IMPLICIT FEEDBACK DEEP COLLABORATIVE FILTERING PRODUCT RECOMMENDATION SYSTEM

by

Karthik Raja Kalaiselvi Bhaskar

A thesis submitted in conformity with the requirements for the degree of Masters of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

© Copyright 2020 by Karthik Raja Kalaiselvi Bhaskar
Abstract

Implicit Feedback Deep Collaborative Filtering Product Recommendation System

Karthik Raja Kalaiselvi Bhaskar
Masters of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto
2020

In this thesis, several Collaborative Filtering (CF) approaches with latent variable methods were studied using user-item interactions to capture important hidden variations of the sparse customer purchasing behaviours. The hidden latent factors are used to generalize the purchasing pattern of the customers and provide product recommendations. CF with Neural Collaborative Filtering (NCF) was shown to produce the highest NDCG performance on the proprietary dataset provided by the Company. Different hyperparameters were tested for applicability in the CF framework. External data sources like click-data and metrics like Clickthrough Rate (CTR) were reviewed for potential extensions to the work presented. The work presented in this thesis provides techniques, the Company can use to provide product recommendations to enhance revenues, attract new customers, and gain advantages over competitors.
Acknowledgements

I would like to thank my supervisors Dr. Kundur, Chair of Electrical and Computer Engineering and Dr. Lawryshyn, Director of the Centre for Management of Technology and Entrepreneurship for the opportunity to complete my M.A.Sc. and the feedback on this work. I would like to thank the Company and the sponsor for the opportunity and the valuable input. Without the support of my family and friends it would not have been possible to devote the time and attention required to complete my graduate studies.
# Contents

1 Introduction ............................................. 1
   1.1 Introduction ....................................... 1
   1.2 Objective .......................................... 3
   1.3 Contributions ...................................... 3
   1.4 Outline ............................................. 4

2 Literature Review ........................................ 5
   2.1 Introduction to Recommendation Systems .......... 5
      2.1.1 Types of Recommendation Systems ............. 6
      2.1.2 Deep Learning-Based Recommendation Systems ... 7
      2.1.3 The Rating's Matrix ............................ 8
   2.2 Review of Deep Learning Techniques ................. 10
      2.2.1 Multi Layer Perceptrons ....................... 10
      2.2.2 Convolutional Neural Networks (CNN) ........... 11
      2.2.3 Autoencoder .................................... 12
      2.2.4 Recurrent Neural Networks .................... 13
      2.2.5 Deep Semantic Similarity Models ............... 14
      2.2.6 Restricted Boltzmann Machine .................. 15
      2.2.7 Dropout and Batch Normalization ............... 16
      2.2.8 Loss Functions ................................... 17
6 Future Work and Conclusions

6.1 Future Work ......................................................... 66
6.2 Recommendations ..................................................... 69
6.3 Conclusions .............................................................. 70

Bibliography .............................................................. 71
List of Tables

3.1 CF methods selected for analysis ........................................... 41

4.1 Hyperparameters for MF with ALS model ............................... 49

4.2 Hyperparameters for BPR model ............................................ 51

4.3 Hyperparameters for NCF model ............................................ 52

4.4 Hyperparameters for ACF model ............................................ 54

5.1 CF average performance metrics .......................................... 57

5.2 Bayesian Optimization for NCF ............................................. 64
List of Figures

2.1 Rating Matrix ................................................................. 9
2.2 Multi Layer Perceptron with 2 hidden layer ............................ 11
2.3 Input signal for a CNN ...................................................... 11
2.4 Maxpooling Operation ...................................................... 12
2.5 Autoencoder ................................................................. 13
2.6 An unrolled recurrent neural network ................................ 14
2.7 DSSM example ............................................................... 15
2.8 RBM with multiple inputs ................................................ 16
2.9 a) Neural Network without dropout b) Neural network with dropout . . 16
2.10 Ensemble methods .......................................................... 18
2.11 Neural Collaborative Filtering example ............................... 20
2.12 Neural Collaborative Filtering Rating Function ........................ 20
2.13 Neural Collaborative Filtering Loss Function .......................... 21
2.14 Session Based RNN Recommendation System Structure ............ 26
3.1 Neural Collaborative Filtering Algorithm [He et al., 2017] ............... 36
4.1 MF-ALS Training Loss Graph ............................................ 49
4.2 BPR Training Loss Graph .................................................. 51
4.3 NCF Training Loss Graph .................................................. 53
4.4 ACF Training Loss Graph .................................................. 55
Chapter 1

Introduction

This section introduces the problem this thesis strives to address and outlines the objective and contributions. Finally, an overview of the organizational structure of the document is given.

1.1 Introduction

With today’s ever-increasing ease of access to the internet and information, we have reached a point of information overload. Users often find themselves in a dilemma where they spend more time sorting through the clutter of information rather than absorbing the information itself. Over the last decade, there has been a push towards creating accurate and efficient recommendation systems to help users sort through this clutter and improve the experience of obtaining information. However, with the recent increase in research in Machine Learning and Deep Learning, researchers have been experimenting with using deep learning techniques to provide recommendations rather than relying on traditional recommendation techniques like probabilistic matrix factorization [Mnih and Salakhutdinov, 2008] and Naive Bayes classifier [Aiguzhinov et al., 2010].

The recent emergence of information overload has caused decreased productivity, increased frustration, and an overall more negative user experience [Zou and Webster, 2014].
For example, users now often spend more time trying to find a movie on Netflix to watch rather than actually watching movies [Gomez-Uribe and Hunt, 2016].

As with any industry, technology companies ultimately focus on their revenues and profits as key metrics for optimization. Recommendation systems play a significant role in increasing revenues, retaining customers and users, and gaining competitive advantages over competitors [Multiple, 2019]. By using recommendation systems, companies can encourage users to stay engaged with their application/website/service, which allows them to show more advertisements, attract new clients, and retain existing clients [Vaidya and Khachane, 2017]. From an e-commerce perspective, recommendation systems can entice customers to buy more. Carefully selected recommendations can make customers more satisfied, leading them to buy more items than they previously anticipated.

Companies store large amounts of data in databases containing transactional, accounting, inventory, and customer contact information. These data can be analyzed to understand customer purchasing behavior, which allows companies to target customers through product recommendations based on past purchases or activities leading to increased revenue. Companies such as Amazon and Netflix are already using algorithms to give recommendations, and other companies are looking to implement similar recommendation system algorithms. The development of e-commerce solutions pioneered by Amazon not only demonstrated the importance of incorporating web data to meet customers’ shopping needs but also leveraging customer data to enhance sales using machine learning algorithms. In another example, Uber Eats food recommendations are based on many factors that take into account real-time information such as user queries, location, time of day/week and historical information about the user’s purchases, restaurants and delivery partners used [Yuyan Wang and Zhang, 2019].

A large parts supply company (the Company) has gathered a large amount of sales and transactional information and has an extensive product database. However, the
Company performed minimal analysis of their databases from the perspective of product recommendations to customers before this project was started. Furthermore, the Company’s databases are large, containing more than 500,000 stock keeping units (SKUs), 20 million rows of transactional data as well as information related to their 200,000 customers. The two primary databases are Invoiced Orders (Invoicedorders.csv) and Current Items (ITEMCURRENT.csv). The “Invoiced Orders” database contains the customers’ transaction data. The “Current Items” database contains detailed information about the products offered by the Company. The Company wants to use the available data to provide a list of 12 products the customers may be interested in purchasing and to improve the user experience of the Company’s website.

1.2 Objective

The objective of this thesis is to build a recommendation system for the Company’s online customers using machine learning techniques, which will allow the Company to predict their customers’ needs better and provide product recommendations tailored to the customers’ purchasing behaviors.

1.3 Contributions

The contributions associated with this work are listed below:

- Pre-processed the Invoiced Orders and Current Items databases from the Company and made the data suitable for the recommendation system problem.

- To test the hypothesis that Collaborative Filtering with latent variable models can capture important hidden variations of sparse customer purchasing behaviors, which will generalize the purchasing pattern of the customers and provide personalized product recommendations. The thesis work continues related efforts in
[Hu et al., 2008], [He et al., 2017], [Rendle et al., 2009], [Moussawi, 2018] to understand the hidden factors in customers’ purchasing behaviors

- The construction and evaluation (in terms of ranking metrics) of four different models based on Matrix Factorization using Alternative Least Squares, Bayesian Personalized Ranking, Neural Collaborative Filtering and Autoencoder Collaborative Filtering.

### 1.4 Outline

A review of different recommender systems for analyzing user/item transactional data and their applicability is presented in Chapter 2. Chapter 3 will outline collaborative filtering and the different methodologies used for this work. The different modelling details and evaluation criteria are presented in Chapter 4, along with an exploration of the databases provided by the Company. Chapter 5, presents the results and discussions. In Chapter 6, conclusions are presented along with directions of future work and recommendations.
Chapter 2

Literature Review

This Chapter will provide an overview of what a recommendation system is and what information can be used as input in these systems. We will also discuss what a rating matrix is and why it is a critical component of all recommendation systems. In addition, we will provide an overview of why deep learning-based techniques are becoming more popular than traditional recommendation techniques.

Section 2.1 provides the necessary background information regarding the recommendation system and the rating matrix. Section 2.2 outlines the fundamentals of deep learning techniques. In section 2.3, deep learning-based modeling techniques will be reviewed. Section 2.4 will outline the existing methodologies for deep learning-based recommendation systems.

2.1 Introduction to Recommendation Systems

A recommendation system, often also called a “recommender system”, is used to estimate user preferences of items or objects they have not seen yet. Recommendation systems commonly use inputs such as user preferences, item/object features, histories of users-items’ past interactions, temporal data, and spatial data.

Recommendations provided by the recommender systems come in two different forms
1. Rating Predictions

- Rating Predictions attempt to predict the rating to a user for an item that has not been rated by the user. Predictions are computed based on the user’s past ratings on a set of items. An example of rating prediction is predicting whether a particular user will like a specific TV show on Netflix based on historical behavior.

2. Ranking Predictions

- Ranking predictions aim to create a “Top-N” ranked list with N items for the user. For example, a system like ranking prediction may recommend the “Top 10 books to read if you like Harry Potter”.

In the thesis, we will be focusing on building a recommendation system based on ranking predictions rather than rating predictions, because the Company wants a list of recommended products instead of ratings.

2.1.1 Types of Recommendation Systems

Generally, there are three types of recommendation systems: Collaborative Filtering systems [Ekstrand et al., 2011], [Zhao and Shang, 2010], [Sarwar et al., 2001], Content-Based Recommendation systems [Zenebe and Norcio, 2009], [Lops et al., 2011], and Hybrid systems[Burke, 2002], [Burke, 2007]. All three types face similar issues with data sparsity, which means the phenomenon of not having enough data to model/build a recommendation system; however, these different types of recommender systems try to resolve the data sparsity issue in various ways.

Collaborative filtering systems [Shambour et al., 2016], [He et al., 2017] primarily learn past user-item interactions. For example, if user $\alpha$ likes item $a$ and item $b$, and user $\beta$
likes item \( b \), then collaborative filtering systems may assume that user \( \beta \) likes item \( a \) as well. Collaborative filtering systems can use explicit or implicit learning techniques. Explicit learning involves a denotative rating; for example, a user explicitly rates a movie as 5 stars. Implicit learning, on the other hand, involves ratings derived from other actions; for example, a user watches many action movies, so action movies are rated as higher for that user. Collaborative filtering systems often ask the question, “What do people like me like?”

Content-based recommendation systems [Kim et al., 2014], [Cantador et al., 2010] use comparisons between item/object information and auxiliary user information. The system tries to understand what the user likes and what items and objects are similar to the user’s preferences.

The last type of recommendation systems are hybrid systems. Hybrid systems [Burke, 2002], [Bostandjievi et al., 2012], [Strub et al., 2016] attempt to integrate content and collaborative-based recommendation strategies to create a more robust and efficient system. The thesis is focused on studying Collaborative Filtering type recommendation systems.

### 2.1.2 Deep Learning-Based Recommendation Systems

Deep Learning [Goodfellow et al., 2016], [LeCun et al., 2015] is a subset within machine learning and is concerned with constructing neural networks with numerous hidden layers. Deep learning systems are universal function approximators that take input and create a mapping function into an output space. Hence, deep learning systems attempt to create an accurate mapping function approximator to predict outputs for inputs they have never seen before. Because of this advantage, deep learning-based recommendation systems [Liu and Wu, 2017] can capture non-linear and nontrivial user-item relationships that are not easily detectable by humans or even traditional recommendation system techniques. Deep learning-based techniques allow for more complex abstractions of data, which in return allows for the detection of hidden patterns. Additionally, deep learning
techniques can capture patterns and relationships within data. This combination of capabilities is useful for achieving accurate, efficient recommendations, especially when data are limited.

In addition, there has been a recent increase in research and publications on deep learning applications [Ker et al., 2017], [Kamilaris and Prenafeta-Boldú, 2018]. This increase is providing new, exciting techniques to be used in various domains, including recommendation systems [Cheng et al., 2016], [Zhang et al., 2019]. Moreover, through access to mass, parallelized computing power utilizing graphics processing units (GPUs) and tensor processing units (TPUs) [Patterson, 2018], deep learning-based recommendation systems are becoming more tractable for significant industry use cases.

2.1.3 The Rating’s Matrix

An essential part of traditional and deep learning-based recommendation systems is the rating matrix, also called a user-item matrix [Koren et al., 2009]. The rating matrix allows quantitative and qualitative data about user-item relations to be efficiently stored in matrix form to allow for simplified mathematical computation when generating recommendations. As with any other deep learning technique, the input format of any of this type of system originates from some vector of information, which is then used to generate recommendations. This section explains the structure of the rating matrix and its shortfalls, which must be taken into account to generate accurate recommendations.

In rating matrix or user-item matrix (Figure 2.1), each row represents a user and each column represents an item and each cell represents a rating given by a user to an item. There are total $n$ users and $m$ items. $A_{ij}$ is the rating given by user $U_i$ on item $I_j$. For example, $A_{ij}$ can be a range from 1 to 5 in movie ratings; sometimes $A_{ij}$ can also be binary, if a matrix represents whether a user $U_i$ has purchased an item $I_j$ or not, that can be represented in $A_{ij}$ as either 0 or 1. Typically, the rating matrix is a sparse matrix, which means that many cells in user-item matrix are empty i.e. zeroes.
While the rating matrix structure does make it simpler to visualize and understand the relationships between individual users and items/objects, there is a significant issue of data sparsity [Sarwar et al., 2001]. This problem is especially exacerbated in rating matrices with massive catalogs of objects/items. The issue is that user $\alpha$ may have rated items $\beta_1, \beta_2, \beta_3$; however, in practical situations, the user never rated every item $\beta_1...\beta_N$. Realistically, each user has probably rated only a tiny portion of the entire catalog, results in a very sparse rating matrix. Sparse matrices like this cause many problems in both traditional and deep learning-based recommendation systems, such as poor recommendations and the numerical instability of mathematical operations being done on values inside the matrix. The techniques presented below demonstrate techniques used by researchers to try to mitigate these issues at scale.

**Figure 2.1: Rating Matrix**
2.2 Review of Deep Learning Techniques

In this section, we will give a brief review of various deep learning techniques in a generalized context. These techniques will include multi layer perceptron, convolutional neural networks, autoencoders, convolutional neural networks, recurrent neural networks, and restrictive boltzmann machines.

2.2.1 Multi Layer Perceptrons

Multiple-layer perceptrons (MLPs), also known as Artificial Neural Networks (ANNs) or feedforward neural networks, are a more general class of machine learning algorithms that try to learn non-linear decision boundaries for a dataset. MLPs are not necessarily just binary classifiers, they can act as multi-class classifiers if needed. An example is provided in Figure 2.2 [Goodfellow et al., 2016] and they consist of three layers:

- Input layer - acts as the input for the features of the data. No computations is performed in any of the input nodes.

- Hidden layer - acts as an intermediate layer between input and output layer, where all the computations like matrix multiplication takes place. It can have one to multiple hidden layers.

- Output layer - produce the results/outputs by applying non linear transformation to the outputs of the hidden layer.
2.2.2 Convolutional Neural Networks (CNN)

In the setting of image data, each image can be interpreted as a 3D input signal (i.e., pixel width and pixel height) with red, green and blue values for each pixel as shown in Figure 2.3 [Karpathy, 2019] The convolutional layer does convolution operation, multiplies a number of filter matrices over the input signal to generate an output map, accepting stride (the amount by which the filter shifts) values for each filter (over the horizontal and vertical directions). Zero-padding refers to the process of symmetrically adding zeros to the input matrix, can be added to the image to guarantee that the matrix operations between the original input and the filters are suitable.
Pooling layers are typically used after convolutional layers, which performs feature down-sampling. These layers have two parameters, horizontal and vertical stride. These layers slide over the feature map depending on the type of pooling (i.e., minimum, maximum, average pooling), the appropriate value of that feature map is obtained. A max-pooling operation is shown in Figure 2.4 [Karpathy, 2019], here shown with a stride of 2. That is, the maximum number is taken over 4 numbers (little 2x2 square) throughout the matrix.

![Maxpooling Operation](image)

**Figure 2.4:** Maxpooling Operation

### 2.2.3 Autoencoder

Autoencoders [Ng et al., 2011] are a class of unsupervised machine learning models. An autoencoder is essentially a type of neural network that tries to reconstruct input data at the output layer. For example, unlike a typical neural network where the training set input is X and the training set output is Y, where we try to predict Y; when training an autoencoder, the training set input is X, and the training set output is also X. For example, in Figure 2.5, the input to the autoencoder is an image of size 16 x 16 (256 input nodes). The inputs are passed to the encoding layer, where the model learns how to reduce the input dimensions into an encoded representation. The encoded representation is converted to a compressed representation, also called a bottleneck layer, which is the lowest possible dimension of the input data. The decoding layer tries to learn how to reconstruct the data from the compressed representations to be as close
as the original input image. Therefore, autoencoders are essentially trying to learn the underlying encoding of the information in the data set. Autoencoders are typically used in dimensionality reduction [Hinton and Salakhutdinov, 2006]. Recently, special types of autoencoders called variational autoencoders [An and Cho, 2015] have been used to learn generative models of data. These work by trying to learn a generative model of the input data; hence, when unseen data are presented to the variational autoencoder, the model can generate new information based on the probabilistic model it has created.

![Figure 2.5: Autoencoder](image)

### 2.2.4 Recurrent Neural Networks

Recurrent neural networks (RNN) [Schmidhuber, 2015] are used for modeling sequential and time-sequenced data. Unlike regular feedforward neural networks, recurrent neural networks have loops and memories inside the network. This allows the network to “remember” and “forget” former computations. The right-hand side Figure 2.6 [Olah, 2015] shows the unrolled version of a RNN. The RNN network gets the first input $x(0)$, from the sequence of input, and it outputs $h(0)$ using activation function (A) of the node, which is used to determine the output of that node given an input $x(0)$. The most commonly used activation functions are ReLU, sigmoid, tanh, etc. The sigmoid activation function,
also called the logistic function, is traditionally a very popular activation function for neural networks. The input to the function is transformed into a value between 0 and 1. Tanh, is a similar nonlinear activation function that outputs values between -1 and 1. The rectified linear activation function (ReLU) is a simple calculation that returns the value provided as input directly, or the value 0 if the input is 0 or less. The output $h(0)$ together with $x(1)$, are the inputs for the next step, which outputs $h(1)$ and so on to the final time step $t$. There are many different variants including Long Short-Term Memory cells (LSTM) [Hochreiter and Schmidhuber, 1997] and Gated Recurrent Units (GRU) [Jozefowicz et al., 2015]. These techniques are used to combat the vanishing gradient problem [Hochreiter, 1998], which occurs as a result of loops inside the recurrent neural network that make it difficult to model long sequences of data.

![Figure 2.6: An unrolled recurrent neural network](image)

### 2.2.5 Deep Semantic Similarity Models

Deep Semantic Similarity Models (DSSM) [Huang et al., 2013], [Yang et al., 2018] are a type of deep neural network used for learning semantic representations of problems/entities. They are also used to measure semantic similarities between the two models. In Figure 2.7 [Yang et al., 2018], we want to find the semantic textual similarity. The intuition for the DSSM model is that the sentences are semantically similar if they have related responses. For example, the two questions are about age, “How old are you?” and “What is your age?”. These questions can be answered in almost similar responses such as “I am 20 years old”. In contrast, the other set of questions are “How are you?” and “How
old are you?” contain practically identical words; they have very distinct meanings and guide to different answers.

![DSSM example](image)

**Figure 2.7:** DSSM example

### 2.2.6 Restricted Boltzmann Machine

A Restricted Boltzmann Machine (RBM) [Sutskever et al., 2009] is a special type of neural network that only has two layers: a visible layer and a hidden layer. There are no output layers. These sorts of machines can often be “stacked” to create a deep network. The restricted portion of the Restricted Boltzmann Machine means that there are no intralayer communications at either layer. RBM is stochastic in nature, which means that each node will have some random behavior when activated. The RBM looks like Figure 2.8 with multiple inputs, which are multiplied with weights and bias in the visible layer and passed through an activation function, like sigmoid, which determines if the hidden state gets activated or not.
2.2.7 Dropout and Batch Normalization

Overfitting is the phenomenon that happens when a machine learning model is incapable of generalizing its predictions outside of the training set due to the model memorizing characteristics of the training data. The deployment of such a model into production could prove to be disastrous. There are methods like dropout and batch normalization to control neural network models from overfitting. Dropout [Srivastava et al., 2014] is a technique used during the training process to control overfitting by randomly dropping out a subset of neurons within the layer with probability $p$. An example of dropout is shown in Figure 2.9.

Figure 2.8: RBM with multiple inputs

Figure 2.9: a) Neural Network without dropout b) Neural network with dropout
Batch Normalization [Ioffe and Szegedy, 2015] is a technique that uses the mean $\mu$ and standard deviation $\sigma$ of a minibatch of data to normalize/standardize the outputs of one layer before processing downstream layers.

### 2.2.8 Loss Functions

The main intention of a neural network is to find a good set of weights so that the network learns to map a set of inputs to a set of outputs from the training data. In order to find a good set of weights, the learning problem is posed as an optimization problem where we can navigate through the possible set of weights to find a desirable set of weights to make good enough predictions. The function used to evaluate those set of weights is referred to as the objective function. Typically in neural networks, we want to minimize the objective function also called a loss function or cost function [Goodfellow et al., 2016]. There are various loss functions in literature like maximum likelihood estimation [White, 1982], focal loss [Lin et al., 2017], generalized cross-entropy loss [Zhang and Sabuncu, 2018], etc.

Section 2.2 introduced a number of general deep learning concepts, some of which will be used to build the recommendation system.

### 2.3 Deep Learning Modeling Techniques

There are many important design decisions that the designer of a deep learning-based recommendation system must make that will ultimately influence the efficiency and accuracy of the system being created. While designing the deep learning part of the recommendation system, the designer has two options for how to use deep learning. The first method is to use a single deep learning technique [Alashkar et al., 2017], [Bansal et al., 2016], [Berg et al., 2017]; for example, using only a recurrent neural network to recommend blogs, social media post [Bansal et al., 2016] or using only a DSSM [Huang et al., 2013]
to attempt to perform semantic matching between users and items/products.

On the other hand, the second method is to use deep composite methods, also referred to as Ensemble methods [Tsai and Hung, 2012]. For example, in Figure 2.10, three different deep learning algorithms called Model A, Model B and Model C are used to make predictions. In Ensemble methods, the deep learning recommendation model uses multiple deep learning techniques and the results are then averaged as demonstrated by [Ebesu and Fang, 2017] and [Chen et al., 2017]. Ensemble methods can be run either in parallel or in a sequential fashion. The benefit of using multiple models is that deep learning models often complement each other and can, therefore, create a more powerful hybrid model [Wang and Wang, 2014]. The issue with the ensemble approach is that the more deep learning techniques are used, the interpretability [Doshi-Velez and Kim, 2017] of the model is reduced.

![Figure 2.10: Ensemble methods](image-url)
2.4 Survey of Deep-Learning Based Recommendation Systems

In this section, we will discuss the most popular deep learning-based recommendation systems and techniques. We will focus on high-level intuition rather than technical details to allow the reader to absorb more practical and usable knowledge of these techniques. We will also explore techniques that are relatively new and still being researched.

Section 2.4.1 outlines the existing methodologies for MLP based recommendation systems. In section 2.4.2, literature regarding autoencoder based recommendation systems will be reviewed. Section 2.4.3 describes the literature for CNN based recommendation systems. Section 2.4.4 and 2.4.5 outlines the literature for RNN and RBM based recommendation systems.

2.4.1 Multiple Layer Perceptron Recommendation Systems

A popular deep learning-based recommendation technique that utilize multi-layer perceptrons are neural collaborative filtering [He et al., 2017] and deep factorization machine [Guo et al., 2017]. In MLP techniques, recommendations are considered as a two-way interactions between users’ preferences and the features of an item/object. Therefore, neural collaborative filtering or deep factorization machine, attempts to build a feed-forward network to model the two-way interactions and therefore the feed-forward model captures the non-linear relationships between items and users.
Figure 2.11: Neural Collaborative Filtering example

An embedding is a mapping of a discrete or categorical variable to a vector of continuous numbers. Embedding is an important step in the MLP based recommender systems. As the input layer (user preferences and item features) can be a large and sparse vector, it is crucial to reduce the dimensionality of the input to create a denser, more informative vector. Let \( s^\text{user}_u \) and \( s^\text{item}_i \) denote the user profile of user \( u \) and item features of item \( i \). The embedding layer as shown in Figure 2.11 is responsible for taking sparse user inputs \( s^\text{user}_u \) and sparse item inputs \( s^\text{item}_i \), and converting them into a dense user embedded latent vector \( U \) and a dense item feature embedded vector \( V \).

\[
\hat{r}_{ui} = f(U^T \cdot s^\text{user}_u, V^T \cdot s^\text{item}_i | U, V, \theta)
\]

Figure 2.12: Neural Collaborative Filtering Rating Function

The rating or scoring function (\( \hat{r}_{ui} \)) is defined in the Figure 2.12, where \( f(.) \) is repre-
sents a MLP model and $\theta$ represents the parameters of the MLP model. The loss function $L$ (shown in Figure 2.13) can be calculated by taking the weighted square error between the difference between user $u'$'s predicted rating of item $i$ ($\hat{r}_{ui}$) and the actual rating for that object, $r_{ui}$. The difference is summed over all combinations of item and all user.

\[
L = \sum_{(u, i) \in \mathcal{O} \cup \mathcal{O}^-} w_{ui}(r_{ui} - \hat{r}_{ui})^2
\]

Figure 2.13: Neural Collaborative Filtering Loss Function

Neural collaborative filtering (NCF) [He et al., 2017] uses the binary cross-entropy loss function for implicit feedback and the weighted square loss for explicit feedback. In NCF, to reduce the number of unobserved training negative instances, techniques like negative sampling [Goldberg and Levy, 2014] are used. Further extension of the work using pairwise ranking loss are proposed by the authors in [Niu et al., 2018], [Song et al., 2018] to improve the results. The NCF model is extended to cross-domain recommendations like [Lian et al., 2017], [Wang et al., 2016]. DeepFM [Guo et al., 2017] is an end-to-end model, able to integrate factorization machine and MLP to find lower and higher-order feature representation. DeepFM is similar to the wide and deep model, a two network deep learning architecture gives recommendations [Cheng et al., 2016]; however, it does not require sophisticated feature engineering. The authors in [Lian et al., 2018] proposed eXtreme deep factorization machine, which models implicit and explicit features together to improve the performance over DeepFM. Covington et al. proposed an MLP based YouTube recommendation model [Covington et al., 2016], and the predictor generates a top-n list of videos based on the nearest neighbor scores from the several hundred videos. The authors in [Alashkar et al., 2017] explored and applied MLP in makeup recommen-
This work uses two identical MLPs to model labeled examples and expert rules, respectively, which provides highly precise recommendations.

2.4.2 Autoencoder Based Recommendation Systems

There are two general techniques for using autoencoders as recommendation systems. First, the autoencoder can be used to learn a lower-dimensional feature representation at the bottleneck layer, which means we employ autoencoder as a dimensionality reduction tool, which is used sequentially with other deep learning techniques for recommendations. On the other hand, we can use autoencoders to fill in the blank values of the rating matrix directly in the reconstruction or decoder layer. Nearly all of the autoencoder alternatives, such as denoising autoencoder, variational autoencoder, contactive autoencoder, and marginalized autoencoder, can be employed to the recommendation task.

AutoRec System

The AutoRec [Sedhain et al., 2015] system is a specific implementation of the technique to use the autoencoder as a generative tool for recommendations. The inputs to the tool are either the partial user vector $r^{(u)}$ or the partial item vector $r^{(i)}$. The goal/output of the system is to reconstruct the input vector entirely in the output layer. The I-AutoRec system is the system where input is based on the partial item vector. Conversely, the U-AutoRec system is the system where input is based on the partial user vector. Generally, the I-AutoRec system performs better than the U-AutoRec system. This is possibly due to the high variance in user data. For the given input $r^{(i)}$ in I-AutoRec, the reconstruction layer is in the form of $h(r^{(i)}; \theta) = f(W \cdot g(V \cdot r^{(i)} + \mu) + b)$, where parameter $\theta = \{W, V, \mu, b\}$, $f(\cdot)$ and $g(\cdot)$ represents the activation functions. The objective function of I-AutoRec is expressed as follows:

$$\arg\min_\theta \sum_{i=1}^{N} \| r^{(i)} - h(r^{(i)}; \theta) \|_\omega^2$$
where $\| \omega \|$ is the observed ratings. L-BFGS (Limited-memory Broyden Fletcher Goldfarb Shanno algorithm) [Moritz et al., 2016] is used to optimize the objective function. The performance of I-Autorec is considerably impacted by the various combinations of the activation functions. Also, increasing the hidden unit size enough will enhance the result as increasing the hidden layer dimensionality gives AutoRec added capability to model the properties of the input data.

**Collaborative Filtering Neural Network (CFN)**

The Collaborative Filtering Neural Network [Strub and Mary, 2015], [Strub et al., 2016] is an extension of the AutoRec System. It uses denoising techniques to make the recommendation system more robust. It additionally uses side information (user profile information / item descriptions) to reduce data sparsity issues and the cold start problem, which in return, increases the training speed and robustness, and improves the prediction accuracy. CFN has two alternatives I-CFN and U-CFN. I-CFN takes $r^{(i)}$ partially observed item vectors as input, and U-CFN takes $r^{(u)}$ partially observed user vectors as input. The authors presented three distinct corruption methods to corrupt the input: Gaussian noise, salt-and-pepper noise, and masking noise. Masking noise is considered as the strong regularizer to deal with the missing values/elements. CFN is further extended by incorporating side information. However, instead of just adding side information in the first layer, CFN injects side information in all layers. Therefore the reconstruction layer changes to:

$$h(\{\hat{r}^{(i)}, s_i\}) = f(W_2, (g(W_1, \{r^{(i)}, s_i\} + \mu), s_i) + b)$$

where $s_i$ is the side information and $\{\hat{r}^{(i)}, s_i\}$ represents the concatenation of $\hat{r}^{(i)}$ and $s_i$. 
Collaborative Denoising Autoencoders (CDAE)

Unlike the other autoencoders, which are designed to output rating predictions, CDAE models [Wu et al., 2016b] are used to output ranking predictions; for example, “What are the top 10 best movies for User A”. Because CDAE provides multiple predictions at once, and are prone to overfitting [Zhang et al., 2016]. The inputs to this system are the user’s partially observed implicit feedback; for example, 1 if a user likes a movie, and 0 if a user dislikes a movie. Because of that, the input can be thought of as a preferences vector that reflects users’ interest in particular items. Since autoencoders are already often prone to overfitting data, there is a need to make the model more robust and learn a better representation of the inputs so a denoising autoencoder [Liang and Liu, 2015] is used in this technique. A denoising autoencoder manually adds noise to regularize the results. Unlike CFN, the results of CDAE are manually corrupted by Gaussian noise as a regularization technique. The authors proposed a negative sampling technique, which reduces the training time without decreasing the ranking quality.

The authors in [Liang et al., 2018] proposed a modification of the variational autoencoder called Multi-VAE and Multi-DAE for recommendation tasks using implicit data. The proposed alternative showed better performance than CDAE. For parameter estimation the authors introduced a principled Bayesian inference approach rather than commonly used likelihood functions, and showed favorable results. Collaborative Deep Learning (CDL) [Wang et al., 2015] is a hierarchical Bayesian model which integrates staked denoising autoencoder (SDAE) into probabilistic matrix factorization. The author proposed a general Bayesian deep learning framework and combined deep learning and recommendation model to balance the influence of side information and interaction history. HRCD [Wei et al., 2017], [Wen et al., 2018] is a hybrid autoencoder based collaborative model and timeSVD++ [Koren, 2009]. HRCD is a time-dependent model, which uses SDAE to learn hidden item representations from raw data and tries at solving the cold start problem.


2.4.3 Convolutional Neural Network based Recommendation Systems

A popular technique for recommendations using convolutional neural networks is the Deep Cooperative Neural Network (Deep CoNN) method [Zheng et al., 2017]. The Deep CoNN method uses two CNNs that run in parallel and are used to model user behaviors and item properties from review texts. The final layer of the network operates as the matrix factorization machine [Koren et al., 2009]. This is used to capture interactions between user behaviors and item properties to generate rating predictions. Using Deep CoNN systems reduces issues associated with data sparsity [Sarwar et al., 2001]. This occurs because the inputs to the system are review texts not just integer ratings. Review texts have rich semantic representations compared to a simple integer rating. By using the rich semantic representation of review texts, the model’s interpretability [Doshi-Velez and Kim, 2017] is enhanced, allowing for more robustness against sparse datasets. Once again, embedding techniques are used to map the semantic representations of the review texts to a lower-dimensional space.

The authors in [Rawat and Kankanhalli, 2016] proposed a context-aware tag recommender system called ConTagNet, which uses CNN to extract image features and uses a fully connected feed-forward neural network to learn the context representation and concatenate these two networks at the final layer with softmax to predict the probability of tags. ConvMF proposed by [Kim et al., 2016] comparable to CDL [Wang et al., 2015], which uses autoencoder to learn the item features, whereas ConvMF uses CNN to learn the high-level item features. ConvMF combines PMF [Mnih and Salakhutdinov, 2008] and CNN seamlessly, which gives the advantage to obtain more precise item feature representation. The authors in [Tuan and Phuong, 2017] proposed using CNN to improve the accuracy of the session-based recommendation by learning feature representation from the item content descriptions. Ying et al. suggested using graph CNNs [Ying et al., 2018], which is known for its scalable properties for the recommendation task. The proposed
model deployed in Pinterest to solve real-world recommendations which create item embeddings by employing both graph structure and feature information with graph CNNs

2.4.4 Recurrent Neural Networks based Recommendation Systems

Time series dependent data can be beneficial for generating accurate predictions at the right time especially true in the case of e-commerce-based recommendation systems in which the sequence of purchases is important to what should be recommended next. Therefore, using session or browser cookie information allows recommendations systems to get a more accurate model of a user’s short-term preferences. However, until recently, RNN based recommendation system was a nascent research field since the training data are extremely sparsely distributed.

Recent research using recurrent neural networks has shown the ability to overcome the sparsity of the training data. Hidasi et al. proposed the use of session-based RNN recommendation systems [Hidasi et al., 2015] based on the use of Gated Recurrent Units (GRUs). As mentioned earlier, GRUs are used to solve the vanishing gradient problem, which makes it difficult to model long-term dependencies in the input time series. GRUs are less computationally expensive than LSTM cells due to the reduced number of internal gates.
As shown in the Figure 2.14, the input layer of the session-based RNN recommendation system takes the current state of the session which, is achieved using a 1 of K coding (the sum of each row should be 1 and each entry should be 0 or 1 in a matrix). A coordinate will consist of 1 if the item is active in the current session, and a coordinate will consist of a 0 if the item is not active in the current session. For example, in Amazon the products that are added to the cart is considered as session, a session vector may consist of items like: “User has a bike in their shop cart?” or “User has a mobile in their shop cart”. The next layer is the embedding layer. The embedding layer tries to harness rich semantic representations from the input session vector to create a strong embedding vector. This also maps the semantic representations to a lower-dimensional space. The next two layers consist of GRU and feedforward layers like most other RNNs. The final output layer outputs the likelihood/prediction of something being in the next session. For example, going back to the Amazon example, if there was a bike in the cart, what is the probability that a bike helmet will be in the cart in the next time step of the session? If the probability is high, the system should recommend a bike helmet to the user.

The follow-up work of [Tan et al., 2016] suggested tactics to improve the model by Hidasi et al. by incorporating the click sequences with dropout regularization and sequence preprocessing. The authors in [Wu et al., 2016a] designed a session-based real-world recommendation model using RNN to predict what the user will buy based on the click history. To reduce the computation cost, the authors merged the older states into a single history state and only used a finite number of latest states. The before mentioned session-based models do not include side information. The follow-up work by [Hidasi et al., 2016] and [Smirnova and Vasile, 2017] include side information and demonstrated that the side information enhances the results of the prediction. Although the success of RNNs in session-based recommendations is high, the authors in [Jannach and Ludewig, 2017] proposed a simple neighbor based model that could achieve almost the same accuracy as
GRU4Rec [Hidasi et al., 2015]. [Donkers et al., 2017] designed a novel type of GRU to explicitly represent individual users for the next item recommendation. The authors in [Okura et al., 2017] designed a GRU with latent factor models to learn user browsing history to recommend news articles.

2.4.5 Restrictive Boltzmann Machines based Recommendation Systems

As discussed earlier, a significant issue with the rating matrix is tremendous data sparsity. Each user has only rated a minimal subset of all the movies in the catalog. Therefore, the question that arises on how to use a Restrictive Boltzmann machine (RBM) to complete the task of matrix factorization.

Professor Hinton and his team at the University of Toronto wanted to treat every single user as a separate training case. Therefore, each user corresponds to a column vector of movie ratings. The authors created a Restrictive Boltzmann Machine [Sutskever et al., 2009] with 100 hidden binary units, the RBM would then try to learn a model and try to fill in the missing values in the matrix using that same model. However, the issue that arose is that it would be intractable to have RBMs with thousands of units with only a few of them being filled in due to the issue with data sparsity. The authors in [Salakhutdinov et al., 2007] came up with a solution, for each user, use a RBM that only has visible units for the movies that the specific user has rated. Therefore, every user might correspond to a different RBM. However, all RBMs in this network still share the same hidden units (and weights). The weight shared between hidden units allows the lack of training cases not to be an issue. Because of shared hidden weights, the average of RBM recommendations provide outstanding results compared with traditional matrix factorization. [Salakhutdinov et al., 2007] is a user-based RBM collaborative filtering model, similarly it is easy to design item-based RBM collaborative filtering. The authors in [Georgiev and Nakov, 2013] proposed a method to combine item-based and
user-based RBM collaborative filtering in a unified framework. In the case, the visible units are determined both by user and item hidden units.

Section 2.4 outlined various deep learning-based recommendation systems using MLP, autoencoder, CNN, RNN, and RBM. RNN, CNN, and RBM based recommendation systems are more suitable for sequential decision making. In this thesis, we focused on using MLP and autoencoder based recommendation system, because these techniques are more suitable for the Company’s data and add the non-linear transformation to the existing recommendation approaches and interpret them into neural extensions.

In this chapter, we discussed the operation of recommendation systems and its different types. We also discussed why we need to consider a deep learning-based recommendation system. We reviewed the fundamental techniques of deep learning and various deep learning-based recommendation systems using MLP, autoencoder, CNN, RNN, and RBM.
Chapter 3

Collaborative Filtering

Chapter 2 describes the various recommendation systems and deep learning techniques. In this chapter, collaborative filtering, types of feedback, and different methodologies selected to solve the thesis problem will be reviewed.

Section 3.1 outlines the collaborative filtering technique; Section 3.2 describes the different types of feedback; Section 3.3 details latent factor models; Sections 3.4, 3.5, 3.6, 3.7 illustrate the chosen Alternating Least Square (ALS), Bayesian Personalized Ranking (BPR), Neural Collaborative Filtering (NCF) and Autoencoder Collaborative Filtering (ACF) algorithms to build the recommendation system. Section 3.8 explains why we have chosen the algorithms mentioned above.

3.1 Collaborative Filtering (CF)

As mentioned in Section 2.1.1, there are three types of recommendation systems; of these, Collaborative Filtering is the most suitable methodology to approach the thesis problem because it doesn’t need anything else except users’ historical preference on a set of items. Recall that the content-based recommendation approach requires a large amount of information of items’ features rather than the users’ interactions and feedback. For example, the features can be the attributes of movies such as the genre, year, director,
actor, etc., or a description of articles. Collaborative Filtering, on the other hand, is an alternative technique that does not need anything more than users’ historical preferences on a set of items. Collaborative Filtering relies on the intuitive idea that similar users tend to like different items similarly. Collaborative filtering methods use an a-priori available set of user-item ratings to learn the interdependencies among users and items, and then predict a user’s rating of an item either via the neighboring items’ ratings (neighbor-based [Das et al., 2007], [Sarwar et al., 2001]) or by inferring latent factors that find similar users and items in a low-dimensional embedding (latent factor-based [Hu et al., 2008], [He et al., 2017], [Rendle et al., 2009], [Moussawi, 2018]). In this thesis, we build on the latent-factor-based approach (more details are given in the following sections).

### 3.2 Types of Feedback

The feedback can be explicit, i.e., the user either likes an item or dislikes an item. For example, the user either clicks “thumbs up” or “thumbs down” on a youtube video or rates a movie on a scale of one to five stars, where more stars denoting a higher rating. Usually, explicit feedback data are either binary, represented as +1, and -1 (or 0), for positive and negative feedback, respectively, or on a multi-level relevance scale, e.g., the 1, 2, 3, 4, 5 rating scale.

Alternatively, the feedback can be implicit, i.e., the feedback derived from the browsing behavior of the user e.g., the user clicked on/purchased a product, checked into a venue, or viewed an article. Implicit feedback data are represented as positive values; usually, either representing that an interaction has happened or positive integers denoting the number of interactions (e.g., the number of times the user has purchased the product). Importantly, implicit feedback data do not contain negative observations; therefore, strategies like negative sampling from unobserved user-item interactions are proposed [Pan et al., 2008].
3.3 Latent Factor Model

Latent-factor or matrix factorization (MF) methods are popular in recommendation systems [Koren et al., 2009] for both for implicit [Rendle et al., 2009], [Hu et al., 2008] and explicit feedback [Mnih and Salakhutdinov, 2008]. The latent factor model tries to predict the users’ rating on an item by optimizing an objective function and by reconstructing the rating matrix into low-rank dimensional latent factors. In simple terms, MF model finds two rectangular matrices (factors) of sizes $M \times k$ and $k \times N$ to represent the big rating matrix of size $M \times N$ (we discussed the rating matrix in Section 2.1.3). The $k$ latent factors otherwise called features, can be found using Singular Value Decomposition (SVD).

Let $p \in \mathbb{R}^{k \times M}$ be the latent factor matrix for the users, where the $u^{th}$ column $p_u \in \mathbb{R}^k$ is the latent factor for user $u$. Similarly, let $q \in \mathbb{R}^{k \times N}$ be the latent factor for the items, where the $i^{th}$ column $q_i \in \mathbb{R}^k$ is the latent factor for item $i$. Effectively, MF is a dimensionality reduction approach, projecting both users and items in the same low-dimensional space, whereas nearby items are similar and users who are near to each other in the embedding are similar. Items that are closer to a user in the embedding are considered to be a good fit for the user.

3.4 Alternating Least Square (ALS)

Matrix Factorization is a technique commonly used in recommendation tasks. A matrix factorization algorithm [Koren et al., 2009] tries to find latent factors that represent the intrinsic user and item attributes in a lower dimension. That is,

\[ \hat{r}_{u,i} = q_i^T p_u \] (3.1)
where \( \hat{r}_{u,i} \) is the predicted ratings for user \( u \) and item \( i \), and \( q_i^T \) and \( p_u \) are latent factors for the item and user, respectively. The challenge of the matrix factorization problem is to find \( q_i^T \) and \( p_u \), which can be achieved by matrix decomposition. A learning approach is therefore developed to converge the decomposition results as close to the observed ratings as possible. Furthermore, to avoid overfitting issues, the learning process is regularized. For example, a basic form of such a matrix factorization algorithm is represented below:

\[
min \sum (r_{(u,i)} - q_i^T p_u)^2 + \lambda((\|q_i\|)^2 + (\|p_u\|)^2)
\]  

(3.2)

where \( \lambda \) is a regularization parameter.

In case explicit ratings are not available, implicit ratings are usually derived from users’ historical interactions with the items (e.g., clicks, purchases, views, etc.). To account for such implicit ratings, the original matrix factorization can be formulated as

\[
min \sum c_{(u,i)} (p_{(u,i)} - q_i^T p_u)^2 + \lambda((\|q_i\|)^2 + (\|p_u\|)^2)
\]  

(3.3)

where \( c_{(u,i)} = 1 + \alpha r_{(u,i)} \) and \( p_{(u,i)} = 1 \) if \( r_{(u,i)} > 0 \) and \( p_{(u,i)} = 0 \) if \( r_{(u,i)} = 0 \). \( r_{(u,i)} \) is a numerical representation of users’ preferences (e.g., number of purchases, number of clicks etc.).

Owing to the term of \( q_i^T p_u \), the loss function is non-convex. The gradient descent method can be applied, but this will incur expensive computations. An Alternating Least Square (ALS) algorithm was therefore developed to overcome this issue [Hastie et al., 2015]. The basic idea of ALS is to learn one of \( q \) or \( p \) at the time of optimization while keeping the other as constant. ALS makes the objective at each iteration convex and solvable. The alternating between \( q \) and \( p \) stops when the convergence is optimal. It is worth noting that the ALS iterative computation can be parallelized and/or distributed, which makes the algorithm desirable [Yu et al., 2014] for use cases where the dataset is large and thus the user-item rating matrix is highly sparse (as typical in recommendation...
system scenarios).

### 3.5 Bayesian Personalized Ranking (BPR)

Bayesian Personalized Ranking (BPR) [Rendle et al., 2009] has emerged as one of the best Top-K recommendation models for implicit data. BPR is also a strong baseline, which makes it difficult to beat. BPR falls under the category of one-class collaborative filtering (for example 0 or 1) and pairwise comparison. It considers the recommendation task as a ranking problem and assumes that the user prefers items that they have already observed/interacted with, rather than unobserved/not interacted with items. Implicit data have positive feedback, indicates the user’s preferences. For example, 1 indicates that the user interacted with the item, and 0 indicates that there is no interaction between the user and the item. As mentioned before, BPR learns to rank the items for each user based on the assumption that the user prefers items observed/interacted with, which means we do not know precisely if the unobserved items are liked by the users or not. In other words, we do not know if the user is interested buying the unobserved items because the user have not yet seen the unobserved items.

To learn the relative ranking of items for each user, BPR needs to model negative feedback. The learning-to-rank problem is approximated by pairwise approach. In pairwise approach, the loss function is defined on the basis of pair of objects whose labels are different. BPR uses the pairwise interpretation of positive-only feedback by creating triplets (user, observed item, unobserved item). The positive-only feedback is then transformed into positive and negative feedback in pairs of item \((i, j)\). The triplets are sampled [Rendle et al., 2009] from

\[
D_u := (u, i, j) | i \in I_u^+ \land j \in I \setminus I_u^+ \tag{3.4}
\]

where \(I\) represent all the items and \(I_u^+\) is defined as the items where user \(u\) gives positive
feedback, \( i \) is positive item taken from \( I^+_u \) and \( j \) is a negative item randomly sampled from unobserved/not interacted with items. \( D_s \) represents all the triplets \((u, i, j)\) such that a user \( u \) prefers item \( i \) over item \( j \).

### BPR Optimization Criterion

The author derives an optimization criterion called BPR-OPT [Rendle et al., 2009], which is an optimization framework. To provide a personalized ranking of items or recommendations, we need to train a separate model using BPR-OPT. Let \( \Theta \) be defined as the parameter of a model where BPR-OPT optimizes the posterior probability \( p(\Theta | >_u) \), where \( >_u \) represents the user’s \( u \) latent structure, \( \sigma \) represent the logistic sigmoid function and \( \lambda_\Theta \) represent the model specific regularization parameters. Here, \( \hat{x}_{uij} \) is an arbitrary real-valued function of the model which captures the relationship between the user \( u \), item \( i \) and item \( j \).

\[
BPR - OPT : = \ln p(\Theta | >_u) \\
= \ln p(>_u | \Theta) p(\Theta) \\
= \ln \prod_{(u,i,j) \in D_s} \sigma(\hat{x}_{uij}) p(\Theta) \\
= \sum_{(u,i,j) \in D_s} \ln \sigma(\hat{x}_{uij}) + \ln p(\Theta) \\
= \sum_{(u,i,j) \in D_s} \ln \sigma(\hat{x}_{uij}) - \lambda_\Theta \|\Theta\|^2
\]

The authors used stochastic gradient descent (SGD) to optimize BPR-OPT. LearnBPR [Rendle et al., 2009], is an overall learning procedure:

\[
\Theta \leftarrow \Theta + \alpha \left( \frac{e^{-\hat{x}_{uij}}}{1 + e^{-\hat{x}_{uij}}} \cdot \frac{\partial x_{uij}}{\partial x} + \lambda_\Theta \cdot \Theta \right)
\]

Note that in the above equation, the input to the function is \( \hat{x}_{uij} \), which is a real valued function, represents the relationship between user \( u \), item \( i \) and \( j \). Using existing tech-
niques like Matrix Factorization (MF) or k Nearest Neighbor (kNN) methods, $\hat{x}_{uij}$ can be decomposed into:

$$\hat{x}_{uij} := \hat{x}_{ui} - \hat{x}_{uj} \quad (3.6)$$

Compared to standard MF or kNN methods, BPR ensures not only the rating predictions but also optimizes the item rankings.

### 3.6 Neural Collaborative Filtering (NCF)

Neural Collaborative Filtering (NCF) [He et al., 2017] is a new neural matrix factorization model that combines both Generalized Matrix Factorization (GMF) and Multi-Layer Perceptron (MLP) to combine the strengths of the linearity and non-linearity given by MF and MLP, respectively, for modeling user-item latent features. NCF can be demonstrated as a framework for GMF and MLP, which is illustrated in Figure 3.1:

![Figure 3.1: Neural Collaborative Filtering Algorithm [He et al., 2017]](image-url)
This Figure 3.1 shows the architecture of NCF and how to utilize latent vectors of items and users. The User (u) and Item (i) are used to create low-dimensional embeddings for user and item. Generalized Matrix Factorization (GMF) combines the two embeddings using the dot product. Multi-layer perceptron (MLP) also produces embeddings for user and items. However, instead of taking a dot product to obtain the rating, the embeddings are concatenated to create a feature vector that can be passed on to the further layers. The outputs from the final layers of the MLP and GMF are concatenated, called a NeuMF layer to obtain the prediction score.

3.6.1 Generalized Matrix Factorization (GMF) Model

In ALS, as mentioned in Section 3.4, the ratings are modeled as:

$$\hat{r}_{u,i} = q_i^T p_u$$

(3.7)

whereas, GMF introduces the neural layer as the output layer of standard MF, thus MF can be generalized and extended. For example, if the weights of the output layer are learned from data, the model results in a variant of MF that learns the latent dimensions [He et al., 2017]. Instead of using a linear activation function, GMF uses a non-linear activation function, which results in learning the non-linear relationships between the user and item and allows the model to generalize. The generalized non-linear MF is more powerful than linear MF model. GMF can be formulated as follows:

$$\hat{r}_{u,i} = a_{out}(h^T(q_i \odot p_u))$$

(3.8)

where $\odot$ is an element-wise product of vectors. Additionally, $a_{out}$ and $h$ represent the activation function and weights of the output layer, respectively. The author argues that the MF can be interpreted as a special case of GMF [He et al., 2017] by using an identity function for $a_{out}$ and forcing it ($h$) to be a uniform vector of 1, therefore exactly recovering
the MF model.

### 3.6.2 Multi-Layer Perceptron (MLP) model

To model users and items, NCF uses 1) element-wise product of user and item vectors, and 2) concatenation of user and item vectors. MLP adopts concatenation to model users and items. The standard MLP model is used to learn the interactions of users and items latent features, by which we learn a model with high flexibility and also we learn the non-linear interactions between \( p_u \) and \( q_i \). The MLP model under NCF framework is defined as:

In the input layer, we concatenate the user latent vector \( p_u \) and item latent vector \( q_i \) as follows:

\[
z_1 = \phi_1(p_u, q_i) = \begin{bmatrix} p_u \\ q_i \end{bmatrix}
\]  

(3.9)

where \( z_1 \) represent the concatenation of \( p_u \) and \( q_i \) at the input layer. The hidden layers are formulated as:

\[
\phi_l(z_l) = a_{out}(W_l^T z_l + b_l), (l = 2, 3, ..., L - 1)
\]  

(3.10)

where \( W_l \), \( b_l \), and \( a_{out} \) denote the weight matrix, bias vector, and activation function for the \( l \)-th layer’s perceptron, respectively and the output layer is formulated as:

\[
\hat{r}_{u,i} = \sigma(h^T \phi(z_{L-1}))
\]  

(3.11)

where \( \hat{r}_{u,i} \) is the predicted rating, \( h \) denote the edge weights and the activation function (sigmoid) of the output layer is defined as \( \sigma(x) = \frac{1}{1+e^{-x}} \) to restrict the predicted score to be in \((0,1)\).
3.6.3 Fusion of GMF and MLP

To have more flexibility in the fused model, we use GMF and MLP to learn the embeddings separately and then combine these two models by concatenating their last hidden layer [He et al., 2017]. We get $\phi^{GMF}$ from GMF and obtain $\phi^{MLP}$ from MLP:

\[
\phi_{u,i}^{GMF} = p_u^{GMF} \odot q_i^{GMF}
\]  

\[
\phi_{u,i}^{MLP} = a_{out} \left( W_L^T \left( a_{out} \left( \ldots a_{out} \left( W_2^T \begin{bmatrix} p_u^{MLP} \\ q_i^{MLP} \end{bmatrix} + b_2 \ldots \right) \right) + b_L \right)
\]  

Lastly, we fuse the output from GMF and MLP:

\[
\hat{r}_{u,i} = \sigma \left( h^T \begin{bmatrix} \phi^{GMF} \\ \phi^{MLP} \end{bmatrix} \right)
\]  

where $\hat{r}_{u,i}$, $\sigma$, $h$, $\phi^{GMF}$ and $\phi^{MLP}$ denote the predicted ratings, sigmoid activation function, edge weights of the output layer, last hidden layer of GMF and last hidden layer of MLP, respectively. This fusion model combines the linearity and non-linearity of MF and Deep Neural Networks (DNN), respectively, for modelling user–item latent structures. We define the likelihood functions as:

\[
P(R, R^- | P, Q, \Theta) = \prod_{(u,i) \in R} \hat{r}_{u,i} \prod_{(u,j) \in R^-} (1 - \hat{r}_{u,j})
\]  

where $R$ denotes the set of observed interactions, and $R^-$ denotes the set of negative instances (unobserved interactions). $P$ and $Q$ denotes the latent factor matrix for users and items, respectively, and $\Theta$ denotes the model parameters. Taking a negative logarithmic of the likelihood, we obtain the objective function to minimize for the NCF.
method:
\[
L = - \sum_{(u,i) \in \mathbb{R} \cup \mathbb{R}^-} r_{u,i} \log \hat{r}_{u,i} + (1 - r_{u,i}) \log (1 - \hat{r}_{u,i}) \tag{3.16}
\]

The optimization can be done by performing Stochastic Gradient Descent (SGD) or Adam.

### 3.7 Autoencoder for Collaborative Filtering (ACF)

Recently, there has been a significant focus in research on using autoencoders for recommendation systems [Sedhain et al., 2015], [Strub and Mary, 2015]. Many different techniques have been proposed, such as denoising architecture [Wu et al., 2016b], dropout to increase efficiency, etc. Here, we use a particular type of autoencoder [Moussawi, 2018] whose model is described in this section. Consider the user-item interaction matrix is represented as \( X \in \{0, 1\}^{U \times I} \) where \( U \) and \( I \) are the set of users and items, respectively. If there is an interaction between user \( u \) and item \( i \) then \( X_{u,i} = 1 \); otherwise, \( X_{u,i} = 0 \).

Given user \( u \) and item \( i \), \( I_u \) represents a set of items, \( u \) has interacted with, and \( U_i \) represents a set of users who have interacted with \( i \).

The autoencoder learns a model \( p(x_u | z_u, \theta) = h(g_\theta(z_u)) \), where \( x_u \) is the user \( u \) vector of interactions, \( z_u \) represents the user latent factor, \( g_\theta \) is an autoencoder parameterized by \( \theta \) and \( h \) is an activation function that maps the output of \( g_\theta \) to probabilities based on the logistic likelihood distribution used to model \( p(x_u | z_u, \theta) \). \( z_u \) can be computed as a function of \( f_\lambda(x_u) \), where \( f_\lambda \) is an autoencoder parameterized by \( \lambda \). The negative log-likelihood loss function of our model to be minimized is then:

\[
- \sum_i \log p(x_u | z_u, \theta)_i = -x_u \cdot \log (g_\theta(z_u)) - (1 - x_u) \cdot \log (1 - g_\theta(z_u)) \tag{3.17}
\]

For regularization, dropouts are applied at the input layers, and also applied L2 weight decay on \( \theta \) and \( \lambda \).
3.8 Modelling Purchasing Behaviors

The following CF latent variable methods were selected to analyze large customer purchasing behavior patterns with implicit feedback between customers and products. The modelling approaches shown in Table 3.1 were selected to examine different aspects of providing product recommendations.

Table 3.1: CF methods selected for analysis

| Analysis                              | Models                                                                 |
|---------------------------------------|------------------------------------------------------------------------|
| Product recommendations based on purchasing behaviors | Matrix Factorization with Alternating Least Square (ALS)               |
|                                       | Bayesian Personalized Ranking (BPR)                                     |
|                                       | Neural Collaborative Filtering (NCF)                                   |
|                                       | Autoencoder for Collaborative Filtering (ACF)                          |

CF with matrix factorization is a good baseline to test the predictions since it constrains the predictions from the user-item factor vectors to linear projections in the same vector space. It is expected that this linear projection between the user-item will discover useful hidden factors.

In BPR, the incorporation of items not purchased by each user may provide additional information on customers’ preferences for items during the minimization of the loss function. The incorporation of items not purchased by user leads to better predictions of customer behavior than the predictions from matrix factorization models that focus on pointwise transactions for model training.

NCF is the most general approach where user-item transactions are concatenated into an input vector that is fed into a neural network. The investigation of NCF may provide insight into whether user-item transactions have a non-linear relationship that can be learned by neural networks.

In autoencoder CF, the input layer has the user-item vector and is expected to reconstruct the user-item vector in the output layer, through which the ACF will discover useful hidden factors.
In this chapter, we reviewed collaborative filtering, types of feedback (implicit and explicit) and a chosen algorithms such as MF-ALS, BPR, NCF and ACF, which we will implement in this thesis to build a product recommendation system.
Chapter 4

Data and Model Specifications

This chapter provides details on the algorithms selected for the analysis of the Company’s data. The data cleaning and pre-processing techniques are outlined. The evaluation criteria for those chosen algorithms are described. We also discussed the model specifications and selection of the hyperparameters for the selected algorithms.

Section 4.1 outlines the exploratory data analysis and pre-processing of the Company’s database. Section 4.2 describes the ranking metrics and evaluation criteria. Section 4.3 outlines the model specifications and hyperparameters for all the chosen algorithms.

4.1 Exploratory Data Analysis

The specific details of the dataset cannot be directly summarized in this thesis since the data are proprietary. Still, the general details of the exploratory data analysis are provided in regard to how the dataset was prepared. The dataset provided by the Company consisted of three years of sales transactions from August 2016 to June 2019, for all the customers. The “Invoiced Orders” database contains the customers’ transaction data and the “Current Items” database contains detailed information about the items offered by the Company. In these databases, a significant amount of information are distributed across different customer IDs. The “Invoiced Orders” databases contains more than 100
columns such as geographic location, selling location, regional order number, rebate value, stock quantity, etc. During the time from August 2016 to June 2019, the Company collected more than 20 million lines of transactions, with over 200,000 customers and 500,000 stock keeping units. The focus of the analysis was to recommend products/items based on past transaction history, therefore most of the columns in the database are not used. The dataset is filtered using structured query language (SQL) where the final dataset has the following information: customer ID, product ID, description of the products, and number of times a customer purchases a product (ratings).

In recommendation systems, the splitting of data into training and testing set requires careful considerations. For example, a recommendation system that recommends products to customers based on the customers’ collaborative rating similarities; the same set of customers for both model training and testing should be used to have statistically sound evaluation metrics. Data splitting is one of the most vital tasks in assessing recommendation systems. The splitting strategy significantly affects the product recommendations, and it should always be taken into careful consideration by the practitioners. The dataset is split into a training and a testing set using a customized function called python_stratified_split, where the split is stratified so that the same group of customers and products will appear in both training and testing sets. The customized function has three critical parameters: “filter_by”, “min_rating”, and “ratio”. The “filter_by” parameter takes two inputs, either “user” or “item” depending on which of the two inputs we want to filter the dataset. The “min_rating” parameter takes an integer as an input, where it filters the dataset based on the minimum number of ratings for user or item. The “ratio” is nothing but the train/test split. For example, if “filter_by” parameter is “user” and the “ratio” is 0.7, it means that the first 70% ratings for each user/customer in the data will be put into the train set while the other 30% is in the test set. After several rounds of discussion with the representatives from the Company, according to their business objectives, we choose “min_rating” = 20, “filter_by” = “user” and “ratio”
= 0.8, i.e., the dataset is split into 80% train set, and 20% test set; and is filtered by the customers who purchased a product at least 20 times.

4.2 Evaluating Recommender Performance

To evaluate the efficiency of a product recommendation, we followed the leave-one-out evaluation, which has been widely used in the literature. In leave-one-out evaluation, for each user, we keep their latest interaction with the item as a test set, and we use the remaining as a training set for the model. During evaluation, it is too tedious and time-consuming to rank all the items for each user. To overcome this problem, we adopted a common strategy used by the practitioners where we sample 100 random items that are not interacted with by the user as a test set and then rank those 100 items with respect to that user. Leave-one-out evaluation with negative sampling is implemented in this work.

The efficiency of the ranked list is measured by two common metrics that are widely used in the learning-to-rank problem, i.e., Normalized Discounted Cumulative Gain (NDCG) [Yilmaz et al., 2008] and Mean Average Precision (MAP) [Yue et al., 2007]. Since predictions from the CF models were ranked lists of the products most likely to be purchased by a customer in test time, the evaluation metrics NDCG and MAP are chosen to measure the fraction of predictions in the ranked list matching the relevant predictions in the test time. NDCG gives more weight to relevant predictions at the starting of the ranked list of items and discounts relevant predictions that occur farther from the beginning of the ranked list. MAP averages relevant predictions over each position in the ranked list. NDCG and MAP are useful metrics for comparing the performance of different CF models. We choose NDCG because it is similar to MAP, except it works for non-binary relevance. According to the business objective of the Company, NDCG would be the best metric to rely on, because it concentrates on the correct, most likely
products to be purchased by a customer.

If the rank position \( k \) of an item in the ranked/predicted list matches the ground truth (user interacted with the item or not), this is referred to as a relevant item, \( rl \). \( rl \) acts as an indicator function which assigns 1 for relevance and 0 for non-relevance. The Discounted Cumulative Gain (DCG) is calculated with logarithmic decay of sum of the relevant items discounted by the weights of the item’s position in the ranked list, where

\[
DCG = \sum_{i}^{k} \frac{rl_i}{\log(i + 1)}.
\]  

(4.1)

NDCG is obtained by dividing with IDCG (Ideal Discounted Cumulative Gain), where IDCG is defined as the DCG of the ranked list for all the relevant items \( RL \) sorted according to the ground truth ratings, where

\[
IDCG = \sum_{i}^{\left| RL \right|} \frac{2^{rl_i} - 1}{\log(i + 1)}.
\]  

(4.2)

NDCG is given by

\[
NDCG = \frac{DCG}{IDCG}.
\]  

(4.3)

One-product Hit Ratio is a metric we devised, which is relevant for the business. One-product Hit Ratio intuitively measures whether the test item is present on the ranked list. For example, if the test item present on the ranked list, then the score is 1; otherwise 0.

### 4.3 Model Specifications

The model specifications include the model training procedures and selection of hyper-parameters for the Collaborative Filtering models. These models are prepared using Tensorflow [Abadi et al., 2016] and PyTorch [Paszke et al., 2017] frameworks with the
Python programming language. These models are trained in a server with 32 cores of CPU and 256 GB of RAM. As mentioned earlier in the Exploratory Data Analysis section of this chapter, the dataset was filtered and split into train and test sets, where we used the train set to train the model and the test set to test the recommendations or predictions made by the model. In machine learning, there is also an essential aspect called hyperparameter tuning, where we choose a set of an optimal parameters for the algorithm. There are three standard ways to do hyperparameter tuning: Grid search [Pedregosa et al., 2011], Random search [Pedregosa et al., 2011], and Bayesian Optimization [Snoek et al., 2012]. Grid search works by searching exhaustively through a specified subset of hyperparameters. Random search works by searching through a specified subset of hyperparameters randomly instead of using brute force. The Bayesian Optimization approach has been proven to outperform other state-of-the-art hyperparameter tuning approaches [Snoek et al., 2012]. Bayesian approach uses past choices made to make a smart choice of hyperparameters for the next set of values to evaluate, through which it reduces the cost of searching for parameters. In this thesis, we use Bayesian Optimization for hyperparameter tuning.

**Bayesian Optimization**

Bayesian Optimization [Snoek et al., 2012] works by building a posterior distribution of functions that best matches the target function we want to optimize. As the number of iterations increases, the posterior distributions improve efficiently, by which the optimization algorithm knows where to look in the parameter space. Also, by taking into account the target function, it knows which regions of parameters are worth exploring. This method uses Upper Bound Confidence (UBC) [Garivier and Moulines, 2011] and Expected Improvement (EI) [Vazquez and Bect, 2010] strategies for exploration purposes. Bayesian approach tries to find the optimal combination of parameters by reducing the number of steps needed to obtain those parameters compared with standard methods.
like Grid search and Random search. The details of the Bayesian Optimization approach are given in (Snoek et al. 2012) paper. In this thesis, we used NDCG as a target function and applied Bayesian Optimization to find the optimal combination of parameters that maximized the NDCG.

Matrix Factorization with ALS

In ALS methodology, we closely follow the implementation of the paper by Hu, Koren, and Volinsky [Hu et al., 2008]. As mentioned section 4.1, we split the dataset into a train and test set, and we use the train set to build the model. In the ALS approach, we construct preferences for each item using the ratings given by a user. Here, the ratings represent the number of times the items purchased by a user. We set the preference \( p_{(u,i)} \) by:

\[
p_{(u,i)} = \begin{cases} 
1, & r_{(u,i)} > 0 \\
0, & \text{otherwise}
\end{cases}
\]  

We use the preference \( p \) to calculate the confidence \( c_{(u,i)} \):

\[
c_{(u,i)} = 1 + \alpha r_{(u,i)}
\]  

where \( \alpha \) is a scaling factor, which is also a hyperparameter. The goal of the method is to minimize the objective function (Equation 3.3),

\[
\min \sum (r_{(u,i)} - q_i^T p_u)^2 + \lambda (\|q_i\|^2 + (\|p_u\|^2)
\]

We find that training the model in this fashion takes more time, so we experimented with different optimizers and used Adam optimizer to reduce the training time. To make the training even faster, we converted the python code to Cython [Behnel et al., 2011], where Cython converts the python code to C code to boost the performance [Behnel et al., 2011].
We used Tensorflow’s embedding layer to create user-item latent vectors. We used Bayesian Optimization to find the optimal hyperparameters.

**Table 4.1: Hyperparameters for MF with ALS model**

| Hyperparameter                  | Value |
|---------------------------------|-------|
| user-item Latent Dimension (k)  | 200   |
| Regularization                  | 0.0001|
| Learning Rate                   | 0.01  |
| Iterations (Epochs)             | 30    |
| Batch Size                      | 512   |
| Scaling Factor (α)              | 15    |

The hyperparameters include user-item latent dimensions, regularization, iterations and scaling factors. We find that the user-item latent dimensions have the most significant impact on performance. To create latent dimension vectors, we used embedding layers whose values were uniform and randomly initialized, and then the values were learned during the model training process. L2 regularization was used. The scaling factor was set to 15. During hyperparameter tuning, the number of iterations was found to be optimal at 50. Table 4.1 displays the hyperparameters of the MF with the ALS model. Figure 4.1 depicts the ALS training loss per epoch established using the hyperparameters.
Bayesian Personalized Ranking

In MF with the ALS method described above, we focused on pointwise loss minimization, which captured the positive or negative user-item interactions separately. However, there may be some hidden information available in the negative user-item interactions for positive user-item interactions. The idea was formulated into pairwise loss minimization by [Rendle et al., 2009]. Recall that in BPR, we construct triplets such that the user $u$ prefers item $i$ over item $j$ using the sampling method (Equation 3.4)

$$D_s := (u, i, j) | i \in I_u^+ \land j \in I \backslash I_u^+$$

We train the triplets using the pairwise loss function, as discussed in Section 3.5. The BPR latent factors are constructed using TensorFlow’s embedding layer, and we find that BPR has the same training characteristics as MF with the ALS method. However, BPR takes more iterations or epochs to converge to local minima. This problem arises because of the pairwise loss function, which needs to compute more gradients during the training procedure. The authors [Rendle et al., 2009] identified the gradient problem, and proposed an alternative negative sampling method [Rendle et al., 2009]. Table 4.2 displays the hyperparameters of the BPR model. The hyperparameter for this methodology consists of user-item latent dimensions (factors), the number of epochs, the learning rate for the optimizer, and regularization to prevent overfitting. By using Bayesian Optimization, we find the best hyperparameters for BPR approach. Figure 4.2 depicts the BPR training loss per epoch established using the hyperparameters.
In ALS and BPR methods, we tried to find the linear relationships that exist between the user and the item. However, in NCF, we try to find both linear and non-linear relationships in user-item interactions by utilizing the power of neural networks. As discussed in Section 2.5.6, we create GMF, MLP, and Fusion of GMF & MLP (NeuMF) models. The goal of the NCF model is to train and minimize the binary cross-entropy.
loss function as defined in Equation 3.16

\[
L = - \sum_{(u,i) \in \mathcal{R} \cup \overline{\mathcal{R}}} r_{u,i} \log \hat{r}_{u,i} + (1 - r_{u,i}) \log (1 - \hat{r}_{u,i})
\]

The hyperparameters for the NCF methodology consist of \( n_{\text{factors}} \), \( \text{layer\_size} \), \( n\_\text{epochs} \), \( \text{learning\_rate} \), and \( \text{batch\_size} \), where \( n_{\text{factors}} \) represents the dimensions of the latent space; \( \text{layer\_size} \) represents the sizes of the input and hidden layers of the MLP; \( n\_\text{epochs} \) is the number of iterations to run the training. In general, we find that increasing \( n_{\text{factors}} \) increases the quality of predictions. The user/item labels are mapped to real-valued latent vectors with Tensorflow’s embedding layers.

| Hyperparameter   | Value          |
|------------------|----------------|
| \( n_{\text{factors}} \) | 16             |
| \( \text{layer\_size} \) | \([64,32,16]\) |
| \( n\_\text{epochs} \) | 50             |
| \( \text{learning\_rate} \) | 0.001          |
| \( \text{batch\_size} \) | 256            |

The Table 4.3 describes the optimal hyperparameters for the NCF models using Bayesian Optimization. We found that, in training a NeuMF model, using pre-trained model weights of GMF and MLP is far better than using gaussian normal sampled initialized weights. The dimensions of the latent space are significantly reduced as compared with the previous methodologies (ALS, BPR) to reduce overfitting. Figure 4.3 depicts the NCF training loss per epoch established using the hyperparameters.
Autoencoder for Collaborative Filtering

The ACF approach is entirely different from previous approaches. Here, we use Autoencoder both as a tool for dimensionality reduction as well as a learning algorithm to discover the hidden user-item latent features in the dataset. We use the PyTorch Framework to build the ACF model. To train the model faster, we convert the filtered dataset into a compressed sparse (CSR) format, and we name this train_matrix, which simply contains the user-item interactions in CSR format; train_matrix was saved in .npz format so that we do not need to create the sparse matrix each time. The objective of the ACF model is to minimize the loss function which was defined in equation 3.17,

$$- \sum_i \log p(x_u|z_u, \theta)_i = -x_u \cdot \log(g_\theta(z_u)) - (1 - x_u) \cdot \log(1 - g_\theta(z_u)).$$
Table 4.4: Hyperparameters for ACF model

| Hyperparameter          | Value |
|-------------------------|-------|
| hidden_layer            | 7     |
| noise_prob              | 0.3   |
| dropout_prob            | 0.2   |
| lr (learning rate)      | 0.001 |
| weight_decay            | 2e-5  |
| batch_size              | 256   |
| num_epochs              | 30    |

Typically, Autoencoder has two parts: an encoder and a decoder. PyTorch’s nn.embedding layers are used to build the encoder, whereas the same nn.embedding layers are used to build the decoder, but we reverse the encoder’s architecture. User/item labels are mapped to the latent space using an encoder. As the name suggests, the decoder is used to decode the encoder to get the original user/item labels from the latent space. Instead of randomly assigning weights to the embedding layer, Xavier’s initialization is used. An Adam optimizer, as well as the ReLU activation function, are used throughout the process. The hyperparameter for the ACF model consists of hidden_layer, noise_prob (dropout probability at the input layer), dropout_prob (dropout probability at the bottleneck layer), lr (learning rate), weight_decay, batch_size, and num_epochs. Like ALS, BPR and NCF methodologies, we use Bayesian Optimization to find the best hyperparameters. Table 4.4 shows the hyperparameters for the ACF model. Figure 4.4 depicts the ACF training loss per epoch established using the hyperparameters.
In this chapter, we discussed the cleaning and pre-processing steps of the database. We reviewed the ranking metric NDCG to evaluate the recommendations given by the model. We also discussed the model specifications and hyperparameters for ALS, BPR, NCF, and ACF.
Chapter 5

Results and Discussions

In this chapter, the results, discussions, directions for future work and recommendations are presented.

Section 5.1 describes the results of the models; Section 5.2 outlines the discussions of the results.

5.1 Results

The average performance of CF model predictions for the test set using leave-one-out evaluation are displayed in Table 5.1. The NDCG metric was used to calculate the performance of the model, which was explained in section 4.2. According to the business objective, we choose twelve products to recommend; hence, we used NDCG@12. NDCG@12 gives us the twelve most relevant ranked products based on customer purchasing behavior.
Table 5.1: CF average performance metrics

| Average Predictions | NDCG@12       | One-Product Hit Ratio |
|---------------------|---------------|-----------------------|
| ALS                 | 0.577 ± 0.055 | 1                     |
| BPR                 | 0.636 ± 0.048 | 1                     |
| NCF                 | 0.724 ± 0.049 | 1                     |
| ACF                 | 0.604 ± 0.069 | 1                     |

The results in the Table 5.1 demonstrate that the NCF model achieved the best average performance for NDCG@12 over the leave-one-out evaluation test interval. One-Product Hit Ratio is 1 for ALS, BPR, NCF, and ACF; which means the algorithms at least predict one product that the customer is intended to buy.

We consider ALS as the baseline since it tries to find the most direct relationship between the user and item using matrix factorization technique. From the results in the Table 5.1, the performance of ALS is good over the test period. BPR performed better than ALS, which was expected because it also implemented matrix factorization with the same user/item latent factors to discover the hidden features in the dataset. BPR is varied from ALS in that we implemented pairwise training between positive and negative interactions, which was described in Section 3.5. NCF performed better than all other models, which was expected since it combines the linearity and non-linearity of matrix factorization and neural networks, respectively, which allows NCF to find more relevant hidden features over the dataset. The ACF performed better than ALS but not better than BPR and NCF, which is not always the case. The efficiency of an autoencoder depends on the data; it learns to capture as much information as possible rather than relevant information. This means that if the information most appropriate for our problem is less in magnitude, the autoencoder may lose this information.

The results demonstrate that NCF is the most effective method for recommendations.
for the dataset investigated. After a significant discussion with the Company’s business representatives and looking at the top twelve recommendation results of each model for different sets of customers, we decided to implement NCF and BPR on the Company’s website to test which algorithm works best in real-time.

5.2 Discussions

For the specific dataset investigated in this thesis, CF with NCF produced the best test performance. MF with ALS is considered as a baseline in this thesis since it focused on direct linear relationships between a user and an item. From the results shown in Table 5.1, we can see that all the models performed relatively well compared with the baseline model. NCF has a significant 14.7% increase in NDCG over the baseline; compared with baseline, the BPR results are also high enough, with a 5.9% increase in NDCG. ACF performed relatively weakly compared with BPR and NCF, with a 2.7% increase in NDCG over the baseline.

The BPR matrix factorization model achieved better NDCG performance than the MF with the ALS model (the baseline). This indicates that the pairwise training in BPR was effective in exploring the parameter space as opposed to the pointwise training used in ALS, leading to the improved performance of NDCG, though both BPR and ALS used matrix factorization with the same user/item latent factor vector dimensions.

The ACF did not perform as well as the BPR approach in terms of NDCG metrics. The huge sparsity in the dataset investigated likely limited the usefulness of the autoencoder-based deep learning approaches. As discussed in section 5.1, autoencoder tries to capture more information as opposed to relevant information; since the dataset is sparse in nature, it is likely to capture non-relevant information on the user-item interactions. Although the autoencoder can be used as a dimensionality reduction tool, the sparsity issue does not allow us to use the autoencoder to its full potential. The
NCF model combines matrix factorization and neural networks at the final layer, which allows the model to create a more effective and useful hidden representation of the sparse dataset that allows the model to perform better than other models.

As discussed in the previous section, after discussing with the Company’s representatives, we decided to implement the BPR and NCF model on the Company’s website. Since the factors determined by BPR and NCF models are hidden, it is difficult to understand which factors of the user/item interaction are driving the results. We know that the hyperparameters of the model greatly affect performance. For the BPR model, the major hyperparameters that drive the results are user-item latent dimensional factors (k) and regularization. From Figure 5.1, we can see that for higher k values, the loss is relatively higher in the first few epochs compared with lower k values (Figure 5.2) by keeping other hyperparameters the same. Also, the model takes a significantly higher time to train when the k value is large.

![Figure 5.1: BPR Training Loss Graph (k=500)](image-url)
Another important hyperparameter is regularization; this is a technique used for tuning the loss function by adding an additional penalty term. From Figure 5.3, we can see that the pairwise loss is starting around 0.90 for regularization value 0.001. From Figure 5.4, it can be seen that the pairwise loss starts approximately 0.68 for regularization value 0.0001. Therefore it is apparent that regularization is a driving factor in reducing the loss.

Figure 5.2: BPR Training Loss Graph (k=200)

Figure 5.3: BPR Training Loss Graph (k=200, reg=0.001)
For the NCF model, pre-training GMF and MLP was also an essential hyperparameter in reducing the loss function and achieving better NDCG performance. As described in Section 3.6, the NCF model was constructed by concatenating the last hidden layer of GMF and MLP to learn the hidden features of user-item interactions. The concatenated
layer is called the NeuMF layer. The training loss for the GMF and MLP are shown in Figure 5.5 and 5.6. From Figure 5.5, we can see that the binary cross-entropy loss for GMF starts approximately at 0.50 and drops down to 0.10 and from Figure 5.6, it can be seen that the binary cross-entropy loss for MLP begins around at 0.35 and ends at 0.07.

![NCF-MLP Training Loss vs Epochs](image)

**Figure 5.6:** MLP Training Loss Graph

We trained the NCF model without pre-training weights and observed the training loss. From Figure 5.7, we can see that the binary cross-entropy loss is very high starts approximately 0.35. [He et al., 2017] argued that the pre-trained weights reduced the training loss; we trained the GMF and MLP model separately with an Adam optimizer. The trained GMF and MLP model weights are used to train the NeuMF layer. As a result, the binary cross-entropy training loss reduced drastically and started at 0.064, which is shown in Figure 5.8; whereas the non-pre-trained model started at 0.35. The pre-trained model achieved 0.724 on the NDCG metric, while the non-pre-trained model achieved 0.698, where the performance is reduced by 2.6%, showing that pre-training has a significant impact on the training loss.
For the ACF model, we implemented different loss functions like mean squared loss and binary cross entropy loss to see any improvement in the overall performance of the model. We found that changing the loss function did not improve the NDCG performance; we believe this is likely due to the sparse nature of the dataset.
Choosing the hyperparameters of the model can be done through Bayesian Optimization (BO). BO [Hutter et al., 2011], [Snoek et al., 2012], [Bergstra et al., 2011] is becoming a standard over the years. Hyperparameters can be defined as any free parameter of a learning algorithm that is not optimized directly by the learning procedure. For example, the number of neurons or layers in a neural network, the learning rate for gradient descent, etc. can be optimized with BO. Hyperparameter optimization may result in overfitting also known as oversearching, the knowledge of the validation data split(s) increases as the number of iterations increases on optimizing hyperparameters. k-fold cross-validation is the standard solution to the problem of overfitting in hyperparameter optimization. The authors [Wainer and Cawley, 2017] conducted an experimental study to find the number of validation folds to be used for cross-validation evaluation, and they found that two or three validation folds are adequate for finding optimal hyperparameters. In this thesis, we used NDCG as a target function and applied Bayesian Optimization to find the optimal combination of parameters that maximized the NDCG. We selected the iterations (number of validation folds) to be three, and the hyperparameters include the number of epochs, learning rate, batch size, weight decay, etc.

| Iterations | NDCG@12 | Learning Rate | n_factors |
|------------|---------|---------------|-----------|
| 1          | 0.698   | 0.003         | 12        |
| 2          | 0.703   | 0.09          | 14        |
| 3          | 0.708   | 0.001         | 16        |

Table 5.2: Bayesian Optimization for NCF

Table 5.2 shows the Bayesian Optimization results for the NCF model with two hyperparameters, namely learning rate and n_factors. From Table 5.2 it can be seen that learning rate 0.001 and n_factors 16 gives the maximum NDCG (0.708). This discussion motivates further work on how to best represent sparse datasets to predict product
recommendations efficiently. The CF with NCF predictions demonstrated in this thesis discovers the hidden features of user-item interactions that lead to a good generalization, though a hybrid approach of combining item-based and user-based recommenders for predictions on a more complex representation of user-item interactions may improve the NDCG results presented in this thesis. The extensions to matrix factorization and the usage of click data for future work are discussed in the next section.

In this chapter, we reviewed and discussed the results provided by the models and the chapter 6 will outline the potential recommendations, future work and conclusions for this thesis problem.
Chapter 6

Future Work and Conclusions

In this chapter the directions of future work, recommendations and conclusions are presented.

Section 6.1 and Section 6.2 illustrates the future work and potential recommendations for this thesis problem and Section 6.3 describes the conclusions.

6.1 Future Work

Research into collaborative filtering with matrix factorization has demonstrated algorithmic, logical, and practical insights that explain the NDCG performance observed in this thesis and has provided future directions of research in predicting user-item interactions with implicit feedback. As discussed in section 3.3, the matrix factorization techniques are used to find user/item latent factors whose dimensions are less than the rank of the user/item matrix. Srebo proposed an alternative approach to standard matrix factorization techniques called large-margin matrix factorization [Srebro, 2004], where the dimensions of user/item latent factor matrices are not limited to the rank of the observed user/item matrix. The observed user/item matrix $X$ with $m \times n$ dimensions and the interaction label $y$ has either +1 or -1 subject to hard or soft threshold constraints. The objective function was optimized to minimize the trace norm of $X$, where the trace norm
∥X∥_{tr} is defined as the sum of singular values of X. This alternative approach by Srebo demonstrated better generalization properties.

Lee focused on the structural properties of collaborative filtering-based latent variable models. By designing a similarity metric to measure the distance between the latent variables, Lee showed that the neighborhood or similarity-based CF model is equivalent to latent factor models [Lee, 2017]. Even though there are adequate overlapped entries in the user/item matrix, the algorithm proposed by Lee converged to a wide class of functions with no restrictions on the dimensions of the latent factors. This work was further developed by Borges et al. to extend the similarity metric over the latent factor models in the sparse dataset [Borgs et al., 2017]. The Optimization of matrix factorization with least square methods was further explored by Gunasekar et al., demonstrated that without any definite constraint or regularization, the optimization was regulated to small trace norm, which leads to good generalization results in different experiments [Gunasekar et al., 2017].

The cold start problem is an active area of research that also applies to the work in this thesis. The cold start problem can be relevant to both users and items that have no reviews or history; in simple terms, what to recommend to users/items that are not part of a training dataset. Coldstart problem is temporarily avoided in this thesis by using the same set of users in both training and testing time, but it is a critical component in real-time. Different procedures have been proposed to address the cold start problem. A user/item deep learning content-based recommender was proposed by [Volkovs et al., 2017] gives a probability of user/item interaction in the absence of particular user/item interaction in the training dataset. In the future, a content-based recommender that addresses the cold start problem along with the NCF model implemented in this thesis could be further investigated.

For the use case studied in this thesis, there is another potential future direction that comprises advancements in variational autoencoders (VAEs). Similar to ACF, which is
implemented in this thesis, VAEs also learn hidden non-linear representations of the data using a probabilistic framework [Kingma and Welling, 2013]. VAEs that implemented CF demonstrated state-of-the-art results for movie recommendations on Netflix dataset outperforming techniques like NCF [Liang et al., 2018]. It is not clear if such a proposal would be suitable for the particular use case studied in this thesis since the dataset is sparse; therefore, the VAEs may not be useful, as illustrated in this thesis, for ACF.

Another critical area of future work is to utilize click-data. In this work, we did not have click-data. However, we suggested the Company’s Software Architect to consider implementing click-data so that it can be incorporated in future versions of the Company’s recommender system. Clickthrough Rate (CTR) is a ratio that indicates how often the people who see your recommendation end up clicking it. CTR can also be used to measure how strongly the recommendations are performing. CTR is defined as the number of clicks that recommendations get divided by the number of times the recommendations are shown, i.e., CTR = clicks % impressions. For example, if we had 10 clicks and 100 impressions, then the CTR is 10%. A high CTR is a good evidence that users find the recommendations beneficial and suitable. CTR is also valuable in understanding which recommendations need to be improved.

Guo et al. further explored the CTR prediction and proposed a method called DeepFM [Guo et al., 2017], which combines the strength of factorization machines for recommender systems and deep learning for feature extraction in a novel neural network architecture that can be used as an end-to-end learning algorithm to discover user-item interactions. DeepFM demonstrated more reliable generalization results across different datasets. This line of work further investigated by Zhou et al., who proposed an even better algorithm called Deep Interest Network (DIN) [Zhou et al., 2018]. The authors constructed a local activation unit to discover the hidden representation of the users’ interactions adaptively from past purchasing behaviors concerning individual recommendations. Alibaba’s advertising system currently uses DIN strategy and achieves
state-of-the-art performance compared to other deep learning methods like NCF, which is implemented in this thesis, overcoming the fixed-length latent vector bottleneck. In the future, after accumulating click-data, the DIN model could be implemented with the current use case and investigated further.

6.2 Recommendations

The specific recommendations for enhancing the results shown in this thesis include the following based on the discussion of the results and the discussion of directions for future work:

- Examine possible advances to CF with latent factor models, as shown by Borgs et al., that concentrate on enhancing CF for sparse datasets [Borges et al., 2017], which applies to the large user-item transactions studied in this thesis.

- Further analyze and implement variational autoencoders (VAEs) for the use case studied in this thesis, as described by Liang et al. using multinomial likelihood [Liang et al., 2018], which has been shown to demonstrate competitive performance for various real-world datasets.

- Investigate the development of a content-based model [Volkovs et al., 2017] for the cold start problem as described by Volkovs et al. for predicting the probability of products for customers not present in the CF training datasets by using additional user/item features.

- Investigate further after gathering real-time click-data and implement the Deep Interest Network [Zhou et al., 2018] as demonstrated by Zhou et al. for the use case analyzed in this thesis, using CTR to generate more trustworthy product recommendations.
6.3 Conclusions

In this thesis, the problem of predicting sparse product recommendations at a specific customer level was studied on a proprietary dataset from August 2016 to June 2019. It was hypothesized that CF with latent variable models could discover hidden features in the broad range of products by exploiting the similarity of customers’ past purchasing behaviors, which could be generalized to future purchases. CF with NCF using latent hidden factors for customers and products presented the highest average performance metrics over the period of the dataset studied. The BPR model is shown to be nearly as efficient as CF with NCF. Deep Learning strategies for CF with latent variables like ACF were investigated and revealed to not be as powerful as NCF. A benefit of the NCF approach is that it has discovered that non-linear hidden relationships exist between customers and products, information that was very valuable in the product recommendations. A limitation of NCF is that the learned hidden representation of user/item matrices is not interpretable, which makes it complicated to comprehend why NCF recommends these products to a customer. Recent work illustrated that matrix factorization with latent variables not limited to low-rank approximation shows excellent generalization properties, enduring sparsity and predicting CTR using Deep Interest Network are promising for further application development in predicting product recommendations where data may be sparse and underlying factors may be complicated, noisy and challenging to apprehend explicitly.
Bibliography

[Abadi et al., 2016] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467.

[Aiguzhinov et al., 2010] Aiguzhinov, A., Soares, C., and Serra, A. P. (2010). A similarity-based adaptation of naive bayes for label ranking: Application to the metalearning problem of algorithm recommendation. In International Conference on Discovery Science, pages 16–26. Springer.

[Alashkar et al., 2017] Alashkar, T., Jiang, S., Wang, S., and Fu, Y. (2017). Examples-rules guided deep neural network for makeup recommendation. In Thirty-First AAAI Conference on Artificial Intelligence.

[An and Cho, 2015] An, J. and Cho, S. (2015). Variational autoencoder based anomaly detection using reconstruction probability. Special Lecture on IE, 2(1).

[Bansal et al., 2016] Bansal, T., Belanger, D., and McCallum, A. (2016). Ask the gru: Multi-task learning for deep text recommendations. In Proceedings of the 10th ACM Conference on Recommender Systems, pages 107–114. ACM.

[Behnel et al., 2011] Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., and Smith, K. (2011). Cython: The best of both worlds. Computing in Science & Engineering, 13(2):31.
[Berg et al., 2017] Berg, R. v. d., Kipf, T. N., and Welling, M. (2017). Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*.

[Bergstra et al., 2011] Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554.

[Borgs et al., 2017] Borgs, C., Chayes, J., Lee, C. E., and Shah, D. (2017). Thy friend is my friend: Iterative collaborative filtering for sparse matrix estimation. In *Advances in Neural Information Processing Systems*, pages 4715–4726.

[Bostandjiev et al., 2012] Bostandjiev, S., O’Donovan, J., and Höllerer, T. (2012). Tasteweights: a visual interactive hybrid recommender system. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 35–42. ACM.

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

[Burke, 2007] Burke, R. (2007). Hybrid web recommender systems. In *The adaptive web*, pages 377–408. Springer.

[Cantador et al., 2010] Cantador, I., Bellogín, A., and Vallet, D. (2010). Content-based recommendation in social tagging systems. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 237–240. ACM.

[Chen et al., 2017] Chen, X., Zhang, Y., Ai, Q., Xu, H., Yan, J., and Qin, Z. (2017). Personalized key frame recommendation. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 315–324. ACM.

[Cheng et al., 2016] Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhya, 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*, pages 7–10. ACM.

[Covington et al., 2016] Covington, P., Adams, J., and Sargin, E. (2016). Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198. ACM.

[Das et al., 2007] Das, A. S., Datar, M., Garg, A., and Rajaram, S. (2007). Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280. ACM.

[Donkers et al., 2017] Donkers, T., Loepp, B., and Ziegler, J. (2017). Sequential user-based recurrent neural network recommendations. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 152–160. ACM.

[Doshi-Velez and Kim, 2017] Doshi-Velez, F. and Kim, B. (2017). Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*.

[Ebesu and Fang, 2017] Ebesu, T. and Fang, Y. (2017). Neural citation network for context-aware citation recommendation. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 1093–1096. ACM.

[Ekstrand et al., 2011] Ekstrand, M. D., Riedl, J. T., Konstan, J. A., et al. (2011). Collaborative filtering recommender systems. *Foundations and Trends® in Human–Computer Interaction*, 4(2):81–173.

[Garivier and Moulines, 2011] Garivier, A. and Moulines, E. (2011). On upper-confidence bound policies for switching bandit problems. In *International Conference on Algorithmic Learning Theory*, pages 174–188. Springer.
[Georgiev and Nakov, 2013] Georgiev, K. and Nakov, P. (2013). A non-iid framework for collaborative filtering with restricted boltzmann machines. In International conference on machine learning, pages 1148–1156.

[Goldberg and Levy, 2014] Goldberg, Y. and Levy, O. (2014). word2vec explained: deriving mikolov et al.’s negative-sampling word-embedding method. arXiv preprint arXiv:1402.3722.

[Gomez-Uribe and Hunt, 2016] Gomez-Uribe, C. A. and Hunt, N. (2016). The netflix recommender system: Algorithms, business value, and innovation. ACM Transactions on Management Information Systems (TMIS), 6(4):13.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep learning. MIT press.

[Gunasekar et al., 2017] Gunasekar, S., Woodworth, B. E., Bhojanapalli, S., Neyshabur, B., and Srebro, N. (2017). Implicit regularization in matrix factorization. In Advances in Neural Information Processing Systems, pages 6151–6159.

[Guo et al., 2017] Guo, H., Tang, R., Ye, Y., Li, Z., and He, X. (2017). Deepfm: a factorization-machine based neural network for ctr prediction. arXiv preprint arXiv:1703.04247.

[Hastie et al., 2015] Hastie, T., Mazumder, R., Lee, J. D., and Zadeh, R. (2015). Matrix completion and low-rank svd via fast alternating least squares. The Journal of Machine Learning Research, 16(1):3367–3402.

[He et al., 2017] He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. (2017). Neural collaborative filtering. In Proceedings of the 26th International Conference on World Wide Web, pages 173–182. International World Wide Web Conferences Steering Committee.
[Hidasi et al., 2015] Hidasi, B., Karatzoglou, A., Baltrunas, L., and Tikk, D. (2015). Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*.

[Hidasi et al., 2016] Hidasi, B., Quadrana, M., Karatzoglou, A., and Tikk, D. (2016). Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 241–248. ACM.

[Hinton and Salakhutdinov, 2006] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

[Hochreiter, 1998] Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Hu et al., 2008] Hu, Y., Koren, Y., and Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272. IEEE.

[Huang et al., 2013] Huang, P.-S., He, X., Gao, J., Deng, L., Acero, A., and Heck, L. (2013). Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2333–2338. ACM.

[Hutter et al., 2011] Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer.
[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

[Jannach and Ludewig, 2017] Jannach, D. and Ludewig, M. (2017). When recurrent neural networks meet the neighborhood for session-based recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 306–310. ACM.

[Jozefowicz et al., 2015] Jozefowicz, R., Zaremba, W., and Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pages 2342–2350.

[Kamilaris and Prenafeta-Boldú, 2018] Kamilaris, A. and Prenafeta-Boldú, F. X. (2018). Deep learning in agriculture: A survey. *Computers and electronics in agriculture*, 147:70–90.

[Karpathy, 2019] Karpathy, A. (2019). Convolutional neural network for visual recognition.

[Ker et al., 2017] Ker, J., Wang, L., Rao, J., and Lim, T. (2017). Deep learning applications in medical image analysis. *Ieee Access*, 6:9375–9389.

[Kim et al., 2016] Kim, D., Park, C., Oh, J., Lee, S., and Yu, H. (2016). Convolutional matrix factorization for document context-aware recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 233–240. ACM.

[Kim et al., 2014] Kim, J., Lee, D., and Chung, K.-Y. (2014). Item recommendation based on context-aware model for personalized u-healthcare service. *Multimedia Tools and Applications*, 71(2):855–872.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*. 
[Koren, 2009] Koren, Y. (2009). Collaborative filtering with temporal dynamics. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 447–456. ACM.

[Koren et al., 2009] Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. Computer, (8):30–37.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. nature, 521(7553):436.

[Lee, 2017] Lee, C. C. E. (2017). Latent variable model estimation via collaborative filtering. PhD thesis, Massachusetts Institute of Technology.

[Lian et al., 2017] Lian, J., Zhang, F., Xie, X., and Sun, G. (2017). Cccfnet: a content-boosted collaborative filtering neural network for cross domain recommender systems. In Proceedings of the 26th international conference on World Wide Web companion, pages 817–818. International World Wide Web Conferences Steering Committee.

[Lian et al., 2018] Lian, J., Zhou, X., Zhang, F., Chen, Z., Xie, X., and Sun, G. (2018). Xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 1754–1763. ACM.

[Liang et al., 2018] Liang, D., Krishnan, R. G., Hoffman, M. D., and Jebara, T. (2018). Variational autoencoders for collaborative filtering. In Proceedings of the 2018 World Wide Web Conference, pages 689–698. International World Wide Web Conferences Steering Committee.

[Liang and Liu, 2015] Liang, J. and Liu, R. (2015). Stacked denoising autoencoder and dropout together to prevent overfitting in deep neural network. In 2015 8th International Congress on Image and Signal Processing (CISP), pages 697–701. IEEE.
[Lin et al., 2017] Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017). Focal loss for dense object detection. In Proceedings of the IEEE international conference on computer vision, pages 2980–2988.

[Liu and Wu, 2017] Liu, J. and Wu, C. (2017). Deep learning based recommendation: A survey. In International Conference on Information Science and Applications, pages 451–458. Springer.

[Lops et al., 2011] Lops, P., De Gemmis, M., and Semeraro, G. (2011). Content-based recommender systems: State of the art and trends. In Recommender systems handbook, pages 73–105. Springer.

[Mnih and Salakhutdinov, 2008] Mnih, A. and Salakhutdinov, R. R. (2008). Probabilistic matrix factorization. In Advances in neural information processing systems, pages 1257–1264.

[Moritz et al., 2016] Moritz, P., Nishihara, R., and Jordan, M. (2016). A linearly-convergent stochastic l-bfgs algorithm. In Artificial Intelligence and Statistics, pages 249–258.

[Moussawi, 2018] Moussawi, A. (2018). Towards large scale training of autoencoders for collaborative filtering. arXiv preprint arXiv:1809.00999.

[Multiple, 2019] Multiple, A. (2019). Recommendation systems increase sales.

[Ng et al., 2011] Ng, A. et al. (2011). Sparse autoencoder. CS294A Lecture notes, 72(2011):1–19.

[Niu et al., 2018] Niu, W., Caverlee, J., and Lu, H. (2018). Neural personalized ranking for image recommendation. In Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, pages 423–431. ACM.
[Okura et al., 2017] Okura, S., Tagami, Y., Ono, S., and Tajima, A. (2017). Embedding-based news recommendation for millions of users. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 1933–1942. ACM.

[Olah, 2015] Olah, C. (2015). Understanding lstm networks.

[Pan et al., 2008] Pan, R., Zhou, Y., Cao, B., Liu, N. N., Lukose, R., Scholz, M., and Yang, Q. (2008). One-class collaborative filtering. In 2008 Eighth IEEE International Conference on Data Mining, pages 502–511. IEEE.

[Paszke et al., 2017] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

[Patterson, 2018] Patterson, D. (2018). 50 years of computer architecture: From the mainframe cpu to the domain-specific tpu and the open risc-v instruction set. In 2018 IEEE International Solid-State Circuits Conference-(ISSCC), pages 27–31. IEEE.

[Pedregosa et al., 2011] 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. Journal of machine learning research, 12(Oct):2825–2830.

[Rawat and Kankanhalli, 2016] Rawat, Y. S. and Kankanhalli, M. S. (2016). Contagnet: Exploiting user context for image tag recommendation. In Proceedings of the 24th ACM international conference on Multimedia, pages 1102–1106. ACM.

[Rendle et al., 2009] Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. (2009). Bpr: Bayesian personalized ranking from implicit feedback. In Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence, pages 452–461. AUAI Press.
[Salakhutdinov et al., 2007] Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted boltzmann machines for collaborative filtering. In Proceedings of the 24th international conference on Machine learning, pages 791–798. ACM.

[Sarwar et al., 2001] Sarwar, B. M., Karypis, G., Konstan, J. A., Riedl, J., et al. (2001). Item-based collaborative filtering recommendation algorithms. WwW, 1:285–295.

[Schmidhuber, 2015] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. Neural networks, 61:85–117.

[Sedhain et al., 2015] Sedhain, S., Menon, A. K., Sanner, S., and Xie, L. (2015). Autorec: Autoencoders meet collaborative filtering. In Proceedings of the 24th International Conference on World Wide Web, pages 111–112. ACM.

[Shambour et al., 2016] Shambour, Q., Hourani, M., and Fraihat, S. (2016). An item-based multi-criteria collaborative filtering algorithm for personalized recommender systems. International Journal of Advanced Computer Science and Applications, 7(8):274–279.

[Smirnova and Vasile, 2017] Smirnova, E. and Vasile, F. (2017). Contextual sequence modeling for recommendation with recurrent neural networks. In Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems, pages 2–9. ACM.

[Snoek et al., 2012] Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In Advances in neural information processing systems, pages 2951–2959.

[Song et al., 2018] Song, B., Yang, X., Cao, Y., and Xu, C. (2018). Neural collaborative ranking. In Proceedings of the 27th ACM International Conference on Information and Knowledge Management, pages 1353–1362. ACM.

[Srebro, 2004] Srebro, N. (2004). Learning with matrix factorizations.
[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

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

[Strub et al., 2016] Strub, F., Gaudel, R., and Mary, J. (2016). Hybrid recommender system based on autoencoders. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 11–16. ACM.

[Strub and Mary, 2015] Strub, F. and Mary, J. (2015). Collaborative filtering with stacked denoising autoencoders and sparse inputs. In *NIPS workshop on machine learning for eCommerce*.

[Sutskever et al., 2009] Sutskever, I., Hinton, G. E., and Taylor, G. W. (2009). The recurrent temporal restricted boltzmann machine. In *Advances in neural information processing systems*, pages 1601–1608.

[Tan et al., 2016] Tan, Y. K., Xu, X., and Liu, Y. (2016). Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 17–22. ACM.

[Tsai and Hung, 2012] Tsai, C.-F. and Hung, C. (2012). Cluster ensembles in collaborative filtering recommendation. *Applied Soft Computing*, 12(4):1417–1425.

[Tuan and Phuong, 2017] Tuan, T. X. and Phuong, T. M. (2017). 3d convolutional networks for session-based recommendation with content features. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 138–146. ACM.
[Vaidya and Khachane, 2017] Vaidya, N. and Khachane, A. (2017). Recommender systems-the need of the ecommerce era. In 2017 International Conference on Computing Methodologies and Communication (ICCMC), pages 100–104. IEEE.

[Vazquez and Bect, 2010] Vazquez, E. and Bect, J. (2010). Convergence properties of the expected improvement algorithm with fixed mean and covariance functions. Journal of Statistical Planning and inference, 140(11):3088–3095.

[Volkovs et al., 2017] Volkovs, M., Yu, G., and Poutanen, T. (2017). Dropoutnet: Addressing cold start in recommender systems. In Advances in Neural Information Processing Systems, pages 4957–4966.

[Wainer and Cawley, 2017] Wainer, J. and Cawley, G. (2017). Empirical evaluation of resampling procedures for optimising svm hyperparameters. The Journal of Machine Learning Research, 18(1):475–509.

[Wang et al., 2015] Wang, H., Wang, N., and Yeung, D.-Y. (2015). Collaborative deep learning for recommender systems. In Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining, pages 1235–1244. ACM.

[Wang et al., 2016] Wang, H., Xingjian, S., and Yeung, D.-Y. (2016). Collaborative recurrent autoencoder: Recommend while learning to fill in the blanks. In Advances in Neural Information Processing Systems, pages 415–423.

[Wang and Wang, 2014] Wang, X. and Wang, Y. (2014). Improving content-based and hybrid music recommendation using deep learning. In Proceedings of the 22nd ACM international conference on Multimedia, pages 627–636. ACM.

[Wei et al., 2017] Wei, J., He, J., Chen, K., Zhou, Y., and Tang, Z. (2017). Collaborative filtering and deep learning based recommendation system for cold start items. Expert Systems with Applications, 69:29–39.
[Wen et al., 2018] Wen, J., She, J., Li, X., and Mao, H. (2018). Visual background recommendation for dance performances using deep matrix factorization. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 14(1):11.

[White, 1982] White, H. (1982). Maximum likelihood estimation of misspecified models. *Econometrica: Journal of the Econometric Society*, pages 1–25.

[Wu et al., 2016a] Wu, S., Ren, W., Yu, C., Chen, G., Zhang, D., and Zhu, J. (2016a). Personal recommendation using deep recurrent neural networks in netease. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 1218–1229. IEEE.

[Wu et al., 2016b] Wu, Y., DuBois, C., Zheng, A. X., and Ester, M. (2016b). Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 153–162. ACM.

[Yang et al., 2018] Yang, Y., Yuan, S., Cer, D., Kong, S.-Y., Constant, N., Pilar, P., Ge, H., Sung, Y.-H., Strope, B., and Kurzweil, R. (2018). Learning semantic textual similarity from conversations. *arXiv preprint arXiv:1804.07754*.

[Yilmaz et al., 2008] Yilmaz, E., Kanoulas, E., and Aslam, J. A. (2008). A simple and efficient sampling method for estimating ap and ndcg. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 603–610. ACM.

[Ying et al., 2018] Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. (2018). Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983. ACM.
[Yu et al., 2014] Yu, H.-F., Hsieh, C.-J., Si, S., and Dhillon, I. S. (2014). Parallel matrix factorization for recommender systems. *Knowledge and Information Systems*, 41(3):793–819.

[Yue et al., 2007] Yue, Y., Finley, T., Radlinski, F., and Joachims, T. (2007). A support vector method for optimizing average precision. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 271–278. ACM.

[Yuyan Wang and Zhang, 2019] Yuyan Wang, Yuanchi Ning, I. L. and Zhang, X. X. (2019). Food discovery with uber eats: Recommending for the marketplace.

[Zenebe and Norcio, 2009] Zenebe, A. and Norcio, A. F. (2009). Representation, similarity measures and aggregation methods using fuzzy sets for content-based recommender systems. *Fuzzy sets and systems*, 160(1):76–94.

[Zhang et al., 2016] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*.

[Zhang et al., 2019] Zhang, S., Yao, L., Sun, A., and Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):5.

[Zhang and Sabuncu, 2018] Zhang, Z. and Sabuncu, M. (2018). Generalized cross entropy loss for training deep neural networks with noisy labels. In *Advances in neural information processing systems*, pages 8778–8788.

[Zhao and Shang, 2010] Zhao, Z.-D. and Shang, M.-S. (2010). User-based collaborative-filtering recommendation algorithms on hadoop. In *2010 Third International Conference on Knowledge Discovery and Data Mining*, pages 478–481. IEEE.
[Zheng et al., 2017] Zheng, L., Noroozi, V., and Yu, P. S. (2017). Joint deep modeling of users and items using reviews for recommendation. In Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, pages 425–434. ACM.

[Zhou et al., 2018] Zhou, G., Zhu, X., Song, C., Fan, Y., Zhu, H., Ma, X., Yan, Y., Jin, J., Li, H., and Gai, K. (2018). Deep interest network for click-through rate prediction. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 1059–1068. ACM.

[Zou and Webster, 2014] Zou, Y. and Webster, J. (2014). Information overload in using content management systems: Causes and consequences. In PACIS, page 235.