PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

Ministry of Higher Education and Scientific Research

UNIVERSITY OF ECHAHID HAMMA LAKHDAR – EL OUED

FACULTY OF EXACT SCIENCE

Computer Science Department

ACADEMIC LICENCE

Presented by:

- MIMOUNA BOUZEGAG
- ATIKA BERRACHED
- AYA GABOUSSA

Under the supervision of:

- D. Guia Sana Sahar

Title:

Building a customer product recommendation.

This thesis was discussed on May 25th, 2025.

Defense Committee

Dr. Khalaifa Abdelnasser – *Supervisor*, University of Martyr Hamma Lakhdar

Dr. Soultani Khaled –*Examiner*, University of Martyr Hamma Lakhdar


بسم الله الرحمن الرحيم
١٤٣٨

إهداء

الحمد لله حمدا كثيرا طيبا مباركا فيه، أن وفقنا و يسر لنا إتمام مذكرتنا، بغير حولٍ منا ولا قوة، وما ذلك إلا بفضل الله وكرمه. ولا يسعنا في هذا المقام إلا أن نُعبّر عن شكرنا وامتناننا لكل من ساندنا ووقف إلى جانبنا، فقد قال رسول الله صلى الله عليه وسلم: **"من لا يشكر الناس لا يشكر الله"**.

نهدي هذه المذكرة إلى **أهالينا، آبائنا وأمهاتنا،** الذين كان لهم بعد الله الفضل في بلوغ هذه المرحلة، فكانوا الداعمين الأوائل، والمصدر الدائم للقوة والدعاء. كما لا ننسى **إخوتنا،** الذين كان لهم نصيب في مسيرتنا، دعمًا ومساندة.

كما تتقدّم الطالبتان **بوزقاق ميمونة وبالراشد عتيقة** بخالص الشكر للطالبة **عوادي محاسن فضيلة** من قسم الرياضيات و اسرتها كذلك، على دعمها ووقوفها إلى جانبنا، فكانت نعم الأخت ونِعم الرفيقة، حفظها الله ورعاها وبشّرها بجنته.

كما أتقدم أنا، الطالبة **بوزقاق ميمونة،** بإهداء خاص لنفسي التي قاومت، وصبرت، رغم التعب، ولكل من أراد لي الخير، أو دعا لي في الخفاء دون علمي، ولكل أستاذ ترك أثرا طيبا في طريقي، ولكل من كان له الفضل من بعد الله في توعيتي، بان أدرك معنى الاستقامة الراشدة. وأخص بالإهداء **خديجتي و عتيقة بالراشد و اسرتها و الاستاذة حبيبة منير و حليلو الأخضر وعائلته.**

وتخص الطالبة **بالراشد عتيقة** بالشكر **جدّها وجدّتها، وبوزقاق ميمونة وأسرتها الفاضلة جزاهم الله كل خير وأسعدهم في الدارين** و كذلك أستاذتها **فايزة الحربي، زغدود هاجر، وليلى صحراوي،** وكامل طاقم التعليم بـ **متوسطة أحمد بوزقاق بالمغير،** على ما قدموه من علم وتوجيه، وكذا صديقاتها **نسرين باوية، جهيدة ثابت، مريم جروني، وآية بوعسرية.**

كما نرفع أكفّ الدعاء لإخواننا المستضعفين في **غزة، وفلسطين، وسوريا، والسودان، والعراق، واليمن، ولبنان، وليبيا، والروهينجا، والإيغور، وسائر بلاد المسلمين،** سائلين الله أن يرحم موتاهم، ويشفي جرحاهم، ويفك أسر المأسورين، ويحفظهم بحفظه من كل سوء، وأن يرزقهم الأمن والنصر والتمكين، ويجعل لهم دارًا خيرًا من دارهم، ومقامًا كريمًا في جنات النعيم.

والله تعالى أعلم، وصلّى الله وسلّم على نبينا محمد، وعلى آله وصحبه أجمعين، ومن تبعهم بإحسان إلى يوم الدين.

وآخر دعوانا أن الحمد لله رب العالمين.

شكر

بسم الله الرحمن الرحيم

نتقدم بجزيل الشكر وعظيم الامتنان للاستاذة الفاضلة **قية سحر سناء** على كل ما بذلتِه من جهد ودعم ومتابعة خلال إعداد هذه المذكرة. لقد كنتِ نعم الموجّه والمشرفة الحريصة، الحاضرة بتوجيهاتك الدقيقة ونصائحك القيّمة في كل مرحلة من مراحل هذا العمل. لقد كان

لملاحظاتها البنّاءة ومداخلاتها السديدة الأثر الكبير في إثراء هذا البحث، وتحسين مستواه العلمي والمنهجي. كما أن تشجيعها المستمر كان وقودًا لنا في الأوقات التي شعرنا فيها بالإرهاق أو التردد.

لم تبخل علينا بعلمها، ولا بوقتها، ولا بصبرها، بل كانت دومًا مثالًا للعطاء والتفاني، تمدّ يد العون دون كللٍ أو ملل، حريصة على أن يظهر عملنا بأفضل صورة ممكنة. وقد تعلمنا منها الكثير، ليس فقط في مجال البحث الأكاديمي، بل في معاني الصبر، والانضباط، والدقة، والعمل المتقن.

نسأل الله أن يبارك في علمها وعملها، وأن يجزيها عنا خير الجزاء، ويجعل كل ما قدمته لنا في ميزان حسناتها، ويرفع مقامها في الدنيا والآخرة، ويزيدها فضلًا ورفعة.

مع خالص الشكر والتقدير والاحترام.

Abstract

Product recommendation systems are pivotal in enhancing user experience by suggesting relevant products based on past behavior, preferences, or similarities with other users. For businesses, a well-designed recommendation system can significantly improve customer acquisition, retention, and satisfaction.

The objective of this report is to develop a product recommendation system by analyzing large datasets, building a predictive model using deep learning, and deploying it as a web application. Specifically, the study investigates Neural Collaborative Filtering (NCF), a modern approach that models user-item interactions through neural networks. This model provides accurate and effective recommendations by uncovering hidden relationships between users and items, significantly improving the accuracy and efficiency of product recommendation systems compared to non-deep learning methods.

Keywords: Recommender Systems, Collaborative Filtering, Neural Collaborative Filtering, Neural Networks, Deep Learning, Artificial Intelligence.

ملخص

تُعد أنظمة توصية المنتجات أداةً محورية في تحسين تجربة المستخدم، حيث تقوم باقتراح منتجات ذات صلة بناءً على السلوكيات السابقة للمستخدمين أو تفضيلاتهم أو أوجه التشابه مع مستخدمين آخرين.

أما بالنسبة للشركات، فإن نظام التوصية المُحكَم التصميم يُسهم بشكل فعّال في تعزيز اكتساب العملاء والاحتفاظ بهم ورفع مستوى رضاهم.

تسعى هذه الدراسة إلى تطوير نظام توصية منتجات عبر تحليل مجموعات بيانات ضخمة ، وبناء نموذج تنبؤي باستخدام تقنيات التعلم العميق ، ثم نشره كتطبيق ويب.

ويركز البحث تحديدًا على أسلوب التصفية التعاونية العصبية كنهج(NCF) حديث يعتمد على الشبكات العصبية في نمذجة التفاعلات بين المستخدمين والمنتجات.

ويتميز هذا النموذج بقدرته على تقديم توصيات دقيقة وفعّالة من خلال الكشف عن العلاقات غير الظاهرة بين المستخدمين والمنتجات، مما يحقق تحسنًا ملحوظًا في دقة وكفاءة أنظمة التوصية مقارنةً بالأساليب التقليدية غير القائمة على التعلم العميق.

الكلمات المفتاحية: أنظمة التوصية، التصفية التعاونية، التصفية التعاونية العصبية، الشبكات العصبية، التعلم العميق، الذكاء الاصطناعي.

Contents

List of Figures

List of Tables

إختصارات

الاختصار		المصطلح الإنجليزي	المصطلح العربي
AI	:	Artfiicial Intelligence	الذكاء الاصطناعي
ML	:	Machine Learning	تعلم الآلة
DL	:	Deep Learning	التعلم العميق
RSs	:	Recommender System	نظام التوصية
NCF	:	Neural Collaborative Filtering	التصفية العصبية التعاونية
CF	:	Collaborative Filtering	التصفية التعاونية
NN	:	Neural Network	الشبكة العصبية
CBF	:	Content-Based Filtering	التصفية المعتمدة على المحتوى
MLP	:	Multi-Layer Perceptron	متعدد الطبقات الإدراكي
GMF	:	Generalized Matrix Factorization	التحليل العاملي للمصفوفات
NeuMF	:	Neural Matrix Factorization	التحليل العصبي للمصفوفات
.csv	:	Comma-Separated Values	ملف مفصول بفواصل
.pkl	:	Pickle File Format	ملف حفظ باستخدام Pickle
.pth	:	PyTorch Model File	ملف نموذج PyTorch
API	:	Application Programming Interface	واجهة برمجة التطبيقات

Introduction

The world is witnessing a rapid digital transformation that has made **e-commerce** a fundamental pillar of the modern economy. With the increasing number of users and the diversity of their interests and behaviors, the need for intelligent solutions that enhance customer experience through personalized suggestions has become essential. In this context, **recommender systems** emerge as powerful tools that rely on data analysis to discover patterns and deliver tailored content to each user.

This report explores the topic of recommender systems within the framework of e-commerce, focusing on a a state-of-the-art deep learning approach known as **Neural Collaborative Filtering (NCF)** which utilizes neural networks to represent both users and items as embedding vectors. This allows the model to capture complex interaction patterns and deliver accurate and personalized recommendations. The proposed architecture combines Generalized Matrix Factorization (GMF) and Multi-Layer Perceptron (MLP) to improve recommendation accuracy and efficiency. This study aims to demonstrate the effectiveness of the Neural Collaborative Filtering model compared to traditional methods and promote the adoption of artificial intelligence techniques in the development of modern recommender systems.

The work is structured into three main chapters:

- The **first chapter** presents a comprehensive overview of e-commerce and recommender systems, discussing their types, importance, and the challenges they face in this domain.
- The **second chapter** delves into the theoretical and technical aspects of the NCF model, introducing machine learning and deep learning, explaining the architecture of the model, its sub-models, training process, and evaluation methods.
- The **third chapter** focuses on the practical implementation, including the design and development of the model using the PyTorch framework. It details data preparation, model construction, performance evaluation using standard metrics, and discusses the challenges encountered during the implementation process.

This study represents a modest contribution towards a deeper understanding of intelligent technologies in e-commerce, aiming to build more accurate and effective recommender systems.

Chapter 1

E-commerce and Recomender systems

1.1 Introduction

With technological advancements and the rise of the internet, e-commerce has become a key method for selling products and services. However, with so many options available, customers struggle to find the right products. To address this issue, recommender systems were introduced to provide personalized suggestions, improving shopping experiences and increasing sales.

1.2 E-commerce overview

1.2.1 Definition

E-commerce refers to the use of electronic means and technologies to conduct commercial activities, including the sale, purchase, transfer, or exchange of products, services, and information. It encompasses interactions within businesses, between businesses (B2B), and between businesses and consumers (B2C). The delivery of goods or services may occur over the Internet or through other channels.

According to the International Organization for Standardization (ISO), e-commerce is a broad term describing the exchange of information between enterprises and between enterprises and customers. The Global Information Infrastructure Committee defines it as economic activities utilizing electronic communications, enabling individuals and businesses to purchase products, advertise goods, and process transactions.

E-commerce integrates various technologies such as information systems, online platforms, and digital payment solutions to facilitate seamless commercial operations, making it a key driver of modern digital economies.[29, 21]

1.2.2 E-commerce types

E-commerce can be categorized into six main types: As shown in Figure 1.1.

1. **B2B (Business to Business)**: Transactions between businesses, such as suppliers selling raw materials to manufacturers.

2. **B2C (Business to Consumer)**: Transactions between businesses and consumers, like buying products online.

3. **C2C (Consumer to Consumer)**: Transactions between consumers, such as selling items on platforms like eBay or social media.

4. **C2B (Consumer to Business)**: Consumers provide services or products to businesses, like offering photos or marketing ideas.

5. **C2A (Consumer to Administration)**: Transactions between individuals and government administrations, like paying taxes online or applying for government services.

6. **B2A (Business to Administration)**: Transactions between businesses and government administrations, such as companies submitting reports or providing services to government agencies.

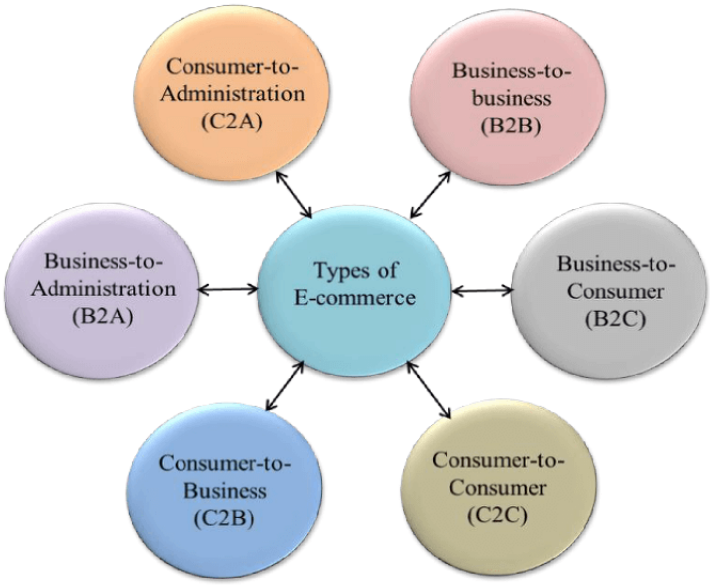


Figure 1.1: Schematic illustration of the Types of the E-commerce based on their Characteristics

1.2.3 E-commerce components

E-Commerce systems consist of key components that ensure secure and efficient transactions, including customer purchases, bank processing, business management, compliance, distribution, and authentication for security.

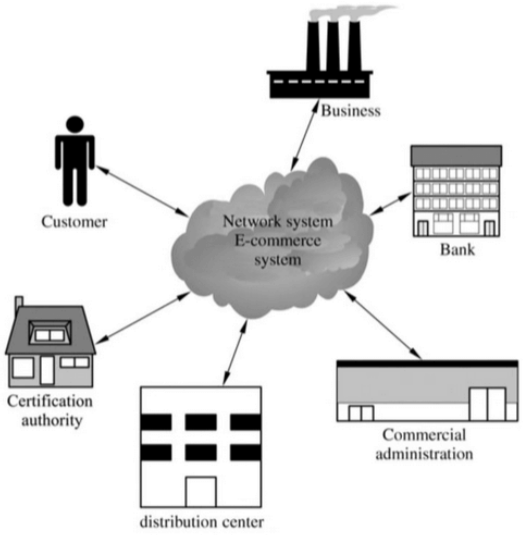


Figure 1.2: Components of E-Commerce

1.3 Recommender Systems

A recommendation system (or recommender system) is a class of machine learning that uses data to help predict, narrow down, and find what people are looking for among an exponentially growing number of options.

1.3.1 Definition

A recommendation system is an **AI algorithm**, often using **machine learning**, that analyzes **Big Data** to suggest products to consumers based on past purchases, search history, and demographic information. These systems help users discover products they may not have found on their own, offering personalized recommendations by understanding user preferences through interactions like clicks, likes, and purchases.

Recommender systems are favored by content and product providers, as they guide consumers to products and services that match their interests, such as books, videos, and clothing..[25]

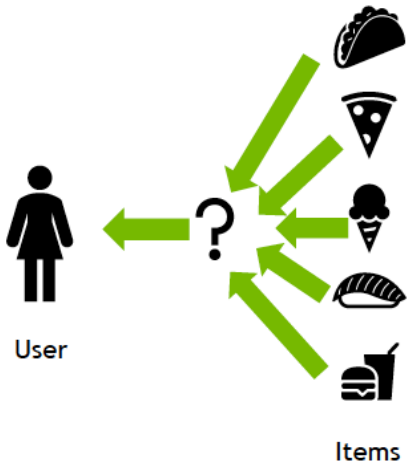


Figure 1.3: Recommender Systems

1.3.2 Approaches

Known user preferences are represented as a matrix of n users and m items, where each cell $r_{u,i}$ corresponds to the rating given to item i by user u. The user ratings matrix is typically sparse, as most users do not rate most items. The recommendation task is to predict what rating a user would give to a previously unrated item. Typically, ratings are predicted for all items that have not been observed by a user, and the highest-rated items are presented as recommendations. The user under current consideration for recommendations is referred to as the *active user*.[23]

		Items					
		1	*2*	...	*i*	...	*m*
Users	*1*	5	3		1	2	
	2		2				4
	:			5			
	u	3	4		2	1	
	:					4	
	n			3	2		
	a	3	5		?	1	

Figure 1.4: User ratings matrix, where each cell $r_{u,i}$ corresponds to the rating of user u for item i. The task is to predict the missing rating $r_{a,i}$ for the active user a.

The myriad approaches to Recommender Systems can be broadly categorized as:

- Collaborative Filtering (CF): In CF systems, a user is recommended items based on the past ratings of all users collectively.

- Content-based recommending: These approaches recommend items that are similar in content to items the user has liked in the past, or matched to attributes of the user.

- Hybrid approaches: These methods combine collaborative and content-based approaches

1.3.2.1 Collaborative filtering

The collaborative recommender system relies on the contribution of a group of users to share their knowledge to provide better recommendations. It is based on the idea that the collective intelligence of the community is more effective than individual intelligence. By combining shared knowledge, the accuracy and quality of recommendations can be significantly improved, making the system more capable of offering relevant and reliable suggestions.[30]

Working principle of Collaborative Filtering

Collaborative recommender systems offer suggestions to users based on the preferences of others, assuming that users with similar tastes tend to like the same items. These systems employ collaborative filtering, a process that predicts an individual's interests based on others' preferences.
- There are two main types:

- **User-based filtering,** which recommends items based on shared preferences among users.

- **Item-based filtering,** which suggests items similar to those the user has liked in the past.

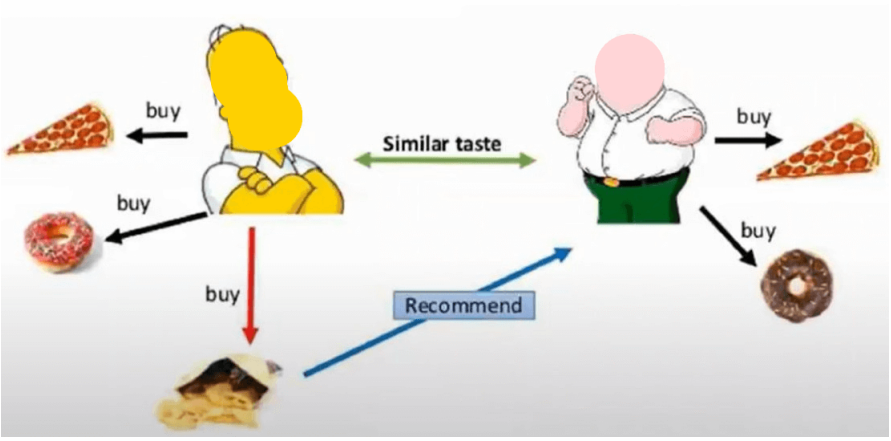


Figure 1.5: Working of Collaborative Filtering

Advantages:

1. It can provide more accurate recommendations than a single recommender system. This is because it can learn from the feedback of multiple users and identify patterns that a single recommender system may not be able to recognize.

2. It can provide recommendations for a wide range of items. This is because it can learn from the feedback of multiple users and identify patterns that a single recommender system may not be able to identify.

3. It can offer suggestions for a wide range of users. This is because it can learn from the feedback of multiple users and identify patterns that a single recommender system might not be able to recognize.

[30]

1.3.2.2 Content-based Filtering

Unlike Collaborative Filtering, which relies only on user ratings, Content-Based Filtering enhances recommendations by analyzing item attributes and user preferences. For example, if a user likes ”Star Wars” and ”Blade Runner,” they might prefer Science Fiction, making ”Twelve Monkeys” a relevant suggestion.

This approach is often treated as an *Information Retrieval* task, where user preferences form a query, and items are ranked based on relevance. Systems like *NewsWeeder* use *tf-idf* word vectors and *cosine similarity* for classification. Alternatively, recommendation

can be framed as a *classification* problem, using models like *Naive Bayes*, *k-nearest neighbor*, and *neural networks* [23].

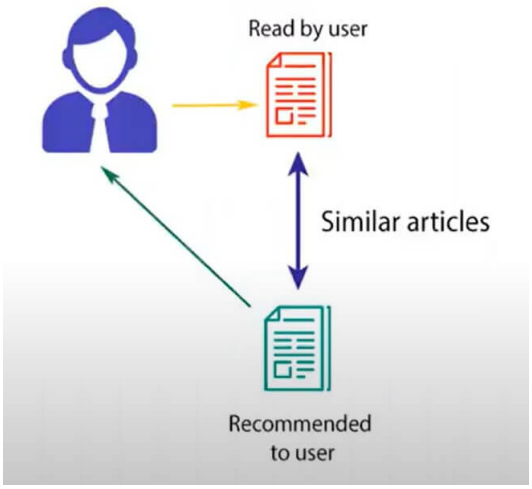


Figure 1.6: Working of Content-based filtering

Advantages:

1. **Increased Accuracy:** Content-based filtering algorithms can provide users with highly accurate recommendations by learning from past behaviors and utilizing this information to refine suggestions.

2. **Improved Relevance:** These algorithms ensure that recommendations are more relevant to the user by analyzing previous interactions and preferences.

3. **Increased Serendipity:** They often suggest items that the user might not have actively searched for, leading to a broader range of recommended products and unexpected discoveries.

4. **Reduced Noise:** Content-based filtering minimizes irrelevant recommendations by filtering out items that do not match the user's interests.

[30]

1.3.2.3 Hybrid recommendations approaches

Hybrid recommendation methods combine content-based and collaborative filtering. Some approaches merge ranked lists, while others use adaptive weighting. Content-boosted CF fills missing ratings using a Naïve Bayes classifier before applying CF, improving accuracy.

Other methods build content-based user profiles for CF. Some hybrids treat recommendation as classification, using rule induction or Latent Semantic Indexing to refine predictions[23].

Advantages:

1. **Increased Customer Engagement:** Hybrid recommendation systems combine both collaborative and content-based filtering techniques to generate more accurate recommendations. This increases the chances that customers engage with the website and make a purchase.

2. **Increased Revenues:** Personalized product recommendations increase the likelihood of purchases, leading to higher revenues for the company.

3. **Improved User Experience:** By using a combination of filtering techniques, these systems provide a more tailored user experience with relevant product recommendations.

4. **Increased Product Awareness:** Customized recommendations help introduce customers to products they might not have discovered otherwise, thereby boosting sales and revenues.

[30]

1.4 Applications of RSs in E-Commerce

RSs embedded in commercial applications provide personalized recommendations to customers, encouraging them to purchase products aligned with their interests, such as watches, cameras, PCs, books, and more. The recommendations are typically based on demographic information or the customer's past purchasing behavior.[19]

RSs play a significant role in enhancing e-commerce sales by:

- Converting browsers into buyers.
- Increasing cross-sell by recommending supplementary products.
- Gaining customer loyalty through personalized experiences.

Examples of commercial recommender systems include:

- Amazon
- eBay
- Drugstore
- Moviefinder
- Reel
- CDNow

1.5 Benefits of RSs in E-Commerce

Recommender systems play a crucial role in enhancing the shopping experience for customers and driving business success in e-commerce. Here are some key benefits:

- **Browsers into buyers:** Visitors to a Web site often look over the site without ever purchasing anything. Recommender systems can help customers find products they wish to purchase. [32]
- **Cross-sell:** Recommender systems improve cross-sell by suggesting additional products for the customer to purchase. If the recommendations are good, the average order size should increase. For instance, a site might recommend additional products in the checkout process, based on those products already in the shopping cart.[32]
  Products are recommended based on the demographics of the consumer, on the top overall sellers on a site, or on an analysis of the past buying behavior of the consumer. The forms of recommendation include suggesting products/services to the consumer, providing personalized product/services information, and providing community critiques.[37]
- **Loyalty:** Recommender systems improve customer loyalty by creating a value-added relationship between the site and the user. Sites invest in learning about their users, utilize recommender systems to operationalize that learning, and present customized interfaces that match customer needs. In return, customers repay these sites by returning to the ones that best align with their preferences.

  The more a customer uses the recommendation system –teaching it what they want–

the more loyal they become to the site. As noted by Pine et al.(1995).

> "Even if a competitor were to build the exact same capabilities, a customer ... would have to spend an inordinate amount of time and energy teaching the competitor what the company already knows."

Finally, fostering relationships between customers can further increase loyalty. Users will return to a site that recommends people they would enjoy interacting with.[32]

- **Improve customer satisfaction:** Customer satisfaction is a key factor in the success of any business, especially in e-commerce, where competition is intense. Companies rely on recommender systems to achieve this goal by:

  1. **Reducing Search Time and Enhancing User Experience**
     - Providing accurate suggestions when visiting supplier websites, minimizing the time and effort needed to find suitable products/services.[37]
     - Offering personalized recommendations based on customer needs, whether through product suggestions, detailed information, or user-generated reviews .or facilitating access to products/services.[37]
  2. **Helping Professional Buyers Make Informed Decisions**
     - In a fast-changing business environment, buyers face an overwhelming amount of data. Recommender systems assist them in analyzing information and selecting the most relevant options.[37]

1.6 Challenges in E-commerce and RSs

1.6.1 Challenges in e-commerce

E-commerce has made it possible to achieve things that were once impossible. However, this technology did not come alone; it also brought some challenges, including:

- **Security challenges(Cyber Security):** Despite businesses' efforts to protect themselves and consumers, hackers constantly seek to bypass security barriers. Recent cases show that even the largest and most well-known companies can be hacked.[3] For example, the **eBay** breach exposed the data of 145 million users, while the **Al-**

ibaba attack compromised information from millions of customers. Additionally, **Equifax** , one of the largest credit agencies, suffered a massive data leak that affected 147 million individuals, resulting in severe financial losses and reputational damage. These cases highlight the growing severity of cyber threats

– **Problem of privacy and data theft:**

. Data is a valuable resource, especially in e-commerce, where the digital shift has led to storing sensitive information electronically, making its security a top priority. With the rise of cyberattacks, data theft has become more dangerous, causing loss of customer trust, financial damages, legal penalties, and threats to individual privacy. Therefore, strengthening cybersecurity and implementing strict measures to protect information has become essential. [5]

– **Rising costs of ensuring security:**

. Cybersecurity threats vary and include social engineering, denial-of-service attacks, malware, and data breaches. Companies worldwide spend huge amounts to combat these threats, which increase every year. However, overcoming these challenges remains complex, as attackers continuously seek new vulnerabilities in individuals, organizations, and technology. [20]

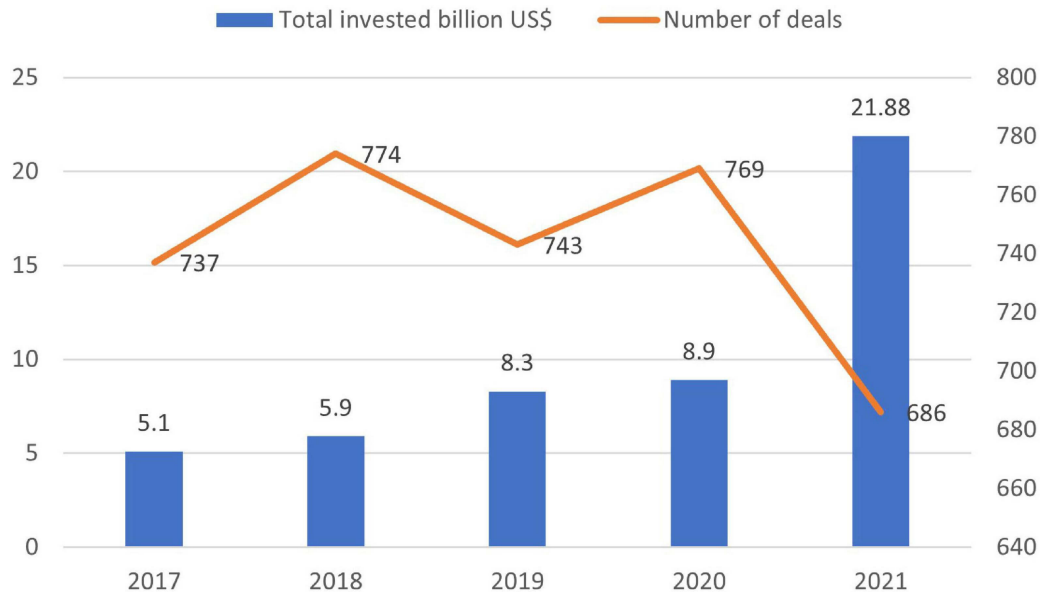


Figure 1.7: Investment in cybersecurity

- **Constant upkeep:** When a business has started as e-commerce, they must be ready to make changes to stay compatible. While technology grows, the systems that support your business must be kept up to date or replaced if needed. There may

be additional overhead in order to keep data bases and applications running. [3]

- **Quality and credibility:** While e-commerce makes everything easily accessible, a consumer cannot touch products until they are delivered to the door. It is important to view the return policy before buying. Always make sure returning goods is an option.[3]
  Additionally, product images on e-commerce websites may be edited or misleading, making the item appear different in terms of quality, color, or size from reality.

1.6.2 Challenges in recommender systems

This subsection describes the most common issues and challenges encountered in deploying RSs, which are considered important in RSs research.[19]

- **Sparse RSs:** Generally, the majority of users do not rate most of the items, leading to a very sparse ratings matrix. This results in a data sparsity problem, reducing the likelihood of finding a set of users with similar ratings. This is a significant drawback of the CF technique. One way to address this issue is by incorporating additional domain information.

- **Cold-start Problem:** This problem arises due to the introduction of new items and new users in RSs. A new item cannot be recommended initially when it has no ratings in a CF system. For instance, MovieLens (movielens.org) cannot recommend new movies until they receive some initial ratings. The new-user problem is even more challenging, as it is difficult to find similar users or create a Content-Based (CB) profile without prior user preferences.

- **Scalability Problem:** One of the most critical issues in RSs is the scalability of algorithms when dealing with large real-world datasets. Managing vast and dynamic datasets generated by user-item interactions, such as preferences, ratings, and reviews, is increasingly challenging. Some recommendation algorithms may perform well on small datasets but become inefficient or ineffective on very large datasets. Advanced large-scale assessment methods are needed to address this challenge.

- **Privacy Issue:** To produce quality personalized recommendations,RSs are bound to gather as much user data as possible and exploit it to the best of their ability. However, this may create a negative impression on the users' minds regarding their

privacy, as the system may appear to know too much about them.Thus, such techniques need to be designed that can sensibly, meticulously, and carefully use user data while ensuring that information about the users' true preferences is not freely accessible to malevolent users.

- **Disavantages of the RS approches:**

1. **Users are reluctant to disclose their information:** Users are reluctant to share their information due to privacy issues.
2. **Users tend to be myopic:** Users tend to focus on the short term and do not consider long-term benefits.
3. **Social loafing:** Social loafing is a problem that often occurs in collaborative recommender systems. It refers to the phenomenon that users are less likely to contribute when they know that others are also contributing.
4. **Difficulty in Determining Similarity:** It can be challenging to accurately measure the similarity between items, affecting recommendation quality.
5. **Computational Complexity:** Generating recommendations can be time-consuming, especially when handling large datasets.
6. **Sensitivity to Similarity Metrics:** The accuracy of recommendations may heavily depend on the choice of similarity measurement method.
7. **Sensitivity to Filtering Method:** Different filtering approaches can yield varying recommendation quality, making the choice of method critical.
8. **Scalability Issues:** Expanding content-based filtering to a vast number of items can be difficult and computationally expensive.
9. **Cost:** Developing and maintaining hybrid recommendation systems can be expensive due to the need for advanced technologies such as machine learning, natural language processing, and data mining.
10. **Complexity:** Building and maintaining these systems require technical expertise, making them challenging to optimize and troubleshoot.
11. **Data Quality:** Poor-quality data can lead to inaccurate recommendations, reducing customer satisfaction and potentially harming the business.

12. **Privacy Concerns:** Handling customer data securely and complying with relevant laws and regulations is essential to maintaining trust and legal compliance.[30]

1.7 Conclusion

In this chapter, we provide an overview of **E-Commerce and Recommender Systems** , discussing their role in enhancing user experience and increasing sales. We also explored key approaches such as collaborative filtering, content-based filtering, and hybrid methods . In the next chapter, we will explore in greater detail **Machine learning algorithms** , focusing on the methods used in our system, where we relied on *collaborative filtering* to improve recommendation accuracy.

Chapter 2

Neural Collaborative Filtering

2.1 Introduction

Recommender systems have witnessed significant advancement through artificial intelligence techniques, especially **machine learning** and **deep learning**, aiming to provide accurate and personalized suggestions to users. Among the most prominent modern models is **Neural Collaborative Filtering (NCF)**, which combines the representational power of **neural networks** with the effectiveness of **collaborative filtering**.

This chapter provides an overview of **machine learning** concepts, explains the role of **deep learning** in **recommender systems**, and then delves into the structure of the NCF model, its different variants, and its training and evaluation mechanisms.

2.2 Machine Learning Overview

2.2.1 what is machine learning ?

Machine Learning is a branch of AI that focuses on developing algorithms that allow computers to learn from data and improve their performance on specific tasks without being explicitly programmed. It relies on identifying patterns from historical data to make predictions or decisions, and is widely applied in areas such as computer vision, medicine, and intelligent recommendation systems.[24]

The following figure shows the relationship between data science, artificial intelligence, machine learning, and deep learning, and how these concepts are hierarchically connected.

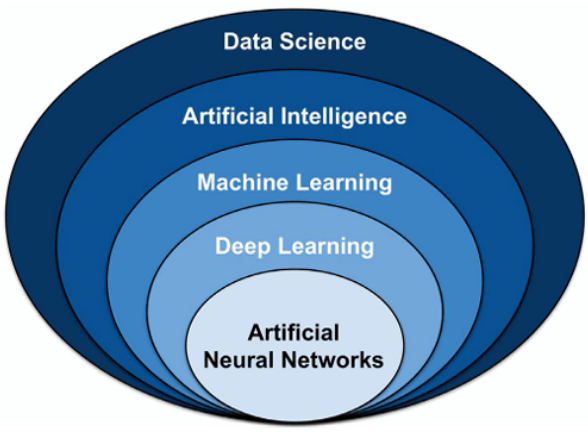


Figure 2.1: The hierarchical relationship between data science, artificial intelligence, machine learning, deep learning, and artificial neural networks.

2.2.2 Machine learning tasks

Machine learning tasks can be broadly classified into the following categories[34]:

- **Supervised Learning**: The model is trained on labeled data, where each input x is paired with an output label t. The objective is to generalize from the training data to predict the output for unseen inputs.
  - *Classification*: The output label is discrete (e.g., spam or not spam).
  - *Regression*: The output is continuous (e.g., predicting temperature).
- **Unsupervised Learning**: The model is provided with unlabeled data and must discover patterns or structure within it. Common tasks include:
  - Clustering
  - Dimensionality Reduction
  - Generative Modeling
- **Semi-supervised Learning**: Combines aspects of supervised and unsupervised learning, using a mixture of labeled and unlabeled data.
- **Reinforcement Learning**: The model learns by interacting with an environment, receiving feedback in the form of rewards or penalties based on its actions. It is suitable for sequential decision-making tasks.

2.2.3 Machine learning algorithms

The following figure illustrates the main classification of machine learning tasks, along with key algorithms used in each category.

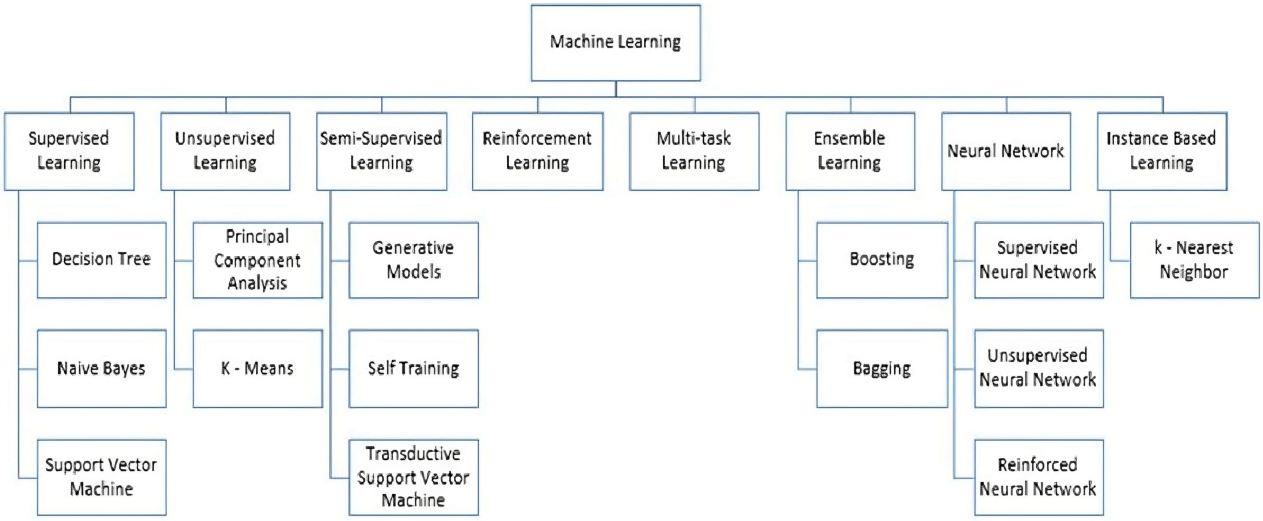


Figure 2.2: A diagram classifying types of machine learning and highlighting common algorithms under each category

2.3 Deep Learning

2.3.1 Deep Learning definition

Deep learning is a subset of machine learning that uses artificial neural networks composed of multiple layers to process and analyze data. When the network includes three or more layers (input, hidden, and output), it is considered ”deep” These models learn from training data to improve prediction accuracy.[6]

2.3.2 Types of Neural Networks

- **Feedforward Neural Networks (FF)**: One of the earliest neural network types. Data flows in one direction through layers until the output is produced.
- **Recurrent Neural Networks (RNN)**: Designed for sequential or time-series data. They retain memory from previous inputs to influence current outputs.
- **Long Short-Term Memory (LSTM)**: A specialized form of RNN that maintains long-term dependencies through memory cells.
- **Convolutional Neural Networks (CNN)**: Common in image processing. Use convolutional and pooling layers to extract spatial features, followed by fully connected layers for classification.
- **Generative Adversarial Networks (GAN)**: Consist of a generator and a discriminator competing in a minimax game to produce realistic synthetic data. [6]

2.3.3 Differences Between ML and DL

- **Machine Learning:**
  ML is the science of training a program or computer system to perform tasks without explicit instructions. ML algorithms process large amounts of data, identify patterns, and predict outcomes for unknown or new scenarios.

- **Deep Learning:**
  Dl is a subset of machine learning that uses neural networks modeled after the human brain. It aims to automate more complex tasks that typically require human intelligence, such as image description, document translation, or speech-to-text conversion.[2]

The following figure illustrates the difference between traditional programming, machine learning, and deep learning in terms of how data is processed and models are built.

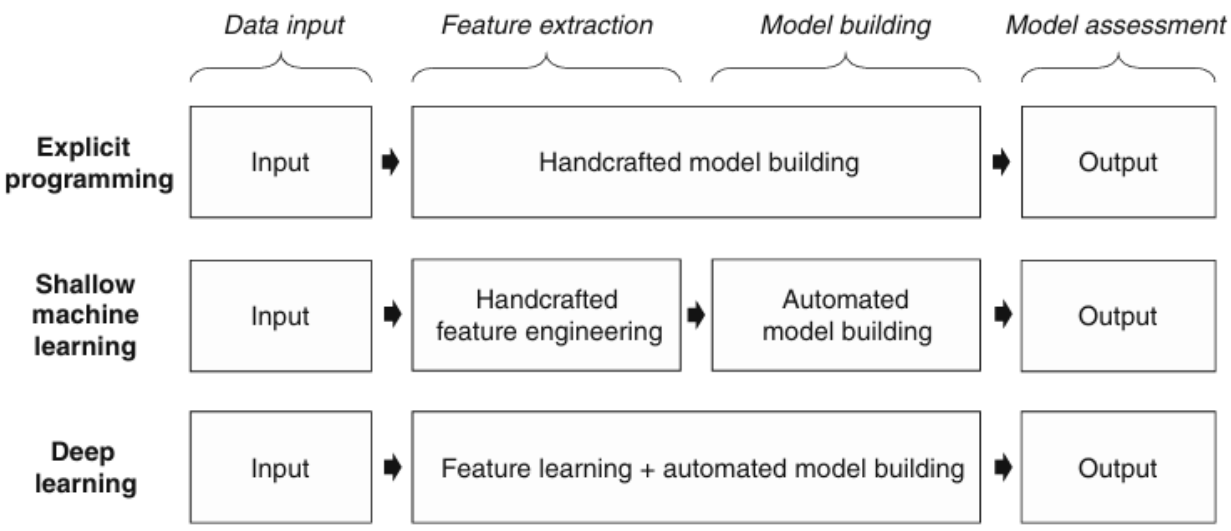


Figure 2.3: Comparison between traditional programming, machine learning, and deep learning in terms of data processing and feature extraction.

2.3.4 Deep Learning in Recommender Systems

In recent years, a radical transformation has occurred in recommender systems, driven by the advanced capabilities of deep learning. These models have become essential tools to enhance user experience and improve service effectiveness across various domains such as E-commerce, advertising, and social media.[36]

Industrial Applications:

Major companies have adopted deep learning-based recommendation models to achieve superior performance:

- **YouTube:** Approximately 60% of video clicks originate from homepage recommendations.
- **Netflix:** Around 80% of watched content is based on recommendations.
- **Google Play:** A recommendation system using a Wide & Deep model was proposed.
- **Yahoo News:** An RNN-based model was implemented for news recommendations.

These models have demonstrated remarkable effectiveness in real-world deployment, showing significant improvements over traditional models through online testing.

Here is a figure showing the Categories of deep neural network based recommendation models:

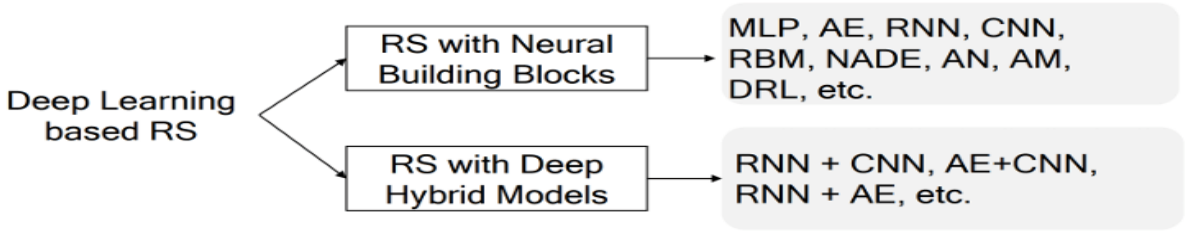


Figure 2.4: Categories of deep neural network based recommendation models.

2.4 Neural Collaborative Filtering (NCF)

2.4.1 NCF Definition

NCF is a modern recommendation model based on deep learning techniques to model user-item interactions. This model combines the strengths of traditional collaborative filtering with the representational power of neural networks, aiming to capture complex non-linear relationships that linear models may fail to represent accurately. NCF is a type of collaborative filtering model that relies on neural networks instead of traditional linear operations. NCF was proposed as a framework that replaces traditional linear matrix factorization techniques with a multi-layer neural network, enabling the modeling of non-linear user-item interactions and thereby improving recommendation accuracy.[17]

2.4.2 NCF Architecture

Neural Collaborative Filtering Framework

We begin by introducing the general framework of **NCF**, emphasizing its ability to learn from implicit data through a probabilistic model that captures the binary nature of user feedback.

Subsequently, we demonstrate that **Matrix Factorization (MF)** can be represented and generalized within the NCF framework. This establishes a connection between traditional linear models and modern neural network-based approaches.

To further explore the use of deep neural networks in collaborative filtering, we propose a concrete implementation of NCF using a **MLP**. This architecture is designed to learn the complex user-item interaction function more effectively than traditional linear models.

The **NCF** model uses a multi-layer neural network to predict user-item interactions, such as watching a movie, reading a book, or purchasing a product.

1. **Input**: Users and items are represented using one-hot encoding.
2. **Embedding Layer**: These sparse representations are transformed into dense vectors that capture latent features of users and items.
3. **Neural Layers**: The embedding vectors are passed through multiple hidden layers to learn complex user-item interaction patterns.
4. **Output**: The final output layer produces a score that estimates the likelihood of interaction between the user and item.
5. **Training**: The model is trained by minimizing the difference between predicted scores and actual interaction data.[17]

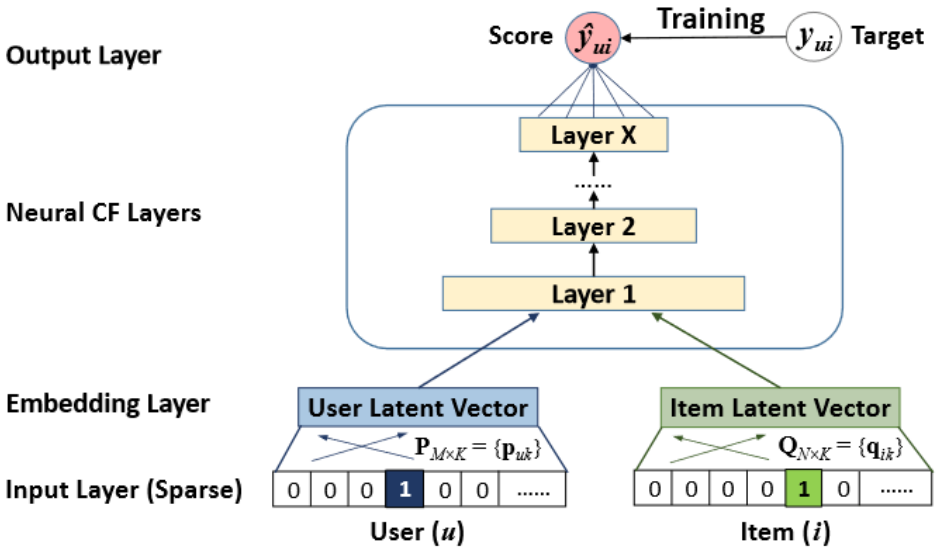


Figure 2.5: Neural collaborative filtering framework

User-Item Embedding

The **embedding layer** is used to initialize the representations of users and items as dense vectors. An **embedding lookup table** is created based on the interaction data between users and items.

Each user u and each item i is represented by an embedding vector, where the user is represented by $e_u \in \mathbb{R}^d$, and the item by $e_i \in \mathbb{R}^d$, with d being the embedding size.

The embeddings are initialized in the following matrix format:

$$E = [e_{u_1}, e_{u_2}, \ldots, e_{u_N}, e_{i_1}, e_{i_2}, \ldots, e_{i_M}]$$

This embedding table serves as the initial state for both user and item embeddings. It is optimized in an end-to-end manner during training, such that the embeddings are updated automatically based on model errors.

While traditional recommendation models rely on embedding user and item IDs and feeding them into an interaction layer to generate predictions, our proposed method propagates these embeddings through a **user-item interaction graph**, enabling the model to capture deeper and more complex structural relationships between users and items.[15]

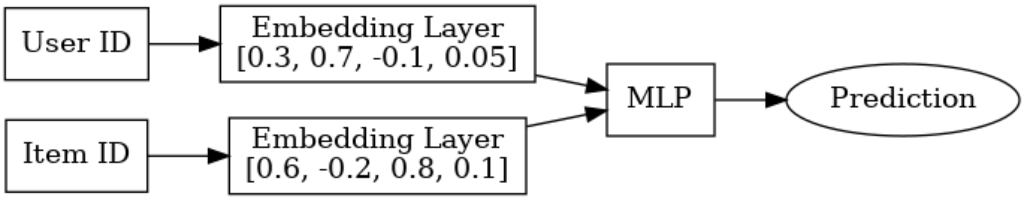


Figure 2.6: User-item embedding

2.4.3 GMF, MLP, and NeuMF Models

Generalized Matrix Factorization (GMF)

GMF is a generalized version of traditional Matrix Factorization. Let $y_{u,i}$ represent the prediction score for the interaction between a user u and an item i. In GMF, the score is computed using the following formula:

$$y_{u,i} = a_{out}(h^T(p_u \circ q_i))$$

where $\circ$ denotes the element-wise product of the user's latent vector p_u and the item's latent vector q_i, h is a vector of weights that is multiplied by the latent product, and a_{out} is the activation function applied to the final weighted sum.

If we use the identity function for a_{out} and a uniform vector of ones for h, this reduces to classic matrix factorization. In the paper [**5**], a sigmoid function was used as the activation function a_{out}, and the weight vector h is learned from the data.[27]

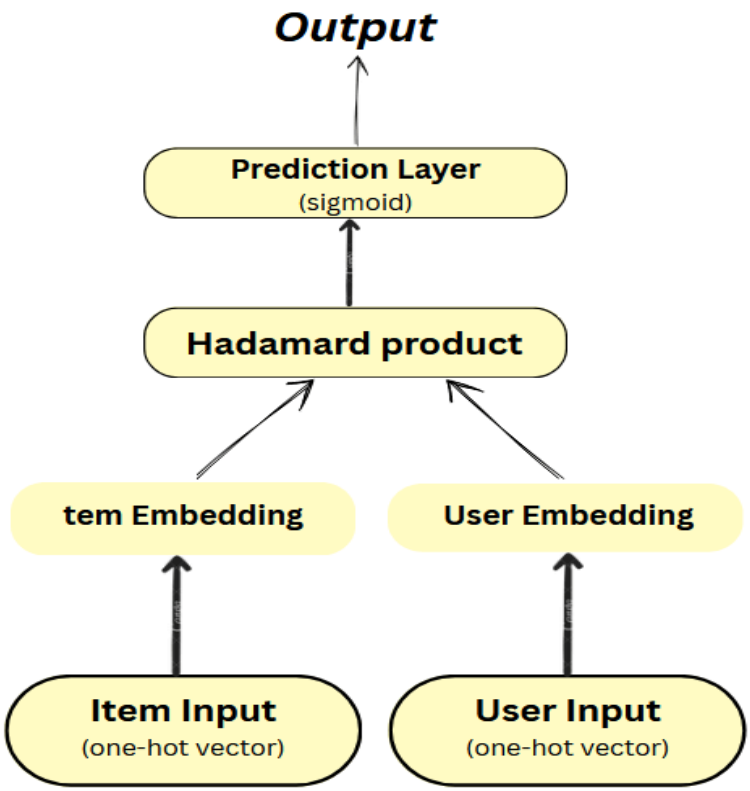


Figure 2.7: Generalized Matrix Factorization (GMF) Architecture

Multi-Layer Perceptron (MLP)

MLP is a type of **feedforward neural network** composed of an input layer, one or more hidden layers, and an output layer. It is designed to model complex relationships between inputs and outputs through layers of artificial neurons, where each neuron processes the input data using weighted connections and a non-linear activation function.

The foundation of MLP is the **Perceptron**, one of the earliest and simplest forms of artificial neural networks. It is based on the **Threshold Logic Unit (TLU)**, which computes a weighted sum of inputs and applies a step function to produce a binary output. Mathematically, the output $h_w(x)$ of a TLU is defined as:

$$h_w(x) = \text{step}(z)$$

Where:

$$z = x^T w = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

Here, x is the input vector and w is the corresponding weight vector.

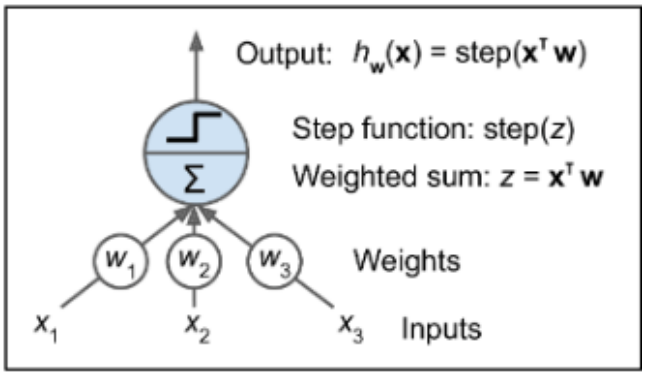


Figure 2.8: Threshold Logic Unit architecture: an artificial neuron that computes the weighted sum of inputs and applies a step function for activation.

In a basic Perceptron, each input is connected to a TLU in a fully connected manner. The outputs of these units are then passed to the next layers. Although historically important, the TLU is rarely used today due to its limitations in learning non-linearly separable patterns.

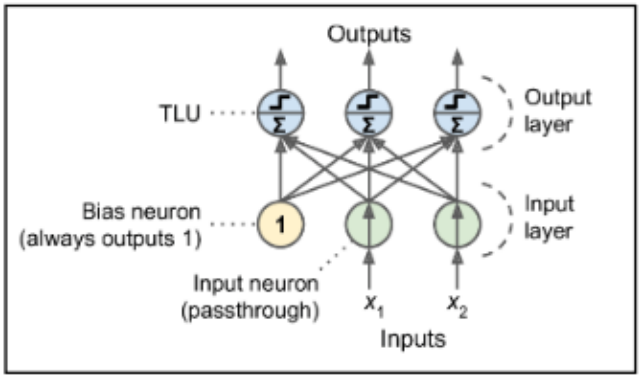


Figure 2.9: A Perceptron architecture with two inputs, one bias and three output neurons.

The **MLP** extends the Perceptron by introducing multiple hidden layers between the input and output layers. Each hidden layer consists of multiple neurons that apply a weighted sum followed by a non-linear activation function, such as **sigmoid**, **tanh**, or more commonly **ReLU (Rectified Linear Unit)**. Each layer, except the output layer, also typically includes a **bias neuron** that helps shift the activation function, enhancing the learning capacity of the network.

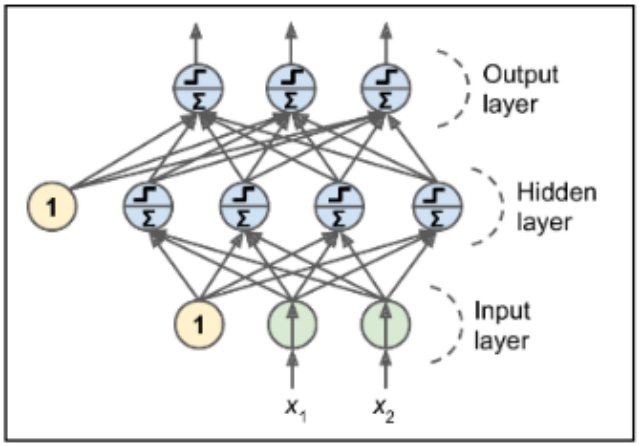


Figure 2.10: An MLP architecture with two inputs, one hidden layer and three outputs.

Among various activation functions, **ReLU** is often preferred in deep learning models due to its ability to mitigate the vanishing gradient problem, promote sparse activations, and enhance convergence speed during training.

In recommendation systems such as **Neural Collaborative Filtering (NCF)**, the MLP component is used to model complex user-item interactions. Specifically, the user latent vector p_u and the item latent vector q_i are concatenated and passed through a series of fully connected layers. Each layer transforms the representation using its own weights, biases, and activation functions.

The general computation in the MLP part of the NCF framework is as follows:

$$z_1 = \phi_1(p_u, q_i) = [p_u \, q_i]$$

$$\phi_2(z_1) = a_2(W_2^T z_1 + b_2)$$

$$\vdots$$

$$\phi_L(z_{L-1}) = a_L(W_L^T z_{L-1} + b_L)$$

$$\hat{y}_{ui} = \sigma(h^T \phi_L(z_{L-1}))$$

Where:

- W_x and b_x: weight matrix and bias vector for layer x
- a_x: activation function used at layer x
- $\phi_L(z_{L-1})$: the final hidden representation

- $\hat{y}_{ui}$: predicted interaction score between user u and item i
- σ: the sigmoid function used to squash the final output to a probability

Empirical studies demonstrate that MLP-based interaction modeling in NCF effectively captures non-linear relationships between users and items, making it a powerful alternative to traditional matrix factorization techniques [27, 17].

Neural Matrix Factorization (NeuMF)

The **NeuMF** model combines two different recommendation paths: GMF and MLP. Each path learns separate embeddings for the users and items, and these embeddings are combined to produce the final result.

The architecture (fig: 2.11) consists of four main components:

1. **Embedding Layer:** In this layer, the discrete user and item IDs are transformed into dense, continuous, and trainable vectors (embeddings). For each user and item, two separate embeddings are learned:
   - One for the GMF path.
   - One for the MLP path.
2. **GMF Path (Generalized Matrix Factorization):** An **element-wise multiplication** is performed between the user's GMF embedding and the item's GMF embedding. The result is a vector with the same dimension as the original embeddings.
3. **MLP Path (Multi-Layer Perceptron):** The user's MLP embedding and the item's MLP embedding are **concatenated** and passed through several fully connected layers with ReLU activation functions. The output of this path has the same dimension as the GMF embeddings.
4. **Prediction:** The outputs from the GMF and MLP paths are concatenated and passed through a final fully connected layer to produce the recommendation score. A **sigmoid function** is then applied to convert the score into a **probability value**, which indicates the user's interest in the item.

By combining GMF and MLP, NeuMF effectively captures both **linear interactions** and **complex non-linear relationships**, making it a powerful recommendation model.[27]

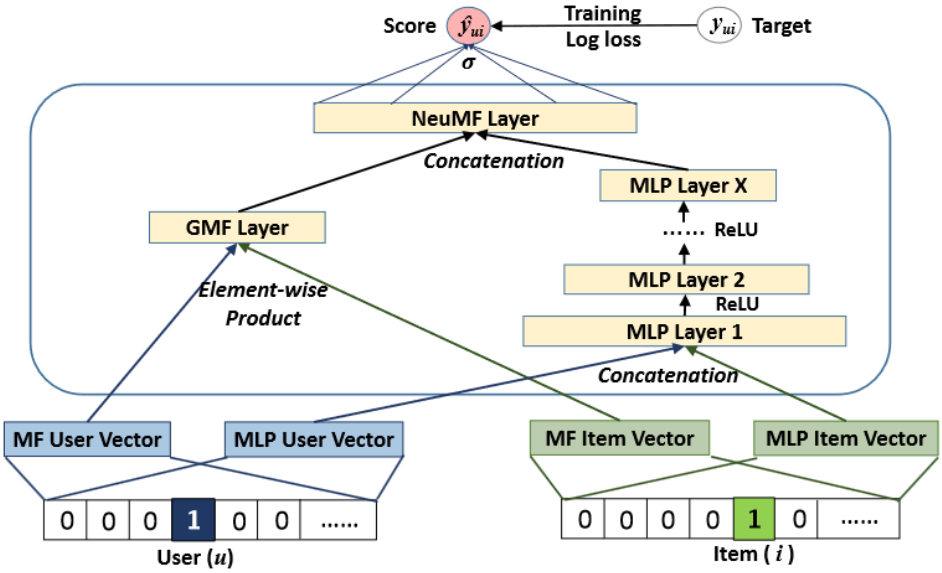


Figure 2.11: Neural matrix factorization architecture.

Table 2.1: Comparison between GMF, MLP, and NeuMF models

Feature	**GMF**	**MLP**	**NeuMF**
Interaction Type	Linear (element-wise product)	Non-linear (deep neural network)	Combination of linear and non-linear
Combination Method	Element-wise multiplication	Concatenation of embeddings	Concatenation of GMF and MLP outputs
Activation Functions	Usually sigmoid	ReLU, sigmoid, or tanh	ReLU (MLP), sigmoid (final layer)
Depth	Shallow (1 layer)	Deep (multiple hidden layers)	Deep (combination of both structures)
Ability to Capture Complex Relations	Limited	High	Very high
Parameters Learned	Embedding weights only	Embeddings, weights, and biases	All parameters from both GMF and MLP
Advantages	Simple and easy to train	Can model complex interactions	Combines strengths of GMF and MLP
Disadvantages	Cannot capture non-linear patterns	More complex, requires longer training	Most complex, higher computational cost

2.4.4 Training and Evaluation

Data Splitting Methods:

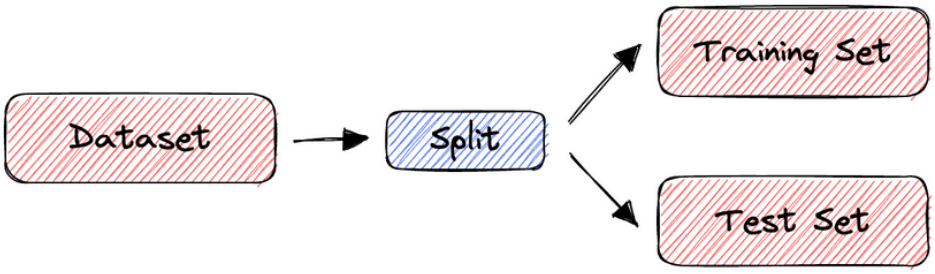


Figure 2.12: Split graph

In the field of machine learning, especially when building models such as **Neural Collaborative Filtering (NCF)**, several data splitting methods are commonly used to evaluate model performance and ensure its ability to generalize to new, unseen data during training. Among the most important of these methods are:

- **Train-Test Split (Holdout Validation):** In this approach, the dataset is divided once into two distinct subsets: a training set used to build the model and a testing set used to evaluate its performance. Sometimes, a third subset called a validation set is separated from the training set to fine-tune hyperparameters. This method is simple and fast, but its results can vary depending on the random partitioning, especially when working with small datasets.[7]

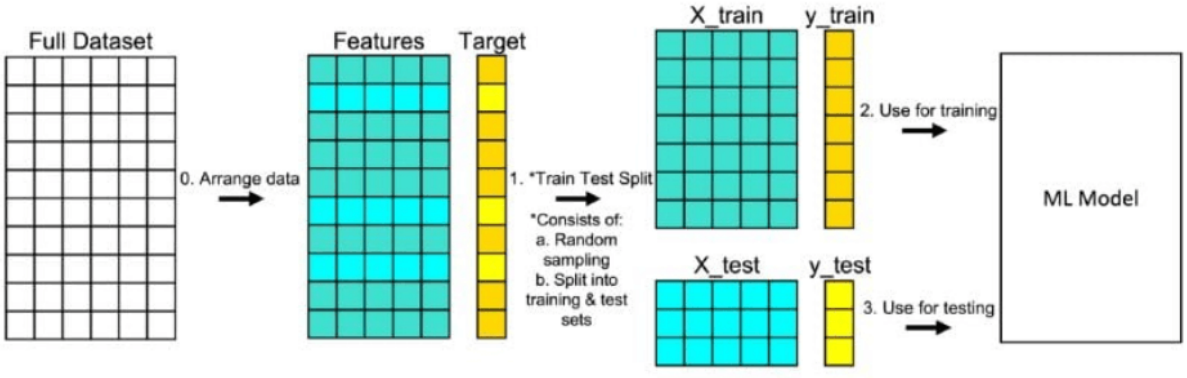


Figure 2.13: figure of Train-Test Split (Holdout Validation) method

- **Leave-One-Out Cross-Validation (LOOCV):** Here, the model is trained on the entire dataset except for one sample, which is used for testing. This process is repeated

for each data point. LOOCV has low bias because all data points are used, but it has high variance and is computationally expensive.[4]

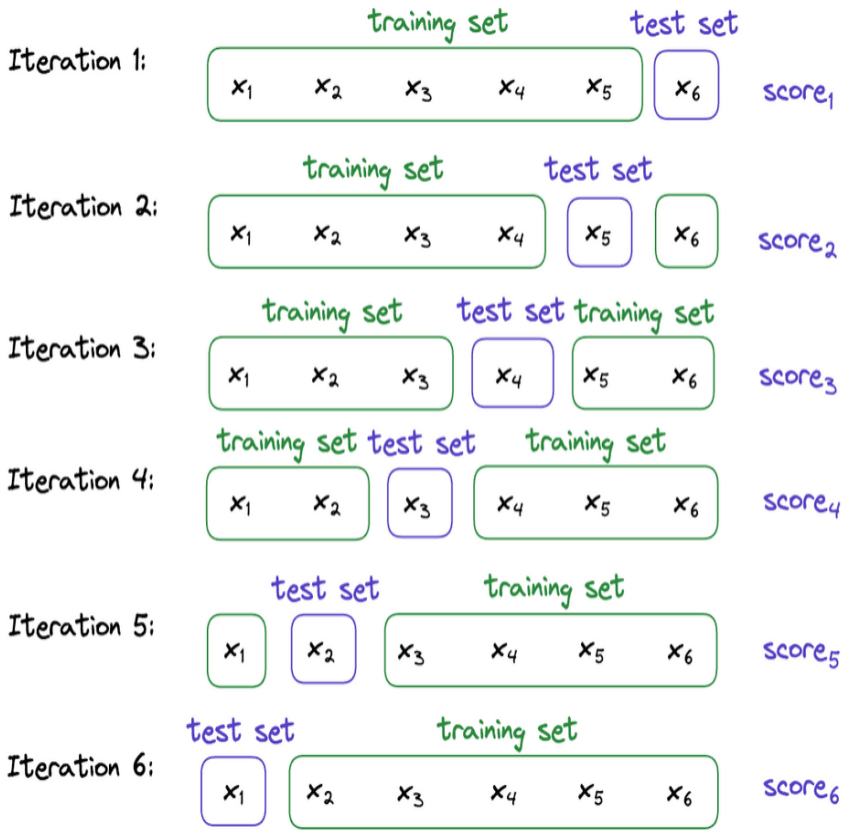


Figure 2.14: Example: For a dataset with 6 samples, we perform 6 iterations. In each iteration, one different sample is used for testing: $S_1 = \{x_1\}$, $S_2 = \{x_2\}$, ..., $S_6 = \{x_6\}$.

- **Stratified Cross-Validation:** This technique ensures that each fold maintains the same class distribution as the original dataset. It is particularly useful for classification problems with imbalanced classes, preserving fairness in each training and testing split.[9]

- **K-Fold Cross-Validation:** The dataset is split into k equal folds. In each iteration, $k - 1$ folds are used for training and one fold is used for testing. This process is repeated k times, and the final performance is estimated by averaging the results across all folds.[9]

Training:

During the training phase of the recommendation system, the goal is to optimize the prediction model by minimizing the discrepancy between the predicted scores and the true interaction values. This is achieved using regression loss functions that quantify the prediction error and guide the learning process.[17]

Among the most widely used regression loss functions are:

- **Mean Absolute Error (MAE)**:
  This loss function calculates the average of the absolute differences between actual and predicted values. It is robust to outliers and belongs to the class of Regression Losses.

- **Mean Squared Error (MSE)**:
  This function measures the average of squared differences between predictions and actual values. It penalizes large errors more heavily and is commonly used in training deep models. It is also a Regression Loss.

- **Root Mean Squared Error (RMSE)**:
  As the square root of MSE, this loss function gives a more interpretable measure of error in the same units as the target variable. It is a Regression Loss and is widely used in evaluating the absolute fit of the model. [35]

In addition to being used during training, MSE and RMSE can also serve as evaluation metrics to assess the model's performance. Their ability to reflect the average magnitude of error makes them suitable for both optimization and performance comparison.

These loss functions help ensure that the model improves over iterations by minimizing the prediction error, leading to better personalization and recommendation accuracy.

Optimization Algorithms

- **Gradient Descent:** A fundamental technique for minimizing the cost function by iteratively adjusting model parameters to improve performance.[11]

- **RMSProp:** An optimization algorithm that uses an adaptive learning rate based on recent gradients, improving efficiency when dealing with large datasets.[13]

- **Adam:** Combines the advantages of both Momentum and RMSProp to dynamically adjust learning rates for model parameters, enhancing training effectiveness.[8]
- **Stochastic Gradient Descent (SGD):** Updates model parameters using a random sample of the data, making it more efficient for large datasets.[12]

2.5 Conclusion

In this chapter, we presented the theoretical foundations of the NCF model. We began with a general overview of machine learning and deep learning concepts, followed by a detailed explanation of the NCF architecture and its main components, along with its training and evaluation process. This theoretical overview sets the stage for the next chapter, which will focus on the practical implementation of this model using real-world data.

Chapter 3

Design and implementation

3.1 Introduction

The previous chapter explores the theoretical foundations of the Neural Collaborative Filtering (NCF) model including its core concepts, architectural components, and training and evaluation mechanisms.

This chapter focuses on putting those principles into practice. We aim to construct a practical model based on real-world data, implementing a complete workflow that involves data preprocessing, model design, training the neural network, and evaluating the resulting performance. This applied study seeks to assess the effectiveness of the NCF model in providing accurate recommendations, and to explore its potential for deployment in real-life recommendation systems.

3.2 Hardware Environment

In this section, we present the technical specifications of the **hardware environment** used to implement and train the recommendation system model. The development begins on a personal computer (3.1), while the training phase relies on the **Google Colab** platform to take advantage of the parallel processing capabilities of a **Graphics Processing Unit (GPU)**, with the goal of accelerating the training process and improving performance efficiency.

Table 3.1: Hardware Specifications

Component	Specification
Processor	Intel Core i5-1135G7, 11th Gen, 4 Cores / 8 Threads
RAM	8 GB
System Type	64-bit Operating System (x64-based processor)
Device Name	DESKTOP-D6AS8OK

3.2.1 Cloud Environment: Google Colab

Due to the hardware limitations of our local machine, particularly in terms of memory and processing power, we adopted **Google Colaboratory (Google Colab)** as the primary environment for training our model.

Google Colab is a free, cloud-based integrated development environment (IDE) based on Jupyter notebooks. It requires no setup and allows code execution, saving and sharing of analyses, and access to powerful computational resources such as GPUs, all directly from the browser.[14]

Figure 3.1: Google COLAB logo.

Using GPU acceleration in Colab significantly helped us reduce training time and achieve better performance, especially when working with large datasets and deep neural network models.

The performance of the central processing unit (CPU) of the computer and the graphics processing unit (GPU) of the Colab platform were compared, and the results showed the following:

- **CPU**: Good for sequential processing but relatively slow in complex neural operations.
- **GPU**: Provides significant acceleration when dealing with large matrices, especially during the training process.

Therefore, we relied on the Colab environment to train the model to ensure more efficient execution and higher performance.

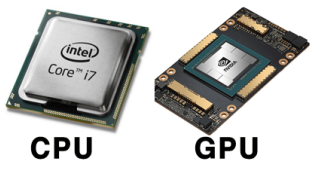


Figure 3.2: CPU and GPU.

Below is a detailed comparison between the core characteristics of the CPU and GPU:

Table 3.2: Comparison Between CPU and GPU Architectures

Feature	CPU (Central Processing Unit)	GPU (Graphics Processing Unit)
Function	General-purpose component that handles main processing functions	Specialized component optimized for parallel computing
Processing	Optimized for sequential instruction processing (low latency)	Designed for parallel instruction processing (high throughput)
Core Design	Fewer cores (typically 4-32) with complex architecture	Thousands of smaller, simpler cores (e.g., 2560 CUDA cores in RTX 3060)
Best Use Cases	Operating systems General computing tasks Single-threaded applications	Deep learning Computer vision Scientific simulations
Memory	Large cache memory (MBs)	High-bandwidth memory (GDDR6/HBM)
Power Efficiency	15-150W typical	100-400W typical

As shown in Table (3.2), the GPU is particularly effective in handling applications that require parallel processing, such as deep learning and computer vision, while the CPU remains more suitable for general-purpose tasks and single-threaded applications [33].

3.3 Software Environment

3.3.1 Programming Language

PyThon

Is a high-level interpreted programming language developed in 1990 by *Guido van Rossum*. It is known for its clear and easy-to-understand syntax, making it ideal for both beginners and experienced developers. Python is used in a wide range of domains such as software development, data science, scientific computing, web development, and artificial intelligence.

Figure 3.3: PyThon logo.

Python supports both object-oriented and procedural programming paradigms. It features a rich standard library that allows developers to perform complex tasks using only a few lines of code. It also supports modules and packages, which promote code reusability and better program organization.

One of Python's greatest strengths is its cross-platform compatibility : Python programs can run on various operating systems like Windows, Linux, and macOS without the need for recompilation. Additionally, it is freely available and supported by a large and active community, along with thousands of third-party libraries.[31]

3.3.2 Libraries and Frameworks

Pandas

The `pandas` library is a foundational Python library for data analysis, developed in 2008 by `Wes McKinney`. It aims to bridge the gap between Python and specialized statistical languages like `R` and `SAS`, by providing advanced tools such as automatic data alignment and hierarchical indexing. `pandas` was created to address the lack of rich, integrated tools for working with labeled (metadata) data in Python, which made statistical analysis difficult compared to other languages.

Figure 3.4: Pandas logo.

The name *pandas* is derived from **"panel data",** a term for multidimensional data sets used in statistics and econometrics. The library handles structured data, typically in the form of two-dimensional tables containing observations and field labels, making operations such as merging, grouping, and time series analysis simple and efficient.[22]

NumPy

Is an open-source Python library developed by *Travis Oliphant* in 2005, through the unification of the Numeric and Numarray libraries. It provides **multidimensional array objects** and efficient functions to operate on them, storing essential metadata such as *data type*, *shape*, and *strides*. NumPy is considered the **cornerstone** of the scientific Python ecosystem and is widely used in libraries such as SciPy, Pandas, and Matplotlib. Today, the **NumPy array** serves as the **standard format** for exchanging array data within the Python environment.[16]

Figure 3.5: NumPy logo.

Scikit Learn

Is an open-source library in Python, developed in **2007** by **David Cournapeau**, specialized in applying machine learning algorithms. It provides an easy-to-use interface tightly integrated with the Python language, supporting algorithms for classification, regression, clustering, as well as techniques for preprocessing and feature selection. It relies on **NumPy** and **SciPy** libraries and integrates **C++** code through libraries like **LibSVM** and **LibLinear** to enhance efficiency. It is known for focusing on imperative programming and ease of installation across different platforms.[1, 28]

Figure 3.6: Scikit Learn logo.

PyTorch

Is a Python-based deep learning library that builds on the Torch framework. It offers GPU acceleration, dynamic computational graphs, and an easy-to-use interface, making it ideal for researchers and developers. By using a ”define-by-run” approach, PyTorch constructs computation graphs dynamically during execution, which enhances debugging and allows flexible model customization.[10]

Figure 3.7: PyTorch logo.

Flask

is a lightweight web application framework for Python, built on the WSGI standard. It is designed for quick and easy development, while also being capable of scaling up to complex applications. Flask relies on libraries such as **Werkzeug** for request handling, **Jinja** for templating, and **Click** for command-line interaction, making it a flexible and powerful tool for building web applications.[26]

Figure 3.8: Flask logo.

3.4 Data preparation and Model Design

3.4.1 Methodology and Implemented Model

In this section, we present the methodology employed to implement a book recommendation system using Neural Collaborative Filtering (NCF). The objective of the model is to predict users' ratings of books based on their previous interactions and preferences.

Approach

The model relies on deep learning techniques to predict ratings. The core architecture is Neural Collaborative Filtering (NCF), which utilizes embeddings for both users and books. These embeddings are combined and passed through a multilayer perceptron (MLP) network to predict the interaction between the user and the book. The model is trained using the Huber Loss function, chosen for its robustness to outliers, providing a balance between the sensitivity of the mean squared error (MSE) and the resilience of the mean absolute error (MAE). For optimization, the AdamW algorithm is employed to adapt the learning rate dynamically during training.

Implemented Model

The implemented model consists of an embedding layer for users and another for books, followed by an MLP network. The neural network captures the nonlinear relationships between user preferences and the books they rate. The model output is a predicted rating ranging between 0 and 1, reflecting the probability that a user will positively rate a given book.

Model Advantages

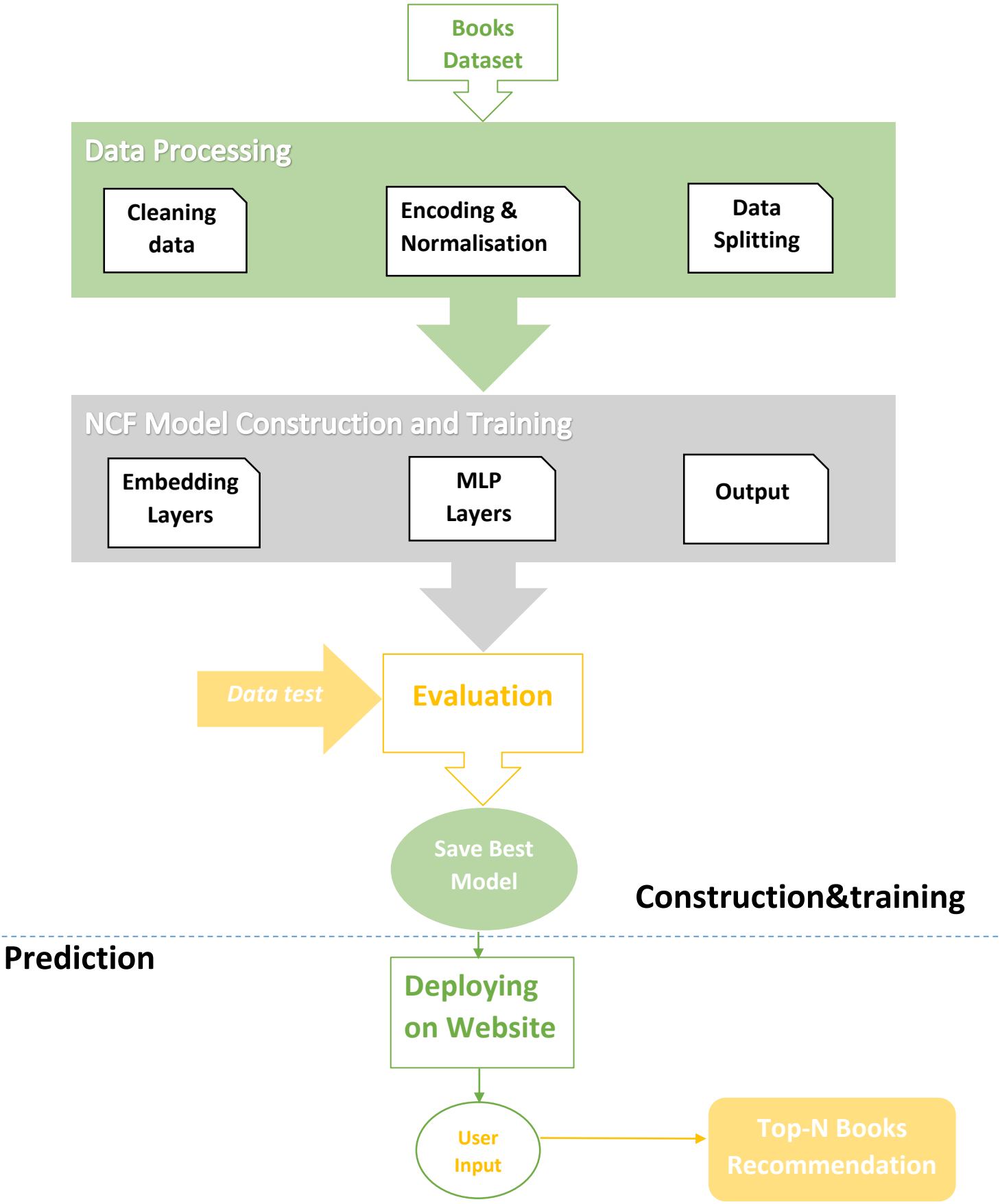
The primary advantages of this model lie in its ability to model complex, nonlinear relationships between users and items (books), leading to more personalized and accurate recommendations.

Expected Outcomes

The model is expected to provide accurate book recommendations by capturing hidden patterns in user-book interactions, thereby achieving higher precision in predicting user ratings.

The flowchart below illustrates the adopted methodology for developing a recommendation model based on NCF. This methodology encompasses the stages of data processing, model construction and training, evaluation and selection of the best-performing version, followed by deployment into a web interface.

Flowchart of the Model Development Methodology



Each stage in the flowchart is detailed as follows:

- **Data Processing:** The process begins with cleaning the books dataset to remove anomalous or missing values, followed by normalizing and preparing the data. Subsequently, the data is split into training, validation, and test sets.
- **Model Building and Training:** The model relies on embedding layers to represent users and books, followed by multilayer perceptron (MLP) layers that learn complex relationships between interactions, and finally an output layer that generates the final predictions.
- **Evaluation:** The model's performance is evaluated using the test set, and the best-performing version is selected based on evaluation metrics.
- **Deployment:** After saving the optimal model, it is integrated into a web interface, facilitating real-world application and allowing users to benefit from personalized book suggestions interactively.

3.4.2 Database

Kaggle Platform

The dataset employed in this project was sourced from **Kaggle**, an innovative digital platform specializing in data science and machine learning, currently owned by **Google**. Kaggle serves as a global hub for data scientists and machine learning engineers, providing access to a rich repository of datasets, collaborative opportunities, project sharing, and participation in international competitions designed to tackle both research-oriented and industrial challenges.

Established in 2010 by **Anthony Goldblum** and **Jeremy Howard**, Kaggle initially focused on hosting global machine learning competitions before expanding its offerings to include educational resources and cloud-based services that further scientific research and technological development.[18]

Figure 3.9: Kaggle logo.

Database Components

Following the acquisition of the dataset, its structure was analyzed and found to comprise three main CSV files covering **books**, **users**, and **ratings**. To efficiently handle the relatively large dataset, **Google Colab** was integrated with **Google Drive**. This integration was necessary given the limited temporary storage of Colab, whereas Google Drive offered a scalable and persistent solution.

Figure 3.10: Google Drive logo.

The compressed dataset files, initially provided in a `.zip` format, were extracted within the Colab environment and organized into a dedicated folder in Drive, ensuring efficient access and management throughout subsequent processing and modeling phases.

The following code snippet illustrates the process of mounting Google Drive to the Colab environment and decompressing the dataset:

```
#    ربط googl colab ب Google Drive
from google.colab import drive
drive.mount('/content/drive')
#  فك الضغط
!cp "/content/drive/MyDrive/Book Recommendation Dataset/BookData.zip" /content/

import zipfile

zip_path = "/content/drive/MyDrive/Book Recommendation Dataset/BookData.zip"# مسار الملف المضغوط
extract_path = "/content/drive/MyDrive/Book Recommendation Dataset/Extract_BookData"# مجلد الوجهة بعد فك الضغط

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print("تم فك الضغط بنجاح!")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
تم فك الضغط بنجاح!
```

Figure 3.11: Google Drive Integration and Data Extraction.

A detailed breakdown of each dataset component is presented below:

1. **Books File (Books.csv)**

   This file contains detailed information regarding the books and includes the following attributes:

- **ISBN**: A unique identifier for each book (International Standard Book Number).
- **Book-Title**: The title of the book.
- **Book-Author**: The name of the author.
- **Year-Of-Publication**: The publication year of the book.
- **Publisher**: The name of the publishing house.
- **Image-URL-S / M / L**: URLs linking to the small, medium, and large cover images of the book, respectively.

ISBN Book ISBN	Book-Title Book Title	Book-Author Book Author	Year-Of-Publicati... year of publication	Publisher publisher of the book	Image-URL-S small image of the book , amazon link	Image-URL-M medium size image of the book , amazon link	Image-URL-L large image size of the book , amazon link
271360 unique values	**242135** unique values	**102024** unique values	0 2050	Harlequin 3% Silhouette 2% Other (259605) 96%	**271044** unique values	**271044** unique values	**271042** unique values
0195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press	http://images.amazon.com/images/P/0195153448.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0195153448.01.MZZZZZZZ.jpg	http://images.amazon.com/images/P/0195153448.01.LZZZZZZZ.jpg
0002005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	http://images.amazon.com/images/P/0002005018.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0002005018.01.MZZZZZZZ.jpg	http://images.amazon.com/images/P/0002005018.01.LZZZZZZZ.jpg
0060973129	Decision in Normandy	Carlo D'Este	1991	HarperPerennial	http://images.amazon.com/images/P/0060973129.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0060973129.01.MZZZZZZZ.jpg	http://images.amazon.com/images/P/0060973129.01.LZZZZZZZ.jpg
0374157065	Flu: The Story of the Great Influenza Pandemic of 1918 and the Search for the Virus That Caused It	Gina Bari Kolata	1999	Farrar Straus Giroux	http://images.amazon.com/images/P/0374157065.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0374157065.01.MZZZZZZZ.jpg	http://images.amazon.com/images/P/0374157065.01.LZZZZZZZ.jpg
0393045218	The Mummies of Urumchi	E. J. W. Barber	1999	W. W. Norton & Company	http://images.amazon.com/images/P/0393045218.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0393045218.01.MZZZZZZZ.jpg	http://images.amazon.com/images/P/0393045218.01.LZZZZZZZ.jpg

Figure 3.12: Example of the Books dataset structure.

The information in this file is critical for distinguishing between different books and enhancing their descriptive profiles, thus contributing significantly to the recommendation system's performance.

2. **Users File (Users.csv)**

This file contains details about the users who interact with the books, structured across the following fields:

- **User-ID**: A unique identifier for each user.
- **Location**: The user's geographical information, generally formatted as "City, State, Country."

- **Age**: The user's age (noting that some entries may be missing or undefined).

The users' dataset enables the analysis of demographic distributions, facilitating more personalized and accurate recommendation models tailored to varying user behaviors and preferences.

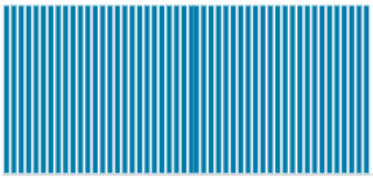
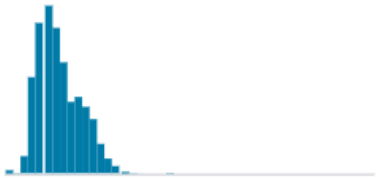
User-ID Unique user id	Location location of the user	# Age User Age
 1 279k	57339 unique values	 0 244
1	nyc, new york, usa	
2	stockton, california, usa	18.0
3	moscow, yukon territory, russia	
4	porto, v.n.gaia, portugal	17.0
5	farnborough, hants, united kingdom	

Figure 3.13: Example of the Users dataset structure.

3. **Ratings File (Ratings.csv)**

This file captures the interactions between users and books through recorded ratings, featuring:

- **User-ID**: Identifier of the user who rated the book.
- **ISBN**: Identifier of the rated book.
- **Book-Rating**: The user's rating score, ranging from 0 (implicit behavior, such as viewing) to 10 (explicit, definitive evaluation).

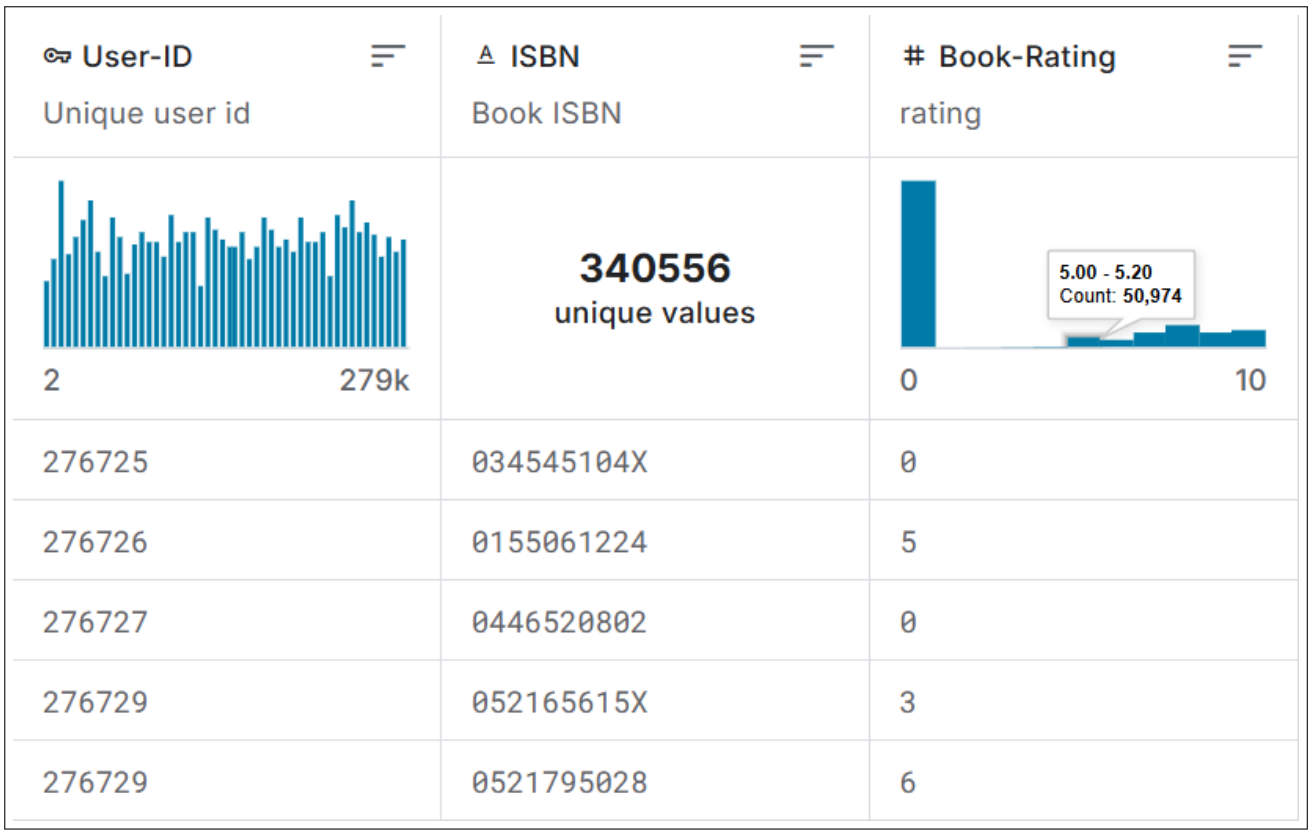


Figure 3.14: Example of the Ratings dataset structure.

The ratings file constitutes the core of user preference data, forming the foundation upon which the recommendation model is trained and from which future book suggestions are inferred.

The dataset contains approximately **271,360** book entries, **278,858** unique user profiles, and about **1,149,780** recorded user-book interactions. Given the considerable volume and heterogeneity of the data, meticulous preprocessing is required to address potential issues such as missing, inconsistent, or noisy values, thereby safeguarding the validity, reliability, and predictive accuracy of the resulting recommendation system. The subsequent section will elaborate on the preprocessing techniques adopted for this purpose.

3.4.3 Data Processing

Given the large size and diverse nature of the utilized datasets, it was essential to implement precise preprocessing steps to ensure data readiness for training the recommendation model. Raw datasets often contain issues such as missing values, incorrect entries, and inconsistencies, all of which could negatively impact the model's efficiency

and accuracy. Accordingly, a systematic data preparation procedure was designed, focusing on enhancing data quality and ensuring the information is structured in a reliable format suitable for building an effective recommendation system.

1. Preliminary Data Inspection

As a first step, the three datasets (Users, Books, and Ratings) were loaded using the `Pandas` library. An initial exploration was conducted employing functions such as `head()`, `info()`, and `isnull().sum()` to understand the overall structure of the datasets and to identify potential issues related to missing or inconsistent values.

The inspection revealed the following:

- **Users.csv**: Contains 278,858 users. Approximately 110,762 missing entries were identified in the *Age* column.
- **Books.csv**: Contains 271,360 book records. Some missing values were observed in the *Book-Author*, *Publisher*, and *Image-URL-L* columns.
- **Ratings.csv**: Comprises 1,149,780 rating entries without any missing values.

Table 3.3 synthesizes the primary findings derived from the preliminary examination of the datasets, highlighting critical aspects to be addressed during the cleaning phase.

Dataset	**Number of Records**	**Observations**
Users	278,858	Significant number of missing values in the *Age* column.
Books	271,360	Some missing values in *Book-Author*, *Publisher*, and *Image-URL-L* fields.
Ratings	1,149,780	No missing values detected.

Table 3.3: Summary of Datasets after Initial Inspection

Moreover, a **DtypeWarning** was observed in the Books dataset, indicating the presence of mixed data types in the *Year-Of-Publication* column. This anomaly was flagged for correction during the cleaning phase.

2. Data Cleaning

Based on the findings from the preliminary inspection, comprehensive data cleaning operations were applied. These operations included handling missing values, correcting data types, and addressing inconsistencies across the datasets to enhance their overall quality and integrity.

- **Users Dataset**

  In order to ensure the quality and suitability of the users dataset for the recommendation system, several cleaning operations were performed, as detailed below:

  – **Handling Implausible and Missing Age Values:**

  During the initial inspection, a considerable number of missing entries (approximately 110,762) were observed in the `Age` column. Additionally, implausible ages such as 0, 1, 183, and 239 were detected. Instead of deleting the corresponding rows, we replaced ages below 5 and above 100 with `NaN` values to preserve as much information as possible. The following code snippet summarizes this process:

```
import pandas as pd
import numpy as np

users = pd.read_csv("/content/drive/MyDrive/Book Recommendation
    Dataset/Extract_BookData/Users.csv")

# Replace implausible ages with NaN
users.loc[(users["Age"] < 5) | (users["Age"] > 100), "Age"] = np.nan
```

  – **Removing Unnecessary Attributes:**

  Given the high number of missing values in the `Age` column and its irrelevance to the recommendation model, we decided to drop it. Similarly, the `Location` column was also removed:

```
# Drop the Location and Age columns
users.drop(columns=["Location", "Age"], inplace=True)
```

  – **Validating the User-ID Field:**

  We ensured the integrity of the `User-ID` column by verifying the absence of missing values, confirming that all entries were of type `int64`, and checking

for uniqueness across users:

```
# Check for missing values, type, and uniqueness
print(users["User-ID"].isnull().sum())
print(users["User-ID"].dtype)
print(users["User-ID"].nunique(), len(users))
```

- **Books Dataset**

  To ensure the quality and consistency of the books dataset and to prepare it for effective integration into the Neural Collaborative Filtering model, several structured data cleaning steps were applied, as detailed below:

  – **Loading and Encoding the Data:**

  The dataset was loaded using the appropriate delimiter and character encoding to avoid textual corruption, especially due to the presence of special or non-ASCII characters.

```
import pandas as pd

books = pd.read_csv("/content/drive/MyDrive/Book Recommendation
    Dataset/Extract_BookData/Books.csv",
                    delimiter=';', encoding='latin-1')
```

  – **Removing Irrelevant Columns:**

  Columns related to image URLs (`Image-URL-S`, `Image-URL-M`, and `Image-URL-L`) were removed, as they are not useful for the recommendation task and increase data dimensionality.

```
books.drop(columns=["Image-URL-S", "Image-URL-M", "Image-URL-L"],
    inplace=True)
```

  – **Handling Missing and Incorrect Values:**

  Missing values were detected in essential columns such as `Book-Title`, `Book-Author`, and `Publisher`. Rows with any of these fields missing were discarded:

```
books.dropna(subset=["Book-Title", "Book-Author", "Publisher"],
    inplace=True)
```

  Initially, the dataset contained **271,360** records. After removing rows with

missing values, **266,719** entries remained, representing a loss of approximately **1.7%** of the data.

Some records included incorrectly encoded characters in text fields (e.g., `Der KÃ ?Ã Â¶nig in Gelb`, `Das BÃ ?Ã Â¶se kommt auf leisen Sohlen`, `QuerschÃ ?Ã Â¼s - Downsize This!`). These were manually corrected during inspection.

In cases where invalid publisher names appeared in the `Publisher` column (e.g., `1378`, `DK Publishing Inc'`), an attempt was made to retrieve correct metadata using the book's `ISBN`. If unsuccessful, the record was removed.

– **Cleaning the Publication Year:**
The `Year-Of-Publication` column contained anomalies such as non-numeric values (e.g., `Gallimard`) or unrealistic years (e.g., `1378`, `2050`). To address this, only numeric values between 1900 and 2022 were retained. The column was also converted from float to integer to ensure type consistency.

```
books = books[pd.to_numeric(books["Year-Of-Publication"],
    errors="coerce").notnull()]
books["Year-Of-Publication"] = books["Year-Of-Publication"].astype(int)
books = books[(books["Year-Of-Publication"] >= 1900) &
               (books["Year-Of-Publication"] <= 2022)]
```

– **Removing Duplicates:**
Duplicate records based on the combination of `Book-Title`, `Book-Author`, and `Year-Of-Publication` were identified and removed, keeping only the latest occurrence. This step reduced the dataset from **266,719** to **263,403** entries, eliminating **3,316** redundant books.

```
books.drop_duplicates(subset=["Book-Title", "Book-Author",
    "Year-Of-Publication"], keep="last", inplace=True)
```

– **Renaming Columns for Simplicity:**
To improve readability and simplify downstream processing, column names were renamed as follows:

```
books.rename(columns={
    "Book-Title": "Title",
    "Book-Author": "Author",
    "Year-Of-Publication": "Year"
```

```
}, inplace=True)
```

In total, approximately **5%** of the records were either modified or removed during this cleaning process. This thorough preprocessing ensured a more reliable and consistent books dataset, forming a strong basis for training and evaluating the recommendation model.

- **Ratings Dataset**

  To ensure that the ratings dataset is clean and ready for training a reliable recommendation system, we conducted several essential preprocessing steps, as outlined below:

  – **Checking for Missing Values:**

  The dataset was inspected for missing entries across the three key columns: `User-ID`, `ISBN`, and `Book-Rating`. No missing values were found:

```
print(ratings.isnull().sum())
```

  – **Keeping Zero Ratings:**

  Although a large portion of the ratings are zeros (representing implicit feedback or lack of opinion), we decided to retain them, as they constitute around 61

```
# We keep all ratings, including zero values
# ratings = ratings[ratings["Book-Rating"] > 0]  # This line was
    intentionally commented out
```

  – **Validating Referential Integrity:**

  To ensure data consistency, we filtered the ratings to include only those with `User-ID` and `ISBN` values that exist in the cleaned users and books datasets:

```
ratings = ratings[ratings["User-ID"].isin(users["User-ID"])]
ratings = ratings[ratings["ISBN"].isin(books["ISBN"])]
```

  – **Renaming Columns:**

  For consistency and simplicity, the column names were renamed:

```
df = df.rename(columns={
    'Book_Rating': 'rating',
    'User_ID': 'user_id',
    'ISBN': 'book_id'
})
```

These steps helped ensure the ratings dataset was consistent, inclusive, and well-aligned with the users and books data. Retaining zero ratings also supports cold-start scenarios by providing broader interaction patterns.

3. Encoding Generation

After completing the data cleaning phase, we found it essential to convert categorical variables such as user identifiers and book identifiers into numerical formats. Most machine learning and deep learning algorithms require numerical input and cannot directly process string-based data. To achieve this, we manually created consistent numerical encodings for both users and books. Specifically, we assigned a unique integer index to each distinct user ID and book ID using the `enumerate()` function in Python. The resulting mappings were applied using the `map()` function to transform the original identifiers into numerical indices starting from zero. This transformation preserved the identity of each user and book while ensuring compatibility with the recommendation model.

```
# Retain only positive ratings
df = df[df['rating'] > 0]

# Normalize ratings to the range [0, 1]
df['rating'] = df['rating'] / 10.0

# Create fixed numerical encodings for user and book IDs
user_id_mapping = {id_: idx for idx, id_ in enumerate(df['user_id'].unique())}
book_id_mapping = {id_: idx for idx, id_ in enumerate(df['book_id'].unique())}

# Apply the encodings to the original data
df['user_idx'] = df['user_id'].map(user_id_mapping)
df['book_idx'] = df['book_id'].map(book_id_mapping)
```

4. Data Splitting

To properly train and evaluate our model, we divided the dataset into three distinct subsets: training, validation, and testing. This separation allows us to evaluate the model's ability to generalize to unseen data by assessing its performance on samples that were not used during training. We utilized the `train_test_split` function from the `sklearn` library for this partitioning. Specifically, we allocated 70% of the data for training and split the remaining 30% evenly between the validation and test sets. This ensures that the model is tested on data that it has never seen during training, providing a more accurate estimate of its performance on new, unseen data.

The details of the data splitting and the resulting dataset distribution are shown in Table(3.4), and the corresponding Python code is provided below:

```
from sklearn.model_selection import train_test_split

# Prepare the final dataset
full_data = df[['user_idx', 'book_idx', 'rating']].values

# Split the data: 70% training, 15% validation, 15% testing
train_data, temp_data = train_test_split(full_data, test_size=0.3,
    random_state=42)
val_data, test_data = train_test_split(temp_data, test_size=0.5, random_state=42)
```

Data Category	Number of Samples	Percentage
Training Data	245073	70%
Validation Data	52516	15%
Test Data	52516	15%
Total	350105	100%
Total Users	63417	-
Total Books	144527	-

Table 3.4: Data Summary Overview

5. Loading and Preprocessing Data

To ensure that the data processing is efficient and repeatable, we implemented a function called `load_and_preprocess_data()`. This function first checks whether a preprocessed version of the dataset already exists, stored in a pickle file. If the

pickle file exists, it loads the data from there, preventing redundant data processing and speeding up the workflow.

If the pickle file does not exist, the function reads the raw data from a CSV file using `pd.read_csv()`. The raw data is then filtered based on the activity of users and books. Specifically, we kept only the top 90

The ratings in the dataset were normalized to a range between 0 and 1 by dividing the ratings by 10, making the scale compatible with the model's expectations.

After processing the ratings, the function then generates unique integer mappings for both users and books, similarly to the encoding generation process described earlier. These mappings are used to convert the user and book IDs into indices that can be directly fed into the model.

Finally, the function calculates the popularity of each book based on its frequency in the dataset, normalizing these values so that they can be used in the recommendation model. The processed data is saved in a pickle file for future use, ensuring that the preprocessing step is only executed once.

The full Python implementation for the data loading and preprocessing steps is available in Appendix**(.1)**

6. Data Loaders

To efficiently feed data into the deep learning model, we created data loaders using PyTorch. These loaders allow us to batch the data and load it in an optimized manner during training and evaluation. We implemented a custom function that converts each data subset into a `TensorDataset`, which is then passed to a `DataLoader`. We enabled shuffling only for the training set to improve model generalization and reduce bias from the data order.

```
import torch
from torch.utils.data import TensorDataset, DataLoader

# Function to create a DataLoader from given data
def create_dataloader(data, batch_size=2048, shuffle=False):
    dataset = TensorDataset(
        torch.LongTensor(data[:, 0]),  # user_idx
        torch.LongTensor(data[:, 1]),  # book_idx
```

```
        torch.FloatTensor(data[:, 2])  # rating
    )
    return DataLoader(dataset, batch_size=batch_size, shuffle=shuffle,
        num_workers=2)

# Create data loaders
train_loader = create_dataloader(train_data, shuffle=True)
val_loader = create_dataloader(val_data)
test_loader = create_dataloader(test_data)
```

3.4.4 Model Building.

In this step, we built the core of our recommendation system. We began by importing all the necessary libraries that support data manipulation, model definition, training, and evaluation. These libraries cover various functionalities such as matrix operations, plotting, data loading, deep learning, and working with pretrained models when needed.

The following listing presents the necessary packages:

```
import os
import torch
import pickle
import numpy as np
import pandas as pd
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
from tqdm import tqdm
from collections import defaultdict
from sklearn.model_selection import train_test_split
from torch.utils.data import DataLoader, TensorDataset
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

Code Listing 3.1: Importing necessary Python packages

We also linked our Google Drive to Colab to allow reading and writing data files and models. The following code enables access to Google Drive:

```
from google.colab import drive
drive.mount('/content/drive')
```

Code Listing 3.2: Mounting Google Drive in Colab

Once mounted, we used paths inside Google Drive to read the dataset and store trained models and results.

- ## NCF Model

  After presenting the fundamental concepts of deep learning and its role in recommendation systems, we introduce the main model developed in this work. It is built upon the principles of Neural Collaborative Filtering (NCF) and employs a Multi-Layer Perceptron (MLP) architecture to capture user-book interaction patterns.

  The model is designed to flexibly learn complex relationships between users and books through dense vector representations (embeddings), which are processed by a deep neural network. This architecture enables the extraction of latent patterns from historical interaction data, leading to more accurate and personalized book recommendations.

  The following section details each component of the model architecture.

  1. **Embedding Layers**

     The model begins by transforming each user and book ID into a dense vector representation using embedding layers. These embeddings allow the model to learn latent features that characterize user preferences and book attributes. Each embedding has a dimensionality of 64.

     ```
     self.user_embedding = nn.Embedding(num_users, embedding_dim)
     self.book_embedding = nn.Embedding(num_books, embedding_dim)
     ```

     To improve convergence and generalization, the weights are initialized using Xavier Uniform:

     ```
     nn.init.xavier_uniform_(self.user_embedding.weight)
     nn.init.xavier_uniform_(self.book_embedding.weight)
     ```

  2. **Concatenation of Embeddings**

     The outputs from the user and book embedding layers are concatenated into a single vector of size 128:

     ```
     user_emb = self.user_embedding(user_ids)
     book_emb = self.book_embedding(book_ids)
     x = torch.cat([user_emb, book_emb], dim=1)
     ```

This vector becomes the input to the multi-layer perceptron.

3. **Multi-Layer Perceptron (MLP)**

   The MLP module contains three fully connected layers with progressively reduced sizes: 256, 128, and 64. Each layer is followed by:

   – **Batch Normalization** to stabilize and accelerate training.

   – **LeakyReLU Activation** to avoid dying ReLU problems.

   – **Dropout** to prevent overfitting.

```
self.mlp = nn.Sequential(
    nn.Linear(embedding_dim*2, 256),
    nn.BatchNorm1d(256),
    nn.LeakyReLU(0.2),
    nn.Dropout(0.4),

    nn.Linear(256, 128),
    nn.BatchNorm1d(128),
    nn.LeakyReLU(0.2),
    nn.Dropout(0.3),

    nn.Linear(128, 64),
    nn.BatchNorm1d(64),
    nn.LeakyReLU(0.1)
)
```

   Each linear layer's weights are initialized using Kaiming Normal, which is suitable for layers followed by LeakyReLU:

```
for layer in self.mlp:
    if isinstance(layer, nn.Linear):
        nn.init.kaiming_normal_(layer.weight, mode='fan_out',
            nonlinearity='leaky_relu')
        nn.init.zeros_(layer.bias)
```

4. **Output Layer**

   The final output of the MLP is passed through a linear transformation followed by a Sigmoid activation function. The Sigmoid maps the result to a value between 0 and 1, representing the estimated probability that a user will positively interact with a given book.

```
self.output = nn.Sequential(
    nn.Linear(64, 1),
    nn.Sigmoid()
)
```

The result represents the estimated probability that a user will like or interact positively with the given book.

5. **Forward Pass**

   All steps described above are combined in the `forward()` method, which defines the flow of data through the network:

```
def forward(self, user_ids, book_ids):
    user_emb = self.user_embedding(user_ids)
    book_emb = self.book_embedding(book_ids)
    x = torch.cat([user_emb, book_emb], dim=1)
    x = self.mlp(x)
    return self.output(x).squeeze()
```

   A full version of the NCF model implementation can be found in the appendix(.2).

An overview of the complete model architecture, including the number of parameters and output shapes for each layer, is provided in Table 3.5, illustrated below as an image:

شكل الإخراج	عدد المعاملات	النوع	الطبقة
user_embedding	Embedding	4,058,688	(batch_size, 64)
book_embedding	Embedding	9,249,728	(batch_size, 64)
mlp.0	Linear	33,024	(batch_size, 256)
mlp.1	BatchNorm1d	512	N/A
mlp.4	Linear	32,896	(batch_size, 128)
mlp.5	BatchNorm1d	256	N/A
mlp.8	Linear	8,256	(batch_size, 64)
mlp.9	BatchNorm1d	128	N/A
output.0	Linear	65	(batch_size, 1)

Table 3.5: Overview of the model architecture and parameter count.

For further details, including per-layer parameter calculations and exact data shapes, please refer to Appendix(27).

- **Model Architecture Design**

The proposed architecture (Figure 3.15) integrates user and book embedding layers with a Multi-Layer Perceptron (MLP) consisting of three fully connected layers with 256, 128, and 64 units, respectively. This configuration was chosen after experimenting with various architectural depths, as it offered an optimal balance between predictive accuracy and training robustness.

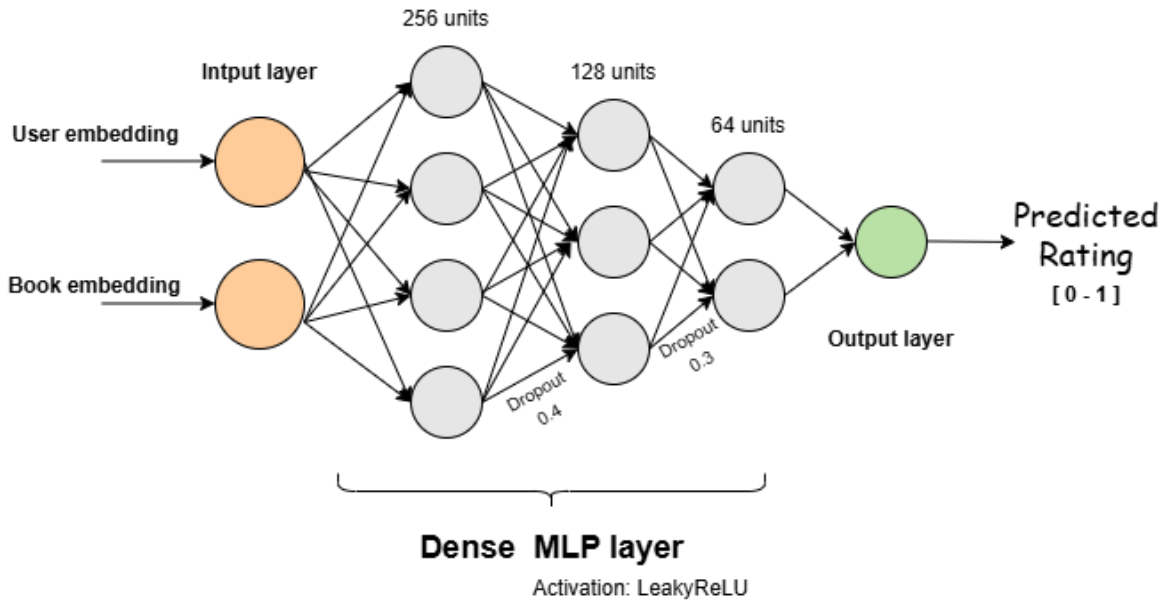


Figure 3.15: Architecture of the proposed NCF model

To enhance the model's ability to capture complex user-book interaction patterns, LeakyReLU was employed as the activation function instead of the standard ReLU. This choice helps mitigate the "dying neuron" problem by allowing a small gradient flow for negative inputs. Dropout layers with rates of 0.4 and 0.3 were also introduced to prevent overfitting by randomly deactivating neurons during training.

The embedding layers were initialized using Xavier initialization to maintain consistent variance between layers, while Kaiming initialization was applied to the MLP layers to better suit the LeakyReLU activations and accelerate convergence.

The model was trained using the Huber loss function, which is more robust to outliers than mean squared error, and optimized using the AdamW algorithm for efficient and stable training.

Compared to simpler models such as Generalized Matrix Factorization (GMF), this architecture achieved superior performance, particularly on top-K evaluation metrics like Hit Ratio (HR) and Normalized Discounted Cumulative Gain (NDCG), demonstrating its ability to learn rich, non-linear user-book interactions in sparse data scenarios.

3.4.5 Training and Evaluation

1.Model Training

Once the architecture of the recommendation model was finalized, we proceeded to the training phase. This stage aims to enhance the model's ability to predict user preferences by learning from historical user-book interactions. The following subsections outline the training setup, optimization techniques, and improvements applied to ensure robust generalization.

And Here's the Figure 3.16 showing the process of training the NCF model:

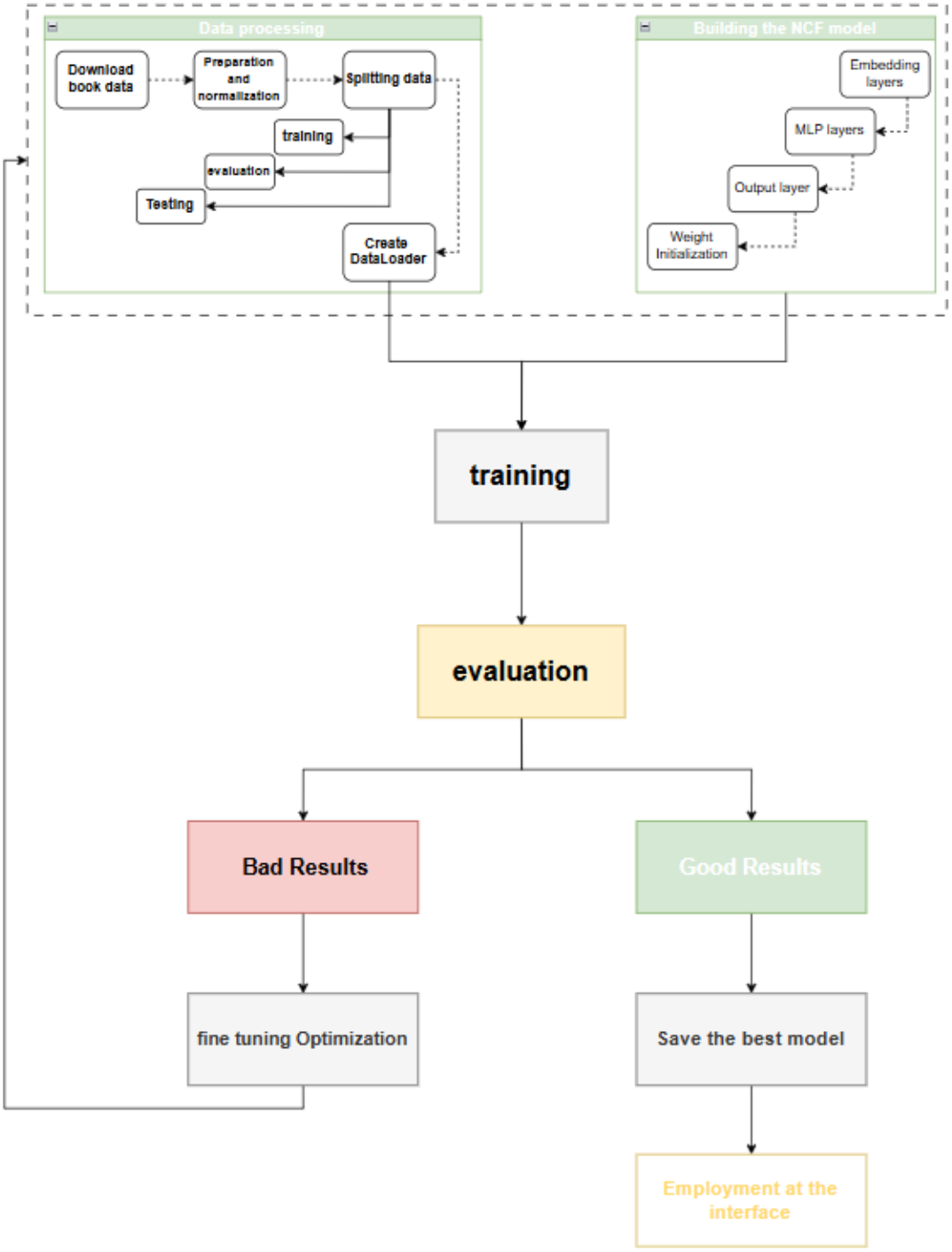


Figure 3.16: The process of the NCF model design

- **Device Configuration**

  To accelerate computations, the model is automatically trained on a GPU if available; otherwise, it defaults to CPU:

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = BookRecommendationModel(num_users, num_books).to(device)
```

- **Loss Function and Optimizer**

  We used the **Huber Loss** function, which combines the robustness of MAE and the smoothness of MSE, with a delta value of 0.3:

```
criterion = nn.HuberLoss(delta=0.3)
```

  For optimization, we adopted **AdamW**, a weight-decay-enhanced version of Adam that improves generalization:

```
optimizer = optim.AdamW(model.parameters(), lr=0.001, weight_decay=1e-3)
```

  To dynamically adjust the learning rate when validation performance stagnates, we used a ReduceLROnPlateau scheduler:

```
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=2,
    factor=0.5)
```

- **Partial Embedding Freezing**

  Because the user and book embeddings form the backbone of the model, we initially froze their parameters during the first two epochs. This prevents the randomly initialized vectors from shifting too quickly, which could destabilize early learning:

```
if epoch < 2:
    model.user_embedding.requires_grad_(False)
    model.book_embedding.requires_grad_(False)
```

  After this initial phase, the embeddings are unfrozen to allow full learning.

- **Regularization and Gradient Clipping**

  To mitigate overfitting and improve generalization, we added manual **L2 regularization** across model parameters:

```
l2_reg = sum(p.pow(2.0).sum() for p in model.parameters())
loss += 0.001 * l2_reg
```

Additionally, we applied **gradient clipping** to prevent gradient explosion:

```
torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
```

- **Validation and Early Stopping**

  The model was evaluated on a validation set after every epoch. If the validation loss improved, the model was saved:

```
if avg_val < best_val_loss:
    best_val_loss = avg_val
    torch.save(model.state_dict(), save_model_path)
```

  To avoid overtraining, we implemented **early stopping** when no improvement was observed for 8 consecutive epochs:

```
if no_improve >= patience:
    print(" Early stopping triggered")
    break
```

- **Training Progress Visualization**

  We tracked training dynamics by plotting loss curves for both training and validation sets:

```
plt.plot(history['train'], label='Train Loss')
plt.plot(history['val'], label='Validation Loss')
```

  The full implementation of the training procedure is available in Appendix(.5).

- **Observations and Outcomes**

  The applied training enhancements—including embedding freezing, L2 regularization, gradient clipping, and dynamic learning rate scheduling—contributed to improved model stability and smoother convergence. Validation loss steadily improved while overfitting was reduced.

  The training process showed consistent progress: the training loss decreased from 1.1976 in epoch 1 to 0.0166 by epoch 18. Similarly, the validation loss dropped from 0.0215 to 0.0138, indicating effective learning and generalization.

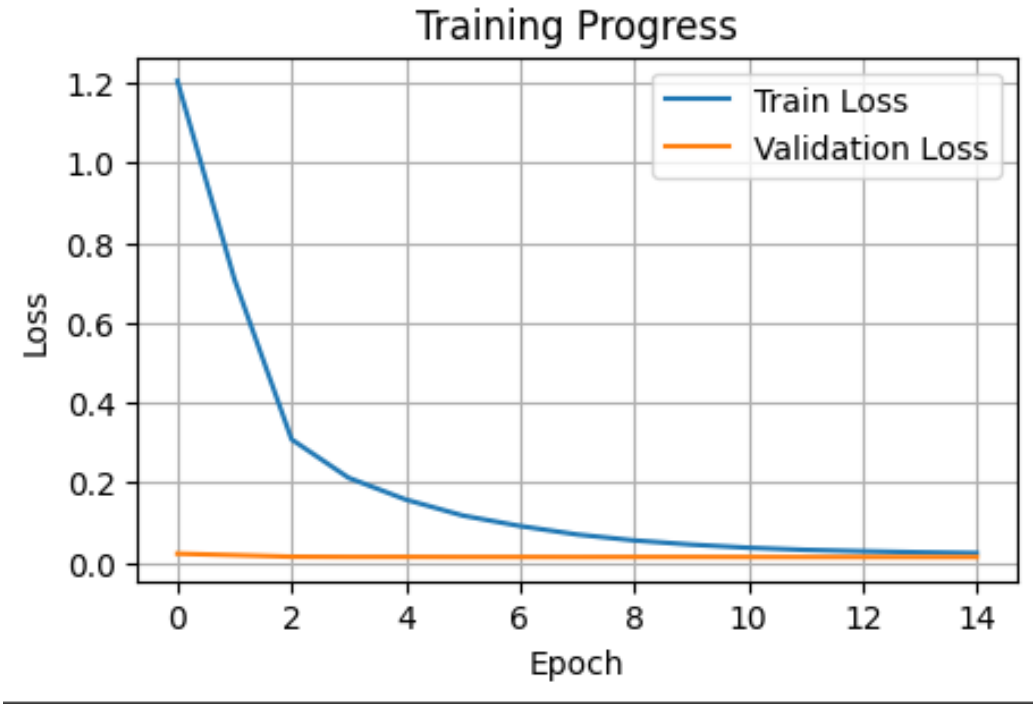


Figure 3.17: Loss curves after optimization

Early stopping was activated once validation loss plateaued, ensuring training stopped at optimal performance. The best model was saved and results remained stable in the final epochs.

The complete training log is provided in Appendix (5).

2. Performance Evaluation

After completing the training phase of our recommendation model, we evaluated its performance using both regression and ranking metrics to gain a comprehensive understanding of how well the model captured user preferences.

- **Mean Absolute Error (MAE)**: Measures the average of the absolute differences between actual ratings and predicted ones. It was computed using the `mean_absolute_error` function from the `sklearn.metrics` module.
- **Root Mean Squared Error (RMSE)**: Gives more weight to larger errors by squaring the differences before averaging and then taking the square root. It was calculated using `np.sqrt(mean_squared_error(...))`.
- **Hit Rate at 10 (HR@10)**: Measures the proportion of times that the true item appeared in the top-10 predicted recommendations. This metric is particularly useful for evaluating top-k recommendation quality.

- **Normalized Discounted Cumulative Gain at 10 (NDCG@10)**: Evaluates not only whether the relevant item is in the top-k list but also considers its position in the ranking. Higher-ranked correct items contribute more to the score.

The evaluation process was implemented in the `evaluate_model` function, which performs the following steps (the full implementation is available in Appendix 4):

1. Switching the model to evaluation mode using `model.eval()`, which disables operations like dropout and prevents gradient computation:

```
model.eval()
```

2. Iterating over the validation or test data to collect actual and predicted ratings:

```
with torch.no_grad():
    for users, books, ratings in loader:
        outputs = model(users, books)
        preds.extend(outputs.cpu().numpy())
        true.extend(ratings.cpu().numpy())
```

3. For each batch, HR@10 and NDCG@10 were computed using a helper function `calculate_hr_ndcg`, which ranks predictions and compares them to true labels.

4. Computing MAE and RMSE:

```
mae = mean_absolute_error(true, preds)
rmse = np.sqrt(mean_squared_error(true, preds))
```

These metrics provided a quantitative understanding of how closely the model's predictions matched real user preferences.

In addition, we visualized the distribution of both true and predicted ratings, which helped to identify any systematic biases or inconsistencies in the model's behavior, as shown in Figure 3.18.

Final evaluation results:

- MAE: **0.1297**
- RMSE: **0.1659**
- HR@10: **0.9564**
- NDCG@10: **1.0000**

The high HR and perfect NDCG values indicate that the model was not only able to include the correct items in the top-10 recommendations but also placed them at top positions, demonstrating excellent ranking performance.Appendix(4).

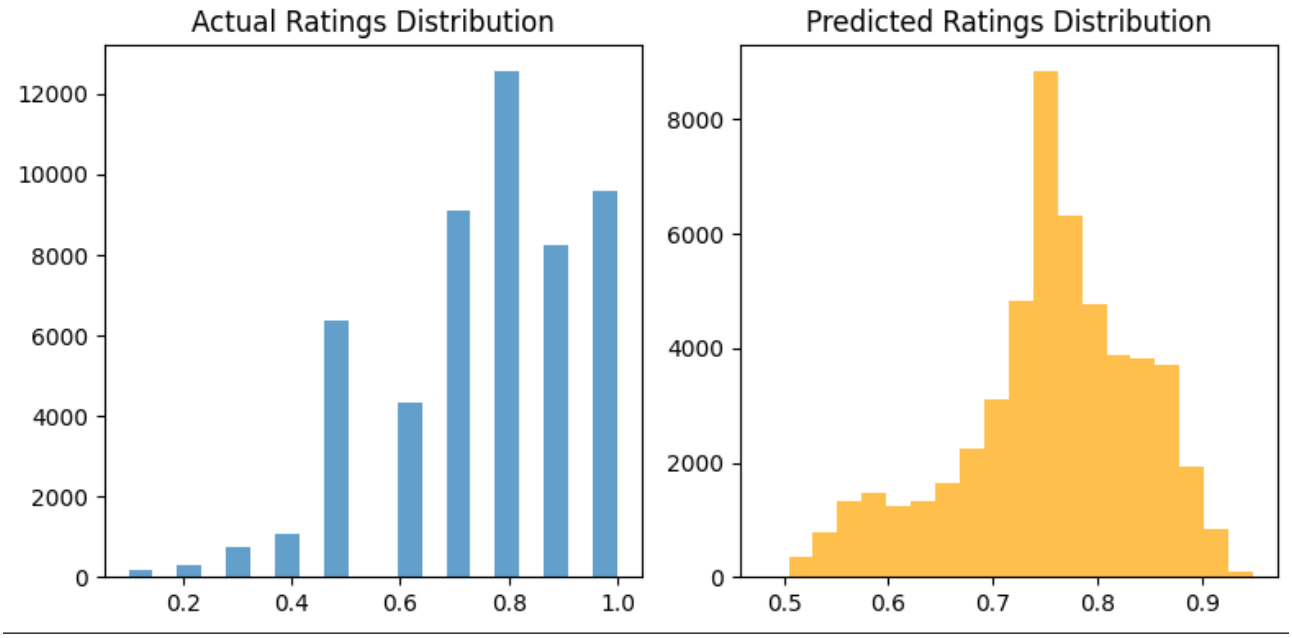


Figure 3.18: Distribution of actual vs. predicted ratings

3.4.6 Enhanced Recommendation Strategy

The final stage of our system introduces an **adaptive book recommendation strategy** that intelligently handles users with varying levels of interaction. The logic is centralized in the function `recommend_books(user_id, top_n, diversity)`, which dynamically selects the appropriate approach based on user history.

1. **New Users – Popularity-Based Recommendations**

   When a user ID is not present in the system (*i.e.*, cold-start user), the system cannot generate predictions due to the lack of training data. In such cases, the recommender falls back on book popularity:

```
if user_id not in user_id_mapping:
    print("New user! Showing popular books...")
    return get_popular_recommendations(top_n)
```

   The popularity is computed based on the frequency and average rating of each book:

```
popular_books = df.groupby('book_id').agg({
    'book_title': 'first',
    'book_author': 'first',
    'rating': 'mean',
    'user_id': 'count'
}).sort_values('user_id', ascending=False).head(top_n)
```

Justification: This ensures user engagement by offering trusted, widely appreciated books even in the absence of personalized history.

2. **Low-Interaction Users – Hybrid Recommendations**

Users with fewer than three ratings are considered to have insufficient interaction data. To address this, the system blends personalized recommendations (using the neural model) with popularity-based suggestions:

```
if len(user_ratings) < 3:
    print("Low activity user! Mixing recommendations...")
    personalized = get_personalized_recommendations(user_idx, top_n//2)
    popular = get_popular_recommendations(top_n - len(personalized))
    return personalized + popular
```

Justification: This hybrid approach improves diversity and accuracy by leveraging partial personalization while maintaining recommendation reliability.

3. **Active Users – Personalized Neural Recommendations**

For users with sufficient interaction history, the system uses the trained Neural Collaborative Filtering (NCF) model to suggest unseen books. The recommendation function incorporates a diversity factor that blends neural predictions with popularity:

```
return get_personalized_recommendations(user_idx, top_n, diversity)
```

Inside the recommendation logic:

- The system filters unseen books.
- It computes predicted scores via the NCF model.
- It combines predictions with popularity using a weighted formula:

```
combined_score = (1 - diversity) * score + diversity * popularity
```

Final book selection is based on top-ranked combined scores:

```
top_indices = np.argsort(combined_scores)[-top_n:][::-1]
```

Justification: This balances **accuracy** (via personalized prediction) and **serendipity** (via popularity smoothing), enhancing both relevance and novelty.

4. **Model Evaluation and Simulation Testing**

The system is evaluated using held-out test data and simulates recommendations for both known and new users. The model is first trained and the best-performing weights are saved:

```
train_model(model, train_loader, val_loader)
model.load_state_dict(torch.load(save_model_path))
evaluate_model(model, test_loader)
```

Sample recommendations are then generated:

```
random_user = np.random.choice(df['user_id'].unique())
recs = recommend_books(random_user, top_n=5)

# New user simulation
new_user_recs = recommend_books(999999, top_n=3)
```

```
🎯 اختبار نظام التوصية:

📚 107809 توصيات للمستخدم:
🔍 مستخدم قليل التفاعل! جاري دمج التوصيات...
1. Harry Potter and the Goblet of Fire (Book 4)... - 7.5 :التقييم المتوقع
2. Harry Potter and the Prisoner of Azkaban (Book 3)... - 7.5 :التقييم المتوقع
3. The Lovely Bones: A Novel... - 0.8 :التقييم المتوقع
4. Wild Animus... - 0.4 :التقييم المتوقع
5. The Red Tent (Bestselling Backlist)... - 0.8 :التقييم المتوقع

👤 مستخدم جديد (غير موجود في النظام):
👋 مستخدم جديد! جاري عرض الكتب الشعبية...
1. The Lovely Bones: A Novel... (0.01 :شعبية)
2. Wild Animus... (0.01 :شعبية)
3. The Red Tent (Bestselling Backlist)... (0.01 :شعبية)
```

Figure 3.19: Recommendations for New and Low-Interaction Users

Justification for model saving/loading: Saving the best model ensures consistent prediction quality and eliminates the need for retraining during deployment:

```
model.load_state_dict(torch.load(save_model_path))
```

Summary: This adaptive recommendation module tailors strategies based on user engagement level, utilizing popularity, hybrid blending, or personalized NCF predictions. The system is designed for robustness, scalability, and improved user experience, addressing common challenges such as data sparsity and cold-start conditions in real-world recommender systems.

For the full code, please see the appendix (5).

3.4.7 Interface

Connection Method

The web interface was connected to the backend using the **Flask** framework in **Python**. This enabled seamless interaction between the user-facing component and the underlying **recommendation model**. Flask handles **user input** (such as the user ID), processes requests, and returns **personalized book recommendations** generated by the trained model. This communication adheres to a **request-response paradigm**, where the server receives user data, feeds it into the model, and responds with relevant outputs.

In addition, the system addresses the well-known **cold-start problem**, which occurs when **new users** lack sufficient interaction history. In such cases, the backend automatically generates a list of the **most popular books**. This strategy promotes initial engagement, enabling the system to progressively infer the user's preferences over time.

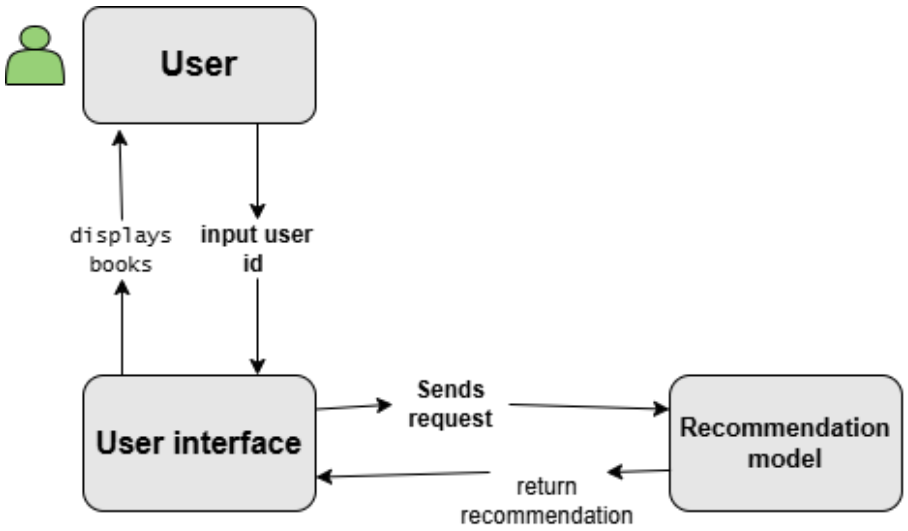


Figure 3.20: Data flow between the user, the interface, and the recommendation model

Interface Description

The **user interface** was developed using standard web technologies such as **HTML**, **CSS**, and **JavaScript**. It provides a **clean and intuitive** environment for users to interact with the system. Users can enter their **user ID** to receive book recommendations, which are presented as **visually structured cards**. Each card displays key information such as the **book title** and **cover image**. When available, the system also includes a brief explanation highlighting the **reason for the recommendation**.

This design facilitates a **progressive personalization experience**. Initially, the interface displays **generic suggestions**—especially for new users—and gradually adapts the content toward **tailored recommendations** as the user's interaction history expands.

Interface Screenshots

To illustrate the web interface of the recommendation system, the following figures showcase different interaction scenarios:

- **Website Interface Showcase:**

Figure 3.21: Initial interface.

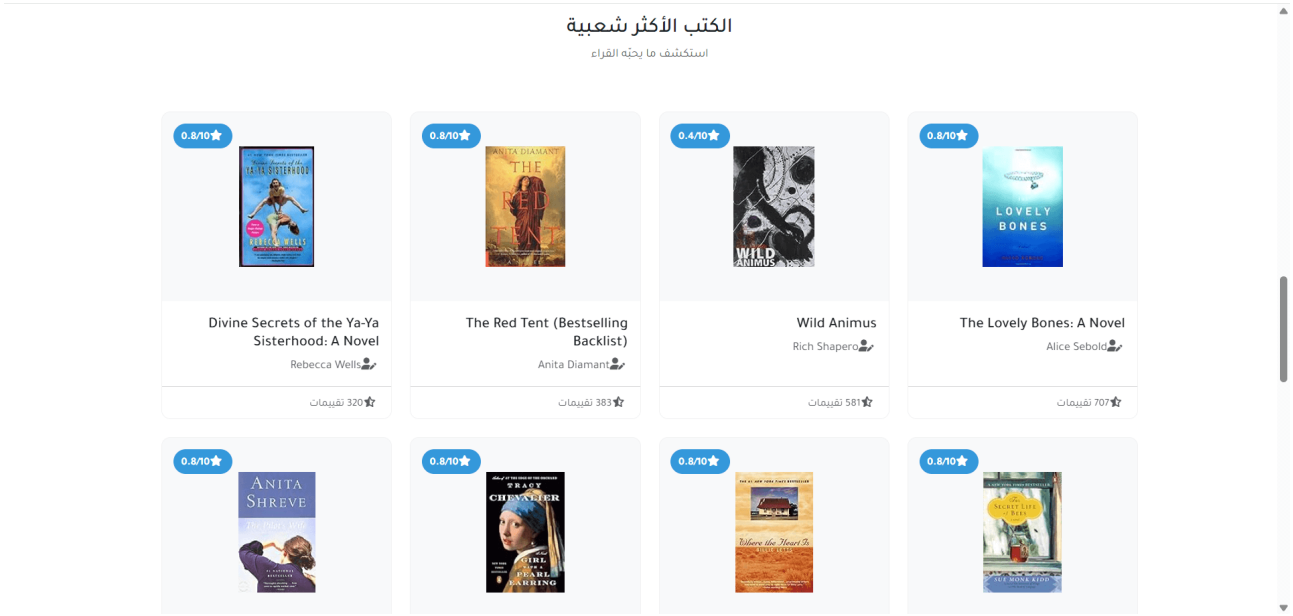


Figure 3.22: Most Popular Books.

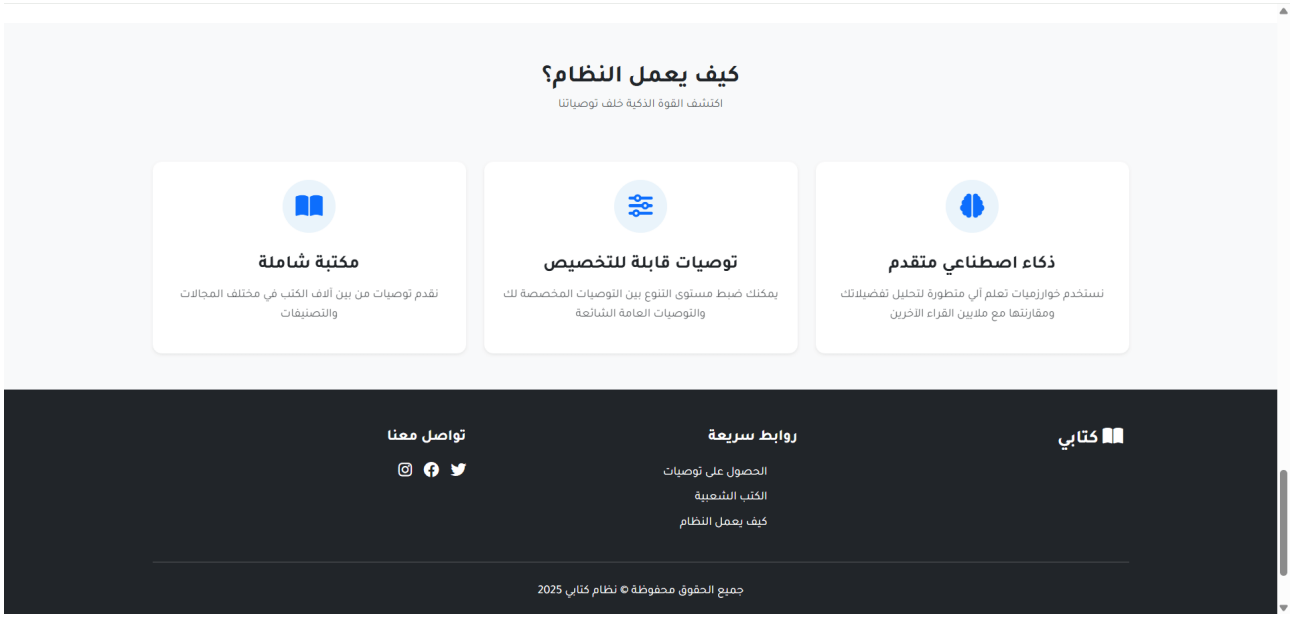


Figure 3.23: How the System Works and footer.

- **Known user:**

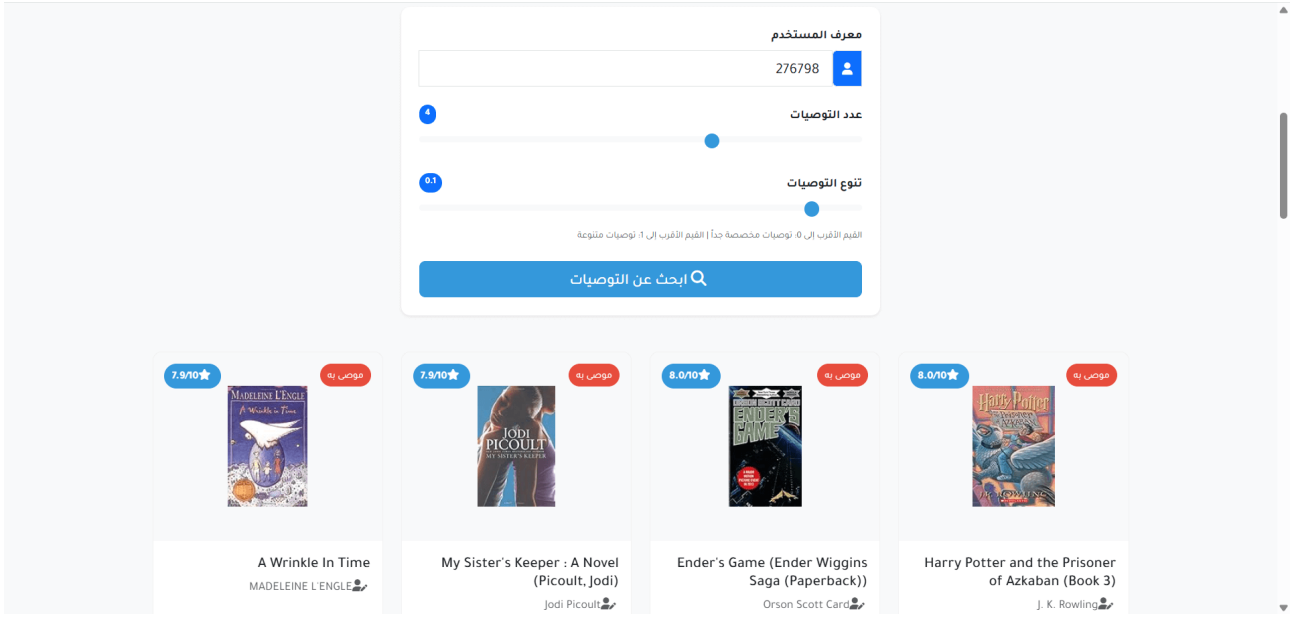


Figure 3.24: Book recommendations for a known user.

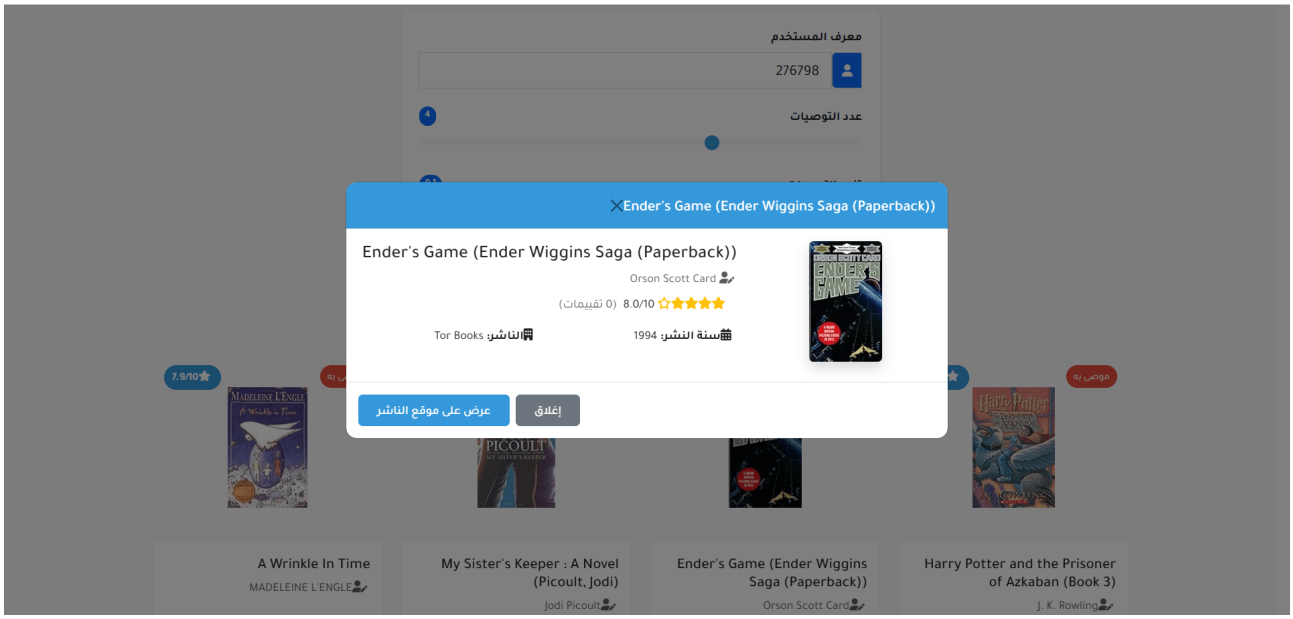


Figure 3.25: Expanded view of a recommended book.

- **New user:**

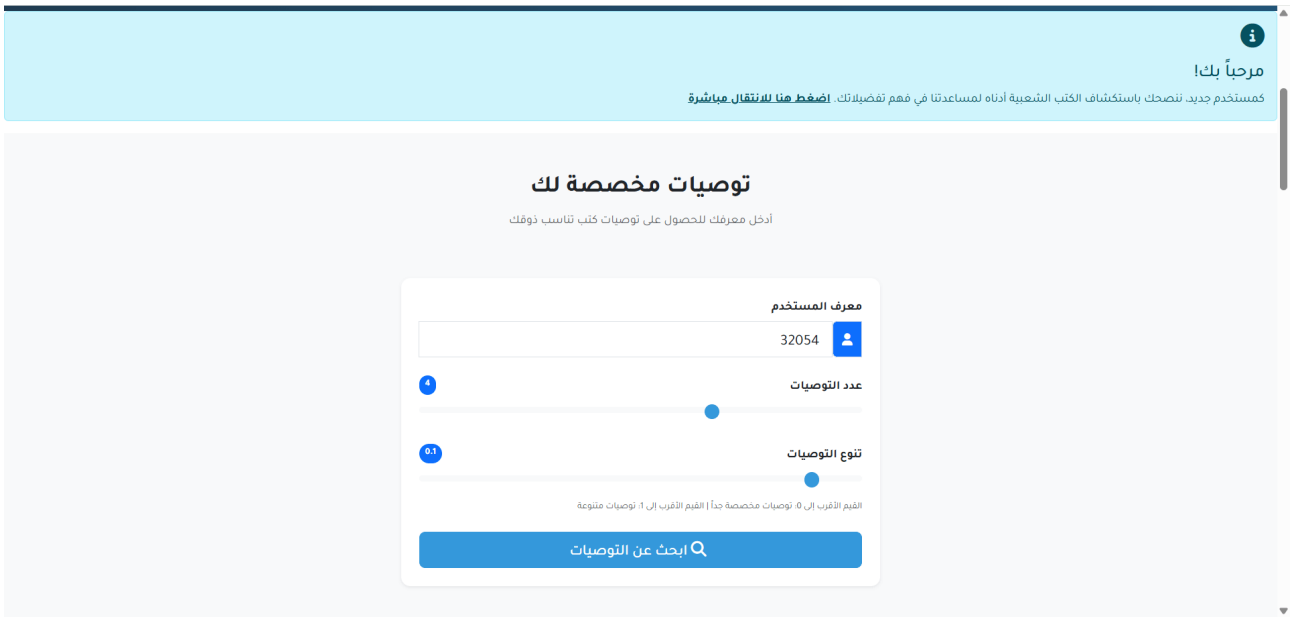


Figure 3.26: Popular books recommended to a new user (cold-start scenario).

3.4.8 Challenges and Limitations

During the development of this project, we faced several challenges that impacted the implementation and evaluation of the proposed recommendation system. These challenges can be summarized as follows:

- **Data Sparsity:** The dataset suffered from a lack of interactions between users and items, making it difficult for the model to learn accurate representations, especially for new or inactive users/items (cold start problem).
- **Computational Constraints:** Training deep learning models like NCF requires significant computational resources. Although we used Google Colab as a cloud environment, limited GPU availability and execution time affected the number of experiments that could be conducted.
- **Hyperparameter Tuning:** Selecting the optimal values for hyperparameters such as learning rate, embedding size, and dropout rate was a real challenge. It required several iterations to reduce overfitting and improve the validation performance.
- **Evaluation Limitations:** In the absence of explicit ratings or precise temporal data, some evaluation metrics like HR and NDCG may not fully reflect user satisfaction or system effectiveness in real-world scenarios.
- **Incorporating External Features:** We attempted to integrate additional information, such as text embeddings of book titles extracted using BERT. However, aligning these features with the current dataset and neural architecture added complexity.
- **Cross-Validation Application:** We tried to implement cross-validation on the training data to improve the general performance estimation of the model. However, this significantly increased the training time and required careful data splitting to avoid data leakage.
- **Transfer Learning:** We also explored improving model performance using transfer learning techniques by incorporating pre-trained embeddings. However, merging these features into the model architecture and training pipeline required additional experimentation and adjustments.

- **Integrating the Model with the Frontend:** We encountered technical difficulties when trying to integrate the model with the frontend and display the recommendations interactively. Although we managed to create an initial interface, accurately presenting recommendations to all users, especially new ones, still requires further refinement.

- **Handling New Users:** The process of adapting the model to handle new users and save their interactions for future learning is still under development. However, due to time constraints, this aspect could not be fully implemented.

Despite these challenges, the project has provided valuable insights into the practical aspects of building recommendation systems, equipping us with hands-on experience in overcoming technical obstacles. It also lays the foundation for further research and development in this field.

3.5 Conclusion

In this chapter, we conducted an in-depth practical experience in building and deploying a **book recommendation system** based on the **Neural Collaborative Filtering (NCF)** technique. We started by setting up the **programming and cloud environment**, then processed **real-world user rating data** before designing and training a model based on a **Multi-Layer Perceptron (MLP)** using the **PyTorch** library.

Our work went beyond the theoretical or programming aspects, as we also evaluated the model's performance using precise metrics such as **MAE**, **RMSE**, and **HR**, which provided us with a realistic insight into the **quality of the recommendations** offered.

This project was crowned by the development of a **simple and efficient user interface**, allowing users to input their ID and instantly receive a **personalized list of book recommendations**, thus highlighting the system's **practical and functional aspect**. We experienced firsthand how **artificial intelligence** can enhance the **user experience** in selecting appropriate books and deliver **smart recommendations** based on simple data inputs.

Conclusion

Throughout this research, we have examined both the theoretical foundations and the practical implementation of a recommendation system based on **Neural Collaborative Filtering (NCF)**. Our study underscores the growing importance of personalized recommendation systems in enhancing user experience, particularly within the e-commerce domain.

Using real-world data, we implemented and evaluated an NCF-based model using **PyTorch**. assessing its performance with ranking-based metrics such as **Hit Ratio (HR)** and **Normalized Discounted Cumulative Gain (NDCG)**. While the results were promising, certain limitations remain, notably the small dataset size and the cold-start issue arising from sparse ratings.

To enhance system performance and applicability, future work can explore hybrid approaches that combine collaborative filtering with content-based features such as book genres and author information. Incorporating **contextual factors** (e.g., user location, reading time, or device type) and integrating a top-K evaluation module may also provide more realistic and robust assessments of recommendation quality.

From a deployment perspective, improvements could include developing a more intuitive **web or mobile interface**, and migrating to a **cloud-based infrastructure** to ensure scalability and real-time responsiveness. These upgrades would not only improve performance and usability but also promote better user engagement and long-term adoption.

In conclusion, this project lays a solid foundation for building more intelligent, context-aware, and user-centric recommendation systems.

Bibliography

[1] Alexandre Abraham et al. "Machine learning for neuroimaging with scikit-learn". In: *Frontiers in neuroinformatics* 8 (2014), p. 14.

[2] Amazon Web Services. *The Difference Between Machine Learning and Deep Learning*. 2024. URL: https://aws.amazon.com/ar/compare/the-difference-between-machine-learning-and-deep-learning/.

[3] K Aswini and S Bama. "Advantages and challenges of e-commerce customers and businesses: in Indian perspective". In: *Shanlax International Journal of Management* 6.S1 (2018), pp. 173–176.

[4] Baeldung. *Cross-Validation: k-Fold and Leave-One-Out*. 2023. URL: https://www.baeldung.com/cs/cross-validation-k-fold-loo.

[5] Tushar Kumar Biswas. "Data and information theft in e-commerce, jurisdictional challenges, related issues and response of Indian laws". In: *Computer Law & Security Review* 27.4 (2011), pp. 385–393.

[6] Google Cloud. *Deep Learning vs. Machine Learning: What's the Difference?* 2024. URL: https://cloud.google.com/discover/deep-learning-vs-machine-learning.

[7] Michael Galarnyk. *Train Test Split: What It Means and How to Use It*. 2025. URL: https://builtin.com/data-science/train-test-split.

[8] GeeksforGeeks. *Adam Optimizer*. 2025. URL: https://www.geeksforgeeks.org/adam-optimizer/.

[9] GeeksforGeeks. *Cross Validation in Machine Learning*. 2025. URL: https://www.geeksforgeeks.org/cross-validation-machine-learning/.

[10] GeeksforGeeks. *Getting Started with PyTorch*. 2025. URL: https://www.geeksforgeeks.org/getting-started-with-pytorch/.

[11] GeeksforGeeks. *Gradient Descent Algorithm and Its Variants*. 2025. URL: https://www.geeksforgeeks.org/gradient-descent-algorithm-and-its-variants/.

[12] GeeksforGeeks. *ML | Stochastic Gradient Descent (SGD)*. 2025. URL: https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/.

[13] GeeksforGeeks. *RMSProp Optimizer in Deep Learning*. 2025. URL: https://www.geeksforgeeks.org/rmsprop-optimizer-in-deep-learning/.

[14] Teddy Surya Gunawan et al. "Development of video-based emotion recognition using deep learning with Google Colab". In: *TELKOMNIKA (Telecommunication Computing Electronics and Control)* 18.5 (2020), pp. 2463–2471.

[15] Yanli Guo and Zhongmin Yan. "Recommended system: attentive neural collaborative filtering". In: *IEEE access* 8 (2020), pp. 125953–125960.

[16] Charles R Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (2020), pp. 357–362.

[17] Xiangnan He et al. “Neural collaborative filtering”. In: *Proceedings of the 26th international conference on world wide web*. 2017, pp. 173–182.

[18] Kaggle Community. *What is a Kaggle?* https://www.kaggle.com/discussions/general/328265. 2022.

[19] Balraj Kumar and Neeraj Sharma. “Approaches, issues and challenges in recommender systems: a systematic review”. In: *Indian journal of science and technology* 9.47 (2016), pp. 1–12.

[20] Xiang Liu et al. “Cyber security threats: A never-ending challenge for e-commerce”. In: *Frontiers in psychology* 13 (2022), p. 927398.

[21] Amir Manzoor. *E-commerce: an introduction*. Amir Manzoor, 2010.

[22] Wes McKinney et al. “pandas: a foundational Python library for data analysis and statistics”. In: *Python for high performance and scientific computing* 14.9 (2011), pp. 1–9.

[23] Prem Melville and Vikas Sindhwani. “Recommender systems.” In: *Encyclopedia of machine learning* 1 (2010), pp. 829–838.

[24] Nils J Nilson. “Introduction to machine learning”. In: *AN EARLY DRAFT OF A PROPOSED TEXTBOOK. Robotics Laboratory. Department of Computer Science Stanford University. Stanford. USA. Recuperado De: Http://Ai. Stanford. Edu/~ Nilsson/MLBOOK. Pdf* (1998).

[25] NVIDIA. *Recommendation System - NVIDIA Glossary*. February 9, 2025. URL: https://www.nvidia.com/en-us/glossary/recommendation-system/.

[26] Pallets Projects. *Flask Documentation (3.1.x)*. Pallets. 2025. URL: https://flask.palletsprojects.com/en/stable/.

[27] Efstratios G Paschalidis. “Exploring Modern Recommendation Systems: A Comparative Study of Different Model Architectures on MovieLens Dataset for Performance, Efficiency and Complexity.” In: (2025).

[28] Fabian Pedregosa et al. “Scikit-learn: Machine learning in Python”. In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830.

[29] Zheng Qin. *Introduction to E-commerce*. Springer science & business media, 2010.

[30] Tanmayee Salunke and Unnati Nichite. “Recommender systems in e-commerce”. In: *arXiv preprint arXiv:2212.13910* (2022).

[31] Michel F Sanner et al. “Python: a programming language for software integration and development”. In: *J Mol Graph Model* 17.1 (1999), pp. 57–61.

[32] J Ben Schafer, Joseph Konstan, and John Riedl. “Recommender systems in e-commerce”. In: *Proceedings of the 1st ACM conference on Electronic commerce*. 1999, pp. 158–166.

[33] Amazon Web Services. *The Difference Between GPUs and CPUs*. Accessed on April 6, 2024. 2023. URL: %5Curl%7Bhttps://aws.amazon.com/compare/the-difference-between-gpus-cpus/?nc1=h_ls%7D.

[34] Osvaldo Simeone et al. “A brief introduction to machine learning for engineers”. In: *Foundations and Trends® in Signal Processing* 12.3-4 (2018), pp. 200–431.

[35] Qi Wang et al. “A comprehensive survey of loss functions in machine learning”. In: *Annals of Data Science* 9.2 (2022), pp. 187–212.

[36] Shuai Zhang et al. “Deep learning based recommender system: A survey and new perspectives”. In: *ACM computing surveys (CSUR)* 52.1 (2019), pp. 1–38.

[37] Xuirui Zhang and Hengshan Wang. “Study on recommender systems for business-to-business electronic commerce”. In: *Communications of the IIMA* 5.4 (2005), p. 8.

Appendices

.1 Data Loading and Processing Code.

```
import os
import pickle
import pandas as pd

def load_and_preprocess_data():
    if os.path.exists(data_pickle_path):
        print("  Loading preprocessed data...")
        with open(data_\input{appendices/appendix_A}pickle_path, "rb") as f:
            return pickle.load(f)

    print("  Preprocessing data from scratch...")
    df = pd.read_csv(data_path)

    # Filter based on user and book activity
    user_counts = df['user_id'].value_counts()
    book_counts = df['book_id'].value_counts()

    # Retain the top 90% of active users and books
    user_threshold = user_counts.quantile(0.1)
    book_threshold = book_counts.quantile(0.1)

    df = df[df['user_id'].isin(user_counts[user_counts >= user_threshold].index)]
    df = df[df['book_id'].isin(book_counts[book_counts >= book_threshold].index)]

    # Normalize ratings
    df = df[df['rating'] > 0].copy()
    df['rating'] = (df['rating'] / 10.0).astype('float32')

    # Create numerical encodings for user and book IDs
    user_id_mapping = {id_: idx for idx, id_ in enumerate(df['user_id'].unique())}
    book_id_mapping = {id_: idx for idx, id_ in enumerate(df['book_id'].unique())}

```

```
    # Apply the encodings to the data
    df['user_idx'] = df['user_id'].map(user_id_mapping)
    df['book_idx'] = df['book_id'].map(book_id_mapping)

    # Calculate book popularity
    book_popularity = df['book_id'].value_counts(normalize=True).to_dict()

    # Save the preprocessed data
    data = {
        "user_id_mapping": user_id_mapping,
        "book_id_mapping": book_id_mapping,
        "book_popularity": book_popularity,
        "df": df,
        "num_users": len(user_id_mapping),
        "num_books": len(book_id_mapping)
    }

    with open(data_pickle_path, "wb") as f:
        pickle.dump(data, f)

    return data

data = load_and_preprocess_data()
user_id_mapping = data["user_id_mapping"]
book_id_mapping = data["book_id_mapping"]
book_popularity = data["book_popularity"]
df = data["df"]
num_users = data["num_users"]
num_books = data["num_books"]

print(f"  Number of users: {num_users}")
print(f"  Number of books: {num_books}")
```

.2 Book Recommendation Model (NCF).

```python
class BookRecommendationModel(nn.Module):
    def __init__(self, num_users, num_books, embedding_dim=64):
        super().__init__()

        # Embedding layers for users and books
        self.user_embedding = nn.Embedding(num_users, embedding_dim)
        self.book_embedding = nn.Embedding(num_books, embedding_dim)

        # MLP layers to process concatenated embeddings
        self.mlp = nn.Sequential(
            nn.Linear(embedding_dim * 2, 256),
            nn.BatchNorm1d(256),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.4),

            nn.Linear(256, 128),
            nn.BatchNorm1d(128),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.3),

            nn.Linear(128, 64),
            nn.BatchNorm1d(64),
            nn.LeakyReLU(0.1)
        )

        # Output layer to produce final rating prediction
        self.output = nn.Sequential(
            nn.Linear(64, 1),
            nn.Sigmoid()
        )
        self._init_weights()  # Initialize weights

    def _init_weights(self):
        # Initialize embedding and linear layer weights
        nn.init.xavier_uniform_(self.user_embedding.weight)
        nn.init.xavier_uniform_(self.book_embedding.weight)
        for layer in self.mlp:
            if isinstance(layer, nn.Linear):
                nn.init.kaiming_normal_(layer.weight, mode='fan_out', nonlinearity='leaky_relu')
                nn.init.zeros_(layer.bias)

    def forward(self, user_ids, book_ids):
        # Compute prediction from user and book IDs
        user_emb = self.user_embedding(user_ids)
        book_emb = self.book_embedding(book_ids)
```

```
        x = torch.cat([user_emb, book_emb], dim=1)
        x = self.mlp(x)
        return self.output(x).squeeze()
```

.3 Torchinfo Output for Model Architecture Analysis.

```
==========================================================================================
Layer (type:depth-idx)                   Input Shape               Output Shape              Param #                   Kernel Shape
==========================================================================================
BookRecommendationModel                  [1]                       --                        --                        --
├─Embedding: 1-1                         [1]                       [1, 64]                   4,058,688                 --
├─Embedding: 1-2                         [1]                       [1, 64]                   9,249,728                 --
├─Sequential: 1-3                        [1, 128]                  [1, 64]                   --                        --
│    └─Linear: 2-1                       [1, 128]                  [1, 256]                  33,024                    --
│    └─BatchNorm1d: 2-2                  [1, 256]                  [1, 256]                  512                       --
│    └─LeakyReLU: 2-3                    [1, 256]                  [1, 256]                  --                        --
│    └─Dropout: 2-4                      [1, 256]                  [1, 256]                  --                        --
│    └─Linear: 2-5                       [1, 256]                  [1, 128]                  32,896                    --
│    └─BatchNorm1d: 2-6                  [1, 128]                  [1, 128]                  256                       --
│    └─LeakyReLU: 2-7                    [1, 128]                  [1, 128]                  --                        --
│    └─Dropout: 2-8                      [1, 128]                  [1, 128]                  --                        --
│    └─Linear: 2-9                       [1, 128]                  [1, 64]                   8,256                     --
│    └─BatchNorm1d: 2-10                 [1, 64]                   [1, 64]                   128                       --
│    └─LeakyReLU: 2-11                   [1, 64]                   [1, 64]                   --                        --
├─Sequential: 1-4                        [1, 64]                   [1, 1]                    --                        --
│    └─Linear: 2-12                      [1, 64]                   [1, 1]                    65                        --
│    └─Sigmoid: 2-13                     [1, 1]                    [1, 1]                    --                        --
==========================================================================================
Total params: 13,383,553
Trainable params: 13,383,553
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 13.38
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 53.53
Estimated Total Size (MB): 53.54
```

Figure 27: Analysis of the structure of the proposed model.

.4 Training log.

```
Start training...
Epoch 1: Train Loss = 1.1976 | Val Loss = 0.0215
=> Improved model saved.
Epoch 2: Train Loss = 0.7049 | Val Loss = 0.0184
=> Improved model saved.
Epoch 3: Train Loss = 0.3062 | Val Loss = 0.0143
=> Improved model saved.
Epoch 4: Train Loss = 0.2119 | Val Loss = 0.0137
=> Improved model saved.
Epoch 5: Train Loss = 0.1544 | Val Loss = 0.0134
=> Improved model saved.
Epoch 6: Train Loss = 0.1177 | Val Loss = 0.0133
=> Improved model saved.
Epoch 7: Train Loss = 0.0925 | Val Loss = 0.0137
Epoch 8: Train Loss = 0.0713 | Val Loss = 0.0134
Epoch 9: Train Loss = 0.0565 | Val Loss = 0.0136
Epoch 10: Train Loss = 0.0421 | Val Loss = 0.0132
=> Improved model saved.
Epoch 11: Train Loss = 0.0377 | Val Loss = 0.0133
Epoch 12: Train Loss = 0.0344 | Val Loss = 0.0136
Epoch 13: Train Loss = 0.0315 | Val Loss = 0.0136
Epoch 14: Train Loss = 0.0244 | Val Loss = 0.0138
Epoch 15: Train Loss = 0.0220 | Val Loss = 0.0138
Epoch 16: Train Loss = 0.0216 | Val Loss = 0.0138
Epoch 17: Train Loss = 0.0184 | Val Loss = 0.0137
Epoch 18: Train Loss = 0.0166 | Val Loss = 0.0138
Early stopping triggered.
```

Code Listing 3: Training Log

.5 training Code.

```

# ============================== 5. Training Setup ==========================
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = BookRecommendationModel(num_users, num_books).to(device)

# Optimized loss function and optimizer
criterion = nn.HuberLoss(delta=0.3)
optimizer = optim.AdamW(model.parameters(), lr=0.001, weight_decay=1e-3)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=2, factor=0.5)

# ============================== 6. Model Training ==========================
def train_model(model, train_loader, val_loader, epochs=50):
    best_val_loss = float('inf')
    patience = 8  # for early stopping
    no_improve = 0
    history = {'train': [], 'val': []}

    for epoch in range(epochs):
        model.train()
        train_loss = 0

        # Partial freezing of embeddings for stability in early epochs
        if epoch < 2:
            model.user_embedding.requires_grad_(False)
            model.book_embedding.requires_grad_(False)
        else:
            model.user_embedding.requires_grad_(True)
            model.book_embedding.requires_grad_(True)

        for users, books, ratings in tqdm(train_loader, desc=f"Epoch {epoch+1}"):
            users, books, ratings = users.to(device), books.to(device), ratings.to(device)

            optimizer.zero_grad()
            outputs = model(users, books)
            loss = criterion(outputs, ratings)

            # L2 regularization
            l2_reg = sum(p.pow(2.0).sum() for p in model.parameters())
            loss += 0.001 * l2_reg

            loss.backward()
            torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)  # gradient clipping
            optimizer.step()

            train_loss += loss.item()
```

```
        # Validation phase
        model.eval()
        val_loss = 0
        with torch.no_grad():
            for users, books, ratings in val_loader:
                users, books, ratings = users.to(device), books.to(device), ratings.to(device)
                outputs = model(users, books)
                val_loss += criterion(outputs, ratings).item()

        avg_train = train_loss / len(train_loader)
        avg_val = val_loss / len(val_loader)
        history['train'].append(avg_train)
        history['val'].append(avg_val)

        print(f"Epoch {epoch+1}: Train Loss = {avg_train:.4f} | Val Loss = {avg_val:.4f}")

        # Learning rate scheduling
        scheduler.step(avg_val)

        # Early stopping & saving the best model
        if avg_val < best_val_loss:
            best_val_loss = avg_val
            no_improve = 0
            torch.save(model.state_dict(), save_model_path)
            print("=> Best model saved!")
        else:
            no_improve += 1
            if no_improve >= patience:
                print("Early stopping triggered.")
                break

    # Plot training and validation loss
    plt.figure(figsize=(5, 3))
    plt.plot(history['train'], label='Train Loss')
    plt.plot(history['val'], label='Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Training Progress')
    plt.legend()
    plt.grid(True)
    plt.show()
```

.6 Model evaluation function.

```python
# ==================== 7. Evaluation ====================
def calculate_hr_ndcg(preds, true, top_k=10):
    """Compute Hit Rate (HR) and Normalized Discounted Cumulative Gain (NDCG) at top-k"""
    # Get indices of top-k predicted scores (descending order)
    top_k_indices = np.argsort(preds)[::-1][:top_k]

    # Hit Rate: check if any relevant item is in top-k
    hr = 1.0 if np.sum(true[top_k_indices] >= 0.5) > 0 else 0.0

    # Calculate DCG
    dcg = 0.0
    for i, idx in enumerate(top_k_indices):
        rel = true[idx]
        dcg += rel / np.log2(i + 2)

    # Calculate IDCG (ideal DCG)
    idcg = 0.0
    true_sorted = np.sort(true)[::-1]
    for i in range(min(top_k, len(true_sorted))):
        rel = true_sorted[i]
        idcg += rel / np.log2(i + 2)

    ndcg = dcg / idcg if idcg > 0 else 0.0

    return hr, ndcg

def evaluate_model(model, loader):
    """Evaluate model performance on a given dataset"""
    model.eval()
    preds, true = [], []
    hr_list, ndcg_list = []

    # Disable gradient calculation for evaluation
    with torch.no_grad():
        for users, books, ratings in loader:
            # Move data to the appropriate device (CPU/GPU)
            users, books, ratings = users.to(device), books.to(device), ratings.to(device)

            # Get model predictions
            outputs = model(users, books)

            preds.extend(outputs.cpu().numpy())
            true.extend(ratings.cpu().numpy())

            # Compute HR and NDCG for each sample
```

```
            batch_preds = outputs.cpu().numpy()
            batch_true = ratings.cpu().numpy()

            for i in range(len(batch_true)):
                hr, ndcg = calculate_hr_ndcg(batch_preds[i:i+1], batch_true[i:i+1])
                hr_list.append(hr)
                ndcg_list.append(ndcg)

    # Compute evaluation metrics
    mae = mean_absolute_error(true, preds)
    rmse = np.sqrt(mean_squared_error(true, preds))
    hr_avg = np.mean(hr_list)
    ndcg_avg = np.mean(ndcg_list)

    # Print evaluation results
    print(f"\ n Model Evaluation:")
    print(f"MAE: {mae:.4f} | RMSE: {rmse:.4f}")
    print(f"HR@{10}: {hr_avg:.4f} | NDCG@{10}: {ndcg_avg:.4f}")

    # Plot rating distributions
    plt.figure(figsize=(8, 4))
    plt.subplot(1, 2, 1)
    plt.hist(true, bins=20, alpha=0.7)
    plt.title('Actual Ratings Distribution')

    plt.subplot(1, 2, 2)
    plt.hist(preds, bins=20, alpha=0.7, color='orange')
    plt.title('Predicted Ratings Distribution')

    plt.tight_layout()
    plt.show()
```

Code Listing 4: Model evaluation function

.7 Improved Recommendation System.

```python
# Get popular books based on number of ratings and average score
def get_popular_recommendations(top_n=50):
    popular_books = df.groupby('book_id').agg({
        'book_title': 'first',
        'book_author': 'first',
        'rating': 'mean',
        'user_id': 'count'
    }).sort_values('user_id', ascending=False).head(top_n)

    recommendations = []
    for book_id, row in popular_books.iterrows():
        recommendations.append({
            'book_id': book_id,
            'title': row['book_title'],
            'author': row['book_author'],
            'predicted_rating': float(row['rating']),
            'popularity': float(row['user_id'] / df['user_id'].nunique())  # Normalized popularity
        })

    return recommendations

# Recommend books based on user type (new, low activity, or active)
def recommend_books(user_id, top_n=10, diversity=0.2):
    if user_id not in user_id_mapping:
        print(" New user! Showing popular books...")
        return get_popular_recommendations(top_n)

    user_idx = user_id_mapping[user_id]
    user_ratings = df[df['user_id'] == user_id]

    if len(user_ratings) < 3:
        print(" Low activity user! Mixing recommendations...")
        personalized = get_personalized_recommendations(user_idx, top_n//2)
        popular = get_popular_recommendations(top_n - len(personalized))
        return personalized + popular

    return get_personalized_recommendations(user_idx, top_n, diversity)

# Get personalized recommendations with diversity (combines prediction and popularity)
def get_personalized_recommendations(user_idx, top_n, diversity=0.2):
    seen_books = set(df[df['user_idx'] == user_idx]['book_idx'])
    all_books = set(range(num_books))
    unseen_books = list(all_books - seen_books)

    if not unseen_books:
```

```
        return get_popular_recommendations(top_n)

    user_tensor = torch.LongTensor([user_idx] * len(unseen_books)).to(device)
    book_tensor = torch.LongTensor(unseen_books).to(device)

    with torch.no_grad():
        pred_scores = model(user_tensor, book_tensor).cpu().numpy()

    combined_scores = []
    for book_idx, score in zip(unseen_books, pred_scores):
        book_id = list(book_id_mapping.keys())[book_idx]
        popularity = book_popularity.get(book_id, 0)
        combined_score = (1 - diversity) * score + diversity * popularity
        combined_scores.append(combined_score)

    top_indices = np.argsort(combined_scores)[-top_n:][::-1]
    recommended_books = []

    for idx in top_indices:
        book_idx = unseen_books[idx]
        book_id = list(book_id_mapping.keys())[book_idx]
        book_data = df[df['book_idx'] == book_idx].iloc[0]

        recommended_books.append({
            'book_id': book_id,
            'title': book_data['book_title'],
            'author': book_data.get('book_author', 'Unknown'),
            'predicted_rating': float(pred_scores[idx] * 10),
            'popularity': book_popularity.get(book_id, 0)
        })

    return recommended_books
```

Code Listing 5: Improved Recommendation System with Popularity and Diversity Handling

.8 Training, Evaluation and Recommendation Test.

```python
# Main function: training, evaluation, and testing recommendations
def main():
    print("\ Starting training...")
    train_model(model, train_loader, val_loader)

    model.load_state_dict(torch.load(save_model_path))
    print("\n Best model loaded")

    print("\n Evaluation on test data:")
    evaluate_model(model, test_loader)

    print("\n Recommendation system test:")
    try:
        random_user = np.random.choice(df['user_id'].unique())
        print(f"\n Recommendations for user {random_user}:")
        recs = recommend_books(random_user, top_n=5)
        for i, rec in enumerate(recs, 1):
            print(f"{i}. {rec.get('title', 'No Title')[:50]}... - Predicted Rating:
                {rec.get('predicted_rating', 0):.1f}")

        print("\n New user (not in system):")
        new_user_recs = recommend_books(999999, top_n=3)
        for i, rec in enumerate(new_user_recs, 1):
            print(f"{i}. {rec.get('title', 'No Title')[:50]}... (Popularity: {rec.get('popularity',
                0):.2f})")

    except Exception as e:
        print(f"\n Error during recommendation: {str(e)}")
        print("Error details:", traceback.format_exc())

if __name__ == "__main__":
    main()
```

Code Listing 6: Model Training, Evaluation and Recommendation Test