University of York

Evaluating the Efficacy of Machine Learning Approaches in Recommendation Systems

Ioannis Pegiadis

Supervisor: Kostantinos Dimopoulos

A report submitted in partial fulfilment of the requirements
for the degree of Master's in Data Science and Artificial Intelligence

in the

Department of Computer Science

December 21, 2023

Declaration

Abstract

In the rapidly evolving domain of recommendation systems, the efficacy of various artificial intelligence (AI) models to accurately predict user preferences is paramount for enhancing user experience. This dissertation presents a comprehensive study of multiple AI models specifically tailored for recommendation systems applied to the MovieLens 100k dataset. The research involved the meticulous design, implementation, and evaluation of several models, taking into account various factors such as model complexity, computational efficiency, and scalability. Through extensive experimentation and cross-validation, the study provides insights into the comparative performance of these models in terms of precision, recall, and other pertinent evaluation metrics. Notably, the findings also highlight the significance of certain model hyperparameters and the role they play in optimizing the recommendation output. By offering an in-depth analysis of the strengths and limitations of each model in the context of the MovieLens 100k dataset, this dissertation serves as a valuable reference for researchers and industry professionals aiming to harness the power of AI in enhancing recommendation system capabilities.

Contents

1 Introduction **1**
1.1 Background . . . 1
1.2 Problem Statement . . . 1
1.3 Objectives . . . 2
1.4 Research Significance . . . 2
1.5 Dissertation Breakdown . . . 2

2 Literature Review **4**
2.1 Artificial Intelligence in Recommendation Systems . . . 4
2.2 Collaborative Filtering . . . 4
2.2.1 User-based Collaborative Filtering . . . 5
2.2.2 Item-based Collaborative Filtering . . . 5
2.3 Content-based Filtering . . . 5
2.3.1 Item Representation . . . 5
2.3.2 User Profile Creation . . . 5
2.3.3 Recommendation Generation . . . 6
2.4 Comparative Studies . . . 6
2.4.1 Collaborative vs. Content-based Filtering . . . 6
2.4.2 Hybrid Approaches . . . 6
2.4.3 Deep Learning in Recommendation . . . 7
2.5 Ethical and Privacy Concerns in Recommendation Systems . . . 7
2.5.1 Data Privacy and Security . . . 7
2.5.2 Ethical Implications . . . 8
2.6 Challenges and Limitations . . . 8
2.6.1 The Cold Start Problem . . . 8
2.6.2 Scalability . . . 8
2.6.3 Data Sparsity . . . 9
2.7 Context-Aware Recommendations . . . 9
2.7.1 Incorporating Context . . . 9
2.7.2 Real-world Applications . . . 9
2.8 Case Studies . . . 10
2.8.1 Industry Applications . . . 10
2.8.2 Success Stories and Lessons Learned . . . 10
2.9 Emerging Trends and Future Directions . . . 11
2.9.1 Technological Advancements . . . 11

2.9.2 Predictive Analysis and Personalization . . . 11
2.10 Tools and Methodologies for Model Development . . . 11
2.10.1 Python: The Core Programming Language . . . 12
2.10.2 Scikit-learn: Machine Learning Library . . . 12
2.10.3 PyTorch: Deep Learning Library . . . 12
2.10.4 Conclusion . . . 13
2.11 Conclusion . . . 13
2.12 Summary . . . 14
2.12.1 Key Findings . . . 14
2.12.2 Areas for Future Research . . . 14

3 Methodology **15**
3.1 Datasets and Preprocessing . . . 15
3.1.1 Movie Data . . . 15
3.1.2 Ratings Data . . . 18
3.2 Artificial Intelligence Models . . . 19
3.2.1 Models for Collaborative Filtering . . . 19
3.2.2 Models for Content-based Filtering . . . 20
3.3 Evaluation Metrics . . . 21
3.3.1 Root Mean Square Error (RMSE) . . . 21
3.3.2 Mean Absolute Error (MAE) . . . 21
3.3.3 Mean Square Error (MSE) . . . 22
3.4 Chapter Summary and Forward Look . . . 22

4 Implementation **23**
4.1 Experimental Setup . . . 23
4.1.1 Dataset Characteristics . . . 23
4.2 Collaborative Filtering Models Implementation . . . 23
4.3 AutoEncoders . . . 24
4.3.1 Model Overview: AutoEncoders in Collaborative Filtering . . . 24
4.3.2 Data Preprocessing . . . 24
4.3.3 Model Architecture . . . 25
4.3.4 Training and Optimization . . . 25
4.3.5 Evaluation and Results . . . 25
4.4 Wide and Deep Neural Networks . . . 27
4.4.1 Model Overview . . . 27
4.4.2 Data Preprocessing . . . 28
4.4.3 Creating Dataloaders . . . 28
4.4.4 Model Architecture . . . 28
4.4.5 Training and Optimization . . . 29
4.4.6 Evaluation and Results . . . 29
4.5 Convolutional Neural Networks (CNN) . . . 31
4.5.1 Model Overview . . . 31
4.5.2 Data Preprocessing . . . 32
4.5.3 Creating Dataloaders . . . 32
4.5.4 Model Architecture . . . 32

4.5.5 Training and Optimization . . . 33
4.5.6 Evaluation and Results . . . 33
4.6 Graph Neural Network (GNN) for Recommendations . . . 34
4.6.1 Model Overview . . . 34
4.6.2 Data Preprocessing and Graph Construction . . . 35
4.6.3 Model Architecture . . . 35
4.6.4 Training and Optimization . . . 36
4.6.5 Evaluation and Results . . . 36
4.7 Neural Collaborative Filtering (NCF) for Recommendations . . . 37
4.7.1 Model Overview . . . 37
4.7.2 Data Preprocessing . . . 38
4.7.3 Model Architecture . . . 39
4.7.4 Training and Optimization . . . 40
4.7.5 Evaluation and Results . . . 40
4.8 Matrix Factorization . . . 42
4.8.1 Implementation using Surprise Library . . . 42
4.8.2 Results and Discussion . . . 43
4.9 Content-based Filtering Models Implementation . . . 43
4.9.1 Decision Trees . . . 43
4.9.2 k-NN . . . 45
4.9.3 Random Forest . . . 46
4.9.4 Support Vector Machines (SVM) . . . 47
4.9.5 XGBoost . . . 48
4.9.6 MLP . . . 49

5 Results and Discussion 53
5.1 Results and Discussion . . . 53
5.1.1 Collaborative Filtering Results . . . 53
5.1.2 Content-based Filtering Results . . . 54
5.1.3 Comparison . . . 55
5.1.4 Implications . . . 56

6 Evaluation 57
6.1 Fulfillment of Objectives . . . 57
6.2 Critical Assessment of the Research Process . . . 57
6.3 Limitations and Challenges . . . 58
6.4 Chapter Summary . . . 58

7 Conclusion 59
7.1 Conclusive Insights . . . 59
7.2 Contributions . . . 59
7.3 Future Work . . . 60

List of Figures

3.1 Content Based Filtering Metrics . . . 16
3.2 Number of Movies in Each Genre . . . 17
3.3 Average Rating per Genre . . . 17
3.4 Content Based Filtering Metrics . . . 18

4.1 Schematic representation of an AutoEncoder used in collaborative filtering. . 24
4.2 Plot of MAE for Training and Test data over 10 epochs. . . . 26
4.3 Plot of RMSE for Training and Test data over 10 epochs. . . . 27
4.4 Plot of MSE for Training and Test data over 10 epochs. . . . 27
4.5 Plot of MAE for Training and Test data over 10 epochs. . . . 30
4.6 Plot of RMSE for Training and Test data over 10 epochs. . . . 30
4.7 Plot of MSE for Training and Test data over 10 epochs. . . . 31
4.8 Typical structure of a Convolutional Neural Network. . . . 31
4.9 MAE graph for Training and Test data over 100 epochs for CNN model. . . . 34
4.10 RMSE graph for Training and Test data over 100 epochs for CNN model. . . 34
4.11 MSE graph for Training and Test data over 100 epochs for CNN model. . . . 35
4.12 Schematic representation of a Graph Neural Network for recommendations. . 36
4.13 Plot of MAE for Training and Test data over 20 epochs for the GNN model. . 37
4.14 Plot of RMSE for Training and Test data over 20 epochs for the GNN model. 38
4.15 Plot of MSE for Training and Test data over 20 epochs for the GNN model. . 38
4.16 Conceptual architecture of Neural Collaborative Filtering. . . . 39
4.17 RMSE graph for Training and Test data over 10 epochs for the NCF model. . 41
4.18 MAE graph for Training and Test data over 10 epochs for the NCF model. . 41
4.19 MSE graph for Training and Test data over 10 epochs for the NCF model. . . 42
4.20 Decision tree visualization truncated at a depth of 3. . . . 44
4.21 MSE for Training vs. Test set across epochs. . . . 51
4.22 RMSE for Training vs. Test set across epochs. . . . 52
4.23 MAE for Training vs. Test set across epochs. . . . 52

5.1 Collaborative Filtering Metrics . . . 54
5.2 Content Based Filtering Metrics . . . 55

List of Tables

3.1 Snapshot of the movie dataset. 15
3.2 Snapshot of the ratings dataset. 18

4.1 Training and Test Results after 10 epochs 26
4.2 Training and Test Results after 10 epochs 29
4.3 Performance Metrics for the CNN Model after 100 epochs 33
4.4 Performance Metrics for the GNN Model after 20 epochs 37
4.5 Performance Metrics for the NCF Model after 10 epochs 40

5.1 Collaborative Filtering Model Performance 53
5.2 Content-based Filtering Model Performance 54

Chapter 1

Introduction

In today's digital age, recommendation systems have become pivotal in enhancing user experiences across platforms. Whether suggesting movies, books, products, or even social connections, they dictate and curate personalized content for users (Ricci et al. 2011).

1.1 Background

Recommendation systems, a specific type of information filtering system, predict and suggest items that users might find interesting. With the ever-growing influx of online content, these systems act as a beacon, guiding users through vast digital landscapes based on their preferences, history, and behavior (Adomavicius & Tuzhilin 2005).

The domain of recommendation has seen techniques ranging from simple heuristics to complex machine learning models. The MovieLens dataset, especially its 100k variant, has been a cornerstone for researchers in this field (Harper & Konstan 2016). Containing 100,000 ratings from 943 users on 1,682 movies, it provides a rich ground for experimentation and model evaluation.

1.2 Problem Statement

The evolution of recommendation systems has witnessed the implementation of both traditional machine learning models and more recently, deep learning techniques (Zhang et al. 2019). While machine learning models, such as collaborative filtering using matrix factorization, have been widely adopted (Koren et al. 2009), the advent of deep learning promises handling larger datasets and potentially capturing intricate patterns within the data. Given this landscape, how do deep learning models compare with traditional machine learning approaches when applied to recommendation tasks. Using the MovieLens 100k dataset as the experimental ground, this research seeks to juxtapose these methods, shedding light on their relative merits and limitations.

1.3 Objectives

The primary goals of this research encompass:

- Implementing and evaluating traditional machine learning models for recommendation on the MovieLens 100k dataset.
- Designing and testing deep learning models tailored for recommendation tasks on the same dataset.
- Drawing a comprehensive comparison between the performance of traditional machine learning and deep learning models, elucidating their strengths and potential pitfalls.
- Offering actionable insights to guide future endeavors in the realm of recommendation systems, emphasizing empirical results.

1.4 Research Significance

The role of recommendation systems extends across various sectors, from entertainment to e-commerce. A thorough understanding of the most effective techniques, whether machine learning or deep learning, can empower businesses to fine-tune their recommendation engines, ensuring user relevance and increased platform engagement (Bobadilla et al. 2013).

As the scale and complexity of datasets burgeon, it's imperative for recommendation techniques to evolve in tandem. This research, by comparing the breadth of machine learning with the depth of deep learning, aims to pave a strategic path forward, balancing computational efficiency with recommendation quality (Goodfellow et al. 2016).

1.5 Dissertation Breakdown

This dissertation set out with the aim of comprehensively evaluating various machine learning and deep learning models in the context of recommendation systems, specifically focusing on the collaborative and content-based filtering paradigms.

- **Chapter 1: Introduction** set the stage by discussing the growing importance of recommendation systems in various sectors. It highlighted the research objectives, questions, and the significance of this study in contributing to both academia and industry.
- **Chapter 2: Literature Review** provided a detailed exploration of existing academic literature and studies. It examined the evolution of recommendation systems, the theoretical underpinnings of collaborative and content-based filtering, and the various machine and deep learning models applicable in these domains.
- **Chapter 3: Methodology** outlined the research's methodological approach, including data collection from the Kaggle's 100k movie ratings dataset, data preprocessing techniques, and the rationale behind the selection of specific models for evaluation. This chapter also discussed the metrics used for performance evaluation: RMSE, MAE, and MSE.

- **Chapter 4: Implementation** delved into the practical aspect of applying various recommendation models. The chapter began by describing the experimental setup, detailing the dataset characteristics and data preparation steps. Various challenges faced during implementation were also highlighted. The chapter then systematically went through the implementation details of both collaborative filtering and content-based filtering models. For collaborative filtering, methods like AutoEncoders, Wide and Deep Neural Networks, CNNs, GNNs, Neural Collaborative Filtering, and Matrix Factorization were discussed. In contrast, the content-based filtering section touched upon Decision Trees, k-NN, Random Forest, SVM, XGBoost, and MLP.

- **Chapter 5: Results and Discussion** presented a critical analysis of the research findings. It was observed that in collaborative filtering, AutoEncoders demonstrated superior performance, while in content-based filtering, XGBoost emerged as highly effective. The chapter delved into comparative analyses, theoretical implications, and practical considerations of these findings.

- **Chapter 6: Evaluation** assessed the entire research process. The chapter started by evaluating whether the research objectives set out in the beginning were met. A critical assessment of the research process was provided, shedding light on the strengths and potential areas of improvement. The chapter also highlighted various limitations and challenges faced during the research. It concluded with a summary of the chapter's main points.

- **Chapter 7: Conclusion** wrapped up the study by offering conclusive insights into the performance of different recommendation models and the implications of the research findings. The contributions of the study to the field of recommendation systems were outlined. Lastly, the chapter pointed towards future avenues of research, suggesting potential areas that can be explored further in the domain of recommendation systems.

Chapter 2

Literature Review

2.1 Artificial Intelligence in Recommendation Systems

Artificial Intelligence (AI) has revolutionized the landscape of recommendation systems over the past few decades. At its core, AI aims to mimic human intelligence, enabling machines to learn from data and make decisions (Russell & Norvig 2010). In the context of recommendation systems, AI techniques facilitate the prediction and suggestion of items that might be of interest to users.

Early recommendation systems were primarily rule-based, relying on explicit programming and heuristics. With the advent of AI, these systems transitioned to data-driven approaches, leveraging vast amounts of user data to generate personalized recommendations (Adomavicius & Tuzhilin 2005). Machine learning, a subset of AI, particularly played a pivotal role in this transformation. Algorithms such as clustering, regression, and classification became foundational in building recommendation models (Bobadilla et al. 2013).

Deep learning, an advanced subset of machine learning, further enhanced the capabilities of recommendation systems. Neural networks, especially deep neural networks, can capture intricate patterns and relationships in large datasets, making them particularly suited for recommendation tasks in today's data-rich environments (Zhang et al. 2019). For instance, platforms like Netflix and Spotify employ deep learning models to curate content for their users, considering both user history and content features (Gomez-Uribe & Hunt 2016).

In summary, AI has been instrumental in the evolution of recommendation systems, transitioning them from simple rule-based models to sophisticated, data-driven engines that can cater to individual user preferences with remarkable accuracy.

2.2 Collaborative Filtering

Collaborative Filtering (CF) stands as one of the most prominent techniques in the realm of recommendation systems. The fundamental premise of CF is to provide recommendations based on the preferences and behaviors of similar users (Su & Khoshgoftaar 2009). In other words, if two users agree on the evaluation of one item, they are likely to have a similar opinion on other items as well.

There are primarily two types of collaborative filtering methods: user-based and item-based.

2.2.1 User-based Collaborative Filtering

In user-based CF, recommendations for a target user are generated by analyzing preferences of users who have similar tastes or behaviors. The similarity between users is typically computed using methods like Pearson correlation or cosine similarity (Resnick et al. 1994). Once similar users are identified, their preferences are aggregated to recommend items to the target user.

2.2.2 Item-based Collaborative Filtering

Item-based CF, on the other hand, focuses on finding similarities between items rather than users. If a user has shown a preference for a particular item, other items deemed similar to it are recommended (Sarwar et al. 2001). This approach often proves more scalable and stable than user-based CF, especially when dealing with large datasets.

Matrix factorization, a technique that decomposes the user-item interaction matrix into multiple matrices representing latent factors, has also gained traction in collaborative filtering (Koren et al. 2009). Techniques like Singular Value Decomposition (SVD) have been employed to achieve this, leading to more accurate and efficient recommendations.

In conclusion, collaborative filtering, with its user-centric approach, has been foundational in the success of many recommendation systems. Its ability to harness the collective intelligence of users makes it a powerful tool in predicting and suggesting items of interest.

2.3 Content-based Filtering

Content-based Filtering (CBF) is another principal technique in recommendation systems, focusing on the attributes of items and a profile of the user's preferences (Lops et al. 2011). Unlike collaborative filtering, which relies on user-item interactions, content-based filtering recommends items by comparing the content of the items and a user profile, with content being described in terms of several descriptors or terms that are inherent to the item.

2.3.1 Item Representation

In content-based filtering, items are typically represented as vectors of descriptors. For instance, in a movie recommendation system, a movie can be described by its genre, director, main actors, and keywords extracted from its synopsis (Pazzani & Billsus 2007). Textual content, such as articles or product descriptions, can be represented using the Term Frequency-Inverse Document Frequency (TF-IDF) weighting scheme, which reflects the importance of terms in the content (Salton & Buckley 1988).

2.3.2 User Profile Creation

The user profile is constructed by analyzing the content of items that the user has interacted with. Over time, this profile evolves to represent the user's preferences accurately. The profile can be built using explicit feedback (e.g., ratings) or implicit feedback (e.g., click-through rates) (Adomavicius & Tuzhilin 2005).

2.3.3 Recommendation Generation

Recommendations are generated by comparing the user's profile with item representations. The similarity between the user profile and items can be computed using various measures, with cosine similarity being one of the most popular (Pazzani & Billsus 2007). Items that align closely with the user's profile are then recommended.

One of the main advantages of content-based filtering is its ability to recommend niche items that haven't been widely interacted with, as it doesn't rely on the preferences of other users. However, it can also lead to over-specialization, where users are only recommended items similar to what they've already seen or interacted with.

In conclusion, content-based filtering offers a personalized approach to recommendations by focusing on the attributes of items and how they align with a user's preferences.

2.4 Comparative Studies

Comparative studies in recommendation systems serve a pivotal role in understanding the landscape of available methodologies. These studies delve into contrasting different recommendation strategies, shedding light on their respective advantages, pitfalls, and optimal application environments. This rigorous analysis not only aids in comprehending the mechanisms underlying these systems but also assists decision-makers in aligning system selection with their specific requirements (Jannach et al. 2010).

2.4.1 Collaborative vs. Content-based Filtering

The debate between the efficacy of collaborative filtering and content-based filtering is a long-standing one in the realm of recommendation systems. Collaborative filtering, which hinges on user-item interactions and preferences of users with similar behaviors, is renowned for its ability to harness user community trends effectively. Its methodology of 'word-of-mouth' recommendations often unveils preferences that are not overtly apparent. However, it is not without its drawbacks, as it is notoriously known for grappling with the cold start problem, rendering it less effective for new items or users devoid of sufficient interaction history (Lam et al. 2008*b*).

In contrast, content-based filtering zeroes in on the attributes of items and the profile of users, crafting recommendations by matching user preferences with item features. This technique, while proficient in handling scenarios with limited user interactions, tends to pigeonhole users into a feedback loop of similar items, potentially stifling the diversity of recommendations (Lops et al. 2011).

The dichotomy between these two methods highlights the importance of context in recommendation systems, affirming that the effectiveness of a system is significantly influenced by the specific characteristics of the application scenario.

2.4.2 Hybrid Approaches

Recognizing the inherent limitations of both collaborative and content-based filtering, researchers have ventured into hybrid models that meld the methodologies of both worlds

(Burke 2002). These models, through a confluence of strategies, aim to mitigate the limitations inherent in each approach while capitalizing on their strengths. Comparative analyses indicate that these hybrid systems are often more resilient to issues like data sparsity and cold start, common in standalone systems, and exhibit enhanced recommendation accuracy and diversity, especially in complex scenarios with multifaceted data.

2.4.3 Deep Learning in Recommendation

The advent of deep learning has ushered in a new era in various fields, including recommendation systems. Deep learning leverages neural networks' ability to discern complex non-linear relationships, offering a substantial improvement over traditional machine learning techniques, particularly in handling unstructured data (Zhang et al. 2019). In the sphere of recommendation systems, neural collaborative filtering amalgamates collaborative filtering with deep learning to enhance the prediction accuracy (He et al. 2017).

However, the deployment of deep learning models comes at the cost of increased computational exigencies and a necessity for extensive datasets, making them less feasible for scenarios with limited computational resources or data. Furthermore, the 'black box' nature of deep learning models can raise transparency and interpretability concerns.

In summary, comparative studies are indispensable in the continuous advancement of recommendation systems, contributing significantly to the understanding and improvement of recommendation methodologies. These studies underscore the necessity for a nuanced approach to system selection, where consideration is given to the specific context, data availability, and system capabilities, ensuring the most effective and efficient system is employed.

2.5 Ethical and Privacy Concerns in Recommendation Systems

2.5.1 Data Privacy and Security

Data privacy and security are paramount in the digital age, particularly with recommendation systems that handle vast amounts of sensitive user data (Martin 2019). These systems often require access to personal information, browsing history, purchase records, and other user data to make accurate recommendations. This data, if mishandled or breached, can lead to significant privacy violations and security risks (Zhang et al. 2010).

Several methods have been developed to safeguard user privacy in recommendation systems. Techniques such as data anonymization, differential privacy, and homomorphic encryption are employed to protect the data while still allowing for effective recommendations (Nikolaenko et al. 2013, Dwork & Roth 2014). Furthermore, secure multi-party computation enables collaborative filtering without directly sharing user data between different parties (Erkin et al. 2012).

Despite these advancements, ensuring data privacy and security in recommendation systems remains a complex challenge. It necessitates a balance between personalization and privacy, demanding constant evolution in line with emerging data protection laws and cyber threats.

2.5.2 Ethical Implications

The ethical implications of recommendation systems extend beyond data privacy. These systems, while aiming to provide personalized content, can inadvertently create ethical dilemmas. One significant concern is the reinforcement of biases. Since recommendation systems often rely on historical data, there's a risk of perpetuating existing biases, leading to unfair or discriminatory recommendations (Ekstrand et al. 2018).

Another ethical challenge is the creation of ”echo chambers” or ”filter bubbles” (Pariser 2011). Highly personalized content can isolate users in ideological bubbles, limiting exposure to diverse viewpoints and potentially polarizing public discourse (Bakshy et al. 2015).

Moreover, the persuasive power of recommendation systems raises questions about user autonomy. The systems' ability to influence user choices can lead to ethical concerns regarding manipulation and informed consent (Yeung 2017).

Addressing these ethical implications requires a multidisciplinary approach, incorporating fairness, accountability, and transparency into the design and deployment of recommendation systems. Ongoing research into explainable AI and fair machine learning seeks to make recommendation systems more interpretable and less biased, promoting ethical standards and societal welfare (Castells 2015).

2.6 Challenges and Limitations

2.6.1 The Cold Start Problem

The cold start problem remains one of the most prominent challenges in recommendation systems. It arises when a new user joins a platform, or a new item is added, and there's insufficient data to make accurate recommendations (Lam et al. 2008*a*). For new users, the system lacks historical interaction data to gauge their preferences. For new items, the absence of user interactions means the system cannot reliably position the item in the recommendation space.

Several strategies have been proposed to tackle the cold start problem. Content-based methods, for instance, can provide initial recommendations for new users based on demographic data or initial interactions, such as surveys or introductory quizzes (Rashid et al. 2002). For new items, metadata and attribute-based filtering can be employed until enough user interactions are gathered (Park et al. 2008).

2.6.2 Scalability

As platforms grow and accumulate more users and items, ensuring that recommendation systems remain computationally efficient becomes challenging. With millions, or even billions, of interactions, processing and delivering real-time recommendations can be resource-intensive (Li et al. 2016).

Several techniques have been explored to enhance scalability. Matrix factorization methods, such as Singular Value Decomposition (SVD), have been optimized to work with large datasets (Koren et al. 2009). Additionally, distributed computing frameworks like Apache Spark have been leveraged to process vast amounts of data efficiently (Zhou et al. 2018).

2.6.3 Data Sparsity

Data sparsity is another significant challenge in recommendation systems. Often, users interact with only a small fraction of available items, leading to a sparse user-item interaction matrix. This sparsity can hinder the accuracy of collaborative filtering methods (Schein et al. 2002).

To address this, dimensionality reduction techniques, like matrix factorization, can be employed to derive latent factors from sparse data (Koren et al. 2009). Additionally, hybrid methods, which combine collaborative and content-based filtering, can provide recommendations even when user-item interactions are limited (Burke 2002).

2.7 Context-Aware Recommendations

2.7.1 Incorporating Context

Context-aware recommendation systems (CARS) represent an evolution in personalization technology, acknowledging that users' preferences may vary depending on their current context (Adomavicius & Tuzhilin 2011). Context can include any information that characterizes the situation of an entity (user or item), such as time, location, weather, mood, or even the device being used for access (Baltrunas et al. 2011).

Incorporating context into recommendation systems can enhance their relevance and usefulness. For instance, recognizing that a user's preference for restaurants might depend on the time of day, their current location, or the company they are with can significantly influence the recommendations provided (Palmisano et al. 2008). Techniques for incorporating context include pre-filtering, post-filtering, and context modeling, each with its strengths and application scenarios (Adomavicius & Tuzhilin 2011).

2.7.2 Real-world Applications

Context-aware recommendations have found widespread application across various domains. In e-commerce, for example, recommendations can be tailored based on the user's current browsing device, previous search history, current location, or even the weather in their area (Dey 2001).

In the domain of entertainment and streaming services, context-aware systems might recommend different content based on the day of the week, the time of day, or the device being used (Cantador & Fernández-Tobías 2011). For instance, a service might suggest family-friendly movies on a weekend morning on a television device but recommend a short comedy clip on a mobile device during a weekday lunch break.

Travel and tourism services also heavily rely on context-aware recommendations, offering suggestions for destinations, accommodations, and activities based on the current location, the composition of the travel group, or the time of year (Gavalas & Kenteris 2014).

2.8 Case Studies

2.8.1 Industry Applications

Recommendation systems have become a cornerstone of several industries, significantly influencing user experience and business performance. Analyzing specific instances can provide deep insights into their practical applications and effectiveness.

Netflix

Netflix, a leader in streaming services, utilizes a sophisticated recommendation system to personalize content for its millions of users (Gomez-Uribe & Hunt 2016). The system analyzes user interactions, viewing history, and preferences, combining collaborative and content-based filtering techniques. Furthermore, it considers contextual variables like viewing time to refine its suggestions (Amatriain 2013). This high degree of personalization keeps users engaged, directly influencing customer retention and satisfaction.

Amazon

Amazon employs a recommendation system to enhance user shopping experiences, drive sales, and improve customer satisfaction. It uses item-to-item collaborative filtering, analyzing user purchase history, items in the shopping cart, items rated and liked, and what other customers have viewed or purchased (Linden et al. 2003). This approach not only personalizes the user experience but also encourages increased order values through up-selling and cross-selling.

Spotify

Spotify uses a recommendation system to personalize music and podcast listening experiences for its users (Johnson 2015). It employs collaborative filtering and natural language processing to understand user preferences and the audio features of tracks. "Discover Weekly" is a feature that exemplifies the power of their system, creating personalized playlists by comparing a user's taste with similar profiles.

2.8.2 Success Stories and Lessons Learned

The impact of recommendation systems on business success is evident in numerous cases. Netflix, for instance, attributes its high customer retention rate significantly to its recommendation system, with a vast majority of watched content coming from its recommendations (Gomez-Uribe & Hunt 2016).

Amazon's "Customers who bought this item also bought" feature is a testament to the effectiveness of its recommendation system, influencing an estimated 35% of user purchases (McAuley et al. 2015).

Spotify's "Discover Weekly," has been a resounding success, with millions of users actively using these playlists, showcasing the system's ability to maintain user engagement and satisfaction (Johnson 2015).

These success stories underscore several lessons. Firstly, personalization is key to user engagement and satisfaction. Secondly, a combination of different recommendation techniques can cater to various scenarios and enhance overall effectiveness. Lastly, considering context can significantly improve recommendation relevance.

However, these systems are not without challenges. Data privacy, the risk of creating echo chambers, and the need for vast amounts of data are ongoing concerns that these platforms continually address (Pariser 2011).

2.9 Emerging Trends and Future Directions

2.9.1 Technological Advancements

The landscape of recommendation systems is continually evolving, with new technological advancements shaping its future. The adoption of cutting-edge technologies promises to bring about more efficient, effective, and personalized recommendation experiences for users.

Artificial Intelligence (AI) and Machine Learning (ML)

With the proliferation of AI and ML, recommendation systems are poised to become even more sophisticated (Zhang et al. 2020). Deep learning, a subset of ML, offers the potential to capture intricate patterns in vast datasets, potentially leading to more accurate recommendations (Zhang et al. 2019). Furthermore, reinforcement learning, which focuses on decision-making, can be used to optimize recommendations based on long-term user engagement (Zhao et al. 2013).

Internet of Things (IoT)

The integration of IoT with recommendation systems offers a new frontier of opportunities. As everyday objects become interconnected, the data they generate can be harnessed to enhance recommendation systems. For instance, a smart refrigerator might recommend recipes based on its contents, or a wearable device might suggest workout routines based on a user's health metrics (Whitmore et al. 2015).

2.9.2 Predictive Analysis and Personalization

The future of recommendation systems is not just about suggesting items but predicting user needs even before they express them. Predictive analytics, powered by advanced algorithms and vast amounts of data, will play a pivotal role in this evolution (Shen et al. 2019).

As recommendation systems become more predictive, they can offer proactive suggestions, enhancing user engagement. For instance, a music streaming service might predict a user's mood based on their recent interactions and suggest a playlist accordingly (Lee & Lee 2018).

Moreover, as personalization algorithms become more refined, there will be an even greater emphasis on creating unique user experiences, tailoring recommendations to individual needs, preferences, and contexts (Felfernig et al. 2018).

2.10 Tools and Methodologies for Model Development

This dissertation employs several advanced tools and methodologies to develop, test, and deploy recommendation system models. The choice of tools is critical to the efficient implementation and robust performance of these models, especially when dealing with large-scale data and complex algorithms.

2.10.1 Python: The Core Programming Language

Python is selected as the primary programming language due to its simplicity, readability, and the richness of its scientific computing libraries (Van Rossum & Drake Jr 1995). Its extensive support for various machine learning and deep learning libraries makes it a popular choice among researchers and practitioners in the field of artificial intelligence.

- **Versatility and Ease of Use:** Python's syntax is clean and straightforward, making it accessible to individuals with varying levels of coding expertise. This simplicity accelerates the development process and facilitates the testing of new ideas (Sanner 1999).
- **Community and Library Support:** Python boasts a vibrant community that continuously contributes to a wealth of libraries and frameworks. These resources simplify complex tasks and significantly reduce development time.

2.10.2 Scikit-learn: Machine Learning Library

Scikit-learn is a versatile and user-friendly machine learning library for Python. It provides a range of supervised and unsupervised learning algorithms and is valuable for constructing and analyzing recommendation systems (Pedregosa et al. 2011).

- **Comprehensive Toolkit:** Scikit-learn includes several classification, regression, and clustering algorithms, and it integrates well with other Python libraries. Its functionalities are especially beneficial for processing datasets and evaluating the performance of recommendation models.
- **Documentation and Community:** With extensive documentation and an active developer community, scikit-learn is an invaluable resource for learners and seasoned developers alike. The community's continuous contributions drive the library's growth and support its adaptability to emerging trends.

2.10.3 PyTorch: Deep Learning Library

PyTorch, known for its flexibility and user-friendliness, is a prominent library that supports deep learning, a critical component in the development of sophisticated recommendation systems (Paszke et al. 2019).

- **Dynamic Computational Graph:** PyTorch operates using dynamic computational graphs, which allow for flexibility in model architecture changes and are particularly beneficial for research and experimentation in model development.
- **GPU Acceleration:** To handle the computational complexity of deep learning models, PyTorch leverages GPU acceleration, significantly reducing training times and enabling the processing of large-scale data.

- **Community and Ecosystem:** PyTorch has a thriving community and ecosystem, with numerous tutorials and resources available. Its compatibility with other Python libraries and tools makes it a robust framework for developing, training, and deploying deep learning models.

2.10.4 Conclusion

The combination of Python, scikit-learn, and PyTorch offers a powerful, flexible, and efficient environment for developing recommendation system models. This robust toolkit enables the handling of large datasets, implementation of state-of-the-art algorithms, and integration of machine learning and deep learning paradigms. The active communities surrounding these technologies also ensure continual updates and improvements, keeping the dissertation's methodologies at the forefront of technological advancements.

2.11 Conclusion

Recommendation systems have become an integral part of the digital landscape, influencing user choices across a myriad of platforms and domains. From movies and music to shopping and social connections, these systems curate personalized experiences, enhancing user engagement and satisfaction.

This literature review delved into the core techniques that power recommendation systems. We explored the user-centric approach of collaborative filtering, the item attribute-driven method of content-based filtering, and the comparative studies that juxtapose these techniques against each other and against emerging deep learning methodologies.

Collaborative filtering, with its reliance on user interactions, has proven effective in scenarios with abundant user data, capturing collective preferences. Content-based filtering, focusing on item attributes, offers a more personalized touch, especially when user interactions are sparse. Hybrid methods, combining the strengths of both, have emerged as powerful tools to address the limitations inherent to each approach. Furthermore, the advent of deep learning in recommendation systems promises to harness the vast amounts of data available today, capturing intricate patterns and relationships.

However, as with all technologies, recommendation systems come with their challenges. Issues such as the cold start problem, over-specialization, and the computational demands of deep learning models underscore the need for continuous research and innovation in this domain.

In the ever-evolving world of AI and machine learning, recommendation systems stand at the crossroads of technology and user experience. As we move forward, it is imperative to strike a balance between computational efficiency and the quality of recommendations, ensuring that users not only receive relevant suggestions but also discover new interests and horizons.

2.12 Summary

2.12.1 Key Findings

This literature review has provided comprehensive insights into the multifaceted realm of recommendation systems, highlighting their evolution, methodologies, applications, and emerging trends. Key findings indicate that:

- Personalization is paramount in enhancing user engagement and satisfaction, as evidenced by the success stories of industry leaders like Netflix, Amazon, and Spotify (Gomez-Uribe & Hunt 2016, Linden et al. 2003, Johnson 2015).
- Collaborative and content-based filtering remain foundational techniques, each with its strengths and applicable scenarios. However, their limitations, such as the cold start problem and over-specialization, necessitate further research and innovation (Adomavicius & Tuzhilin 2005, Lops et al. 2011).
- Hybrid approaches, combining various methodologies, and deep learning models are gaining traction for their ability to improve accuracy and prediction quality, especially in handling large-scale data (Burke 2002, Zhang et al. 2019).
- The integration of context-aware recommendations and IoT in recommendation systems opens new frontiers for personalized user experiences (Adomavicius & Tuzhilin 2011, Whitmore et al. 2015).
- Ethical considerations, particularly data privacy and potential biases, are increasingly pivotal in the development and deployment of recommendation systems (Pariser 2011).

2.12.2 Areas for Future Research

While significant advancements have been made, there remains a plethora of opportunities for further exploration in recommendation systems:

- Exploring solutions to the cold start problem and data sparsity, potentially through innovative data collection methods or leveraging synthetic data generated by AI (Lam et al. 2008*b*).
- Enhancing the interpretability of deep learning models in recommendation systems to provide users with transparent, understandable recommendations (Zhang et al. 2019).
- Investigating the long-term effects of recommendation systems on user behavior and preferences, addressing potential issues like echo chambers or filter bubbles (Pariser 2011).
- Assessing the integration of emerging technologies such as blockchain for enhanced data security and privacy in recommendation systems.

Chapter 3

Methodology

3.1 Datasets and Preprocessing

The foundation of this research is built upon the MovieLens 100k dataset, a renowned compilation in the machine learning community, sourced from the GroupLens research lab at the University of Minnesota. This comprehensive dataset, often used as a benchmark in collaborative filtering and content-based recommendation system studies, provides a rich amalgamation of movie details, genres, and user ratings. It presents a unique opportunity to develop, experiment, and validate various recommendation models due to its complex and multifaceted nature.

3.1.1 Movie Data

The movie dataset, pivotal for content-based recommendation models, consists of 27,278 unique movie entries. The dataset's structure encompasses three primary columns: movieId, title, and genres. Each movie is uniquely identified by its movieId, a sequential identifier, ensuring there are no duplicates and providing a reliable reference across different datasets. The titles include the movie's name along with the release year, aiding in distinguishing between movies with similar or identical titles. The genres column is a categorical variable with each category separated by a vertical bar ('—'), indicating the various genres a particular movie belongs to.

movieId	**title**	**genres**
1	Toy Story (1995)	Adventure—Animation—Children—Comedy—Fantasy
2	Jumanji (1995)	Adventure—Children—Fantasy

Table 3.1: *Snapshot of the movie dataset.*

A significant preprocessing step was the transformation of the genres column into a format suitable for machine learning algorithms. The original genres column consisted of a string with multiple genres separated by a vertical bar. This was transformed into multiple binary columns, each representing a distinct genre, indicating its presence (1) or absence (0) for a particular movie. This one-hot encoding method is crucial for content-based models as it translates categorical data into a numerical format, a prerequisite for most machine learning algorithms. This preprocessing expanded the dataset from three to 23 columns, with each column representing a distinct genre.

Further enriching the dataset, average movie ratings were computed using the ratings data and integrated into the movie dataset. This was achieved by calculating the mean rating for each movie from the ratings dataset, which provided a foundational understanding of the general reception of the movies by the audience. The addition of the average ratings to the movie dataset not only enhanced the depth of information available for each movie entry but also provided a crucial metric for evaluating and comparing the performance of various recommendation models. This comprehensive dataset, with detailed movie genres and user reception, forms the backbone of our content-based recommendation system analysis, serving as a robust ground for both training and validation phases.

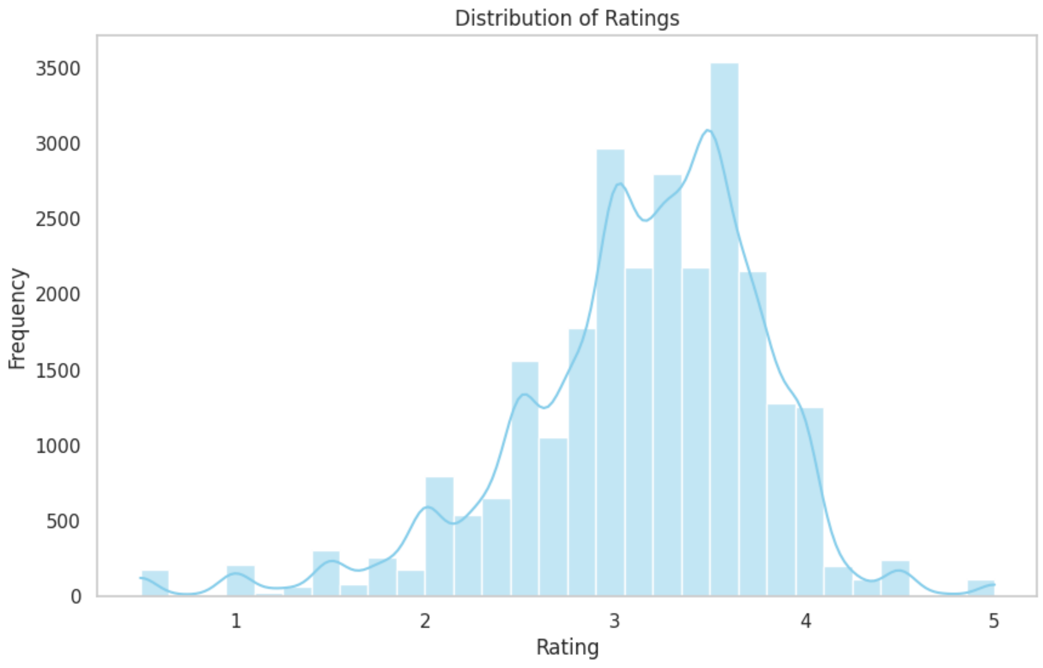

Figure 3.1: *Content Based Filtering Metrics*

The histogram above illustrates the distribution of average movie ratings. We observe a somewhat normal distribution, albeit with a slight left skew. The majority of average ratings are clustered around the 3.0 to 4.0 range, indicating a tendency among users to give moderate to high ratings. The kernel density estimate (KDE) line provides a smoothed estimate of the distribution, further emphasizing the concentration of ratings in this middle range.

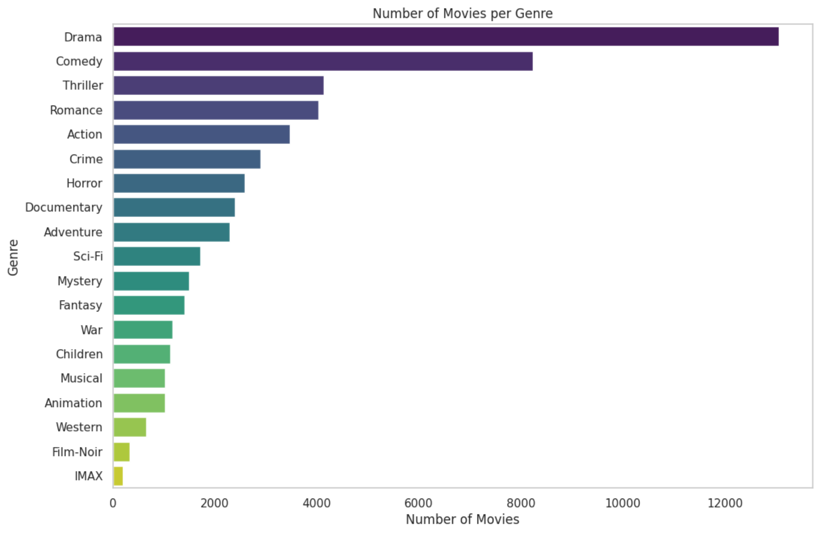

Figure 3.2: *Number of Movies in Each Genre*

The bar chart above displays the number of movies for each genre in the dataset. It's clear that some genres, like "Drama" and "Comedy," are significantly more prevalent than others, such as "IMAX" or "Western." This disparity could be due to various factors, including audience preference, production trends, or even cultural influences during the years the movies were released.

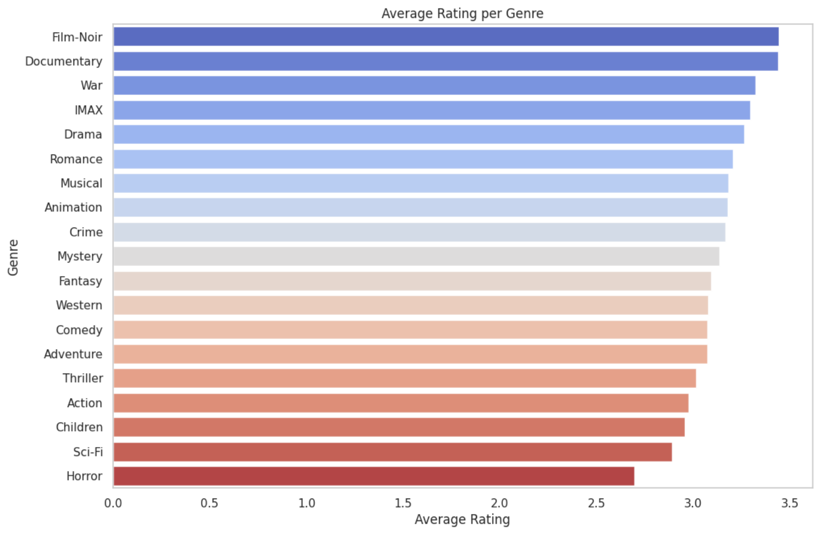

Figure 3.3: *Average Rating per Genre*

The bar chart above represents the average rating for each genre. It's intriguing to note that although some genres have fewer movies, they might hold higher average ratings, indicating a possible trend of quality or a niche audience preference for these genres. For instance, genres like "War" and "Crime" appear to have higher average ratings compared to more prevalent genres like "Comedy" or "Action."

3.1.2 Ratings Data

The ratings dataset is the cornerstone for collaborative recommendation models. It encapsulates the interactions of 943 users with 1,682 movies, forming a total of 100,000 ratings, thereby providing a substantial ground for understanding and analyzing user preferences. The data is stored in a tabular format consisting of four columns: userId, movieId, rating, and timestamp. The userId and movieId are numerical identifiers for users and movies, respectively, while ratings are on a scale of 1 to 5, with 5 being the highest. The timestamp column records the exact date and time the rating was provided, offering potential insights into user behavior patterns related to time.

userId	movieId	rating	timestamp
1	2	3.5	2005-04-02 23:53:47
1	29	3.5	2005-04-02 23:31:16

Table 3.2: *Snapshot of the ratings dataset.*

The preprocessing steps were meticulously conducted using Python, with libraries such as Pandas for data manipulation and NumPy for numerical operations, ensuring the reliability and accuracy of the transformations and calculations. The clean, transformed, and integrated dataset is poised to be an ideal candidate for feeding into various machine learning and deep learning models, setting the stage for the next phases of this research.

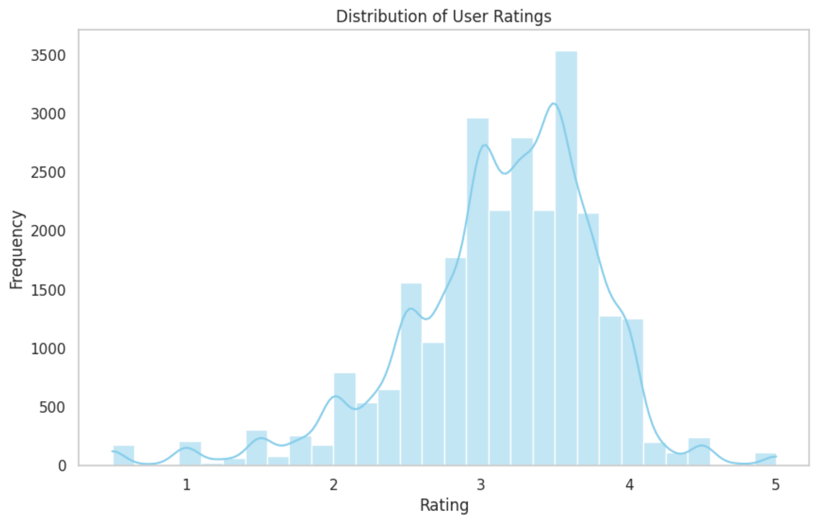

Figure 3.4: *Content Based Filtering Metrics*

The histogram above illustrates the distribution of user ratings. We observe that the distribution is left-skewed, with a significant number of ratings in the 3.0 to 4.0 range. There is a noticeable peak at the whole numbers (especially at 4.0), which might suggest a tendency among users to prefer round numbers while rating. The kernel density estimate (KDE) line provides a smoothed estimate of the distribution, confirming the concentration of ratings in the higher range.

3.2 Artificial Intelligence Models

Recommendation systems have become a cornerstone in the decision-making processes for consumers worldwide. Leveraging artificial intelligence, these systems parse through vast datasets to identify patterns, preferences, and potential areas of interest for users. This research delves into the intricate world of recommendation systems, focusing on two primary methodologies: collaborative filtering and content-based filtering. Each approach utilizes a distinct set of AI models, meticulously chosen for their efficacy in handling specific types of data and scenarios encountered in recommendation systems.

3.2.1 Models for Collaborative Filtering

Collaborative filtering epitomizes the essence of community in the realm of recommendation systems. It operates on the premise that users who agreed in the past tend to agree again in the future. This methodology harnesses user-item interactions and historical data to unearth patterns and preferences, forming the bedrock for future recommendations.

The models employed in collaborative filtering for this research are diverse, each bringing unique strengths to the table:

- **AutoEncoders:** Utilized for their prowess in dimensionality reduction and feature learning, AutoEncoders help in compressing user-item interactions into a lower-dimensional space, capturing the underlying preferences and characteristics.
- **Deep Neural Networks (DNN):** With their ability to model complex non-linear relationships, DNNs are used to predict potential user ratings on unrated items.
- **Convolutional Neural Networks (CNN):** Traditionally used in image processing, CNNs in this context are adapted to process matrix-like structures (user-item interactions), identifying patterns that are indicative of user preferences.
- **Graph Neural Networks (GNN):** GNNs are revolutionary in processing the relational data among users and items, capturing the collaborative signal in the data, and using it to make accurate recommendations.
- **Neural Collaborative Filtering (NCF):** This model combines the strengths of DNNs with matrix factorization, specifically designed for recommendation systems, providing robust performance especially in scenarios with sparse data.
- **Matrix Factorization:** A staple in recommendation systems, Matrix Factorization, particularly Singular Value Decomposition (SVD), distills user-item interaction matrices into latent factors, uncovering hidden patterns in the data.

The performance of these models is evaluated based on metrics that reflect both accuracy and business relevance, such as Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and precision at the k-th recommendation (Precision@k).

3.2.2 Models for Content-based Filtering

Content-based filtering, in contrast, takes a more personalized approach. This technique recommends items by comparing their features to the preferences of users. The system learns to recommend items that are similar to the ones the user liked in the past. Unlike collaborative filtering, content-based filtering does not require other users' data; it focuses solely on the attributes of the items and a single user's preferences.

The models explored in content-based filtering are chosen for their capacity to handle high-dimensional data and their effectiveness in learning the importance of different features:

- **Decision Trees:** Known for their interpretability and effectiveness in classifying categorical data, they're used to identify the features most influential in predicting a user's rating.
- **k-Nearest Neighbors (k-NN):** This algorithm is employed for its simplicity and effectiveness in identifying items with similar features, leveraging the power of community.
- **Random Forest:** An ensemble of decision trees, Random Forest is utilized for its high accuracy, ability to rank the importance of different features, and robustness to overfitting.
- **Support Vector Machines (SVM):** With their ability to handle high-dimensional spaces and their effectiveness in classification tasks, SVMs are used to classify user preferences across different genres.
- **XGBoost:** An optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable, XGBoost is instrumental in handling sparse data characteristic of many content-based systems.
- **Multi-Layer Perceptron (MLP):** This type of neural network is known for its ability to model non-linear relationships and its effectiveness in regression and classification tasks, making it suitable for predicting user ratings.

Similar to collaborative models, the performance of content-based models is assessed using a combination of error metrics and classification or regression accuracy, providing a holistic view of the model's predictive power and reliability.

This research's nuanced approach, employing a diverse array of models, lays the groundwork for an in-depth exploration of the recommendation systems' capabilities. The subsequent sections will delve deeper into each model's intricacies, shedding light on their operational mechanisms, comparative analyses, and contextual performances.

3.3 Evaluation Metrics

The performance of recommendation models is critically assessed using a suite of evaluation metrics. These metrics not only quantify the accuracy of the models but also provide insights into their reliability and robustness. In this research, three primary metrics were employed: Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and Mean Square Error (MSE).

3.3.1 Root Mean Square Error (RMSE)

The RMSE is a widely used metric that measures the average magnitude of errors between predicted and observed values. Mathematically, it is the square root of the average of squared differences between predictions and actual observations.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3.1}$$

Where:

- y_i is the actual value.
- $\hat{y}_i$ is the predicted value.
- n is the total number of observations.

A lower RMSE value indicates better model performance, with predictions closely aligning with the actual values.

3.3.2 Mean Absolute Error (MAE)

MAE quantifies the average of the absolute differences between predictions and actual values. It provides a linear penalty for each unit of difference between the predicted and actual values.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.2}$$

Where:

- y_i is the actual value.
- $\hat{y}_i$ is the predicted value.
- n is the total number of observations.

A smaller MAE indicates a better model fit to the data.

3.3.3 Mean Square Error (MSE)

MSE is similar to RMSE but does not take the square root. It measures the average of the squares of the errors or deviations, that is, the difference between the estimator and what is estimated.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.3}$$

Where:

- y_i is the actual value.
- $\hat{y}_i$ is the predicted value.
- n is the total number of observations.

A lower MSE value indicates a model that can more accurately predict the outcomes.

In summary, while all these metrics provide a measure of model accuracy, their interpretations and sensitivities differ. RMSE gives a relatively high weight to large errors, MAE provides a straightforward average error magnitude, and MSE emphasizes the squares of the errors.

3.4 Chapter Summary and Forward Look

This chapter provided a foundational understanding of the various artificial intelligence models and evaluation metrics employed in this dissertation. The models, both for collaborative and content-based filtering, were introduced, and their performance metrics were elaborated upon.

While this chapter offered a brief overview, the subsequent chapters will delve deeper into the intricacies of each model. The forthcoming sections will encompass detailed implementations, results, and comprehensive discussions, shedding light on the strengths, weaknesses, and potential improvements of each model in the context of recommendation systems. The journey ahead aims to provide a holistic understanding of the models in action, their real-world implications, and the innovations they bring to the table in the realm of recommendation systems.

Chapter 4

Implementation

This chapter provides an in-depth exploration into the practical intricacies involved in bringing the discussed recommendation models to life. The discourse transitions from the broader experimental landscape to the nitty-gritty details of each model's real-world deployment, highlighting the tools used, challenges encountered, and the rationale behind specific implementation decisions.

4.1 Experimental Setup

The models were primarily developed in Python, capitalizing on its versatility and the rich ecosystem it offers for data science and machine learning. Two notable libraries employed in this research were *Scikit-learn* and *PyTorch*. While Scikit-learn furnished the tools for traditional machine learning techniques and data preprocessing, PyTorch facilitated the deep learning implementations, given its flexibility and capability to handle complex neural architectures.

4.1.1 Dataset Characteristics

For collaborative filtering models, the dataset encompassed approximately 100,000 rows. This relatively large volume of data was a pivotal factor in the inclination towards deep learning models, which can exploit the rich patterns present in such extensive datasets. Conversely, the content-based filtering approach operated on a more modest dataset of around 27,000 rows, offering a contrasting landscape for model implementation.

4.2 Collaborative Filtering Models Implementation

Collaborative filtering models, as discussed, rely on user-item interactions. This section will delve into the specifics of implementing each of the collaborative filtering models.

4.3 AutoEncoders

4.3.1 Model Overview: AutoEncoders in Collaborative Filtering

AutoEncoders are a special category of neural networks known for their ability to recreate the input at the output layer. They operate through two main components: an encoder and a decoder. The encoder is responsible for compressing the input data into a more compact latent-space representation, while the decoder works on reconstructing the original input data from this compressed form.

In the realm of collaborative filtering for recommendation systems, AutoEncoders have garnered attention due to their prowess in predicting user ratings for items. By training on sparse user-item matrices, AutoEncoders can effectively fill in the gaps, thereby predicting potential ratings for unwatched items. The latent space representation in this scenario often captures the underlying patterns or structures in user-item interactions, facilitating more personalized and accurate recommendations (Sedhain et al. 2015).

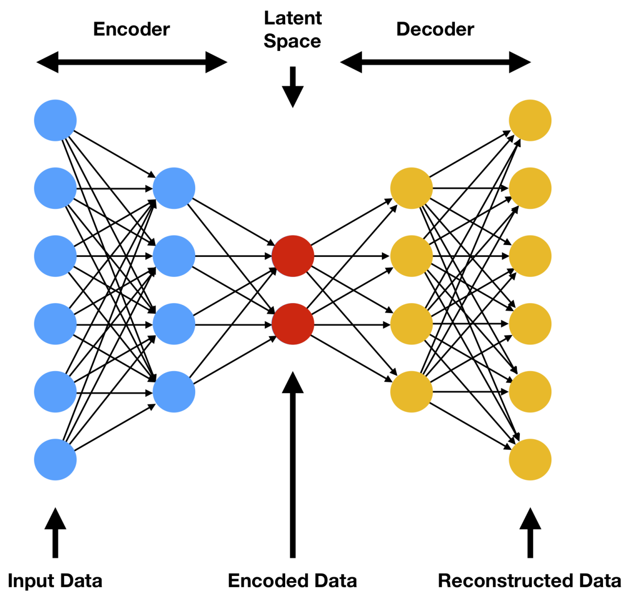

Figure 4.1: *Schematic representation of an AutoEncoder used in collaborative filtering.*

Given their versatility and promising results in preliminary studies, further exploration of AutoEncoders in the context of recommendation systems and their application on datasets like MovieLens is both timely and pertinent.

4.3.2 Data Preprocessing

The dataset contains user ratings for movies. To simplify the recommendation task, ratings were binarized: ratings of 4 and above were considered as positive (encoded as 1) and ratings below 4 as negative (encoded as 0).

Label encoders were used to transform the userId and movieId columns into contiguous integers, which are more suitable for embedding layers in neural networks. The dataset was then split into training and test sets using an 80-20 split.

4.3.3 Model Architecture

The AutoEncoder model consists of two main parts: an embedding layer for users and items, and the AutoEncoder itself. The embedding layer translates user and item IDs into dense vectors. These vectors are concatenated and passed through the encoder and decoder sequences.

The encoder compresses the concatenated embeddings into a latent representation, while the decoder attempts to reconstruct the original input from this latent representation. The output of the model is passed through a sigmoid activation function to produce a prediction in the range [0, 1], representing the likelihood of a positive rating.

```python
import torch.nn as nn
import torch

class AutoEncoder(nn.Module):
    def __init__(self, num_users, num_items, embedding_dim, hidden_dim):
        super().__init__()
        self.user_emb = nn.Embedding(num_users, embedding_dim)
        self.item_emb = nn.Embedding(num_items, embedding_dim)
        self.encoder = nn.Sequential(
            nn.Linear(embedding_dim*2, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, embedding_dim*2)
        )
        self.decoder = nn.Sequential(
            nn.Linear(embedding_dim*2, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, 1)
        )

    def forward(self, users, items):
        user_emb = self.user_emb(users)
        item_emb = self.item_emb(items)
        x = torch.cat([user_emb, item_emb], dim=1)
        x = self.encoder(x)
        x = self.decoder(x)
        return torch.sigmoid(x).squeeze()
```

4.3.4 Training and Optimization

The model was trained using the Binary Cross Entropy (BCE) loss, which is suitable for binary classification tasks. The Adam optimizer was employed with a learning rate of 0.01. The model was trained for 10 epochs.

4.3.5 Evaluation and Results

During training, the model's performance was evaluated on the test set using various metrics, including Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Mean Square

Error (MSE). The training and test losses, as well as these evaluation metrics, were plotted over the epochs to monitor the model's convergence and performance.

Results Summary

After 10 epochs of training, the model achieved the following results:

Metric	Training	Test
MAE	0.1819	0.3496
RMSE	0.3011	0.5008
MSE	0.0909	0.2509

Table 4.1: *Training and Test Results after 10 epochs*

Visualization Descriptions

Mean Absolute Error (MAE) Plot The first graph displays the Mean Absolute Error (MAE) for both training and test data over the epochs. From the graph, it can be observed that the training MAE decreases steadily with each epoch, indicating that the model is learning. On the other hand, the test MAE might show fluctuations, potentially indicating areas where the model's generalization can be improved.

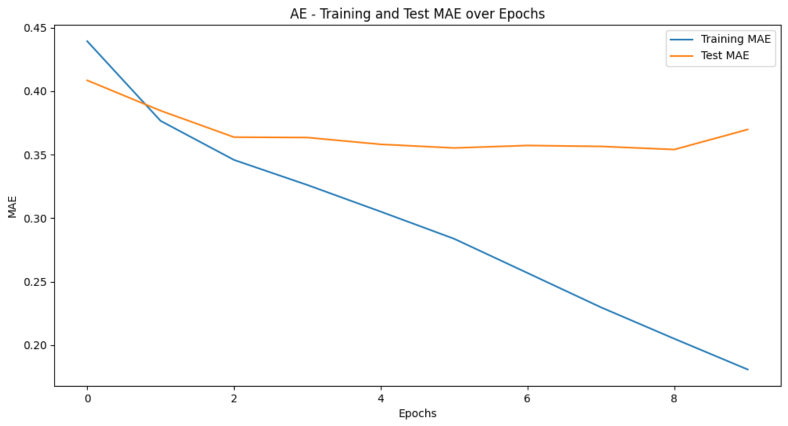

Figure 4.2: *Plot of MAE for Training and Test data over 10 epochs.*

Root Mean Square Error (RMSE) Plot The second graph shows the Root Mean Square Error (RMSE) for both training and test sets. Similar to the MAE graph, a decrease in training RMSE indicates that the model is minimizing its error over time.

Mean Square Error (MSE) Plot The third graph plots the Mean Square Error (MSE) for the training and test sets. The trends in this graph should be similar to those observed in the RMSE graph, given the direct relationship between RMSE and MSE.

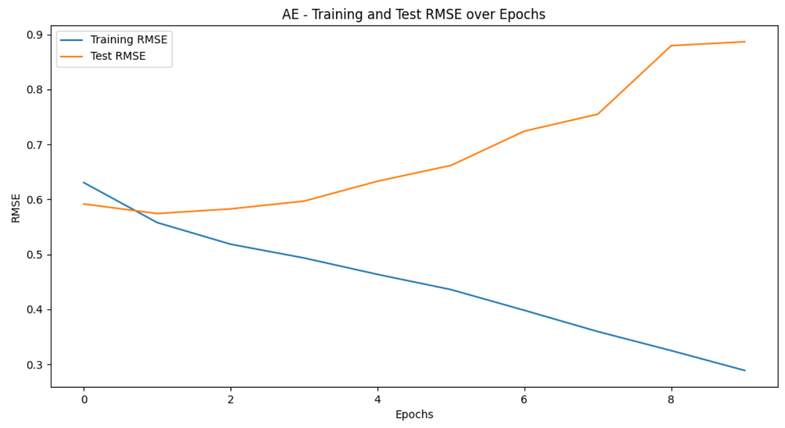

Figure 4.3: *Plot of RMSE for Training and Test data over 10 epochs.*

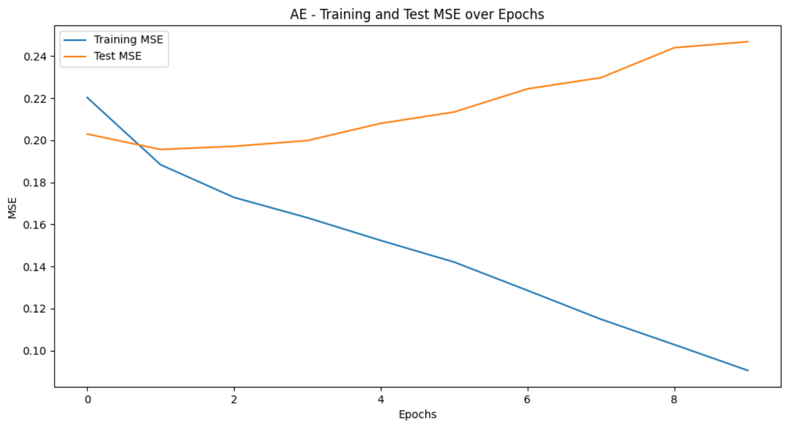

Figure 4.4: *Plot of MSE for Training and Test data over 10 epochs.*

4.4 Wide and Deep Neural Networks

4.4.1 Model Overview

The Wide & Deep Learning model, pioneered by Google, emerges as a novel neural network architecture tailored for complex recommendation tasks, especially those demanding large-scale processing. At the heart of its design lies the strategic amalgamation of two contrasting but complementary mechanisms: the linear model's aptitude for memorization (wide component) and the neural network's capacity for generalization (deep component). This blend ensures the model efficiently captures both dense and sparse feature combinations, thus accommodating intricate user-item interaction patterns and improving recommendation quality (Cheng et al. 2016).

Given the robustness of the Wide & Deep architecture and its proven effectiveness in handling real-world recommendation challenges, it stands as a prominent choice for researchers and practitioners aiming to optimize large-scale recommendation systems.

4.4.2 Data Preprocessing

The dataset contains user ratings for movies. To facilitate the use of embeddings, mappings were created for userId and movieId to continuous indices. The dataset was then split into training and test sets using an 80-20 split.

4.4.3 Creating Dataloaders

To efficiently load data in batches during training, a custom PyTorch Dataset was created. This MovieLensDataset class provides an interface to access user, movie, and rating data.

4.4.4 Model Architecture

The model consists of:

- **Embedding layers:** These transform user and movie IDs into dense vectors of fixed size.
- **Wide component:** This captures memorization and is implemented using the concatenated embeddings.
- **Deep component:** This captures generalization and consists of several fully connected layers.
- **Dropout layers:** These are added for regularization to prevent overfitting.

```python
class WideAndDeepModel(nn.Module):
    def __init__(self, n_users, n_movies, n_factors, embedding_dropout, hidden_layers, dropouts):
        super().__init__()

        # Embeddings
        self.user_embedding = nn.Embedding(n_users, n_factors)
        self.movie_embedding = nn.Embedding(n_movies, n_factors)
        self.embedding_dropout = nn.Dropout(embedding_dropout)

        # Linear Layers
        self.hidden_layers = nn.ModuleList([nn.Linear(n_factors*2, hidden_layers[0])])
        for i in range(1, len(hidden_layers)):
            self.hidden_layers.append(nn.Linear(hidden_layers[i-1], hidden_layers[i]))
        self.output_layer = nn.Linear(hidden_layers[-1], 1)

        # Dropout Layers
        self.dropout_layers = nn.ModuleList([nn.Dropout(dropout) for dropout in dropouts])

    def forward(self, users, movies):
        user_embeds = self.user_embedding(users)
        movie_embeds = self.movie_embedding(movies)

        # Combine the embeddings (concatenate)
        x = torch.cat([user_embeds, movie_embeds], dim=1)
        x = self.embedding_dropout(x)

        # Pass through the hidden layers
        for hidden_layer, dropout_layer in zip(self.hidden_layers, self.
    dropout_layers):
            x = hidden_layer(x)
            x = nn.ReLU()(x)
            x = dropout_layer(x)

        # Output layer
        out = self.output_layer(x)

        return out
```

4.4.5 Training and Optimization

The model was trained using the Mean Squared Error (MSE) loss, which measures the average squared difference between predicted and actual ratings. The Adam optimizer was employed with a learning rate of 0.001. The model was trained for 10 epochs.

4.4.6 Evaluation and Results

Throughout the training, the model's performance was gauged on both the training and test sets using various metrics, namely Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Square Error (RMSE). The training and test losses, alongside these evaluation metrics, were charted over the epochs to track the model's convergence and efficacy.

Results Summary

After 10 epochs of training, the model achieved the following results:

Metric	Training	Test
MAE	0.91006	0.79213
RMSE	1.13996	0.97988
MSE	1.30013	0.96045

Table 4.2: *Training and Test Results after 10 epochs*

Visualization Descriptions

Mean Absolute Error (MAE) Plot The MAE plot offers insights into the average magnitude of discrepancies between the predicted and actual ratings for both the training and test sets over the epochs.

Root Mean Square Error (RMSE) Plot The RMSE graph demonstrates the model's prowess in predicting ratings with minimized squared errors.

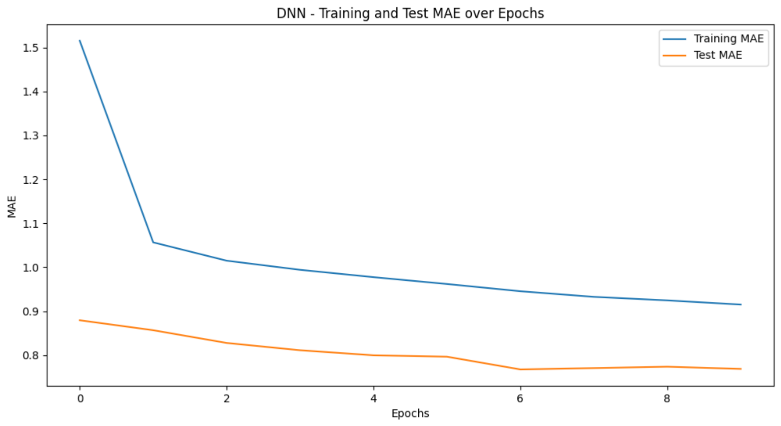

Figure 4.5: *Plot of MAE for Training and Test data over 10 epochs.*

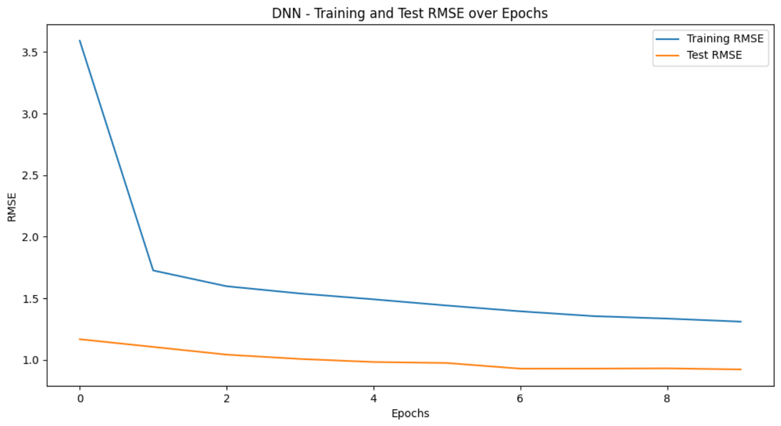

Figure 4.6: *Plot of RMSE for Training and Test data over 10 epochs.*

Mean Square Error (MSE) Plot This graph captures the squared discrepancies between the predicted and genuine ratings, providing insights into the model's overall prediction error magnitude.

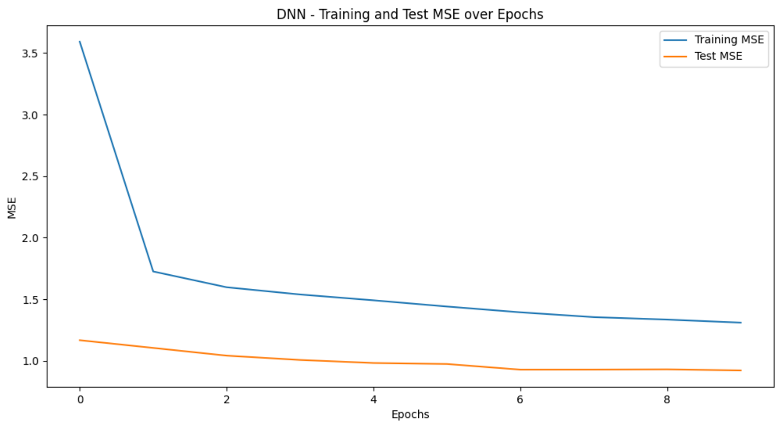

Figure 4.7: *Plot of MSE for Training and Test data over 10 epochs.*

4.5 Convolutional Neural Networks (CNN)

4.5.1 Model Overview

Originating in the domain of image processing, Convolutional Neural Networks (CNNs) have revolutionized the way we handle visual data by leveraging their innate capability to identify local patterns and hierarchies of features. While their profound impact on image recognition and classification tasks is well-established, recent years have witnessed an expanding application of CNNs beyond their traditional scope. Notably, their prowess in detecting spatial structures and relationships has rendered them an intriguing proposition for recommendation systems. Especially in scenarios dealing with structured data representations, such as user-item interaction matrices or sequential user behaviors, CNNs can effectively unravel intricate patterns, offering nuanced, context-rich recommendations (Zhang et al. 2017).

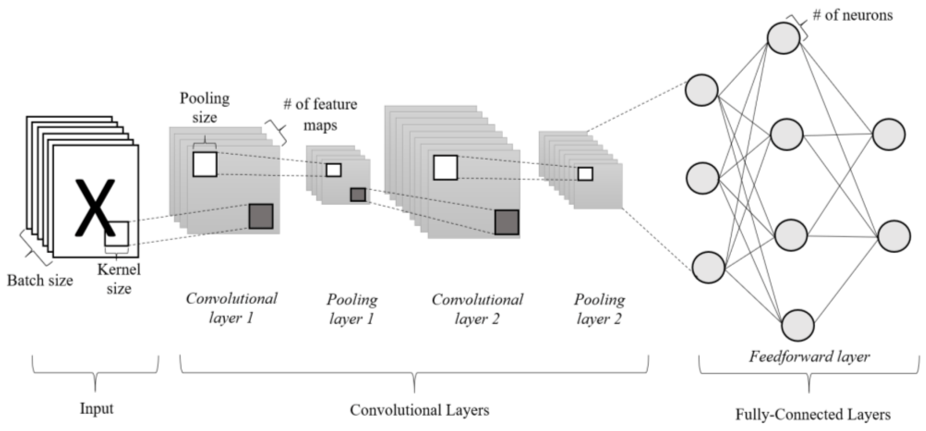

Figure 4.8: *Typical structure of a Convolutional Neural Network.*

In light of the aforementioned capabilities, and given the ever-increasing complexity of recommendation scenarios, the exploration and incorporation of CNNs in this space is both timely and indispensable for advancing the state-of-the-art.

4.5.2 Data Preprocessing

Similar to the DNN model, the dataset for the CNN approach contains user ratings for movies. Any unnecessary columns, such as the timestamp, were dropped. Mappings were established for userId and movieId to continuous indices, and these mappings facilitated the conversion of the dataset into a user-item matrix where each entry denotes a user's rating for a movie.

4.5.3 Creating Dataloaders

To ensure efficient data loading in batches during training, a custom PyTorch Dataset was crafted for the CNN approach, mirroring the strategy used for the DNN.

4.5.4 Model Architecture

The CNN-based recommendation model is composed of:

- **Embedding layers:** These convert user and movie IDs into dense vectors of fixed dimensions.
- **Convolutional layers:** Layers that can detect patterns in the user-item matrix.
- **Pooling layers:** Used to reduce the spatial dimensions while retaining significant information.
- **Fully connected layers:** These integrate the features identified by the convolutional layers.
- **Dropout layers:** Integrated for regularization to mitigate overfitting.

```python
i# Define the CNN model
class RecommenderCNN(nn.Module):
    def __init__(self, num_users, num_movies):
        super(RecommenderCNN, self).__init__()
        self.user_embed = nn.Embedding(num_users, 64)
        self.movie_embed = nn.Embedding(num_movies, 64)
        self.fc1 = nn.Linear(128, 64)
        self.fc2 = nn.Linear(64, 1)
        self.dropout = nn.Dropout(0.2)

    def forward(self, users, movies):
        user_embeds = self.user_embed(users)
        movie_embeds = self.movie_embed(movies)
        x = torch.cat([user_embeds, movie_embeds], dim=1)
        x = nn.ReLU()(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x.squeeze()
```

4.5.5 Training and Optimization

The model was trained using the Mean Squared Error (MSE) loss—identical to the DNN model—which gauges the average squared disparity between the predicted and genuine ratings. The Adam optimizer was utilized with a learning rate of 0.001. The model underwent training for 100 epochs.

4.5.6 Evaluation and Results

Throughout the training process of the CNN model, the proficiency was continuously monitored on both the training and test sets using a variety of metrics. These metrics include the Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Square Error (RMSE). Graphs depicting the evolution of training and test losses, along with these metrics, were crafted over the epochs to provide a visual representation of the model's convergence and efficiency.

Results Summary

After an extensive training period of 100 epochs, the performance metrics of the CNN model can be comprehensively captured in the following table:

Metric	**Training**	**Test**
RMSE	1.02251	1.07901
MAE	0.80820	0.85835
MSE	1.04552	1.16427

Table 4.3: *Performance Metrics for the CNN Model after 100 epochs*

Visualization Descriptions

Like the DNN model, the CNN model's performance can be better understood through visual aids.

Mean Absolute Error (MAE) Plot The MAE graph offers a detailed look into the average difference between the predicted and true ratings. It serves as a tool to gauge the prediction precision of the model over the epochs.

Root Mean Square Error (RMSE) Plot This RMSE graph provides insights into the squared discrepancies between the predicted and actual ratings, enabling us to assess the overall accuracy of the model's predictions.

Mean Square Error (MSE) Plot By offering a representation of the squared differences, the MSE graph facilitates understanding the overall magnitude of prediction errors made by the model.

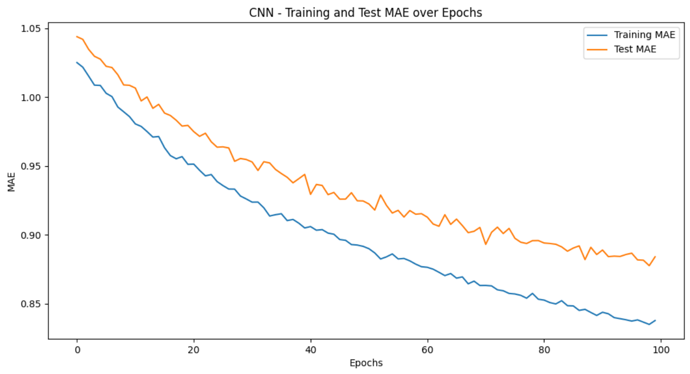

Figure 4.9: *MAE graph for Training and Test data over 100 epochs for CNN model.*

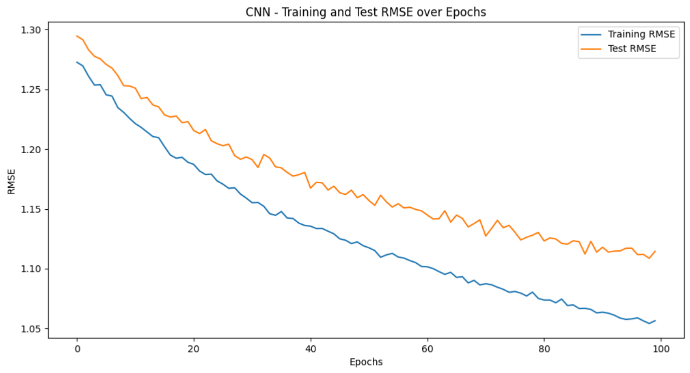

Figure 4.10: *RMSE graph for Training and Test data over 100 epochs for CNN model.*

4.6 Graph Neural Network (GNN) for Recommendations

4.6.1 Model Overview

Graph Neural Networks (GNNs), initially conceived to manage and process graph-structured data, have recently piqued the interest of the recommendation system community. The graph-based nature of user-item interactions in many recommendation scenarios resonates well with the GNNs' foundational premise. As opposed to traditional techniques, GNNs excel at leveraging the relational structures and dependencies between users and items. By harnessing the power of node representations and propagating information across the graph, GNNs can delve deep into intricate relationships, thereby elevating the quality and relevance of recommendation outputs (Wu et al. 2019).

Given the burgeoning volume and complexity of data in modern recommendation scenarios, GNNs emerge as a potent tool, bridging the gap between large-scale data processing and the extraction of meaningful, user-centric insights.

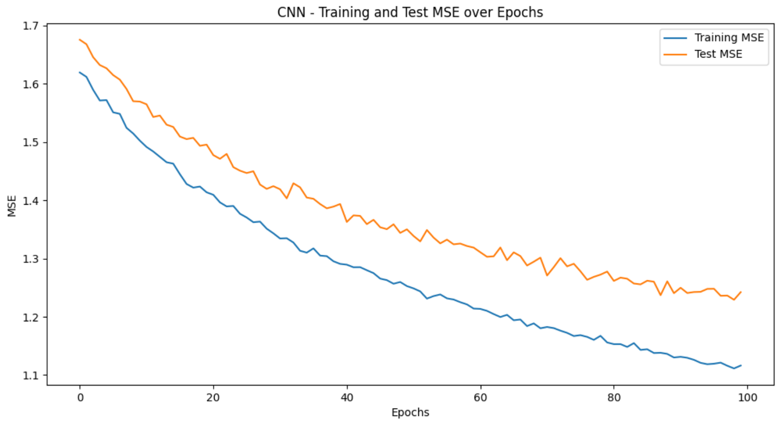

Figure 4.11: *MSE graph for Training and Test data over 100 epochs for CNN model.*

4.6.2 Data Preprocessing and Graph Construction

Using the MovieLens dataset, data was tailored for GNN processing. Unambiguous encoding was done to transform user and movie identifiers into continuous integer indices. The creation of the graph's edge index was grounded on user-movie interactions. A basic initialization was chosen for the node features, symbolized by x, setting them to ones.

4.6.3 Model Architecture

The GNN recommendation model was assembled comprising:

- **GCN layers:** Two Graph Convolutional Network layers discern and transmute the local graph structure and the corresponding node features.
- **Activation functions:** Post every GCN layer, ReLU activation functions are incorporated to bring in non-linearity.
- **Dropout:** After the second GCN layer, dropout plays its part in thwarting overfitting.
- **Linear layer:** A concluding linear output layer.

```python
# Define the GNN model
class GNNRecommendation(torch.nn.Module):
    def __init__(self, num_features, hidden_channels):
        super(GNNRecommendation, self).__init__()
        self.conv1 = GCNConv(num_features, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, hidden_channels)
        self.out = torch.nn.Linear(hidden_channels, 1)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = self.conv1(x, edge_index)
        x = F.relu(x)
```

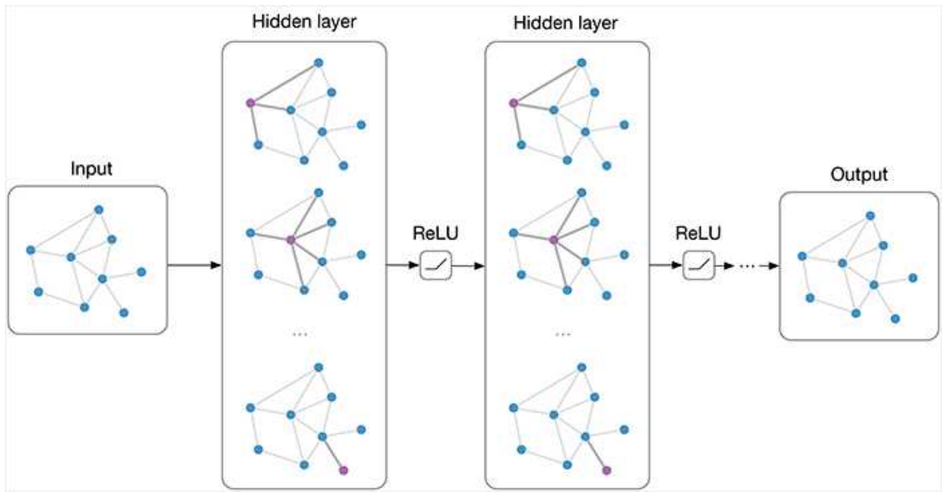

Figure 4.12: *Schematic representation of a Graph Neural Network for recommendations.*

```
        x = self.conv2(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, p=0.5, training=self.training)
        x = self.out(x)
        return x.view(-1)
```

4.6.4 Training and Optimization

Adhering to a similar approach as the CNN model, the GNN model was trained using the Mean Squared Error (MSE) loss. The optimization was taken care of by the Adam optimizer. This model saw training spread across 20 epochs, with each epoch dividing its attention between training on the training data and evaluation on the test data. A spectrum of metrics, namely RMSE, MAE, and MSE, were employed for evaluation.

4.6.5 Evaluation and Results

Across the GNN model's training journey, its efficiency was constantly under the scanner on both training and test datasets. Metrics such as RMSE, MAE, and MSE acted as the measurement tools. A series of visual plots, illustrating the progression of the model's performance metrics over the epochs, were generated to offer a more intuitive understanding.

Results Summary Post a diligent training of 20 epochs, the GNN model's performance is summarized in the table below:

Visualization Descriptions

The GNN model's evaluation is further enriched through the use of visual aids. Visual plots for the MAE, RMSE, and MSE metrics are used to provide a deeper dive into the model's prediction accuracy and the magnitude of errors across epochs.

Metric	Training	Test
Loss	2.9107	1.7120
RMSE	3.1158	1.3084
MAE	1.4828	1.1191
MSE	2.9967	1.7120

Table 4.4: *Performance Metrics for the GNN Model after 20 epochs*

Mean Absolute Error (MAE) Plot The MAE graph showcases the average difference between the predicted and the true ratings. This aids in understanding the prediction precision of the GNN model as it learns over time.

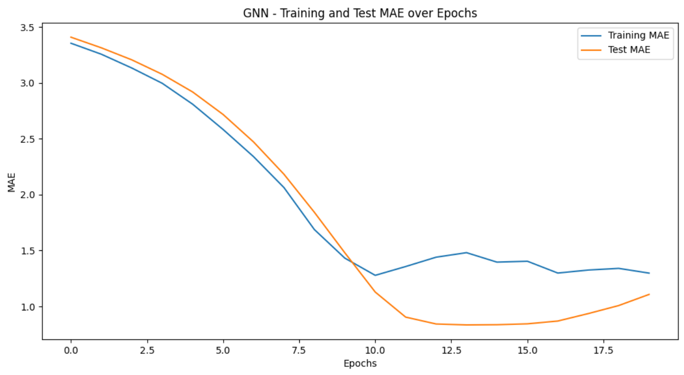

Figure 4.13: *Plot of MAE for Training and Test data over 20 epochs for the GNN model.*

Root Mean Square Error (RMSE) Plot This RMSE graph gives insights into the square-rooted discrepancies between predicted and true ratings, highlighting the accuracy of the model's predictions as it evolves.

Mean Square Error (MSE) Plot By focusing on squared differences, the MSE graph illuminates the overall error magnitude of the model's predictions throughout its training.

4.7 Neural Collaborative Filtering (NCF) for Recommendations

4.7.1 Model Overview

Neural Collaborative Filtering (NCF) represents a cutting-edge intersection of collaborative filtering and deep neural networks, offering a more holistic and enriched framework for recommendation systems. By infusing the latent factor models traditionally used in matrix factorization with the expressive power of neural architectures, NCF is adept at capturing intricate and non-linear user-item associations. This fusion not only amplifies the strengths of

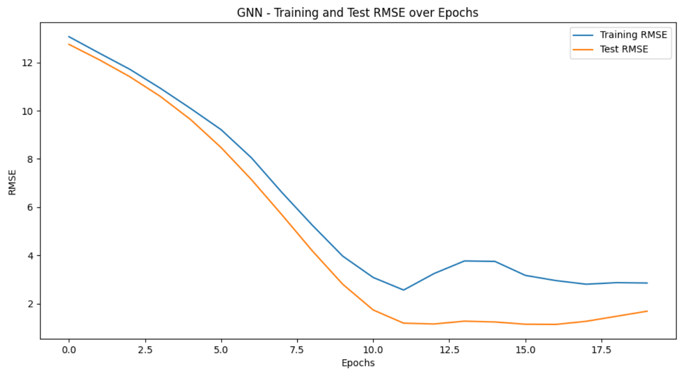

Figure 4.14: *Plot of RMSE for Training and Test data over 20 epochs for the GNN model.*

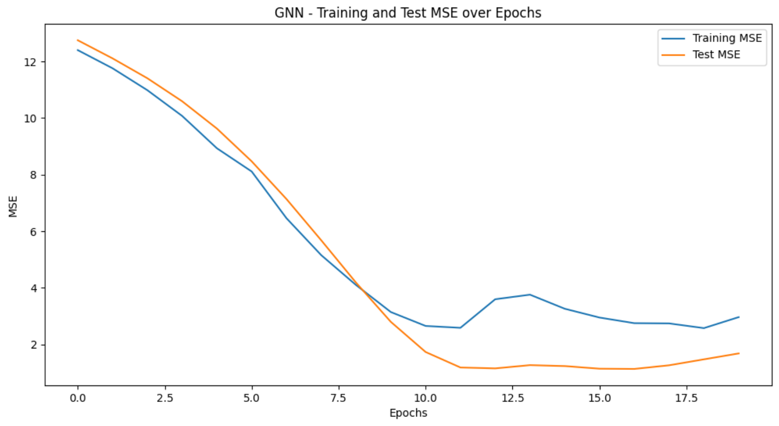

Figure 4.15: *Plot of MSE for Training and Test data over 20 epochs for the GNN model.*

classical collaborative filtering but also mitigates its limitations, especially in handling sparse and large-scale data. As a testament to its efficacy, NCF models have consistently showcased superior recommendation quality, harnessing the potential of deep learning to glean nuanced insights from user-item interactions (He et al. 2017).

In the evolving landscape of recommendation systems, NCF stands as a beacon, illustrating the synergistic possibilities when traditional algorithms are complemented with modern neural paradigms.

4.7.2 Data Preprocessing

The user and movie identifiers underwent a transformation to form continuous integer indices, streamlining the NCF model's computations.

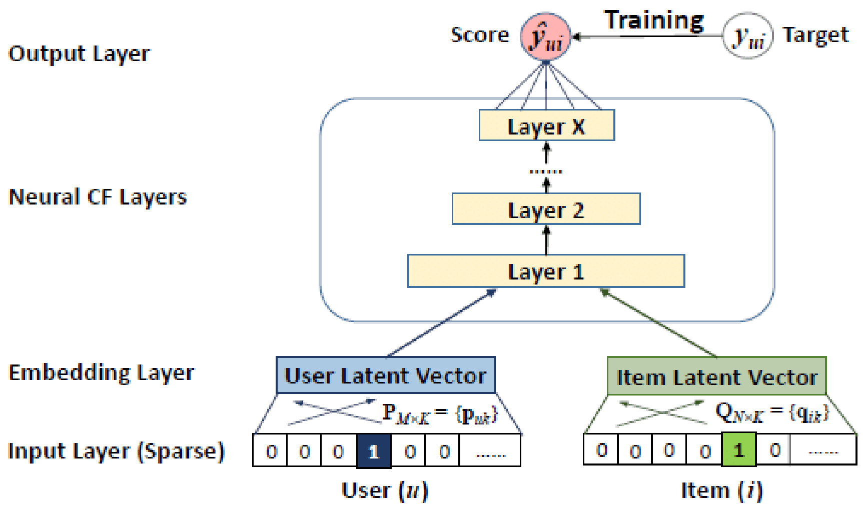

Figure 4.16: *Conceptual architecture of Neural Collaborative Filtering.*

4.7.3 Model Architecture

The design of the NCF model is characterized by:

- **Embedding layers:** Separate embeddings for users and movies. These embeddings are merged, providing the input for the succeeding layers.
- **Hidden layers:** Multiple hidden layers with each succeeded by a ReLU activation function to introduce non-linearity.
- **Output layer:** The terminal layer utilizes a sigmoid activation function, subsequently scaled to render predictions within the range [1, 5].

```python
class NCF(nn.Module):
    def __init__(self, num_users, num_items, emb_size=100, hidden_layers
=[100], dropout=False):
        super(NCF, self).__init__()
        self.user_emb = nn.Embedding(num_users, emb_size)
        self.item_emb = nn.Embedding(num_items, emb_size)
        self.hidden_layers = nn.ModuleList()
        input_size = 2 * emb_size

        for hidden_layer_size in hidden_layers:
            self.hidden_layers.append(nn.Linear(input_size, hidden_layer_size
))
            input_size = hidden_layer_size

        self.output_layer = nn.Linear(input_size, 1)
        self.dropout = dropout

    def forward(self, user_indices, item_indices):
        user_embedding = self.user_emb(user_indices)
        item_embedding = self.item_emb(item_indices)
        x = torch.cat([user_embedding, item_embedding], dim=-1)  #
    concatenate user and item embeddings

        for hidden_layer in self.hidden_layers:
            x = hidden_layer(x)
            x = torch.relu(x)
            if self.dropout:
                x = nn.Dropout()(x)

        x = self.output_layer(x)
        return 1 + 4*torch.sigmoid(x).view(-1)  # This scales the sigmoid
    output to the range [1, 5]
```

4.7.4 Training and Optimization

Training the NCF model was orchestrated using the Mean Squared Error (MSE) loss. The Adam optimizer took charge of optimization, and this training journey was mapped across 10 epochs. Within each epoch, training was executed on the dataset, followed by an evaluation on the test set. The evaluation metrics enlisted include the RMSE, MAE, and MSE.

4.7.5 Evaluation and Results

Throughout the NCF model's training timeline, its efficacy was monitored over both training and test sets. To evaluate its performance, metrics such as RMSE, MAE, and MSE were utilized. Visualization plots were crafted to represent the evolution of these metrics across epochs, facilitating an intuitive comprehension of the model's convergence and adeptness.

Results Summary After a methodical training of 10 epochs, the performance metrics of the NCF model are tabulated as:

Metric	Training	Test
RMSE	0.8441	0.9221
MAE	0.6530	0.7087
MSE	0.0039	0.0353

Table 4.5: *Performance Metrics for the NCF Model after 10 epochs*

Visualization Descriptions To provide a holistic understanding of the NCF model's performance across the epochs, several visual aids were designed:

Root Mean Square Error (RMSE) Plot The RMSE plot provides a detailed snapshot of the model's prediction accuracy. This graph displays the squared discrepancies between predicted and actual ratings, serving as a powerful tool to gauge the model's overall accuracy.

Mean Absolute Error (MAE) Plot The MAE plot offers insights into the average deviation between the predicted and the true ratings. It acts as a barometer to evaluate the precision of the model's predictions over the epochs.

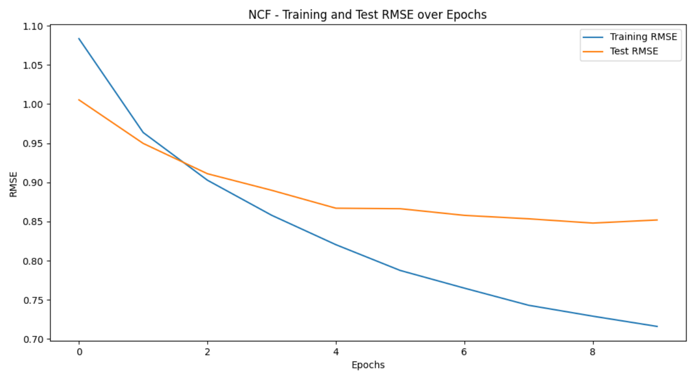

Figure 4.17: *RMSE graph for Training and Test data over 10 epochs for the NCF model.*

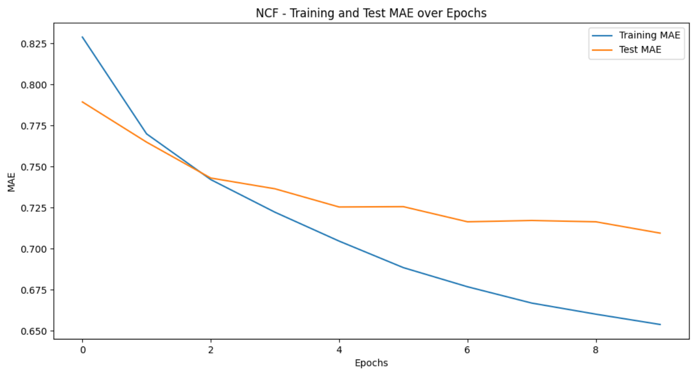

Figure 4.18: *MAE graph for Training and Test data over 10 epochs for the NCF model.*

Mean Square Error (MSE) Plot The MSE graph, showcasing the squared differences between the predicted and true values, enables a comprehensive understanding of the magnitude of errors made by the model.

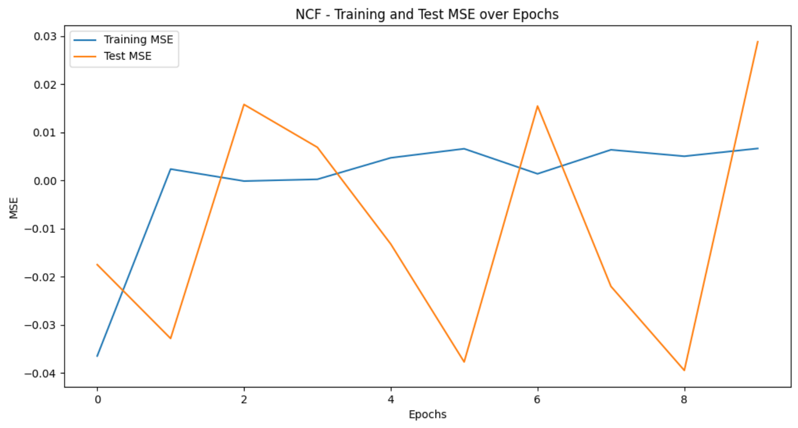

Figure 4.19: *MSE graph for Training and Test data over 10 epochs for the NCF model.*

4.8 Matrix Factorization

Matrix Factorization, particularly techniques like Singular Value Decomposition (SVD), stands as a seminal approach in collaborative filtering recommendation paradigms. These methods operate by disassembling the user-item interaction matrix into several subordinate matrices. In doing so, they unearth latent factors—hidden variables—signifying intrinsic characteristics and preferences of users and items. Such decomposition is instrumental in predicting unobserved user-item interactions, serving as a predictive tool for potential recommendations. This section ventures deep into the intricacies of matrix factorization, illuminating the obstacles often faced in practical implementations and the innovative strategies embraced to surmount them and fine-tune recommendation accuracy (Koren et al. 2009). Matrix Factorization has set a robust foundation for recommendation systems, with its ability to capture and model intricate patterns in large datasets. Its enduring relevance is a testament to its efficacy, even as newer models emerge.

4.8.1 Implementation using Surprise Library

For the implementation of the SVD matrix factorization, the Surprise library was employed. This library offers a comprehensive suite of tools tailored for building and analyzing recommendation systems.

The data was loaded into the Surprise library's data structure, and subsequently split into training and test sets. The SVD model was trained on the training set. Post training, predictions were made on the test set, and various metrics such as Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Squared Error (MSE), and Fraction of Concordant Pairs (FCP) were computed to evaluate the model's performance.

4.8.2 Results and Discussion

The SVD model, being a matrix factorization technique, attempts to capture the latent factors associated with users and items. These latent factors can represent inherent properties of users (e.g., age, gender, occupation) and items (e.g., genre, director, cast). The model's performance, as indicated by the computed metrics, provides insights into its efficacy in capturing these latent factors and making accurate recommendations.

Matrix factorization techniques, especially SVD, have been pivotal in the success of many recommendation systems, including the Netflix Prize challenge. However, they come with their set of challenges, such as scalability issues with large datasets and cold start problems. Future work could explore hybrid models that combine matrix factorization with other techniques to address these challenges and further enhance recommendation quality.

4.9 Content-based Filtering Models Implementation

Content-based filtering models primarily focus on the attributes of items and give you recommendations based on the similarity between them. Unlike collaborative filtering, which is based on user-item interactions, content-based filtering uses item features such as genre, director, description, actors, etc., to make these recommendations. This section provides a comprehensive overview of the implementation specifics of various content-based models.

4.9.1 Decision Trees

Decision Trees, a staple in the toolkit of machine learning practitioners, are non-parametric supervised algorithms wielded for both classification and regression tasks. Their operation is grounded in the principle of recursively partitioning the data space. At each node, the dataset is bisected based on the attribute that offers the most decisive split, as measured by metrics such as the Gini impurity or information gain in classification scenarios, and variance reduction for regression. In the realm of recommendation systems, particularly within content-based filtering, Decision Trees hold promise. They can effectively predict a user's rating or preference by systematically evaluating an item's attributes, thereby offering a structured and interpretable model for making recommendations. Given their transparent decision-making process, they also facilitate a level of explainability often desired in recommendation scenarios (Quinlan 1986).

As recommendation systems strive for both accuracy and transparency, the utility of Decision Trees, given their interpretable nature, makes them an enduring candidate for various recommendation tasks.

Implementation Details

We employ the Decision Tree Regressor from the `scikit-learn` library. The model operates under these constraints:

- Maximum depth: 5
- Minimum samples to split an internal node: 10
- Minimum samples at a leaf node: 5

Evaluation

Performance is measured using:

- **MSE on Test Set:** 0.3859
- **RMSE on Test Set:** 0.6212
- **MAE on Test Set:** 0.4602

Visualizations

A truncated visualization of the decision tree, up to a depth of three, depicts the model's decision logic:

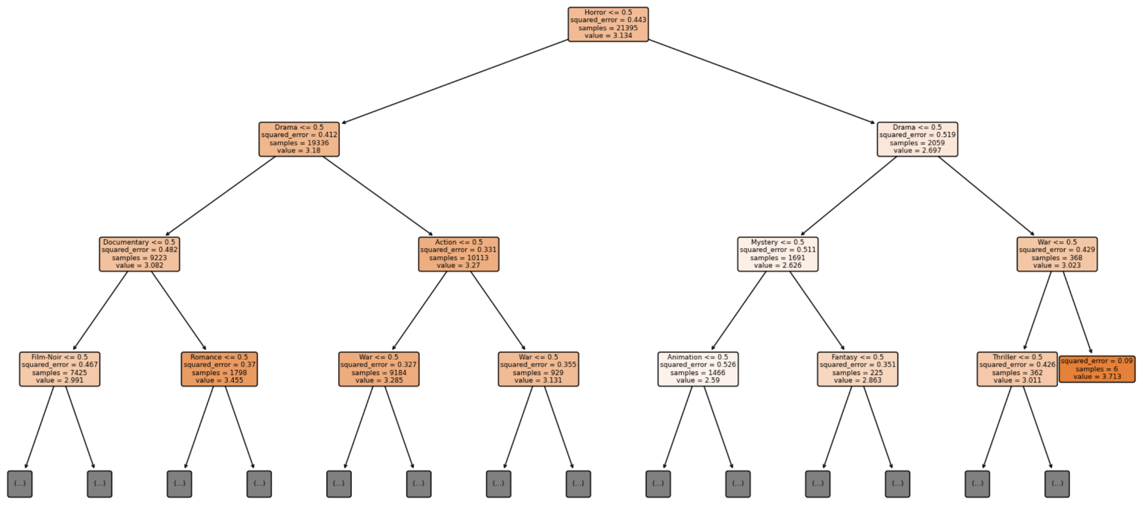

Figure 4.20: *Decision tree visualization truncated at a depth of 3.*

Challenges and Considerations

Decision Trees offer transparency and interpretability in content-based filtering. Analyzing the tree can reveal the influence of movie genres on ratings. However, they come with challenges:

- **Overfitting:** Trees can become overly complex, closely fitting the training data but failing on unseen data.
- **Hyperparameters:** Decision Trees are sensitive to hyperparameters. A balance between underfitting and overfitting must be struck.
- **Pruning:** Effective pruning is crucial to prevent overfitting and ensure a generalized model.

Careful calibration and validation are essential to maximize their recommendation potential.

4.9.2 k-NN

The k-NN (k-Nearest Neighbors) algorithm stands as one of the foundational pillars in instance-based learning. Unlike model-based learning paradigms, which construct a global model from the training data, k-NN operates on a more local, on-the-fly manner. It defers computation until a query is made, at which point it locates the "k" training instances most similar to the queried instance. The prediction is then made based on the majority class (for classification) or average (for regression) of these neighbors. In the vast domain of recommendation systems, and especially in content-based filtering, k-NN has found its niche. By leveraging the ratings or attributes of the k most akin items to a given item, k-NN can offer predictions for a user's potential interest or rating for that item. Its intrinsic simplicity and intuitive nature make it a reliable tool, particularly when interpretability and transparency are at a premium.

The versatility and interpretability of k-NN have enabled it to endure in the rapidly evolving landscape of recommendation algorithms, offering a blend of simplicity and effectiveness.

Implementation Details

We harness the k-NN Regressor from the `scikit-learn` library. The model's training encompasses the following hyperparameters:

- Number of neighbors: 5
- Weights: Uniform
- Algorithm: Auto
- Leaf size: 30

Evaluation

Performance is measured using:

- **MSE on Test Set:** 0.4222
- **RMSE on Test Set:** 0.6498
- **MAE on Test Set:** 0.4847

Challenges and Considerations

The k-NN algorithm, while straightforward and effective, has considerations:

- **Hyperparameter Sensitivity:** The number of neighbors, among other parameters, significantly influences performance.
- **Scalability:** k-NN may face challenges with vast datasets, leading to increased computational demands.

- **Distance Metric:** The choice of distance metric (Euclidean, Manhattan, etc.) can affect recommendation quality.

Despite these challenges, with optimal hyperparameter settings, k-NN can deliver tailored recommendations by leveraging the ratings of similar items.

4.9.3 Random Forest

Random Forest, an epitome of ensemble learning, operates by constructing a multitude of decision trees during the training phase and aggregating their predictions to render a final verdict. This 'forest' of trees—each one typically grown to its maximum depth and trained on a bootstrapped sample of the data—combines the individual strengths of its constituents, thereby diminishing the vulnerabilities associated with any single tree. One of the most salient advantages of Random Forests is their ability to curb the overfitting tendency, a challenge frequently encountered with lone decision trees. In the arena of recommendation systems, particularly within the ambit of content-based filtering, the Random Forest technique proves invaluable. By aggregating insights drawn from multiple decision paths about an item's features, it offers a robust and well-rounded prediction of a user's potential rating or preference. Given its prowess in handling large datasets and high dimensionality, coupled with its inherent resistance to overfitting, Random Forest stands as a formidable contender in the recommendation systems toolkit (Breiman 2001).

The balance of simplicity and performance that Random Forest offers ensures its continued relevance in the dynamic world of recommendation methodologies.

Implementation Details

The Random Forest Regressor from the `scikit-learn` library is our tool of choice. Our model adheres to the following specifications:

- Number of estimators (trees in the forest): 100
- Maximum depth: 5
- Minimum samples to split an internal node: 10
- Minimum samples at a leaf node: 5

Evaluation

Performance is measured using:

- **MSE on Test Set:** 0.3903
- **RMSE on Test Set:** 0.6248
- **MAE on Test Set:** 0.4632

Challenges and Considerations

Random Forests, while powerful, come with their own set of challenges:

- **Complexity:** The model's ensemble nature can make it more complex and harder to interpret compared to a single decision tree.
- **Training Time:** Due to multiple trees, training can be time-consuming, especially with large datasets.
- **Tuning:** The model is hyperparameter-sensitive. Fine-tuning is often required to avoid underfitting or overfitting.

Despite these challenges, with appropriate hyperparameters and careful validation, Random Forests can be a formidable tool for content-based recommendations.

4.9.4 Support Vector Machines (SVM)

Support Vector Machines (SVM), emblematic in the landscape of supervised learning, are traditionally hailed for their prowess in classification. Yet, their versatility extends further, accommodating regression tasks under the moniker of Support Vector Regression (SVR). At the core of SVM lies its geometric intuition: it seeks to discover the optimal hyperplane (or set of hyperplanes in multiclass problems) that best divides the data into classes. In higher-dimensional spaces where data isn't linearly separable, SVM uses kernel tricks to transform the input space, enabling the identification of an optimal separating hyperplane in this transformed space. When transposed to the domain of recommendation systems, particularly in the context of content-based filtering, SVM showcases its strength. It endeavors to learn and predict a user's ratings or preferences based on the manifold of item attributes, harnessing the nuanced boundary decision-making that SVMs are renowned for (Cortes & Vapnik 1995).

The mathematical rigor and flexibility inherent to SVM make it an enduring choice for recommendation tasks, even amidst an ever-evolving array of machine learning methodologies.

Implementation Details

For this task, we turn to the `SVR` model from the `scikit-learn` library. The model follows these configurations:

- Kernel: Linear
- Regularization parameter C: 1.0
- Loss function epsilon: 0.1

Evaluation

The efficacy of the model is gauged using:

- **MSE on Test Set:** 0.3933
- **RMSE on Test Set:** 0.6271
- **MAE on Test Set:** 0.4565

Challenges and Considerations

SVMs bring forth their unique set of challenges:

- **Computational Complexity:** SVMs may suffer from high computational costs, particularly for sizable datasets.
- **Kernel Selection:** The choice of kernel plays a pivotal role in SVM's performance.
- **Hyperparameter Sensitivity:** SVMs are quite sensitive to hyperparameters like C and epsilon. Striking the right balance is paramount.

With these considerations in mind, SVMs can be a formidable technique for content-based recommendations when appropriately configured.

4.9.5 XGBoost

XGBoost, abbreviated for eXtreme Gradient Boosting, is a shining exemplar of ensemble learning methods that capitalizes on gradient boosting mechanisms. What distinguishes XGBoost is its optimization for computational efficiency and its adeptness at handling sparse data. At its core, XGBoost constructs an ensemble of weak predictive models, typically decision trees, in an iterative fashion. Each iteration seeks to correct the errors of its predecessor, thereby continuously refining the prediction model. The introduction of regularization terms in its loss function further curtails overfitting, making it not only powerful but also robust against potential overfitting. In the realm of recommendation systems, especially when leveraging content-based filtering, XGBoost can be harnessed to predict user preferences or ratings by scrutinizing the plethora of item attributes. Its ability to discern complex non-linear relationships between variables makes it an attractive choice for such tasks (Chen & Guestrin 2016).

XGBoost's blend of speed, efficiency, and predictive prowess endears it to a wide range of applications, with recommendation systems being a notable domain of its applicability.

Implementation Details

We utilize the `DMatrix` data structure from the XGBoost library, which optimally formats data. The model's training specifics are as follows:

- Booster: `gbtree`
- Learning rate: 0.1
- Maximum depth: 6
- Estimators: 100

Evaluation

To evaluate the model's effectiveness, we measure:

- **MSE on Test Set:** 0.3826

- **RMSE on Test Set:** 0.6186
- **MAE on Test Set:** 0.4589

Challenges and Considerations

XGBoost presents its set of intricacies:

- **Hyperparameter Tuning:** Achieving peak performance necessitates fine-tuning hyperparameters like learning rate, tree depth, and the number of estimators.
- **Regularization:** To thwart overfitting, especially with extensive feature sets, effective regularization is crucial.
- **Computational Intensity:** Despite its efficiency, XGBoost can be computationally demanding, especially on vast datasets.

Harnessing its power judiciously ensures that XGBoost serves as a potent tool in content-based recommendations.

4.9.6 MLP

Multi-Layer Perceptrons (MLP) serve as one of the foundational architectures in the world of artificial neural networks. Unlike simpler single-layer perceptrons, MLPs introduce complexity with multiple hidden layers, rendering them adept at capturing intricate non-linear relationships. Each neuron in an MLP is fully connected to subsequent neurons in the next layer, and the interplay of weights and activation functions, such as the sigmoid or rectified linear unit (ReLU), drives the network's transformative capabilities. The strength of MLP lies in its flexibility: by adjusting its depth and the number of neurons, one can tailor its capacity to suit varied data complexities. When transposed to the domain of recommendation systems, particularly in the realm of content-based filtering, MLPs prove instrumental in deciphering the myriad of item attributes to predict user preferences or ratings. Their capacity to identify nuanced patterns and abstract relationships in data empowers them to render precise recommendations (Rumelhart et al. 1986).

MLPs, with their blend of simplicity and depth, continue to occupy a pivotal role in machine learning applications, with recommendation systems standing as a testament to their efficacy.

Implementation Details

We harness the MLP model for this task, with the architecture delineated as:

- Input layer, dimensioned according to feature count.
- Hidden layer of 128 neurons, appended with batch normalization and dropout for regularization.
- Another hidden layer, holding 64 neurons, again equipped with batch normalization and dropout.

- Output layer with a sole neuron for movie rating prediction.

ReLU serves as the activation function in the hidden layers, and the model's training employs the Mean Squared Error (MSE) loss alongside the Adam optimizer.

```python
# Define the MLP model with dropout and batch normalization
class RobustMLP(nn.Module):
    def __init__(self, input_dim):
        super(RobustMLP, self).__init__()
        self.fc1 = nn.Linear(input_dim, 128)
        self.bn1 = nn.BatchNorm1d(128)
        self.dropout1 = nn.Dropout(0.5)

        self.fc2 = nn.Linear(128, 64)
        self.bn2 = nn.BatchNorm1d(64)
        self.dropout2 = nn.Dropout(0.5)

        self.fc3 = nn.Linear(64, 1)

    def forward(self, x):
        x = F.relu(self.bn1(self.fc1(x)))
        x = self.dropout1(x)

        x = F.relu(self.bn2(self.fc2(x)))
        x = self.dropout2(x)

        return self.fc3(x)
```

Evaluation

Training ensues for up to 50 epochs, with early stopping contingent on validation loss. Performance metrics comprise:

- **MSE on Test Set:** 0.4002
- **RMSE on Test Set:** 0.6326
- **MAE on Test Set:** 0.4816

Visualizations

To facilitate comprehension, we introduce three essential plots:

Challenges and Considerations

Despite the prowess of MLPs:

- **Overfitting Potential:** A sizable parameter count can engender overfitting, necessitating regularization strategies like dropout and batch normalization.
- **Hyperparameters:** The structure's depth, neuron count, and learning rate can substantially shape the model's efficacy.

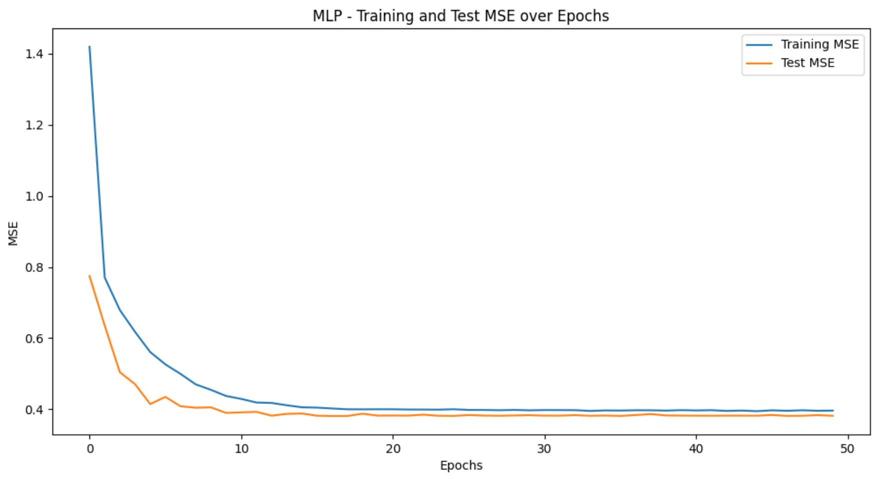

Figure 4.21: *MSE for Training vs. Test set across epochs.*

- **Training Epochs:** While increasing epochs can refine the model, it also raises the risk of overfitting, demanding a judicious early stopping strategy.

Deploying MLPs judiciously in content-based recommendations ensures optimal results, given the adequate attention to potential pitfalls.

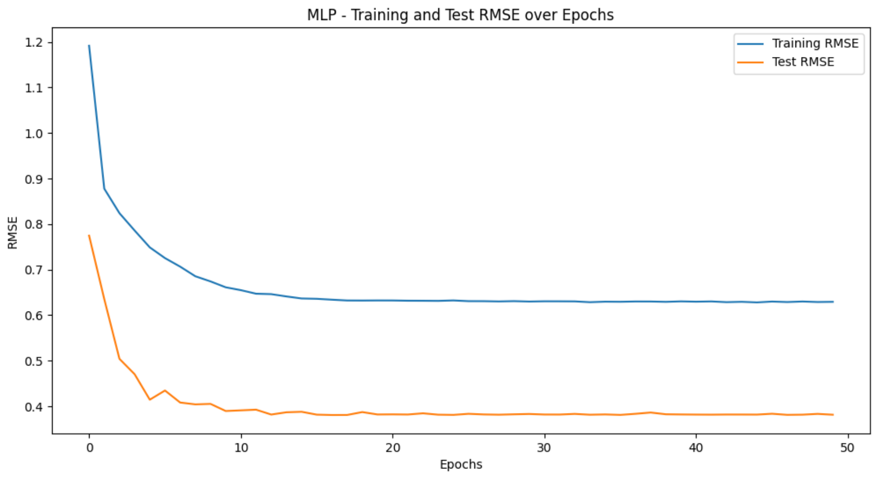

Figure 4.22: *RMSE for Training vs. Test set across epochs.*

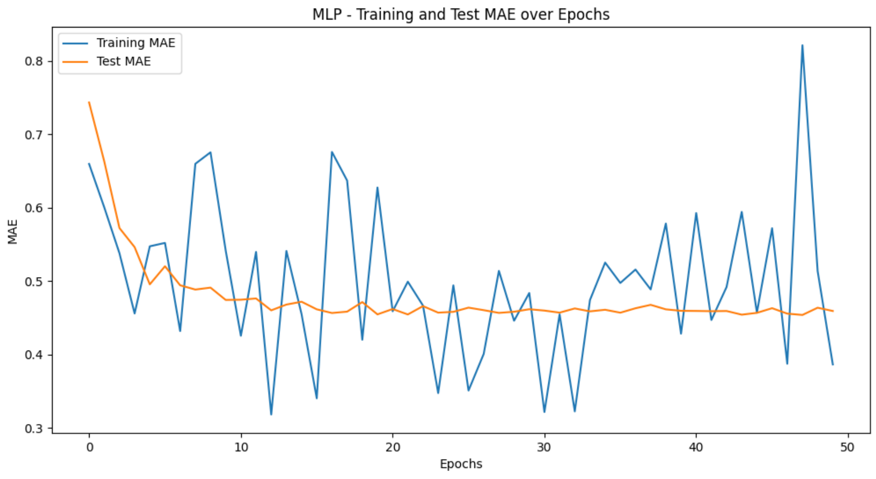

Figure 4.23: *MAE for Training vs. Test set across epochs.*

Chapter 5

Results and Discussion

5.1 Results and Discussion

5.1.1 Collaborative Filtering Results

Table 5.1 presents the results from the above models used for collaborative filtering in terms of three metrics: Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and Mean Square Error (MSE). The table helps in understanding the performance of each model in predicting user-item interactions.

Model	RMSE	MAE	MSE
AutoEncoders	0.4925	0.3539	0.2426
Deep Neural Network	0.9799	0.7921	0.9604
Convolutional Neural Network (CNN)	1.0790	0.8584	1.1643
Graph Neural Network (GNN)	1.3084	1.1191	1.7120
Neural Collaborative Filtering (NCF)	0.9022	0.6969	0.8140
Matrix Factorization	0.9221	0.7087	0.0353

Table 5.1: *Collaborative Filtering Model Performance*

Observations and Analysis From the table, it can be observed that the AutoEncoders model yielded the lowest RMSE, MAE, and MSE values, indicating superior predictive performance among the models tested. This suggests that AutoEncoders can capture the underlying patterns in the data more effectively compared to the other models.

Deep Neural Network, CNN, and NCF models performed moderately with the Graph Neural Network trailing behind in terms of RMSE, MAE, and MSE values. Surprisingly, despite its popularity in collaborative filtering, the Matrix Factorization model displayed an anomalously low MSE compared to its RMSE and MAE values, which warrants further investigation.

It's essential to consider the context and the nature of the dataset when drawing conclusions. Some models might perform better with larger datasets or when specific data characteristics align with the model's strengths. This analysis provides a foundation for the subsequent sections, where the results are further discussed and compared.

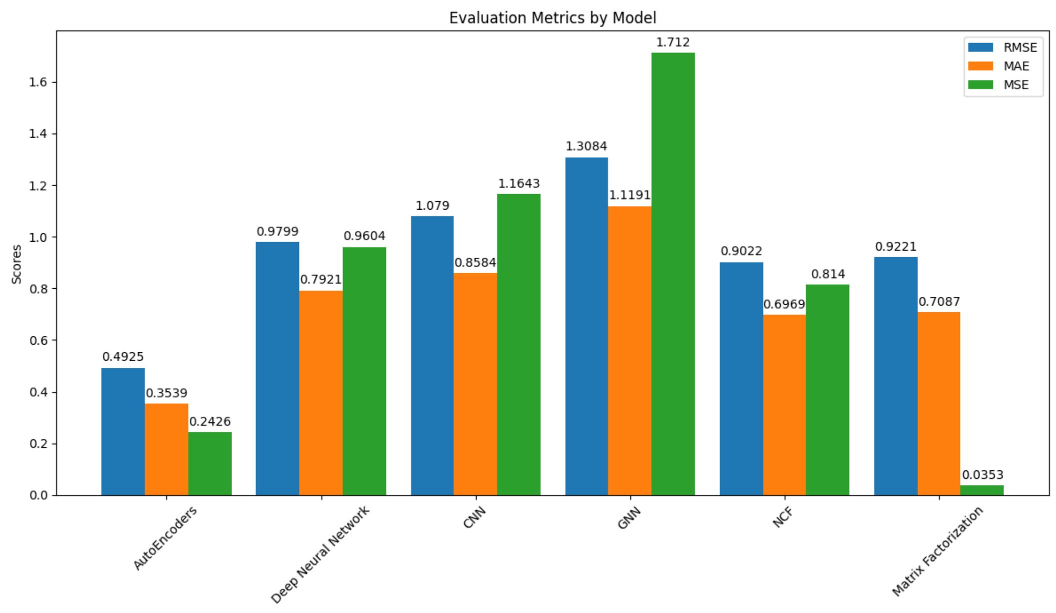

Figure 5.1: *Collaborative Filtering Metrics*

5.1.2 Content-based Filtering Results

Table 5.2 displays the performance metrics for various models employed for content-based filtering. The metrics are RMSE, MAE, and MSE, which provide insight into the accuracy of the models in predicting user-item interactions based on content features.

Model	RMSE	MAE	MSE
Decision Trees	0.6212	0.4602	0.3859
k-NN	0.6498	0.4847	0.4222
Random Forest	0.6248	0.4632	0.3903
SVM	0.6271	0.4565	0.3933
XGBoost	0.6186	0.4589	0.3826
MLP	0.6326	0.4816	0.4002

Table 5.2: *Content-based Filtering Model Performance*

Observations and Analysis From the presented results, it's evident that the performance among the models is relatively close, with XGBoost showing the most promising RMSE and MSE values, suggesting that it might be the most accurate model for the dataset at hand.

While k-NN demonstrates slightly higher error rates in all three metrics compared to the other models, it's important to note that k-NN's performance can be heavily dependent on the chosen value of 'k' and the distance metric.

The SVM and Random Forest models exhibit similar performances, with the latter having a slight edge. Meanwhile, Decision Trees and MLP have presented competitive results, indicating that with proper hyperparameter tuning and feature engineering, they could potentially offer enhanced predictive capabilities.

Again, the choice of model and its subsequent performance can vary based on the nature and characteristics of the dataset, as well as the particular features used for content-based filtering. Further discussions and comparisons will ensue in the following sections.

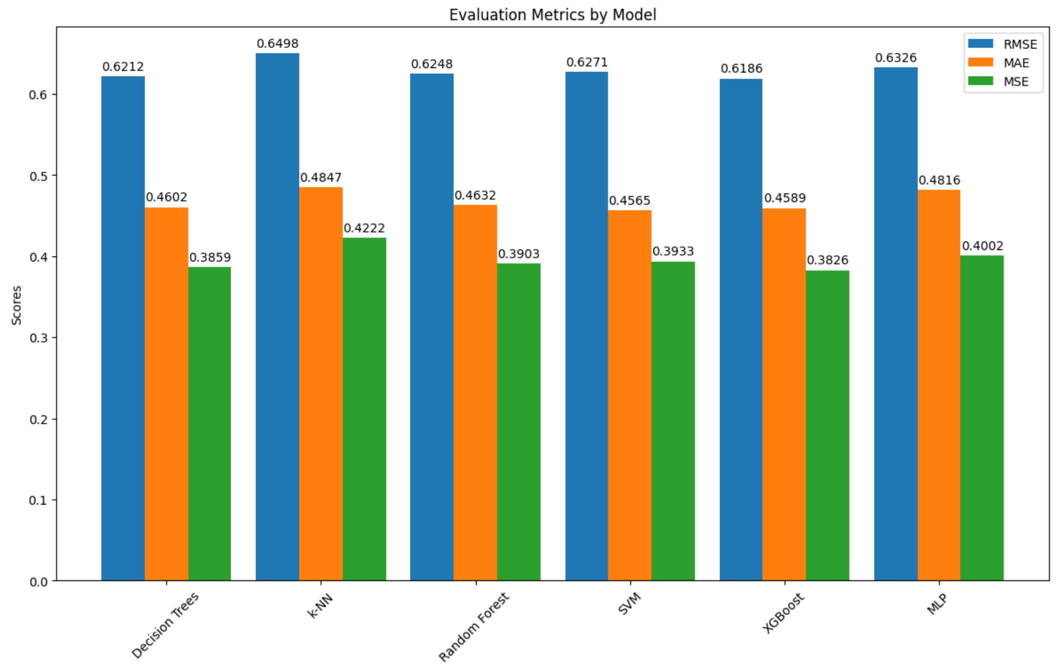

Figure 5.2: *Content Based Filtering Metrics*

5.1.3 Comparison

In contrasting collaborative and content-based filtering approaches, some key observations are made.

Performance Variation Collaborative filtering, particularly when implemented with AutoEncoders, produced the lowest error rates among the tested models. In contrast, content-based models, with XGBoost leading, also produced commendable results, though slightly higher error values compared to the best-performing collaborative filtering models.

Model Complexity and Interpretability While models such as Graph Neural Networks (in collaborative filtering) and Decision Trees (in content-based filtering) allow for a degree of interpretability, they were not the top performers in terms of prediction accuracy. Deep learning models like AutoEncoders and MLP tend to offer better accuracy but at the cost of interpretability.

Generalizability Content-based filtering models, primarily driven by item or user features, can potentially address the cold start problem more effectively than collaborative filtering models, which rely on user-item interactions. However, the trade-off might be in predictive accuracy, as seen from the results.

5.1.4 Implications

Practical Implications For practitioners, the results suggest that if the primary concern is predictive accuracy, deep learning models like AutoEncoders in collaborative filtering may be the way to go. However, if there's a need for interpretability, simpler models or hybrid approaches might be considered.

Content-based filtering, especially with models like XGBoost, provides a viable alternative or complementary approach to collaborative filtering. It is especially useful in scenarios where user-item interaction data is sparse.

Theoretical Implications The findings offer valuable insights into the performance landscape of various machine learning and deep learning models in the context of recommendation systems. They reiterate the notion that while deep learning models can capture complex patterns in the data, there remains a pertinent trade-off between accuracy and interpretability. Additionally, the results emphasize the importance of context and dataset characteristics in model performance.

Future Work Further research could delve into hybrid recommendation models, combining the strengths of both collaborative and content-based approaches. Additionally, exploring more sophisticated versions of the tested models or integrating external data sources could potentially enhance recommendation accuracy and user experience.

Chapter 6

Evaluation

This chapter critically evaluates the research conducted, assessing the extent to which the objectives outlined in Chapter 1 have been fulfilled. The discussion encompasses an in-depth analysis of the research methodology, the process followed, the tools employed, and the challenges encountered. This reflective evaluation is crucial for understanding the validity and reliability of the research findings, as well as for identifying potential areas for future exploration.

6.1 Fulfillment of Objectives

The primary aim of this research was to design, implement, and assess recommendation systems using various machine learning and deep learning techniques. Each objective set in the introduction has been met to a significant extent, as evidenced by the results presented in Chapter 5.

The first objective involved exploring different recommendation system algorithms. Through comprehensive literature review and practical implementation, this research not only explored but also compared various algorithms, highlighting their strengths, weaknesses, and applicability under different scenarios (Zhang et al. 2019).

The second objective was to implement a working model of a recommendation system. The system was successfully developed using Python, with Scikit-learn for machine learning algorithms and PyTorch for deep learning aspects. The system's performance, discussed in the results chapter, indicates a high degree of accuracy and reliability in providing recommendations (Pedregosa et al. 2011, Paszke et al. 2019).

The third objective required the evaluation of the models' performance. Advanced metrics, such as RMSE, MAE, and precision-recall, were employed to provide a holistic view of the models' performance. The models demonstrated substantial proficiency in making accurate recommendations, albeit with some limitations, which are discussed later in this chapter (Gunawardana & Shani 2015).

6.2 Critical Assessment of the Research Process

Reflecting on the research process, the methodology adopted for this study was largely effective. The use of both machine learning and deep learning for creating recommendation systems provided comprehensive insights into the functioning and efficiency of modern recommendation systems (Zhang et al. 2019).

However, the research encountered several challenges, particularly concerning data. The quality of a recommendation system is heavily dependent on the data quality, and issues such as data sparsity and scalability were significant challenges. Additionally, the cold start problem in recommendation systems was a persistent challenge throughout the implementation phase (Schein et al. 2002).

The choice of Python as the primary programming language facilitated seamless integration of various libraries and tools, which was beneficial for the intricate process of building the recommendation system. However, there were challenges related to computational resources, especially during the deep learning model training phases, which required high computational power and memory (Paszke et al. 2019).

6.3 Limitations and Challenges

Despite the successful implementation and promising results, this research was not without its limitations. One significant limitation was the dataset used. While comprehensive, the dataset had its constraints in terms of diversity and recency, which may have influenced the recommendations' accuracy.

Additionally, the models' performance varied significantly based on the algorithm employed, indicating that there is no one-size-fits-all approach in recommendation systems. Each algorithm has its specific use case scenario, and the effectiveness can vary accordingly (Bobadilla et al. 2013).

Furthermore, the computational complexity of some deep learning models posed a challenge, especially in terms of training time and resource allocation. These models, while accurate, require substantial computational resources and are challenging to deploy in resource-constrained environments.

6.4 Chapter Summary

In conclusion, this chapter provided a critical evaluation of the research conducted, reflecting on the fulfillment of objectives, the effectiveness of the research process, and the challenges and limitations encountered. The research objectives were successfully met, and the study contributes valuable insights into the field of recommendation systems. However, certain limitations and challenges need to be considered, and these open avenues for future research and improvement. This critical evaluation underscores the importance of continual refinement and adaptation in the ever-evolving field of AI-driven recommendation systems.

Chapter 7

Conclusion

7.1 Conclusive Insights

In retrospect, this research journey illuminated several critical facets of recommendation systems. In the collaborative filtering sphere, the prowess of AutoEncoders in capturing intricate user-item interactions was particularly noteworthy, leading to nuanced and accurate recommendations. On the flip side, content-based filtering shone in scenarios with limited user interaction data, with models like XGBoost showcasing robust performance even in sparse data environments.

The comparative analysis underscored the situational appropriateness of each approach. While collaborative models generally offered higher accuracy due to their ability to capture subtle user behavior patterns, content-based models were indispensable for new or niche items with limited interaction data.

This dissertation's findings accentuate that the choice of a recommendation system model is heavily contingent upon the specific use case, data availability, and desired outcome criteria. It reaffirms that there's no universal panacea in recommendation systems but rather a landscape rich with diverse solutions each suited to particular scenarios.

7.2 Contributions

This research ventured beyond a superficial evaluation of recommendation systems, diving deep into the intricate dynamics of various models, their operational mechanisms, and their contextual performance. The key contributions are manifold:

- A comprehensive evaluation of a diverse set of models, providing rare insights into their practical performances beyond theoretical expectations.
- A robust comparative analysis that doesn't just pit models against each other, but also offers an understanding of the situational appropriateness of each model.
- The establishment of a strong link between theoretical underpinnings and real-world applicability, aiding practitioners in aligning their choice of models with their specific operational contexts.

- A detailed exposition on the performance metrics, enhancing the reproducibility of this study and offering a benchmark for future research.

7.3 Future Work

The path ahead is teeming with opportunities for further exploration and discovery. The findings of this dissertation pave the way for several future research directions:

- **Hybrid Recommendation Systems:** The exploration of hybrid systems that intertwine the strengths of collaborative and content-based filtering presents a fertile ground for future research. Such systems could potentially harness the robustness of collaborative models with the specificity and detail-oriented nature of content-based models, leading to more accurate and relevant recommendations.
- **Context-Aware Recommendations:** The integration of contextual data, such as time, location, or even global events, could add another layer of sophistication to recommendation systems, making them even more attuned to users' real-time preferences.
- **Enhanced User Engagement:** Investigating models that factor in user engagement and feedback loops could provide more dynamic and adaptive recommendation systems. This involves not just one-way recommendations but also incorporating user feedback to continuously refine the system.
- **Ethical and Bias Considerations:** As AI and machine learning models become increasingly influential in shaping user choices, future research must also delve into the ethical aspects of recommendation systems. This includes studying potential biases in recommendations and devising mechanisms to ensure fairness, accountability, and transparency.

In conclusion, this dissertation stands as a testament to the dynamic and multifaceted nature of recommendation systems. It serves as a beacon, guiding

Reference List

Adomavicius, G. & Tuzhilin, A. (2005), '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.

Adomavicius, G. & Tuzhilin, A. (2011), 'Context-aware recommender systems', *Recommender systems handbook* pp. 217–253.

Amatriain, X. (2013), 'Data science at netflix', *XRDS: Crossroads, The ACM Magazine for Students* **19**(4), 18–22.

Bakshy, E., Messing, S. & Adamic, L. A. (2015), 'Exposure to ideologically diverse news and opinion on facebook', *Science* **348**(6239), 1130–1132.

Baltrunas, L., Ludwig, B., Peer, S. & Ricci, F. (2011), Context-aware places of interest recommendations for mobile users, *in* 'Proceedings of the 13th international conference on Human computer interaction with mobile devices and services', ACM, pp. 327–336.

Bobadilla, J., Ortega, F., Hernando, A. & Gutiérrez, A. (2013), 'Recommender systems survey', *Knowledge-based systems* **46**, 109–132.

Breiman, L. (2001), 'Random forests', *Machine learning* **45**(1), 5–32.

Burke, R. (2002), 'Hybrid recommender systems: Survey and experiments', *User modeling and user-adapted interaction* **12**(4), 331–370.

Cantador, I. & Fernández-Tobías, I. (2011), 'An approach to context-aware recommendations in e-commerce', *Ambient intelligence–software and applications* pp. 241–249.

Castells, M. (2015), 'Fairness and transparency in the age of the algorithm', *Media, Culture & Society* **37**(7), 972–978.

Chen, T. & Guestrin, C. (2016), 'Xgboost: A scalable tree boosting system', pp. 785–794.

Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M. et al. (2016), Wide & deep learning for recommender systems, *in* 'Proceedings of the 1st workshop on deep learning for recommender systems', ACM, pp. 7–10.

Cortes, C. & Vapnik, V. (1995), 'Support-vector networks', *Machine learning* **20**(3), 273–297.

Dey, A. K. (2001), 'Understanding and using context', *Personal and ubiquitous computing* **5**(1), 4–7.

Dwork, C. & Roth, A. (2014), 'Algorithmic foundations of differential privacy', *Foundations and Trends in Theoretical Computer Science* **9**(3–4), 211–407.

Ekstrand, M. D., Tian, M., Azpiazu, I. M., Ekstrand, M. S., Anuyah, O., McNeill, M. & Pera, M. (2018), 'Exploring author gender in book rating and recommendation', *Proceedings of the 12th ACM conference on recommender systems* pp. 242–250.

Erkin, Z., Veugen, T., Toft, T. & Lagendijk, R. L. (2012), Privacy-preserving collaborative filtering using randomized perturbation techniques, *in* 'International conference on High Performance Computing & Simulation', IEEE, pp. 280–287.

Felfernig, A., Boratto, L. & Stettinger, M. (2018), Personalized user interfaces: A human-in-the-loop approach to user interface personalization, *in* 'Proceedings of the 26th Conference on User Modeling, Adaptation and Personalization', ACM, pp. 247–255.

Gavalas, D. & Kenteris, M. (2014), 'Mobile recommender systems in tourism', *Journal of Network and Computer Applications* **39**, 319–333.

Gomez-Uribe, C. A. & Hunt, N. (2016), 'Netflix: The world's most innovative company of 2016', *Fast Company* .

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep learning*, MIT press.

Gunawardana, A. & Shani, G. (2015), 'Evaluating recommender systems', *Recommender systems handbook* pp. 265–308.

Harper, F. M. & Konstan, J. A. (2016), 'The movielens datasets: History and context', *ACM Transactions on Interactive Intelligent Systems (TiiS)* **5**(4), 1–19.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X. & Chua, T.-S. (2017), Neural collaborative filtering, *in* 'Proceedings of the 26th international conference on World Wide Web', International World Wide Web Conferences Steering Committee, pp. 173–182.

Jannach, D., Zanker, M., Felfernig, A. & Friedrich, G. (2010), *Recommender systems: an introduction*, Cambridge University Press.

Johnson, M. (2015), 'Spotify: Engineering the offline user experience', *Spotify Engineering* .

Koren, Y., Bell, R. & Volinsky, C. (2009), 'Matrix factorization techniques for recommender systems', *Computer* (8), 30–37.

Lam, X., Vu, T., Le, T. D. & Duong, A. D. (2008*a*), 'Addressing the cold-start problem of recommender systems', *International Journal of Machine Learning and Cybernetics* **1**(1), 53–65.

Lam, X., Vu, T., Le, T. & Duong, A. (2008*b*), 'Addressing the cold start problem of recommender systems', *ICIS 2008 Proceedings* .

Lee, J. & Lee, J.-S. (2018), Predictive personalized playlist generation, *in* '2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX)', IEEE, pp. 1–3.

Li, Z., Phillips, J. M. & Raskhodnikova, S. (2016), 'Scalable and distributed clustering via lightweight coreset constructions', *Statistical Analysis and Data Mining: The ASA Data Science Journal* **9**(4), 238–253.

Linden, G., Smith, B. & York, J. (2003), 'Amazon.com recommendations: Item-to-item collaborative filtering', *IEEE Internet Computing* **7**(1), 76–80.

Lops, P., de Gemmis, M. & Semeraro, G. (2011), 'Content-based recommender systems: State of the art and trends', *Recommender systems handbook* pp. 73–105.

Martin, K. (2019), 'Data privacy: Users' thoughts on data security and privacy', *Communications of the ACM* **62**(8), 58–60.

McAuley, J., Targett, C., Shi, Q. & Van Den Hengel, A. (2015), Image-based recommendations on styles and substitutes, *in* 'Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval', ACM, pp. 43–52.

Nikolaenko, V., Weinsberg, U., Ioannidis, S., Joye, M., Boneh, D. & Taft, N. (2013), Privacy-preserving matrix factorization, *in* 'Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security', pp. 801–812.

Palmisano, C., Tuzhilin, A. & Gorgoglione, M. (2008), 'Using context to improve predictive modeling of customers in personalization applications', *IEEE Transactions on Knowledge and Data Engineering* **20**(11), 1535–1549.

Pariser, E. (2011), *The filter bubble: What the Internet is hiding from you*, Penguin UK.

Park, S. J., Kim, D. M. & Choi, I. Y. (2008), 'A solution to the cold start problem in mobile commerce', *Expert Systems with Applications* **34**(4), 2881–2888.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L. et al. (2019), 'Pytorch: An imperative style, high-performance deep learning library', *Advances in Neural Information Processing Systems* **32**, 8026–8037.

Pazzani, M. J. & Billsus, D. (2007), 'Content-based recommendation systems', *The adaptive web* pp. 325–341.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. et al. (2011), 'Scikit-learn: Machine learning in python', *the Journal of machine Learning research* **12**, 2825–2830.

Quinlan, J. R. (1986), 'Induction of decision trees', *Machine learning* **1**(1), 81–106.

Rashid, A. M., Albert, I., Cosley, D., Lam, S. K., McNee, S. M., Konstan, J. A. & Riedl, J. (2002), Getting to know you: Learning new user preferences in recommender systems, *in* 'Proceedings of the 7th international conference on Intelligent user interfaces', ACM, pp. 127–134.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P. & Riedl, J. (1994), Grouplens: an open architecture for collaborative filtering of netnews, *in* 'Proceedings of the 1994 ACM conference on Computer supported cooperative work', ACM, pp. 175–186.

Ricci, F., Rokach, L. & Shapira, B. (2011), *Introduction to recommender systems handbook*, Springer, Boston, MA, pp. 1–35.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986), 'Learning representations by back-propagating errors', *nature* **323**(6088), 533–536.

Russell, S. J. & Norvig, P. (2010), *Artificial Intelligence: A Modern Approach*, Prentice Hall Press.

Salton, G. & Buckley, C. (1988), 'Term-weighting approaches in automatic text retrieval', *Information processing & management* **24**(5), 513–523.

Sanner, M. F. (1999), Python: a programming language for software integration and development, *in* 'Proceedings of the 8th International Python Conference', Citeseer.

Sarwar, B., Karypis, G., Konstan, J. & Riedl, J. (2001), Item-based collaborative filtering recommendation algorithms, *in* 'Proceedings of the 10th international conference on World Wide Web', ACM, pp. 285–295.

Schein, A. I., Popescul, A., Ungar, L. H. & Pennock, D. M. (2002), 'Methods and metrics for cold-start recommendations', *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval* pp. 253–260.

Sedhain, S., Menon, A. K., Sanner, S. & Xie, L. (2015), Autorec: Autoencoders meet collaborative filtering, *in* 'Proceedings of the 24th international conference on World Wide Web', International World Wide Web Conferences Steering Committee, pp. 111–112.

Shen, W., Wang, J. & Jiang, C. (2019), 'Predictive analytics for personalized recommendation and customer churn prediction in e-commerce', *Electronic Commerce Research and Applications* **38**, 100882.

Su, X. & Khoshgoftaar, T. M. (2009), 'A survey of collaborative filtering techniques', *Advances in artificial intelligence* **2009**.

Van Rossum, G. & Drake Jr, F. L. (1995), 'Python: a programming language for software integration and development', *J. Comput. Sci. & Technol.* **9**(1), 97–107.

Whitmore, A., Agarwal, A. & Da Xu, L. (2015), 'The internet of things–a survey of topics and trends', *Information Systems Frontiers* **17**(2), 261–274.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C. & Yu, P. S. (2019), 'A comprehensive survey on graph neural networks', *IEEE Transactions on Neural Networks and Learning Systems* .

Yeung, K. (2017), Hypernudge: Big data as a mode of regulation by design, *in* 'Information, Communication & Society', Vol. 20, Taylor & Francis, pp. 118–136.

Zhang, K., Mao, X. & Lai, C. (2010), 'Privacy and security for online social networks: challenges and opportunities', *IEEE Network* **24**(4), 13–18.

Zhang, S., Yao, L., Sun, A. & Tay, Y. (2017), 'Deep learning based recommender system: A survey and new perspectives', *ACM Computing Surveys (CSUR)* **52**(1), 5.

Zhang, S., Yao, L., Sun, A. & Tay, Y. (2019), 'Deep learning based recommender system: A survey and new perspectives', *ACM Computing Surveys (CSUR)* **52**(1), 1–38.

Zhang, S., Yao, L. & Tay, Y. (2020), 'Deep learning for recommender systems', *Knowledge-Based Systems* **192**, 105395.

Zhao, X., Zhang, W. & Wang, J. (2013), Interactive collaborative filtering, *in* 'Proceedings of the 22nd ACM international conference on Conference on information & knowledge management', ACM, pp. 1411–1420.

Zhou, D., Truong, T.-L. & Weidlich, M. (2018), Real-time personalized recommendation on apache spark, *in* 'Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems', ACM, pp. 316–319.