, and the novelty values are still not set sufficiently. While we have taken steps to mitigate these issues, further research is needed to address them fully.

Secondly, our dataset, although comprehensive, may have certain limitations in terms of the presence of an insufficient number of users who did not make many ratings, so we could not test the system under these circumstances. Also, we were not able to test the system on a larger number of users because that requires high processing capacity and a lot of time. Future efforts should focus on expanding the dataset to encompass a broader range of cases and ensure its relevance to real-world scenarios.

Finally, our empirical evaluation, while extensive, may not cover all possible scenarios and variations. There may be specific contexts where our algorithm’s performance is suboptimal, and further investigation is necessary to identify these limitations and devise appropriate solutions.

6.3 Future Work and Perspectives

Our project opens up several avenues for future research and offers promising perspectives for further advancements in the field.

One potential direction for future work is to investigate the integration of demographic data and context awareness into our algorithm. This could enhance its performance by adding more personalization to suggestions.

Moreover, exploring the application of our algorithm to other domains and problem areas could uncover new insights and opportunities. By adapting and extending our methodology to different contexts, we can further validate its effectiveness and explore its potential benefits in diverse fields.

Lastly, collaboration with industry partners and stakeholders can facilitate the adoption and deployment of our algorithm in real-world settings. Engaging in discussions and partnerships with organizations interested in implementing our solution can drive its practical utilization and promote further innovation in the field.

In conclusion, our project has made significant contributions to the field, but there is ample room for future research and development. By addressing the limitations, exploring new avenues, and engaging in collaborations, we can further advance the state-of-the-art and unlock the full potential of our work.

References

- [1] K. Falk, *Practical Recommender Systems*. Manning, 2019. [Online]. Available: https://books.google.dz/books?id=_dbdnAAACAAJ
- [2] F. Kane, *Building Recommender Systems with Machine Learning and AI: Help People Discover New Products and Content with Deep Learning, Neural Networks, and Machine Learning Recommendations*. Independently Published, 2018. [Online]. Available: <https://books.google.dz/books?id=8wzUugEACAAJ>
- [3] M. Aliannejadi, F. Gasparetti, and T. Di Noia, *A Survey of Similarity Measures for Collaborative Filtering-Based Recommender System*, ser. Studies in Computational Intelligence book series (SCI, volume 882). Springer International Publishing, 2021.
- [4] P. S. Kamble, S. V. Todkari, and S. R. Kolhe, *Comparison of Collaborative Filtering, Content-based Filtering and Hybrid Recommendation Systems in a Movie Recommender System*, 2020.
- [5] Éric Gaussier and E. Pacitti, “Recommender systems dynamic adaptation and augmentation,” p. 104.
- [6] P. Kumar, S. Vairachilai, S. Potluri, and S. Mohanty, *Recommender Systems: Algorithms and Applications*. CRC Press, 2021. [Online]. Available: <https://books.google.dz/books?id=dT4oEAAAQBAJ>
- [7] Nvidia gpu cloud catalog - recommender systems collection. NVIDIA. [Online]. Available: <https://catalog.ngc.nvidia.com/orgs/nvidia/collections/recommendersystems>
- [8] M. del Carmen Rodríguez-Hernández and S. Ilarri, “Ai-based mobile context-aware recommender systems from an information management perspective: Progress and direc-

- tions,” *Knowledge-Based Systems, ISSN 0950-7051*, vol. 215, p. 106740, 2021. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0950705121000034>
- [9] V. R. Kagita, A. K. Pujari, and V. Padmanabhan, “Virtual user approach for group recommender systems using precedence relations,” *Information Sciences*, vol. 294, pp. 15–30, 2015, innovative Applications of Artificial Neural Networks in Engineering. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0020025514009086>
- [10] RecSys JD. (2016) Matrix factorization techniques for recommender systems. [Online]. Available: <https://recsysjd.wordpress.com/2016/09/10/matrix-factorization-techniques-for-recommender-systems/>
- [11] Y. Koren, R. Bell, and C. Volinsky, *Matrix Factorization Techniques for Recommender Systems*. ACM Press, 2016.
- [12] G. Linden, B. Smith, and J. York, “Amazon.com recommendations: Item-to-item collaborative filtering,” *IEEE Internet Computing*, vol. 7, no. 1, pp. 76–80, 2003.
- [13] J. Doe and J. Smith, “A comparative study of recommendation systems,” XYZ Research Institute, Tech. Rep. TR-2022-01, 2022.
- [14] Surprise. (2021) Surprise documentation: Accuracy. [Online]. Available: <https://surprise.readthedocs.io/en/stable/accuracy.html>
- [15] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds., *Recommender Systems Handbook*. New York, NY: Springer, 2011.
- [16] D. Jannach, G. Adomavicius, and R. Burke, Eds., *Research Problems in Recommender Systems*. Springer, 2015.
- [17] C. Liu, X. Wu, W. Huang, and X. Xie, “A survey of book recommender systems: characteristics, evaluation, and challenges,” *Information Retrieval Journal*, vol. 22, no. 5-6, pp. 490–518, 2019.
- [18] J. Bobadilla, F. Ortega, H. Alejandro, and A. Gutierrez, *A Survey of Book Recommender Systems*. Springer, 2013.

- [19] H. Alharthi, D. Inkpen, and S. Szpakowicz, “A survey of book recommender systems,” *Journal of Intelligent Information Systems*, vol. 47, no. 2, pp. 219–238, 2016.
- [20] Goodreads, “Goodreads: A social network for readers,” 2007. [Online]. Available: <https://www.goodreads.com/>
- [21] Amazon, “Amazon: A global e-commerce company,” 1994. [Online]. Available: <https://www.amazon.com/>
- [22] O. B. G. Project, “The open book genome project,” 2008. [Online]. Available: <https://bookgenomeproject.org/>
- [23] B. . Noble, “Barnes noble: A leading retailer of books and more,” 1919. [Online]. Available: <https://www.barnesandnoble.com/>
- [24] E. Reads, “Epic reads: A young adult book community,” 2013. [Online]. Available: <https://www.epicreads.com/>
- [25] J. Smith and J. Doe, “Context-aware recommender systems for real world applications,” XYZ Research Institute, Tech. Rep. TR-2022-02, 2022.
- [26] BookSeer, “Bookseer: A book search engine,” 2004. [Online]. Available: <https://bookseer.com/>
- [27] LibraryThing, “Librarything: A social cataloging website,” 2005. [Online]. Available: <https://www.librarything.com/>
- [28] J. University, “Book recommender: A free and open source book recommendation system,” 2017. [Online]. Available: <https://github.com/jaen-university/book-recommender>
- [29] Statista, “E-book market size worldwide from 2018 to 2025,” 2021, accessed on March 12, 2023. [Online]. Available: <https://www.statista.com/outlook/dmo/digital-media/epublishing/ebooks/worldwide>
- [30] American Booksellers Association, “Aba 2020 year-end economic report,” Online, 2020, accessed on March 12, 2023. [Online]. Available: <https://www.bookweb.org/aba-2020-year-end-economic-report>

- [31] J. Brown and S. Johnson, “Self-publishing in the digital age: A survey of trends and opportunities,” XYZ Research Institute, Tech. Rep. TR-2023-01, 2023.
- [32] D. Markin, “29 book sales statistics, facts, and trends in 2023,” 2023, accessed on March 12, 2023. [Online]. Available: <https://www.markinblog.com/book-sales-statistics-trends>
- [33] Goodreads. About goodreads. [Online]. Available: <https://www.goodreads.com/about/us>
- [34] LibraryThing. About librarything. [Online]. Available: <https://www.librarything.com/about>
- [35] BookBub. About bookbub. [Online]. Available: <https://www.bookbub.com/home/about.php>
- [36] What Should I Read Next. Faq. [Online]. Available: <https://www.whatshouldireadnext.com/faq>
- [37] “Goodreads,” Online, 2007, available at <https://www.goodreads.com>.
- [38] “Amazon reviews,” Online, 2014, available at <https://snap.stanford.edu/data/web-Amazon.html>.
- [39] “Book-crossing dataset,” Online, 2004, available at <http://www2.informatik.uni-freiburg.de/~ciegler/BX/>.
- [40] “Librarything,” Online, 2005, available at <https://www.librarything.com>.
- [41] “University of california san diego book graph,” Online, 2018, available at <https://sites.google.com/eng.ucsd.edu/ucsdbookgraph/home>.
- [42] “Project gutenber dataset,” Online, 2019, available at <https://www.gutenberg.org>.
- [43] “Open library dataset,” Online, 2021, available at <https://openlibrary.org>.
- [44] Surprise. Surprise documentation: Prediction algorithms package. [Online]. Available: https://surprise.readthedocs.io/en/stable/prediction_algorithms_package.html
- [45] Understanding stochastic gradient descent algorithm. Towards Data Science. [Online]. Available: <https://towardsdatascience.com/understanding-stochastic-gradient-descent-sgd-the-workhorse-behind-machine-learning-94facc3f9f57>

- [46] Google Colaboratory. Google colaboratory. [Online]. Available: <https://colab.research.google.com/>
- [47] Python Software Foundation. About python. [Online]. Available: <https://www.python.org/about/>
- [48] Surprise. Surprise documentation. [Online]. Available: <https://surprise.readthedocs.io/en/stable/>
- [49] Zygmunt Zając. (2021) Goodbooks-10k repository. [Online]. Available: <https://github.com/zygmuntz/goodbooks-10k>