UNIVERSITY OF CAPE TOWN

MASTER'S DISSERTATION

A Review-Aware Multi-Modal Neural Collaborative Filtering Recommender System

Author:
Pavan SINGH

Supervisor:
Assoc. Professor Ian DURBACH
Dr. Allan E CLARK

A dissertation presented for the degree of
Master of Science Advanced Analytics

from the

Department of Statistical Sciences

April 1, 2024

Acknowledgements

I would like to thank many people who made this journey happy, inspiring and rewarding. First and foremost, I would like to thank my advisors Ian Durbach and Allan Clark. I have learned a lot from their deep insights in statistics and their passion about developing scientific and practically useful methodologies. They were both very generous with their time and are always there to help. I am deeply grateful for all the inspiring discussions we had and the constructive feedback I received. This project would not be possible without their valuable guidance and input.

I am also very grateful for the great faculty and staff at Department of Statistics. Finally, I am thankful to my sibling and parents for their unconditional love and support. Because of them, no matter where I am and what I am doing, I always feel encouraged and loved.

I would additionally like to acknowledge the assistance and support from my friends and fellow statistics masters students, who have provided persistent support and advice during this year.

Declaration of Authorship

I, Pavan SINGH, declare that this dissertation titled, "A Review-Aware Multi-Modal Neural Collaborative Filtering Recommender System" and the work presented in it are my own. I confirm that:

- This work was done wholly while in candidature for a research degree at this University.
- The contents of this dissertation has not been previously submitted for a degree or any other qualification at this University or any other institution.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

[illegible]

Signed:

Date: April 1, 2024

UNIVERSITY OF CAPE TOWN

Abstract

Department of Statistical Sciences

Master of Science Advanced Analytics

A Review-Aware Multi-Modal Neural Collaborative Filtering Recommender System

by Pavan SINGH

Online shopping has become a ubiquitous aspect of modern life and recommender systems have become a crucial tool for e-commerce giants to efficiently sift through vast amounts of data to locate the information that users are seeking. Within e-commerce, recommender systems aim to provide users with personalised product recommendations based on their preferences and behaviours. They analyse user data, for example their browsing history, purchase history, and ratings to understand their preferences and make recommendations that align with these preferences. They have become fundamental for information retrieval and provide a particularly lucrative landscape for e-commerce platforms, providing suggestions that effectively prune large information spaces so that users are directed toward those items that best meet their needs and preferences.

This paper looks at developing a neural collaborative filtering (NCF) recommender system model which incorporates data from multi-modalities, textual data and explicit ratings data (and review sentiment). The primary objectives of this study are twofold. Firstly, the aim is to create and assess the efficacy of the, relatively new, deep learning-based collaborative filtering approach - NCF - in comparison to other more traditional collaborative filtering models, commonly used. Secondly, the study seeks to investigate the potential impact of incorporating product review text and review text sentiment in improving the accuracy of recommendations. Our model shall be trained and evaluated on the Amazon Product Reviews dataset, which contains millions of user reviews and feedback on thousands of different products across different categories. The metrics used to evaluate the model include predictive accuracy metrics such as mean absolute error, amongst others, as well as top-n evaluation metrics such as recall@n and precision@n. Our methodology is based on a literature analysis and aims to clearly extrapolate on the recent works which have established a framework for NCF.

The results of our study show that the NCF model outperforms all the benchmark models in terms of predictive accuracy and top-n evaluation. The results also show that the inclusion of review text in the NCF model improves the predictive accuracy of the model significantly. The results of this study are significant as they demonstrate the potential benefits of incorporating review text into deep learning-based approaches for collaborative filtering for improved rating prediction.

Contents

List of Abbreviations

DNN	**D**eep **N**eural **N**etwork
IBCF	**I**tem-**B**ased **C**ollaborative **F**iltering
MAE	**M**ean **A**bsolute **E**rror
MF	**M**atrix **F**actorisation
MLP	**M**ulti-**L**ayer **P**erceptron
MSE	**M**ean **S**quare **E**rror
NCF	**N**eural **C**ollaborative **F**iltereing
NeuMF	**Neu**ral **M**atrix **F**actorisation
NMF	**N**on-**N**egative **M**atrix **F**actorisation
RMSE	**R**oot **M**ean **S**quare **E**rror
UBCF	**U**ser-**B**ased **C**ollaborative **F**iltering

Chapter 1

Introduction

In an age where there is an exponentially increasing volume of data being produced, the ability for a user to get the information they seek has become ever more challenging [132]. The abundance of information which gives an overwhelming number of options to a user when making a decision is known as the information overload problem [10]. With the vast amount of information available, it can be challenging for users to find items or products that meet their preferences or needs. Recommender systems are tools which directly aim to address the challenge of information overload [100]. The general idea behind a recommender system is to narrow down the perceived available options and present a limited set of personalised choices (recommendations) based on said user's preferences, behaviour and or other relevant factors [100]. As such, a recommender system can be seen as a filtration tool which greatly washes out undesirable results and brings forth content desired or more relevant to a user's current interests and needs. In this way they are able to help to reduce information overload and make the decision-making process easier and more efficient. A lucrative byproduct of this is an improved overall user experience, increased engagement and user satisfaction. Given these outcomes, recommender systems have become a highly researched area over recent years [124]. Recommender systems are a crucial tool in the arsenal of e-commerce platforms which try to help consumers navigate through an abundance of product options. To this end, recommender systems are capable of enhancing the customer experience by showing products that they are inclined to want and thus also boost sales of these products. The apparent effectiveness of these systems has led to this surge in research in this domain. In this thesis, we aim to directly contribute to this area of research by developing and extending upon an established recommender system method by incorporating data from multiple modalities and investigates the potential impact of incorporating this additional modality in improving the accuracy of recommendations.

This chapter will introduce recommender systems in Section 1.1 along with the different recommender paradigms which have gained a lot of attention in this study area. In Section 1.2 we draw light to the research problem and objectives of this thesis. The specific research questions and significance of this thesis is explained in Section 1.3 in detail before concluding the chapter by explaining the structure of the remaining sections of this thesis in Section 1.4.

1.1 Recommender Systems: definition and types

A recommender system is a type of information filtering system that seeks to predict user preferences or interests by generating *recommended* items such as products, services or content that the user is likely to be interested in [124]. "Item" is the general term used to denote what the system recommends. Instead of sifting through irrelevant information and products, users are presented with content and products that are more likely to be of interest to them. Recommender systems have quickly become a necessity, given that users cannot search through an abundance of content to connect with products, services or knowledge (i.e., items) that are important to them [124].

Recommender systems deal with two types of information: characteristic information, which includes details about the items, such as keywords or categories; and user-item interactions, which encompass data like user ratings or likes and so on. With respect to the types of recommender systems available, the data (ratings data, item characteristics data, etc.) also plays an important role in determining which recommender system would be effective. In the field of recommender systems, there exist three main branches: collaborative filtering, content-based filtering and hybrid systems [143].

Collaborative-based filtering, probably the most extensively implemented type of recommender, is based on historical user-item interactions [143]. It operates by analysing patterns of user preferences across items to make predictions or suggestions. The collaborative filtering methods do not rely on explicit domain knowledge or item attributes but rather on the collective wisdom embedded within the data. At a high level, there are two types of collaborative filtering algorithms which have been proposed in the literature: memory-based algorithms, which rely on computing the similarity between users or items in order to make recommendations, and model-based algorithms, which rely on building a model or mathematical representation of the data to make recommendations [143]. Both memory-based and model-based algorithms have their strengths and weaknesses which we discuss further later on.

Content-based filtering, by contrast, is based on characteristic information and works by understanding the underlying features (or characteristics) of each item [143]. There is no general high-level categories for which content-based filtering algorithms are universally recognised, as like we have for collaborative-based filtering (model-based vs memory-based). Regardless, the idea is that a content-based filtering algorithm analyses the features of an item and then recommends other items with similar features. These features could be the attributes of an item, like the colour or style of a product, or they could be keywords, like the genres or artist names of a song.

We often find in practice that these two types of recommender system techniques (collaborative and content-based) are combined to form **hybrid systems** [143]. These use both types of information, with the idea to leverage the strengths of each individual method to improve the accuracy and relevance of the recommendations, whilst also avoiding problems that are generated when working with just one kind of system [143]. These hybrid modelling approaches have been shown to outperform individual algorithms in many cases, however they also carry with it additional complexities in terms of model building and efficiency [20]. Furthermore, whilst combining different models can often produce a more accurate model, there are other approaches that can be used to improve the accuracy and relevance of recommendations. One such approach is using multi-modal recommender systems, which leverage multiple types of data or information sources to make recommendations [20]. Figure 1.1 shows at a high-level, the main types of recommender systems and the popular subcategories within these branches. Note that hybrid recommenders, like content-based filtering, do not necessarily have recognised high-level categories, however some approaches can be bundled into some high-level categories such as weighted approaches.

Multi-modal refers to the integration of different types or modes of data in a system [144]. In the context of recommender systems, multi-modal simply refers to the use of multiple types of data to make recommendations. By combining information from different modalities, such as text, images, audio, and video, multi-modal recommender systems can provide a more comprehensive understanding of user preferences and offer more personalised recommendations [144]. Using multi-modalities also poses the opportunity to alleviate data sparsity[1] by leveraging or including auxiliary information that may encode additional clues on how users consume items. Examples of such data (referred to as modalities) are social networks, item's descriptive text, or customer product reviews [144]. Studies on multi-modal recommender systems have become a popular topic in the field as many understand the potential upside of incorporating multiple data types into a prediction algorithm [144]. In fact, multi-modal recommendation systems have become the industry standard and are widely used across many domains as companies try to personalise their products and services to better meet the needs and preferences of their customers [87]. For example, in e-commerce, a multi-modal recommender system can take advantage of product images and descriptions, user reviews, and purchase history to make recommendations [87]. Beyond

[1]Sparsity refers to the situation where the available data is highly incomplete, resulting in numerous missing values in the user-item interactions. This occurs due to the vast number of items and limited user interactions, making accurate predictions for less observed items challenging.

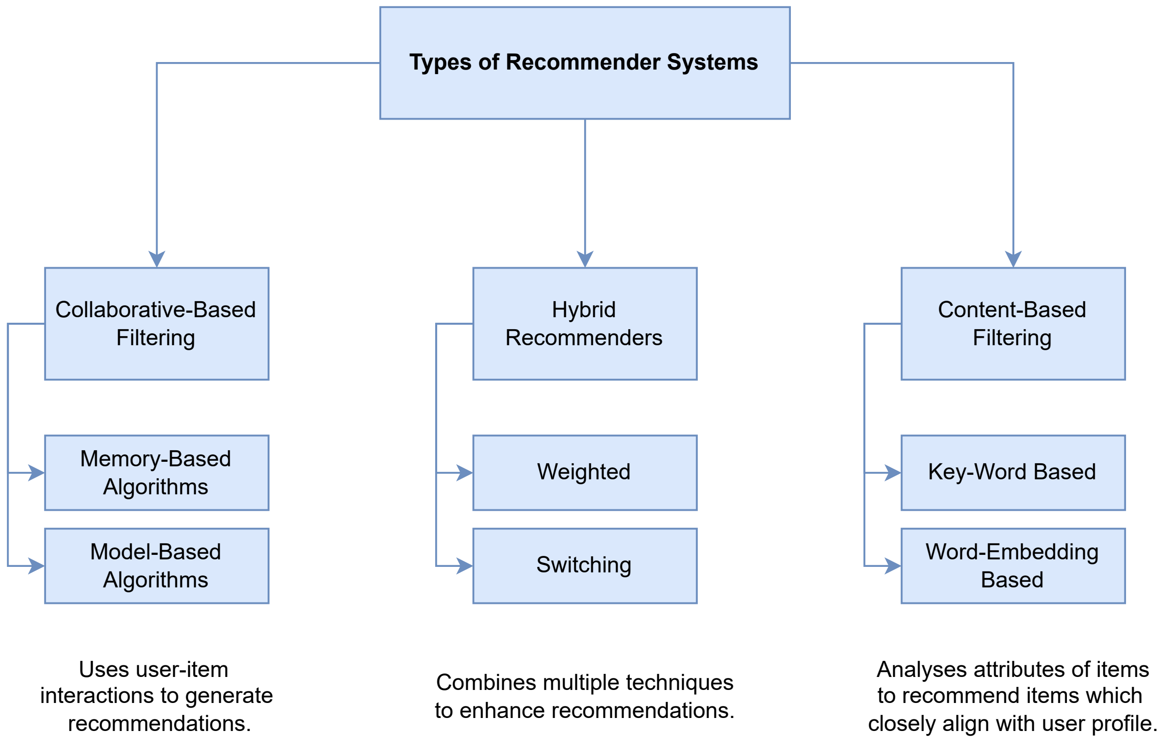

Types of Recommender Systems
Collaborative-Based Filtering
Memory-Based Algorithms
Model-Based Algorithms
Uses user-item interactions to generate recommendations.
Hybrid Recommenders
Weighted
Switching
Combines multiple techniques to enhance recommendations.
Content-Based Filtering
Key-Word Based
Word-Embedding Based
Analyses attributes of items to recommend items which closely align with user profile.

Figure 1.1: The Different Types of Recommender Systems.

multi-modalities, recommender systems have evolved to incorporate deep learning technologies. Traditional recommender systems have shown to be effective in many cases, however the recent advances in deep learning over the past decade have opened up new opportunities for improving the accuracy and personalisation of recommendations.

Deep learning is a a subset of machine learning that deals with models that are comprised of multiple layers (hence the "deep" in deep learning). In the context of recommender systems, deep learning algorithms can be used to effectively extract complex patterns and relationships from large amounts of data [56]. This is particularly attractive since the goal is to predict user preferences based on past behaviours or interactions. We shall explore the application of deep learning in the context of our recommender system by employing a neural collaborative framework which essentially generalises and puts forth a neural network-based approach to matrix factorisation (which is extensively used in collaborative filtering) to improve the accuracy of rating predictions [56]. Amazon, amongst many other internet giants, also use neural networks in their recommendation engine to make product recommendations [139].

Although these systems can easily be mistaken as simply a tool for information retrieval and data discovery - in industry their importance is unwavering [56]. With the tremendous amount of data available online, recommender systems are held in high regard amongst large internet corporations for research and investment [139].

1.2 Research Problem and Objectives

The aim of this thesis is to directly enhance the predictive accuracy of recommender systems by developing a neural collaborative filtering (NCF) model that incorporates data from multiple modalities (types), exploiting explicit numeric product ratings and product review text data as well user review sentiments. The idea is that by using this auxiliary information we can aid the NCF system account for the nuances and complexities of user preferences that may be expressed through textual data. We build on the existing work done by [56] on NCF systems and several other works ([136]; [154]) which laid the groundwork to effectively incorporate auxiliary information into collaborative filtering systems (including user reviews). As such, this thesis seeks to investigate the potential of incorporating product review text and review text sentiment as additional sources of information to improve the predictive accuracy of recommendations.

This is achieved by building a NCF recommender system which processes the explicit ratings data and is also augmented with textual features (product reviews) and sentiments (review sentiments) to form a multi-modal system. Building this neural network architecture (for collaborative filtering) enables the system to learn a non-linear mapping between the input features (i.e., item attributes) and the target output (i.e., user preferences) [56]. Through comparative evaluation with benchmark recommender models, the efficacy of the proposed multi-modal recommender will be assessed, with a particular focus on its ability to provide accurate ratings for unseen items. To this end, a number of different models will be trained, evaluated and compared to one another in an effort to establish the performance benefits of different models. The research questions presented in Section 1.3 will guide this process.

Ultimately, this thesis aims to contribute to the development of a more effective and efficient recommender system that can potentially leverage the additional textual information provided by users for products which can potentially aid in generating recommendations tailored to the unique preferences and needs of individual users.

1.3 Research Questions and Significance

As mentioned, this thesis seeks to explore the impact of incorporating product reviews and review sentiments into a recommender system model whilst also evaluating the performance of the NCF model. Recommender systems have become pivotal to the way companies interact and sell their products to their consumers [139]. A recommender system's ability to accurately predict consumer interests and desires on a highly personalised level make them a very valuable tool for content and product providers, like Amazon, Google amongst other technology giants. The key research questions addressed in this thesis are discussed and explained below.

1. **How does incorporating product reviews into a recommender system model impact the predictive accuracy and ability to recommend relevant items?**

   Here, we are interested in determining whether integrating user reviews into a recommendation algorithm leads to more accurate and relevant product suggestions for users. Another way of looking at this is examining the accuracy of a multi-modal recommender system compared to that of a single-modal recommender system - i.e., a recommender that relies on only one type of data. User reviews for products could potentially provide additional insights into user preferences or product features that are not captured by the metadata. However, reviews may also introduce noise or biases into the recommendation algorithm which can lead to less accurate suggestions [154]. Findings from this study could provide further scope for incorporating review text into product recommendation designs or highlight potential pitfalls to avoid when integrating user reviews. The study also may prove to provide further support for multi-modal recommender systems.

2. **How does incorporating review sentiment as well as review text in a recommender system impact the predictive accuracy and ability to recommend relevant items?**

   Incorporating sentiment analysis into a recommender system introduces an avenue for the integration of additional features that can contribute to enhancing recommendation accuracy. Sentiment analysis, as a technique to assess the emotional tone and polarity of textual content such as product reviews, offers a means of extracting valuable insights from user-generated content [95]. By incorporating sentiment analysis outcomes alongside traditional recommendation factors, the system gains access to a richer set of attributes that can potentially capture nuanced user preferences and sentiments.

The impact of sentiment analysis on accuracy of recommender systems provides scope to further utilise review text in possibly improving recommendation accuracy. Potential findings from this study could provide insights into the most effective sentiment analysis techniques for collaborative-based recommender systems and inform the design of more accurate and effective recommendation algorithms. Conversely, the study could highlight potential challenges and limitations of using sentiment analysis in a recommender system, such as accuracy issues due to sarcasm, ambiguity, or variations in language use.

3. **How does the performance of the collaborative-based filtering recommender system using neural collaborative filtering compare to that of popular benchmark collaborative-based recommender systems?**

   The general idea is that incorporating deep learning into a collaborative filtering model can alleviate the fragility of simpler more traditional collaborative-based filtering models in capturing user preference profiles, and as such improve accuracy of recommendations. This is another key area of interest in our study - examining the potential capacity of neural collaborative filtering system to outperform other popular collaborative-based recommender systems. In evaluating our recommender system, we assess its predictive accuracy (as well as its top-n generation) against other benchmark models. The architecture and technical details of all the developed models are provided in detail in Chapter 4 - the methodology. Deep learning models are by design more complex and often have greater computational expense and slower processing than more traditional models under collaborative filtering [56]. By investigating the potential advantage these deep learning systems have could highlight the potential trade-off between the level of desired complexity and performance. Whilst this will be domain specific and depend on the needs and resources available, the study will contribute to providing additional insight into this field.

4. **What are the potential trade-offs of incorporating product reviews into a recommender system, such as increased complexity or potential biases in the recommendations?**

   Product reviews can often contain large amounts of unstructured data which is cause for concern for recommender systems, which need to quickly process and integrate this information in their algorithm. Another potential concern may be the bias inherent in user reviews. These will be discussed in depth later on in this thesis. However, findings from this study could prove to be useful in highlighting the potential benefits and drawbacks of incorporating product reviews into a recommender system. Furthermore, it could open up paths for discussion on potential strategies for mitigating the trade-offs of using reviews in recommendation algorithms, such as developing more sophisticated sentiment analysis techniques.

These key research questions will be referenced greatly throughout the whole thesis. The significance of the findings from the study ultimately provides further support for possible future design and implementations in this space. More specifically, findings will directly contribute to the existing literature on recommender systems and provide insights into the effectiveness and limitations of incorporating product reviews and sentiment analysis features in collaborative models, as well as the trade-offs associated with these approaches.

1.4 Dissertation Outline

In this chapter we introduced what recommender systems are and the importance of them as tools to help users find the content and services they seek (or may want). Specifically, recommenders alleviate the information overload problem directly and enable users to find information as well as discover items that align to their preferences. We have also described the aim of this thesis, and the research questions it will seek to answer in 1.3. It has also introduced the primary collaborative-based multi-modal recommender system which we will be developing and examining in this thesis - NCF.

Chapter 2 is a detailed literature review which provides context for this research by first addressing the history and application of recommender systems in e-commerce. This history will start with a brief exploration of the fundamental concepts in recommender systems which have remained relevant since their inception in Section 2.1. We shall also describe the immediate benefit and advantage that recommender systems have provided for the e-commerce industry. From here, in Section 2.2, we shall explore the main recommender system paradigm of concern for this thesis: collaborative filtering. Here, a brief history of the algorithms are discussed as well as the latest breakthroughs. This will lead into a discussion on the deep learning era in Section 2.3, where recent advances are of particular importance. Additionally, we explore the prevalence of text and its applications for recommender systems in 2.4, other types of recommenders in section 2.5 and finally the key evaluation methods used in recommender system research in Section 2.6. The chapter closes with addressing the various limitations and challenges faced by collaborative filtering and recommender systems as a whole. Chapter 3 addresses all matters relating to data. In order to critically examine our NCF recommender system and assess the impacts of additional features, it is necessary to have a source of user-item data. Our source is the Amazon Product Review dataset, which is publicly available and is discussed in further detail in Section 3.1. In addition to describing and explaining the variables (Section 3.2) and the data collection process (Section 3.3), we summarise our cleaned dataset in Section 3.4 and then provide some exploration of the dataset in Section 3.5. Finally, the data partitioning process, in Section 3.6 is looked at. Chapter 4 looks at the methods used in our thesis. It provides all the technical details necessary to understand the models utilised and trained. We begin the Chapter by detailing out our overall modelling approach in Section 4.1 before diving into the details. The groundwork for this will be built in Section 4.3, where the basic architecture and training procedure for neural collaborative filtering will be discussed in length. Additionally, Section 4.4 discusses the various benchmark models that shall be built to assess the performance of the neural collaborative model. Finally, Section 4.5 details the evaluation criteria that shall be employed to assess our recommender models. Chapter 5 documents the observed results (Section 5.1) as well as a detailed discussion of them (Section 5.2) with frequent reference to the research questions raised from this Chapter (1). Chapter 6 will summarise (Seection 6.1), expand on limitations (Section 6.2) and further work (Section 6.3), as well as conclude this thesis (Section 6.4).

Chapter 2

Literature Review and Related Work

Chapter 1 briefly introduced the key concepts around recommender systems, the problem they solve and the main ideas relevant to this thesis. Chapter 2 will build on this by providing historical context through an overview of recommender systems. This overview will focus on e-commerce applications, reviewing literature done on recommenders incorporating similar features and characteristics relevant to the work done in this thesis. We divide this overview into several different sections, each addressing key discoveries and background information relevant to this thesis. To this end, the sections covered will focus on work which harnessed either collaborative-based filtering, deep learning, text analysis or a combination of these approaches all within the context of recommender systems.

2.1 Recommender Systems: history and applications

Recommender systems have evolved significantly since their inception, becoming a pivotal component of contemporary information retrieval and personalised content delivery. Due to their success they play a key part in many platforms, offering services or showcasing items to users across various different domains. In this section, we first look at the history and various applications of recommender systems (Section 2.1.1) and then we shall focus on the literature behind recommender systems within e-commerce (Section 2.1.2) specifically.

2.1.1 History and Applications

The roots of recommender systems can be traced back 1992, where [48] proposed the Tapestry system, which was the first information filtering system based on collaborative filtering through human evaluation. This system laid the groundwork for collaborative recommendation approaches for the years to come. Similarly, the GroupLens project at the University of Minnesota in 1992, introduced a framework for recommender systems which involved generating recommendations by analysing users' historical preferences and behaviours [73]. Their work significantly influenced the development of recommendation models and architectures ([73], [65]). In 1998, Amazon developed their own version of a collaborative-based filtering recommendation system, coined item-based filtering. This system changed and shaped the landscape for e-commerce [84]. Although it is unclear when content-based filtering was first introduced [9], the method became popularised in 1999 [59], where items were recommended based on their inherent attributes and user preferences. In the years that followed, research interest on recommender systems grew significantly, leading to the exploration of diverse methodologies and variations of collaborative-based or content-based (and sometimes both) recommender system methods [17].

An important milestone in recommender systems occurred in 2006 when Netflix initiated an open competition offering a $1 million prize for an algorithm that could improve on the accuracy of their system, Cinematch[1]. This competition spurred significant interest and development in recommender system applications and use cases, with the prize claimed in 2009 by a group of three researchers [11]. Their winning algorithm, an ensemble method that combined various collaborative filtering techniques, improved the accuracy of Cinematch by 10%. This competition was a significant milestone in the development of recommender systems, as it demonstrated the potential for significant (and lucrative) improvements in

[1]Cinematch was a recommendation algorithm developed by Netflix in the year 2000, which aimed to provide personalised movie recommendations to users based on their viewing history and preferences

recommendation accuracy and the value of collaborative filtering and matrix factorisation in recommendation systems.

Over the next decade, recommender systems have become critical for the success of major Internet companies such as Netflix, Amazon and Facebook, amongst others [26]. The widespread adoption of internet-based services further fuelled the proliferation of recommender systems, extending their use across diverse domains. In broad terms, when a wide variety of items exist and users differ from each other, personalised recommendations can assist in delivering suitable content to the respective individuals. Applications of such systems range from tourism, encompassing hotels, restaurants, and parks ([150],[88]); advertising [29]; business and retail [46]; medical diagnosis [109]; and music selection [13].

Table 2.1 shows some popular e-commerce sites using recommender systems and a high-level description of what these systems recommend to users of these platforms. Streaming platforms like Netflix leverage recommender systems to personalise content recommendations, significantly impacting user engagement and retention [49]. In fact, Netflix used recommender systems so extensively that their Chief Product Officer, Neil Hunt, indicated that more than 80% of movies watched on Netflix came through recommendations and placed the value of Netflix recommendations at more than $1 billion per year [49]. Social media platforms like Facebook and LinkedIn employ recommender systems to curate users' news feeds and suggest connections [6]. Recommendations were used so extensively by Amazon that a Microsoft research report estimated that 30% of Amazon's page views were from recommendations [127].

Site/Platform	**What is Recommended**
Netflix	Movies, TV shows
Amazon	Books, Fashion, and other products
Facebook	Friends, posts, articles
LinkedIn	Posts, articles, jobs
Spotify	Music, podcasts

Table 2.1: Content Recommendations on Different Platforms.

Ultimately, recommender systems continue to evolve and play a crucial role in delivering personalised content across various industries, adapting to the dynamic needs of users and businesses alike. The ongoing development and integration of these systems underscore their enduring significance in shaping the landscape of information retrieval and content delivery. We shall now describe the use of recommender systems in the e-commerce domain specifically.

2.1.2 Recommenders in E-commerce: Amazon case study

The exponential growth of the internet has transformed the operations of companies, particularly within the realm of e-commerce. E-commerce, characterised by the buying and selling of goods and services over the internet, has become a platform ubiquitous with consumers exploring and purchasing products [123]. Users' navigating through these platforms are confronted with a multitude of decisions, prompting questions such as "Which product should I purchase?" or "What brand should I choose?". For a successful e-commerce platform, efficiency in showcasing relevant products is paramount, and recommender systems have emerged as a pivotal tool in achieving this goal [122]. Recommender systems have evolved from novelties to indispensable tools that shape the world of e-commerce [122]. They are proven to have significant impacts on sales, diversity, customer retention, and revenue generation [84].

Today, prominent e-commerce websites, including Amazon, Netflix, eBay, Alibaba, and Etsy, leverage recommender systems to assist users in discovering products for purchase [6]. Recommenders are identified as able to turn browsers into buyers, addressing the challenge of users abandoning a platform when their desired item is not readily visible. In fact, research has demonstrated that users often exhibit a tendency to abandon a system when their desired item does not appear within the initial 5 or 10 search results and said user tries again on another system [84]. Recommender systems have the capability to assist these customers in discovering the products they wish to buy. Moreover, recommender systems enhance cross-selling efforts by suggesting supplementary products during the checkout process. For example, during the checkout process, a website could suggest additional items based on the products already present in the shopping cart. Moreover, effective recommenders can contribute to a 10-30% increase in cross-selling revenue [34]. Cultivating customer loyalty in a competitive online landscape is another notable outcome of recommenders, as they establish value-enhanced relationships between platforms (e.g. Amazon) and users [84]. These platforms' investment into recommenders which are able to comprehend their user's preferences, and then accordingly translate this understanding into action, result in users tending to favour their platform - since they align well with their preferences.

Looking at the history of recommender systems within e-commerce, the paper by [122] offers a comprehensive exploration of how early e-commerce platforms employ their technologies, using notable businesses like Amazon as a case study. The paper unveils key insights by scrutinising recommender system implementations and their contributions to platform profitability. Several of these recommendation strategies are still used today and stand out as revenue drivers, some of which are discussed below. These include, "Similar Items", "Text Comments", "Average Ratings", and "Top-*N* Recommendations". The "Similar Items" approach prompts customers to explore related products based on their previous purchases. "Text Comments" allow users to read (or leave) impartial comments on a product so as to get a better understanding. And "Average Ratings" provides a numerical rating which gives a convenient gauge of item quality, and "Top-*N* Recommendations" provides lists of top unrated items aligning to a users preferences. These strategies collectively contribute to the effectiveness of recommender systems in optimising the e-commerce landscape and, additionally, many of these strategies are still offered on Amazon's platform today.

Looking at Amazon speicfically beyond these strategies employed, the company says their recommender systems play a critical role in leveraging the "long tail" concept, contributing to substantial profits from their products with infrequent purchases [81]. While individually rare, the collective presence of such items can yield substantial profits. For instance, Amazon attributes a noteworthy portion of its sales, ranging from 20-40%, to products that do not fall within its top 10 000 best-selling items [15]. The e-commerce giants approach involves dynamically personalising the online store for each customer, aligning with the concept of having "2 million stores on the web" as articulated by Jeff Bezos, the CEO of Amazon [84]. Amazon's technical implementation of recommender systems involves applying a variety of recommender algorithms, such as item-based collaborative filtering. These recommender systems extend across various pages on Amazon's platform, including the homepage, search results page, and shopping cart, offering personalised recommendations based on customers' past behaviour and preferences. The paper by [84] discusses the recommender system - item-based collaborative filtering algorithm - in great detail.

Ultimately, recommender systems serve as communication bridges between products and users, enhancing customer experiences and increasing the likelihood of a sale. Amazon's success stands as a testament to the effectiveness of quality recommender systems, continuously adapting to user preferences and fostering customer loyalty. Within e-commerce, recommenders greatly impact the revenue generation and cross-selling ability of platforms by directly being able to understand customer preferences and show or highlight items aligning well to these user preferences. Having established the background and history of recommender systems (with a focus on e-commerce), we now look toward one of the primary types of recommender systems - collaborative-based filtering.

2.2 Collaborative Filtering

The basic concept behind collaborative-based filtering methods is that it harnesses the collective behavior of a community of users to make personalised recommendations, without relying on explicit domain knowledge or item attributes. The paper done by [114] serves as one of the seminal pieces of work which played a pivotal role in popularising collaborative filtering techniques for recommenders. While not the absolute first use of recommender systems, it brought attention to the concept's potential applications, especially in personalised recommendations. In 1992, the Tapestry system [48] introduced collaborative filtering, leveraging users' experiences to tailor recommendations without requiring external information about items or users. Specifically, collaborative filtering is defined as a recommendation technique used to personalise the experience of users through recommendations tailored to their interests, leveraging the experiences of other users with similar profiles [59]. It does this without need for exogenous information about either items or users [125]. It comprises two primary approaches: memory-based, recommending based on similarity between users or items, and model-based, recommending by developing a predictive model based on user interactions [140].

Both memory-based and model-based approaches have their strengths. Both practical experience and related research have reported that memory-based algorithms present excellent performance, in terms of accuracy, for multi-value (e.g., 1-5 rating scale) rating data. On the other hand, model-based algorithms can efficiently handle scalability[2] to large data sets [140]. This is not to say that these methods have no weaknesses. These traditional collaborative filtering methods face two primary challenges in a growing e-commerce landscape: real-time scalability and recommendation quality. More details about scalability are discussed later on in Section 2.7.3. Regardless, a major appeal of collaborative filtering methods in general is that it is domain free, yet it can address data aspects that are often elusive and difficult to profile using other recommender techniques such as content filtering [76]. In fact, collaborative filtering is recognised as the most successful recommender system, able to produce user-specific recommendations based on patterns of ratings or usage without the need for additional information [76].

The following subsections will outlay the details, background and literature into the two primary branches of collaborative filtering: memory-based and model-based approaches.

2.2.1 Memory-Based Algorithms

As mentioned, memory-based methods are techniques within collaborative filtering systems which involve recommending items based on a similarity measure. One prominent family of methods within memory-based collaborative filtering are neighbourhood-based methods, which focus on creating a "neighbourhood" of users (or items) based on their similarity to a target user (or item) [2]. Specifically, these methods make recommendations by first identifying similar users (or items) to a target user (or item) and then utilises the preferences of these entities to make personalised recommendations. The two most prominent neighbourhood methods are user-based and item-based collaborative filtering [60].

The user-based approach for neighbourhood methods is one of the most popular applications of memory-based collaborative filtering, largely due to its simple implementation [59]. In the mid-1990s, collaborative filtering predominantly followed a user-based methodology, where the initial step involved searching for other users with similar preferences, such as users having comparable purchase patterns. That is, identifying a set of customers whose purchased and rated items overlap with the target user's preferences (perhaps they purchased similar items or rated common items similarly in the past). By aggregating items from these similar customers, the algorithm eliminates those already purchased or rated by the user and recommends the remaining items [134]. The original GroupLens system [114] (that we have previosuly

[2]Scalability refers to the ability of a system to handle increasing amounts of data or workload without sacrificing performance.

mentioned) from 1994, implemented a user-based collaborative filtering algorithm, using users' similarities to identify a neighbourhood of nearest users. Numerous improvements to user-based algorithms have since been suggested, enhancing the predictive capabilities of these methods [14]. The original GroupLens system used Pearson correlations[3] to weight user similarity, used all available correlated neighbours, and computed a final prediction by performing a weighted average of deviations from the neighbour's mean. The Ringo music recommender [126] represented a significant advancement beyond the original GroupLens algorithm. Ringo's innovation involved achieving superior performance by calculating similarity weights through a constrained Pearson correlation coefficient[4]. Since, "similarity" is a key component in this (and all memory-based) approach, the way in which similarity is measured had garnered significant research in the early beginnings of neighbourhood-based approaches. An empirical analysis of various neighbourhood-based collaborative filtering algorithms and, specifically, similarity measures, namely Pearson correlation and cosine vector similarity, was conducted by [14]. The findings indicated that Pearson correlation demonstrated superior performance, although subsequent research suggests potential equivalence with cosine similarity [108]. User-based collaborative filtering is still synonymous with collaborative filtering till this day. Due to its simplicity and ease of understanding, it is often employed in simple recommendation tasks.

The other neighbourhood-based method is item-based collaborative filtering which was first introduced in the late 1990s and was popularised by Amazon - it is based on the items' similarities (instead of users) for a neighbourhood generation of nearest items [119]. It is a prominent technique under neighbourhood methods, where it evaluates a user's preference for an item based on the ratings of "neighbouring" items by the same user. Amazon has successfully employed item-item collaborative filtering since 2003, generating real-time, scalable, and high-quality recommendations [134]. It's successful implementation has led to a large number of research to be focused on this technique. In contrast to user-based approaches, item-to-item collaborative filtering focuses on finding similar items rather than similar customers, offering a unique recommendation approach [84]. Specifically, the item-to-item collaborative filtering algorithm builds a similar-items table to recommend highly correlated items, leveraging the cosine measure for similarity calculations. It excels in recommendation quality even with limited user data and has gained widespread use across platforms like YouTube and Netflix due to its simplicity, scalability, explainability, and immediate updates based on new customer information [40]. However, like user-based approach, item-based collaborative filtering has its flaws, such as the risk of trapping users in a "similarity hole"[5] by offering overly similar recommendations [111]. The method, like user-based approach, also struggles to scale well to larger volumes of data.

Further details about these two memory-based approaches will be explored in Chapter 4, as they serve as important comparative benchmarks in our analysis. At this point, we can highlight that all memory-based algorithms face scalability challenges when dealing with large volumes of data. To address this, dimensionality reduction techniques have been proposed to balance the trade-off between accuracy and execution time of collaborative filtering algorithms [118]. More details on this limitation (scalability) shall be explored in greater detail later on in this chapter, in Section 2.7.3.

[3]Pearson correlation is a measure of the linear correlation between two variables. It is often used to quantify the similarity between users based on their item ratings. A higher correlation coefficient indicates a stronger similarity between users' preferences.

[4]Constrained Pearson correlation is a variation of the Pearson correlation coefficient that incorporates constraints to address issues such as sparsity or data scarcity.

[5]The term "similarity hole" refers to a situation where a recommendation system continuously suggests items that are too similar to those already consumed by the user, potentially leading to a lack of diversity in recommendations and limiting the discovery of new and potentially relevant items.

2.2.2 Model-Based Algorithms

Model-based algorithms are the second branch of collaborative-based filtering systems. In contrast to memory-based methods, model-based algorithms harness the overall dataset of ratings to build predictive models, often employing statistical or machine learning techniques [3]. So, the key distinction between model-based techniques and memory-based approaches lies in their approach to generate predictions. Model-based techniques rely on models (i.e., mathematical representations or algorithms that capture patterns and relationships within the data) to derive predictions whilst memory-based methods generate predictions from the weighted similarity of preferences among users or items [3].

There are a variety of different model-based algorithms. Bayesian models [30], probabilistic relational models [45], linear regressions [119], and Latent Dirichlet Allocation [93] are among the diverse array of model-based methods. Some of the most successful realisations of model-based methods (and collaborative filtering in general) are latent factor models, namely matrix factorisation [76]. Matrix factorisation is characterised by transforming both items and users into separate latent factor spaces. Each latent factor space represents a set of latent variables (or features) that capture the underlying characteristics of either items or users [76]. Each latent factor is a dimension in the space, and the value of each dimension represents the extent to which the user or item possesses that characteristic.The model is trained to learn these latent factors, and once learned, the model can predict the rating of an item by a user by computing the dot product of the user and item latent factor vectors. Generally, the model is trained to minimise the difference between the predicted ratings and the actual ratings in the training data. The model is then used to predict the ratings of items that the user has not yet rated, and the highest predicted ratings are recommended to the user - i.e., once these latent factors are learned, the recommender system can provide personalised recommendations for each user [74]. For instance, in the context of movies, the identified factors could encompass straightforward aspects like categorising films into genres like comedy or drama, gauging the level of action, or assessing suitability for children. Additionally, these factors might delve into more ambiguous facets such as the depth of character development or the presence of unique qualities, or even include entirely unexplainable dimensions. Regarding users, each factor signifies the extent to which a user favours movies that align with the specific characteristics identified in the corresponding movie factor.

Matrix factorisation, particularly highlighted by its success in the Netflix competition, has rose in popularity, demonstrating superiority over nearest-neighbour techniques for product recommendations [76]. The acclaim for matrix factorisation stems from its exceptional scalability, effective handling of sparse data, and the ability to incorporate additional information, such as implicit feedback[6] [76], temporal effects, and confidence levels[7] [76]. However, this is not to say that matrix factorisation techniques (and additionally all model-based) approaches do not have their own set of limitations. Model-based approaches, like all collaborative filtering approaches, struggle when it comes to recommending items to new users (the cold start problem). Details of these limitations are discussed in greater detail in section 2.7. Now, having set the stage by discussing the background and different types of collaborative filtering, we shall begin to explore deep learning and the impact it has had on the recommender landscape.

2.3 Deep Learning in Recommender Systems

Over the past few decades, deep learning has achieved remarkable success in a diverse range of domains like computer vision and speech recognition [124]. Both academia and industry have quickly embraced

[6]Implicit feedback refers to user interactions with items that are not explicitly expressed as ratings, such as clicks, views, or purchases. It includes any user behaviour that can be used to infer preferences or satisfaction with items, such as consuming, using or buying an item

[7]Confidence levels represent the degree of certainty associated with observed ratings or interactions. Matrix factorisation algorithms can adaptively weight or adjust the influence of ratings based on their associated confidence levels

its application, driven by its ability to tackle complex tasks and deliver state-of-the-art results ([28]; [31]).

The most basic deep learning model entails a multilayer feed-forward neural network which undergoes training, employing back-propagation[8] or other supervised algorithms, to create a predictive statistical model for a specific input–output mapping [153]. The network learns from a set of examples, embedding information in connection weights, allowing it to generalise to examples beyond the training set. Effectively, the neural network serves as a flexible tool that can apply its learned knowledge to new data points, showcasing its ability to generalise beyond the specific examples encountered during training.

The recent (in the last decade) advancements in machine learning and artificial intelligence, have paved the way for refining and enabling recommender systems to first, integrate deep learning models, and second, achieve more accurate and relevant recommendations [56]. The successful integration and apparent advantageous performance of deep learning in recommenders has led to there being a significant increase in the number of research publications on deep learning-based recommendation methods. The majority of which have provided further strong evidence of the inevitable pervasiveness of deep learning in recommender system research [153].

Traditionally, the state-of-the-art for collaborative filtering has predominantly entailed matrix factorisation (or some derivative of it) for modelling the interaction between user and item features (see Section 2.2.2). This involved applying an inner product on the latent features of users and items. Much research effort has been devoted to enhancing matrix factorisation, such as integrating it with neighbor-based models [74], combining it with topic models[9] of item content [147], and extending it to factorisation machines[10] [112] for a generic modelling of features. Despite these numerous efforts to enhance matrix factorisation, its performance has been limited by the simplicity of the inner product interaction function (which simply combines the multiplication of latent features linearly), and may not capture the complexity of user interaction data [56]. One approach to address this limitation is increasing the number of latent factors, but this can lead to overfitting, particularly in sparse settings [112]. To tackle this challenge, deep learning based approaches have been proposed, which use a neural network architecture to model the interaction between users and items (replacing the inner product), enabling the model to learn arbitrary functions from data [56]. Furthermore, the paper by [56] has established a general framework for using this deep learning based approach for collaborative filtering, named Neural Collaborative Filtering (NCF). In their paper, their NCF model, leveraging a multi-layer perceptron for non-linearities, demonstrated consistent and statistically significant improvements over state-of-art matrix factorisation approaches such as eALS[11] and basic baselines such as item-based collaborative filtering on MovieLens and Pinterest datasets.

In addition to NCF, neural networks have been applied to develop Deep Matrix Factorisation, a hybrid technique merging matrix factorisation and deep learning [153]. In this approach, a deep neural network is employed to factorise the user-item interaction matrix, creating low-dimensional representations for both users and items [153]. This stands in contrast to NCF, where the approach involves using neural networks to directly learn representations of users and items, eliminating the need for explicit factorisation of the interaction matrix.

[8]Back-propagation is a supervised learning algorithm used to train neural networks by adjusting the network's weights in order to minimise the error between the predicted and actual outputs. It works by propagating the error backward from the output layer to the hidden layers, updating the weights based on the error gradients with respect to each parameter.

[9]Topic models are statistical models used to discover the abstract topics that occur in a collection of documents

[10]Factorisation machines are a type of machine learning model that extends traditional matrix factorisation techniques by incorporating additional features, such as user and item attributes, into the factorisation process.

[11]eALS, or explicit Alternating Least Squares, is a matrix factorisation algorithm that explicitly models observed user-item interactions using least squares optimisation.

In the contemporary landscape, numerous companies leverage deep learning to enhance the quality of their recommendations, showcasing a notable shift in the recommender system paradigm ([27]; [35];[101]). Specifically, there has been a deep neural network-based recommendation algorithm tailored for video recommendations on YouTube [35], a wide and deep model for an App recommender system on Google Play [27], and an RNN-based news recommender system designed for Yahoo! News [101]. These models underwent rigorous online testing and demonstrated significant improvements over traditional counterparts, illustrating the transformative impact of deep learning on industrial recommender applications. A comprehensive review of deep learning-based recommendation models and their applications is provided by [153], which serves as an invaluable resource for understanding the evolution of deep learning within the realm of recommender systems.

Ultimately, we follow the approach done in [56] for NCF. That is, embracing a neural network-based collaborative filtering paradigm that operates to capture and learn the user-item interactions through using multi-layer network, offering non-linear transformations in our recommender system. This can be particularly useful when the relationships between input features and user preferences are complex and non-linear, which may be the case in many real-world scenarios [56]. Furthermore, neural networks and their inherent layered structure make it easy to incorporate auxiliary information since each layer can process different aspects of the data. This provides an effective platform for which we can evaluate the influence of textual features within our NCF model, aligning with our specified research objectives (see Section 1.3). To that end, we will look into the history and literature of text within recommender systems, also exploring the relevance of text-based recommender systems.

2.4 Text Analysis in Recommender Systems

An important shortcoming of collaborative filtering systems is that these algorithms are not able to capture the rationale for a user's rating, and thus can not holistically capture a target user's preference. This is a key (and unique) problem for collaborative filtering, which generally relies on numerical ratings. To tackle these challenges, integrating additional information to collaborative filtering models have been looked at, including tags [157], geo-location [63] and user reviews [136]. In this thesis, we focus on the integration of textual information such as user reviews as a solution to the aforementioned challenge (lack of rationale). Textual information is a rich source of data that can provide nuanced insights into user preferences, and has been utilised to enhance recommender systems in many domains, including movies [43], hotels [97], and e-commerce [54]. Reviews, often in unstructured text, present a complex yet valuable pool of information [129]. These reviews offer a fine-grained, nuanced, and reliable source of user preference information, enabling the system to construct detailed user preference representations [154].

Recommender systems using text reviews generally use one of three key elements extracted from review texts: review words, review topics, and overall opinions [23]. Specifically, using term frequency-inverse document frequency[12] on review texts, we can identify representative words that capture the essence of the review [23]. Topics can be detected through methods like latent dirichlet allocation[13] [23], which can reveal the aspects discussed in reviews. Additionally, overall opinions, reflecting sentiments, can be deduced using sentiment analysis techniques.

Several works have used review texts and their related rich information like review words, review topics and review sentiments, for improving the rating-based collaborative filtering recommender systems

[12]Term frequency-inverse document frequency (TF-IDF), a statistical measure which is used to evaluate the importance of a word in a document relative to a collection of documents. TF-IDF weighting helps capture representative terms in reviews by giving higher weights to words that are frequent in the review but rare in the entire corpus.

[13]Latent Dirichlet Allocation (LDA) is a generative probabilistic model used for topic modeling, which aims to uncover latent topics present in a collection of documents. It can uncover the underlying themes or aspects discussed in the reviews.

([55]; [53]; [154]). A detailed survey of recent works that integrate review texts and also discusses how these review texts are exploited in order to mitigate the main issues of the standard rating-based systems like sparsity and prediction accuracy problems has been conducted [136]. The paper reiterated that the elements that can be extracted and used for recommenders are the actual review words, topics and opinions. They also concluded that the inclusion of textual features was associated with positive outcomes in terms of recommender accuracy. The experimental results from the paper demonstrated that the performance of the recommender system by incorporating information from reviews, produces recommendations with higher quality in terms of rating prediction accuracy compared to the baseline methods (which included matrix factorisation).

As mentioned, user sentiment or opinion is another piece of information we can extract from review text and used for recommender systems. To extract this opinion from the review we use sentiment analysis[14]. Extracting sentiments and augmenting a recommender model with it, is one of the the more conventional approaches towards the incorporating review text within recommenders [72]. Techniques like sentiment-based matrix factorisation have been proposed to integrate review sentiments into recommendation models [128]. This model has been shown to improve the quality of recommendations, particularly in the presence of sparse data [128]. In fact, sentiment analysis has been shown to be a valuable tool to complement traditional ratings, improving recommendation quality in several works ([128]; [72];[38]; [39]; [43]). Effectively, sentiment analysis on review texts to extract user opinion, poses as a viable option to not only addresses sparsity but also as a tool to complement traditional ratings, improving recommendation quality [129].

Ultimately the integration of review texts into collaborative filtering recommender systems has shown positive impacts on system performance both from a perspective of augmenting the existing user feedback (review words) and incorporating the user sentiment from review texts ([39]; [61]). As such we pursue integrating both user reviews and review sentiment into our neural collaborative filtering architecture - the details of which will be discussed further in Chapter 4.

2.5 Other Recommenders: a brief overview and history

While this thesis shall focus solely on the application and enhancement of established techniques in collaborative filtering, it is essential to acknowledge the broader landscape of recommender systems, encompassing various methodologies beyond collaborative filtering. For recommender systems, there are three predominant categories: collaborative filtering, content-based filtering, and hybrid models [118]. Collaborative filtering, the central theme of this thesis, leverages user preferences and behaviours to provide personalised recommendations [134]. In contrast, content-based filtering relies on the intrinsic attributes and characteristics of items to offer suggestions [89]. Hybrid models, as the name implies, integrates elements of both collaborative and content-based approaches, aiming to harness the strengths of each [143]. In this section, we briefly explore the historical development and distinctive features of content-based filtering and hybrid models within the realm of recommender systems, emphasising how they differ from collaborative filtering. Hereby, we can provide a greater depth of understanding of recommender systems and also place collaborative filtering within the broader spectrum of methodologies.

2.5.1 Content-Based Filtering

Content-based filtering approaches the recommendation problem as a quest to find related items - that is, to find other items with similar attributes [9]. The method relies on the item's information, neglecting

[14]Sentiment analysis, also known as opinion mining, is a natural language processing technique used to determine the sentiment or opinion expressed in a piece of text.

contributions from other users - in contrast to the workings of collaborative filtering - focusing on recommending items akin to those the user has liked in the past [106]. The simplicity of content-based filtering lies in its three-step process: extracting item attributes, comparing them with user preferences, and recommending items based on matching features [84]. One notable example of successful content-based filtering is the Music Genome Project, used by Pandora for its internet radio service /citekoren2009matrix. Content-based methods are a popular approach when it comes to domains such as streaming platforms for TV shows, music or movies where the items have a rich set of attributes that can be used to describe them and user preferences can be easily inferred from the attributes of the items [24].

Research suggests that most content-based recommender systems use relatively simple algorithms, such as keyword matching or a vector space model with basic TF-IDF weighting [98]. These models operate by representing user preferences and item descriptions as vectors in a high-dimensional space, measuring their similarity using cosine similarity or other distance metrics [148]. Recent developments in content-based filtering include word embedding techniques like Latent Semantic Indexing[15] [22] and Word2Vec[16] [102]. Effectively, these techniques represent items and user profiles in a low-dimensional vector space, allowing for measuring similarity based on semantic understanding and improving the quality of recommendations [96].

Despite its success in several domains, collaborative filtering methods often outperform content-based ones [143]. However, content-based filtering, unlike collaborative methodologies, are unaffected and alleviate the cold-start problem [24] - the challenge when it comes to recommending items to new users (or new items to users). In addition to alleviating the cold-start problem, content-based filtering methods also offer transparency in recommendation explanations by explicitly listing content features that influenced the recommendations, unlike collaborative systems, which are often perceived as black boxes [89]. However content-based algorithms often deliver recommendations that are either too broad, like best-selling drama DVDs, or overly specific, such as all books by the same author - this problem is termed overspecialisation [84]. The recommendations tend to be confined to items similar to those already rated by the user, limiting the system's ability to suggest novel or unexpected items, a phenomenon known as the serendipity problem. Introducing randomness or filtering out items that are too similar to those the user has seen before are potential solutions that have been explored to overcome these shortfalls [89].

2.5.2 Hybrid Models

Despite the successes of content-based filtering and collaborative-based approaches, each method has its own limitations, including issues like overspecialisation, cold-start problems, sparsity, and scalability. This was acknowledged early into these model inceptions and led to hybrid filtering implementations [146].

Hybrid filtering, entails a recommender system which combines two or more filtering techniques (such as collaborative-based and content-based filtering), with the aim to enhance recommender system performance and mitigate the limitations associated with the individual approaches [124]. These hybrid systems leverage the strengths of each method while compensating for their inherent weaknesses [158]. Several comparative studies demonstrate the superior performance of hybrid recommender systems compared to standalone approaches ([18];[146];[20]).

[15] Latent Semantic Indexing (LSI) is a technique to analyse the relationships between terms and documents in a corpus. It works by representing documents and terms as vectors in a high-dimensional space, where the similarity between documents or terms is calculated based on the cosine similarity between their corresponding vectors

[16] Word2Vec is a word embedding technique that learns distributed representations of words in a continuous vector space. It is based on neural network architectures, such as the skip-gram and continuous bag-of-words models, which are trained on large text corpora to predict the context or neighboring words of a target word

The most common type of hybrid models are the weighted hybrid, cascade hybrid, and switching hybrid recommender systems [20]. The weighted hybrid model combines the scores from different recommenders, the cascade hybrid model uses the output of one recommender as input to another, and the switching hybrid model select the most appropriate recommendation model based on certain conditions or user preferences [20]. All of these types of hybrid models typically involve combining collaborative and content-based filtering methods.

Although hybrid models have been shown to outperform standalone methods, they are not without their challenges. These type of models are more complex and require more computational resources [20]. Additionally, the performance of hybrid models is highly dependent on the quality of the individual methods being combined [20]. Additionally, selecting appropriate weighting or integration strategies for combining different recommendation sources can be non-trivial and may impact system performance [143]. Despite these challenges, hybrid models have been widely adopted in practice, with many commercial recommender systems, such as Netflix and Amazon, using hybrid models to provide recommendations ([20]; [143]).

2.6 Evaluation Methods

In general, evaluating recommender systems poses inherent challenges, with algorithms exhibiting variable performance across different datasets [60] due to the fact that generally collaborative filtering models (and indeed all recommender models) have been developed or tailored for specific datasets. The diverse goals of evaluations further complicate matters, as early research focused on predictive accuracy[17], while more recent efforts have looked to assess a recommender comprehensively, both from a perspective of predictive accuracy as well as its ability to generate useful and relevant recommendations [151].

Predictive accuracy metrics have facilitated early algorithm comparisons and are still widely used in evaluating recommender systems, since they are easy to understand and interpret [151]. There have been many different predictive accuracy metrics applied to collaborative filtering results, including (but not limited to) mean absolute error ([14]; [59];[108]; [114]; [126]), correlation ([62]; [119]) and mean squared error ([126]; [19]). Empirical experiments have shown that mean absolute error correlates strongly with many other proposed metrics for collaborative filtering [59], yet is easier to measure and has well understood significance measures. Furthermore, mean absolute error is still the most frequently used metric among collaborative filtering researchers [151]. We used mean absolute error, mean squared error and root mean square error as our chosen predictive accuracy metrics to report the performance of the prediction problem because they are most commonly used and easiest to interpret directly [151]. More details on these metrics will be discussed in Chapter 4.

Although most early research in the field has focused on improving accuracy of recommender systems, an accurate predictive recommender system does not necessitate that the recommender will generate useful or relevant recommendations. As such there have been additional, more user-centric, metrics that have been proposed to evaluate recommender systems. These metrics assess the ability of the recommender to generate useful and relevant recommendations [91]. Generally, these user-centric metrics are more appropriate for evaluating the performance of a recommender system when a list of recommendations is the final output. Here, the focus is on the quality of the recommendations, rather than the accuracy of the predictions [151]. This is termed top-n evaluation, where the goal is to evaluate the quality of the top-n recommendations generated by the system [36]. In top-n evaluation, we use certain metrics to evaluate a recommenders ability to recommend useful or relevant items to users. These metrics include precision, recall and F1 score [36]. Given that top-n evaluation is particularly valuable

[17]Predictive accuracy measures how close the recommender system's predicted ratings are to the true user ratings.

in e-commerce settings, we shall use these metrics to evaluate the performance of our recommender system. We shall expand on these top-*n* metrics further in Chapter 4.

Often, depending on the goals of the evaluation, different metrics may be more appropriate. For instance, if the goal is to assess the ability of the recommender to generate useful and relevant recommendations in a list of recommendations only, then top-*n* evaluation metrics are more appropriate. However, if the goal is to assess the accuracy of the predictions only, then predictive accuracy metrics are more appropriate. However, in practice, it is often beneficial to use both predictive accuracy and top-*n* evaluation metrics to evaluate the performance of a recommender system in a holistic manner [151]. To this end, we acknowledge the importance of a holistic view of a recommender system. To facilitate this, we incorporate predictive accuracy metrics as well as top-*n* evaluation metrics to get a clearer picture as to the performance of our recommender and assess it in a comprehensive manner.

2.7 Limitations and Challenges

Recommenders, particularly collaborative filtering, have achieved success across diverse domains by delivering personalised content tailored to users on various platforms [26]. Despite their widespread effectiveness, these systems are not immune to certain limitations. While we have briefly touched upon these limitations throughout this Chapter, this Section aims to provide a more comprehensive overview of the challenges and limitations that collaborative filtering recommenders face. These limitations are specific to collaborative filtering models; however, we also provide some domain-specific challenges that e-commerce recommender applications pose.

2.7.1 Sparsity

Sparsity, formally, refers to the situation where the available data in a collaborative filtering recommender system is limited, resulting in a sparse user-item interaction matrix with a significant number of missing values [65]. In scenarios where the pool of available items is exceptionally large, as observed in major e-commerce platforms with millions of items [46], the overlap between items and users becomes minimal. Effectively, we will often have instances where users don't purchase the same items. In e-commerce, it is not uncommon for users to rate or purchase only a small fraction of the available items, leading to a high degree of sparsity in the user-item interaction matrix - frequently exceeding 99 percent sparsity [46].

A high level of sparsity critically hinders the effectiveness of collaborative filtering approaches [37], since it diminishes the ability of the (collaborative) system to identify meaningful patterns or similarities between users or items, ultimately impacting the quality and reliability of the generated recommendations [19].Specifically, it impedes the calculation of similarities among users, a fundamental aspect of collaborative filtering. When the sparsity is extensive, the system struggles to find a sufficient number of overlapping preferences between users, leading to ineffective similarity computations. And, even when similarities are calculable, they become unreliable due to the inadequacy of information caused by the sparsity [37].

There has been several approaches to mitigate the sparsity problem in collaborative filtering. Namely, augmenting the user-item interaction matrix with additional information, such as user demographics, item attributes, or user reviews, has been proposed to alleviate the sparsity problem [136]. Another approach is to use content-based filtering to alleviate the sparsity problem, as content-based filtering relies on the intrinsic attributes and characteristics of items to offer suggestions, and is not affected by the sparsity problem [89].

Ultimately, the challenge when users rate on a few items makes it increasingly difficult to discern their interests accurately. While this thesis does not primarily focus on this limitation, it is nonetheless addressed due to the augmenting our collaborative model with review text - an approach well-known for mitigating or assisting recommenders in coping with sparsity issues [136].

2.7.2 Cold Start Problem

The cold start problem is a persistent age-old challenge encountered in recommender systems when dealing with new users or items that lack sufficient information in the system [83]. The problem emerges when novel users or items are introduced to the user-item matrix[18], preventing collaborative filtering methods from generating accurate recommendations due there being not enough ratings available about them [65]. This differs from the sparsity problem, which is concerned with the lack of ratings across the entire user-item matrix, whereas the cold start problem is concerned with the lack of ratings for new users or items specifically [83].

Effectively, the cold start problem encompasses three distinct scenarios within the recommender system domain [83]. The first scenario arises when a new user joins the system, and no prior information is available about their preferences, constituting the *new user* cold-start problem. The second scenario emerges with the introduction of a completely new item to the system, lacking any associated ratings—an issue known as the *new item* cold start problem. And finally, the third scenario occurs during the initial launch of the system when both user and item information are absent, characterising the cold start system problem. In these instances, content-based solutions, renowned for their effectiveness in handling information scarcity, can be applied to mitigate the challenges associated with cold start problems.

To address this issue, hybrid recommender techniques, combining both content and collaborative data, have been employed as solutions [83]. This is because content-based filtering leverages the inherent characteristics of items, such as textual or numerical features, to provide recommendations even when user-item interactions are limited or absent [83]. Another approach is for a recommender to ask for some base information (such as age, location and preferred genres) from the users, also known as active learning ([19]; [156]). Using this information, the system can profile a user and generate initial recommendations for them, and then use the user's feedback to improve the recommendations [156]. Notably, another approach has proposed a novel technique that tracks individual users' activities across multiple e-commerce sites, allowing recommendations for a cold-start user in one site to be informed by their records in other sites [86].

In our thesis, addressing the cold start problem is not considered and is beyond the scope of the work, as such we mitigate any risk of the cold start problem by selecting or using a pool of users and items with sufficient information available (see Section 3.3).

2.7.3 Scalability

Scalability is formally defined as the ability of a system or process to handle a growing amount of work, resources, or an expanding user base, without compromising performance, efficiency, or overall functionality [19]. As such, scalability becomes a key concern for many platforms as they grow, and their user base or item catalog rises in numbers. For example, Amazon deals with millions of customers and has a catalog of items equally as big [134]. The enormity of this available data provides both opportunities and challenges. The challenge lies in the computational complexity of collaborative filtering algorithms, particularly memory-based methods, which grow quadratically with the number of users or items [130]. This is due to the fact that collaborative filtering algorithms (specifically memory-based methods) rely on the computation of a similarity measure, which becomes prohibitively expensive with larger datasets,

18

hampering scalability and necessitating significant memory and computational power [134]. The scale at which recommender systems operate, especially for successful internet companies like Amazon, incurs substantial infrastructure costs and limitations due to the escalating volume of processed data [130]. This added dimension makes an interesting problem for building recommenders.

In practice, the issue of scalability is addressed by dividing the prediction generation steps of the recommendation process into offline and online components, where the offline part requires extensive computation, and the online component dynamically generates predictions for users in real time ([130]; [118]). Additionally, it is also not uncommon for companies to incorporate collaborative filtering algorithms into distributed computing engines such as Apache Hadoop or Spark, leveraging their speed and efficiency in parallel large-scale data processing ([19];[134]). Other more simplistic techniques to handle scalability involve methods such as sampling users, data partitioning, and omitting high or low-frequency items to manage scalability, yet these strategies face the risk of compromising the quality of recommendations [130]. Dimensionality reduction techniques like clustering and principal component analysis have also been considered too, however, similar to aforementioned techniques they can potentially adversely impact recommendation quality by eliminating low-frequency items [121].

These challenges of scalability have been long standing issues in recommender systems landscape and have been present since the inception of collaborative filtering. However, they have become more pertinent obstacles nowadays with the vast amounts of data readily available. Effectively, we acknowledge and discuss this limitation to raise awareness, though it is not an immediate concern for this thesis. To address this potential obstacle in our work, we opted to reduce our dataset to a more manageable size. This allows us to eliminate any concerns related to scalability, ensuring a smoother and more efficient execution of our analysis.

2.7.4 Domain and Data Challenges

Another key challenge faced by collaborative filtering methods, in particular, are rooted in their reliance on users' numeric ratings as the primary source of preference information, leading to a significant limitation in recommendation accuracy due to the often inadequate semantic explanation of scalar rating information ([81]; [129]). Recognising this drawback, efforts have been made to enhance recommendation accuracy by integrating other information with ratings - such as user reviews. This was briefly touched on in Section 2.4, when discussing the role of text for recommenders.

Furthermore, each domain has their inherent challenges. In the context of e-commerce, there are a variety of characteristics of the field which make it slightly more challenging for collaborative filtering methods. Some of these challenges have been discussed already (cold-start problem and scalability), however, we discuss them below in the context of e-commerce.

1. **Scalability and need for real-time results**: As mentioned in the previous subsection (see Section 2.7.1), recommenders in E-commerce are often faced with scalability obstacles. Large retailers contend with extensive data, managing tens of millions of customers and millions of distinct catalog items. Navigating this vast landscape poses a considerable challenge to generate useful recommendations. This coupled with the fact that many applications demand real-time results, requiring recommendations to be generated in no more than half a second while maintaining high-quality outputs. This necessity adds a layer of complexity to the operational efficiency of recommendation algorithms [84].
2. **Cold Start Problem**: There is limited information for new customers or items on an e-commerce platform - as discussed in earlier subsections (see Section 2.7.2). Given the limited or non-existent user history, it is difficult to generate accurate recommendations for new user. In e-commerce platforms, this is particularly challenging as the user base is constantly growing and new items are being added to the catalog [84].

3. **Volatile Customer Data**: Customer interactions in e-commerce are dynamic and volatile, with each new interaction offering valuable data. Effectively, the preferences and interests of users are constantly evolving, and their interactions with items are continuously changing. Recommendation algorithms must promptly respond to this evolving information to ensure relevance and accuracy [84].

Besides these aforementioned challenges, many novel issues have begun to appear more recently. Generally, progress and propagation of new techniques brings new challenges [134]. For example, GPS equipped mobile phones have become mainstream and internet access is ubiquitous, as such location-based recommendation is now feasible and increasingly significant[19].

Additionally, one of the more interesting challenges for research on recommender systems is that there are a large number of factors which affect recommendation quality; namely, data partitioning (train and test sizes), similarity measures, recommendation list size amongst some others. These factors are greatly explored in the paper by [140]. We have thus used the literature from this paper to infer the most commonly used approaches and techniques used in selecting the optimum values for these certain factors. These will be discussed in detail in Chapters 3 and Chapters 4.

Given these discussions on the specific limitations for this domain, the scope of this thesis does not extend to a comprehensive discussion or addressing these issues. Primarily, we are concerned with development and analysis of a neural collaborative filtering network together with multiple modalities (text and ratings data) to better improve the performance (in terms of predictive accuracy and top-*n* generation) of recommender systems. Effectively, we have taken the neccesary steps to adjust the dataset to mitigate any potential issues related to scalability and cold start problems. The limitations that these recommender systems approaches suffer from inherently as discussed in this section were to simply build a better picture of collaborative filtering approach.

2.8 Conclusion

This Chapter reviewed past work which is related to our research. We began this Chapter by presenting a giving a brief history of recommender systems as well as their impact and application in e-commerce, where they play a significant role in better engaging and driving sales between content and consumer. Recommender systems prove to be a key element in the operation of e-commerce platforms looking to meet their customer's needs. These recommender systems filter out information desired or would perhaps be of interest to the user. Ultimately, a good recommender system staves off a user's time, keeps them engaged, enhancing their experience and consequently leads to increased revenue.

In Section 2.2, we delved into the details and literature around collaborative-based filtering, one of the three main recommender system paradigms. We found that this method has its own unique strengths as well as weaknesses. For example, collaborative filtering can excel in providing accurate recommendations based on user preferences, historical behaviors, or item similarities, but its performance can be significantly negatively affected by user feedback sparsity.

We also looked at deep learning and text analysis for recommender systems in Sections 2.3 and 2.4 respectively, particularly focusing on the recent implementations of neural collaborative filtering. This historical analysis provides context for the choices made in this thesis, such as the decision to use a deep learning approach for collaborative filtering as well as why we augment our NCF model to incorporate textual features and user sentiments. Specifically, there is scope that neural collaborative filtering enhances traditional collaborative filtering methods by capturing intricate patterns and non-linear relationships in user-item interactions, while the incorporation of textual features and user sentiments has

[19]Websites like Foursquare, Gowalla, Google Latitude, Facebook, Jiapang, and others already provide location-based services and show that many people want to share their location information and get location-based recommendations.

been shown to enrich a model's understanding of user preferences, resulting in more nuanced and tailored recommendations (as well as addressing sparsity concerns in the data).

We also addressed all the various aspects of evaluation for recommender systems in Section 2.6. This serves as a guide to help support the decision for the metrics used in this thesis. We have decided to use predictive accuracy metrics as well as top-*n* evaluation metrics to assess our recommender systems going forward. This decision enables us to have a more comprehensive perspective of the performance of the recommender system as well as alleviate the problems pertaining to the narrow rating prediction (predictive accuracy) metrics.

The final Section in this Chapter addressed the limitations and challenges faced by collaborative-based filtering systems and recommender systems as a whole in general. This section discussed in length data sparsity, cold start problem, scalability as well as domain specific problems such as the volatile nature of customer preferences. To this end we outlined that the scope of this thesis does not seek to answer or address all of these problems. As such we have taken the necessary steps to mitigate any potential issues related to scalability and cold start problems. The concern of data sparsity is inadvertently addressed through us augmenting our collaborative filtering with textual features.

Ultimately, this review has established context for the rest of the thesis. As we transition to the next Chapter on the data used in this thesis, it becomes evident that the success of these models is tied to the quality and characteristics of the data they leverage. Understanding the nuances of data used in recommender systems is crucial for establishing our methodologies. In the forthcoming section, we focus on the the data used in our thesis.

Chapter 3

Data Exploration and Analysis

Chapter 2 provided a review of literature done on the history and applications of collaborative filtering models, as well as the research conducted within the space of text analysis, deep learning and collaborative filtering in the context of e-commerce. Chapter 3 will discuss the details of the dataset used for the development of the neural collaborative filtering recommender system using both user sentiment and user reviews. This Chapter is divided into several sections; the first of which will introduce the data set chosen for this thesis (Section 3.1), followed by some descriptions of the variables in the dataset (Section 3.2). We shall then provide some technical details about how we collected and prepared the data (Section 3.3), also detailing how the user review text was cleaned as well as the sentiment analysis (Section 3.3.2). Following this, we shall provide a summary of the data, giving some high-level figures and statistics (Section 3.4). We shall then explore the data by identifying trends and patterns within the dataset (Section 3.5). To conclude this chapter, we provide a description of how we partitioned the dataset for the subsequent analyses (Section 3.6).

3.1 Amazon Review Dataset

Amazon, as one of the world's largest online retailers, facilitates an immense volume of transactions and interactions daily [70]. Like many other large e-commerce platforms, they contain a wealth of user-generated content, particularly in the form of product reviews. For this research, we use the Amazon Review dataset [94]. This dataset has gained significant popularity and recognition within the research community due to its size, diversity, and real-world relevance ([70]; [52]; [133]). It spans numerous product categories, ranging from electronics and books to fashion and home appliances. Due to the vast number of reviews and diverse range of products, the Amazon product review dataset offers a large-scale and real-world representation of user preferences and behaviours in an e-commerce setting. Specifically, this dataset contains just over 75 million reviews on the Amazon products between May 1996 and February 2018 along with user profiles and item metadata. The data gathered over this 22-year period, enables it to be a rich and dynamic source of information reflecting the evolving landscape of consumer choices on the Amazon platform.

Specifically, each review in the dataset is accompanied by a written text component where users express their opinions, experiences, and feedback regarding the products they have purchased. The significance of the Amazon product review dataset in the context of this thesis lies in its ability to provide a rich source of information for building and evaluating a review-aware neural collaborative-based filtering recommender system. By incorporating user reviews (and additionally, review sentiments) into the recommendation process, the system may gain a deeper understanding of user preferences and the factors that influence their satisfaction with products, as well as perhaps alleviate the limitations of traditional collaborative filtering methods (namely, sparsity) as discussed in Section 2.7.1. The dataset also enables us to directly ask and answer the questions posed in this thesis (see Section 1.3).

In summary, the Amazon review dataset serves as a foundational element for this thesis, offering a real-world, diverse and expansive source of information to build and evaluate a review-aware neural collaborative filtering recommender system. The diverse nature of products (spanning across different categories) and wide array of information (including review text and product ratings) available within the Amazon dataset, enables us to effectively study the impact of incorporating sentiment and user reviews in recommendation systems.

3.1.1 Feedback and Data Types

Recommender systems rely on different types of input data, which are often placed in a matrix with one dimension representing users and the other dimension representing items. Specifically, in recommender systems, user feedback falls into two primary categories: explicit and implicit. Explicit feedback is directly provided by users through numerical ratings - often accompanied by their text review which is another form of explicit feedback [75]. Implicit feedback refers to users' implicit preferences derived from their behaviour, such as purchase history or browsing patterns. Implicit feedback can be inferred from indicators like the number of times a product was purchased, the frequency of clicks, or the length of time spent on a product page [75].

In our thesis, we concentrate on explicit feedback, which (as mentioned) involves direct and intentional input from users regarding their preferences and satisfaction with products. This explicit feedback is represented through numerical ratings (on a numerical scale from 1-5), user reviews in our dataset, and review sentiments. Review sentiments are considered to be derived explicit feedback, since they are inferred from the explicit review text provided by the user[25]. Additionally, the written review goes beyond simple numerical ratings and provides a more detailed and contextual understanding of users' experiences, highlighting specific features, pros and cons of the product.

While this thesis only uses explicit feedback, it is worth noting that almost all platforms nowadays gather implicit feedback along with explicit [75]. A list of examples of these (explicit and implicit feedback) are shown in Table 3.1.

Company	**Recommendations**	**Explicit Data Captured**	**Implicit Data Captured**
Netflix	Movies, TV Shows	User ratings, Watched or not	Viewing history, like what you watched, how long you watched it, etc. Scroll and hover behavior on content.
Amazon	Products, Movies, Books, etc	Product reviews and ratings	Products viewed, including categories and items. Shopping cart activity.
Spotify	Music, Podcasts	User-generated playlists and liked songs	Play count of songs, skip behavior of songs, time of day you listen to certain music.
YouTube	Videos, Music	User interactions (likes, dislikes, comments, shares)	Watch history like what you watch, how long you watch, engagement, Click-through rate, likelihood of users clicking on recommendations based on thumbnails.
LinkedIn	Connections (people)	Recommendations, join groups, endorse skills	Profile views, job searches.

Table 3.1: Comparison of Data Captured by Different Companies

In summary, our thesis focuses on explicit feedback, encompassing numerical ratings, user reviews and review sentiments. This approach allows for a comprehensive examination of user preferences and sentiments, providing the foundation for the development and evaluation of a neural collaborative filtering recommender system that leverages direct user input, via ratings and reviews, to enhance the accuracy of recommendations.

3.2 Variable Description

As highlighted, the Amazon product reviews dataset contains millions of records. Each record corresponds to a user review of a certain product in the Amazon catalog. It is important to note that the dataset was anonymised and stripped of any personally identifiable information to ensure (individual's) privacy and that it complies with ethical considerations. Each review (record) comprises a variety of features - namely the product ID, review ID, rating and review text amongst several others. The wealth of available item information presents opportunities for exploring additional features that could potentially enhance recommendations, including features like price, brand or images. Although incorporating these features are not studied in this thesis, they are discussed for potential future work in Section 6.3. We are particularly interested in the user review text and the user product ratings. A detailed breakdown of all the features for each record is shown in Table 3.2, including the feature name in the dataset, the feature description and examples of how they appear in the data.

Feature	**Feature Description**	**Example**
reviewerID	ID of the reviewer	A2SUAM1J3GNN3B
asin	ID of the product	0000013714
vote	Helpful votes of the review	23
style	Style or variant of the product	Black
reviewText	Text of the review	“This product exceeded my expectations”
overall	Rating of the product	4
summary	Summary of the review	"Does what it needs to"
unixReviewTime	Time of the review (unix time)	1426777271
reviewTime	Time of the review (raw)	2019-08-15 14:30:00
title	Name of the product	“Samsung HDTV”
description	Description of the product	“High-definition smart television with HDR”
price	Price in US dollars	$799.99
imageURL	URL of the product image	(http://example.com/image.jpg)
brand	Brand name	Samsung Electronics
categories	List of categories the product belongs to	Electronics

Table 3.2: Variable Descriptions

3.3 Data Collection and Preprocessing

The specific data (Amazon product reviews) used in this thesis was sourced from an online repository, which contained a comprehensive collection of Amazon product reviews (between 1996 and 2018) organised in 29 JSON files categorised by product categories [104] (See Appendix B for more details on the code and data). That is, each JSON file has reviews associated with products from a specific category. There were 29 product categories, and hence 29 JSON files, with each category containing on average over 8 million reviews. Despite the categorisation, the size of each file posed a computational challenge and demanded substantial computational resources for data management. The volume of reviews within individual files proved too large for efficient processing using Pandas[1], rendering direct loading into memory unfeasible.

To address this issue, we designed specific code which selectively reads random selections of rows from each JSON file. Specifically, for each category (JSON file), we extracted up to 500 000 randomly chosen reviews. For categories having fewer than 500 000 reviews, we opted to read all available entries. Following the extraction of each subset, we stored these randomly selected reviews in a CSV file, reset the environment, and proceeded to load another file representing another distinct category. Once we had repeated this process for all categories, we successfully merged the randomly selected reviews from the 29 different CSV files into a unified dataset, creating a comprehensive CSV file containing reviews spanning various categories.

Subsequently, we aimed to further reduce this dataset, as well as address potential cold start problems (see Section 2.7.2). The unified dataset was still relatively large (just over 9 million reviews), but also highly sparse. For example, over 30% of users had only one review, making it difficult to evaluate collaborative filtering algorithms as we would encounter the cold start problem. As such, we filtered the dataset and only retained users with at least 13 reviews. The minimum threshold of 13 was chosen to ensure that we had enough data to train and evaluate the recommender system. To achieve this filtering, we first identified which users from the merged file have over 13 reviews, and then took that user's ID (*reviewerID*) and identified each and every review associated with this ID. The result of this was a significantly reduced subset. We followed a similar process to ensure that each product had at least 13 reviews. Effectively, after ensuring that each user and product had at least 13 reviews, we were left with a dataset of 91 235 rows (or reviews), 3801 unique users and 3283 unique products. This reduction in the dataset size alleviates the cold start problem, and also enables us to handle the dataset using the available computation resources more effectively. Having loaded and reduced the number of reviews, we sought to conduct a thorough data cleaning process, prior to analysis and building of recommender models.

3.3.1 Data Preprocessing

Upon acquiring, loading and filtering the data, our focus shifted to applying certain preprocessing steps aimed at cleaning the dataset. Data cleaning is a crucial step in any data analysis project as it helps eliminate inconsistencies, missing values, and irrelevant information that could affect the accuracy and reliability of the results [149]. To begin the data cleaning process, the dataset was loaded into a Pandas DataFrame[2], making it easier to manipulate and analyse the dataset. In order to focus the analysis on the relevant variables and reduce computational complexity, certain features that were not directly related to the research question were identified and removed from the DataFrame. Some of these columns included the fields 'image', 'description', 'price', to name a few - these were dropped since we were not considering these features in our analysis. In fact, the only fields that were kept were: *reviewerID*, *asin*, *reviewText* and *overall* (see Table 3.2 for definitions). We also made sure to address missing values appropriately to ensure the integrity of the dataset. Any rows that contained a null value were considered unnecessary for the analysis as they lacked the essential information needed for building our recommender system. Therefore, such rows were removed from the DataFrame. We also identified that there were many users who rated an item but provided no written review. We removed these reviews from the dataset as well. Duplicates in the dataset can introduce bias and impact the reliability of the analysis [149]. Therefore, duplicate rows in the DataFrame were identified and removed. Another issue we picked up on was that some users had reviewed the same product multiple times. To address this, we decide to take the first (earliest) review (based off the review time) from the user for said product - removing the other reviews. Moreover, in preparation for further analysis, we transformed the ratings

[1]A popular data manipulation library in Python.

[2]A two-dimensional, tabular data structure provided by the Pandas library in Python.

feature, by normalising this value. Two of the most popular rating normalisation schemes that have been proposed to convert individual ratings to a more universal scale are mean-centering and the Z-score [41]. We opted for mean centering approach, since it is a simpler method, that involves subtracting the mean rating from each individual rating. By bringing all ratings to a common scale, biases arising from variations in rating scales were mitigated [57].

During this preprocessing we had removed some reviews from the dataset. As a result, we need to re-validate and ensure that each user and product had at least 13 reviews. To that end, we were left with 83 139 reviews in the dataset, where we had 3668 unique users and 3249 unique products. Having completed the high-level data cleaning process, we proceeded to conduct a more detailed pre-processing of the review text, as well as sentiment analysis, which is discussed in the following section.

3.3.2 Text Analysis and Cleaning

As outlined in Section 2.4, the idea is to use the review text (and sentiments) as additional information to supplement the ratings for collaborative filtering. The motive is that this additional information (reviews and sentiments) provides potentially valuable insights and opinions expressed by users about the items they have reviewed. By including the review text as a feature in our dataset, we can potentially leverage natural language processing techniques to extract meaningful information and sentiment from the text to better inform the recommendation process [81]. However, to achieve or study these potential outcomes, the text data needs to undergo a cleaning process, ensuring its readiness and appropriateness for subsequent analysis [23]. To fulfil this requirement, we conduct a through text cleaning procedure.

Text cleaning refers to the process of preparing textual data for analysis by removing unnecessary or irrelevant information and transforming the text into a suitable format [47]. In the context of this thesis, text cleaning specifically refers to the steps taken to preprocess and clean the review text data before incorporating it into recommenders as well as performing sentiment analysis. As such, the textual data (*reviewText*) was subjected to several principal text preprocessing steps - which we will discuss in detail. These text analysis techniques were applied so as to extract meaningful information from the review text and enable it to be incorporated with our explicit numerical ratings for the recommender model to be built at a later stage. It was also pivotal to clean the text before conducting sentiment analysis to generate the sentiment feature. The cleaning steps as well as an explanation of the techniques used and choices made shall be made clearer, next.

We began by normalising[3] the text in the *reviewText* column, by first converting the string to lowercase.

We began by normalising[4] the text in the *reviewText* column, by first converting the text to lowercase. We then removed all of the stop words[5] and punctuation, including any non-alphabetic characters, as well as the removal of any HTML tags and any other special symbols present in the review text which provided no additional information. This step effectively strips away the noise and enhances the focus on meaningful content [117]. Subsequently, we tokenised the field which facilitated uniformity in subsequent analyses. Tokenisation is the process of breaking down the text into individual words (sentences, etc.) or tokens [117]. This step also proves quite useful preparation for sentiment analysis as we will be able to get the sentiment of each word individually and capture the overall sentiment of the text - more on this in Section 3.3.3. One of the final steps we took in our text analysis cleaning is to lemmatise the *reviewText*. Lemmatisation is a technique used to reduce words to their root form [117]. For

[3]Normalisation in text processing involves converting text to a standard form or representation to make it easier to analyse or compare. In this context, normalising the text in the *reviewText* column typically involves transforming it to lowercase to ensure consistency in subsequent text processing tasks.

[4]Normalisation in text cleaning involves converting text to a standard form or representation to make it easier to analyse or compare. It often involves several steps.

[5]Stop words are commonly used words (e.g., "the," "is," "and") that do not carry significant meaning in the sentiment analysis process.

example, stemming would convert "running", "runs" and "ran" to the root word "run". This step helps to consolidate similar words and reduces the dimensionality of the text data [47]. After executing all these steps, we were left with a cleaned version of the review text data, which was stored in a new column in the DataFrame, which we named *cleanedText*. This column will be used in the subsequent sentiment analysis and recommendation system development.

3.3.3 Sentiment Analysis

Having undergone a comprehensive text cleaning process on the *reviewText* column, our dataset is now primed for sentiment analysis. In the context of this thesis, we aim to augment our recommender system with two sources of information: review text and review sentiments. To get the review sentiments we need to extract it from the review text. Sentiment analysis is tool that we can use to determine the sentiment of the review texts [128]. Using the sentiments, we can gauge the user's satisfaction or dissatisfaction with the product. For recommender systems, sentiment analysis can provide a deeper understanding of user preferences and opinions, which can perhaps be leveraged to enhance the accuracy and relevance of recommendations [25].

Formally, sentiment analysis, also known as opinion mining, refers to the task of identifying or associating sentiment with a given text, such as a review [95]. This process, sentiment analysis, employs a variety of methods, encompassing machine learning algorithms [4], lexicons [141], rule-based approaches [67], and ensemble methods [71]. The overarching goal is to automatically categorise text as positive, negative, or neutral, or alternatively, assign sentiment scores that reflect the degree of positivity or negativity within the text [95]. These sentiment scores or labels subsequently facilitate our analysis by providing a more nuanced understanding of customer opinions, as well as valuable insights into the prevailing sentiments regarding the products under consideration. At times, these sentiments might more accurately reflect a user's preference or liking for an item compared to a numerical 5-scale rating alone [23].

In our experimentation, we explored three sentiment lexicon-based[6] methods: VADER [67], BING [64], and AFINN [99]. These methods were chosen due to their simplicity, ease of use, and effectiveness in sentiment analysis tasks ([141]; [67]).

VADER operates as a lexicon and rule-based[7] sentiment analysis tool, systematically assigning sentiment scores to each token in the text.

VADER operates as a lexicon and rule-based[8] sentiment analysis tool, systematically assigning sentiment scores to each token in the text [67]. This evaluation takes into account both the polarity (positive or negative) and the intensity of sentiments. The underlying mechanism involves a pre-built lexicon containing words associated with varying sentiment intensities, forming the basis for calculating the sentiment scores. In contrast, BING adopts a lexicon-based sentiment analysis approach, where each word is distinctly labeled as positive or negative in the lexicon [64]. Sentiment scores for a given text are determined through a tally of positive and negative words present in the text. The ultimate sentiment determination typically relies on the majority of positive or negative terms within the analysed content. AFINN, similar to BING, is characterised by a list of pre-computed integer-valued sentiment scores

[6]Sentiment lexicon-based methods rely on predefined dictionaries or lexicons containing words or phrases annotated with sentiment scores or polarity labels. These lexicons are used to determine the sentiment of text by matching words in the text with entries in the lexicon and aggregating their sentiment scores.

[7]In sentiment analysis, lexicon and rule-based approaches rely on predefined dictionaries or lexicons containing words or phrases annotated with sentiment scores or polarity labels. Additionally, they may incorporate rules or heuristics to handle linguistic nuances and context-specific sentiment expressions.

[8]Lexicon and rule-based approaches rely on predefined dictionaries or lexicons containing words or phrases annotated with sentiment scores or polarity labels. Additionally, they may incorporate rules or heuristics to handle linguistic nuances and context-specific sentiment expressions.

assigned to words [99]. In this lexicon-based method, each word is given a numerical score, and the overall sentiment of a text is calculated as the sum of the scores of its constituent words. A positive sum denotes a positive sentiment, while a negative sum signifies a negative sentiment. AFINN differs from BING in that it assigns a numerical score to each word, as opposed to a binary positive or negative label [95].

We implemented sentiment analysis using the three methods on the cleaned review text data. To implement these methods we used the NLTK library[9] in Python. The sentiment scores generated by each method were then compared to determine the most effective method for our dataset. To compare the sentiment scores from AFINN, BING and VADER, we normalised the scores from AFINN and BING to be between -1 and 1 - the same scale as VADER.

For the subsequent analysis, we decided to use VADER as the sentiment analysis method of choice for generating our desired sentiment feature - to be augmented to the recommender system. VADER is specifically designed for social media text and considers sentiment intensity alongside polarity, which aligns well with the informal and emotive nature of user-generated reviews and comments in our dataset [131]. We also found that VADER had a stronger correlation (see 3.9) with the user ratings compared to AFINN and BING - which suggests that VADER is more effective in capturing the sentiment of the review text in our dataset. As such, this concluded our sentiment analysis process. The generated sentiment scores were then stored in a new column in the DataFrame, which we named *sentiment*. This column will be used in the subsequent recommender system development.

In summary, using VADER we were able to generate sentiment scores for each review in our dataset. These scores provide a measure of the sentiment expressed in the review text, which can be used to augment the recommender system or contribute an additional layer of understanding to perhaps enhance the accuracy and relevance of recommendations.

3.4 Data Summary

Having cleaned and preprocessed the review text, as well as generated the user sentiments feature, we have transformed the dataset into a suitable format and laid the groundwork for subsequent analysis. We summarise the dataset that we have cleaned and preprocessed in Table 3.3, including general statistics and numbers.

Metric	
Number of Reviews	83 139
Number of Unique Products	3249
Number of Unique Reviewers	3668
Median Number of Reviews per Reviewer	18.0
Median Number of Reviews per Product	19.0
Average Ratings Overall	4.4
Mean Sentiment Score Overall	0.6
Average Word Count of Reviews	127
Most Words in a Review	5741
Least Words in a Review	1

Table 3.3: Summary Metrics for the Dataset

[9]The Natural Language Toolkit (NLTK) is a popular Python library for natural language processing tasks, including sentiment analysis.

We have a dataset of just 83 139 reviews from 3668 unique users and have 3249 unique products after cleaning the data. This diversity in users and products offers an opportunity to test the robustness of our recommender models and challenges them to generalise well beyond the data the models were trained on. The dataset is 190mb in size, which makes it manageable for the computational resources available locally. The numbers in Table 3.3 also takes into account that we further filtered our dataset to ensure that each user and product had at least 13 reviews, which ensures that we do not encounter any cold-start problems and also offers us space to partition the dataset accordingly for subsequent analysis. Our review ratings are highly skew (more on this in Section 3.5.3), with the majority of reviews having relatively high ratings given. As such, we opted to show the median rating in Table 3.3 to provide a more accurate representation of the central tendency of the ratings. The numbers from Table 3.3 provide a high-level overview of the dataset, which warrant further exploration and analysis to uncover deeper insights and patterns that can inform the development of the recommender system - this will be discussed in the next section.

3.5 Trends and Patterns

In this section we will explore the dataset to identify trends and patterns that may inform the development of the recommender system. Exploring the dataset at an early stage (before modelling and further analyses) helps identify potential issues or patterns that may require attention during the data cleaning and analysis process [149]. We divide this section into several subsections to support better clarity.

3.5.1 User Engagement

We begin by exploring the users within the dataset and their review habits. The distribution of user engagement, measured by the number of reviews per user - shown in Figure 3.1 - provides an initial understanding of user activity. Notably, the Figure highlights that the vast majority of users have fewer than 40 reviews associated with their ID, with the peak frequency exceeding 800 users. As the number of reviews increases, the frequency sharply decreases, indicating that user engagement in terms of review contribution is skewed towards lower values. That is, the distribution is highly right skewed. As explained in Section 3.3, we ensured that each user had at least 13 reviews to alleviate the cold start problem, thus Figure 3.1 shows the minimum number of reviews per user in the dataset, starts at 13. The median number of reviews per user is 18. This observation suggests that while some users are highly engaged, contributing numerous reviews, the bulk of users tend to contribute fewer reviews. Understanding this distribution can help in providing context around the level of data sparseness within the dataset.

Additionally, a temporal analysis, shown in Figure 3.2, illustrates an increasing trend in the number of reviews over time which illustrates an increasing trend in user engagement. Specifically, the line chart in Figure 3.2 displays the trend in the number of reviews per year from 6 to 2018. It is clear that the number of reviews has experienced exponential growth over the years. The review count remains relatively low until around 2010, after which there is a sharp increase, indicating a surge in user engagement on the platform. This trend of exponential growth is a common characteristic of user-generated content on digital platforms, reflecting the rapid adoption and increasing popularity of online shopping and product reviews [7]. In 2017, the review count reaches its peak at just over 20 000, demonstrating the high level of user engagement during this period. However, there is a noticeable dip in the number of reviews in 2018, with the count dropping to just under 5000 reviews. This decline is not indicative of a decrease in user engagement but it is likely due to the dataset only containing reviews up until February 2018. Therefore, the data for 2018 is not complete, which explains the sudden drop.

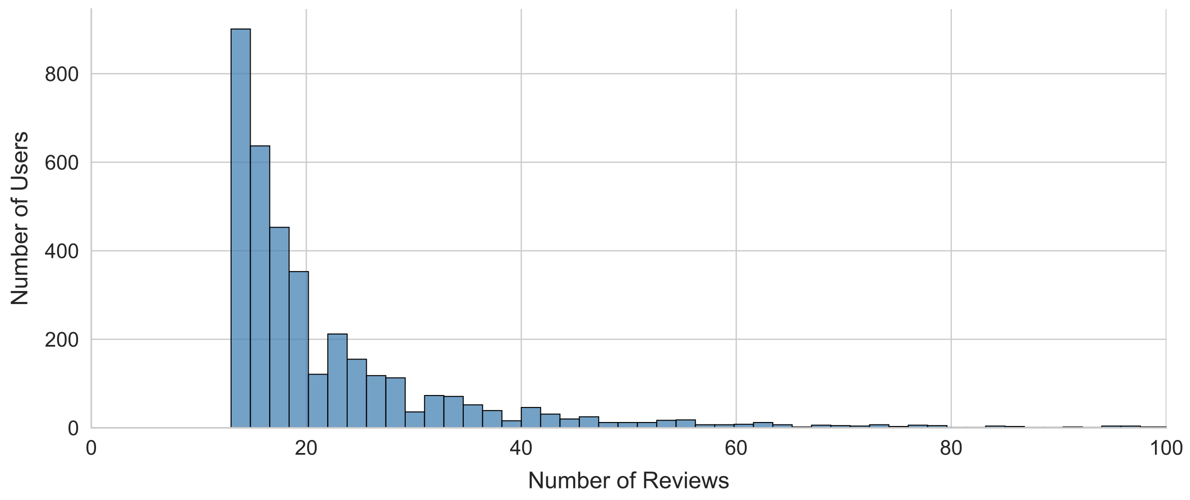


Figure 3.1: Distribution of Reviews per Reviewer.

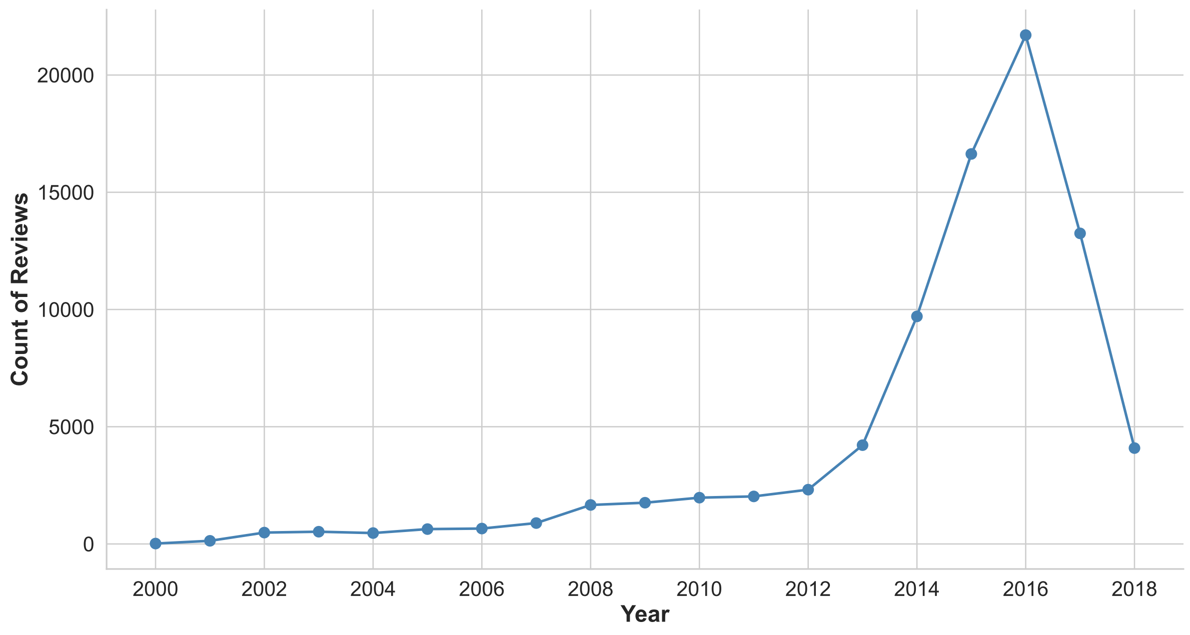


Figure 3.2: Number of Reviews per Year.

These figures have provided valuable insights into the general user engagement, and in extension the temporal dynamics of user engagement in terms of review contribution on Amazon's e-commerce platform. Although the temporal aspect is not considered in this thesis, it is worthwhile showing to provide context on the dataset chosen.

3.5.2 Products

Similar to Figure 3.1, we can look at the number of reviews per product. Analysing this is essential for shedding light on the dynamics and characteristics of the dataset that extend beyond individual user behaviors. Figure 3.3 illustrates the number of reviews per product. From the Figure, it is evident that a significant number of products have received fewer than 40 reviews, with a peak frequency observed for products with around 13 and 20 reviews. This suggests that while some products receive a high volume of reviews, the majority tend to have a relatively lower count.

Importantly, the figure aligns with our data subsetting process of ensuring that each product has at least 13 reviews. Again, this was done to alleviate the cold start problem, ensuring that there is sufficient interaction data for each product to generate reliable recommendations. Similar to Figure 3.1, this product distribution is right skewed and the median number of reviews per product is 19.

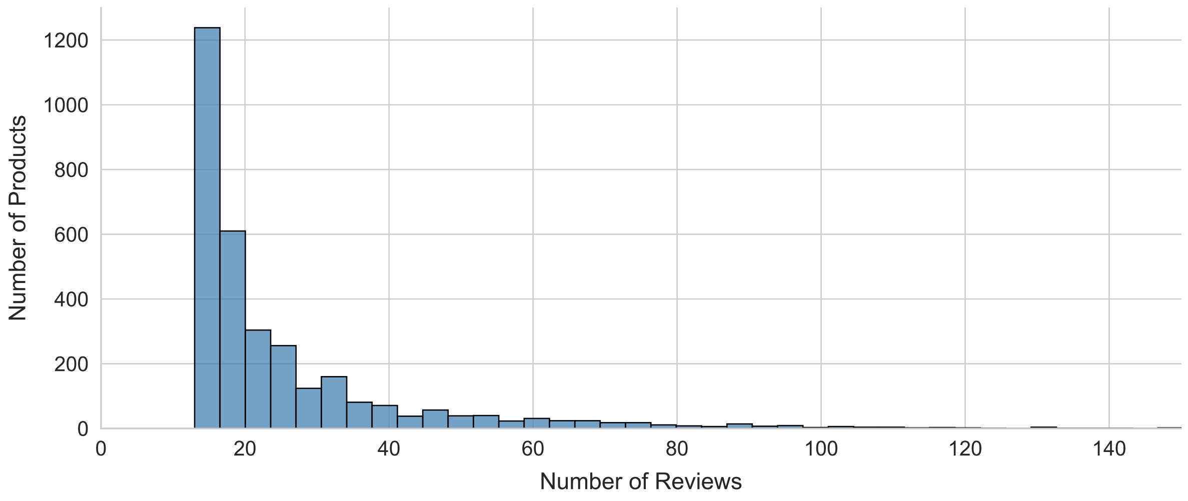


Figure 3.3: Distribution of Reviews per Product.

3.5.3 Ratings

We can also generate an overall picture of the dataset by looking at the distribution of ratings - allowing us to discern the user's general perception toward the products in our dataset. Figure 3.4, illustrates the the spread and central tendency of ratings. As mentioned in Section 3.1, the rating scale is 1 to 5 for this Amazon Product Reviews dataset. From the Figure, it is clear that the majority of users have given a rating of 5, with the count exceeding 50 000 reviews. This suggests a high level of customer satisfaction, indicating that most users had positive experiences with the products they reviewed. On the other end of the spectrum, few users have given ratings of 1 and 2, with counts of 2000 and 3000 reviews respectively. This suggests that negative experiences are relatively rare among the users. Nevertheless, this scarcity could be indicative of two potential scenarios: either negative encounters are genuinely uncommon, or users may not be as prompt to express dissatisfaction compared to their enthusiasm in sharing positive experiences ([23];[133])

Ultimately, understanding the distribution of user ratings is crucial for building and optimising our recommender system as it plays a large role in informing us of the potential user bias. We complement this Figure by looking at the the summary statistics in Table 3.4, for the ratings attribute which again reiterates how a large portion of the ratings are highly rated (5). Furthermore, the mean rating of 4.4 highlights the positive trend in the user feedback. Sentiment analysis scores could perhaps adjust for this bias, as they provide a more nuanced understanding of user preferences and opinions, which can potentially enhance the recommenders ability to better capture the user's preferences and satisfaction with products and ultimately provide more accurate and relevant recommendations [38].

3.5.4 Review Text and Sentiments

Regarding the review, we have spent a considerable amount of time cleaning and preprocessing the this feature. We can now explore it, both from a lens of the textual content and the sentiments - the goal being to visualise and identify any patterns within the user reviews or sentiments.

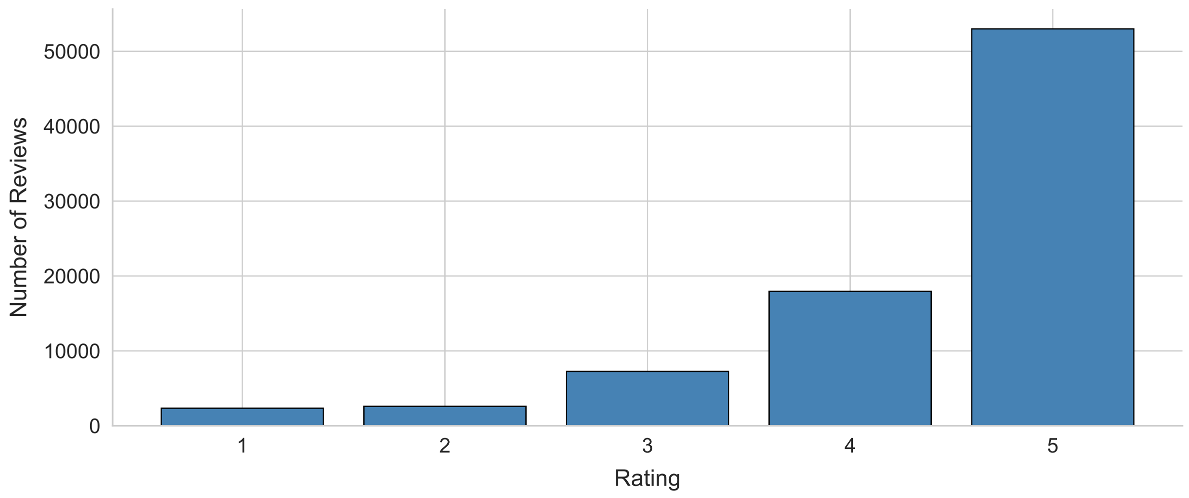


Figure 3.4: Distribution of Ratings from Users.

Statistic	Value
Count	83 139
Mean	4.40
Standard Deviation	0.97
Minimum	1
25% Percentile	4
50% Percentile	5
75% Percentile	5
Maximum	5

Table 3.4: Summary Statistics of Ratings

On average, we found that there are 127 words in each review. To supplement this, we visualise the distribution of the word count in Figure 3.5. From the Figure, it is clear that a majority of the reviews are relatively short, with most falling under 200 words - with large proportion of reviews containing below 100 words. This suggests that users typically prefer to leave concise feedback on products. Since we are looking at incorporating these reviews into NCF models, this could have several implications. Given that shorter reviews are more frequent, they might have a more significant impact on the prediction outcomes. However, shorter reviews might not always provide comprehensive insights into a user's preferences or experiences, as they might lack detailed descriptions or explanations. On the other hand, longer reviews, although less frequent, might offer more detailed insights into a user's preferences, potentially enhancing the accuracy of the recommendations. However, they might also introduce noise into the model due to the increased likelihood of off-topic content or unnecessary details. A potential outcome of this is that the model might be biased towards shorter reviews, as they are more frequent [136]. This shall be explored further in the Chapter 5 - the results.

Using the review text we can also explore the prevalence of specific phrases or consecutive words, commonly known as *n*-grams, where *n* is the number of words [94]. In Figure 3.6, we specifically examine the top 10 frequently occurring consecutive tri-grams (Figure 3.6a) and four-grams (Figure 3.6b). This analysis provides insights into the distinctive linguistic structures that frequently appear in the reviews, improving our understanding of the language dynamics within the dataset.

From the Figure 3.6, it is clear that phrases like "metal gear solid", "grand theft auto", and "super mario bros" are among the most frequent tri-grams. These tri-grams appear to be related to popular video game titles, suggesting that our dataset includes a significant amount of reviews for these games or

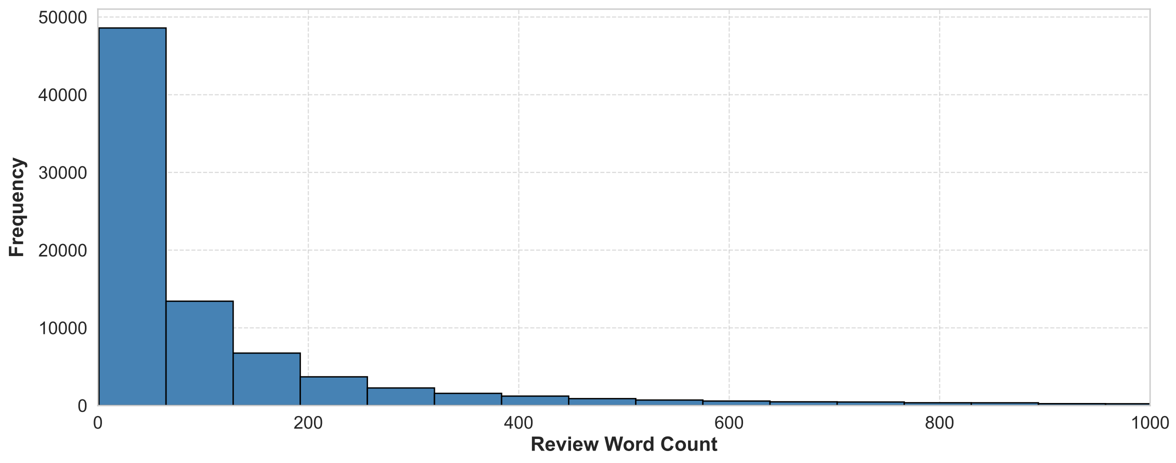
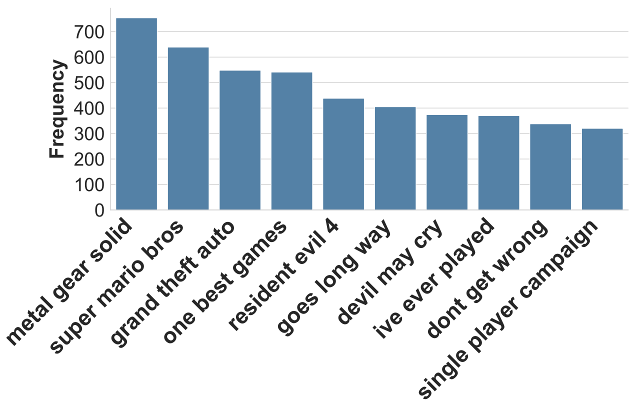
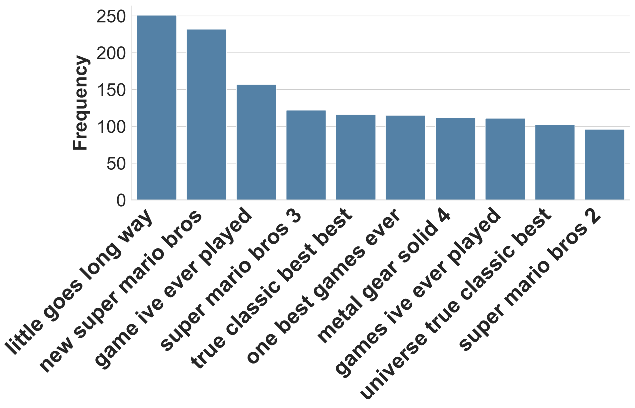


Figure 3.5: Distribution of Review Word Count.



(a) Tri-gram



(b) Four-gram

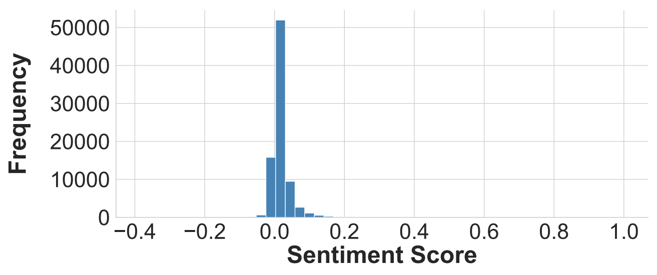
Figure 3.6: Top 10 N-grams in Review Text from Users

rather, users frequently mention these games in their reviews. This could be indicative of the popularity of these games among users. Other tri-grams like “one best games”, “can’t go wrong”, and “goes long way” could be common phrases users employ to express their positive experiences or high regard for a product. We find similar results when we look at the four-grams (Figure 3.6b) where some of the phrases are “little goes long way”, “now super mario bros”, and “one best games ever” are among the most frequently used.

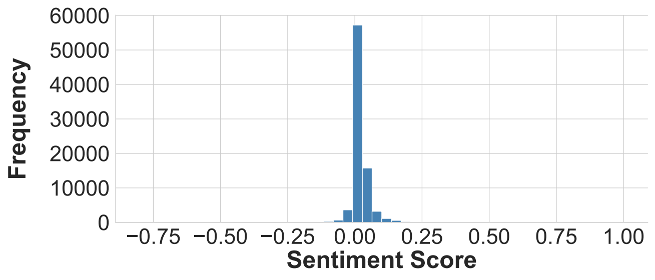
These N-grams are useful in text analysis as they provide context that individual words might not. By considering sequences of words (like tri-grams and four-grams), we can capture more meaningful phrases that often occur in our text data. This can help in understanding common themes or topics in our reviews [33]. Furthermore, in the context of our recommender system, these n-grams can provide valuable insights into the aspects that users frequently discuss or value in products [136]. However, it’s important to note that while n-grams can capture local context (i.e., within the n-gram), they might not capture the overall sentiment of a review if it’s expressed through complex or long-range dependencies between words. Techniques like sentiment analysis, which we employed, are more effective in capturing the overall sentiment of a review, as they consider the entire review text [33].

To that end, we can also explore the distribution of the sentiment scores from from each of the three sentiment analysis lexicon-based methods in Figure 3.8. From Figure 3.8, it is clear that the distribution of sentiment scores from AFINN (Figure 3.7a) and BING (Figure 3.7b) are similar, while the distribution of sentiment scores from VADER (Figure 3.7c) is distinct. For AFINN and BING, their distributions are uni-modal and skewed to the right, with most sentiment scores concentrated around 0. This suggests that these sentiment analysis techniques tends to classify a significant portion of the reviews as neutral. In contrast to Figures 3.7b and 3.7a, the sentiment score distribution for VADER (Figure 3.7c) illustrates sentiment scores not uniformly distributed. Most of the sentiment scores are clustered around 0 and 1, indicating that most reviews are either neutral or positive. There is a significant spike in frequency close to the 1.00 score, suggesting a large volume of highly positive sentiments in the reviews. Given that we have spikes at 0, 1 scores, as well as some reviews classified negatively (as shown in Figure 3.7c), we find that the distribution of VADER sentiments illustrates that this technique is capturing both the positive, neutral, and negative sentiments in our review data, albeit with a higher frequency of positive sentiments. We expect this behaviour since the distribution of ratings (as shown in Figure 3.4) is heavily skewed towards the higher ratings.

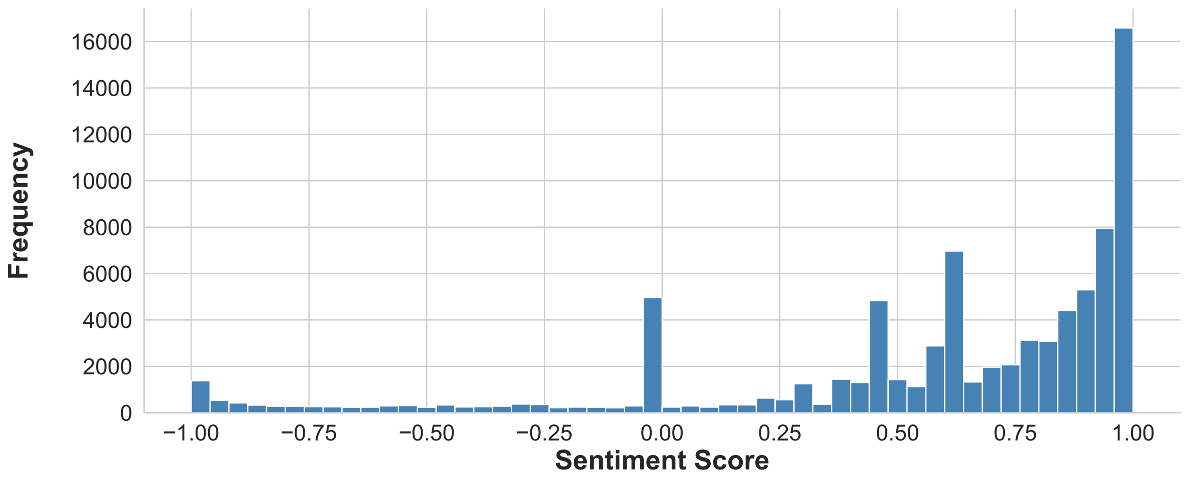
As we mentioned in Section 3.3.3 we opted for VADER as it held the strongest correlation with the user ratings. This decision seems well-supported as VADER, unlike AFINN and BING, captures a wider range of sentiments (as shown from Figure 3.7c), which could lead to a more nuanced understanding of user opinions. This can be particularly useful for our recommender system as it can help in capturing more detailed user preferences, potentially enhancing the accuracy of our recommendations [38].



(a) AFINN



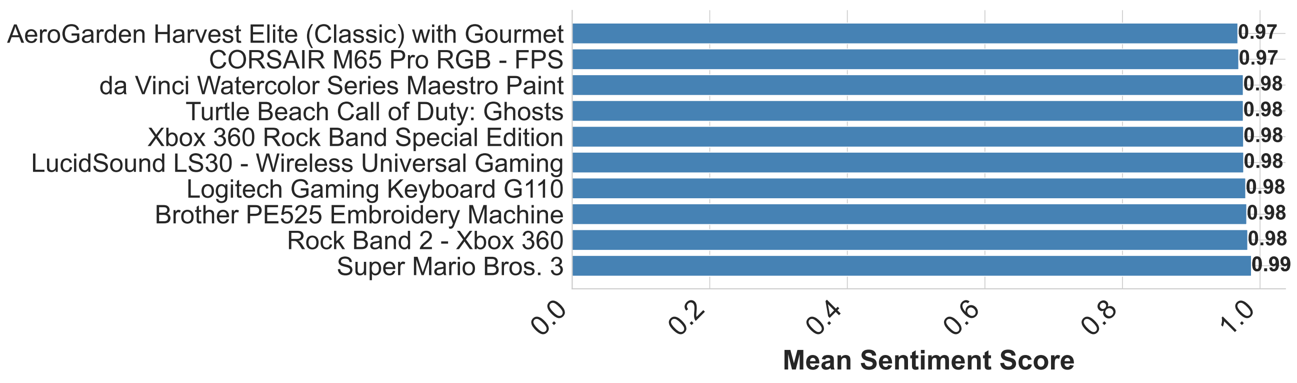
(b) BING



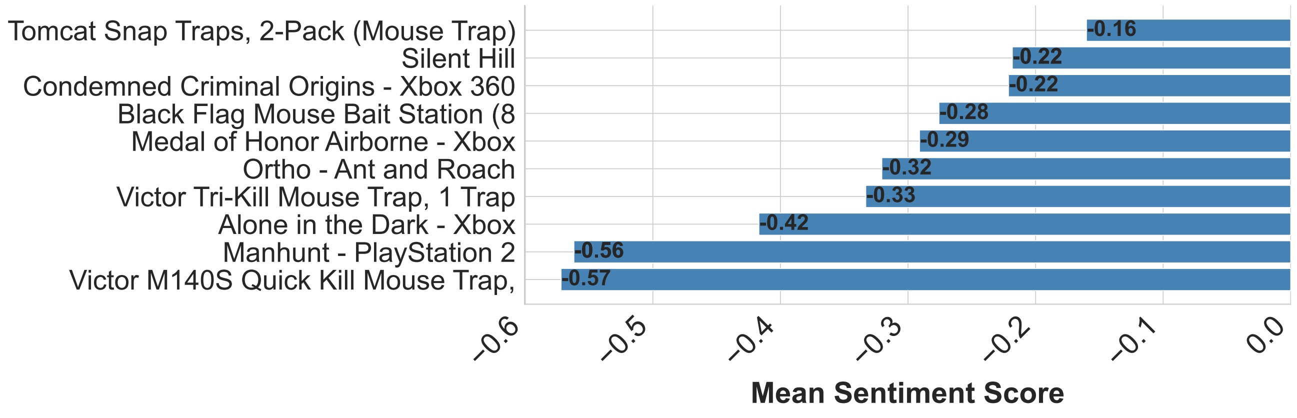
(c) VADER

Figure 3.7: Distribution of Sentiment Scores for AFINN, BING, and VADER algorithms.

In Figures 3.8a and 3.8b, we identify the the top 10 products and bottom 10 products based on their average sentiment scores. We see that the top products have similar mean sentiment scores. In contrast, there is a clearer difference between the bottom 10 products. Additionally, Tables 3.5 and 3.6 illustrates that the ratings for the top 10 and bottom 10 products (by sentiment) line up quite well. This is indicative of a consistent correlation between product (VADER) sentiment and user ratings, suggesting that products with higher sentiment scores tend to receive more positive ratings, while those with lower sentiment scores align with lower ratings.



(a) Top 10 Products Sentiments



(b) Bottom 10 Products Sentiments

Figure 3.8: Comparison of Sentiments for Top and Bottom 10 Products.

The products with high sentiments (see Figure 3.8a) have received high positive feedback from users and furthermore, from Table 3.5 we see that these products also have at least a mean rating of 4 (and more often than not closer to 5). This further reinforces the positive sentiment towards these products, indicating that they are well-received by users in terms of both ratings and review sentiments. We can say the same for the for bottom 10 products shown in Figure 3.8b and Table 3.8b - these low mean sentiments achieved by these products reviews are also reflected in their mean ratings being mostly below 4.

To end off this visual analysis, Figure 3.9 is a correlation heat map displaying the relationship between the various sentiment scores generated from AFINN, BING and VADER as well as the ratings and review word count. The goal here is to illustrate whether there is a correlation between any of these sentiment scores with the ratings, and whether review length has a correlation with sentiments or ratings. From Figure 3.9, it is clear that VADER has the highest correlation with rating (0.32), suggesting a moderate positive relationship. This indicates that as the VADER sentiment score increases, the rating tends to increase as well. This is an important observation and was pivotal to our decision to use VADER as the sentiment analysis method of choice to generate sentiments to be used in our model. Specifically, AFINN and BING have lower correlations with rating (0.09 and 0.13 respectively), indicating weaker relationships with the rating. Word count has a negative correlation with rating (-0.13), suggesting that longer reviews tend to have lower ratings - albeit the correlation is weak. However, it has positive correlations with all three sentiment analysis techniques, with the highest being with AFINN (0.59). However, this is also an expected result, as AFINN (and BING) assign sentiment scores to each word in a review, and the cumulative score is influenced by the number of words [52]. Consequently, reviews with more words have a greater chance of containing positive words, resulting in higher sentiment scores.

Ultimately, this section has provided us with a holistic understanding of the dataset's inherent structures and offers context-rich insights that will guide subsequent modelling, interpretation and discussions.

Product Title	Mean Rating	Count of Reviews	Mean Sentiment
Super Mario Bros. 3	4.733	15	0.987
Super Mario Bros. 3	4.733	15	0.987
Rock Band 2 - Xbox 360	4.6	15	0.982
Brother PE525 Embroidery Machine	4.692	13	0.981
Logitech Gaming Keyboard G110	4	13	0.979
LucidSound LS30 - Wireless Universal Gaming	4.75	16	0.976
Xbox 360 Rock Band Special Edition	4.267	15	0.976
Turtle Beach Call of Duty: Ghosts	4.714	14	0.976
da Vinci Watercolor Series Maestro Paint	4.947	19	0.976
CORSAIR M65 Pro RGB - FPS	4.8	15	0.969
AeroGarden Harvest Elite (Classic) with Gourmet	4.75	16	0.968

Table 3.5: Details about the Top 10 Products by Average Sentiment.

Product Title	Mean Rating	Count of Reviews	Mean Sentiment
Tomcat Snap Traps, 2-Pack (Mouse Trap)	2.062	16	-0.16
Silent Hill	4.56	25	-0.218
Silent Hill	4.56	25	-0.218
Condemned Criminal Origins - Xbox 360	4.115	26	-0.221
Black Flag Mouse Bait Station (8	4.286	14	-0.276
Medal of Honor Airborne - Xbox	3	13	-0.291
Ortho - Ant and Roach	4.357	14	-0.321
Victor Tri-Kill Mouse Trap, 1 Trap	3.615	13	-0.333
Alone in the Dark - Xbox	2.316	19	-0.417
Manhunt - PlayStation 2	3.895	19	-0.562
Manhunt - PlayStation 2	3.895	19	-0.562
Victor M140S Quick Kill Mouse Trap,	4.333	15	-0.572

Table 3.6: Details about the Bottom 10 Products by Average Sentiment

However, before we detail the next steps, it is important to establish what data is used to train our models and what data is used to validate our models - this is studied in the next section.

3.6 Data Partitioning

When analysing a recommendation model, we are more concerned with its future performance on new (unseen) data, rather than its performance on past data [149]. In order to test future performance and estimate the prediction error, we must properly partition the original dataset into training and test subsets. This section outlines the data partitioning strategy employed for this thesis, considering the cleaned and transformed reviews dataset consisting of over 83 139 reviews.

Training data refers to the data that is used by one or more learning methods to develop the model and the test data is used to evaluate the quality of the model by giving the trained model this unseen (test) data [149]. The test set must be different and independent from the training set in order to obtain a reliable estimate of the true error [149]. It is important that performance is estimated on data which take no part in the formulation of the model. Some learning schemes, such as our case, need also a validation set in order to optimise the model parameters. In summary, to evaluate the performance of our recommender models, we need to train, validate (to optimise) and test our recommender model using different data. To do so, is not as simple as randomly allocating reviews (records) into three distinct sets.

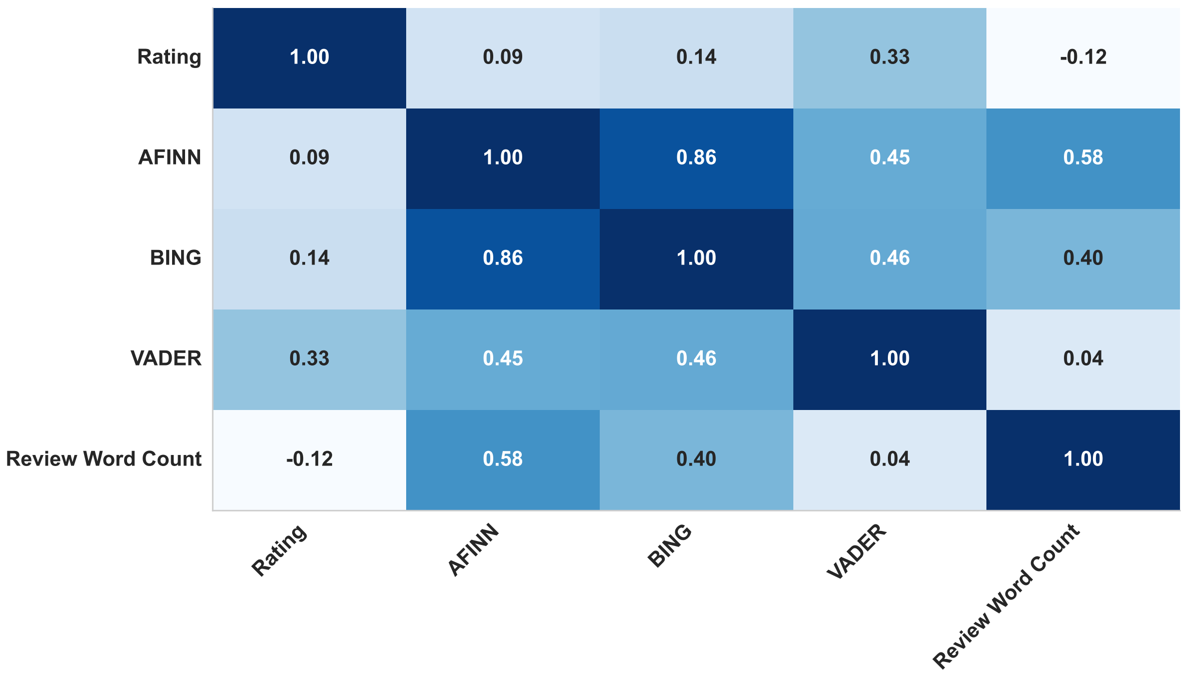


Figure 3.9: Relationship Between Lexicon-Based Methods and Ratings.

For recommender systems, the partitioning is performed by randomly selecting some ratings from all (or some of) the users. That is, for each user, we randomly select some ratings they have given and hide them, whilst keeping their remaining ratings in tact and to be used for training. The selected ratings constitute the test set, while the remaining ones are the training set. This method is called leave-k-out[10]. In [120], they split the dataset into 80% training and 20% test data following this leave-k-out approach. In [59] the test set is made by 10% of users: here only 5 ratings for each user in the test set are withheld. This leave-k-out approach is in fact also the method that we employ in our thesis.

We opted to hide 3 random ratings from each user and allocate these to the test set, and hide another 2 (again random) ratings which were allocated to the validation set. We allocated the remaining reviews for the user to the training set. To further ensure the integrity of the partitioning process, we also ensured that the reviews for each user were randomly shuffled before allocating them to the respective subsets. This randomisation helps to mitigate any potential biases that may be present in the original dataset and ensures a representative distribution of user-item interactions across the subsets. We used a seed of 2207 to ensure reproducibility of the results. Note, there are other methods of splitting the dataset such as k-fold cross-validation[11], however for our research we opted to use leave-k-out due to its simplicity and popularity in recommender research.

The choice for the partition was made to ensure that the recommender system is trained on a large enough dataset to capture the underlying patterns and relationships between users and items, whilst also ensuring that the test set is large enough to provide a reliable estimate of the model's performance. To that end, the training data, the largest partition of the data containing at least 7 reviews per user, comprises approximately 77.9% of the dataset, or 64 799 reviews. The validation set comprises of approximately 8.8% percent of the dataset (7 336 reviews). Finally, the test set is represented by approximately 13.2% of the dataset (11 004 reviews). These datasets are summarised in Table 3.7.

[10]method involves selecting a subset of k ratings from the dataset as the test set, while the remaining ratings form the training set

[11]k-fold cross-validation divides the dataset into k equally sized folds. The model is trained k times, each time using $k-1$ folds for training and one fold for validation. The performance is then averaged over the k iterations.

Dataset	Purpose	Number of Reviews	Proportion of Reviews
Training	Training the model	64 799	77.9%
Validation	Model validation	7 336	8.8%
Testing	Model evaluation	11 004	13.2%

Table 3.7: Dataset Partition

The partitioning of the dataset into training, validation, and test sets is crucial for achieving reliable and accurate results. It helps to evaluate the recommender system's performance on unseen data and provides an estimate of its effectiveness in real-world scenarios. The training set facilitates the learning of user preferences, the validation set aids in fine-tuning the model, and the test set provides an independent assessment of the system's performance. Having established this partitioning process, our data is ready to be used for our analysis.

3.7 Conclusion

In this chapter we have discussed a variety of aspects of the Amazon product review dataset, ranging from data collection and preparation to exploratory data analysis and data partitioning for recommender systems.

Specifically, in Section 3.1, we provided some background and context on the Amazon Product Reviews dataset, which we have chosen for this thesis. In this same section we also dived into the details of the type of user feedback that we have in the dataset and explained that we are using explicit feedback provided through numerical ratings and review text variables. In section 3.2, we provided a brief overview of the variables in the dataset and gave some descriptions of these.

The data that consists of over 75 million reviews over a 22-year period. Through data management and filtering (see Section 3.3), we were able to extract a subset of the data that contains 91 235 reviews from 3801 unique users and 3283 unique products. In this same section, we also explained in detail the preprocessing steps taken to ready the data for further analysis, which included handling of nulls and duplicates amongst other steps. Subsection 3.3.2 and 3.3.3 summarised the work done to analyse and clean the textual data - the user review text. These sections outlined the text cleaning steps taken from normalising to lemmatising the review text. We also covered what sentiment analysis is (see Section 3.3.3) and why we opted for the VADER lexicon-based approach to generate the sentiment feature for the dataset. At this stage we had filtered our dataset to 83 139 reviews in the dataset, where we had 3668 unique users and 3249 unique products

Sections 3.4 and 3.5 provided a brief high level visual summary of the dataset from different angles, looking at user engagement, review text and ratings data. We looked at the distributions and identified any trends in the data. Over the entire cleaned data set of reviews (approximately 83 139 reviews), the average ratings are 4.4 - where the ratings are on a five-point scale. We also identified that the general sentiment of users vary, however they are majority positive. Lastly, we outlined the method of partition of our dataset in Section 3.6. Data partitioning is a pivotal step in the development and evaluation of recommender systems, serving to gauge the system's ability to generalise to unseen user-item interactions. We opted for the leave-k-out approach where we simply, hid or removed 5 ratings at random from each user and allocated 2 of those hidden ratings to a validation set and 3 of them to a testing set, the balance of the ratings were used as the training set.

As we transition to the next chapter on methodology, we build upon this foundational understanding of the dataset to look into the intricacies of our approach for building and evaluating recommender systems.

Chapter 4

Methodology

Having established context for the dataset chosen for our analysis, Chapter 4 looks into the methodology adopted in this thesis. We begin this chapter by outlining the overall modelling approach (Section 4.1) that provides a high-level view of our analysis, encompassing the general idea and process in formulating the individual recommendation model. After this overview, we look in detail at the NCF algorithm (Section 4.3 - a recently implemented approach that leverages deep learning algorithms for collaborative filtering. This section will cover the background of NCF, the algorithm formulation, the neural network architecture, model training process as well as how we incorporated the review text and sentiments into the models. After this, in Section 4.4, we clearly establish and describe the benchmark models used for comparison with the NCF models. The three benchmark models are discussed, and their underlying algorithms briefly explained. Additionally, Section 4.5 addresses the evaluation criteria that will be used to assess the performance of our recommender models. Details of the software and hardware used in the implementation of the models are discussed in Section 4.6. The chapter concludes with Section 4.7 giving a summary of details discussed in this chapter.

4.1 Modelling Approach

One can view the recommendation problem as a prediction problem. Within this framework, a pool of users interact with a diverse range of items, presenting the challenge of predicting a user's rating for an unseen item. The predictive algorithm is simply some tool that uses the users' historical purchases (or reviews in our case) to learn the users' preferences to be able to predict *liking* for future items. To augment the predictive algorithm, we propose supplementing the algorithm with additional information, namely review text and sentiment. The conjecture is that this augmentation could yield more accurate predictions and provide users with recommendations that resonate with their preferences more closely. To effectively explore this hypothesis, we devise a comprehensive approach that balances the workings of collaborative filtering, neural network architectures, and the incorporation of additional information. The details of which shall be described in this section.

4.1.1 Formulating the Problem

For recommender systems, the problem space can be formulated as a matrix of users versus items, with each cell representing a user's rating on a specific item [68]. That is, the users are represented as rows while the products are represented as columns, each cell value points to the ratings given to the product by a particular user. Under this formulation, the problem is to predict the values for specific empty cells (i.e. predict a user's rating for an item they have not rated). We acknowledge the sparsity of this matrix, a characteristic often present in e-commerce data, as discussed in Section 2.7.1. Given the vast array of items available, each user interacts with only a fraction of the total amount of products. A depiction of a user-item matrix is shown in Table 4.1. The blank spaces indicates the user has not reviewed (or purchased) the item before.

Formally, we denote the user-item interaction matrix as $\mathbf{R}$, where r_{ij} represents the rating user i gives to item j. Furthermore, we introduce the predicted interaction matrix, denoted as $\hat{\mathbf{R}}$, which will contain our model's estimations of the user-item interactions. Other notations will be introduced throughout this chapter where required. Our recommender system paradigm falls within collaborative filtering, which (as

	Item 1	Item 2	Item 3	Item 4	$\cdots$	Item K
User 1	3	2			$\cdots$	
User 2	4			4	$\cdots$	5
User 3	5	5			$\cdots$	5
User 4	5		2		$\cdots$	
User 5		5			$\cdots$	3
User 6	5	4		5	$\cdots$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\cdots$	$\vdots$
User N	4		5		$\cdots$	4

Table 4.1: User-Item Matrix

mentioned in Section 2.2) leverages the collective wisdom embedded in user-item interactions to discern inherent patterns or preferences, enabling the system to make predictions for unobserved user-item pairs [75]. Effectively, a collaborative filtering system will navigate $\mathbf{R}$, the user-item matrix, aiming to uncover latent relationships and preferences by assessing how users with similar tastes have rated similar items. The intuition is that users who have historically exhibited analogous preferences are likely to align in their ratings for items they have not yet interacted with [75]. The goal is to completely populate the empty user-item pairs in $\mathbf{R}$ to generate $\hat{\mathbf{R}}$.

Fundamentally, this is the problem we aim to solve. We seek to build several models (using different algorithms) that can accurately predict the ratings for unseen user-item pairs and then compare their performance. The manner in which these models learn to estimate these ratings follows the collaborative filtering approach. One such model is a NCF recommender system, which use neural networks to directly learn representations of users and items from the user-item interaction matrix $\mathbf{R}$, and then uses these representations to predict the ratings for unseen user-item pairs [56]. The next subsection looks at how we solve the prediction problem specifically using this model.

4.1.2 Implementation and Approach

Initially, our focus is on constructing a neural collaborative filtering (NCF) model using only the available ratings. Subsequently, we progress to integrating review text and sentiments into our model sequentially. In this initial phase (using ratings only) we initiate our modelling process by transforming the cleaned data into a user-item ratings matrix. Mathematically, this translates to defining U as the set of users, I as the set of items, and $\mathbf{R}$ as the user-item ratings matrix $\mathbf{R} \in \mathbb{R}^{|U| \times |I|}$. The subsequent steps in constructing the various recommender models in this thesis, including the neural collaborative filtering and benchmark models, follow identical procedures:

1. partition the user-item matrix
2. train an algorithm using training set
3. optimise an algorithm using validation sets (if required)
4. evaluate an algorithm using testing set

Thus, having converted the cleaned data into a user-item ratings matrix, we proceed to partition the matrix following the methods detailed in Section 3.6. The partitioning strategy involves randomly selecting 5 ratings from each user and assigning them to either the validation or testing set. Specifically, the validation set comprises of 7 336 ratings, and the testing set consists of 11 004 ratings. This ensures that our models are trained on a minimum of 7 ratings per user (training data is approximately 64 799 reviews).

With the data appropriately partitioned, we are now poised to integrate the recommender algorithm for training. The construction of the recommenders were all done from fundamentals, using Python and the TensorFlow[1] library. Each recommender was trained using the training data. We used the validation data to optimise the hyperparameters[2] of the recommender models. The testing data was used to evaluate the performance of the models. The models are assessed based on their predictive accuracy (how accurately they can predict unrated items) and their ability to recommend top-*n* items (how relevant the recommendations are) to users. The specific evaluation metrics used to assess the performance of the models are discussed in Section 4.5.

This process of training, optimising (for NCF), and evaluating is iteratively repeated for scenarios (the NCF models) where we incorporate review text and, subsequently, review sentiment. It is imperative to conduct these steps separately and then in combination to discern the impact of additional information (features) on the model's performance. This approach allows us to assess whether the inclusion of review text and sentiment enhances the predictive capabilities of our recommender system. As such, we will have three different NCF models - more on this in the next section.

The workings and approach described in this section is summarised by the flow chart in Figure 4.1. As we embark on the development of the NCF model, the next section will provide some background into neural networks, providing a high-level overview of how they work. Subsequently we will look at NCF, exploring its architecture, training process, and how it operates within the general collaborative filtering approach.

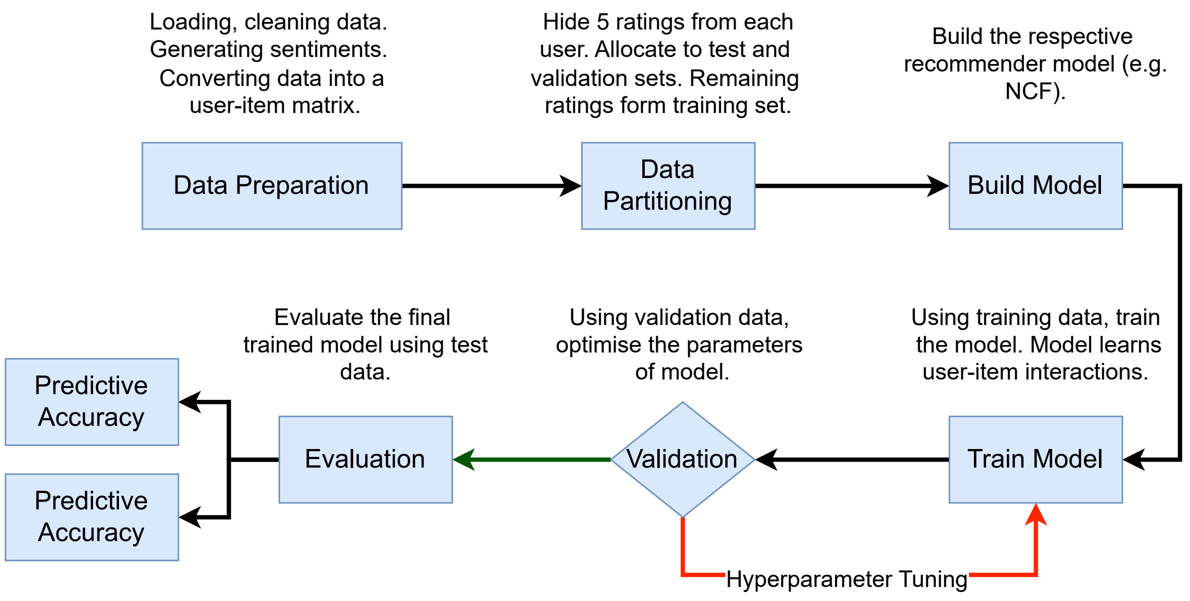

Loading, cleaning data. Generating sentiments. Converting data into a user-item matrix.
Hide 5 ratings from each user. Allocate to test and validation sets. Remaining ratings form training set.
Build the respective recommender model (e.g. NCF).
Data Preparation
Data Partitioning
Build Model
Evaluate the final trained model using test data.
Using validation data, optimise the parameters of model.
Using training data, train the model. Model learns user-item interactions.
Predictive Accuracy
Predictive Accuracy
Evaluation
Validation
Train Model
Hyperparameter Tuning

Figure 4.1: Flowchart outlining the general process followed in applying and evaluating the recommender models.

4.2 Neural Networks

Neural networks have become a cornerstone of modern artificial intelligence and machine learning [1]. They have evolved from over the past two decades to complex deep learning architectures that are capable of learning complex patterns and making predictions from data [51]. At its core, a neural network consists of interconnected nodes, or neurons, organised into layers [1]. Figure 4.2 provides a visual representation of a neural network, showing an example of an input layer, hidden layers, and an output layer.

[1]TensorFlow is an open-source machine learning framework developed by Google. It is widely used for building and training machine learning models.

[2]A hyperparameter is a parameter whose value is set before the learning process begins. Unlike model parameters, which are learned during training, hyperparameters are typically tuned.

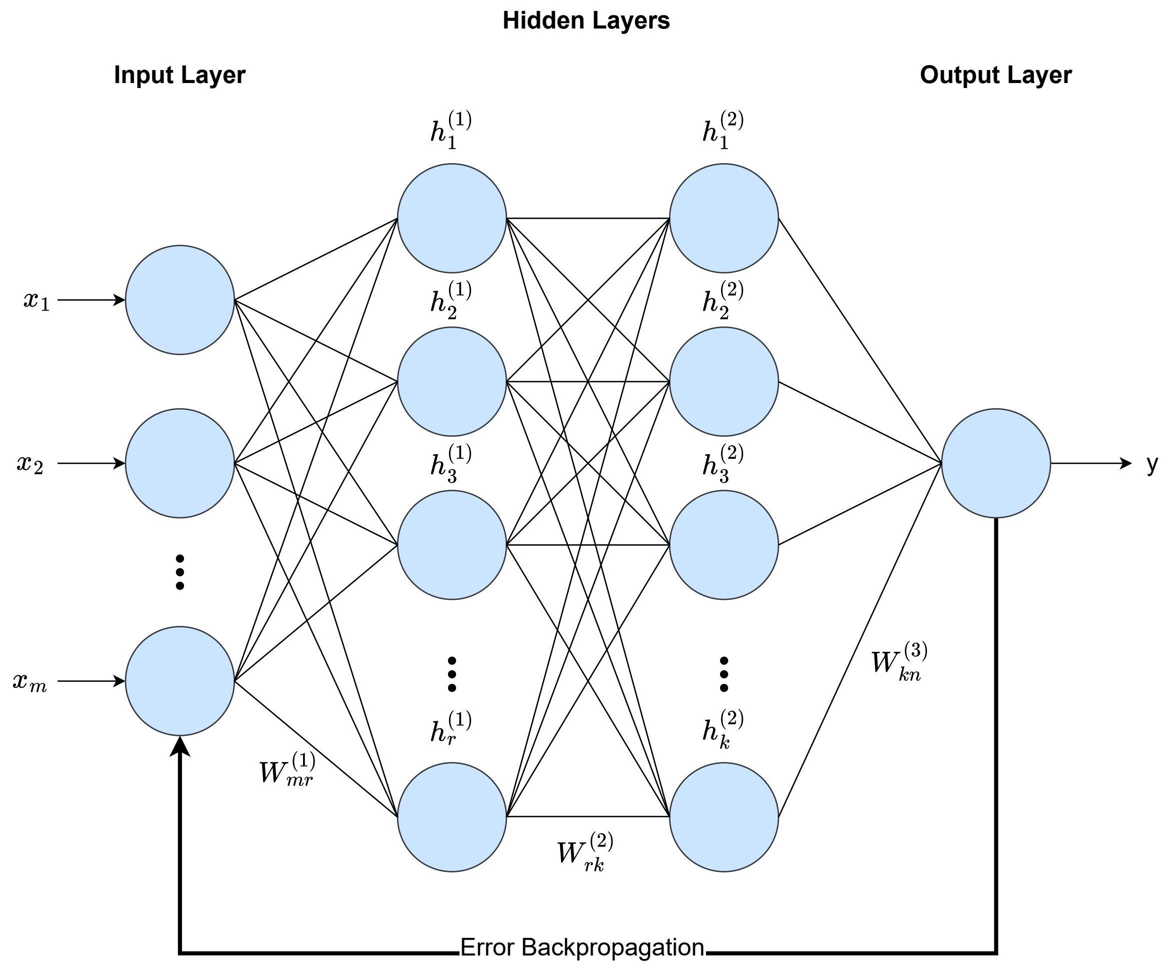


Figure 4.2: Neural Network Architecture showing an example of an input layer, hidden layers, and an output layer.

Neural networks, as mentioned, are composed of interconnected nodes (called neurons) that work together to process complex data inputs and produce an output [51]. The nodes are organised into layers, with each layer performing a specific function. The input layer receives the data (x), the hidden layers process the data, and the output layer produces the result (y). The connections between the nodes are weighted ($\mathbf{W}$), and these weights are adjusted during the training process to improve the accuracy of the model [1].

Specifically, the training process involves feeding the network with data and adjusting the weights to minimise a loss function, which generally measures the difference between the predicted output and the actual output [1]. This is done using an optimisation algorithm to find the optimal weights that minimise the error. The weights are adjusted iteratively until the model produces the desired or optimal output. The process of adjusting the weights is known as backpropagation, and it is a fundamental concept in training neural networks. The backpropagation algorithm calculates the gradient of the loss function with respect to the weights and uses this information to update the weights. This process is repeated until the model converges to an optimal set of weights that minimises the error (i.e., the model learns the patterns in the data). Effectively, backpropagation entails propagating the error backwards through the network to adjust the weights and improve the model's performance [1].

In neural networks, the weights are the parameters that the model learns during training [51]. Neural networks also have many hyperparameters that need to be set before training, such as the number of layers, the number of neurons in each layer, the activation functions, and the learning rate. These hyperparameters can significantly affect the performance of the model and need to be tuned carefully to achieve the best results [51]. Table 4.3 provides a summary and description of common hyperparameters that need

Hyperparameter	Description	Example Values
Learning Rate	Step size for updating weights	0.001, 0.01, 0.1
Number of Layers	Number of layers in the network	1, 2, 3
Number of Neurons	Number of neurons in each layer	32, 64, 128
Activation Function	Function applied to the output of a neuron	ReLU, Sigmoid, Tanh
Batch Size	Number of samples processed at once	32, 64, 128
Optimiser	Algorithm used to update weights	Adam, SGD, RMSprop
Loss Function	Function used to measure the error	MSE, MAE, Cross-Entropy
Epochs	Number of times the model sees the data	10, 50, 100

Table 4.2: Hyperparameters for training a neural network

to be set when training a neural network. To select the optimal combination of hyperparameters, a grid search can be performed [51]. A grid search involves training the model with different combinations of hyperparameters and selecting the one that produces the best results (i.e., the lowest error) [51].

Each hyperparameter plays a crucial role in the performance of the model. The learning rate determines how quickly the model learns the patterns in the data. A high learning rate can cause the model to converge too quickly and miss important patterns, while a low learning rate can cause the model to converge too slowly and take longer to learn the patterns. The number of layers and neurons in each layer also affect the model's performance. A deeper network with more layers can learn more complex patterns, but it can also be more prone to overfitting [155]. The activation function is another important hyperparameter that determines how the output of a neuron is transformed. Common activation functions include ReLU, Sigmoid, and Tanh. The batch size is the number of samples processed at once during training. A larger batch size can speed up training but can also require more memory [44]. The optimiser is the algorithm used to update the weights during training [155]. Common optimisers include Adam, SGD, and RMSprop. The loss function is used to measure the error between the predicted output and the actual output. Common loss functions include Mean Squared Error (MSE), Mean Absolute Error (MAE), and Cross-Entropy. The number of epochs is the number of times the model sees the data during training [44]. Specifically, one epoch is one pass through the entire dataset. Training for more epochs can improve the model's performance, but it can also lead to overfitting [51].

Neural networks have been widely used in various fields, including computer vision, natural language processing, and recommendation systems. They have been shown to be effective in learning complex patterns and relationships in data, making them suitable for tasks that involve high-dimensional data and non-linear relationships. In the context of recommendation systems, neural networks have been used to learn user and item representations from user-item interactions and make predictions about user preferences - as mentioned in Section 2.3. The next section will focus on the specifics of the neural collaborative filtering algorithm, which is a recommendation model that leverages neural networks to learn user and item representations and make predictions about user preferences.

4.3 Neural Collaborative Filtering

In this section, we focus on the details of NCF, providing first some high-level workings of the algorithm (Section 4.3.1), then a comprehensive overview encompassing the algorithm's formulation (Section 4.3.2), the details of the training procedure (Section 4.3.5) and model optimisation (Section 4.3.6). Following this, we discus the method in which the review text and sentiments are incorporated into the NCF model in Section 4.3.4, training and tuning results (Section 4.3.7) and finally of the final models (Section 4.3.8).

4.3.1 Background

The advent of deep learning has marked a paradigm shift in various domains, demonstrating its prowess in extracting intricate patterns and representations from complex data [139]. While deep learning had been explored in recommender systems before, the paper by [56] in 2017 holds significance as it delineates a generalised approach for integrating deep learning into recommender systems, specifically introducing Neural Collaborative Filtering (NCF). Serving as a guiding reference for this thesis, the work provides insights that aid in constructing, optimising, and making specification choices for our very own NCF model.

Traditional collaborative filtering methods, while effective, often struggle with capturing intricate user-item relationships [139]. Deep learning has the ability to model intricate relationships in high-dimensional spaces, presenting a compelling solution to this problem [56]. NCF, as a deep learning-based recommender system, leverages neural networks to directly learn user and item representations from the user-item interaction matrix, enabling it to make predictions for unseen user-item pairs [56]. The neural network architecture within NCF is designed to capture complex interactions, offering a flexible and possibly more accurate recommendation mechanism compared to traditional collaborative-based counterparts [56].

4.3.2 Algorithm Formulation

An initial high-level overview of the algorithm is portrayed by Figure 4.3, which we will make reference to often in this subsection as we build up and discuss the workings of the NCF algorithm.

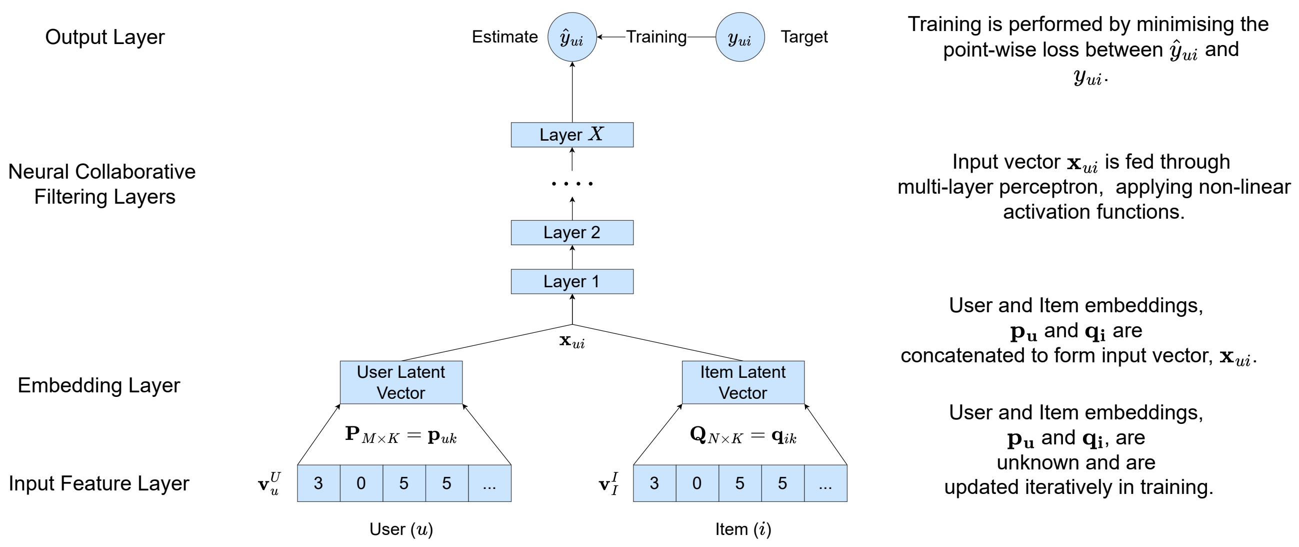

Output Layer
Estimate
$\hat{y}_{ui}$
Training
y_{ui}
Target
Training is performed by minimising the point-wise loss between $\hat{y}_{ui}$ and y_{ui}.
Neural Collaborative Filtering Layers
Layer X
Layer 2
Layer 1
Input vector $\mathbf{x}_{ui}$ is fed through multi-layer perceptron, applying non-linear activation functions.
$\mathbf{x}_{ui}$
User and Item embeddings, $\mathbf{p_u}$ and $\mathbf{q_i}$ are concatenated to form input vector, $\mathbf{x}_{ui}$.
Embedding Layer
User Latent Vector
Item Latent Vector
$\mathbf{P}_{M\times K} = \mathbf{p}_{uk}$
$\mathbf{Q}_{N\times K} = \mathbf{q}_{ik}$
User and Item embeddings, $\mathbf{p_u}$ and $\mathbf{q_i}$, are unknown and are updated iteratively in training.
Input Feature Layer
$\mathbf{v}_u^U$
3 0 5 5 ...
$\mathbf{v}_I^I$
3 0 5 5 ...
User (u)
Item (i)

Figure 4.3: Neural Collaborative Filtering Algorithm.

Consider Figure 4.3, the bottom input layer consists of two feature vectors $\mathbf{v}_u^U$ and $\mathbf{v}_i^I$ that describe user u and item i features, respectively. NCF initiates the recommendation process by embedding[3] users and item feature vectors ($\mathbf{v}_u^U$ and $\mathbf{v}_i^I$) into low-dimensional embeddings. For a user u and an item i, the user and item embeddings are denoted as $p_u \in \mathbb{R}^d$ and $q_i \in \mathbb{R}^d$, respectively. These embeddings are initialised randomly and updated during the training process [56]. The concatenation of user and item embeddings forms the input vector $x_{ui} \in \mathbb{R}^{2d}$ (Equation 4.1) for the subsequent neural collaborative filtering layers.

[3]Embedding is a technique used to represent categorical variables, such as users and items, as dense vectors in a continuous vector space. These embeddings capture the latent features and relationships between users and items, enabling the neural network to learn from their interactions.

This is shown in the embedding layer, where the user and item embeddings (p_u and q_i) are concatenated to form the input vector $\mathbf{x}_{ui}$ (for the neural network).

$$x_{ui} = \text{ Concatenate } (p_u, q_i)\, \forall ui \tag{4.1}$$

The input vector ($\mathbf{x}_{ui}$) is then fed into the neural collaborative filtering layers, which consist of multiple fully connected layers that learn the latent representations of users and items [56]. Specifically, to permit a full neural network treatment of collaborative filtering, the NCF layers consist of multiple fully connected perceptrons[4] with non-linear activation functions such as rectified linear units (ReLU)[5]. The activation functions are applied to the output of each perceptron, enabling the model to learn complex patterns and relationships in the data [56]. Thus, we can view the multi-layer perceptron as a series of stacked layers, each applying non-linear transformations through activation functions, enabling the model to learn hierarchical feature representations [139]. Effectively, through this architecture, the neural network learns the latent representations of users and items by transforming the input vector ($\mathbf{x}_{ui}$) through the multiple layers of non-linear transformations (output of one layer serves as the input of the next one) [56]. The final output layer is the predicted score $\hat{y}_{ui}$ (i.e., the predicted rating of user u for item i), and model training is performed by minimising the pointwise[6] loss between $\hat{y}_{ui}$ and its target value y_{ui} [56].

Formally, we can express the output of the neural network, denoted by $f(x_{ui})$, as a series of ReLU activation functions applied to weighted sums. Let $f(x_{ui})$ represent the output of the neural network, and $\hat{y}_{ui}$ denote the predicted rating for user u on item i.

$$f(x_{ui}) = \text{ReLU}\left(W_l \cdot \text{ReLU}\left(W_{l-1} \cdot \ldots \text{ReLU}\left(W_1 \cdot x_{ui} + b_1\right) + b_{l-1}\right) + b_l\right). \tag{4.2}$$

$$\hat{y}_{ui} = \left(f(x_{ui})\right). \tag{4.3}$$

Here, W_l represents the weight matrix of the l-th layer, b_l represents the bias vector of the l-th layer, and $\cdot$ denotes matrix multiplication. Equation 4.2 denotes that the input x_{ui} is passed through each layer of the neural network, with ReLU activation functions applied to the weighted sums at each layer. Equation 4.3 shows that the final output $f(x_{ui})$ represents the predicted rating $\hat{y}_{ui}$ for user u on item i - i.e., we apply a linear activation function[7] to the output of the last layer to predict the rating.

Thus, we can formulate and generalise the NCF's predictive model as

$$\hat{y}_{ui} = f\left(\mathbf{P}^T \mathbf{v}_u^U, \mathbf{Q}^T \mathbf{v}_i^I \,|\, \mathbf{P}, \mathbf{Q}, \Theta_f\right). \tag{4.4}$$

In this equation, $\mathbf{P}$ ($\mathbf{P} \in \mathbb{R}^{M \times K}$) is a matrix where each column represents a latent vector for a user. $\mathbf{Q}$ ($\mathbf{Q} \in \mathbb{R}^{N \times K}$) is a matrix where each column represents a latent vector for an item. $\mathbf{P}^T \mathbf{v}_u^U$ represents the user embedding obtained by multiplying the transpose of $\mathbf{P}$ with the user's latent vector $\mathbf{v}_u^U$. This operation extracts the relevant user embeddings from matrix $\mathbf{P}$. $\mathbf{Q}^T \mathbf{v}_i^I$ represents the item embedding obtained by multiplying the transpose of $\mathbf{Q}$ with the item's latent vector $\mathbf{v}_i^I$. Similarly, this operation extracts the relevant item embeddings from matrix $\mathbf{Q}$. Finally, $\mathbf{\Theta}_f$ essentially encapsulates all the learnable parameters of the neural network model. This includes the weights associated with the connections between

[4]A perceptron is a single-layer neural network that can learn linear decision boundaries.

[5]The ReLU activation function is defined as $f(x) = \max(0, x)$.

[6]Pointwise loss functions calculate the loss for each individual prediction separately.

[7]The linear activation function simply outputs the weighted sum of the inputs without applying any non-linear transformation.

neurons in each layer, as well as the biases, which are additional parameters added to each neuron that allow the network to learn more complex functions.

Effectively, equation 4.4 shows that $\hat{y}_{ui}$ represents the predicted rating for user u on item i. The prediction is generated using a function f (which represents the neural network) that takes as input two sets of latent vectors: $\mathbf{v}_u^U$ representing the user u, and $\mathbf{v}_i^I$ representing the item i. These latent vectors are multiplied with corresponding matrices $\mathbf{P}$ and $\mathbf{Q}$, respectively, which capture the interactions between users and items [56]. These extracted embeddings are then concatenated and fed as input to the function f, along with additional parameters $\mathbf{\Theta}_f$. The function f represents the neural network architecture, which learns the latent representations of users and items and outputs the predicted rating $\hat{y}_{ui}$ [56].

As mentioned, for the model to learn the latent representations of users and items effectively, the parameters $\mathbf{P}$ and $\mathbf{Q}$ are updated during the training process. Similarly, the parameters $\mathbf{\Theta}_f$ (includes the weights and biases) are also updated to optimise the neural network architecture. Specifically, the model optimises these parameters to minimise the disparity between predicted and actual ratings. This optimisation is achieved through a mean squared error loss function, penalising deviations of predictions from the actual ratings. The mean squared error loss is given as $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}(y_{ui} - \hat{y}_{ui})^2$, where n is the number of observations, $\hat{y}_{ui}$ is the predicted rating, and y_{ui} is the actual rating [56].

4.3.3 Algorithm Summary

The NCF algorithm is designed to learn the intricate user-item interactions through the use of neural networks. The following steps provide a concise summary of the NCF algorithm that we have detailed in Section 4.3.2. The summary is for building a recommender under the NCF algorithm using ratings only (not including review texts or sentiments). The algorithm is flexible for incorporating additional features, such as reviews and sentiments - which will be discussed further in Section 4.3.4.

The algorithm summary encapsulates the key steps of the NCF algorithm employed for this thesis. Effectively, the NCF algorithm utilises user and item embeddings combined with a neural network to generate predictions. The model is trained by optimising its parameters to minimise the mean squared error loss. These components collectively contribute to the effectiveness of NCF in capturing complex user-item relationships for our current recommendation tasks. The subsequent sections will look at how we integrated the review texts and sentiments (Section 4.3.4), as well as the general training procedure (Section 4.3.5), hyperparameter optimisation (Section 4.3.6), and final model summaries (Section 4.3.8).

4.3.4 Incorporating Text and Sentiments

The algorithm outlined in Section 4.3.2 and summary in Section 4.3.3 provides the blueprint for building a recommender system using the NCF algorithm using ratings only. However, to possibly enhance the predictive capabilities of the model, we propose incorporating additional information, specifically review text and sentiments. In this section, we detail the process of integrating review text and sentiments into the NCF model, providing a comprehensive overview of the steps involved.

The inclusion of text and sentiments necessitates slight algorithmic adjustments in the NCF model. We begin by considering augmenting NCF with review texts only. In Sections 3.3.2 we discussed how we cleaned (and normalised) the review text for each review. The last step before integrating this review text into the NCF model is to convert the text into numerical representations. This is done using a technique called word embedding, which maps words to dense vectors in a continuous vector space [8]. Word embeddings capture the semantic relationships between words, enabling the neural network to learn from

```
1: Embedding Initialisation
   Initialise user embeddings ($p_u$) and item embeddings ($q_i$) randomly in a low-dimensional space.
2: Concatenation of Embeddings
   Concatenate the user and item embeddings to form the input vector ($x_{ui}$) for the neural network.
   Input vector given as $x_{ui} = \text{Concatenate}(p_u, q_i)$.
3: Neural Network Layers
   Employ a neural network architecture with fully connected layers and non-linear activation functions
   (e.g., ReLU) to transform the concatenated embeddings.
   $f(x_{ui}) = \text{ReLU}\,(W_l \cdot \text{ReLU}\,(W_{l-1} \cdot \ldots \text{ReLU}\,(W_1 \cdot x_{ui} + b_1) + b_{l-1}) + b_l)$.
4: Prediction Score
   Apply a linear activation function to the output of the last layer to obtain the numerical rating pre-
   diction score ($\hat{y}_{ui}$).
5: Training Objective
   Define a mean squared error loss function ($L$) to measure the dissimilarity between predicted and
   actual ratings. Formally, $L = \text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}(y_{ui} - \hat{y}_{ui})^2$.
6: Optimisation
   Optimise the model parameters (embedding vectors, weights, and biases) using a suitable optimisa-
   tion algorithm.
7: Prediction Accuracy
   Assess the model's performance by comparing predicted ratings ($\hat{y}_{ui}$) against actual ratings.
8: Top-N Evaluation
   Evaluate the model's ability to generate the top-n recommendations by ranking predicted ratings for
   unrated items.
```

Algorithm 1: Neural Collaborative Filtering

the textual data [96]. Specifically, we use the pre-trained Universal Sentence Encoder (USE)[21] embeddings[8] to convert the review text into numerical representations. We used and loaded the USE model from the TensorFlow Hub[9] in Python. We passed each review text through the USE model, generating an output vector of fixed length for each review. Each vector encapsulates the semantic information of the review text and the values (in the vector) are learned during the training of the USE model. The dimensionality of the USE embeddings is typically set to 512, representing a default configuration for the pre-trained USE model [21].

Once we have the word embeddings (i.e., numerical representations) for the review text, we can integrate them into the NCF model. This is achieved by concatenating the review text embeddings with the user embeddings p_u, item embeddings q_i, to form the new expanded input vector $\mathbf{x}_{ui}$. This concatenation of results in a combined feature vector that captures the latent features and relationships between users, items, and the review text. During the training process, the model learns to jointly optimise the user, item, and review text embeddings along with the other parameters of the model (such as weights and biases) to minimise the chosen loss function.

To integrate the sentiment feature into the NCF model, we follow a similar approach as integrating the review text. Section 3.3.3 detailed how we used VADER to assign sentiment scores to each review, translating emotional cues (reviews) into quantifiable features for the model. As such, we have the sentiment feature in a numerical representation, a scalar value between -1 and 1, where positive values indicate

[8]The Universal Sentence Encoder (USE) is a pre-trained model developed by Google that converts text input into fixed-length numerical representations. It encodes input text into dense vectors that capture semantic information. These embeddings are learned from a large corpus of text data and are capable of capturing the semantic similarity and contextual information of text inputs.

[9]An open-source repository for reusable machine learning modules.

positive sentiment and negative values indicate negative sentiment. We then, simply, concatenate the user embeddings (p_u), item embeddings (q_i) and review text embeddings with the sentiment embedding. This concatenation combines the latent features of users, items and review text with the sentiment information, creating a single vector that incorporates all of the information. The concatenated vector ($\mathbf{x}_{ui}$)is fed as an input to the NCF model. During model training, the model learns to capture the relationships between users, items, review texts, and sentiment features.

Ultimately, incorporating additional features, namely the user reviews and review sentiments, is achieved by extending the input vectors ($\mathbf{x}_{ui}$) to accommodate these new dimensions. During the training phase, the neural collaborative filtering layers are adjusted to handle the augmented input vector. The model learns to weigh the significance of both numerical ratings, user reviews, and sentiments in making accurate predictions.

4.3.5 Model Training

We outlined the general algorithm for NCF in Section 4.3.2. In this section we look into the specifics of model training. The goal is to adjust the model parameters such that we minimise the loss function using the training dataset we established in Section 3.6. The model parameters include the user (p_u) and item (q_i) embeddings, the weights (W_i) and biases (b_i) of the neural network (which form $\mathbf{\Theta}_f$) - refer to Figure 4.2 for reference. When we augment the NCF model with review text and sentiments, we also include the review text and sentiment embeddings as part of the model parameters - i.e., we would need to learn the embeddings for the review text and sentiment features during training. Weights define the strength of connections between layers, adapting to reveal latent patterns in user-item interactions [1]. Larger weights amplify influences, while smaller weights diminish them, allowing the model to discern complex structures within the data [51]. Biases bring flexibility, representing baseline values or thresholds. Specifically, biases handle asymmetries in preferences, accommodating nuances in user-item interactions [56]. As such, the training process involves updating these parameters ($\mathbf{P}, \mathbf{Q}$ and $\mathbf{\Theta}_f$) iteratively to find the optimal values that minimise the error. The model sees the entire training dataset multiple times (epochs), allowing it to possibly learn the patterns in the data and perhaps improve its predictive capabilities. The way in which the parameters are updated is through an optimisation algorithm, such as stochastic gradient descent (SGD), or advanced optimisation algorithms like Adam, for optimising the parameters.

SGD stands as a fundamental optimisation algorithm widely employed in training neural networks [1]. Initially, the model parameters ($\mathbf{P}, \mathbf{Q}$ and $\mathbf{\Theta}_f$), undergo random initialisation (we set random seed to 10 for reproducability). Subsequently, during each training iteration, a mini-batch[10] of training examples is sampled from the dataset. For each example within the mini-batch, the input features traverse through the neural network, leading to the computation of predicted outputs. Concurrently, the loss function gets evaluated based on the predicted rating ($\hat{y}_i$) and the actual rating (y_i). This is know as the forward pass. Following the forward pass, the algorithm proceeds with the crucial step of backward pass or backpropagation - shown in Figure 4.2. Backpropagation entails computing the gradients of the loss function concerning the model parameters. This process involves calculating the gradient of the loss with respect to the output layer's activations and propagating these gradients backward through the network, thereby computing the gradients of the loss concerning the hidden layer activations and parameters. With the gradients in hand, the model parameters are updated to minimise the loss. Effectively, the gradients are calculated to discern the direction and magnitude of the steepest ascent[11] in the loss landscape. Subsequently, the model parameters are updated by moving against the gradients, with the learning rate

[10]A mini-batch is a subset of the training data containing a small number of examples, which is used to compute the gradient and update the model parameters during each iteration of training.

[11]The steepest ascent refers to the direction of the greatest increase in the loss function.

determining the step size. The model parameters are updated iteratively for multiple epochs, allowing the model to gradually converge towards a configuration ($\mathbf{P}, \mathbf{Q}$ and $\mathbf{\Theta}_f$) that minimises the loss. The choice of the optimisation algorithm influences the efficiency of this parameter-tuning journey, with alternative methods like Adam offering distinct approaches. Adam optimisation, for instance, offers adaptive learning rates and momentum[12]. Unlike SGD, which maintains a fixed learning rate throughout training, Adam dynamically adjusts the learning rate for each parameter based on estimates of the first and second moments of the gradients. This adaptive nature allows Adam to converge faster and has been shown to achieve better performance on various deep learning tasks over SGD [155].

Effectively, in the training phase, the model refines its knowledge of optimal user and item embeddings, weights, and biases by iteratively adjusting parameters, using an optimisation algorithm, to minimise the combined loss function. Minimising this loss function guides the model towards more accurate predictions [1]. MSE was chosen as the appropriate since it is widely used in regression tasks, such as predicting ratings, and amongst the literature ([1]; [51];[103]). We extend our loss function by incorporating regularisation to prevent overfitting. A regularisation term is added to the loss function to penalise large weights and biases, thereby preventing the model from fitting the training data too closely and improving its generalisation capabilities [51]. Specifically, we apply the L2 regularisation[13] term to the loss function, which penalises the sum of the squared weights and biases. The regularisation term is controlled by a hyperparameter λ, which determines the strength of the regularisation. Effectively, λ encourages the model to keep its parameters small, preventing the model from becoming too complex. In turn, this can help the model improve its ability to generalise well to unseen data [51]. Thus, the regularised loss function used in training the NCF model is given as

$$L = \frac{1}{N}\sum_{i=1}^{n}\sum_{j=1}^{m}(y_{ui} - \hat{y}_{ui})^2 + \lambda \sum_{p} \|\mathbf{\Theta}_f\|^2. \tag{4.5}$$

In Equation 4.5, N represents the total number of user-item pairs in the training set. The symbol (u,i) denotes a specific user-item pair from the training set. The term $\hat{y}ui$ represents the predicted rating for user u on item i, and yui is the actual rating provided by user u for item i [56]. The regularisation term $\lambda \sum_p \|\mathbf{\Theta}_f\|^2$ penalises the sum of the squared weights and biases (i.e, the neural network parameters $\mathbf{\Theta}_f$) in the model. The hyperparameter λ controls the strength of the regularisation term, with larger values of λ leading to stronger regularisation. As mentioned, The regularisation term helps prevent overfitting by discouraging the model from fitting the training data too closely and improving its generalisation capabilities [51]. We do not regularise the user and item embeddings, since regularising them could hinder the model's ability to learn the latent features and relationships between users and items - may lead to the loss of important information [56].

Ultimately, the optimisation algorithm, and hence training in general, guides the model through the complex parameter space with respect to the regularised loss function 4.5, facilitating adjustments to model parameters and enhancing its predictive accuracy capabilities.

4.3.6 Hyperparameter Tuning

Once we have completed the training phase and the model has acquired suitable parameters to minimise the desired loss function, we can leverage the validation dataset to optimise the model's hyperparameters. While model training is focused on configuring the parameters of the model, hyperparameter tuning, on

[12]Momentum is a technique used in optimisation algorithms to accelerate convergence by adding a fraction of the previous update to the current update.

[13]L2 regularisation, also known as Ridge regularisation, is a technique used to prevent overfitting in machine learning models by penalising the sum of the squared model parameters.

the other hand, specifically details the optimisation of the hyperparameters to achieve the best performance and generalisation capabilities [44]. Again, like model training, the goal is to find the optimal configuration that maximises the model's effectiveness on both the training and validation data. The performance on the validation data provides an indication of how well the model generalises to unseen data.

The hyperparameters that need to be tuned for the NCF model include the learning rate, the number of layers, the number of neurons in each layer, the batch size, the optimiser, the dropout, the number of epochs and the regularisation term. Specifically, the learning rate (see Section 4.3.5), determines how quickly the model learns the patterns in the data. A high learning rate can cause the model to converge too quickly and miss important patterns, while a low learning rate can cause the model to converge too slowly and take longer to learn the patterns [82]. The number of layers and neurons in each layer also affect the model's performance. A deeper network with more layers can learn more complex patterns, but it can also be more prone to overfitting [44]. Additionally, the batch size is the number of samples processed at once during training. A larger batch size can speed up training but can also require more memory [51]. The optimiser (see Section 4.3.5) is the algorithm used to update the weights during training [51]. Dropout is a regularisation technique that helps prevent overfitting by randomly setting a fraction of input units to zero during training [137]. Larger dropout rates can help prevent overfitting but can also reduce the model's capacity to learn complex patterns [137]. The number of epochs is the number of times the model sees the data during training [51]. Specifically, one epoch is one pass through the entire dataset. Training for more epochs can improve the model's performance, but it can also lead to overfitting and longer training times [44]. Finally, the regularisation term is a penalty term added to the loss function to prevent overfitting [51]. A larger regularisation term can help prevent overfitting but can also hinder the model's ability to learn complex patterns [51].

These hyperparameters mentioned are the ones chosen to be tuned for the NCF models. There are additional hyperparameters such as the loss function and the activation function(s), but for purpose of this thesis, we focus on the aforementioned hyperparameters. This is due to the computational tax of tuning all hyperparameters. To that end, the process of hyperparameter tuning involves selecting a range of values for each hyperparameter and training the model with different combinations of hyperparameters. The model's performance is evaluated on the validation dataset, and the hyperparameters that produce the best results (from the pool of hyperparameters provided) are selected. The process is typically performed using a grid search, which involves training the model with different combinations of hyperparameters and selecting the one that produces the best results (i.e., the lowest error) [12]. The grid search is an exhaustive search technique that evaluates all possible combinations of hyperparameters within a predefined range. The hyperparameter combination that produces the best results are then used to train the final model, which is evaluated on the test dataset to assess its performance [12].

Ultimately, these hyperparameters play a crucial role in the performance of the model and need to be tuned carefully to achieve the best results [12]. Table 4.3 provides a summary and brief description of common hyperparameters that need to be set when training the NCF models. As mentioned, using a grid search, we can train the model with different combinations of hyperparameters (shown in Table 4.3) and select the one (combination) that produces the best results.

From Table 4.3, we opted to tune only the number of units in the first layer of the NCF model. This is a consequence of the design choice we made. We followed the common practice where a pyramid pattern is employed - the bottom layer is the widest and each successive layer has a smaller number of neurons [78]. The premise is that by using a small number of hidden units for higher layers, they can learn more abstract features of data [80]. We empirically implement the tower structure, halving the layer size (number of units) for each successive higher layer, as such we only opted to optimise the number of units for the first layer.

Hyperparameter	**Description**	**Combinations**
Learning Rate	Controls the step size during the optimisation process.	0.01, 0.001, 0.0001
Epochs	Defines the number of times the entire dataset is passed forward and backward through the neural network.	50, 150, 300
Batch Size	Determines the number of samples processed in each iteration of training.	32, 64, 128
Number of Hidden Layers	Specifies the architecture of the neural network.	1, 2, 3, 6, 8
Number of Units in First Layer	Determines the size (number of neurons) in the first hidden layer.	128, 256, 512, 1024
Dropout Rate	Controls the rate at which randomly selected neurons are ignored during training.	0 (no dropout), 0.02, 0.05, 0.08
Optimisers	Optimisation algorithm used iteratively adjusting the model's parameters	SGD, Adam
L2 Regularisation	Regularisation term added to the loss function to prevent overfitting.	0.01, 0.001, 0.0001

Table 4.3: Hyperparameter values considered in grid search for neural collaborative filtering model

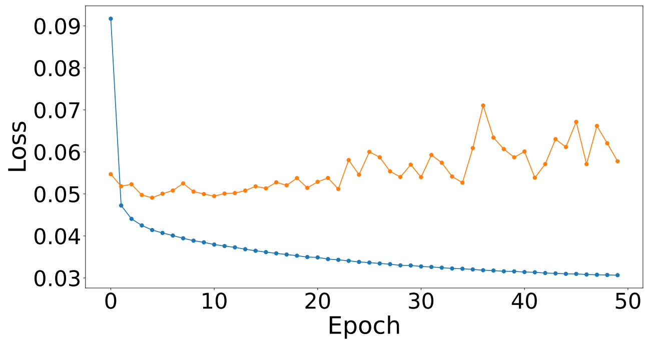
We previously explained that we opted to not tune the activation function hyperparameter. It is worth noting, that there are several options such as sigmoid, hyperbolic tangent, and ReLU, among others to choose from. Unlike linear models, deep neural networks leverage non-linear activations, to capture intricate patterns in user-item interactions. For this thesis, we specifically opted for the ReLU as the activation function as it encourages sparse activations[14], aiding in preventing overfitting as the network to focus on the most relevant features of the data [78]. ReLU, represented by $f(x) = max(0,x)$, is a piecewise linear function that yields the input directly for positive values and zero for negative values. This activation function was applied for all layers in the neural network architecture - and across all NCF models.

4.3.7 Training Results

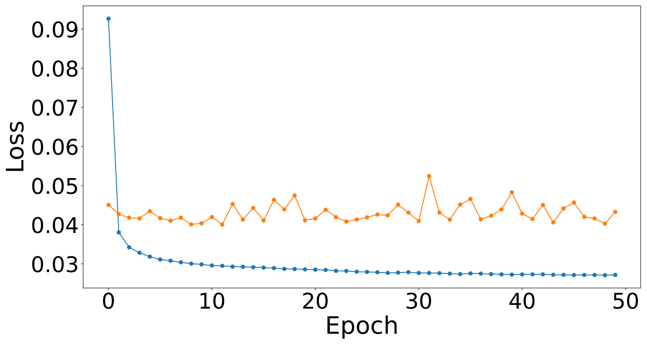
In this section we detail the results from training (Section 4.3.5) and validation (Section 4.3.6) from the different NCF models executed in our analysis. Specifically, we present the training and validation curves for a random combination of hyperparameters that we used for three different NCF models built in Figure 4.4. We also provide the curves for the best hyperparameter combination for each model in Figure 4.5. Table 4.4 and 4.5 display the randomly chosen combination of hyperparameters and the final (or optimal) combination of hyperparameters identifed through grid search, respectively. The training curve illustrates the training loss as a function of the number of epochs, while the validation curve shows the validation loss as a function of the number of epochs [1]. The training loss is the error on the training dataset, while the validation loss is the error on the validation dataset. The training and validation curves provide insights into the learning dynamics of the models, showing how the loss changes over time. The

[14]Refers to the situation where only a small subset of neurons (units) in a neural network's layer are activated or have non-zero outputs

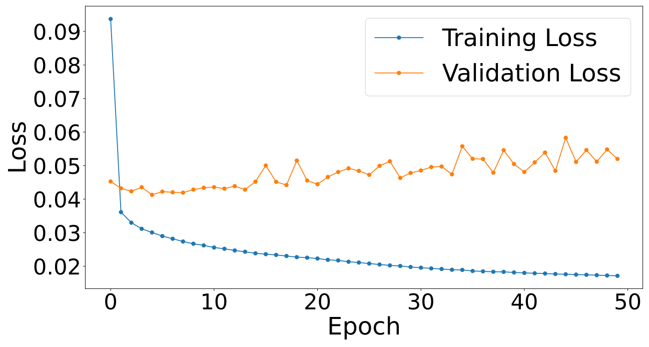
curves can help identify issues such as overfitting, underfitting, or convergence problems, and guide the selection of the best model [50].



(a) Model 1: ratings only.



(b) Model 2: ratings and review texts.



(c) Model 3: ratings, review texts and sentiments.

Figure 4.4: Training and validation curves for example combination of hyperparameters for all three NCF models.

Hyperparameter	**Value**
Learning Rate	0.001
Epochs	50
Batch Size	32
Number of Layers	6
Number of Units in First Layer	512
Dropout Rate	0.0
Optimiser	Adam
L2 Regularisation	0.001

Table 4.4: Example combination of hyperparameters for NCF models.

For Figure 4.4, we observe the training loss decreasing over each epoch for all models, indicating that the models are learning the patterns in the data and improving their predictive capabilities. The validation loss, however, appears to show a general increasing trend for Figures 4.4a and 4.4c. This is likely due to the model's inability to generalise well to unseen data, suggesting that the models may be overfitting the training data. Additionally, the training and validation curves are also far apart (and increasing in distance), suggesting that the models are not generalising well to unseen data [44]. There are several possible reasons as to why this is the case. Firstly, our model architecture could be too complex given the dataset. It may be memorising the training data. A simple fix would be to simplify the model. This can be achieved by reducing the hidden layers or units, or applying regularisation [12]. Table 4.4, indicates that there was no dropout rate applied (0.0), as well as a relatively high number of hidden layers (6). This combination of hyperparameters may be leading the model to not generalise well to unseen data and overfit the training data.

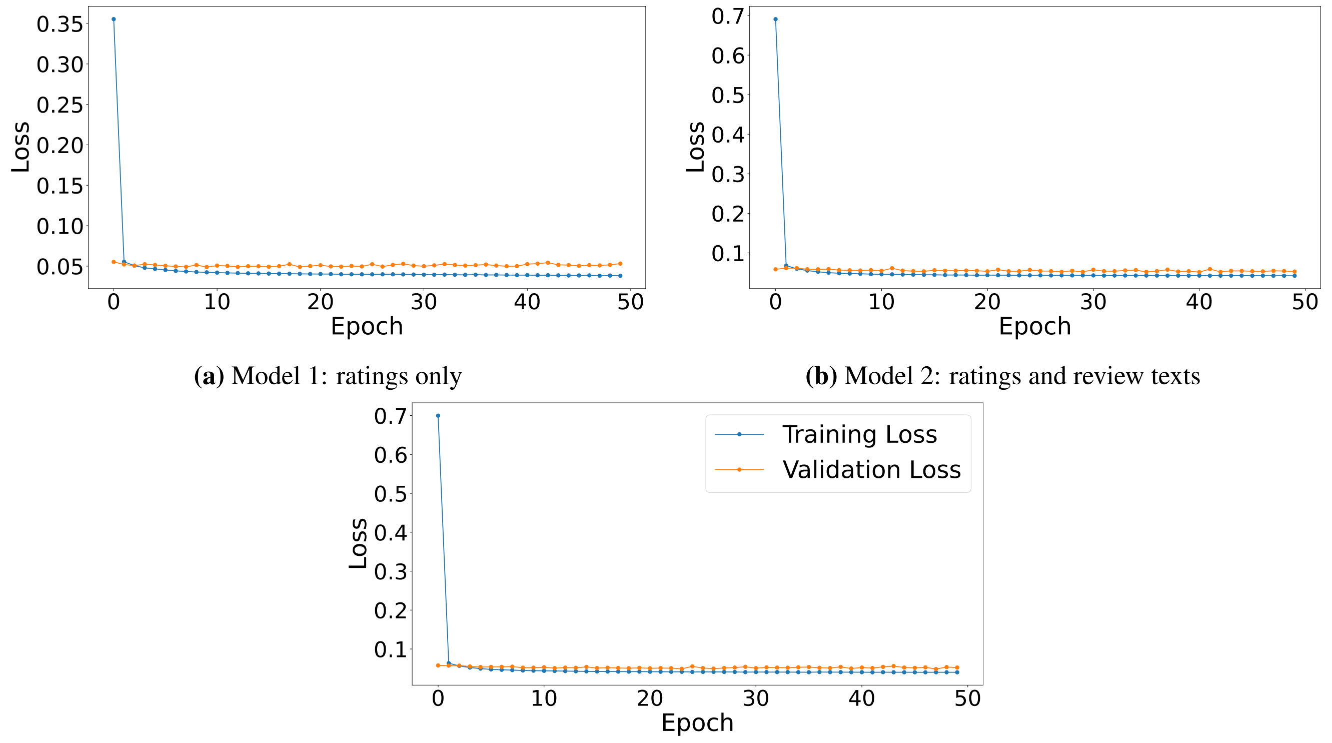
Model	**Hyperparameter**	**Value**
Model 1	Learning Rate	0.001
	Epochs	50
	Batch Size	128
	Number of Layers	2
	Number of Units in First Layer	512
	Dropout Rate	0.5
	Optimiser	Adam
	L2 Regularisation	0.01
Model 2	Learning Rate	0.001
	Epochs	50
	Batch Size	128
	Number of Layers	3
	Number of Units in First Layer	512
	Dropout Rate	0.5
	Optimiser	Adam
	L2 Regularisation	0.01
Model 3	Learning Rate	0.001
	Epochs	50
	Batch Size	128
	Number of Layers	3
	Number of Units in First Layer	512
	Dropout Rate	0.5
	Optimiser	Adam
	L2 Regularisation	0.01

Table 4.5: Final combination of hyperparameters for NCF models.

The training and validation curves for the best hyperparameter combinations for each of the three NCF models are shown in Figure 4.5. The curves indicate that all the models are learning the patterns in the data effectively and are performing well on both the training and validation datasets. This is illustrated by both the training and validation curves decreasing for all the models in Figure 4.5. Similarly, the training and validation curves are close to each other, indicating that the models are generalising well to unseen data. The curves are also smooth, suggesting that the models are converging well and not experiencing convergence problems [12]. Overall, the training and validation curves suggest that the models are learning the patterns in the data effectively and are performing well on both the training and validation datasets. Table 4.5 displays the final combinations of hyperparameters for each of the three models. As previously mentioned, these combinations of hyperparameters were selected based on the best performance on the validation dataset.

4.3.8 Summary

This section began by providing an overview of the workings and implementation of our NCF models. Effectively, the NCF model comprises multiple layers (see Figure 4.2), each serving a unique purpose in the model process. The input feature layer accepts user and item feature vectors, $\mathbf{v}_u^U$ and $\mathbf{v}_i^I$, respectively. From here, the embedding layer transforms these user and item vectors into user and item embeddings respectively. Subsequently, the user and item embeddings are concatenated to form the input vector, $\mathbf{x}_{ui}$, which serves as the input vector to the NCF layer. Additionally, for model 2 and 3 we discussed how we would extend the input vector $\mathbf{x}_{ui}$ to include the review text embeddings and sentiment embeddings,



(a) Model 1: ratings only

(b) Model 2: ratings and review texts

(c) Model 3: ratings, review text and sentiments

Figure 4.5: Training and validation curves for the best combination of hyperparameters from the set for all three NCF models.

respectively. This concatenation of results in a combined feature vector that captures the latent features and relationships between users, items, and the augmented features (embeddings). The concatenated input vector then traverses through multiple NCF layers, each leveraging rectified linear units (ReLU) as activation functions. The final output layer predicts the rating $\hat{y}_{ui}$ for a user u for item i.

We also delved into the training process, where we optimise the model parameters ($\mathbf{\Theta}_f$) which includes the weights and biases of the neural network and the user and item embeddings. For model 2 and 3, we would also include the review text embedding as a model parameter, and this too will be optimised in the training process. These parameters are optimised such that we minimise the regularised mean square error loss function (Equation 4.5). Additionally, we discussed in detail the hyperparameter tuning process and results. Hyperparameter tuning was done using the validation data and using a grid search technique. The best combination of hyperparameters (from the pool of options chosen) was used found for each model and used to create the final specification of that model. We conclude this section by outlining the three respective NCF models built and their architectures. Note, the optimal hyperparameters for all three models were summarised in Table 4.5.

Model 1: Ratings Only

Model 1 is the simplest of the three models, utilising only numerical ratings from the users to make predictions. The neural network model architecture (or NCF layers) are composed of two hidden layers, each with 512 units. The dropout rate is set to 0.5, and the L2 regularisation term is set to 0.01. The model is trained for 50 epochs with a batch size of 128. The learning rate is set to 0.001, and the optimiser used is Adam. The final output layer predicts the rating for a user-item pair. The hyperparameters for Model 1 are shown in Table 4.5. The achieved validation loss for this model is 0.050.

Model 2: Ratings Only

Model 2 extends Model 1 by incorporating user review text into the model. The neural network model architecture (or NCF layers) are composed of three hidden layers, each with 512 units. The dropout rate is set to 0.5, and the L2 regularisation term is set to 0.01. The model is trained for 50 epochs with a batch size of 128. The learning rate is set to 0.001, and the optimiser used is Adam. Similar to model 1, the final output layer predicts the rating for a user-item pair. The hyperparameters for model 2 are shown in Table 4.5. The achieved validation loss for this model is 0.043.

Model 3: Ratings, Review Text and Sentiments

Model 3 is the most complex of the three models, incorporating user review text and sentiments into the model. The neural network model architecture (or NCF layers) are composed of three hidden layers, each with 512 units. The dropout rate is set to 0.5, and the L2 regularisation term is set to 0.01. The model is trained for 50 epochs with a batch size of 128. The learning rate is set to 0.001, and the optimiser used is Adam. The final output layer predicts the rating for a user-item pair. The hyperparameters for model 3 are shown in Table 4.5. The achieved validation loss for this model is 0.043.

4.4 Benchmark Models

In our analysis and development of NCF, which serves as the cornerstone of this thesis, the imperative now lies in comparing its performance and discerning whether the integration of deep learning approaches actually brings about substantial improvements in collaborative filtering. To facilitate this comparison, we introduce three additional benchmark models more aligned and often associated with conventional collaborative filtering methodologies. These models are constructed to generate both rating predictions and top-*n* ranked lists (like the NCF models). By discussing and applying these models we can directly facilitate the comparison of a deep learning recommender approach to those more traditional techniques. The ensuing subsections looks into the technical details and specifications of these benchmark models, providing a comprehensive understanding of their design and functionality.

4.4.1 Non-Negative Matrix Factorisation

As mentioned in Section 2.2, matrix factorisation has emerged as one of the most successful implementations of collaborative filtering, particularly within latent factor models and the model-based framework of collaborative filtering [75]. The fundamental concept of matrix factorisation involves representing both users and items through vectors of factors derived from item-rating interactions [75]. This method gained popularity post the Netflix Competition for its commendable scalability coupled with predictive accuracy.

Non-Negative Matrix Factorisation (NMF) is a variant of matrix factorisation that has been widely applied in collaborative filtering tasks [79]. NMF is a dimensionality reduction technique that decomposes a given matrix into two non-negative matrices, $\mathbf{W}$ and $\mathbf{H}$, where $\mathbf{W}$ can represent the user-feature matrix, where each row corresponds to a user and the columns capture latent features, whilst $\mathbf{H}$ can represent the item-feature matrix, where each column corresponds to an item and the rows capture latent features. Furthermore, NMF imposes non-negativity constraints on these factor matrices, such that we ensure that the derived features are inherently non-negative, promoting interpretability and facilitating meaningful representations in diverse applications [76]. This particular matrix factorisation method was chosen due to its simplicity, interpretability, and effectiveness in capturing latent features within the data ([76]; [79]).

4.4.1.1 Algorithm Formulation and Summary

Specifically, a matrix factorisation model establishes a mapping of users and items to a shared latent factor space of dimensionality f [76]. This mapping enables the modelling of user-item interactions as inner products within this factor space. For NMF, each item i is represented by a vector $\mathbf{q}_i \in \mathbb{R}^f$, and each user u is associated with a vector $\mathbf{p}_u \in \mathbb{R}^f$. For a given item i, the elements of $\mathbf{q}_i$ measure the extent to which the item possesses those factors, with higher values indicating a stronger association between the item and the factor [76]. Similarly, for a given user u, the elements of $\mathbf{p}_u$ measure the extent of interest the user has in items that are high on the corresponding factors, again, higher values indicate a stronger association between the user and the factor. The dot product $\mathbf{q}_i^T \mathbf{p}_u$ captures the interaction between user u and item i, approximating the user's rating (r_{ui}) for item i [76]. . By extension, $\mathbf{Q}$ and $\mathbf{P}$ represent the factor matrices for all items and users, respectively. That is, these matrices are are composed of the factor vectors $\mathbf{q}_i$ and $\mathbf{p}_u$ as their columns, respectively. Thus, we can estimate the user's rating for an item using Equation 4.6.

$$\hat{r}_{ui} = \mathbf{Q}_i^T \mathbf{P}_u \tag{4.6}$$

The primary challenge for NMF lies in computing the factor matrices $\mathbf{Q}$ and $\mathbf{P}$ for all items and users. This is achieved by minimising some objective function using an optimisation algorithm. The objective function is typically the squared error between the observed ratings and the predicted ratings. The objective function for NMF can be formulated as shown in Equation 4.7.

$$\min_{Q,P} \sum_{(u,i)\in\kappa} \left(r_{ui} - \mathbf{Q}_i^T \mathbf{P}_u\right)^2 \tag{4.7}$$

In Equation 4.7, κ represents the set of observed user-item ratings. $\mathbf{Q}$ is the matrix containing the latent factors associated with items, whilst $\mathbf{P}$ is the matrix containing the latent factors associated with users. As such, $\mathbf{Q}_i^T$ represents the i-th column of the transpose of matrix $\mathbf{Q}$, corresponding to the latent factors for item i (i.e., $\mathbf{q}_i$). Similarly, $\mathbf{P}_u$ represents the u-th row of matrix $\mathbf{P}$, corresponding to the latent factors for user u (i.e., $\mathbf{p}_u$). Additionally, r_{ui} denotes the observed rating for user u on item i.

The goal in NMF, is to learn the factor vectors $\mathbf{Q}$ and $\mathbf{P}$ that minimise the squared error between the observed ratings (r_{ui}) and the predicted ratings $(\hat{r}_{ui})$. To achieve this, an optimisation algorithm is used with training data to update the factor vectors $\mathbf{Q}$ and $\mathbf{P}$ iteratively. Specifically, the factor matrices $\mathbf{Q}$ and $\mathbf{P}$ are updated using the gradient of the objective function with respect to the factor matrices. The gradient provides information on the direction and magnitude of the steepest ascent in the objective function. By moving against the gradient, the factor matrices are updated to minimise the objective function. To this end, the optimisation algorithm iteratively updates the factor vectors $\mathbf{Q}$ and $\mathbf{P}$ to minimise the objective function. The algorithm continues to update the factor matrices until the objective function converges to a minimum or a predefined stopping criterion is met. The factor matrices $\mathbf{Q}$ and $\mathbf{P}$ capture the latent features of the users and items, respectively. Effecitvely, once this mapping is achieved, the recommender system can estimate user ratings for any item using Equation 4.6. For this thesis, to minimise equation 4.8, there are two popular optimisation algorithms - SGD and alternating least squares[15]. We opted for SGD due to its popularity in similar research papers ([113]; [79]) where NMF models were developed.

An additional area of concern is the inherent sparsity in the user-item ratings matrix (see Section 2.7.1). Addressing only observed entries risks overfitting [76], particularly in scenarios with a high proportion of missing values. To mitigate this, some approaches ([105];[142]) propose modeling observed ratings directly while preventing overfitting through regularisation. This in fact the approach taken for this

15

building this benchmark model. The system minimises the regularised squared error on the set of known ratings. As such, the updated objective function for NMF with regularisation can be formulated as shown in Equation 4.8.

$$\min_{Q,P} \sum_{(u,i)\in\kappa} \left(r_{ui} - \mathbf{Q}_i^T \mathbf{Q}_u\right)^2 + \lambda \left(\|Q\|^2 + \|P\|^2\right) \tag{4.8}$$

In Equation 4.8, **Q** and **P** (as well as $\mathbf{Q}_i^T$ and $\mathbf{P}_u$) are defined as previously mentioned. The parameter λ is the regularisation parameter that controls the strength of the regularisation term and is usually determined by cross-validation [76]. The regularisation term penalises the sum of the squared elements of the factor matrices **Q** and **P**. The regularisation term discourages the factor vectors from becoming too large, thereby preventing overfitting and improving the generalisation capabilities of the model [76]. Once the factor matrices **Q** and **P** have been learned, the model can predict the rating for any user-item pair using Equation 4.6. To evaluate the performance of the NMF model, we assess how accurate the rating predictions are for unseen data, using the test dataset.

We provide the steps for NMF in Algorithm 2. In summary, the algorithm begins by initialising the factor matrices **Q** and **P** with random values. The algorithm then iteratively updates the factor matrices using the gradient of the objective function with respect to the factor matrices. The algorithm continues to update the factor matrices until the objective function converges to a minimum or a predefined stopping criterion is met. The factor matrices **Q** and **P** capture the latent features of the users and items, respectively. Once the factor matrices have been learned, the model can predict the rating for any user-item pair using Equation 4.6. The algorithm outputs the predicted ratings for the test dataset, which can be used to evaluate the models ability to correctly predict ratings for users. Subsequently, we can rank every predicted rating for each user and provide a list of top-n recommendations for each user, and hence evaluate this.

```
1: Initialisation
     Begin with the initial user and item matrices, P and Q, containing non-negative values.
     Set the number of latent factors, $f$, representing the dimensionality of the latent factor space.
2: Factorisation
     Iteratively update the factor matrices P and Q by minimising the squared error on the set of known
     ratings.
     Utilise optimisation techniques such as stochastic gradient descent to achieve convergence.
3: Regularisation
     Optionally introduce regularisation terms to the objective function to prevent overfitting.
     Adjust the regularisation parameter ($\lambda$) through cross-validation to optimise model performance.
4: Optimisation
     Employ cross-validation to determine optimal parameter values and ensure model robustness.
5: Prediction Accuracy
     Assess the model's performance by comparing predicted ratings ($\hat{y}_{ui}$) against actual ratings, consis-
     tent with the evaluation process for Neural Collaborative Filtering.
6: Top-$N$ Evaluation
     Evaluate the model's ability to generate top-$n$ recommendations by ranking predicted ratings for
     unrated items, consistent with the evaluation process for Neural Collaborative Filtering.
```

Algorithm 2: Non-Negative Matrix Factorisation Algorithm Summary.

The NMF model was built in Python from fundamentals. NMF has several hyperparameters that need to be set when training the model. These hyperparameters include the number of latent factors (k), the learning rate, the regularisation parameter (λ), and the number of epochs. The learning rate is the same as previously defined in NCF. Epochs in the context of NMF, refers to the number of iterations over the entire dataset during training. The regularisation parameter (λ) is a hyperparameter that controls the strength of the regularisation term in the objective function. The number of latent factors (k) is the number of latent factors that the model will learn to represent the data. Specifically, k is the number of latent factors that we assume exist in the data [79]. These latent factors are learned during the factorisation process and are represented by the columns of matrices $\mathbf{Q}$ and $\mathbf{P}$. Effectively, k determines the number of dimensions in which the user and item vectors are represented. That is, the matrix $\mathbf{Q}$ has dimensions $m \times k$, where m is the number of items, and the matrix $\mathbf{P}$ has dimensions $n \times k$, where n is the number of users [152]. Thus, the choice of k determines the number of latent factors that the model will learn to represent the data. A higher value of k allows the model to capture more complex patterns in the data but may also increase the risk of overfitting. Conversely, a lower value of k results in a more simplified representation of the data but may lead to underfitting [152]. To determine the optimal hyperparameters for the NMF model, a grid search was performed over a range of hyperparameters, albeit, the number of combinations was limited due to the computational tax incurred in running the developed NMF model. Thus, we limited the number of combinations for NMF for this thesis. Table 4.6 displays the hyperparameter combinations for the NMF model.

Hyperparameter	**Description**	**Combinations**
Number of Latent Factors	Determines the number of latent factors that the model will learn to represent the data.	25, 50, 100
Learning Rate	Controls the step size during the optimisation process.	0.01, 0.001
Regularisation Parameter	Regularisation term added to the loss function to prevent overfitting.	0.01, 0.001
Epochs	Defines the number of times the entire dataset is passed forward and backward through the neural network.	50, 100

Table 4.6: Hyperparameters for Non-Negative Matrix Factorisation

The final model had 25 latent factors, a learning rate of 0.001, a regularisation parameter of 0.01, and was trained for 50 epochs. This is summarised in Table 4.7. The model was evaluated using the test dataset to assess its performance in predicting user-item ratings. The model was also evaluated using the test dataset to assess its performance in generating top-n recommendations for each user. The model was compared with the NCF models to determine its effectiveness in generating recommendations. The results of the NMF model are presented in the subsequent chapter.

Hyperparameter	**Value**
Number of Latent Factors	25
Learning Rate	0.001
Regularisation Parameter	0.01
Epochs	50

Table 4.7: Final combination of hyperparameters for Non-Negative Matrix Factorisation

4.4.2 Item-Based Collaborative Filtering

As discussed in Section 2.2, collaborative filtering comprises two main branches: memory-based methods (Section 2.2.1) and model-based methods (Section 2.2.2). The Non-Negative Matrix Factorisation

(NMF) introduced in Section 4.4.1 serves as an example of a model-based method. Memory-based methods are distinguished by their reliance on the inherent patterns and relationships within the existing user-item interactions, without explicitly creating a predictive model [119]. These (memory-based) methods, and in particular neighbourhood-based approaches, are quite popular due to their simplicity [60].

In this section we focus on item-based collaborative filtering (IBCF), a prominent neighbourhood-based technique and thus memory-based approach for collaborative filtering. Item-based collaborative filtering centres around leveraging the similarities between items to make recommendations [85]. The premise is that users who have positively interacted with similar items in the past are likely to appreciate new items that share those similarities.

4.4.2.1 Algorithm Formulation

IBCF begins by constructing an item-item similarity matrix, which captures the similarity between items based on user-item interactions. Effectively, this enables the core principle of IBCF - measuring the similarity between items [58]. To achieve this item-item similarity matrix, we begin my transposing the user-item matrix (such as Table 4.1) to obtain an item-user matrix. Then for each pair of items i and j, we compute the similairty between the item vectors. This similarity between items is calculated using a similarity metric, such as the Pearson correlation coefficient or cosine similarity. Unlike the Pearson correlation, which assesses linear associations, the cosine similarity focuses on the angular similarity between vectors representing items or users in a multidimensional space [58]. For this thesis and the development of the IBCF model, we adhered to the empirical findings of [58] and chose to employ cosine similarity for IBCF as a metric to assess the closeness between items. Formally, given a set of all items I, all users U, and the user-item matrix $\mathbf{R}$ where r_{ui} represents the rating given by user u to item i, the cosine similarity $(\text{sim}(i,j))$ between items i and j is defined in Equation 4.9. In Equation 4.9, the numerator computes the dot product of the item vectors, while the denominator normalises the similarity metric by the product of the Euclidean norms of the item vectors.

$$\text{sim}(i,j) = \frac{\sum_{u \in U} r_{ui} \cdot r_{uj}}{\sqrt{\sum_{u \in U} r_{ui}^2} \cdot \sqrt{\sum_{u \in U} r_{uj}^2}} \tag{4.9}$$

The result of applying the cosine similarity metric to the item-user matrix is an item-item similarity matrix, which captures the similarity between items based on user-item interactions. The next step is to determine which other users' data will be used in the computation of a prediction for the active user—that is who will be selected to be in the active user's neighbourhood. We don't want to use all the users since it would not be feasible when trying to maintain real-time performance [58]. Moreover, many of the users do not have similar tastes to the active user, thus using them as predictors will only increase the error of the prediction. As such, we select k neighbours. Thus, the similarity matrix is used to predict the rating for a user-item pair by taking a weighted average of the ratings of the k most similar items (i.e., neighbouring items) to the target item - hence the term neighbourhood-based approach [85]. The predicted rating $\hat{r}_{ui}$ for a user u and item i is calculated using Equation 4.10. Here, $N(i;u)$ denotes the set of k items similar to i that user u has rated. This prediction mechanism (Equation 4.10) outlined here, effectively leverages the preferences of user u on similar items to estimate their preference for item i. The similarity values and ratings contribute to the weighted sum, providing a personalised and collaborative prediction for the user-item interaction.

$$\hat{r}_{ui} = \frac{\sum_{j \in N(i;u)} \text{sim}(i,j) \cdot r_{uj}}{\sum_{j \in N(i;u)} |\text{sim}(i,j)|} \tag{4.10}$$

A summary of the steps discussed in Section 4.4.2.1, are provided in Algorithm 3. In summary, we begin by computing the similarity between items using the cosine similarity measure. We then predict the rating $\hat{r}_{ui}$ for each user u and each item i not yet rated by u. This is achieved by taking the weighted sum of ratings for k nearest (similar) items in the calculation. Subsequently, we rebuild the user-item matrix $\hat{\mathbf{R}}$ using predicted ratings and historical ratings from the training set. The model's performance is then evaluated by comparing predicted ratings ($\hat{y}_{ui}$) against actual ratings. Finally, the model's ability to generate top-n recommendations is evaluated by ranking predicted ratings for unrated items. This evaluation procedure is consistent with the evaluation process for NCF (Section 4.3) and NMF (Section 4.4.1).

```
1: **Similarity Calculation**
   Compute the similarity between items using the cosine similarity measure.
2: **Prediction Computation**
   Predict the rating $\hat{r}_{ui}$ for each user $u$ and each item $i$ not yet rated by $u$.
   For each unrated item $i$ and user $u$, identify the $k$ most similar items to $i$ based on the item-item similarity matrix.
   Compute the predicted rating $\hat{r}_{ui}$ for item $i$ and user $u$ as a weighted sum of the ratings of the $k$ most similar items, with weights determined by their similarities to item $i$.
3: **Rebuild the user-item matrix $\hat{\mathbf{R}}$**
   Using predicted ratings and historical ratings from the training set, rebuild a completed user-item matrix.
4: **Prediction Accuracy**
   Assess the model's performance by comparing predicted ratings ($\hat{y}_{ui}$) against actual ratings, consistent with the evaluation process for Neural Collaborative Filtering.
5: **Top-$N$ Evaluation**
   Evaluate the model's ability to generate top-$n$ recommendations by ranking predicted ratings for unrated items, consistent with the evaluation process for Neural Collaborative Filtering.
```

Algorithm 3: Item-based collaborative filtering algorithm summary.

The IBCF was developed in Python from fundamentals. No packages or modules were used to implement the models. For our thesis, evaluated the performance of IBCF using various neighbourhood sizes. The values of k considered were 5, 10, 20, 40. The optimal value of k was determined through using the validation dataset. Specifically, we used the training data to compute the item-item similarity matrix. The optimal value of k was determined by evaluating the performance of the IBCF model using different values of k on the validation dataset. The metric used here to evaluate the performance of the IBCF model was the root mean squared error - the root of the average of the squared differences between the predicted and actual ratings. The value of k that resulted in the best performance on the validation dataset was selected as the optimal value of k for the IBCF model. The IBCF model was then evaluated using the test dataset to assess its performance in predicting user-item ratings. The model was also evaluated using the test dataset to assess its performance in generating top-n recommendations for each user.

4.4.3 User-Based Collaborative Filtering

IBCF focuses on leveraging the similarities between items to make recommendations [85]. In contrast, user-based collaborative filtering (UBCF) focuses on the similarities between users, operating on the premise that users who have exhibited similar preferences in the past are likely to continue having similar preferences in the future [69].

UBCF was the first iteration of collaborative filtering and is another type of neighbourhood-based technique (and hence memory-based approach) [119]. The workings of UBCF are very similar to IBCF, the only difference is lies in similarity calculation approach. While IBCF emphasises leveraging item similarities to make predictions based on users' past interactions with similar items, UBCF looks at establishing user similarities and making predictions based on the preferences of users who share similar tastes and preferences [69].

4.4.3.1 Algorithm Formulation

UBCF methodology is based off using the similarities between users to make personalised recommendations. The formulation involves calculating the similarity between users based on their historical ratings. Suppose for each pair of users u and v, we compute the similarity between the user vectors. Similar to IBCF, we use cosine similarity, due to the empirical finds from [58]. Formally, given a set of all items I, all users U, and the user-item matrix $\mathbf{R}$ where r_{ui} represents the rating given by user u to item i, the cosine similarity ($\text{sim}(u,v)$) between users u and v is defined in Equation 4.11. In Equation 4.11, the numerator computes the dot product of the user vectors, while the denominator normalises the similarity metric by the product of the Euclidean norms of the user vectors.

$$\text{sim}(u,v) = \frac{\sum_{i \in I} r_{ui} \cdot r_{vi}}{\sqrt{\sum_{i \in I} r_{ui}^2} \cdot \sqrt{\sum_{i \in I} r_{vi}^2}} \tag{4.11}$$

Having applied the cosine similarity metric to the user-item matrix, we obtain a user-user similarity matrix. We then select a set of K neighbours for the active user to be used in prediction computation. The k neighbours contribute to the prediction by calculating an average of their ratings. The prediction $\hat{r}_{ui}$ for the rating of item i by user u is computed using Equation 4.12. Here, $N(u;i)$ denotes the set of k users similar to u that have rated item i. The prediction mechanism (Equation 4.12) leverages the preferences of similar users to estimate the preference of user u for item i. The similarity values and ratings contribute to the weighted sum, providing a personalised and collaborative prediction for the user-item interaction.

$$\hat{r}_{ui} = \frac{\sum_{v \in N(u;i)} \text{sim}(u,v) \cdot r_{vi}}{\sum_{v \in N(u;i)} |\text{sim}(u,v)|} \tag{4.12}$$

We summarise the process for UBCF in Algorithm 4. In summary, the process is almost identical to IBCF. The difference lies in the similarity calculation. For UBCF we compute the similarity between users using the cosine similarity measure. We then predict the rating $\hat{r}_{ui}$ for each user u and each item i not yet rated by u. This is achieved by taking the weighted sum of ratings for k nearest (similar) users in the calculation. Subsequently, we rebuild the user-item matrix $\hat{\mathbf{R}}$ using predicted ratings and historical ratings from the training set. The model's performance is then evaluated by comparing predicted ratings ($\hat{y}_{ui}$) against actual ratings. Finally, the model's ability to generate top-n recommendations is evaluated by ranking predicted ratings for unrated items. This evaluation procedure is consistent with the evaluation process for NCF (Section 4.3) and the other benchmark models.

The UBCF model was developed in Python from fundamentals. No packages or modules were used to implement the models. For our thesis, we followed the same process of iderntifying k for UBCF than we did in IBCF. That is, we evaluated the performance of UBCF using various neighbourhood sizes. The values of k considered were 5, 10, 20, 40. The optimal value of k was determined through using the validation dataset. The value of k that resulted in the best performance on the validation dataset was selected as the optimal value of k for the UBCF model. The UBCF model was then evaluated using the test dataset to assess its performance in predicting user-item ratings. The model was also evaluated using the test dataset to assess its performance in generating top-n recommendations for each user.

```
1: Similarity Calculation
   Compute the similarity between users using the cosine similarity measure.
2: Prediction Computation
   Predict the rating $\hat{r}_{ui}$ for each user $u$ and each item $i$ not yet rated by $u$.
   For each unrated item $i$ and user $u$, identify the $k$ most similar items to $i$ based on the item-item similarity matrix.
   Compute the predicted rating $\hat{r}_{ui}$ for item $i$ and user $u$ as a weighted sum of the ratings of the $k$ most similar items, with weights determined by their similarities to item $i$.
3: Rebuild the user-item matrix $\hat{\mathbf{R}}$
   Using predicted ratings and historical ratings from the training set, rebuild a completed user-item matrix.
4: Prediction Accuracy
   Assess the model's performance by comparing predicted ratings ($\hat{y}_{ui}$) against actual ratings, consistent with the evaluation process for Neural Collaborative Filtering.
5: Top-$N$ Evaluation
   Evaluate the model's ability to generate top-$n$ recommendations by ranking predicted ratings for unrated items, consistent with the evaluation process for Neural Collaborative Filtering.
```

Algorithm 4: User-based collaborative filtering algorithm summary.

4.5 Evaluation

With the foundation laid in defining our models and outlining the approach for building, training, and optimising them (in Sections 4.3 and 4.4), our attention now turns to model evaluation — a facet briefly mentioned earlier in Section 4.1.2 yet deserving of a more comprehensive description. In this section, we first delve into the technical details of our approach to evaluating these models. Additionally, we explore the specific evaluation metrics utilised for this thesis for each paradigm of evaluation considered.

4.5.1 Evaluation Approach

To evaluate the performance of the recommenders, we adopted the leave-k-out partitioning approach to generate a set of unseen test data, as detailed in section 3.6. Specifically, for each and every user, we held-out at random 3 ratings as the test set. Next, we would build the models and optimise them using the training and validation datasets, respectively. Once we finalise our models, the next step involves an evaluation of its predictive capabilities using the designated test set [66]. For our thesis, the evaluation of the recommenders encompasses two distinct dimensions: predictive accuracy evaluation of the ratings and the top-n recommendation evaluation, each offering unique insights into the performance of our recommender system.

The evaluation of predictive accuracy looks at the model's ability to predict user ratings for items - that is, how our model's predictions compare to the actual hidden ratings (in the test set) [91]. So, effectively, the comparison between the real ratings, which were intentionally concealed during the leave-k-out process, and the predicted ratings generated by our final model, defines how good our recommender system is in terms of predictive accuracy. To achieve this, the actual ratings in our test set are used and compared to the predicted values. A larger difference between the predicted and actual, indicates that the model may struggle to accurately capture the nuanced preferences of users, leading to less reliable predictions. On the contrary, a small difference between the predicted and actual ratings signifies a higher level of precision in the model's understanding of user preferences, showcasing its ability to generalise well to unseen data and accurately predict ratings for users [138]. This nuanced evaluation provides a granular assessment of how well our recommender system aligns with the actual preferences of users for items that were omitted from the training process. By scrutinising the disparity between predicted and actual ratings, we gain valuable insights into the model's precision and generalisation to unseen items.

Additionally, our evaluation extends to the realm of top-n recommendations, assessing the model's proficiency in identifying and suggesting the most relevant items to users [138]. Formally, top-n evaluation process entails assessing the recommender system's effectiveness in recommending a relevant subset of items to users [138]. The process begins by taking all the unrated items and getting predicted ratings for them. Subsequently, we rank these items based on the predicted ratings. We then evaluate how well the model can prioritise the items that users are likely to interact with (i.e., relevant items). The goal is to ascertain whether the recommender system can accurately identify and suggest the most relevant items to users, tailored to their individual preferences. The items deemed relevant are the items which are highly rated in the test set for that user. Any item that was rated above a certain threshold in the test set was considered highly rated by a user and, hence, was deemed relevant to user. For this thesis, we chose to use the rating 3 as the threshold value, which was arbitrarily chosen. Ultimately, a recommender which can accurately identify these items among the unrated items, is considered to be effective in generating top-n recommendations. We assess the recommender for three different values of n: 10, 100 or 1000. This evaluation process offers valuable insights into the practical utility of our recommender system, gauging its effectiveness in generating personalised and meaningful recommendations tailored to individual user tastes [138].

Ultimately, in navigating both dimensions of evaluation (predictive accuracy and top-n evaluation), we ensure a comprehensive understanding of our model's strengths and areas for refinement. This multifaceted approach to assessment not only gauges the accuracy of individual predictions but also examines the broader utility of the recommender system in recommending the most suitable items to users in a real-world context [60]. Whilst not fully a holistic view of the recommender system as mentioned in Section 2.6, it is a step in the right direction to providing a more balanced perspective on the performance of our recommender systems. Several metrics have been proposed in order to evaluate the performance of the various models employed by a recommender system. The next two sections will look at the specific predictive accuracy (Section 4.5.2) and top-n (Section 4.5.3) based metrics used in this thesis in detail.

4.5.2 Predictive Accuracy Metrics

Given a user u, a model should be able to predict ratings for any unrated items. The pair $< p_i, a_i >$ refers to the prediction on the i^{th} test instance and the corresponding actual value given by the user. We chose to consider the predictive accuracy metrics mean square error (MSE), mean absolute error (MAE) and root mean square error (RMSE). These metrics measure how close the prediction p_i is to the true numerical rating a_i expressed by the user [60]. Consequently, the evaluation can be done only for items that have been rated. We compare the predicted rating for the items that are in the test set, for which we have their actual ratings.

MSE, adopted in [16], is an error metric defined as the average of the squared differences between the predicted and actual ratings. The formula for MSE is given by Equation 4.13. Although MSE is simple to compute, it tends to exaggerate the effects of possible outliers, that is instances with a very large prediction error.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(p_i - a_i)^2 \tag{4.13}$$

RMSE, used in [11], is a variation of MSE that takes the square root of the average of the squared differences between the predicted and actual ratings. The formula for RMSE is given by Equation 4.14.

RMSE is used to address the issue of the scale of the error metric, as it gives the same dimension as the predicted value itself.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - a_i)^2} \tag{4.14}$$

As MSE and RMSE squares the error before summing it, they both suffer from the same outlier problem [138]. MAE, used in [60], is an error metric defined as the average of the absolute differences between the predicted and actual ratings. The formula for MAE is given by Equation 4.15. MAE is less sensitive to outliers compared to MSE and RMSE, as it does not square the error before summing it.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|p_i - a_i| \tag{4.15}$$

Lower MSE, RMSE and MAE values correspond to higher prediction accuracy. Since RMSE squares the error before summing it, it tends to penalise large errors more heavily. MAE is the most used metric because of its easy implementation and direct interpretation ([60]; [118]; [59]). Notably, the defined metrics in Equations 4.13, 4.14 and 4.15 are not an exhaustive list of metrics for predictive accuracy - there are other metrics widely used such as Pearson correlation which essentially calculate the correlation between the predicted and the true ratings [138]. Yet, due to their simplicity, MSE, RMSE and MAE find themselves widely used in the evaluation of recommender systems and as such are also used in this thesis for the predictive accuracy task.

4.5.3 Top-*N* Evaluation Metrics

The top-*n* evaluation task has been widely adopted within e-commerce, since the goal in top-*n* recommendation is to provide a ranked list of items that are most likely to be of interest to the user [36]. This is particularly useful in scenarios where there is a limited amount of space to display recommendations, such as in online shopping platforms or news websites [36]. Specifically, the performance of a ranked list can be judged using several metrics. For this thesis, we consider the precision, recall, F1-score at *n*, since these metrics are widely used in information retrieval and are well-suited for evaluating the performance of top-*n* recommendation systems [36]. For the metrics used, we calculated the scores per user and then averaged them across all users to obtain the final scores for each metric.

Precision at *n* (precision@*n*) is a metric that measures the proportion of relevant items among the top-*n* recommendations. The formula for precision at *n* is given by Equation 4.16. Precision at *n* is calculated by dividing the number of relevant items from the test set that appear in the top-*n* recommendations by the total number of items in the top-*n* recommendations (i.e., *n*). A higher precision at *n* value indicates that a higher proportion of the top-*n* recommendations are relevant to the user [36].

$$\text{Precision@n} = \frac{\text{Number of relevant items in top-}n\text{ recommendations}}{n} \tag{4.16}$$

Recall at *n* (recall@*n*) is a metric that measures the proportion of relevant items in the top-*n* recommendations out of all the relevant items in the test set. The formula for recall at *n* is given by Equation 4.17. Recall at *n* is calculated by dividing the number of relevant items from the test set that appear in the top-*n* recommendations by the total number of relevant items in the test set. A higher recall at *n* value indicates that a higher proportion of the relevant items in the test set are present in the top-*n* recommendations [36].

$$\text{Recall@n} = \frac{\text{Number of relevant items in top-}n\text{ recommendations}}{\text{Total number of relevant items in the test set}} \tag{4.17}$$

F1-score at n (F1-score@n) is a metric that combines precision and recall at n into a single metric. The formula for F1-score at n is given by Equation 4.18. F1-score at n is calculated by taking the harmonic mean[16] of precision at n and recall at n. A higher F1-score at n value indicates that the recommender system is able to provide a balanced trade-off between precision and recall at n [36]. Thus, the F1-score at n is a useful metric for evaluating the overall performance of a recommender system in generating top-n recommendations.

$$\text{F1-score@n} = \frac{2 \times \text{Precision@n} \times \text{Recall@n}}{\text{Precision@n} + \text{Recall@n}} \tag{4.18}$$

4.6 Hardware and Software

The models were developed and trained on a local machine (2018 MacBook Pro) with the following specifications: Intel Core i5-8259U CPU, 8GB LPDDR3 RAM, and Intel Iris Plus Graphics 655 integrated graphics. The software used for the development and training of the models include Python 3.9.12, TensorFlow 2.6, and scikit-learn 0.24.2. The TensorFlow library was used to build and train the neural network models, while scikit-learn was used for data preprocessing and evaluation.

The code for the models was written in Python and is available on GitHub[17]. The code is seperated into 8 different Jupyter Notebook's, each notebook corresponding to a different section of the dissertation, such as the data loading, data cleaning, sentiment analysis etc. Jupyter Notebook was used extensively for all development, experiementation and visualisations. There is also 4 distinct notebooks each concerned with building a specific recommender model. All the benchmark models (see Section 4.4) were built from scratch, using no packages. The NCF model was built upon the TensorFlow programming framework. The details regarding each model's computational run time for training is observed and will be provided in Chapter 5.

4.7 Conclusion

This chapter began by walking through the generalised modelling approach taken for this thesis in Section 4.1, starting with how we formulated the recommendation problem and then describing how we would convert the cleaned data into a user-item matrix and then partition the data, to finally start building, optimising and evaluating our recommendation models. We provided some background into the workings of neural networks in Section 4.2. In Section 4.3, we adressed the workings of the primary model for this thesis, the neural collaborative filtering recommender system, including the algorithm, training procedure and hyperparameter selection. We detailed how NCF essentially uses neural networks to learn intricate representations of users and items, enabling it to make predictions for user-item pairs. Similarly, in Section 4.4, we briefly summarised the benchmark models used to compare our results from the NCF model. This detailed the workings for non-negative matrix factorisation, as well as the neighbourhood-based approaches, namely item-based and user-based collaborative filtering. We ended of this chapter with section 4.5 which looked at the technical details of evaluation process; specifically, how we evaluated the recommender systems using predictive accuracy metrics such as MSE and MAE which measure the closeness of predicted ratings to the true ratings, as well as top-n metrics such as

[16]The harmonic mean is a type of average that is calculated by dividing the number of observations by the reciprocal of each number in the dataset, and then taking the reciprocal of the result.

[17]The code for the models is available at: `https://github.com/pavsingh7/Masters-Dissertation`

recall and precision at n which measure the recommender's ranking ability. Lastly, we provided details on the hardware and software used for the development and training of the various recommender models covered in this chapter.

This chapter serves as a road map for our thesis, guiding our exploration of the NCF recommender system (as well as the benchmark models) and establishes a solid foundation and context of the evaluation criteria for us to speak about the results from this thesis. Thus, in the subsequent chapter we examine the results from the analysis in this thesis.

Chapter 5

Results & Discussion

In Chapter 3 we established the source and details of the Amazon product review dataset and Chapter 4 addressed the details of the models and techniques adopted in our analysis of building several recommender systems. This has provided a platform for which we can critically address the primary research questions for this thesis, raised in Section 1.3, by evaluating the various recommender system's predictive accuracy and top-*n* capability.

Specifically, this chapter will present the results of the experiments conducted to evaluate the performance of the models described in Chapter 4. The results will be presented in the context of the research questions and the previous literature within this domain. We first look at the results of the experiments in Section 5.1. Section 5.2 explores the results of the models in the context of the research questions and the previous literature within this domain. The chapter concludes by providing a summary of the results and discussion, in Section 5.3.

5.1 Results

In this section we display the results from our analysis, comparing the performance of several different recommenders under two different tasks - rating prediction (Section 5.1.1) and top-*n* evaluation (Section 5.1.2). As discussed in Chapter 4, the models used in our analysis include the two neighbourhood-based collaborative based approaches (UBCF and IBCF), a model-based collaborative approach (NMF) and the relatively new deep-learning based approach to collaborative filtering (NCF). We built these models for ratings prediction - however, as highlighted in Section 4.5.1, we evaluated our models under both predictive accuracy and top-*n* paradigms.

5.1.1 Predictive Accuracy Results

Predictive accuracy, in the context of a recommender system, refers to how well a model is able to predict user preferences or ratings for items [14]. It is usually measured by comparing the predicted ratings to the actual ratings in a dataset. For each user-item pair in the test set, the model makes a prediction about the rating the user would give to the item. The predicted ratings are then compared to the actual ratings in the test set. The measure of how well the model's predictions match the actual ratings is called predictive accuracy [65]. The metrics of choice are discussed in detail in Section 2.6. Table 5.1 illustrates the results of our various recommender models. As discussed in Section 4.1, the NCF model was tested under three different scenarios: using ratings only, using ratings and reviews, and using ratings, reviews, and sentiments. Again, we only include reviews and sentiments in our NCF model - not any of the benchmark models.

The baseline model in Table 5.1 is a simple model that predicts the average rating for all items. Among these models shown in Table 5.1, the NCF model, trained with ratings and reviews (model 5) delivered the most accurate predictions, with an MAE of 0.490, an MSE of 0.591, and an RMSE of 0.769. On the contrary, the NMF model trained with ratings only exhibited the poorest accuracy, recording an MAE of 1.583, an MSE of 3.467, and an RMSE of 1.862. We also noted during our analysis, that both neighbourhood-based approaches, IBCF and UBCF, yielded comparable results. This similarity might be attributed to their shared reliance on the user-item interaction matrix [69].

	Recommender Model	Data Used	MAE	MSE	RMSE
1	Item-Based Collaborative Filtering	Ratings Only	0.581	0.830	0.911
2	User-Based Collaborative Filtering	Ratings Only	0.59	0.984	0.992
3	Non-Negative Matrix Factorisation	Ratings Only	1.583	3.467	1.862
4	Neural Collaborative Filtering	Ratings Only	0.572	0.815	0.903
5	Neural Collaborative Filtering	Ratings & Reviews	**0.490**	**0.591**	**0.769**
6	Neural Collaborative Filtering	Ratings, Reviews, Sentiments	0.492	0.607	0.779
7	Baseline Model	Ratings Only	0.67	1.270	1.311

Table 5.1: Predictive Accuracy Results for Recommender Models Built.

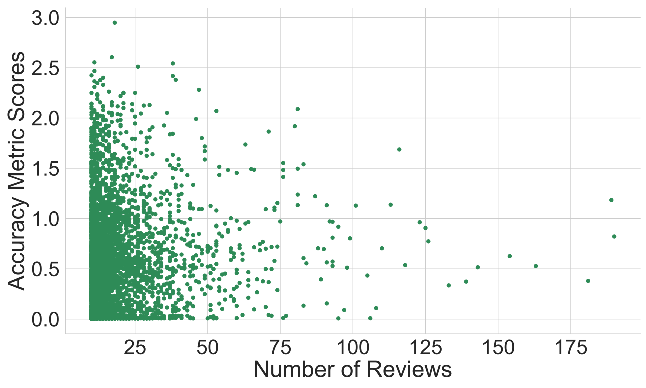
To determine if the differences in performance between the models are statistically significant, we conducted a paired t-test. This statistical test evaluates whether the differences in the means of two groups are significant or if they could have occurred by random chance [92]. There are a several assumptions that need to be met for the t-test to be valid, including the normality of the data, the homogeneity of variances, the independence of observations and random sampling [92]. For this analysis, we checked whether the assumptions were met and found that they were. Specifically, we found that the data within each group (referring to each recommendation model being evaluated) should follow a normal distribution - the Shapiro-Wilk test[1] was used to confirm this. We also checked for homogeneity of variances between the groups (referring to the different recommendation models) using Levene's test[2]. Independence was met since we established consistent data partitioning for each model - so each model was trained and tested on the same data. Lastly, the data was randomly sampled using data partitioning (see Section 3.6). Thus, we were able to confirm whether the performance differences between NCF with reviews and other models (IBCF, UBCF, and NMF) were statistically significant. The results of the t-test indicated a significant difference in the performance of NCF with reviews (model 5) compared to IBCF ($p < 0.01$), UBCF ($p < 0.01$), and NMF ($p < 0.01$). This confirmed our initial findings that NCF with reviews outperformed other models and that these differences were not due to random chance.

From Figures 3.1 and 3.3 in Section 3.5, we identified that a key feature of the data is that most products have relatively few reviews (<50) and most users have reviewed only a few things (<20). This may have implications for the accuracy of the models. For example, the recommender models may be better at predicting ratings for products with more reviews, as it can learn more about the product from the reviews. Similarly, the models may be better at predicting ratings for users with more reviews, as it can learn more about the user from the reviews. To explore this, we can look into the overall accuracy results to see if accuracy depends on any factors - for example, is accuracy higher for users or products with more reviews, or for users with longer review text?

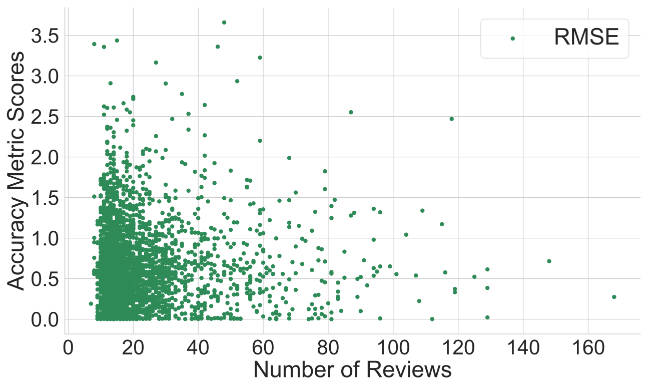
We first look at the impact of the number of reviews per user or item to see if there is a difference in the accuracy of the models for users or items with more reviews. To achieve this, we took users and items in the test set, and assigned them into four groups based on the number of reviews they had. We created these groups by dividing the users and items into quartiles based on the number of reviews they had. We then calculated the average MAE, MSE and RMSE for each group. We can then compare these results to see if the number of reviews has any impact on the accuracy of predictions for users or items who have more reviews. Figures 5.1a and 5.1b show the MSE, RMSE and MAE against the number of reviews for users and items respectively.

[1]The Shapiro-Wilk test assesses the normality of data. A p-value greater than the chosen significance level (e.g., 0.05) indicates that the data can be assumed to be normally distributed.

[2]Levene's test evaluates the homogeneity of variances between groups. A non-significant p-value suggests that the variances are approximately equal across groups, which is a prerequisite for t-tests.



(a) Accuracy against the number of reviews for users.



(b) Accuracy against the number of reviews for items.

Figure 5.1: Comparison of RMSE for user and item ratings.

From Figures 5.1a and 5.1b, we can see that the for users and items, those with lower review counts appear to have a wide spread in accuracy (for RMSE). The resulting plots for MSE and MAE show similar results. Those users and items which have higher number of reviews, seem to generally achieve better (lower) accuracy scores. This suggests that the recommender model is better at predicting ratings for users with more reviews. This is likely because the models can learn more about the users, since they have more reviews. We also show the mean RMSE, MAE and MSE for each group in Tables 5.2 and 5.3 for users and items respectively. For items, we see that the accuracy metrics appear to be similar across the groups. This suggests that the recommender model is not better at predicting ratings for items with more reviews, nor is it worse at predicting ratings for items with fewer reviews.

To test this formally, we conduct an analysis of variance (ANOVA) test to see if the differences in RMSE between the groups are statistically significant. The results of the ANOVA test indicate that the differences in RMSE between the groups are statistically significant for users ($p < 0.001$). However, they are not statistically significant for items ($p = 0.1$). Thus, to ascertain which groups are significantly different from each other for the users, we conduct a post-hoc test. The post-hoc test we use is the Tukey HSD test[3] which is used to determine which groups are significantly different from each other. The results of the Tukey HSD test are shown in Tables A.1 in Appendix A for users. We did not use the Tukey HSD test for items as the ANOVA test did not show any statistically significant differences between the groups.

The results of the Tukey HSD test for users show that the differences in RMSE between the groups are statistically significant, however not for all groups. Specifically, the differences between the low and very high group, and the low and high group are statistically significant. Thus, there is evidence to suggest that the recommender model is better at predicting ratings for users with more reviews. This is likely because the models can learn more about the users, since they have more reviews. This is consistent with previous research that has found that users with more reviews tend to have better accuracy in predicting ratings [136]. By segmenting the users into groups based on the number of reviews they have, we can see that the accuracy of the models varies across the groups. Effectively, our analysis here suggests that the number of reviews a user has can impact the accuracy of the model's predictions for that user. This finding suggests that the quality of the recommendations accuracy can be influenced by the amount of data available for each user. For items, we did not find any statistically significant differences between the groups. This suggests that the number of reviews an item has does not impact the accuracy of the model's predictions for that item.

[3]The Tukey HSD test is a post-hoc test used after an ANOVA test to determine which groups are significantly different from each other. It is used to identify the differences between group means.

Group	Number of Reviews	MAE	MSE	RMSE
Low	0-12	0.517	0.686	0.635
Medium	13-15	0.525	0.641	0.622
High	16-21	0.486	0.636	0.589
Very High	21+	0.448	0.560	0.544

Table 5.2: Mean MAE, MSE, and RMSE for each group of users.

Group	Number of Reviews	MAE	MSE	RMSE
Low	0-13	0.526	0.681	0.652
Medium	14-16	0.518	0.659	0.638
High	6-25	0.519	0.671	0.644
Very High	26+	0.523	0.705	0.644

Table 5.3: Mean MAE, MSE, and RMSE for each group of items.

Additionally, Figure 3.5 from Section 3.5 shows that most reviews are relatively short, with a word count of less than 100. We shall explore if this has any implications for the accuracy of the models. For example, is the NCF model better at predicting ratings for reviews with more words? To explore this, we follow a similar approach as previously done. We begin by getting the review length of all the reviews (user-item interactions) that appear in the test set. We then assigned these reviews to four diffent groups, split based on the quartiles. We then calculated the average MAE, MSE and RMSE for each of these groups. Figure 5.2 shows the RMSE against the review length in words for users. Table 5.4 shows the mean RMSE, MAE and MSE for each group of reviews, assigned based off the review length in words. It appears that reviews with less words have better accuracy in predicting the ratings. Using the ANOVA test, we identify that the differences in RMSE between the groups are statistically significant (p<0.001). The resulting Tukey HSD test is shown in Appendix A in Table A.2. We found that reviews with less words had better accuracy in predicting the ratings. This insight, suggests that the length of the review text can impact the accuracy of the model's predictions. Perhaps, the models are better at predicting ratings for shorter reviews, as they may contain more concise and relevant information. Additionally, reviews with more words may contain more noise or irrelevant information, which could impact the accuracy of the predictions. Effectively, we found that the length of the review text can impact the accuracy of the model's predictions.

Group	Number of Reviews	MAE	MSE	RMSE
Low	0-6	0.262	0.276	0.262
Medium	7-31	0.443	0.585	0.443
High	32-118	0.610	0.808	0.601
Very High	118+	0.682	0.878	0.682

Table 5.4: Mean MAE, MSE, and RMSE for each group of users based on review length.

5.1.2 Top-*N* Results

As detailed in Section 2.6, top-*n* recommendation represent an important aspect of recommender systems, as they reflect the system's ability to generate relevant suggestions for users [90]. As such, this evaluation paradigm involves evaluating the effectiveness of a recommender system by considering only

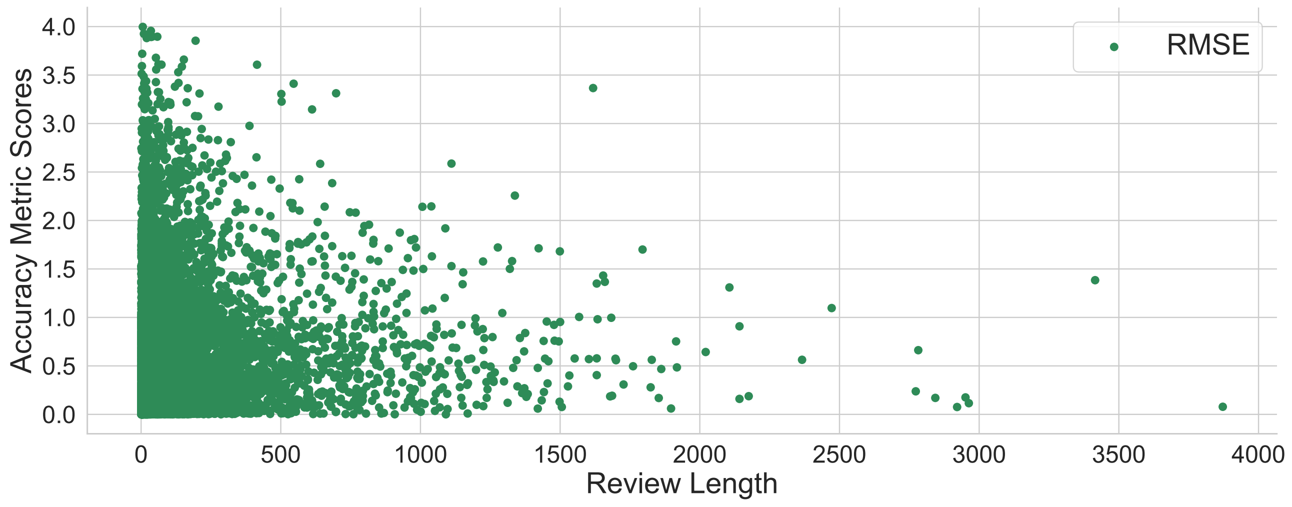


Figure 5.2: RMSE against the review length in words for users.

the top-*n* recommendations it provides for each user. Our recommender models generated a list of *n* items for each user. The list of *n* items were ranked by the system's estimated rating of each item. We can now evaluate the system's performance based on how many of the items in the top-*n* list match with items that the user has actually interacted with or rated positively - the test set. The result of which is displayed in Table 5.5.

	Model	Data Used	Recall@100	Precision@100	F1-Score@100
1	Item-Based Collaborative Filtering	Ratings Only	0	0.0009	0.0001
2	User-Based Collaborative Filtering	Ratings Only	0	0	0
3	Non-Negative Matrix Factorisation	Ratings Only	0	0.0007	0
4	Neural Collaborative Filtering 1	Ratings Only	0.0007	0.0010	0.0006
5	Neural Collaborative Filtering 2	Ratings, Reviews	0.0007	0.0012	0.0006
6	Neural Collaborative Filtering 3	Ratings, Reviews, Sentiments	0.0007	0.0012	0.0006

Table 5.5: Top-N Results for Recommender Models Built.

Recall@100, Precision@100, and F1-Score@100 are metrics used to evaluate the performance of a recommender system in generating a list of top-*n* (where *n* is 100) recommendations for each user. As stipulated in 4.5.1, we execute the top-*n* evaluation for each user and then aggregate the result, to get a final mean score for each of the metrics. As mentioned, recall@100 represents the proportion of relevant items that were included in the top 100 recommendations, out of all the relevant items in the dataset [36]. A higher recall@100 value indicates that the recommender system is doing a better job of capturing relevant items in its recommendations [36]. In our case, all the benchmark models have a recall@100 of 0, meaning that none of the relevant items were included in the top 100 recommendations. Similarly, for NCF models, the majority of users did not have any relevant items in their top-100 recommendations. In contrast, precision@100, which simply represents the proportion of relevant items among the top 100 recommendations [36], has scores generally above 0. The IBCF model has a precision@100 of 0.0009, which means on average, 9 out of every 100 users had 1 of their top 100 recommendations relevant, while the remaining 91 had none. The NMF model has a precision@100 of 0.0007. The NCF models had the highest precision@100 scores - NCF model with reviews (and with sentiments), achieved the highest precision score of 0.0012. Effectively, on average, 12 out of every 100 users had 1 of their top 100 recommendations relevant, while the remaining 88 had none. Finally, the F1-Score@100, which represents the harmonic mean of precision@100 and recall@100, was obtained for each model. Note, a higher F1-score@100 value indicates better overall performance in terms of capturing relevant items in the top recommendations. In our case, IBCF was the only benchmark model to have a value above 0 - F1-score@100 of 0.0001. In contrast, the NCF model's all have an F1-score@100 of 0.0006.

5.2 Discussion

In this section we discuss the practical implications of the study's findings in the context of research questions. We also compare our findings with existing literature and studies in the field, highlighting areas where our results align or diverge from previous research.

From Table 5.1 we identified that the family of NCF models outperformed all the other models in terms of predictive accuracy. This suggests that integrating neural networks into the collaborative framework can improve predictive accuracy. The success of NCF models can be attributed to the deep learning techniques they utilise, enabling them to learn complex relationships and patterns in the user-item interaction data [56] - which they appear to achieve in this problem task. Neural networks are capable of performing complex non-linear transformations, making them well-suited for handling the high-dimensional nature of the Amazon product review dataset. Additionally, the use of embedding layers in neural networks enables the models to capture latent features and relationships between users and items, leading to more accurate predictions [56].

Regarding the top-*n* evaluation, we observed that the results were less favourable. Given the results 5.5, our models do not appear to be doing any better than random[4] for Top-*n* prediction. These results can be possibly attributed to two factors. Firstly, our recommender models were primarily designed for ratings prediction rather than top-*n* recommendations. Effectively, we adapted the models to evaluate top-*n* recommendations, but they were not optimised for this task. Specifically, our recommender models predict ratings, so for each user we get 11 004 predicted ratings (test set). Now, suppose that only 3 products are relevant (have above the specified threshold rating, 3) for a user, on average. Our adaption of top-*n* recommendation is then trying to see if the 3 products' ratings will come up in the top 100 of 3249. This is a tough task given that the models were not optimised for. Had the recommender models been redesigned to optimise for top-*n* recommendations as well, we would expect the results to be very different. Several adjustments would be needed to adapt the current recommender models, such as using a different loss function like Bayesian Personalised Ranking loss, which is designed to optimise for top-*n* recommendations [112].

Secondly, the poor performance of the models in the top-*n* evaluation may be due to the lack of diversity in the recommendations. The models may be recommending the same popular items to most users, leading to low recall and precision scores. This lack of diversity in recommendations is a common issue in collaborative filtering-based recommenders, as they tend to recommend popular items over more diverse or niche items [3]. To address this issue, we can incorporate diversity measures into the recommendation process, such as item diversity or novelty, to ensure that the recommender system is recommending a variety of items to users [138]. This can help to improve the relevance of the recommendations and increase the chances of capturing relevant items in the top-*n* recommendations.

[4]The probability that a random ranking correctly predicts a relevant product in the top 100 is 1-in-100.

Additionally, it may be a consequence of the dataset's characteristics. From Figure 3.4, we identified that the ratings distribution was heavily skewed, with most ratings being 5. This skewed distribution may have impacted the system's ability to recommend diverse or niche items. When most ratings are high, it becomes challenging to differentiate between items. The heavily skewed rating distribution, in which most ratings are 5, poses a challenge to the recommender system for top-*n* recommendations. In such cases, it becomes difficult to differentiate between good items since the predicted ratings are all close to 5. This means that the recommender system may struggle to identify items that stand out from the rest. Consequently, the system may recommend popular, mainstream items over more diverse or niche items that might also be of interest to users. To mitigate the impact of the skewed rating distribution, we can employ several strategies. We could look at incorporating implicit feedback alongside explicit ratings can provide a more comprehensive understanding of user preferences [110]. Moreover, re-weighting an individual user's rating using their review text could provide a more detailed preference measure of their interest. By adjusting the original ratings, we can achieve a more accurate and reflective feedback that takes into account the nuances expressed in the reviews. This approach leverages the qualitative information provided by users, enabling a more personalised and effective recommendation system. This is another approach for incorporating review text into recommenders that has shown promising results [53].

Ultimately, one of the foremost reasons for the poor top-*n* results can be attributed to this analysis being focused on predictive accuracy, a task for which the models performed well, rather than for top-*n* recommendation. Consequently, the poorer results highlighted by Table 5.5 underscore the importance of aligning the purpose of the recommender system with its evaluation criteria. This finding resonates with previous research suggesting that there is no universal best recommendation method. Instead, the success of a recommendation system is contingent upon the context and density of available data, with different methods adapting to particular applications being most likely to excel [90].

5.2.1 Research Questions

We now revisit the research questions posed in Section 1.3 and discuss the implications of our findings in the context of these questions.

1. **How does incorporating product reviews into a recommender system model impact the predictive accuracy and ability to recommend relevant items?**

The inclusion of review text enhanced the performance of the NCF model, as evidenced by a notable improvement in the MAE, MSE and RMSE for rating prediction. While the improvement in the accuracy of the predictions was significant, it is important to note that the impact on the top-*n* evaluation was relatively smaller.

This finding aligns with our hypothesis that leveraging review text alongside ratings data would provide a more comprehensive understanding of user preferences and lead to more accurate recommendations. The results suggest that user reviews contain valuable information that can be utilised to improve the predictive accuracy of recommender systems, especially in tasks such as rating prediction.

2. **How does incorporating review sentiment as well as review text in a recommender system impact the predictive accuracy and ability to recommend relevant items?**

The results between the NCF model with ratings, review text and review sentiments were comparable to those of the NCF model with review and ratings only. There were no substantial improvements observed with the inclusion of sentiments.

These findings suggest that the sentiment analysis technique used in this study may not have provided significant additional value in the context of the problem being addressed. However, it is important to note that this does not rule out the possibility that a more refined sentiment analysis approach could yield better results in future studies. Future work may involve exploring alternative sentiment analysis techniques or refining the existing one to further investigate the potential impact of incorporating review sentiments into recommender systems.

In contrast to our results, several studies in the literature have found that incorporating sentiment analysis into recommender systems can improve their performance ([136]; [116]). This contrast could be due to several factors, such as the specific characteristics of the Amazon product review dataset, as well as the way sentiment analysis was incorporated into the model, or even the model itself - NCF. Moreover, the choice of sentiment analysis was somewhat more sophisticated in the papers mentioned.

3. **How does the performance of the collaborative-based filtering recommender system using neural collaborative filtering compare to that of popular benchmark recommender systems?**

NCF outperformed all benchmarks for both rating prediction and top-*n* evaluation metrics. These results signify the potential of neural architecture or deep learning introduction to enhance the accuracy and effectiveness of recommender systems. Research claims that neural networks can capture complex patterns and relationships in data, which is likely the reason for the superior performance of NCF in this study [56]. Although the results for top-*n* evaluation were less favourable, hypothesis testing confirmed that the differences in performance between NCF and the benchmark models were statistically significant.

The superior performance of NCF over other more traditional recommender systems has been widely reported in previous literature ([56]; [113]). Similarly, other work which have used Amazon product reviews for building recommenders have found that incorporating deep learning-based approached for collaborative filtering have found improved results [115]. Specifically, [115] found that a deep neural network architecture for collaborative filtering outperformed traditional models in predicting review rating scores. Our findings are consistent with these studies, suggesting that neural collaborative filtering models are more effective in capturing complex user-item interactions and generating more accurate rating predictions.

4. **What are the potential trade-offs of incorporating product reviews into a recommender system, such as increased complexity or potential biases in the recommendations?**

Having implemented the solution, we acknowledge that incorporating product reviews into a recommender system introduces additional work, but not necessarily increased complexity. These reviews require additional text pre-processing which was discussed in great detail in Section 3.3.2, and then finally require text embedding. For NCF, review text is integrated into the input layer alongside user and items. These embeddings are then concatenated with the user and item embeddings to form the input vector for the neural network - as discussed in Section 4.1. Within the neural network, the review text is processed alongside the user and item embeddings to generate the final rating prediction. This process is not necessarily more complex than the existing models.

To explore the computational runtimes associated with incorporating reviews compared to those models that were built without reviews, we monitored the training run times. Table 5.6 provides a comparison of the run-times of various collaborative filtering models. The benchmark models show higher runtimes compared to all the NCF model instances, indicating that they may require more computational resources compared to the NCF model. This is likely due to the fact that these benchmark models were built from scratch using Python, while the NCF model was built using TensorFlow, which is optimised for deep learning tasks. Additionally, incorporating the reviews into the recommender model did increase the run time of the NCF model from 12 minutes to 13 minutes. Moreover, we note that including sentiments as well as reviews to the NCF model did not increase the computation run time compared to using reviews and ratings only.

Overall, incorporating product reviews into a recommender system, appears to not greatly impact the runtime of the model. This is likely due to the additional complexity of incorporating reviews is not significantly higher than the existing models. Furthermore, NCF models in general, even with the additional augmented review and sentiments data run faster (13 minutes) than the quickest benchmark model - UBCF at 125 minutes. Again, this is largely due to the fact that the NCF model is built using TensorFlow, which is optimised for deep learning tasks, whilst the benchmark models were built from scratch using Python.

Model	**Data Used**	**Runtime (minutes)**
Item-Based Collaborative Filtering	Ratings Only	140
User-Based Collaborative Filtering	Ratings Only	125
Non-Negative Matrix Factorisation	Ratings Only	457
Neural Collaborative Filtering 1	Ratings Only	12
Neural Collaborative Filtering 2	Ratings & Reviews	13
Neural Collaborative Filtering 3	Ratings, Reviews, Sentiments	13

Table 5.6: Computational Details of Various Collaborative Filtering Models

5.3 Conclusion

In this chapter, we presented the outcomes of our various recommender models and their implications within the context of our research questions, contrasting them with existing literature on review-aware recommenders. Effectively, this chapter is the culmination of the literature review (Chapter 2), data exploration (Chapter 3), and methodological approach (Chapter 4).

Most notably, the NCF model outperformed all the benchmark models in terms of both predictive accuracy and top-*n* evaluation. Our findings closely align with existing research on incorporating deep learning-based techniques into the collaborative filtering network [56], where there has been some significant attention and improvements discovered. Additionally, one-sampled paired t-tests revealed that all improvements were statistically significant at a significance level of $p < 0.01$ for predictive accuracy results. Moreover, we note that the review-aware NCF model augmented with sentiments did not exhibit significant performance improvements in terms of predictive accuracy. Additionally, top-*n* evaluation results were discussed in detail, focusing on addressing the less favourable results observed across all the recommenders. We emphasised the importance of tailoring a recommender for a specific task. Since our primary concern was predictive accuracy, our recommenders yielded poor results in top-*n* evaluation - often not performing better than random. Strategies for improvement were outlined, and additional factors contributing to the poor results were also addressed - namely the skew rating distribution. Effectively, this chapter highlights the importance of incorporating reviews into recommender systems, particularly through the review-aware NCF model, which provided evidence that incorporating reviews improves predictive accuracy. Although there was no substantial improvement with the inclusion of sentiments, the overall performance of the NCF model suggests promising opportunities for future exploration and optimisation.

Importantly, we have addressed each of the four research questions posed in Section 1.2. To summarise, incorporating reviews into a recommender model does lead to an improvement in predictive accuracy. Incorporating sentiments, did not yield improved results, however there is ample opportunities for further exploration, especially in choosing more sophisticated sentiment extraction techniques. The NCF model consistently outperformed all the benchmark models, showing improvements in both predictive accuracy and top-*n* evaluation, suggesting that the deep learning-based approach is more effective in capturing complex user-item interactions than the traditional collaborative filtering methods. Finally, incorporating reviews into a recommender system did not yield large increases in the computational complexity of the models, evidenced by the marginal increase in run times for the NCF model.

Chapter 6

Conclusion

In Chapter 5, we looked into the results from this thesis and their implications concerning our research questions, including a deeper discussion of what was observed from our analysis in general. In this chapter, we aim to offer a comprehensive overview of the primary findings derived from this thesis, representing the culmination of our analysis, and outlining their implications moving forward.

The chapter begins with an overview of the results in Section 6.1. Following this, Section 6.2 addresses the limitations of the thesis, both from a perspective of limited experimentation, as well as from a perspective of the various assumptions made. These identified shortcomings pave the way for Section 6.3, which explores potential avenues for further exploration or enhancement of the methodologies employed. Finally, we draw this thesis to a close with Section 6.4, which serves to summarise and raise key points to conclude this thesis.

6.1 Summary of Results and Findings

In Section 5.1, we evaluated the results from our study for two problems: rating prediction and top-*n* generation. We summarise the key findings from our study.

Rating Prediction

- The NCF model outperformed all benchmark models across all predictive accuracy metrics, including MAE, RMSE, and MSE.
- The review-aware NCF model, integrating ratings and review text, exhibited the best performance, achieving an MAE of 0.490 and RMSE of 0.769.
- The inclusion of sentiment analysis alongside reviews marginally decreased performance, resulting in marginally higher RMSE and MAE values (MAE of 0.492 and RMSE of 0.779).
- A one-sample paired t-tests showed that there is evidence that the NCF model significantly outperforms all benchmark models at a significance level of $p < 0.01$.
- Users with more reviews had lower MAE and RMSE values, indicating that the NCF model performed better for users with more reviews.
- The predictive accuracy did not change significantly for items with more reviews, indicating that the NCF model performed similarly for items with varying review counts.
- Reviews with fewer words had lower MAE and RMSE values, indicating that the NCF model performed better for reviews with fewer words.

Top-*N* Generation

- The NCF model struggled to recommend relevant products for users' top-*n* recommendations, particularly within the Top-100 list.
- All models performed poorly on the top-*n* tasks, although the NCF model outperformed the benchmarks models.

Computational Details

- The NCF model with review text and sentiments required only 13 minutes for model training

- The slowest NCF model, NCF model with reviews, was quicker (13 minutes) than the fastest benchmark model, user-based collaborative filtering (125 minutes).
- NMF took the longest runtime at 457 minutes.
- The efficiency of the NCF models was particularly noteworthy with respect to computation, with shorter run-times compared to benchmark models, even when augmented with additional information such as review text and sentiments. Largely attributed to using TensorFlow for NCF, whilst the benchmark models were built from scratch.

Directly addressing our research questions, the incorporation of review text enhanced rating prediction performance within recommender systems. However, the addition of sentiments did not yield further improvements. Moreover, the efficacy of the NCF models was evident, performing better than all the benchmark models across the predictive accuracy metrics used. Furthermore, the efficiency of the NCF models was particularly noteworthy with respect to computation, with shorter run-times compared to benchmark models, even when augmented with additional information such as review text and sentiments.

6.2 Limitations

In building our recommender system, our primary focus centered on rating predictions aimed at minimising the disparity between actual and predicted ratings. The overarching goal was to develop a recommendation system capable of accurately predicting unknown ratings for items, leveraging a training set to discern user preferences, and assessing the system's efficacy by evaluating its performance on a testing set containing concealed ratings provided by users for various items. This evaluation process involved measuring predictive accuracy metrics such as RMSE and MAE. Having achieved predicted ratings for all unrated items for each user we were able to identify a list of items (top-n) by sorting the ratings for unrated items to identify a top-n list of recommendations for each user. We measured this list using metrics such as recall@n and precision@n. The results of which were not impressive. In hindsight, these observations shed light on several limitations inherent in our study, prompting a critical re-evaluation of our methodologies and approaches.

Firstly, we built the model for a specific goal - rating prediction, however we tested its capability on different task - top-n generation. We have established that there is no one universal recommender system [90] and that a recommender's performance is not transferable from one objective to another. Consequently, the results obtained from the top-n generation task offer limited insights into the effectiveness of our review-aware NCF model in this specific use case. Indeed, while our recommender system performed relatively well in predicting ratings for unknown products, its performance in generating top-n recommendations was considerably less satisfactory. That is to say, we observed that the model struggled to recommend relevant (items in test set) products for a users Top-100. In fact, the performance was not better than random recommender. Thus, given the misalignment between the model's training objective and the evaluation task, the results obtained from the top-n generation task should be interpreted with caution, and the conclusions drawn from them should be viewed in light of this limitation.

To that end, the misalignment between the training objective and the evaluation task underscores the importance of aligning the recommender system's training and evaluation objectives to ensure that the model is optimised for the task at hand. With the evolving understanding that recommendation accuracy alone does not guarantee an effective and satisfying user experience, our approach of generating a recommender solely for predicting unrated items provides a limited scope for building a good or useful recommender system. To address this limitation, it becomes imperative to extend beyond simple accuracy metrics and optimise recommenders for tasks beyond rating prediction. This necessitates a more nuanced evaluation approach that encompasses multiple dimensions of performance, rather than focusing solely on rating prediction accuracy. Such multi-objective recommenders can be designed to optimise for a range of criteria, including diversity, novelty, serendipity, and user satisfaction, thereby ensuring a more comprehensive recommendation capability.

Another key challenge and limitation encountered in this study pertained to the computational resources allocated to meet the thesis's requirements. For loading and handling the dataset, we relied on our local machine, which presented several challenges in managing the data effectively and efficiently. Thus, we limited our dataset by sampling records from the original repository, which contained over 140 million records. Additionally, we further restricted the dataset to users with 13 or more ratings and items with 13 or more reviews. While these constraints were necessary to ensure the feasibility of our analysis on the local machine, they also introduced limitations that may have impacted the performance and utility of our recommender system.

With respect to the computational restrictions, beyond dataset handling, the model building and training occurred on our local machine, which inhibited the path of additional experimentation. Despite our efforts, all the benchmark models were restricted (due to their excessive run-times and memory allocations) in their tuning capabilities, with hyperparameter adjustments limited to only two or three options for each parameter. While it is not expected that the performance will be substantially improved had we been able to perform a more extensive hyperparameter search, it could be worthwhile trying to quantify how much improvement can be gained by performing a large hyperparameter search as compared to using a standard set of hyperparameters. Additionally, we compared an NCF model with extensive hyperparameter tuning with benchmark models that had minimal hyperparameter tuning. This comparison could have been more equitable had we performed a more extensive hyperparameter search for the benchmark models. Ultimately, the computational constraints as well as the limited time dictated a lot of the decisions made in the methodology. This challenge also touches upon a broader issue of scalability, an aspect that was acknowledged in Section 2.7.3 but was not addressed within the scope of this thesis. Nonetheless, given the context of the recommender system within e-commerce, scalability emerges as a primary concern. Addressing this concern could significantly enhance the run-times of benchmark models, particularly matrix factorisation approaches, by leveraging packages specifically designed to handle the inherent sparsity of the user-item matrix. The benchmark models, detailed in Section 4.4, were constructed from scratch in Python, with minimal effort directed towards mitigating scalability issues or managing the computational overhead of training.

One final limitation inherent in our thesis is the omission of addressing the cold start problem. We chose to mitigate the cold start problem, or rather circumvent it entirely, by restricting the dataset to users with 13 or more ratings and items with 13 or more reviews. While this approach provides a starting point, it limits the generalisability of our findings and the applicability of our recommender system to real-world scenarios. The cold start problem is a pervasive challenge in recommender systems, particularly for new users or items with limited interaction history. Addressing this challenge requires the development of innovative strategies to handle users or items with limited historical data. While the cold start problem was not the primary focus of our study, it represents a critical limitation that warrants further exploration and consideration in future research.

6.3 Future Work

6.3.1 Developing Multi-Objective Recommender Systems

We have stablished the importance of a comprehensive evaluation approach to recommender systems, one that extends beyond rating prediction accuracy to encompass a broader range of performance metrics. Avenues for developing multi-objective recommenders that optimise for diverse criteria, including diversity, novelty, serendipity, and user satisfaction, represent a promising direction for future research.

By adopting a more holistic evaluation approach, recommender systems can be designed to cater to a wider range of user preferences and needs, thereby enhancing the overall user experience. This approach aligns with the evolving understanding that recommendation accuracy alone does not guarantee an effective and satisfying user experience, underscoring the need for a more nuanced evaluation framework that captures the multifaceted nature of recommender systems. Various strategies can be employed to address this, including direct enhancement of recommendation list diversity and the integration of hybrid recommendation methods to meet different task objectives ([135];[159];[66];[158]). Ultimately, future endeavors should prioritise the adoption of a more comprehensive approach to developing multi-objective recommender systems,and hence, evaluating recommenders based on a broader range of performance metrics.

6.3.2 Enhancing Neural Collaborative Filtering with NeuMF

The development of our NCF was based off the framework established by [56]. The findings from our analysis underscored the improvements in predictive accuracy achieved by the NCF model when compared to conventional collaborative filtering methods. The work in this thesis can be extended to hybridising the architecture of NCF by incorporating matrix factorisation, resulting in the algorithm Neural Matrix Factorisation (NeuMF) - as detailed by [56]. The rationale behind this approach stems from the recognition that traditional matrix factorisation can be viewed as a specialised instance of NCF. Therefore, by fusing the neural interpretation of matrix factorisation with Multilayer Perceptron (MLP), NeuMF emerges as a more generalised model harnessing the linearity of matrix factorisation and the non-linearity of MLP to enhance recommendation quality. Notably, the empirical findings from studies such as [153] and [56] corroborate the performance benefits offered by NeuMF over NCF. This naturally extends our comparative analysis, inviting us to integrate and adapt the advancements put forth by [56] to incorporate NeuMF into our framework. This avenue for future can further be extended by investigating the augmentation of review text and sentiments within this enhanced framework — an area that, to the best of our knowledge, has not yet been explored.

6.3.3 Exploring Advanced Word Embedding and Sentiment Analysis Techniques

For our analysis, we employed a simple word embedding technique (USE) to convert review text into numerical vectors, which were subsequently integrated into our NCF model. However, the choice of word embedding technique can significantly impact the performance of the recommender system [8], with advanced methods such as Word2Vec, GloVe, and BERT offering more sophisticated representations of textual data ([96]; [107]; [42]). Therefore, it stands to reason that exploring additional word embedding techniques beyond the simple implementation undertaken in this thesis would represent a natural extension of our research efforts. Similarly, the sentiment analysis component of our study was based on a lexicon-based approaches only, which may not capture the full complexity and nuances of user sentiments. By incorporating more advanced sentiment analysis techniques, such as deep learning-based models, we can possibly enhance the accuracy and granularity of sentiment analysis within our recommender system. Notably, the choice of sentiment analysis technique has been documented as an important determinant in feature creation for subsequent analysis in existing literature [5].

Ultimately, the integration of advanced word embedding and more sophisticated sentiment analysis techniques can enrich the insights derived from user reviews, thereby enhancing the effectiveness of the recommender system. Such an approach can perhaps lead to more representative sentiments from the review text, thereby furnishing the model with richer insights into user preferences. Thus, by experimenting with more methodologies, there is an opportunity to further enhance the effectiveness of our recommender system.

6.3.4 Addressing the Cold Start Problem

A natural extension to our analysis is to address the cold start problem, a challenge in recommender systems that arises when new users or items with limited interaction history are introduced. While we circumvented this issue by restricting our dataset to users with 13 or more ratings and items with 13 or more reviews, this approach limits the generalisability of our findings. To address the cold start problem, innovative strategies can be employed to handle users or items with limited historical data, thereby enhancing the robustness and versatility of the recommender system. For instance, hybrid recommendation approaches that combine collaborative filtering with content-based filtering can be employed to mitigate the cold start problem by leveraging the strengths of both methods. Content-based filtering methods can be leveraged to provide recommendations based on item attributes or user profiles, thereby circumventing the need for historical interaction data [32]. By addressing the cold start problem, the developed recommender systems can cater to a wider range of users and items, thereby enhancing their utility and effectiveness in real-world scenarios. This avenue for future research represents a critical step towards developing more comprehensive and generalisable recommender systems capable of handling the challenges posed by the cold start problem.

6.3.5 Leveraging Additional Data Modalities

The Amazon Product Review Data (Section 3.1) utilised in this study presents additional opportunities for experimentation. Notably, each product within the dataset is accompanied by an image feature — an input that has garnered attention in research within several industries, particularly fashion E-commerce, where images have been leveraged to enrich the top-*n* generation process ([145];[77]). By augmenting our recommender system to incorporate an additional data modality, such as images, there is opportunity for enhancing the efficacy of the model, building upon the foundation laid in this thesis. Furthermore, the dataset offers a valuable temporal dimension, with each review provided by a user being timestamped, spanning a considerable time frame from 1996 to 2018. Harnessing this temporal aspect and tracking trends in user preferences over time could furnish the recommender system with valuable insights, enabling it to adapt and evolve in response to shifting user preferences. This not only facilitates a more comprehensive evaluation of the recommender system but also ensures its relevance and efficacy under realistic and dynamic conditions.

6.4 Conclusion

Recommender systems play a crucial role in modern digital landscapes, acting as indispensable tools that guide users towards items they are likely to find valuable based on their individual preferences and the vast array of available options [68]. By doing so, these systems empower users to efficiently navigate through many options, connecting them with products, services, or knowledge that resonate with their needs and interests [68]. This ability to streamline the decision-making process has large implications for businesses, ranging from increased cross-selling opportunities to heightened customer satisfaction, ultimately translating into tangible gains in revenue and market competitiveness [81].

Collaborative filtering stands as one of the cornerstone paradigms within the realm of recommender systems [19]. At its core, collaborative filtering operates on the principle of leveraging user-generated ratings and feedback to generate personalised recommendations. By harnessing the collective 'wisdom' (history) of users who have rated items they've interacted with, this technique effectively identifies patterns and similarities among user preferences [19]. This approach, traditionally involved calculating and using similarity metrics, or perhaps matrix factorisation methods to build models to predict interest. However, there has been a growing interest in adapting traditional collaborative filtering framework to incorporate deep learning-based approaches - one such approach is neural collaborative filtering [56].

The focal point of this thesis was centered on harnessing a neural network-based approach for collaborative filtering, NCF, with the objective of adapting the framework provided by [56] to accommodate nuanced user preferences by incorporating review text and sentiments. The primary goals of the thesis were to assess whether the incorporation of a neural network architecture in collaborative filtering enhances performance and whether augmenting a recommender with review text and sentiments yields similar improvements. Leveraging the Amazon product review dataset, extensive text cleaning and pre-processing, including word embedding and sentiment analysis, were conducted to prepare the data for integration into our neural architectures within the recommender models. Adhering to the widely adopted leave-*k*-out approach, we partitioned our data and designated 3 items from each user to form the test set. Our evaluation involved comparing the performance of our NCF systems with that of traditional collaborative filtering methods, namely IBCF, UBCF, as well as NMF. All recommender models were constructed and trained for rating prediction task, with the NCF model featuring a relatively simple MLP neural architecture augmented with additional layers to accommodate additional data inputs. Evaluation metrics such as MAE and RMSE were employed to assess the models' rating prediction capabilities. Additionally, we leveraged predicted ratings to generate top-*n* recommendations for each user and evaluated the recommender systems' effectiveness using recall@*n* and precision@*n* metrics.

The outcomes of this study underscored that incorporating review text into the NCF model enhances its predictive accuracy, with the NCF model with reviews and ratings outperforming all benchmark models across all predictive accuracy metrics. Incorporating sentiments alongside review text did not yield further improvements, with the sentiment-augmented NCF model exhibiting marginally higher RMSE and MAE values. For the top-*n* generation task, all the models struggled to recommend relevant products for users within the Top-100 list, however, the NCF models outperformed the benchmark models for this task. Additionally, we found that incorporating sentiments alongside review text did not increase the runtimes of the models significantly.

This thesis concluded by addressing both the limitations inherent in the study and outlined potential avenues for future research that could build upon the current efforts. One of the primary limitations identified was the misalignment between the training objective and the evaluation task, which underscored the importance of aligning the recommender system's training and evaluation objectives to ensure that the model is optimised for the task at hand. To that end, potential avenues for future research include the development of multi-objective recommender systems. In addition, we addressed possibly enhancing the NCF model with NeuMF, exploring advanced word embedding and sentiment analysis techniques, addressing the cold start problem, and leveraging additional data modalities. These avenues represent promising directions for future research that could build upon the current study's findings and enhance the effectiveness and utility of recommender systems in real-world scenarios.

In culmination, this thesis aimed to make incremental contributions to the evolving landscape of recommender systems, with a specific focus on harnessing textual information to augment recommendation accuracy. While our endeavor sought to pave the way for enhanced recommendation accuracy, we acknowledge the hurdles and constraints inherent in crafting recommendation systems capable of adeptly catering to user preferences and item characteristics. The avenues for future work lay bare and offer great potential for expanding the scope of our endeavors and enriching recommendation algorithms.

Appendix A

Appendix A

Group 1	Group 2	Difference	Lower Bound	Upper Bound	Null
Low	Medium	0.013	-0.091	0.001	True
Low	High	0.046	0.031	0.061	False
Low	Very High	0.091	0.081	0.114	False
Medium	High	0.033	-0.040	0.052	True
Medium	Very High	0.078	-0.004	0.098	True
High	Very High	0.045	-0.001	0.052	True

Table A.1: Tukey HSD test results for users.

Group 1	Group 2	Difference	Lower Bound	Upper Bound	Null
Low	Medium	-0.181	-0.214	0.001	True
Low	High	-0.339	-0.349	0.319	False
Low	Very High	-0.420	-0.464	-0.391	False
Medium	High	-0.158	-0.168	0.004	True
Medium	Very High	-0.239	-0.241	-0.219	False
High	Very High	-0.081	-0.008	0.094	True

Table A.2: Tukey HSD test results for review lengths.

Appendix B

Appendix B

The code for this thesis can be found at the link: https://github.com/pavsingh7/Masters-Dissertation.

This repository contains all the Python code used to generate and build the recommender models. It also includes notebooks for the data loading, preprocessing, feature engineering, and data exploration. The actual data used in this thesis is not included in the repository due to its size, however, we provide the scripts to load, preprocess and clean the dataset.

The data used in this thesis is the Amazon Product Reviews dataset. The dataset can be found at the link: https://cseweb.ucsd.edu/~jmcauley/datasets/amazon_v2/. The dataset is available in multiple categories and can be downloaded in JSON format. We chose to download and use the "5-core" datasets for each category available on the page. Cumulatively (summing up across all categories), there are just over 75 million reviews.

Bibliography

[1] Hervé Abdi, Dominique Valentin, and Betty Edelman. *Neural networks*. Number 124. Sage, 1999.

[2] Panagiotis Adamopoulos. Beyond rating prediction accuracy: on new perspectives in recommender systems. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 459–462, 2013.

[3] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749, 2005.

[4] Basant Agarwal, Namita Mittal, Basant Agarwal, and Namita Mittal. Machine learning approach for sentiment analysis. *Prominent feature extraction for sentiment analysis*, pages 21–45, 2016.

[5] Ravinder Ahuja, Aakarsha Chug, Shruti Kohli, Shaurya Gupta, and Pratyush Ahuja. The impact of features extraction on the sentiment analysis. *Procedia Computer Science*, 152:341–348, 2019.

[6] Markos Aivazoglou, Antonios O Roussos, Dionisis Margaris, Costas Vassilakis, Sotiris Ioannidis, Jason Polakis, and Dimitris Spiliotopoulos. A fine-grained social network recommender system. *Social Network Analysis and Mining*, 10:1–18, 2020.

[7] H Alzoubi, M Alshurideh, B al Kurdi, K Alhyasat, and T Ghazal. The effect of e-payment and online shopping on sales growth: Evidence from banking industry. *International Journal of Data and Network Science*, 6(4):1369–1380, 2022.

[8] Deepak Suresh Asudani, Naresh Kumar Nagwani, and Pradeep Singh. Impact of word embedding models on text analytics in deep learning environment: a review. *Artificial intelligence review*, 56(9):10345–10425, 2023.

[9] Marko Balabanović and Yoav Shoham. Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997.

[10] David Bawden and Lyn Robinson. Information overload: An overview. 2020.

[11] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, 2007.

[12] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

[13] Dmitry Bogdanov, Martín Haro, Ferdinand Fuhrmann, Anna Xambó, Emilia Gómez, and Perfecto Herrera. Semantic audio content-based music recommendation and visualization based on user preference examples. *Information Processing & Management*, 49(1):13–33, 2013.

[14] John S Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. *arXiv preprint arXiv:1301.7363*, 2013.

[15] Erik Brynjolfsson, Yu Hu, and Michael D Smith. Consumer surplus in the digital economy: Estimating the value of increased product variety at online booksellers. *Management science*, 49(11):1580–1596, 2003.

[16] Anna Buczak, John Zimmerman, and Kaushal Kurapati. Personalization: Improving ease-of-use, trust and accuracy of a tv show recommender. 2002.

[17] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12:331–370, 2002.

[18] Robin Burke. Hybrid web recommender systems. *The adaptive web: methods and strategies of web personalization*, pages 377–408, 2007.

[19] Robin Burke, Michael P O'Mahony, and Neil J Hurley. Robust collaborative recommendation. *Recommender systems handbook*, pages 961–995, 2015.

[20] Erion Çano and Maurizio Morisio. Hybrid recommender systems: A systematic literature review. *Intelligent data analysis*, 21(6):1487–1524, 2017.

[21] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*, 2018.

[22] Nguyen Ngoc Chan, Walid Gaaloul, and Samir Tata. A web service recommender system using vector space model and latent semantic indexing. In *2011 IEEE International conference on advanced information networking and applications*, pages 602–609. IEEE, 2011.

[23] Guanliang Chen and Li Chen. Augmenting service recommender systems by incorporating contextual opinions from user reviews. *User Modeling and User-Adapted Interaction*, 25:295–329, 2015.

[24] Hung-Wei Chen, Yi-Leh Wu, Maw-Kae Hor, and Cheng-Yuan Tang. Fully content-based movie recommender system with feature extraction using neural network. In *2017 International conference on machine learning and cybernetics (ICMLC)*, volume 2, pages 504–509. IEEE, 2017.

[25] Li Chen, Guanliang Chen, and Feng Wang. Recommender systems based on user reviews: the state of the art. *User Modeling and User-Adapted Interaction*, 25:99–154, 2015.

[26] Li Chen and Pearl Pu. Critiquing-based recommenders: survey and emerging trends. *User Modeling and User-Adapted Interaction*, 22:125–150, 2012.

[27] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

[28] Vladimir Cherkassky, Jerome H Friedman, and Harry Wechsler. *From statistics to neural networks: theory and pattern recognition applications*, volume 136. Springer Science & Business Media, 2012.

[29] Kwok-Wai Cheung, James T Kwok, Martin H Law, and Kwok-Ching Tsui. Mining customer product ratings for personalized marketing. *Decision Support Systems*, 35(2):231–243, 2003.

[30] Yung-Hsin Chien and Edward I George. A bayesian model for collaborative filtering. In *AISTATS*. Citeseer, 1999.

[31] John W Clark, Thomas Lindenau, and Manfred L Ristig. Scientific applications of neural nets. *Scientific Applications of Neural Nets*, 522, 1999.

[32] Mark Claypool, Anuja Gokhale, Tim Miranda, Paul Murnikov, Dmitry Netes, and Matthew Sartin. Combing content-based and collaborative filters in an online newspaper. In *Proc. of Workshop on Recommender Systems-Implementation and Evaluation*, 1999.

[33] Andrew Collins and Joeran Beel. Document embeddings vs. keyphrases vs. terms for recommender systems: a large-scale online evaluation. In *2019 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pages 130–133. IEEE, 2019.

[34] McKinsey & Company. The value of getting personalization right—or wrong—is multiplying. June 2021. Accessed on 17 Dec 2023.

[35] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

[36] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 39–46, 2010.

[37] Joao Felipe Guedes da Silva, Natanael Nunes de Moura Junior, and Luiz Pereira Caloba. Effects of data sparsity on recommender systems based on collaborative filtering. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018.

[38] Cach N Dang, María N Moreno-García, and Fernando De la Prieta. An approach to integrating sentiment analysis into recommender systems. *Sensors*, 21(16):5666, 2021.

[39] Nhan Cach Dang, María N Moreno-García, and Fernando De la Prieta. Sentiment analysis based on deep learning: A comparative study. *Electronics*, 9(3):483, 2020.

[40] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296, 2010.

[41] Christian Desrosiers and George Karypis. A comprehensive survey of neighborhood-based recommendation methods. *Recommender systems handbook*, pages 107–144, 2010.

[42] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[43] Qiming Diao, Minghui Qiu, Chao-Yuan Wu, Alexander J Smola, Jing Jiang, and Chong Wang. Jointly modeling aspects, ratings and sentiments for movie recommendation (jmars). In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 193–202, 2014.

[44] Gonzalo I Diaz, Achille Fokoue-Nkoutche, Giacomo Nannicini, and Horst Samulowitz. An effective algorithm for hyperparameter optimization of neural networks. *IBM Journal of Research and Development*, 61(5):9–1, 2017.

[45] Lise Getoor, Mehran Sahami, et al. Using probabilistic relational models for collaborative filtering. In *Workshop on Web Usage Analysis and User Profiling (WEBKDD'99)*, pages 1–6, 1999.

[46] Rayid Ghani and Andrew Fano. Building recommender systems using a knowledge base of product semantics. In *Proceedings of the Workshop on Recommendation and Personalization in ECommerce at the 2nd International Conference on Adaptive Hypermedia and Adaptive Web based Systems*, pages 27–29. Citeseer, 2002.

[47] Sandesh Gharatkar, Aakash Ingle, Tanmay Naik, and Ashwini Save. Review preprocessing using data cleaning and stemming technique. In *2017 international conference on innovations in information, embedded and communication systems (ICIIECS)*, pages 1–4. IEEE, 2017.

[48] David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[49] Carlos A Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):1–19, 2015.

[50] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[51] Kevin Gurney. *An introduction to neural networks*. CRC press, 2018.

[52] Tanjim Ul Haque, Nudrat Nawal Saber, and Faisal Muhammad Shah. Sentiment analysis on large scale amazon product reviews. In *2018 IEEE international conference on innovative research and development (ICIRD)*, pages 1–6. IEEE, 2018.

[53] Negar Hariri, Bamshad Mobasher, Robin Burke, and Yong Zheng. Context-aware recommendation based on review mining. In *ITWP@ IJCAI*, 2011.

[54] Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*, pages 507–517, 2016.

[55] Xiangnan He, Tao Chen, Min-Yen Kan, and Xiao Chen. Trirank: Review-aware explainable recommendation by modeling aspects. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 1661–1670, 2015.

[56] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[57] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 549–558, 2016.

[58] Jon Herlocker, Joseph A Konstan, and John Riedl. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Information retrieval*, 5:287–310, 2002.

[59] Jonathan L Herlocker, Joseph A Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 230–237, 1999.

[60] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[61] María Hernández-Rubio, Iván Cantador, and Alejandro Bellogín. A comparative analysis of recommender systems based on item aspect opinions extracted from user reviews. *User Modeling and User-Adapted Interaction*, 29(2):381–441, 2019.

[62] Will Hill, Larry Stead, Mark Rosenstein, and George Furnas. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 194–201, 1995.

[63] Longke Hu, Aixin Sun, and Yong Liu. Your neighbors affect your ratings: on geographical neighborhood influence to rating prediction. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 345–354, 2014.

[64] Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177, 2004.

[65] Zan Huang, Hsinchun Chen, and Daniel Zeng. Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, 22(1):116–142, 2004.

[66] Neil Hurley and Mi Zhang. Novelty and diversity in top-n recommendation–analysis and evaluation. *ACM Transactions on Internet Technology (TOIT)*, 10(4):1–30, 2011.

[67] Clayton Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of the international AAAI conference on web and social media*, volume 8, pages 216–225, 2014.

[68] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender systems: an introduction*. Cambridge University Press, 2010.

[69] Zhiyang Jia, Yuting Yang, Wei Gao, and Xu Chen. User-based collaborative filtering for tourist attraction recommendations. In *2015 IEEE international conference on computational intelligence & communication technology*, pages 22–25. IEEE, 2015.

[70] Kapil Kaushik, Rajhans Mishra, Nripendra P Rana, and Yogesh K Dwivedi. Exploring reviews and review sequences on e-commerce platform: A study of helpful reviews on amazon. in. *Journal of retailing and Consumer Services*, 45:21–32, 2018.

[71] Jacqueline Kazmaier and Jan H Van Vuuren. The power of ensemble learning in sentiment analysis. *Expert Systems with Applications*, 187:115819, 2022.

[72] Donghyun Kim, Chanyoung Park, Jinoh Oh, Sungyoung Lee, and Hwanjo Yu. Convolutional matrix factorization for document context-aware recommendation. In *Proceedings of the 10th ACM conference on recommender systems*, pages 233–240, 2016.

[73] Joseph A Konstan, Bradley N Miller, David Maltz, Jonathan L Herlocker, Lee R Gordon, and John Riedl. Grouplens: Applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[74] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434, 2008.

[75] Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 447–456, 2009.

[76] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[77] Zühal Kurt and Kemal Özkan. An image-based recommender system based on feature extraction techniques. In *2017 International Conference on Computer Science and Engineering (UBMK)*, pages 769–774. IEEE, 2017.

[78] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[79] Daniel Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. *Advances in neural information processing systems*, 13, 2000.

[80] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, pages 609–616, 2009.

[81] Juha Leino and Kari-Jouko Raiha. Case amazon: ratings and reviews as part of recommendations. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 137–140, 2007.

[82] Frank Hung-Fat Leung, Hak-Keung Lam, Sai-Ho Ling, and Peter Kwong-Shun Tam. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE Transactions on Neural networks*, 14(1):79–88, 2003.

[83] Blerina Lika, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. Facing the cold start problem in recommender systems. *Expert systems with applications*, 41(4):2065–2073, 2014.

[84] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

[85] Gregory D Linden, Jennifer A Jacobi, and Eric A Benson. Collaborative recommendations using item-to-item similarity mappings, July 24 2001. US Patent 6,266,649.

[86] Jin-Hu Liu, Tao Zhou, Zi-Ke Zhang, Zimo Yang, Chuang Liu, and Wei-Min Li. Promoting cold-start items in recommender systems. *PloS one*, 9(12):e113457, 2014.

[87] Qidong Liu, Jiaxi Hu, Yutian Xiao, Jingtong Gao, and Xiangyu Zhao. Multimodal recommender systems: A survey. *arXiv preprint arXiv:2302.03883*, 2023.

[88] Stanley Loh, Fabiana Lorenzi, Ramiro Saldaña, and Daniel Licthnow. A tourism recommender system based on collaboration and text analysis. *Information Technology & Tourism*, 6(3):157–165, 2003.

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

[90] Linyuan Lü, Matúš Medo, Chi Ho Yeung, Yi-Cheng Zhang, Zi-Ke Zhang, and Tao Zhou. Recommender systems. *Physics reports*, 519(1):1–49, 2012.

[91] Kasra Madadipouya and Sivananthan Chelliah. A literature review on recommender systems algorithms, techniques and evaluations. *BRAIN. Broad Research in Artificial Intelligence and Neuroscience*, 8(2):109–124, 2017.

[92] XU Manfei, Drew Fralick, Julia Z Zheng, Bokai Wang, FENG Changyong, et al. The differences and similarities between two-sample t-test and paired t-test. *Shanghai archives of psychiatry*, 29(3):184, 2017.

[93] Benjamin M Marlin. Modeling user rating profiles for collaborative filtering. *Advances in neural information processing systems*, 16, 2003.

[94] Julian McAuley and Jure Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 165–172, 2013.

[95] Walaa Medhat, Ahmed Hassan, and Hoda Korashy. Sentiment analysis algorithms and applications: A survey. *Ain Shams engineering journal*, 5(4):1093–1113, 2014.

[96] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[97] C-C Musat, Yizhong Liang, and Boi Faltings. Recommendation using textual opinions. In *IJCAI International Joint Conference on Artificial Intelligence*, number CONF, pages 2684–2690, 2013.

[98] Cataldo Musto, Giovanni Semeraro, Marco De Gemmis, and Pasquale Lops. Word embedding techniques for content-based recommender systems: An empirical evaluation. *Recsys posters*, 1441, 2015.

[99] Finn Årup Nielsen. A new anew: Evaluation of a word list for sentiment analysis in microblogs. *arXiv preprint arXiv:1103.2903*, 2011.

[100] John O'Donovan and Barry Smyth. Trust in recommender systems. In *Proceedings of the 10th international conference on Intelligent user interfaces*, pages 167–174, 2005.

[101] Shumpei Okura, Yukihiro Tagami, Shingo Ono, and Akira Tajima. Embedding-based news recommendation for millions of users. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1933–1942, 2017.

[102] Makbule Gulcin Ozsoy. From word embeddings to item recommendation. *arXiv preprint arXiv:1601.01356*, 2016.

[103] Tulasi K Paradarami, Nathaniel D Bastian, and Jennifer L Wightman. A hybrid recommender system using artificial neural networks. *Expert Systems with Applications*, 83:300–313, 2017.

[104] Rajiv Pasricha and Julian McAuley. Translation-based factorization machines for sequential recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 63–71, 2018.

[105] Arkadiusz Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD cup and workshop*, volume 2007, pages 5–8, 2007.

[106] Michael J Pazzani. A framework for collaborative, content-based and demographic filtering. *Artificial intelligence review*, 13:393–408, 1999.

[107] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[108] David M Pennock, Eric J Horvitz, Steve Lawrence, and C Lee Giles. Collaborative filtering by personality diagnosis: A hybrid memory-and model-based approach. *arXiv preprint arXiv:1301.3885*, 2013.

[109] Yuliana Pérez-Gallardo, Giner Alor-Hernández, Guillermo Cortes-Robles, and Alejandro Rodríguez-González. Collective intelligence as mechanism of medical diagnosis: The ipixel approach. *Expert systems with applications*, 40(7):2726–2737, 2013.

[110] Ladislav Peska and Peter Vojtas. Using implicit preference relations to improve recommender systems. *Journal on Data Semantics*, 6:15–30, 2017.

[111] Al Mamunur Rashid, Istvan Albert, Dan Cosley, Shyong K Lam, Sean M McNee, Joseph A Konstan, and John Riedl. Getting to know you: learning new user preferences in recommender systems. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 127–134, 2002.

[112] Steffen Rendle. Factorization machines. In *2010 IEEE International conference on data mining*, pages 995–1000. IEEE, 2010.

[113] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. Neural collaborative filtering vs. matrix factorization revisited. In *Proceedings of the 14th ACM Conference on Recommender Systems*, pages 240–248, 2020.

[114] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186, 1994.

[115] Mohammad R Rezaei. Amazon product recommender system. *arXiv preprint arXiv:2102.04238*, 2021.

[116] Deepjyoti Roy and Mala Dutta. A systematic review and research perspective on recommender systems. *Journal of Big Data*, 9(1):59, 2022.

[117] M Saravanan, PC Reghu Raj, and S Raman. Summarization and categorization of text data in high-level data cleaning for information retrieval. *Applied Artificial Intelligence*, 17(5-6):461–474, 2003.

[118] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Analysis of recommendation algorithms for e-commerce. In *Proceedings of the 2nd ACM Conference on Electronic Commerce*, pages 158–167, 2000.

[119] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295, 2001.

[120] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Fifth international conference on computer and information science*, volume 1, pages 27–8. Citeseer, 2002.

[121] Badrul Sarwar, George Karypis, Joseph Konstan, and John T Riedl. Application of dimensionality reduction in recommender system-a case study. 2000.

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

[123] J Ben Schafer, Joseph A Konstan, and John Riedl. E-commerce recommendation applications. *Data mining and knowledge discovery*, 5:115–153, 2001.

[124] Rakhi Seth and Aakanksha Sharaff. A comparative overview of hybrid recommender systems: Review, challenges, and prospects. *Data Mining and Machine Learning Applications*, pages 57–98, 2022.

[125] Bracha Shapira, Lior Rokach, and Francessco Ricci. Recommender systems handbook. 2022.

[126] Upendra Shardanand and Pattie Maes. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 210–217, 1995.

[127] Amit Sharma, Jake M Hofman, and Duncan J Watts. Estimating the causal impact of recommendation systems from observational data. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, pages 453–470, 2015.

[128] Rong-Ping Shen, Heng-Ru Zhang, Hong Yu, and Fan Min. Sentiment based matrix factorization with reliability for recommendation. *Expert Systems with Applications*, 135:249–258, 2019.

[129] Babak Maleki Shoja and Nasseh Tabrizi. Customer reviews analysis with deep neural networks for e-commerce recommender systems. *IEEE access*, 7:119121–119130, 2019.

[130] Monika Singh. Scalability and sparsity issues in recommender datasets: a survey. *Knowledge and Information Systems*, 62(1):1–43, 2020.

[131] Nikhil Kumar Singh, Deepak Singh Tomar, and Arun Kumar Sangaiah. Sentiment analysis: a review and comparative analysis over social media. *Journal of Ambient Intelligence and Humanized Computing*, 11(1):97–117, 2020.

[132] SINTEF. Big data, for better or worse: 90last two years. *ScienceDaily*, 2013.

[133] Stephen Skalicky and Scott Crossley. A statistical analysis of satirical amazon. com product reviews. *The European Journal of Humour Research*, 2(3):66–85, 2015.

[134] Brent Smith and Greg Linden. Two decades of recommender systems at amazon. com. *Ieee internet computing*, 21(3):12–18, 2017.

[135] Barry Smyth and Paul McClave. Similarity vs. diversity. In *International conference on case-based reasoning*, pages 347–361. Springer, 2001.

[136] Mehdi Srifi, Ahmed Oussous, Ayoub Ait Lahcen, and Salma Mouline. Recommender systems based on collaborative filtering using review texts—a survey. *Information*, 11(6):317, 2020.

[137] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[138] Harald Steck. Evaluation of recommendations: rating-prediction and ranking. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 213–220, 2013.

[139] Harald Steck, Linas Baltrunas, Ehtsham Elahi, Dawen Liang, Yves Raimond, and Justin Basilico. Deep learning for recommender systems: A netflix case study. *AI Magazine*, 42(3):7–18, 2021.

[140] Panagiotis Symeonidis, Alexandros Nanopoulos, Apostolos N Papadopoulos, and Yannis Manolopoulos. Collaborative recommender systems: Combining effectiveness and efficiency. *Expert Systems with Applications*, 34(4):2995–3013, 2008.

[141] Maite Taboada, Julian Brooke, Milan Tofiloski, Kimberly Voll, and Manfred Stede. Lexicon-based methods for sentiment analysis. *Computational linguistics*, 37(2):267–307, 2011.

[142] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Major components of the gravity recommendation system. *Acm Sigkdd Explorations Newsletter*, 9(2):80–83, 2007.

[143] Poonam B Thorat, Rajeshwari M Goudar, and Sunita Barve. Survey on collaborative filtering, content-based filtering and hybrid recommendation system. *International Journal of Computer Applications*, 110(4):31–36, 2015.

[144] Quoc-Tuan Truong, Aghiles Salah, and Hady Lauw. Multi-modal recommender systems: Hands-on exploration. In *Proceedings of the 15th ACM Conference on Recommender Systems*, pages 834–837, 2021.

[145] Hessel Tuinhof, Clemens Pirker, and Markus Haltmeier. Image-based fashion product recommendation with deep learning. In *Machine Learning, Optimization, and Data Science: 4th International Conference, LOD 2018, Volterra, Italy, September 13-16, 2018, Revised Selected Papers 4*, pages 472–481. Springer, 2019.

[146] Charalampos Vassiliou, Dimitris Stamoulis, and Drakoulis Martakos. A hybrid content-based clustering architecture: minimising uncertainty in personalised multimedia content. *International Journal of Intelligent Systems Technologies and Applications*, 1(3-4):319–345, 2006.

[147] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1235–1244, 2015.

[148] He Weihong and Cao Yi. An e-commerce recommender system based on content-based filtering. *Wuhan University Journal of Natural Sciences*, 11:1091–1096, 2006.

[149] Ian H Witten, Eibe Frank, Mark A Hall, Christopher J Pal, and Mining Data. Practical machine learning tools and techniques. In *Data mining*, volume 2, pages 403–413. Elsevier Amsterdam, The Netherlands, 2005.

[150] Wan-Shiou Yang and San-Yih Hwang. itravel: A recommender system in mobile peer-to-peer environment. *Journal of Systems and Software*, 86(1):12–20, 2013.

[151] Eva Zangerle and Christine Bauer. Evaluating recommender systems: survey and framework. *ACM computing surveys*, 55(8):1–38, 2022.

[152] Sheng Zhang, Weihong Wang, James Ford, and Fillia Makedon. Learning from incomplete ratings using non-negative matrix factorization. In *Proceedings of the 2006 SIAM international conference on data mining*, pages 549–553. SIAM, 2006.

[153] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM computing surveys (CSUR)*, 52(1):1–38, 2019.

[154] Zhenxue Zhang, Dongsong Zhang, and Jianwei Lai. urcf: user review enhanced collaborative filtering. 2014.

[155] Zijun Zhang. Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th international symposium on quality of service (IWQoS)*, pages 1–2. Ieee, 2018.

[156] Vincent W Zheng, Yu Zheng, Xing Xie, and Qiang Yang. Collaborative location and activity recommendations with gps history data. In *Proceedings of the 19th international conference on World wide web*, pages 1029–1038, 2010.

[157] Dengyong Zhou, Olivier Bousquet, Thomas Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. *Advances in neural information processing systems*, 16, 2003.

[158] Tao Zhou, Zoltán Kuscsik, Jian-Guo Liu, Matúš Medo, Joseph Rushton Wakeling, and Yi-Cheng Zhang. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515, 2010.

[159] Cai-Nicolas Ziegler, Sean M McNee, Joseph A Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, pages 22–32, 2005.