Deep Pairwise Ranking Approaches for Hybrid Recommender Systems

Andre Chan

Keble College

University of Oxford

A thesis submitted in partial fulfilment of the requirements for the degree of

Master of Science in Statistical Science

June 1st 2019 – September 9th 2019

MSc in Statistical Science – Feedback for Dissertations

UNIVERSITY OF OXFORD

DEPARTMENT OF STATISTICS

Title: Deep Pairwise Ranking Approaches for Hybrid Recommender Systems

Candidate: Andre Chan

Date: 24th September 2019

Structure	Serious lack of organisation					■	Exceptionally clear and coherent throughout
Theory and Literature	Inadequate use of literature, often irrelevant					■	Very comprehensive and logical analysis of relevant issues
Exposition	Seriously incoherent, poorly written					■	Articulate and insightful, showing outstanding thought
Methodology	Inappropriate approaches, carelessly undertaken					■	Highly rigorous and thorough, demonstrating strong initiative
Conclusions	Lack of comprehension of relevant issues					■	Exceptionally strong insights, accurate and analytically rigorous
Presentation	Unclear graphics or tables, inadequate referencing					■	Professional, excellent quality and meticulous in all regards

Grade: Starred Distinction

General Comments

This dissertation assesses and develops upon state-of-the-art methods for recommender systems. It studies a variety of techniques from recent machine learning literature in depth, with a focus on pairwise ranking and deep content-based recommendations. Notably, a novel contribution of the project is a hybridisation scheme for item features in the *Neural Collaborative Ranking* model.

The investigation further aims to provide a benchmark study of the joint effectiveness of these methods applied to three large-scale datasets and produced Python implementations of high quality.

Comments of Examiner A:

The project was well structured from beginning to end, the scientific approach was overall sound, the conclusions clearly discussed contributions and limitations, and the presentation was a pleasure to read. Notably, the work was carried through with minimal supervision and demonstrated Andre's ability of performing independent research. My only suggestion would have been to compare the models on more datasets, to make the conclusions more robust.

Comments of Examiner B:

The dissertation overall is very well written, with a clear exposition and a very good grasp of issues. It would be have been good to better indicate the parts that come from the literature, and the novel contributions.

Contents

Chapter 1

Introduction

The Internet has transformed our lifestyles, including how we shop, communicate and entertain ourselves. Businesses are increasing their online presence to maximise exposure of their products and increase sales. As a result, Internet users face an increasing need for systems that can manage the huge selection of choices that they have, for almost any online service – filtering information relevant to the user from the irrelevant – as a prerequisite to be able to make decisions. The challenge addressed is known as the *Information Overload*, which Milford in 1997 [1] described as 'when the amount of input to a system exceeds its processing capacity'.

Recommender systems are a major example of such an information filtering system. Their goal is to provide users with suggestions of items that they are likely to purchase, often personalised towards their preferences or tastes [2]. They are utilised successfully in many domains, including e-commerce, advertising, music, movies and research articles. In the e-commerce sector, major companies, such as eBay and ASOS, report personalised recommendations as being a main driver of increases in sales, as customers may purchase an item that is recommended to them but might not explicitly seek it out otherwise. In particular, Amazon estimates that 35% of its sales are generated from its recommendations [3]. This highlights the importance of developing effective recommender systems, which motivates the research in this dissertation.

1.1 Background and Motivation

1.1.1 Recommender Systems and Ranking

In a recommender system, there is a set of users U, a set of items I and a dataset S, containing *user-item interactions* or feedback that users from U have given towards items from I. In order to produce item recommendations for a user, the system attempts to rank the user's *unobserved* items (items that the user has not interacted with) in descending order of preference, as inferred from S. The top-k ranked items are then presented as item recommendations. This highlights a natural link between

recommender systems and the task of ranking. The two most popular approaches to ranking are *pointwise* and *pairwise* methods.

Pointwise methods are the more traditional approach for recommender systems. They model a predicted score for every user-item combination. For each user, the ranked list of item recommendations is obtained by sorting the predicted scores for the user's unobserved items in descending order [4]. However, pointwise methods tend to focus on the prediction of accurate user-item *scores*, instead of accurate *rankings* of item recommendations which are more important for the task of recommender systems.

Pairwise ranking approaches instead capture the intuition that ranking is an inherently relative exercise. From the user feedback of items in S, parts of users' pairwise preferences of one item over another are inferred. The goal of pairwise ranking is to accurately classify each user's pairwise preferences over all pairs of items I^2 [4], from which a ranked list of item recommendations can be inferred. As research into pairwise ranking approaches for recommender systems has developed in recent years, it is becoming increasingly accepted that pairwise ranking approaches are better suited for the task of accurately ranking item recommendations than pointwise approaches [5]. In alignment with this consensus, the recommender systems developed in this dissertation are based on a pairwise ranking framework, called Bayesian Personalised Ranking (BPR) [6].

1.1.2 Nature of User Feedback

Feedback from users may be *explicit*, such as in the form of a rating of a product, from 1 to 5. Alternatively, feedback may be *implicit*, reflecting the user's preferences towards items indirectly through their behaviour: for example, when they view an item, add it to their 'cart' or make a transaction. This illustrates that in e-commerce, implicit feedback is often collected in multiple channels. Since only relative comparisons are usually possible between the feedback channels, pairwise ranking approaches are naturally suitable for this setting. [7] demonstrates promising results of a pairwise ranking approach leveraging multiple channels of user feedback, in the form of interaction counts and graded implicit feedback derived from ratings. This motivates our investigation into pairwise ranking approaches for implicit feedback occurring in multiple channels.

Depending on the nature of user feedback in S, different methods to infer pairwise preferences between items have been proposed in the literature. A direct approach

used in [8] is to simply ask users which amongst a pair of items they prefer; however, this is unfeasible given the large scales of users and items in e-commerce settings. Instead, our approach is inspired by the framework used in [9], where S consists of binary implicit feedback ('click' and 'unobserved') and it is assumed that items that are 'clicked' by a user are preferred to 'unobserved' items.

To extend this idea to the setting of implicit feedback occurring in multiple channels, we introduce *positive* and *negative levels* of interactions for each feedback channel occurring in S. Each level reflects a different strength of preference or non-preference towards an item: for example, a transaction indicates a preference towards an item, whilst a 1-star rating or no interaction with an item reflects a degree of non-preference. Under the assumption that users prefer items with which they have *positively* interacted over items with which they have either *negatively* interacted or are *unobserved*, we devise a sampling scheme that attempts to sample users' pairwise preferences of an item over another. Greater sampling weights are placed on stronger positive levels for the preferred item as well as negative levels for the less preferred item, in order to obtain item pairs with a discriminating preference of the preferred item over the other.

1.1.3 Hybridisation

Recommender systems are typically classified into three main categories: content-based filtering (CBF), collaborative filtering (CF) and hybrid approaches.

In *content-based filtering* (CBF), recommendations are made based on *features* derived from the content of items and user profiles. For example, item features may be extracted from the text of given item descriptions and a user's profile capturing their preferences may be obtained by aggregating the profiles of items with which the user has interacted [10]. In *collaborative filtering* (CF), preference information is collected collaboratively from many users and the interdependencies between users and items are learned. Recommendations are made based on the assumption that similar users tend to have similar item preferences.

Hybrid approaches combine CBF and CF, thus benefiting from their complementary advantages. For example, CF tends to produce more diverse recommendations (known as *serendipity* [11]) than CBF, as it considers items that are enjoyed by similar users, but which may be different in content to items that were previously enjoyed by the user. However, CF is unable to make recommendations for new users or items that

have made no interactions with the system (known as the *cold-start problem* [12]), whilst CBF is able to mitigate this problem by using information derived from user and item features. Motivated by the advantages of hybridisation, we propose to extract textual features from available item descriptions to incorporate into our CF models using two approaches.

Our first approach uses a topic model called *Latent Dirichlet Allocation* (LDA), which infers the main themes occurring within each item description that we anticipate to influence users' preferences towards the item. [13] achieves notable improvements over CF approaches through the incorporation of features derived from LDA, particularly in sparse datasets where CF is more susceptible to the cold-start problem. Our second approach uses a natural language processing technique called Word2Vec, which assigns a vector representation for each word based on its meaning. Our approach is closely related to [14], in which we propose to use an average of the vector representations for the words in the item description, as an item feature. We anticipate that this captures the overall semantic meaning of the item description, containing information that explains users' preferences towards the item.

1.1.4 Deep Learning

Deep learning refers to a subset of machine learning based on neural networks, that in recent years has yielded tremendous contributions to domains such as natural language processing and audio processing. Hybrid recommender systems commonly leverage the potential of deep learning through the modelling of available auxiliary information for items, to use as side features in the recommender system. For example, Word2Vec, which we have described, uses deep learning to learn word representations that we use to form item features and [15] uses a convolutional neural network to learn the acoustic features of songs which are incorporated with a CF model, leading to notable improvements in recommendation quality.

On the other hand, exploration of the direct use of deep learning for learning the *interaction function* between users and items in recommender systems has only developed traction relatively recently and demonstrated promising initial results [16]. Many popular recommender system models are limited in their capacity to learn complex user-item relationships due to a restrictive choice of interaction function. For example, Matrix Factorisation (MF) methods use a fixed inner product between latent user and item features as their interaction function, which is only capable of learning linear relationships between users and items. Extensions to MF that generalise the

inner product, for example by using a kernel function [17], have been widely demonstrated to improve recommendation performance over MF. This inspires the designing of more complex interaction functions. In particular, neural networks are able to learn arbitrary interaction functions between the latent features of users and items, thus surpassing the capabilities of interaction functions such as the inner product. Major companies such as Google and Yahoo have declared substantial improvements from using deep learning for this purpose in their recommender systems [18].

Inspired by the theoretical and demonstrated potential of using deep learning for learning the user-item interaction function, we adopt a pairwise ranking approach called *Neural Collaborative Ranking* (NCR), in which a *multi-layer perceptron* (MLP) learns users' complex pairwise preferences between items. NCR is inspired by state-of-the-art approaches that use an MLP for *pointwise* ranking, such as *Neural Collaborative Filtering* (NCF) and Google's *Wide & Deep* approach [19]. Based on their success, it is anticipated that within the hidden layers of a MLP, a hierarchy of non-linear relationships between users and items can be learned, thereby exceeding the capabilities of pairwise models which use a simpler interaction function. In addition, we propose a novel architecture for NCR that is able to incorporate textual features of items extracted from item descriptions for hybridisation, as described in **1.1.3**.

1.2 Course of Investigation

In summary, the objective of this dissertation is to develop and evaluate e-commerce recommender systems based on pairwise ranking and deep learning, that leverage information from multiple channels of user feedback. Furthermore, we investigate whether using textual features derived from item descriptions using topic modelling and natural language processing techniques improves recommendation performance.

This dissertation is organised into seven chapters. In **Chapter 2**, we outline the theoretical foundations of the *Matrix Factorisation* (MF) model as a baseline for comparison with our models and in detail introduce the multi-layer perceptron (MLP) for binary classification. In **Chapter 3**, a pairwise ranking recommender system framework called *Bayesian Personalised Ranking* (BPR) is introduced and extended to accommodate multiple channels of user feedback. The BPR framework is then applied to the Matrix Factorisation model, leading to the BPR-MF model. **Chapter 4** describes *Neural Collaborative Ranking* (NCR), which generalises the BPR framework by using a neural network architecture to model the structure of pairwise preferences between

items. We explain our methods for extracting textual features from the item descriptions to incorporate into the NCR model using topic modelling (LDA) and natural language processing (TF-IDF, Word2Vec), leading to the NCR-LDA and NCR-W2V models. In **Chapter 5**, we introduce the Amazon and RetailRocket datasets to which our recommender systems are applied and explain our experimental and evaluation methodologies. We then describe our technique for hyperparameter tuning, using Bayesian Optimisation. **Chapter 6** serves to present and analyse the experimental results between the different models, datasets and key hyperparameters. Finally, the conclusions of our research are summarised in **Chapter 7** and ideas for further investigation are suggested.

1.3 Research Contributions

The main contributions of our research are threefold. Firstly, we combine state-of-the-art approaches for many active research areas in recommender systems, including pairwise ranking, multiple channels of user feedback, deep learning and hybridisation. We propose several improvements for the approaches and provide insights about their joint effectiveness, bridging related research areas that are not usually investigated collaboratively in the literature. Secondly, we develop a novel architecture for the Neural Collaborative Ranking (NCR) model that facilitates the input of item features. We design approaches to extract item features from text descriptions by drawing inspiration from techniques in topic modelling and natural language processing. Thirdly, we produce Python implementations for the recommender system models that are developed in this dissertation.

Chapter 2

Theoretical Foundations

This chapter introduces the Matrix Factorisation (MF) recommender system, which serves as a baseline model to be compared with the recommender systems developed in this dissertation. We then introduce the *multi-layer perceptron* (MLP) model for binary classification in detail, which is used in Chapter 4 for Neural Collaborative Ranking.

2.1 Matrix Factorisation

The most popular example of CF is a pointwise ranking approach called Matrix Factorisation (MF), which was popularised by the Netflix Prize in 2006 [20]. This was an open competition for the best CF algorithm to predict user ratings of movies, in which approaches based on MF were demonstrated to be particularly successful. We present the most traditional approach of MF for explicit feedback [21] and simply refer to it as MF.

Suppose that the dataset of user feedback S consists of user-item scores r_{ui}, for $u \in U$ and $i \in I$ from a subset of $U \times I$. For example, $r_{ui} \in \{1, 2, 3, 4, 5\}$ might be a rating of the item i given by the user u. The scores form a matrix $R \in \mathbb{R}^{|U| \times |I|}$, which is typically very sparse because most users will not have rated most of the items in I.

As a pointwise ranking method, MF attempts to predict the missing user-item scores in R. Its underlying modelling assumption is that user-item scores are the inner product of low-dimensional user and item *latent factors* which capture relevant user and item information. Therefore, R is approximated as a product of two low-rank matrices $P \in \mathbb{R}^{|U| \times d}$ and $Q \in \mathbb{R}^{d \times |I|}$, where $d \ll \min(|U|, |I|)$, which contain d-dimensional *latent factors* p_u and q_i for all users $u \in U$ and items $i \in I$, where $d \ll \min(|U|, |I|)$:

$$\hat{R} \coloneqq PQ.$$

The predicted score of a user u for item i is $\hat{r}_{ui} = p_u^T q_i$. For each user u, the items that are unobserved by a user are sorted in descending order according to their predicted

scores $\hat{r}_{ui} = p_u^T q_i$, which generates a ranked list of item recommendations for the user.

2.1.1 Loss Function

In order to learn the model parameters, the entries in P and Q, it is necessary to specify a loss function to minimise for when the model's predictions $\hat{r}_{ui}$ disagree with the true scores r_{ui} that the model seeks to fit. A common choice is the *quadratic loss* function, leading to the objective function J to be minimised over the entries in P and Q:

$$J = \sum_{(u,i):\, r_{ui} \in S} \left(r_{ui} - p_u^T q_i\right)^2 + \lambda_P ||P||_F^2 + \lambda_Q ||Q||_F^2.$$

The last two terms *regularise* the model parameters P and Q with an L2 penalty, shrinking their values, which helps to prevent overfitting particularly for sparse datasets [22]. The hyperparameters λ_P and λ_Q control the amount of regularisation with respect to the original loss function.

We remark that the quadratic loss function optimises for accurate predictions of user-item scores, which is characteristic of pointwise approaches (**1.1.1**). In the following example, we demonstrate that this may lead to suboptimal rankings of items. Consider a true user-item rating of 3-stars: $r_{ui} = 3$. Under the quadratic loss function, a prediction $\hat{r}_{ui}$ of 5-stars is penalised equally to 1-star. However, in the ranked list of items sorted by predicted item ratings, the item's ranking positions would differ significantly between the two predictions. As mentioned in **1.1.1**, obtaining an accurately ranked list of items is far more important than accurate predictions of the scores for the task of recommender systems, thus exemplifying a limitation of pointwise ranking.

2.1.2 Model Training

The model parameters are learned by minimising the objective function J over the entries of P and Q. The objective function J is differentiable, which motivates the use of *gradient descent* approaches to perform the minimisation. The gradients of J with respect to the model parameters, p_u and q_i, are given by:

$$\frac{\partial J}{\partial p_u} = 2 \sum_{i:\, r_{ui} \in S} -q_i \left(r_{ui} - p_u^T q_i\right) + 2\lambda_P p_u,$$

$$\frac{\partial J}{\partial q_i} = 2 \sum_{u:\, r_{ui} \in S} -p_u \left(r_{ui} - p_u^T q_i\right) + 2\lambda_Q q_i.$$

In standard gradient descent, the model parameters are updated iteratively, by taking steps proportional to the negative of the gradient of J with respect to the model parameters, at the current point. The gradient descent updates for p_u and q_i are given by:

$$p_u \leftarrow p_u - \alpha \frac{\partial J}{\partial p_u},$$

$$q_i \leftarrow q_i - \alpha \frac{\partial J}{\partial q_i},$$

where α is the *learning rate*. However, computation of the gradients of J requires the sum over all of the items with which a user u has interacted (for p_u), or the sum over all users that interacted with an item i (for q_i), which may be unfeasible for large scales of users and items. Furthermore, the summation leads to model parameters corresponding to frequent users and popular items receiving more extreme updates than others.

Instead, *stochastic gradient descent* (SGD) uses a single, randomly sampled training example $\{(u, i)\colon r_{ui} \in S\}$ to compute a stochastic approximation of the gradient for the model parameter update which alleviates the problems of standard gradient descent, leading to the updates:

$$p_u \leftarrow p_u + \alpha\big(q_i\big(r_{ui} - p_u^T q_i\big) - \lambda_P p_u\big),$$

$$q_i \leftarrow q_i + \alpha\big(p_u\big(r_{ui} - p_u^T q_i\big) - \lambda_Q q_i\big).$$

We use an implementation of MF from the *surprise* package in Python. This initialises the entries in P and Q from a $N(0, 0.1^2)$ distribution and uses SGD to learn the model parameters by optimising the objective function J, as we have described. MF serves as a baseline model to be compared with the other recommender system models developed in this dissertation.

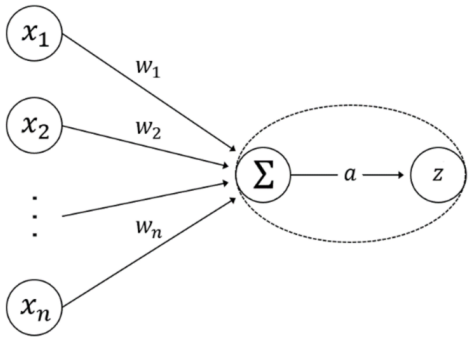
2.2 Deep Learning

Deep learning refers to a subset of machine learning based on *neural networks*, a class of models that consist of layers of nonlinear processing units called *neurons*. In this section, we introduce the framework of a *multi-layer perceptron* (MLP) as an example of a neural network in detail, which is used in the NCR models in Chapter 4.

2.2.1 The Multi-Layer Perceptron

Given input data $x \in \mathbb{R}^n$, a *neuron* has an associated weight $w \in \mathbb{R}^n$ for each input dimension and a nonlinear *activation function* a. A neuron outputs a weighted sum of its inputs acted on by the activation function: $z = a(\sum_{i=1}^{n} w_i x_i)$. A *perceptron* is a model composed of a single neuron and is depicted in Figure 2.1.

Figure 2.1: Example of a Perceptron Model



Further neurons can be augmented to the model in the same *layer*, totalling n_1 neurons: each having their own weights with respect to their inputs. The layer is associated with a weight matrix $W^{(1)} \in \mathbb{R}^{n_1 \times n}$ containing the weights for all of the neurons and an activation function a_1; we also usually include a bias term, b_1. The output of this layer is:

$$z_1 = a_1(W^{(1)}x + b_1).$$

Additional layers can be added to the model to obtain a *multi-layer perceptron* (MLP). We define the *input layer* as that which contains the units that propagates the data x into the *hidden layers* of the network, wherein a hierarchy of arbitrary data representations, such as nonlinear relationships between user and item features, can be learned. This data representations are finally fed into the *output layer*, which produces the output of the network.

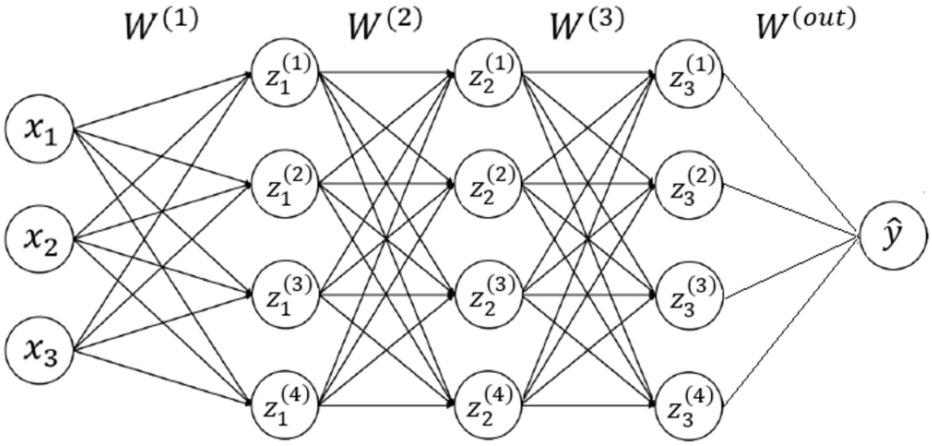
Suppose that there are L hidden layers that contain a specified number n_l of neurons, $l \in \{1, ..., L\}$. Each layer has a weight matrix $W^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$, a bias term $b_l \in \mathbb{R}^{n_l}$ and an activation function a_l. Setting $z_0 = x$, the output of all of the hidden layers is z_L, such that:

$$z_l = a_l\left(W^{(l)}z_{l-1} + b_l\right), \qquad l \in \{1, ..., L\}.$$

The output layer produces the output $\hat{y} = a_{out}(W^{(out)}z_L + b_{out})$. The model parameters Θ in an MLP are all of the weights in the network (note that the bias term is the

weight corresponding to an input of 1). An example of an MLP with 3 hidden layers is depicted in Figure 2.2.

Figure 2.2: Example of a Multi-Layer Perceptron with 3 Hidden Layers (without bias terms)



Whilst MLPs are capable of many supervised learning tasks such as classification and regression, we describe their application to *binary classification*.

2.2.2 MLPs for Binary Classification

Consider a dataset consisting of m observations: $D = \{(x_i, y_i)\}_{i=1}^m$, where $x_i \in \mathbb{R}^n$ is a feature vector and $y_i \in \{0, 1\}$ is the true binary label. Given x_i, the MLP outputs the predicted probability $\hat{y}_i \in [0, 1]$ that the label of i is 1. Assuming independence between the observations, the likelihood of the data given the model parameters Θ is:

$$p\left(D \mid \Theta\right) = \prod_{i=1}^{m} \hat{y}_i{}^{1\{y_i=1\}}(1-\hat{y}_i)^{1\{y_i=0\}}. \tag{2a}$$

Taking the negative logarithm of (2a), we obtain the *binary cross-entropy* loss function:

$$\begin{aligned} J = J(\Theta; D) = & -\sum_{i:\ y_i=1} \log \hat{y}_i \ -\sum_{i:\ y_i=0} \log\left(1-\hat{y}_i\right) \\ = & -\sum_{i=1}^{m} y_i \log \hat{y}_i \ + (1-y_i)\log\left(1-\hat{y}_i\right). \end{aligned}$$

To learn the model parameters Θ, the weights in the network, we use gradient descent approaches to minimise the loss function J. This requires the gradients of J with respect to the weights, which are computed using a technique called *backpropagation*.

In *backpropagation*, the gradients of J with respect to the output layer weights are calculated first and are iteratively reused in the computation of the gradient for the previous layers, using the 'chain rule' of calculus [23]. This allows for more efficient computation of the gradient with respect to each layer, in contrast with naively calculating the gradient of each layer separately.

As a result of the chain rule, the gradients of J with respect to an early hidden layer l involve a long multiplicative chain of the gradients of activation functions $a_{l+1}, \dots a_L$. Therefore, if the gradient of the activation functions becomes small, then the gradient of J with respect to the layer l becomes very small. These *vanishing gradients* [24] are problematic because, during gradient descent, they cause the weights in the early layers to stop being updated or lead to very small updates.

2.2.3 Choice of Activation Functions

For the *hidden layers* of an MLP, we consider three popular choices of activation functions: sigmoid, hyperbolic tangent (tanh) and ReLU, in light of the problem of *vanishing gradients*. Their properties are summarised in Table 2.1:

Table 2.1: Hidden Layer Activation Functions and their Gradients

Name	**Mathematical Expression**	**Gradient**
Sigmoid	$\sigma(x) \coloneqq 1/1 + e^{-x}$	$\sigma(x)\,(1 - \sigma(x))$
Hyperbolic Tangent	$\tanh(x) \coloneqq (e^{x} - e^{-x})/(e^{x} + e^{-x})$	$1 - \tanh^2(x)$
ReLU	$\max(0,\ x)$	$1\{x > 0\}$

The sigmoid function $\sigma(x)$ suffers from vanishing gradients at extreme values of x, where its gradient becomes small. Hyperbolic tangent, whilst it has been widely adopted, suffers from the same issue as it is a rescaled version of the sigmoid: $\tanh\left(\frac{x}{2}\right) = 2\sigma(x) - 1$. Therefore, we propose the ReLU activation function, which is proven to not suffer from vanishing gradients [25], for the hidden layers of the MLP in NCR: which is introduced in Chapter 4. Furthermore, its sparse output reduces the likelihood of overfitting particularly in our sparse datasets of user-item interactions.

For the *output layer* of the MLP in NCR, we propose the sigmoid activation function $\sigma(x) \in [0, 1]$, which is a popular choice for binary classification. It is natural choice because it can be shown to be the inverse function of the canonical parameter of a Bernoulli distribution, motivating its use in logistic regression (an MLP with no hidden layers), which can also be derived as a maximum entropy model [26].

2.2.4 Mini-Batch Stochastic Gradient Descent

In this section, we describe mini-batch stochastic gradient descent (MBSGD), that we propose to use to train NCR in Chapter 4. As before, let Θ be the model parameters (all of the weights) in the neural network and $D = \{(x_i, y_i)\}_{i=1}^{m}$ be a dataset containing features and the corresponding binary labels. Define $J(\Theta; D^*)$ to denote the objective function restricted to an arbitrary training dataset $D^* \subset D$.

In **2.1.2**, we introduced stochastic gradient descent (SGD), which uses a single, randomly sampled training example $(x_i, y_i) \in D$ to compute a stochastic approximation of the gradient for the model parameter update:

$$\Theta \leftarrow \Theta - \alpha \cdot \nabla_{\Theta} J(\Theta; (x_i, y_i)).$$

Instead, *mini-batch stochastic gradient descent* (MBSGD) samples a *batch* $B \subset D$ of training examples, such that $1 < |B| < m$, without replacement. At each training iteration, the batch B is used to compute a stochastic approximation of the gradient, leading to the update:

$$\Theta \leftarrow \Theta - \alpha \cdot \nabla_{\Theta} J(\Theta; B).$$

MBSGD approaches are much more popular than standard SGD in deep learning, where many training epochs (full passes over the training dataset) are required for the parameters to converge. This is because MBSGD is less memory-intensive than SGD and can also make use of highly optimised matrix operations in DL libraries such as *keras*, which we use for our implementation of NCR, making computation of the gradient very efficient. MBSGD also reduces the variance of parameter updates by using a larger training sample for each update, leading to more stable convergence [27]. We propose to use MBSGD to train NCR in Chapter 4. Following popular convention, we hereinafter refer to 'MBSGD' as 'SGD'.

2.2.5 Adagrad and RMSProp Optimisers

We propose two modifications of SGD called Adagrad and RMSProp for optimising the objective function J, as they have been demonstrated to significantly improve convergence performance on sparse datasets.

Adagrad, which stands for adaptive gradient algorithm, adapts the learning rate based on how often and by how much parameters are updated [28]. We describe the Adagrad update for the parameter Θ_j. Let $g_j^{(\tau)} = \nabla_{\Theta_j} J^{(\tau)}(\Theta; B)$ be the gradient with respect to the j^{th} parameter at iteration $\tau \in \{1, \dots t\}$ and let $G_j = \sum_{\tau=1}^{t} \left(g_j^{(\tau)}\right)^2$ be the

cumulative sum of the squared gradients up to iteration t. Using G_j to scale the learning rate, we obtain the Adagrad update:

$$\Theta_j^{(t+1)} \leftarrow \Theta_j^{(t)} - \frac{\alpha}{\sqrt{G_j}} g_j^{(t)}.$$

This shows that the effective learning rates are smaller for parameters Θ_j that have received more frequent and larger updates, and vice versa. This is important for our datasets with sparse user-item interactions, where the weights corresponding to frequent users or popular items may receive more extreme updates than others.

Whilst Adagrad often improves performance over SGD for sparse datasets, its main weakness is that for parameters Θ_j with consistently extreme updates, the term G_j can accumulate to become very large leading to the effective learning rate becoming very small, which causes Θ_j to stop learning.

RMSProp [29] attempts to address this issue by replacing the sum of all squared gradients in G_j with a decaying average of past squared gradients, recursively defined as:

$$E\left[g_j^2\right]_t = \gamma E\left[g_j^2\right]_{t-1} + (1-\gamma)\left(g_j^{(t)}\right)^2.$$

The parameter γ controls the decay rate and we use its default value of $\gamma = 0.9$. Using the $E\left[g_j^2\right]$ to scale the learning rate similarly to Adagrad, we obtain the RMSProp update:

$$\Theta_j^{(t+1)} \leftarrow \Theta_j^{(t)} - \frac{\alpha}{\sqrt{E\left[g_j^2\right]_t}} g_j^{(t)}.$$

The RMSProp update achieves adaptive learning rates similarly to Adagrad, yet without causing the effective learning rates to vanish.

In addition to standard SGD, we propose the Adagrad and RMSProp optimisers to train the NCR models in Chapter 4.

2.3 Neural Collaborative Filtering

In this section, we describe a framework called *Neural Collaborative Filtering* (NCF) proposed by He et al [16], which motivates *Neural Collaborative Ranking* in Chapter 4. In particular, we demonstrate that MF can be reduced to a special case of NCF.

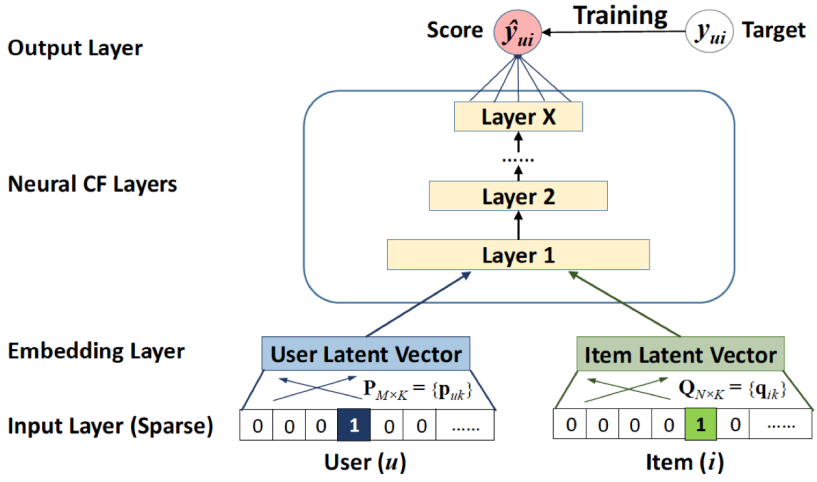
NCF uses an MLP to learn the user-item interaction function in the setting of *binary* implicit feedback represented by $y_{ui} \in \{0, 1\}$ for (u, i) from a subset of $U \times I$. For example, 0 might represent 'no interaction' and 1 might represent a 'transaction'. The

input layer receives one-hot-encoded representations for a user u and item i, which are fully connected to separate embedding layers that learn user and item *embedding vectors* p_u and q_i. These embedding vectors are analogous to the latent factors p_u and q_i in MF.

In contrast with the approach of MF (**2.1**) which uses an inner product of p_u and q_i to obtain the predicted user-item score $\hat{y}_{ui}$, instead, the embedding vectors p_u and q_i are fed into the hidden layers of a MLP, which map the vectors to the predicted score $\hat{y}_{ui} \in [0, 1]$. As NCF solves a binary classification problem, the model is trained by minimising the *binary cross-entropy* loss function (**2.2.2**). The ranked list of item recommendations is obtained in the same manner as for MF: by ranking each user's unobserved items according to their predicted scores.

As motivated in **1.1.4**, the MLP is able to *learn* an arbitrary interaction function that can capture *nonlinear* relationships between users and items. In contrast, the inner product in MF is a *fixed* interaction function that can only capture *linear* user-item interactions. The NCF model is depicted in Figure 2.3.

Figure 2.3: Neural Collaborative Filtering Model [16]



MF can be viewed as a special case of NCF, by the following construction. As mentioned, the user and item embeddings, p_u and q_i, are analogous to the latent factors learned by MF. Set the input of first layer as the element-wise product of p_u and q_i (denoted by $\odot$), which is then projected to the output layer:

$$z = p_u \odot q_i,$$

$$\hat{y}_{ui} = a_{out}\big(w^T z + b\big).$$

If a_{out} is set to be the identity function and w is enforced to be a vector with each element as 1, then we recover the MF framework. By instead using nonlinear activation functions, allowing weights to be learned from the data and adding hidden layers, the resulting NCF model is able to learn more complex user-item relationships. In particular, the state-of-the-art results in [16] demonstrate that NCF offers improvements in recommendation performance over MF that become more significant as the number of hidden layers used in the MLP increases. In Chapter 4, we introduce NCR, which draws inspiration from the architecture of NCF to solve a pairwise ranking problem.

Chapter 3

Bayesian Personalised Ranking

In this chapter, we describe the end-to-end approach for our first recommender system, based on *Bayesian Personalised Ranking* (BPR), proposed by Rendle et al [6]. First, we introduce an optimisation criterion that targets *pairwise ranking*, which we have motivated over pointwise ranking approaches (**1.1.1**). We then describe a method to learn the model parameters using stochastic gradient descent, which utilises a sampling scheme that can accommodate multiple channels of user feedback (e.g. view, add-to-cart) (**1.1.2**). Finally, we demonstrate how the BPR framework is applied to the Matrix Factorisation recommender system, leading to the BPR-MF model.

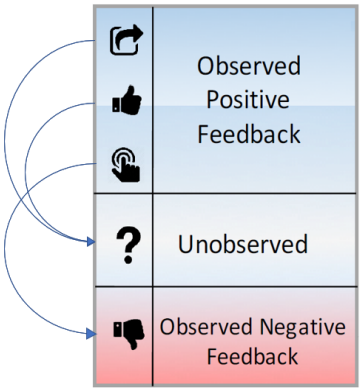
3.1 Problem Setting

Let U and I be the set of all users and items respectively and let $\boldsymbol{L} = \{L_1, \dots L_p\}$ be an ordered set of *feedback levels*. We write $L_i \succ L_j$ to denote that L_i represents a stronger preference than L_j. We distinguish between three types of levels: *positive* and *negative*, which indicate a user's preference or non-preference towards an item respectively, in addition to the *unobserved* level $L_{uo} \in \boldsymbol{L}$ corresponding to no interaction between a user and an item. Let $S \subset U \times I \times \boldsymbol{L}$ be the set of all user-item feedback, including unobserved feedback. Correspondingly, let $S_L \subset U \times I$ be the set of user-item feedback in level L and $S_{L,u} \subset I$ the set of items that u interacted with in level L.

To set up the pairwise framework, we make the same fundamental assumption as in [9], that all items with which the user *positively* interacts are preferred over all items with which they *negatively* interact or are *unobserved*. The preference between items in all other pairs of levels cannot be directly inferred. Therefore, for all positive levels $P \in \boldsymbol{L}^+$ and negative levels $N \in \boldsymbol{L}^-$, we have $P \succ L_{uo}$ and $P \succ N$.

From S, we use the above assumptions to reconstruct parts of users' pairwise preferences of one item over another, as exemplified in Figure 3.1. The set of all combinations

Figure 3.1: Examples of Pairwise Preferences, between Positive and Unobserved/Negative Levels [6]



of pairwise preferences that can be inferred from S can be defined as:

$$D_S := \{(u, i, j) \mid i \in S_{P,u} \ \wedge \ j \in S_{N,u} \colon P \in \boldsymbol{L}^+, N \in \{L_{uo}, \boldsymbol{L}^-\}\}.$$

For a user u, the set of inferred pairs of item preferences can be defined as $>_u = \{(i, j) : (u, i, j) \in D_S\} \subset I^2$. Therefore, given S, we denote a preference of an item i over item j by user u by either $(u, i, j) \in D_S$ or $(i, j) \in \ >_u$ (we often write $i >_u j$).

3.2 Objective Function for Pairwise Ranking

In this section, using a Bayesian analysis [6], we derive an optimisation criterion which targets the objective of pairwise ranking that is applied to pointwise recommender system models.

Let Θ be the model parameters of a pointwise recommender system model, such as the latent factor matrices in Matrix Factorisation. The posterior probability of the model parameters, conditional on all inferred pairwise item preferences D_S, can be expressed as:

$$p\left(\Theta \mid D_S\right) \propto p\left(D_S \mid \Theta\right) p(\Theta).$$

Assuming that all users act independently of each other and that the ordering of the pairs $(i, j) \in \ >_u$ are independent, the likelihood function can be expressed as follows:

$$p\left(D_S \mid \Theta\right) = \prod_{u \in U} p(>_u \mid \Theta) = \prod_{(u,i,j) \in D_S} p(i >_u j \mid \Theta).$$

We model the probability $p\left(i >_u j \mid \Theta\right) \coloneqq \sigma(\hat{r}_{ui} - \hat{r}_{uj})$, where σ is the sigmoid function and $(\hat{r}_{ui}, \hat{r}_{uj})$ are user-item scores that are predicted using the pointwise recommender system model. In this manner, a larger difference between $\hat{r}_{ui}$ and $\hat{r}_{uj}$ is associated with a higher predicted probability of item i being preferred over j by u.

For the prior distribution $p\left(\Theta\right)$, we use a normal distribution $N\left(0,\ \lambda_\Theta I\right)$. This has the effect of regularising the model parameters Θ with an L2 penalty, reducing the complexity of the model and preventing overfitting, particularly for our sparse datasets.

The maximum a-posteriori (MAP) estimator of the model parameters maximises the logarithm of the posterior probability of $p\left(\Theta \mid D_S\right)$, given by:

$$\begin{aligned}
\log p\left(\Theta \mid D_S\right) &\propto \log p\left(D_S \mid \Theta\right) p\left(\Theta\right) \\
&= \log \prod_{(u,i,j) \in D_S} p\left(i >_u j \mid \Theta\right) p\left(\Theta\right) \\
&\overset{\text{c}}{=} \sum_{(u,i,j) \in D_S} \log \sigma(\hat{r}_{ui} - \hat{r}_{uj}) - \frac{\lambda_\Theta}{2} \|\Theta\|_2^2, \qquad \text{(3a)}
\end{aligned}$$

where $\overset{\text{c}}{=}$ denotes equality up to a constant and λ_Θ is a hyperparameter controlling the amount of L2 regularisation. Following the Bayesian paradigm, we use (3a) as the optimisation criterion to maximise for Bayesian Personalised Ranking.

3.3 Pairwise Ranking as Binary Classification

Given a dataset of user-item interactions S, define the set of *negative pairwise preferences* $\bar{D}_S = \{(u, j, i) : (u, i, j) \in D_S\}$. We frame the pairwise ranking task as a binary classification problem on all triplets $(u, i, j) \in D_S \cup \bar{D}_S$ of positive and negative pairwise preferences inferred from S.

Define the indicator variable:

$$y_{uij} = \begin{cases} 1, & (u, i, j) \in D_S \\ 0, & (u, i, j) \in \bar{D}_S \end{cases}$$

and the predicted probability $\hat{y}_{uij} \coloneqq p\left(i >_u j \mid \Theta\right) = \sigma\left(\hat{r}_{ui} - \hat{r}_{uj}\right)$. Using the property of the sigmoid function: $\sigma(t) = 1 - \sigma\left(-t\right)$ for all $t \in \mathbb{R}$, the property $p\left(i >_u j \mid \Theta\right) = 1 - \ p\left(j >_u i \mid \Theta\right)$ holds. As a result:

$$\hat{r}_{ui} > \hat{r}_{uj} \quad \Leftrightarrow \quad \hat{y}_{uij} = \sigma\left(\hat{r}_{ui} - \hat{r}_{uj}\right) > \frac{1}{2} \quad \Leftrightarrow \quad p(i >_u j \mid \Theta) > p(j >_u i \mid \Theta).$$

Therefore, the model predicts a preference of item i over j by user u if and only if $\hat{r}_{ui} > \hat{r}_{uj} \Leftrightarrow \hat{y}_{uij} > \frac{1}{2}$. This induces binary classification problem on $(u, i, j) \in D_S \cup \bar{D}_S$ with classification threshold $\frac{1}{2}$.

3.4 Connection of Bayesian Personalised Ranking with AUC

The AUC is a popular objective function for binary classifiers. It may be defined as the empirical probability that a classifier assigns a higher prediction score for a randomly chosen positive example compared to a randomly chosen negative example [30]. Therefore, a higher AUC indicates a better performance of a binary classifier.

We define the AUC metric for BPR. Let $S_u^+ \subset I$ be the set of items positively interacted with by a user u and $S_u^- \subset I$ the set of items that are unobserved or negatively interacted with by u: such that $i \in S_u^+,\ j \in S_u^- \Leftrightarrow i >_u j$. The AUC for a user u is given by:

$$AUC(u) = \frac{1}{|S_u^+|\,|S_u^-|} \sum_{\substack{i \in S_u^+, \\ j \in S_u^-}} 1\left\{\hat{y}_{uij} > \frac{1}{2}\right\} = \frac{1}{|S_u^+|\,|S_u^-|} \sum_{\substack{i \in S_u^+, \\ j \in S_u^-}} 1\{\hat{r}_{ui} - \hat{r}_{uj} > 0\}.$$

The AUC averaged over all users in U is:

$$AUC = \frac{1}{|U|} \sum_{u \in U} AUC(u) = \sum_{(u,i,j) \in D_S} z_u 1\{\hat{r}_{ui} - \hat{r}_{uj} > 0\}, \tag{3b}$$

where $z_u = \frac{1}{|U|\left|S_u^+\right|\left|S_u^-\right|}$ is a normalising constant. Comparing (3a) with (3b), besides the normalising constant z_u and regularisation, our objective $\log p\left(\Theta \mid D_S\right)$ only differs from the AUC objective in the choice of loss function. [31] notes that when optimising for AUC, it is common to replace $1\{x > 0\}$ with a differentiable loss function. In (3b), we have substituted it with $\log \sigma(x)$.

3.5 Learning Algorithm

In this section, we describe a learning algorithm to maximise the log-posterior $\log p\left(\Theta \mid D_S\right)$ and obtain a MAP estimate for the model parameters Θ. In contrast with the AUC, $\log p\left(\Theta \mid D_S\right)$ is differentiable, which motivates the use of gradient

descent algorithms to minimise $-\log p\left(\Theta \mid D_S\right)$, which we call BPR-MIN:

$$\begin{aligned}\text{BPR-MIN} \coloneqq & -\log p\left(\Theta \mid D_S\right) \\ \overset{\text{c}}{=} & \sum_{(u,i,j)\in D_S} -\log \sigma\left(\hat{r}_{ui} - \hat{r}_{uj}\right) + \frac{\lambda_\Theta}{2}\,||\Theta||_2^2.\end{aligned}$$

Its gradient with respect to the model parameters Θ is:

$$\frac{\partial \text{BPR-MIN}}{\partial \Theta} = \sum_{(u,i,j)\in D_S} \frac{1}{1+e^{\hat{r}_{ui}-\hat{r}_{uj}}} \frac{\partial(\hat{r}_{ui}-\hat{r}_{uj})}{\partial \Theta} + \lambda_\Theta \Theta.$$

The standard gradient descent update of the model parameters Θ is:

$$\Theta \leftarrow \Theta - \alpha \frac{\partial \text{BPR-MIN}}{\partial \Theta},$$

where α is the learning rate. However, computation of the full gradient $\frac{\partial \text{BPR-MIN}}{\partial \Theta}$ is unfeasible as there are $O\left(|S|\,|I|\right)$ training triplets in D_S. Therefore, analogously to the learning approach for MF (**2.1.2**), we use SGD with a single training triplet $(u,i,j) \in D_S$, leading to the model parameter update:

$$\Theta \leftarrow \Theta - \alpha \left(\frac{1}{1+e^{\hat{r}_{ui}-\hat{r}_{uj}}} \frac{\partial(\hat{r}_{ui}-\hat{r}_{uj})}{\partial \Theta} + \lambda_\Theta \Theta \right). \tag{3c}$$

3.6 Sampling Scheme for Multiple Channels of User Feedback

Different schemes have been proposed to sample the training triplets for each SGD update. In [6], a uniform sampling scheme for $(u,i,j) \in D_S$ is proposed in favour of an approach which traverses D_S user-wise or item-wise, as the latter approach led to many consecutive updates for certain model parameters, causing slower convergence. Instead, we devise a sampling scheme, inspired by [32], which accommodates user-item feedback S that occurs in multiple levels. This scheme samples $(u,i,j) \in D_S$ weighted according to the strength of the pairwise preferences that are implied by the different feedback levels.

To sample a triplet $(u,i,j) \in D_S$, we first sample a positive user-item pair (u,i). The pair is sampled through a positive level $P \in \boldsymbol{L}^+$, from the joint sampling distribution $p\left(u,i,P\right) = p\left(u,i \mid P\right) p(P)$. We choose $p(u,i \mid P)$ to be a uniform distribution over S_P: the user-item feedback in P.

In sampling the positive level $P \in \boldsymbol{L}^+$ with sampling distribution $p(P)$, we would like to account for the strength of user preference implied from the level, in addition to the number of user-item interactions within the level denoted by $|S_P|$. A non-uniform distribution is proposed as follows:

$$p\left(P\right) = \frac{w_P \left|S_P\right|}{\sum\limits_{L \in \boldsymbol{L}^+} w_L \left|S_L\right|}. \tag{3d}$$

The weight parameters w_L reflect the strength of the positive levels. In implicit feedback settings where the weights are not known a-priori (for example, how significant a *transaction* is compared to *add-to-cart*), we optimise the weights using a hyperparameter optimisation algorithm (Chapter 5).

Conditional on (u, i) and P, we now sample the item j through sampling the level N to complete the triplet (u, i, j). In the approach taken by [32], the level N can be chosen amongst any level representing a weaker preference than P, i.e. any N such that $P \succ N$. Instead, we propose only sampling N from the unobserved level L_{uo} and the set of negative levels $\boldsymbol{L}^-$, to ensure a discriminating pairwise preference of the item i over item j.

Define the joint sampling distribution of the less preferred item j and level N as $p\left(j, N \mid u\right) = p\left(j \mid N, u\right) p\left(N \mid u\right)$. We first sample N from $p\left(N \mid u\right)$, followed by j from $p\left(j \mid N, u\right)$.

In sampling N from the negative levels $\boldsymbol{L}^-$, analogously to sampling the positive level $P \in \boldsymbol{L}^+$**,** we propose to place more sampling weighting on levels representing stronger negative preferences and proportional to the amount of feedback given in the level by the user u, as follows:

$$p_{neg}(N \mid u) = \frac{w_N \left|S_N\right|}{\sum\limits_{L \in \boldsymbol{L}^-} w_L \left|S_L\right|}.$$

As mentioned, N can also be sampled from the unobserved level, L_{uo}, uniformly. A hyperparameter β controls the proportion of samples of N that are from the unobserved feedback level L_{uo}, the effect of which is investigated in Chapter 6. The full sampling distribution $p\left(N \mid u\right)$ is given by:

$$p(N \mid u) = \begin{cases} \left(1 - \beta\right) p_{neg}(N \mid u), & N \neq L_{uo} \\ \beta, & N = L_{uo}. \end{cases}$$

Given N (and u), the item j is sampled from $p(j \mid N, u)$, chosen to be a uniform sampler over the items $S_{N,u}$, the items interacted with by u in level N. This completes the sampled triplet $(u, i, j) \in D_S$ used for the SGD update.

3.7 Application of Bayesian Personalised Ranking to Matrix Factorisation

We use the framework of Matrix Factorisation (introduced in **2.1**) to model the user-item scores $\hat{r}_{ui} = p_u^T q_i$ used in the predicted probability $p(i >_u j \mid \Theta) = \sigma(\hat{r}_{ui} - \hat{r}_{uj})$. The model parameters Θ are optimised with respect to the *pairwise* BPR-MIN loss function, leading to the *BPR-MF* model. This is in contrast with MF, whose parameters are optimised with respect to the *pointwise* quadratic loss function. Our goal is not to predict the score $\hat{r}_{ui}$, but instead to classify the difference between $\hat{r}_{ui} - \hat{r}_{uj}$. This approach is closely linked to the Bradley-Terry model [33].

In MF, the model parameters are the matrices $\Theta = (P, Q)$. The entries of the matrices are randomly initialised from a $N(0, 0.1^2)$ distribution. They are optimised by minimising the BPR-MIN objective, using SGD with the multi-feedback sampling scheme (**3.5, 3.6**). The SGD update (3c) requires $\hat{r}_{ui} - \hat{r}_{uj}$ and its gradient with respect to the model parameters $\Theta = \{P,\ Q\}$, which are given by:

$$\hat{r}_{ui} - \hat{r}_{uj} = p_u^T q_i - p_u^T q_j,$$

$$\frac{\partial(\hat{r}_{ui} - \hat{r}_{uj})}{\partial \theta} = \begin{cases} q_{ki} - q_{kj}, & \theta = p_{uk} \\ p_{uk}, & \theta = q_{ki} \\ -p_{uk}, & \theta = q_{kj} \\ 0, & \text{otherwise.} \end{cases}$$

Furthermore, we use three regularisation constants $\lambda_\Theta = (\lambda_u, \lambda_i, \lambda_j)$ for the model parameters corresponding to the user p_u, preferred item q_i and less preferred item q_j.

When the model parameters have been learned, recommendations are made in the same way as in the MF model, described in **2.1**. For each user u, the items that are unobserved by a user are sorted in descending order of their predicted scores $\hat{r}_{ui} = p_u^T q_i$, which generates a ranked list of item recommendations for the user. The BPR-MF model is evaluated in Chapter 6.

Chapter 4

Neural Collaborative Ranking

In the previous chapter, we introduced *Bayesian Personalised Ranking*, a pairwise ranking optimisation criterion that can be applied to pointwise recommender system models. In particular, it was demonstrated that the pairwise ranking task can be reframed as a binary classification problem (**3.3**), of predicting the correct pairwise item preference for $(u, i, j) \in D_S \cup \bar{D}_S$. In this chapter, we describe a deep learning framework called *Neural Collaborative Ranking* (NCR), proposed by Song et al [34], that uses a *multi-layer perceptron* (MLP) to solve this binary classification problem. Furthermore, we propose two techniques to hybridise NCR by using textual features extracted from item descriptions. The features are obtained using a popular topic modelling technique called *Latent Dirichlet Allocation* (LDA) and a novel NLP approach using *TF-IDF* and *Word2Vec embeddings*.

NCR is inspired from pointwise ranking approaches that use an MLP to model user-item interactions, such as *Neural Collaborative Filtering* (NCF) (**2.3**) and Google's state-of-the-art *Wide & Deep* approach [19]. As we have motivated in **1.1.4**, it is anticipated that more complex user-item interactions can be learned within the hidden layers of an MLP, in comparison with MF, for example, whose user-item interaction function is the inner product.

NCR applies this idea to BPR, in which the goal is to accurately classify users' pairwise preferences between items. Our anticipation is that a *multi-layer perceptron* (MLP) is more effective at capturing the complex structures between the learned embeddings of u, i and j, in classifying a preference $(u, i, j) \in D_S$ or non-preference $(u, i, j) \in \bar{D}_S$. This generalises BPR, where this classification is based simply on the difference between the predicted user-item scores, $\hat{r}_{ui} - \hat{r}_{uj}$, which are modelled as a fixed inner product between user and item latent factors $\hat{r}_{ui} = p_u^T q_i$, whose limitation we have described in **1.1.4**. In particular, we demonstrate that the MLP model can be reduced to BPR-MF as a special case.

4.1 Problem Setting

We reintroduce the key notations from Chapter 3. Let $S \subset U \times I \times \boldsymbol{L}$ be the set of all user-item feedback, including unobserved feedback. Define the set of all positive pairwise preferences inferred from S as

$$D_S = \{(u, i, j) \mid i \in S_{P,u} \ \wedge \ j \in S_{N,u} \colon P \in \boldsymbol{L}^+, N \in \{L_{uo}, \boldsymbol{L}^-\}\}$$

and the set of all negative pairwise preferences as $\bar{D}_S = \{(u, j, i) : (u, i, j) \in D_S\}$. For all triplets $(u, i, j) \in D_S \cup \bar{D}_S$, define the indicator variable

$$y_{uij} = \begin{cases} 1, & (u, i, j) \in D_S \\ 0, & (u, i, j) \in \bar{D}_S. \end{cases}$$

The NCR model takes a triplet $(u, i, j) \in D_S \cup \bar{D}_S$ and its corresponding label $y_{uij} \in \{0, 1\}$ as an input and outputs a score $\hat{y}_{uij} \in [0, 1]$. This solves a binary classification problem, in which the model associates the semantics of $(u, i, j) \in D_S$ with the label 1 and $(u, i, j) \notin D_S$ with the label 0.

Note that $\hat{y}_{uij}$ and $\hat{y}_{uji}$ are only *indicative* of the model's predicted probability of $(u, i, j) \in D_S$ and $(u, j, i) \in D_S$ respectively. The model does not guarantee the property $\hat{y}_{uij} = 1 - \hat{y}_{uji}$, unlike in BPR where it does hold (**3.3**). Instead, we instantiate the predictive rule that

$$\hat{y}_{uij} > \hat{y}_{uji} \quad \Rightarrow \quad \text{predict } (u, i, j) \in D_S, \ \textit{otherwise} \text{ predict } (u, j, i) \in D_S. \tag{4a}$$

4.2 NCR Framework

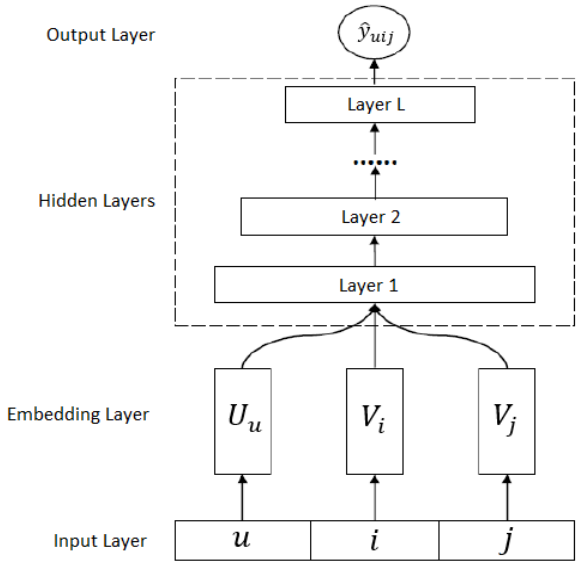
In this section, we introduce the MLP model for pairwise ranking and explain how BPR-MF can be derived as a special case, obtaining *Neural BPR* (NBPR). The MLP and NBPR models are concatenated, resulting in the NCR model.

4.2.1 Multi-Layer Perceptron for Pairwise Ranking

We introduce the MLP framework as one of the two base models for NCR, depicted in Figure 4.1, layer-by-layer:

1. The bottom *input layer* consists of a triplet $(u, i, j) \in D_S \cup \bar{D}_S$. The identities of the user u and items i and j are transformed into one-hot-encoded sparse vectors $\boldsymbol{u}$, $\boldsymbol{i}$ and $\boldsymbol{j}$, of length $|U|$, $|I|$ and $|I|$ respectively.

Figure 4.1: MLP Framework for Pairwise Ranking, without Item Features [34]



2. Each vector $\boldsymbol{u}$, $\boldsymbol{i}$ and $\boldsymbol{j}$ in the input layer is fully connected to a separate *embedding layer*. This projects each sparse representation onto a dense representation, called an *embedding vector*, in a low-dimensional latent feature space. Define the three embedding vectors corresponding to $\boldsymbol{u}$, $\boldsymbol{i}$ and $\boldsymbol{j}$ as $U_u,\ V_i,\ V_j$ respectively. In the context of a latent factor model such as MF, they are analogous to the latent factors $p_u,\ q_i$ and q_j respectively.

3. The embedding vectors $U_u,\ V_i,\ V_j$ are concatenated together, resulting in a dense vector jointly encoding user preferences and item attributes:

$$a_{\text{cat}}\left(U_u,\ V_i,\ V_j\right) = z_0 = \begin{bmatrix} U_u \\ V_i \\ V_j \end{bmatrix}.$$

In **4.6**, we describe how side information is extracted from the item descriptions of i and j, which are captured in feature vectors, F_i and F_j. If they are available as inputs, we propose to concatenate them alongside $U_u,\ V_i,\ V_j$, allowing deeper latent structures within the item preferences of users to be learned by the model:

$$a_{\text{cat}}\left(U_u,\ V_i,\ V_j,\ F_i,\ F_j\right) = z_0 = \begin{bmatrix} U_u \\ V_i \\ V_j \\ F_i \\ F_j \end{bmatrix}.$$

4. The concatenated vector is then fed into a stack of L fully connected *hidden layers* of an MLP, whose framework was introduced in **2.2.1**. As we have motivated in **2.2.3**, we use the nonlinear ReLU activation function for the hidden layers, which does not suffer from the *vanishing gradients* problem and is well-suited for sparse datasets. The hidden layers have the capacity to learn nonlinear relationships between the embedding vectors of u, i and j:

$$z_l = a_l\left(W^{(l)} z_{l-1} + b_l\right), \qquad l \in \{1, ..., L\}.$$

5. The *output layer* maps the output of the final hidden layer to the prediction score $\hat{y}_{uij}$, which is indicative of how likely it is that triplet (u, i, j) belongs to D_S. As motivated in **2.2.3**, for a binary classification problem, we use the sigmoid activation function in the output layer:

$$\hat{y}_{uij} = \sigma(W^{(out)} z_L + b_{out}).$$

4.2.2 Concatenation of MLP with BPR-MF models

We demonstrate that *BPR-MF* (**3.7**) is a special case of the MLP described in **4.2.1** [34]. Let U_u, V_i, V_j be the embedding vectors corresponding to u, i and j. Set the interaction function:

$$a\left(U_u, V_i, V_j\right) = \begin{bmatrix} U_u \odot V_i \\ -\,U_u \odot V_j \end{bmatrix}$$

where $\odot$ denotes element-wise product. This vector is projected to the output layer:

$$\hat{y}_{uij} = \sigma\left(w^T \begin{bmatrix} U_u \odot V_i \\ -\,U_u \odot V_j \end{bmatrix}\right).$$

BPR-MF is exactly recovered if we enforce w to be a vector containing only 1s. When w is learned from the data, we term the model *Neural BPR (NBPR).*

As the MLP model contains many weight parameters, it carries a risk of overfitting, particularly when there is an insufficient amount of data [35]. On the other hand, NBPR does not contain hidden layers, thus it may lack the capacity to learn complex pairwise preference structures between items. Thus, following [34], we fuse the MLP and NBPR models together to obtain the *Neural Collaborative Ranking* (NCR) model, depicted in Figure 4.2. The MLP and NBPR models learn separate embeddings U_u^M, V_i^M, V_j^M and U_u^N, V_i^N, V_j^N and are concatenated at their last hidden layer:

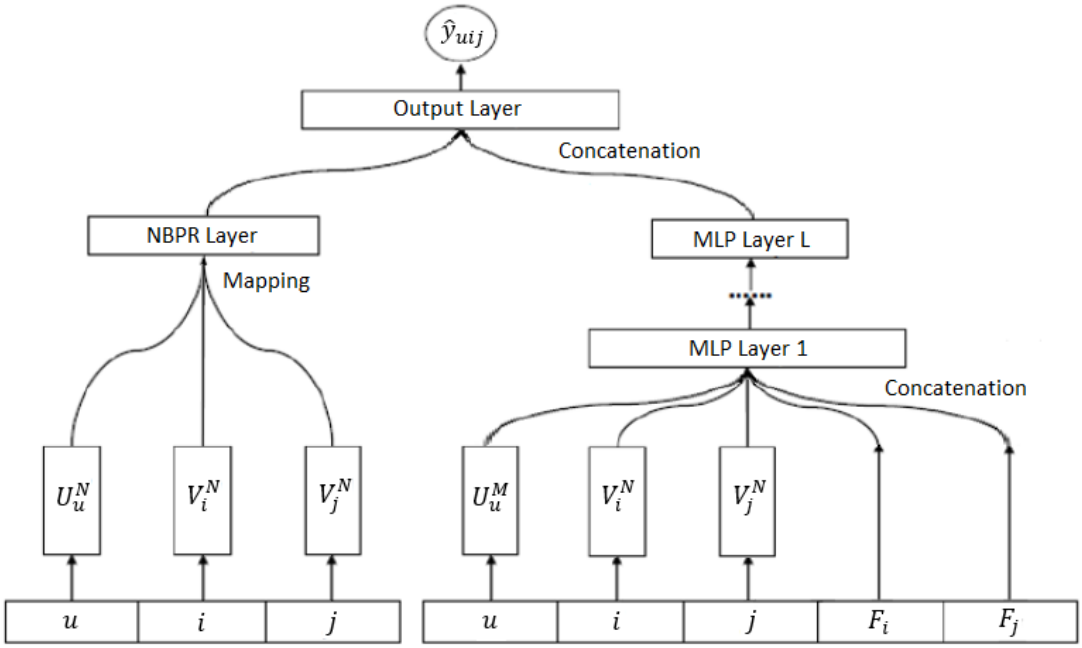
$$f^{MLP} = z_L = a_L(W^{(L)} a_{L-1}(\ \dots\ a_1(W^{(1)} z_0 + b_1)\dots)) + b_L),$$

$$f^{NBPR} = a\left(U_u^N,\ V_i^N,\ V_j^N\right) = \begin{bmatrix} U_u^N \odot V_i^N \\ -U_u^N \odot V_j^N \end{bmatrix},$$

$$\hat{y}_{uij} = \sigma\left(w^T \begin{bmatrix} f^{NBPR} \\ f^{MLP} \end{bmatrix}\right),$$

where $z_0 = \begin{bmatrix} U_u^M \\ V_i^M \\ V_j^M \\ F_i \\ F_j \end{bmatrix}$ if item features F_i and F_j are available; otherwise $z_0 = \begin{bmatrix} U_u^M \\ V_i^M \\ V_j^M \end{bmatrix}$.

Figure 4.2: NCR Framework: Concatenation of MLP and NBPR Models, with Item Features



4.3 Loss Function

We motivate the *binary cross-entropy* loss function (introduced in **2.2.2**) that is minimised with respect to the weights in NCR. Recall that $\hat{y}_{uij} \in [0, 1]$ indicates how likely it is that the triplet (u, i, j) belongs to D_S. Thus, we seek to maximise $\hat{y}_{uij}$ whenever $(u, i, j) \in D_S$ and minimise $\hat{y}_{uij}$ (equivalently, maximise $1 - \hat{y}_{uij}$) whenever $(u, i, j) \in \bar{D}_S$, leading to the objective function:

$$\prod_{(u,i,j)\in D_S} \hat{y}_{uij} \prod_{(u,i,j)\in \bar{D}_S} (1 - \hat{y}_{uij}). \tag{4b}$$

Taking the negative logarithm of (4b), we obtain the *binary cross-entropy loss*:

$$\begin{aligned} J = & - \sum_{(u,i,j) \in D_S} \log \hat{y}_{uij} - \sum_{(u,i,j) \in \bar{D}_S} \log (1 - \hat{y}_{uij}) \\ = & - \sum_{(u,i,j) \in D_S \cup \bar{D}_S} y_{uij} \log \hat{y}_{uij} + (1 - y_{uij}) \log (1 - \hat{y}_{uij}). \end{aligned}$$

Similarly to BPR, we use L2 regularisation on all of the weights in the neural network, called *weight decay*, which prevents overfitting. This augments the loss function J with the sum of L2 norms of all of the weights. The L2 norms are scaled by three separate regularisation hyperparameters: λ_{NBPR} and λ_{MLP} for the embedding weights in NBPR and MLP respectively and λ_{hidden} for the weights in the hidden layers in the MLPs.

4.4 Model Training

The weights in NCR are learned by minimising the loss function J. We use mini-batch stochastic gradient descent with batches B (**2.2.4**) using one of the optimisers: Adagrad, RMSProp or SGD (**2.2.5**). The optimiser and batch size are treated as hyperparameters to be optimised. We initialise the weights in the embedding and output layers with a $N(0, 0.05^2)$ distribution. In the remaining layers, we use independent $N\left(0, \sqrt{\frac{2}{n}}\right)$ distributions (*He initialisation* [36]), where n is the number of neurons in the previous layer. *He initialisation* scales the variance of the initialisations based on the size of the previous layer. This has been demonstrated to improve convergence performance, particularly for the ReLU activation function that we use in the hidden layers.

Each batch B consists of $|B|$ training examples $((u, i, j), y_{uij}) \in (D_S \cup \bar{D}_S) \times \{0, 1\}$. Similar to our approach in Chapter 3, we use the multi-feedback sampling scheme (**4.6**) to sample the training examples. However, the scheme samples triplets $(u, i, j) \in D_S$ which always correspond to the label $y_{uij} = 1$, which would lead to a *class imbalance* in the training data. Instead, we can convert a positive training example $(u, i, j) \in D_S$, $y_{uij} = 1$ into a negative training example $(u, j, i) \in \bar{D}_S,\ y_{uji} = 0$. Define a hyperparameter $r \in \mathbb{Z}^+$ called the *negative sampling ratio* which governs the proportion of negative examples to include in the batch B. Each sample $((u, i, j), 1)$ is added to B with probability $\frac{1}{1+r}$, otherwise $((u, j, i), 0)$ is added to B with probability $\frac{r}{1+r}$. The effect of the negative sampling ratio on model performance is investigated in Chapter 6.

4.5 Generation of Recommendations

The trained model is able to attempt to predict, for any $(u, i, j) \in U \times I \times I$, which item amongst i and j is preferred over the other by the user u, using the *predictive rule* (4a): for user u, predict a preference of item i over j if and only if $\hat{y}_{uij} > \hat{y}_{uji}$. Using this rule, we describe an algorithm [34] that iteratively constructs a ranked list of the top-k preferred items for a user u, amongst the user's unobserved items. At each iteration, the algorithm initialises a 'best' item and sequentially compares it with the remaining unobserved items, using the predictive rule. Whenever an unobserved item is predicted a preference over the current 'best' item, it becomes the new 'best' item. The 'best' item at the end of the full scan of the remaining unobserved items is then added onto the ranked list.

Algorithm 1 NCR Recommendation

Input: u, k, $S_{L_{uo},u}$ (set of unobserved items by u)
Output: R (ranked list of top-k item recommendations)

Initialise ranked list as empty set: $R \leftarrow \phi$.
for $p = 1 : k$ **do**
  Update $S_{L_{uo},u} \leftarrow S_{L_{uo},u} \setminus R$.
  Randomly permute $S_{L_{uo},u}$.
  Set $best = S_{L_{uo},u}[1]$.
  for $i = 2 : |S_{L_{uo},u}|$ **do**
    Set $proposal = S_{L_{uo},u}[i]$.
    if $\hat{y}_{(u,\ proposal,\ best)} > \hat{y}_{(u,\ best,\ proposal)}$ **then**
      Set $best = proposal$.
    end if
  end for
  Append $best$ to the end of R.
end for

Algorithm 1 is a simple approach to the common machine learning problem of inferring a top-k ranked list from pairwise comparisons. As a greedy algorithm, one of its disadvantages is that the item added to the list at each iteration is dependent on the sequence in which the unobserved items are compared with the current 'best' item. On the other hand, only $O(|I|)$ pairwise comparisons are required at each iteration, therefore $O(k|I|)$ comparisons are required to construct the ranked list, which is feasible for the sizes of k and $|I|$ that are considered.

4.6 Hybridisation

In the Amazon datasets that we consider, text descriptions for the set of items I are available. In this section, we describe two approaches, using Latent Dirichlet Allocation and Word2Vec embeddings, to derive textual features $\{F_i\}_{i \in I}$ from the descriptions. For a triplet (u, i, j) in the NCR model, we propose to concatenate the features F_i and F_j onto the embedding vectors (U_u, V_i, V_j) and the concatenated vector is then propagated through the MLP layers (**4.2.1**). It is anticipated that this additional information aids the model in learning deeper latent structures within the item preferences of users and helps to alleviate the *cold-start* problem for items. In the following sections, we assume that each item description is preprocessed so that only relevant keywords remain, the process of which is described in **5.1.2**.

4.6.1 Latent Dirichlet Allocation

Our first proposed approach to extract features from the item descriptions uses *topic modelling*, which extracts a set of 'topics' from a large *corpus* of documents (item descriptions). A topic is a distribution over words, that is typically biased around those words associated with a single theme [37]. For example, fitting the LDA topic model to the item descriptions in the *Amazon: Home & Kitchen* dataset, Table 4.1 shows the words with the highest density for 5 particular topics, chosen out of 30:

Table 4.1: Words with Highest Density for 5 Particular Topics out of 30, for LDA applied to *Amazon: Home & Kitchen* dataset

Topic 4	**Topic 9**	**Topic 10**	**Topic 18**	**Topic 22**
office	quick	cook	last	pan
plastic	clean	safe	steam	cook
task	think	day	clean	one

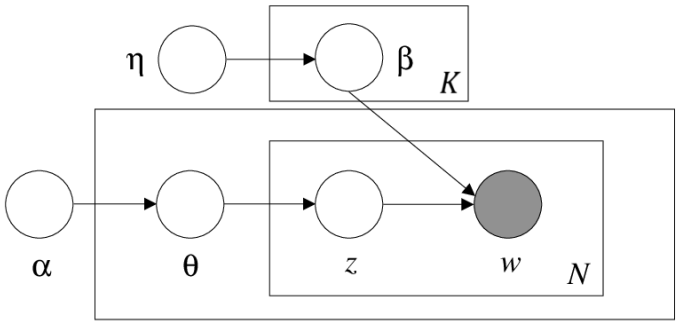
Latent Dirichlet Allocation (LDA) is the most well-known topic model and the seminal paper [38] has defined the field. Assume there are K topics $\boldsymbol{\beta} = \beta_{1:K}$, each of which is a distribution over the full set of words, with prior distribution Dir (η). The generative model of LDA is described in Algorithm 2.

Algorithm 2 Generative Model of Latent Dirichlet Allocation

For each item description w,

1. Sample the distribution over K topics for w: $\theta \sim \text{Dir}(\alpha)$.
2. For each word $n = 1 : N$,
   (a) Draw the topic associated with word n: $z_n \sim \text{Mult}(\theta)$.
   (b) Draw the word from the topic z_n: $w_n \sim \text{Mult}(\beta_{z_n})$.

Figure 4.3: Graphical Model Representation of LDA



Note that this demonstrates how the words in each description come from a mixture of topics. The topic proportions are specific to each description, but the topics are shared by the whole dataset.

Given the parameters α and $\boldsymbol{\beta}$ (a random matrix parameterised by η), the joint distribution of θ, z and w can be expressed as:

$$p(\theta, z, w \mid \alpha, \boldsymbol{\beta}) = p(\theta \mid \alpha) \prod_{n=1}^{N} p(z_n \mid \theta)\, p(w_n \mid z_n, \boldsymbol{\beta})$$

and the posterior distribution of θ and z given an observed item description w is:

$$p(\theta, z \mid w, \alpha, \boldsymbol{\beta}) = \frac{p(\theta, z, w \mid \alpha, \boldsymbol{\beta})}{p(w \mid \alpha, \boldsymbol{\beta})}. \tag{4c}$$

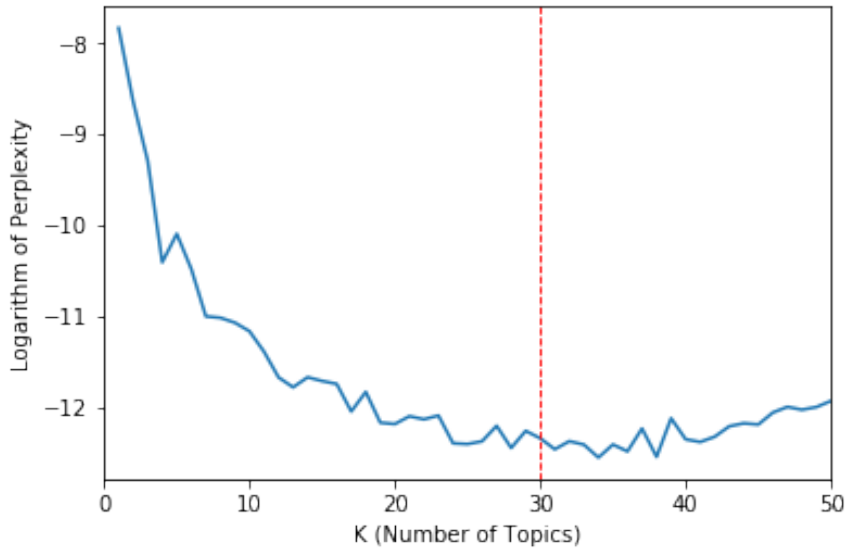
Computation of the marginal likelihood $p(w \mid \alpha, \boldsymbol{\beta})$ is intractable, therefore variational inference (details in [39]) is used to maximise (4c), from which an MAP estimate $\hat{\theta} \in \mathbb{R}^K$ of θ is obtained. $\hat{\theta}$ is the distribution over the K topics that likely generated the item description, which can be seen as an interpretable, low-dimensional representation of the description [40]. We use $\hat{\theta}$ as a feature vector for the item, that we anticipate captures latent properties that influences user preferences towards the item.

In order to determine a suitable number of topics K, we split the corpus of item descriptions into training (90%) and testing (10%) sets and compute the *perplexity* [38] on the testing set, to evaluate the generalisation performance of the LDA models for different values of K. The perplexity is monotonically decreasing in the likelihood of the testing set; therefore, a lower perplexity indicates a better generalisation performance. Given a trained LDA model M, for a testing set containing D item descriptions $\left\{w^{(d)}\right\}_{d=1}^{D}$, each of length $N^{(d)}$, the perplexity is defined as:

$$\text{perplexity} = \exp\left(-\frac{\sum_{d=1}^{D}\log\left(p\left(w_d \mid M\right)\right)}{\sum_{d=1}^{D}N^{(d)}}\right).$$

Based on the plot in Figure 4.4, of the test log-perplexity against K for the *Amazon: Groceries* dataset, there is a large range of K for which the LDA model with K topics attains a low perplexity on the testing set. We propose to use $K = 30$ topics, which ensures that the dimension of the feature vector $\hat{\theta}$ is similar to the values of the embedding dimensions of U_u, V_i, V_j that are considered in Chapter 5. This is important to ensure that the feature vector $\hat{\theta}$ has a similar influence on the MLP layers compared with the embeddings U_u, V_i, V_j.

Figure 4.4: Plot of Test Log-Perplexity against K for the Item Descriptions in *Amazon: Groceries*



We obtain the 30-dimensional LDA feature vectors from all of the item descriptions: $F_i = \hat{\theta}_i$, $i \in I$. For a triplet $(u, i, j) \in D_S \cup \bar{D}_S$ that is input to the NCR model, we concatenate the features F_i and F_j onto the embedding vectors (U_u, V_i, V_j) in the embedding layer, as described in **4.2.1**. This results in the NCR-LDA model.

4.6.2 TFIDF-weighted Word2Vec Embeddings

We describe our second approach for extracting textual features from the item descriptions, using a natural language processing technique based on *word embeddings*. A word embedding is a dense vector representation of a word from a large text, such that the word embeddings corresponding to semantically similar words are positioned in close proximity in the vector space. Word2Vec, developed by Google in 2013 [41], refers to two popular word embedding models based on neural networks, called skip-gram (which we do not consider) and CBOW. Imminently, we describe the CBOW model and how it produces word embeddings.

The principle of CBOW (for *continuous bag-of-words*) is to use the context of a given word in a text to predict that word in the vocabulary. Given a text T consisting of ordered words $t_1, t_2, t_3 \dots t_{|T|}$, define the *context* (of width s) for word t_i as $\{t_{i-s}, \dots t_{i-1},\ t_{i+1}, \dots t_{i+s}\}$. For example, the context of width 1 for the word 'ate' in 'dog ate food' is $\{\text{dog},\ \text{food}\}$.

Define the vocabulary to be the set of all unique words in the text, of cardinality n, and $x_j \in \mathbb{R}^n$ to be the one-hot-encoded representation of a word t_j. CBOW solves a multiclass classification problem: given the context of a word x_i as input, the model outputs a vector of probabilities $\hat{y}_i \in \mathbb{R}^n$ over all of the words in the vocabulary, which is optimised for predicting the word $y_i \coloneqq x_i$. We summarise the neural network architecture of the CBOW model in Algorithm 3.

To learn the embedding matrices W and W' (in Algorithm 3), the CBOW model minimises the *cross-entropy* loss J, which is analogous to the binary-cross entropy loss for binary classification (**2.2.2**):

$$J = -\sum_{i=1}^{|T|} \sum_{j=1}^{n} y_i^{(j)} \log \hat{y}_i^{(j)}.$$

The CBOW model is trained on each word and its corresponding context in the full corpus of text T. The final word embedding vectors for all of the words in the vocabulary are then given as the rows of the learned output embedding matrix $W' \in \mathbb{R}^{n \times d}$. Intuitively, this positions word embedding vectors in close proximity when the corresponding words occur in similar contexts.

Note that in considering the text used for training the CBOW model, we anticipate that the meanings of words are linguistic properties inherent of the English language, rather any particular text. Therefore, we have used pretrained word embeddings with

Algorithm 3 Neural Network Architecture for Continuous Bag-of-Words Model

Input: True word: $y_i \coloneqq x_i$
Context of x_i: $C^{(i)} = \{x_{i-s}, \dots x_{i-1}, x_{i+1}, \dots x_{i+s}\} \in \mathbb{R}^{n \times 2s}$
Embedding dimension: d
Model parameters:
Input embedding matrix $W \in \mathbb{R}^{d \times n}$
Output embedding matrix $W' \in \mathbb{R}^{n \times d}$

1. Each word in $C^{(i)}$ is fully connected to its own *input embedding layer*, producing the d-dimensional input embedding vectors: $WC^{(i)} = \{v_{i-s}, v_{i-s+1}, \dots, v_{i+s}\}$.
2. Average the input embedding vectors $\{v_{i-s}, v_{i-s+1}, \dots, v_{i+s}\}$:

$$\hat{v}_i = \frac{v_{i-s} + v_{i-s+1} \dots v_{i+s}}{2s}.$$

3. Pass $\hat{v}_i$ into a fully connected *output embedding layer*, obtaining output embedding vector $z_i = W'\hat{v}_i \in \mathbb{R}^n$.
4. Obtain predicted probabilities for each word:

$$\hat{y}_i = \text{softmax}\left(z_i\right) \coloneqq \left(\frac{\exp\left(z_i^{(j)}\right)}{\sum_{k=1}^{n} \exp\left(z_i^{(k)}\right)} \right)_{j=1}^{n} \in \mathbb{R}^n.$$

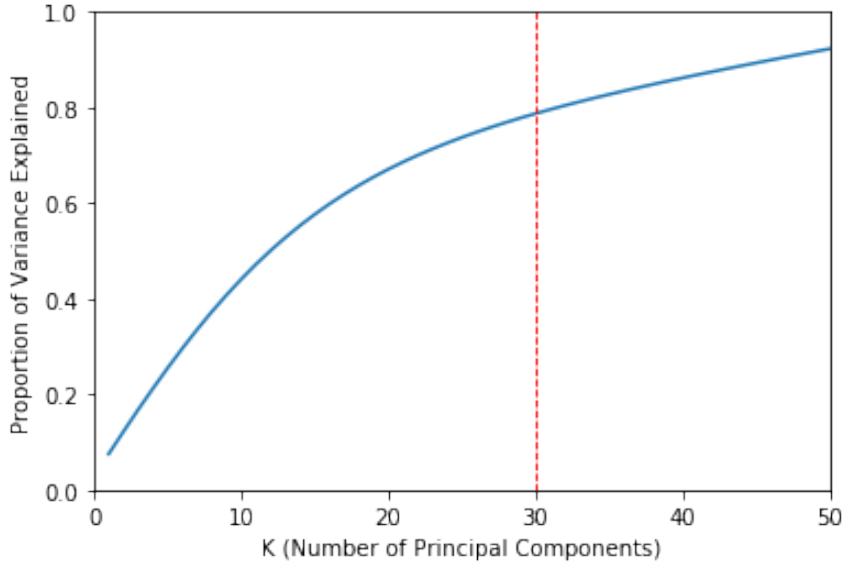
300 dimensions obtained from a much larger text corpus (Google News dataset, approximately 100 billion words) instead of training the CBOW model from the corpus of item descriptions, to ensure that the word embeddings reflect the accurate underlying meanings of words.

We propose to use a technique called *Principal Component Analysis* (PCA) [42] to reduce the number of dimensions of the word embeddings to improve upon their utility [43] and for consistency between the dimensions of the word embeddings and the feature vectors obtained from LDA. PCA applies an orthogonal transformation on the sample correlation matrix[1] of the set of word embeddings, returning a basis of K uncorrelated *principal components* (PCs), to which the word embeddings are mapped. The first PC accounts for as much variance in the data as possible, whilst each succeeding PC accounts for as much of the remaining variance as possible. For

[1]The correlation matrix is the covariance matrix of the normalised word embeddings with unit variance. [44] justifies that using normalised word embeddings often improves performance on word similarity tasks.

the *Amazon: Groceries* dataset, we plot the proportion of variance explained by K PCs against the number of PCs, K, in Figure 4.5.

Figure 4.5: Cumulative Variance Explained by the First K PCs of Word2Vec Embeddings against K, in *Amazon: Groceries*



We choose $K = 30$ because the first 30 PCs explain most (approximately 80%) of the variance in the Word2Vec embeddings (Figure 4.5) and ensures that the dimensions of the PCA-reduced embeddings are consistent with the features derived from LDA (4.6.1). This facilitates a fair comparison of the influence of the features derived from LDA and Word2Vec on the NCR model.

To obtain the item feature vector, inspired by the approach of [14], we propose to use an average of the dimensionality-reduced Word2Vec embeddings for the words occurring in the description, weighted by an importance statistic called TF-IDF (for Term Frequency Inverse Document Frequency) [45]. We hope that this average captures the semantic meaning of an item description, weighted by the important words within it, that influences user preferences towards the item.

Given a word x and an item description d, the TF-IDF score is defined as:

$$\text{TF-IDF}(x, d) = \frac{\text{Term Frequency}}{\log\left(\frac{\text{Number of Descriptions}}{\text{Document Frequency}}\right)}.$$

The Term Frequency is the number of occurrences of x in d and Document Frequency is the number of item descriptions containing the word x. TF-IDF therefore reflects

how important a word is to a description, whilst adjusting for the fact that some words occur in many descriptions.

We obtain the 30-dimensional feature vectors $\{F_i\}_{i \in I}$ from all of the item descriptions, using a TF-IDF weighted average of Word2Vec embeddings. As previously described (**4.2.1)**, for a triplet (u, i, j) input to the NCR model, we concatenate the features F_i and F_j onto the embedding vectors (U_u, V_i, V_j) after the embedding layer, which leads to the NCR-W2V model.

Chapter 5

Experimental Setup

5.1 Recommender System Datasets

In this section, we introduce the three datasets to which our recommender systems are applied: *Amazon: Groceries*, *Amazon: Home & Kitchen* and *RetailRocket*.

5.1.1 Data Description

1. *Amazon: Groceries* and *Amazon: Home & Kitchen*

   Both datasets have been extracted from publically accessible customer ratings from the website of Amazon, a major e-commerce retailer, spanning from 1996 to 2014. They are both of the same format. The user feedback, in the form of ratings, is *explicit*. We utilise the following subset of the available information:

   - **User ID:** Unique identification number for the *user*.
   - **Item ID:** Unique identification number for the *item*.
   - **Item Description:** Text description of the *item*, provided by the retailer.
   - **Rating:** An integer, between 1 to 5 inclusive, of the *item* by the *user*.
   - **Timestamp:** The time of the rating being given.

2. *RetailRocket*

   This dataset is from an anonymous Russian e-commerce retailer, made publically accessible for a *Kaggle* competition. The user feedback is *implicit* and occurs in multiple channels. We utilise the following information:

   - **User ID:** Unique identification number for the *user*.
   - **Item ID:** Unique identification number for the *item*.
   - **Feedback:** An ordered, three-level categorical variable of interactions: *'view'*, *'add-to-cart'*, *'transaction'*.
   - **Timestamp:** The time of the user feedback.

5.1.2 Data Preprocessing

For both datasets, to reduce their overall size for computational reasons, we eliminate any users with fewer than 10 interactions overall. We note that this somewhat alleviates the *sparsity* problem, thereby possibly inflating results.

1. *Amazon: Groceries and Amazon: Home & Kitchen*

   As user feedback takes the form of user-item ratings in the Amazon datasets, we explain how they are adapted for the multi-feedback sampling scheme (**3.6**). Consider each rating $\in [1, 2, 3, 4, 5]$ as a separate feedback level L. For each user, we define the ratings *above or equal to* the user's average rating as *positive* feedback $\boldsymbol{L}^+$ and the ratings *below* the user's average rating as *negative* feedback $\boldsymbol{L}^-$. Define the positive sampling weights w_L for a rating $L \in \boldsymbol{L}^+$ to be *proportional* to its value and the negative sampling weights for a rating $L \in \boldsymbol{L}^-$ to be *inversely proportional* to its value. This weighting scheme attempts to sample item pairs with a strong discriminating preference of the preferred item over the less preferred item.

   Prior to extracting textual features from the item descriptions in the Amazon datasets, several transformations are applied to convert each text description into a *bag-of-words* representation, leaving only informative keywords. The following preprocessing steps are applied:

   (a) Removal of punctuation and conversion of numbers into text.

   (b) Converting text to lower-case.

   (c) Splitting of sentences into individual words.

   (d) Removal of words containing fewer than two characters.

   Furthermore, we apply the following transformations that are common in natural language processing applications:

   (a) Lemmatisation

   Lemmatisation refers to the process of reducing words down to their base form, whilst maintaining their correct intended meaning (e.g. running $\rightarrow$ run). This condenses the overall vocabulary which reduces computational load, whilst still maintaining linguistic intuition.

(b) Removal of stop-words

Stop-words refer to commonly occurring words (e.g. the, and) that provide no useful information about the contextual meaning of the text. We remove words that occur in a defined list of stop-words. In addition, we use a heuristic filter: removing words that occur in more than half of the text descriptions. The objective of a frequency-based filter is to remove stop-words occurring within the particular domain of item descriptions.

2. *RetailRocket*

In the *RetailRocket* dataset, user feedback occurs in the following levels: {view, add-to-cart, transaction}, ordered ascendingly. They are nested such that if a user has performed an interaction with an item, then they have necessarily performed all of the lower level interactions, depicted as follows:

$$\begin{gathered}\text{view} \\ \text{view} \rightarrow \text{add-to-cart} \\ \text{view} \rightarrow \text{add-to-cart} \rightarrow \text{transaction}.\end{gathered}$$

For each user-item combination, we propose to only retain a single instance of the highest-level interaction. For example, if a user's interaction history with an item is {view, add-to-cart, view}, then we only keep {add-to-cart}. This reduces the overall size of the dataset (by 81%) and ensures compatibility with the design of the multi-feedback sampling scheme in **3.6**.

However, we remark that this approach may lose informative data about an item preference: for example, if a user's strong preference of an item leads them to make a *transaction* many times. On the other hand, exploration of the data has highlighted significant discrepancies in the number of interactions made by some users with certain items. This distorts the capability of the multi-feedback sampling scheme to fairly sample pairwise item preferences, justifying the need to remove duplicates and lower-level interactions.

We propose the following positive and negative levels of interactions for the multi-feedback sampling scheme: {transaction, add-to-cart} $\in \boldsymbol{L}^+$ and {view} $\in \boldsymbol{L}^-$. It is intuitive that *transaction* and *add-to-cart* represent positive interactions with an item. Although *view* may suggest a positive preference towards an item, after the preprocessing step that removes all higher-level interactions, a *view* means that the customer clicked into an item but did not pursue *add-to-cart*, which we hypothesise represents a possible negative preference.

Lastly, we remove users with fewer than two transactions, in order to facilitate a split of the full dataset into training, validation and testing sets, as described in **5.3**.

5.1.3 Data Exploration

A summary of the datasets after preprocessing is presented in Table 5.1. The datasets are very sparse because most users have not interacted with most items, as shown by the low densities in Table 5.1. *RetailRocket* is the least sparse dataset, followed by *Amazon: Groceries* and *Amazon: Home & Kitchen*.

Table 5.1: Density Comparison between Amazon and RetailRocket Datasets

Dataset	**Users**	**Items**	**Interactions**	**Density**
Amazon: Groceries	4323	8569	86885	0.23%
Amazon: Home and Kitchen	13694	27292	226503	0.06%
RetailRocket	1089	9230	141180	1.40%

Tables 5.2–5.4 show the proportions of positive, negative and unobserved feedback for each of the datasets, as well as the distribution of levels within each feedback type.

In the Amazon datasets, there is more positive feedback than negative feedback because in **5.1.2**, we have considered a rating to be a positive level $P \in \boldsymbol{L}^+$ if it is greater *or equal to* the user's mean rating, otherwise it is a negative level $N \in \boldsymbol{L}^-$. There are significantly fewer items rated as 3 stars in both the positive and negative feedback; we hypothesise that this is because users would tend to rate items about which they either feel particularly positive or negative.

In the RetailRocket dataset, there is significantly less positive feedback (0.39%) than negative feedback (1.01%). It is believed that this is because customers are able to *view* a large range of items, particularly in a convenient e-commerce setting, before deciding to commit to purchasing any (i.e. *transaction* and *add-to-cart*).

Table 5.2: Proportions of Feedback and Distribution of Levels within Feedback in *Amazon: Groceries*

Positive (0.14%)	**Negative (0.09%)**	**Unobserved (99.77%)**
5 Stars (55%)	4 Stars (12%)	
4 Stars (39%)	3 Stars (12%)	
3 Stars (5%)	2 Stars (29%)	
2 Stars (0%)	1 Stars (48%)	

Table 5.3: Proportions of Feedback and Distribution of Levels within Feedback in *Amazon: Home & Kitchen*

Positive (0.04%)	**Negative (0.02%)**	**Unobserved (99.94%)**
5 Stars (63%)	4 Stars (9%)	
4 Stars (30%)	3 Stars (12%)	
3 Stars (7%)	2 Stars (30%)	
2 Stars (0%)	1 Stars (49%)	

Table 5.4: Proportions of Feedback and Distribution of Levels within Feedback in *RetailRocket*

Positive (0.39%)	**Negative (1.01%)**	**Unobserved (98.60%)**
Transaction (46%)	View (100%)	
Add-to-Cart (54%)		

5.2 Evaluation Methodology

In order to evaluate a list of item recommendations for a user, it would be desirable to compare the list with the item(s) that they had truly positively interacted with in the future, called the *ground truth*, defined as follows:

Dataset	**Ground Truth**
Amazon: Groceries	Items rated 5 stars. If none exist, items
Amazon: Home and Kitchen	with the highest rating.
RetailRocket	Items given *transaction*.

As we do not have access to the items that the user interacted with in the future, consider reserving a *ground truth* item i for each user $u \in U$, such that when training the model, i is an unobserved item to u, instead of a positively interacted item. The trained model outputs a ranked list of item recommendations for each user u, amongst the user's unobserved items. However, due to the sparsity of interactions in

the datasets (Table 5.1), the set of unobserved items for each user u, $S_{L_{uo},u}$, is very large, leading to computational challenges in producing the full ranked list for each user.

Instead, we follow the common practice [34] of uniformly sampling 100 items from $S_{L_{uo},u}$, obtaining $\tilde{S}_{L_{uo},u}$. The 101 items $\tilde{S}_{L_{uo},u} \cup i$ are used to formulate the ranked list instead. The model performs well if the *ground truth* item i is ranked lowly in the list; the evaluation metrics: AUC, Mean Reciprocal Rank (MRR) at k and Recall at k [46] capture this intuition. In the definitions which follow, let u be a user and i be their reserved ground truth item. The metrics are defined for the user u and averaged over all users U.

1. *AUC*

   Applying the same characterisation of the AUC as introduced in **3.4**, the AUC for a user u is the empirical probability of the *ground truth* item i being predicted a ranking in favour of an unobserved item. For MF and BPR-MF, the AUC for user u is defined as:

   $$AUC(u) = \frac{1}{\left|\tilde{S}_{L_{uo},u}\right|} \sum_{j \in \tilde{S}_{L_{uo},u}} 1\{\hat{r}_{ui} > \hat{r}_{uj}\}.$$

   For NCR, the corresponding definition of the AUC for user u is:

   $$AUC(u) = \frac{1}{\left|\tilde{S}_{L_{uo},u}\right|} \sum_{j \in \tilde{S}_{L_{uo},u}} 1\{\hat{y}_{uij} > \hat{y}_{uji}\}.$$

   The AUC averaged over each user u is considered hereinafter as an evaluation metric:

   $$AUC = \frac{1}{|U|} \sum_{u \in U} AUC(u).$$

2. *Mean Reciprocal Rank (MRR) at k*

   In practice, each user u is only shown the top-k items in their ranked recommendation list, where k is dependent on the context of the recommender system. The Reciprocal Rank (RR) returns a higher score when the ground truth item is ranked lower in the top-k list and it returns 0 if it does not appear in the top-k. Let rank_i be the rank of the ground truth item i in the user's recommendation list. The Reciprocal Rank at k (for user u) is defined as:

   $$RR\,(u) = \frac{1}{\text{rank}_i} 1\{\text{rank}_i \leq k\}.$$

The *Mean Reciprocal Rank (MRR) at* k is the average over all users:

$$MRR = \frac{1}{|U|} \sum_{u \in U} RR(u).$$

3. *Recall at* k

   The *Recall at* k (for user u) is an indicator denoting whether the ground truth item appears in the top-k list:

   $$Recall(u) = 1\{\text{rank}_i \leq k\}.$$

   The *Recall at* k is the average over all users:

   $$Recall = \frac{1}{|U|} \sum_{u \in U} Recall(u).$$

   In contrast with the Reciprocal Rank, Recall does not reward better ranks of the ground truth item in the top-k list. However, it is more interpretable as it may be easily compared with its expected value of $\frac{k}{100}$, when the list is generated randomly.

5.2.1 Choice of k

The parameter k should be representative of how many recommended items are shown to the user from the full list of recommendations. Whilst applicational contexts are not available for RetailRocket, on the Amazon homepage [47], two 'slides' of 5 items each are displayed to the customer. If the customer clicks to see the full selection, 30 items are shown on the first page. Guided by this information, we propose to consider the *MRR* and *Recall* at $k = 5, 10$ and 30.

5.3 Experimental Methodology

Each recommender system model depends on a set of hyperparameters γ; for example, all models have an L2 regularisation hyperparameter λ. In order to maximise the performance of the model, we optimise the set of hyperparameters γ based on the AUC (**5.4** for full details), on a further set of reserved data called the *validation* set. We propose the AUC metric because of its natural relationship with pairwise ranking methods (**3.4**) and because the AUC does not require a choice of k.

Following the classical machine learning paradigm, we partition the dataset as follows:

Dataset	Contents	Purpose
Training	All interactions, except for the two most recent (according to timestamp) *ground truth* items, for each user.	Learning model parameters
Validation	The second most recent *ground truth* item, for each user.	Optimising model hyperparameters
Test	The most recent *ground truth* item, for each user.	Model evaluation

An overview of the experimental methodology is summarised as follows:

1. Given a sequence of sets of hyperparameters γ:
   (a) The training dataset is used to learn the model parameters $\Theta = \Theta(\gamma)$. The trained model outputs a ranked list of item recommendations for each user.
   (b) Evaluate $AUC_{\Theta(\gamma)}$ on the validation dataset.
2. Select $\gamma = \gamma^*$ with the highest validation AUC.
3. Using γ^*, retrain the model on both the *training* and *validation* datasets, obtaining the optimal parameters $\Theta^* = \Theta(\gamma^*)$.
4. Evaluate this model on the *test* dataset, reporting generalisation performance in terms of AUC, MRR and $Recall$ at $k = 5,\ 10,\ 30$.

5.4 Hyperparameter Optimisation

In this section, we introduce a method for optimising the hyperparameters γ, which attempts to maximise the *validation* AUC over a hyperparameter space Γ. We frame the problem as:

$$\gamma^* = \text{argmin}_{\gamma \in \Gamma} \underbrace{-AUC_{\Theta(\gamma)}}_{:= f(\gamma)}.$$

The function $f(\gamma)$ is stochastic and usually expensive to evaluate for models such as neural networks that take a long time to train.

A traditional method to optimise hyperparameters is called *grid-search*. This performs an exhaustive search over all hyperparameter combinations γ of a specified discrete subset $\Gamma^* \subset \Gamma$, called a grid, and selects γ^* as the value that minimises $f(\gamma)$. However, for large numbers of hyperparameters, the grid size increases exponentially in dimension, making it unfeasible to train the model using every grid combination

[48], particularly for models that take a long time to train. An alternative approach is *random-search*, where hyperparameter combinations are instead selected randomly, which has been demonstrated to discover better models than grid-search, granted the same computational budget [49].

5.4.1 Bayesian Optimisation

In contrast with grid-search and random-search, *Bayesian Optimisation* (BO) attempts to perform a more informed search of the space Γ, by using past evaluation results to choose the next set of hyperparameters to evaluate. We propose to use BO because it has been demonstrated to require significantly fewer evaluations of hyperparameter sets to obtain models as good as, or better than, those obtained from grid-search or random-search [50]. This is important for models such as NCR that are very expensive to train.

At each iteration of BO, a *surrogate probabilistic model* is fitted to all evaluations of $f(\gamma)$ so far. The surrogate model is inexpensive to compute compared with f and therefore it is optimised instead. Then, an *acquisition function* uses the predictive distribution of the surrogate model to determine the next location in Γ to evaluate f. The acquisition function attempts to trade-off *exploration* (learning more about f) and *exploitation* (attempting to optimise for γ using the current surrogate model).

Let $y = \{y_1, \dots y_n\}$ denote the evaluations of f at $\boldsymbol{\gamma} = \{\gamma_1, \dots \gamma_n\}$. We use an implementation of BO using *Tree Parzen Estimators* [51] from the *hyperopt* package, which model the densities $p(\gamma \mid y < y^*)$ and $p(\gamma \mid y > y^*)$, where y^* is defined to be an α-quantile of y (usually $\alpha = 0.15$). Intuitively, the two densities correspond to strong and weak hyperparameter combinations, respectively. By construction $\alpha = p\,(y < y^*)$ and $p(\gamma) = \alpha \times p\,(\gamma \mid y < y^*) + (1 - \alpha) \times p\,(\gamma \mid y > y^*)$, so the surrogate model of y given y may be obtained using Bayes' Theorem:

$$p\,(y \mid \gamma) = \frac{p\,(\gamma \mid y)\,p(y)}{p(\gamma)}.$$

It can be shown that choosing γ that maximises $\frac{p(\gamma|y<y^*)}{p(\gamma|y>y^*)}$ at each iteration - intuitively, γ with high probability under 'strong combinations': $p(\gamma \mid y < y^*)$ and low probability under 'weak combinations': $p(\gamma \mid y < y^*)$ - is equivalent to maximising the Expected Improvement (EI) acquisition function, defined as:

$$EI_{y^*}(\gamma) = \int_{-\infty}^{y^*} (y^* - y)\,p\,(y \mid \gamma)\ \ dy.$$

EI may be interpreted as the expected improvement in the value of the surrogate model from a near-minimal value y^*, when using the hyperparameter set γ. We choose y^* to be an α-quantile of y instead of the minimal value of y, to allow for exploration instead of pure exploitation.

5.4.2 Hyperparameter Subspaces

Bayesian Optimisation requires a given hyperparameter subspace $\subset \Gamma$ to search. We perform BO on the MF, BPR-MF and NCR models, with 100 iterations. For each model, we explicitly specify their hyperparameters to optimise and the respective hyperparameter subspaces for BO to search. In addition, we specify a *fixed* number of training epochs (full passes of SGD over the training dataset) used to train each model for fairness of comparison across the datasets. The hyperparameter subspaces and number of training epochs were chosen based on preliminary experimentation.

1. Bayesian Personalised Ranking applied to Matrix Factorisation (BPR-MF)

   The model parameters to be learned in BPR-MF are the latent factor matrices: $\Theta = \{P,\ Q\}$. We use 50 training epochs of SGD, for the training dataset D_S.

Hyperparameter	**Description**	**Hyperparameter Subspace**
Learning rate	Learning rate for SGD for all model parameters	[0.01, 0.1]
λ_u	L2-regularisation for the user latent factor, p_u	[0.01, 0.1]
λ_i	L2-regularisation for the preferred item latent factor, q_i	[0.01, 0.1]
λ_j	L2-regularisation for the less preferred item latent factor, q_j	[0.01, 0.1]
d	Dimension of latent factors in $P \in \mathbb{R}^{\lvert U\rvert \times d}$ and $Q \in \mathbb{R}^{d \times \lvert I\rvert}$	{20, 50, 100, 150, 200, 250, 300}
β	Proportion of unobserved sampling for the less preferred item j, in $(u, i, j) \in D_S$	{0, 0.2, ... 0.8, 1.0}
Positive weight ratio (Retail-Rocket only)	Ratio of positive sampling weights for *transaction* to *add-to-cart*: $w_{\text{transaction}}/w_{\text{add-to-cart}}$	{1.5, 2.0, 2.5, 3.0, 3.5, 4.0}

2. Neural Collaborative Ranking (NCR, NCR-LDA, NCR-W2V)

The model parameters Θ to be learned in NCR are the weights in every layer of the NBPR and MLP models. For NCR-LDA and NCR-W2V, the weights corresponding to input item features are learned in addition. We use 50 training epochs of mini-batch SGD for the training dataset $D_S \cup \bar{D}_S$.

Hyperparameter	**Description**	**Hyperparameter Subspace**
SGD Optimiser	Optimiser for Stochastic Gradient Descent	{SGD, Adagrad, RMSProp}
Batch size	Batch size for SGD, $\|B\|$	{32, 64, 128}
Learning rate	Learning rate for SGD for all model parameters	[0.0001, 0.001]
d_{NBPR}	Output dimension for *each* embedding of an input (u, i, j) in *NBPR*	{20, 30, 40, 50}
d_{MLP}	Output dimension for *each* embedding of an input (u, i, j) in *MLP*	{20, 30, 40, 50}
λ_{NBPR}	L2-regularisation for embedding layer weights in *NBPR*	[0.0001, 0.001]
λ_{MLP}	L2-regularisation for embedding layer weights in *MLP*	[0.0001, 0.001]
Layers	Hidden layer sizes in *MLP* (for 2 or 3 hidden layers)	{{10, 5}, {20, 10}, {40, 20}, {80, 40}, {160, 80}, {20, 10, 5}, {40, 20, 10}, {80, 40, 20}, {160, 80, 40}, {320, 160, 80}}
λ_{hidden}	L2-regularisation for all hidden layer weights in *MLP*	[0.0001, 0.001]
β	Proportion of unobserved sampling for the less preferred item j, in $(u, i, j) \in D_S$	{0, 0.2, ... 0.8, 1.0}
r	Negative sampling ratio: number of negative training examples per positive example, to sample for the batch B	{2, 4, 6, 8}
Positive weight ratio (Retail-Rocket only)	Ratio of positive sampling weights for *transaction* to *add-to-cart*: $w_{\text{transaction}}/w_{\text{add-to-cart}}$	{1.5, 2.0, 2.5, 3.0, 3.5, 4.0}

3. Matrix Factorisation (MF)

   The model parameters in MF are the latent factor matrices: $\Theta = \{P,\ Q\}$. We use 100 training epochs of SGD for the training dataset S.

Hyperparameter	**Description**	**Hyperparameter Subspace**
Learning rate	Learning rate for SGD for all model parameters	$[0.01,\ 0.1]$
λ	L2-regularisation for all model parameters	$[0.01,\ 0.1]$
d	Dimension of latent factors for $P \in \mathbb{R}^{\lvert U\rvert \times d}$ and $Q \in \mathbb{R}^{d \times \lvert I\rvert}$	{20, 50, 100, 150, 200, 250, 300}

Chapter 6

Results and Discussion

In this chapter, we present the results of the following five recommender system models applied to the datasets: *Amazon: Groceries, Amazon: Home & Kitchen* and *RetailRocket*.

1. Matrix Factorisation (MF)
2. Bayesian Personalised Ranking applied to Matrix Factorisation (BPR-MF)
3. Neural Collaborative Ranking (NCR)
4. Neural Collaborative Ranking with LDA features (NCR-LDA)
5. Neural Collaborative Ranking with TF-IDF-weighted Word2Vec features (NCR-W2V)

The following remarks are made in light of the nature of feedback in the datasets.

- The MF model for explicit rating feedback (as described in **2.1**) serves as a baseline for comparison with our models. Note that MF is not applied to the RetailRocket dataset, as the user feedback is implicit. However, we remark that MF may be modified to accommodate for implicit feedback datasets, through learning the values of ratings corresponding to implicit feedback levels, in addition to user and item latent factors.
- NCR-LDA and NCR-W2V cannot be applied to the RetailRocket dataset, as item descriptions are not available.

This chapter is structured by firstly presenting the results in full. This is followed by an analysis of the effects of the hyperparameters β and negative sampling ratio on the generalisation performance of the BPR-MF and NCR models. Finally, we discuss the SGD Optimisers and the ratios between the positive sampling weights for a *transaction* compared to *add-to-cart* (in RetailRocket), which were used in the final optimised models.

6.1 Analysis of Model Performance

We employ the experimental methodology explained in **5.3**, for the application of the five models on the datasets. For each model optimised on the validation set, Tables 6.1–6.3 present the validation AUC and the generalisation performance of the model on the test set, based on AUC, Recall and MRR at $k = 5, 10, 30$ across all datasets.

In our discussion of the overall performance of a model, we consider the *test* AUC as the primary evaluation metric. This is because the AUC does not require a choice for k and has a natural interpretation as the probability of the positively interacted test item being preferred over a random, unobserved item. Furthermore, Tables 6.1–6.3 also show that generalisation performance on the test set is consistent across the evaluation metrics. Figure 6.1 summarises the performance of each model in terms of test AUC, across the three datasets.

Amazon: Groceries and Home & Kitchen

In *Amazon: Groceries*, the best performing models are NCR-LDA followed by NCR (AUC = 0.828 and 0.791 respectively), whilst in *Amazon: Home & Kitchen,* NCR outperforms NCR-LDA (AUC = 0.711 and 0.698 respectively). This suggests that whether using LDA features enhances the performance of NCR depends on the dataset. The latent topics in the item descriptions of *Amazon: Groceries* may be informative about a user's preference towards such items, whilst the topics derived from *Amazon: Home & Kitchen* may either be uninformative or cause the model to be too complex. On the other hand, the inconsistencies in the test AUC between the two datasets may arise due to variance of the generalisation performance on the test sets.

Across both datasets, NCR and NCR-LDA performed comparably strongly and significantly outperformed NCR-W2V. We conclude that the quality of the features derived from item descriptions are critical to the generalisation performance of the model. The input architectures for both the LDA and Word2Vec features in the NCR model, concatenation onto the user and item embeddings, are identical and we infer that it is effective at propagating item information into the MLP layers to influence the model's predictions of pairwise preferences.

We hypothesise two main reasons for the ineffectiveness of NCR-W2V. Firstly, since the distance between Word2Vec embeddings represents the similarity between words,

using PCA (which only preserves distances along principal components) causes information contained within the distances between the dimensionality-reduced embeddings to be lost. Secondly, word embeddings are designed such that their position in the vector space is indicative of the latent meaning of the word. However, taking an average of word embeddings may not necessarily result in a combined meaning of the corresponding words, even though this approach has been used in the literature to summarise the latent meanings of combinations of words. We remark that the first reason may be addressed by considering another dimensionality reduction technique that preserves distances, such as *random projection* [52], or by retraining the Word2Vec model using the corpus of item descriptions to obtain word embeddings with the desired number of dimensions.

In both *Amazon: Groceries* and *Amazon: Home & Kitchen*, NCR and NCR-LDA outperformed BPR-MF and all four pairwise models outperformed the baseline model, MF. From BPR-MF's outperformance of MF, we conclude that the pairwise BPR objective is a more effective optimisation criterion for the task of ranking item recommendations than the quadratic loss function, which is tailored for accurate predictions of user-item scores. However, the superior performance of the NCR models compared with MF and BPR-MF suggests that in these datasets, the complex structures of pairwise preferences between items are more effectively captured by a neural network model.

From Tables 5.2–5.4, we note that for the Amazon datasets, there is a larger *class imbalance* between the cardinalities of the positive interactions and the negative or unobserved interactions, compared to RetailRocket. As explained in **4.4**, the NCR models use *negative sampling* to convert a positive training triplet $\in D_S$ to a negative triplet $\in \bar{D}_S$, which helps to alleviate the class imbalance problem [34], as further supported by the investigation of the negative sampling ratio in **6.2**. This may be a contributing factor to the NCR models outperforming BPR-MF on the Amazon datasets.

RetailRocket

In contrast with the Amazon datasets, BPR-MF outperforms NCR in terms of *test* AUC (AUC = 0.666 and 0.640 respectively). However, the *validation* AUC is lower for BPR-MF compared to NCR (AUC = 0.692 and 0.704 respectively), which is indicative that NCR may be overfitting to the validation dataset. In NCR, the MLP architecture may be learning nonlinear pairwise preference structures that are too

complex in comparison with the true pairwise preference structures exhibited in the data. This would lead to the model capturing the 'noise' in the validation set, instead of the 'signal'. We conclude that BPR-MF, which uses a simple difference between the inner product of user-item features to classify pairwise preferences, may be more effective for datasets with simpler pairwise preference structures.

Table 6.1: Test Results for *Amazon: Groceries*

		MF	**BPR-MF**	**NCR**	**NCR-LDA**	**NCR-W2V**
	Val. AUC	0.706	0.792	0.843	**0.880**	0.824
	Test AUC	*0.661*	*0.763*	*0.791*	***0.828***	*0.761*
$k = 5$	Recall	0.076	0.136	0.177	**0.214**	0.143
	MRR	0.029	0.048	0.061	**0.073**	0.053
$k = 10$	Recall	0.191	0.269	0.341	**0.382**	0.259
	MRR	0.072	0.096	0.120	**0.128**	0.089
$k = 30$	Recall	0.461	0.571	0.694	**0.802**	0.582
	MRR	0.144	0.163	0.178	**0.192**	0.164

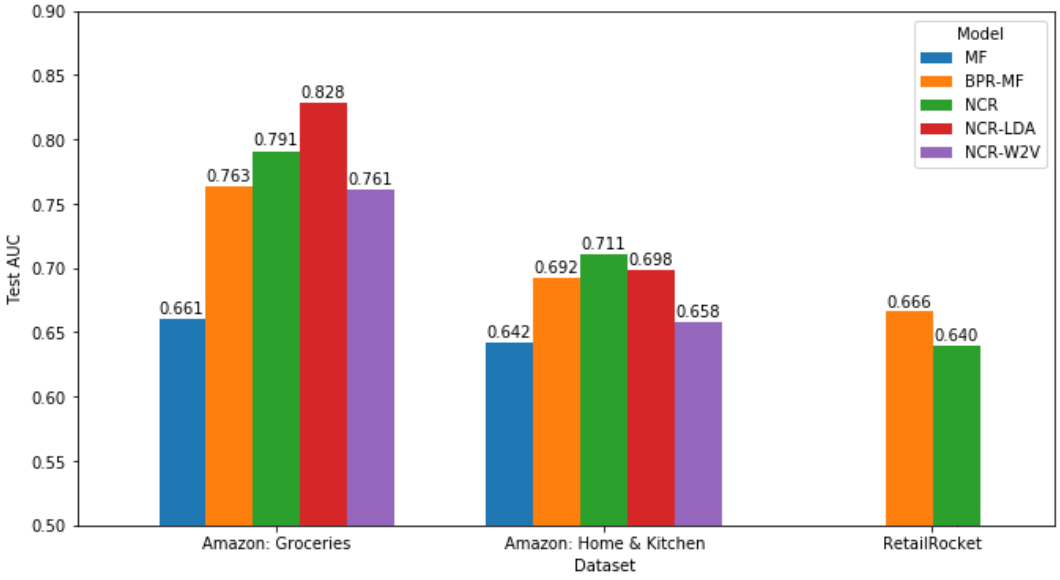
Table 6.2: Test Results for *Amazon: Home & Kitchen*

		MF	**BPR-MF**	**NCR**	**NCR-LDA**	**NCR-W2V**
	Val. AUC	0.682	0.733	**0.774**	0.749	0.693
	Test AUC	*0.642*	*0.692*	***0.711***	*0.698*	*0.658*
$k = 5$	Recall	0.083	0.092	**0.116**	0.095	0.086
	MRR	0.022	0.031	**0.041**	0.033	0.021
$k = 10$	Recall	0.165	0.190	**0.258**	0.196	0.172
	MRR	0.061	0.070	**0.086**	0.073	0.058
$k = 30$	Recall	0.422	0.489	**0.543**	0.512	0.434
	MRR	0.133	0.147	**0.156**	0.153	0.139

Table 6.3: Test Results for *RetailRocket*

		BPR-MF	**NCR**
	Val. AUC	0.692	**0.704**
	Test AUC	***0.666***	*0.640*
$k = 5$	Recall	**0.091**	0.078
	MRR	**0.032**	0.019
$k = 10$	Recall	**0.166**	0.155
	MRR	0.054	**0.058**
$k = 30$	Recall	**0.442**	0.382
	MRR	**0.134**	0.131

Figure 6.1: Summary of Test AUC Performance across Datasets and Models



6.2 Analysis of β and r on Test AUC

In this section, we perform further experiments to investigate the effects of the key hyperparameters β (unobserved sampling proportion) and r (negative sampling ratio) on the test AUC. For brevity, we only consider BPR-MF and NCR; the results for NCR-LDA and NCR-W2V exhibit comparable patterns to NCR.

We perform an analogous procedure to that described in **5.3**. For each value of β (for MF-BPR) or β and r (for NCR) in the grid in Table 6.4, BO is used to optimise the *remaining* hyperparameters on the validation set. Using the best hyperparameter set, we retrain the model on both the training and validation set and report the generalisation performance on the test set.

Table 6.4: Descriptions and Grids for β and Negative Sampling Ratio

Hyperparameter	**Description**	**Grid**
β	Proportion of unobserved sampling for the less preferred item j, in $(u, i, j) \in D_S$	$\{0,\ 0.2, \ldots 0.8, 1.0\}$
r	Negative sampling ratio: number of negative training examples per positive example, to sample for the batch B	$\{2,\ 4,\ 6,\ 8\}$

For NCR, the test AUC is plotted against β for different values of r in Figures 6.2-6.4 across the three datasets. More solid lines represent higher values of NSR. For BPR-MF, the test AUC is plotted against β across the three datasets in Figure 6.5.

Effect of β on Test AUC

Figures 6.2–6.5 depict a general pattern: for low values of β, particularly $\beta \leq 0.6$, the test AUC increases with β. This implies that higher proportions of sampling the less preferred item from negative levels of feedback, compared to unobserved feedback, leads to poor generalisation of recommendation performance on the test set. Therefore, we conclude that the 'negative' levels of feedback that we have defined (Amazon: items rated below the user's mean rating; RetailRocket: *view* but no *add-to-cart*) may not necessarily constitute a strong lack of preference for such items. For example, if an Amazon item is rated 1 out of 5 stars, the user has nevertheless purchased it; if a RetailRocket item was viewed but not purchased, the user may be interested in it despite not committing to purchasing it.

The test AUC is maximised most frequently for $\beta = 0.8$, followed by $\beta = 1$ and $\beta = 0.6$. This suggests that in sampling the less preferred item, including a small proportion of negative feedback compared to unobserved feedback optimises the generalisation performance on the test set. This implies that items interacted with negatively are associated with a non-preference in some cases, but that unobserved items are associated with a much higher level of non-preference. This aligns with the conclusions in the original paper that introduces the multi-feedback sampling scheme [32].

Effect of Negative Sampling Ratio on Test AUC for NCR

For each dataset, Table 6.5 ranks the values of the negative sampling ratio $r \in \{2, 4, 6, 8\}$ in descending order of test AUC performance (measured by the average rank of r in terms of test AUC). In the three datasets, *Amazon: Home & Kitchen* favours the highest negative sampling ratio, followed by *Amazon: Groceries* and *RetailRocket*. Cross-referencing with Tables 5.2–5.4, it is observed that a higher negative sampling ratio leads to better performances for NCR in datasets with a larger *class imbalance* between the positive interactions compared to the negative/unobserved interactions. This supports the hypothesis made in **6.1**: that negative sampling helps to alleviate the class imbalance problem.

Figure 6.2: NCR Test AUC against β as r varies, for *Amazon: Groceries*

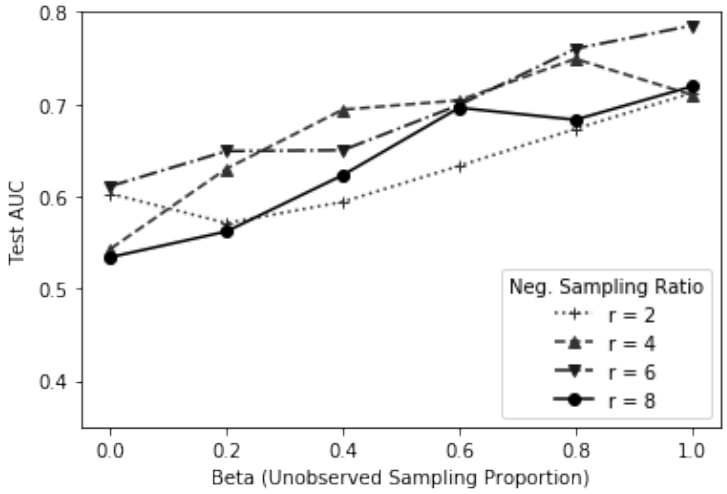


Figure 6.3: NCR Test AUC against β as r varies, for *Amazon: Home & Kitchen*

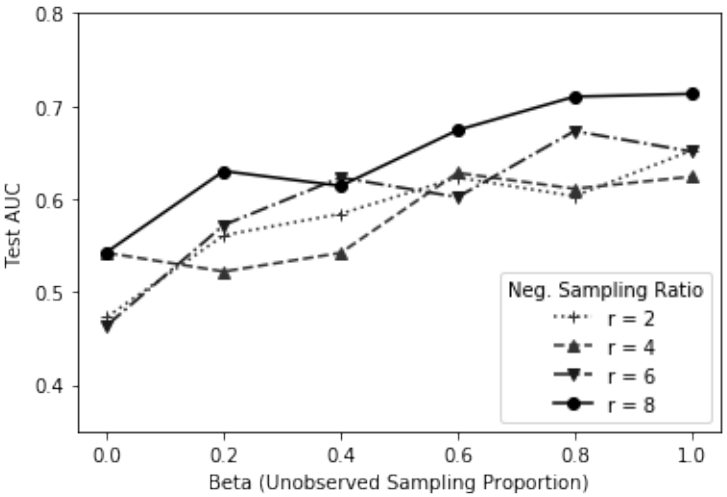


Figure 6.4: NCR Test AUC against β as r varies, for *RetailRocket*

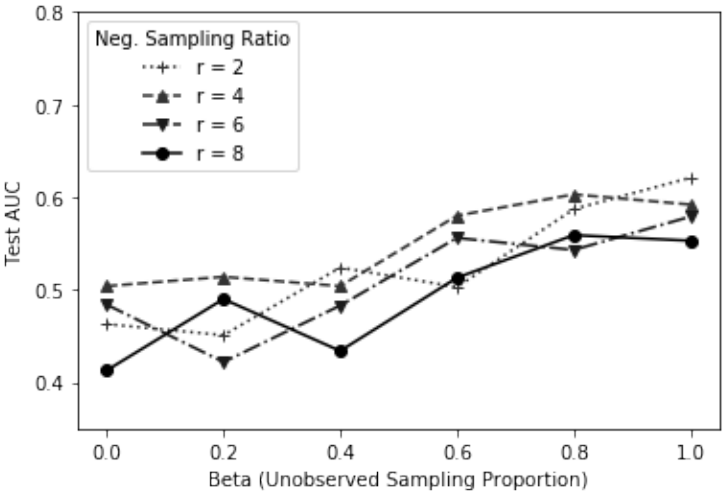
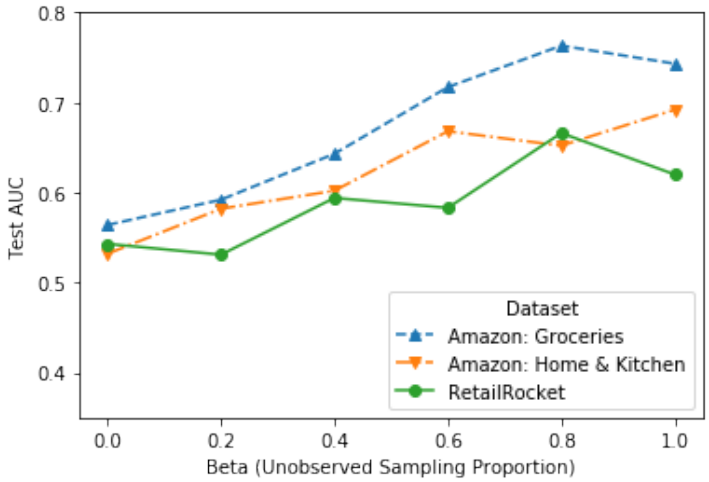


Table 6.5: Negative Sampling Ratio r in Descending Test AUC Performance across Datasets

Dataset	**Negative Sampling Ratio, r**
Amazon: Groceries	$\{6,\ 4,\ 8,\ 2\}$
Amazon: Home & Kitchen	$\{8,\ 6,\ 2,\ 4\}$
RetailRocket	$\{4,\ 2,\ 6,\ 8\}$

Figure 6.5: BPR-MF Test AUC against β, across all datasets



6.3 Analysis of SGD Optimiser and Ratio of Positive Sampling Weights

In this section, we analyse the values of two important hyperparameters that were chosen in the optimal models: SGD Optimiser (for NCR models) and the ratio between the positive sampling weights of *transaction* to *add-to-cart* (for RetailRocket). This enables conclusions about whether our choices proposed for the SGD Optimiser are appropriate and whether the multi-feedback sampling scheme for the positive levels in RetailRocket is effective.

Table 6.6 presents the values for the SGD Optimiser in the NCR models, optimised on the validation set from the hyperparameter subspace: {SGD, Adagrad, RMSProp}. The Adagrad and RMSProp optimisers are more frequently selected than SGD, which aligns with our hypothesis in **2.2.5** that Adagrad and RMSProp would lead to better learning of model parameters for the sparse recommender system datasets than standard SGD.

Table 6.7 presents the values of the sampling weight ratio between the positive levels: *transaction* and *add-to-cart* (for the RetailRocket dataset), optimised amongst the

values: {1.5, 2.0, 2.5, 3.0, 3.5, 4.0}.

The optimised ratios consistently take high values of around 3. It is inferred that a *transaction* of an item does represent a stronger preference than *add-to-cart*, which is leveraged by the multi-feedback sampling scheme for the positive levels.

Table 6.6: Optimised Values for SGD Optimiser in the NCR Models across Datasets

Dataset	**NCR**	**NCR-LDA**	**NCR-W2V**
Amazon: Groceries	RMSProp	Adagrad	SGD
Amazon: Home & Kitchen	Adagrad	Adagrad	Adagrad
RetailRocket	RMSProp	SGD	RMSProp

Table 6.7: Optimised Ratios of Positive Sampling Weights between *Transaction* and *Add-to-Cart* in the BPR-MF and NCR Models, for RetailRocket

	BPR-MF	**NCR**	**NCR-LDA**	**NCR-W2V**
RetailRocket	3.0	4.0	2.5	3.5

Chapter 7

Conclusions

In this dissertation, we have developed recommender systems based on pairwise ranking and deep learning. We have extended the frameworks of two popular models to learn the pairwise preferences of items by users: Matrix Factorisation (BPR-MF) and the Multi-Layer Perceptron (NCR). Gradient descent approaches were used to train our models, using training examples that were sampled according to the strength of users' pairwise preferences between items. In addition, we proposed a novel architecture to hybridise NCR using item features, derived from the text descriptions of items using two methods: Latent Dirichlet Allocation (NCR-LDA) and TF-IDF weighted Word2Vec Embeddings (NCR-W2V). We performed experiments on e-commerce datasets from Amazon and RetailRocket, which consist of both explicit (ratings) and implicit (multiple channels) feedback. The models were evaluated based on metrics that reward a strong placement of a known preferred test item within a ranked list of item recommendations.

We first summarise our conclusions about the multi-feedback sampling scheme. Our results imply that items that are unobserved by a user generally signal a stronger dispereference towards them than those with which users negatively interacted. On the other hand, stronger user preferences can be inferred from more positive levels of user feedback: such as *transaction* compared to *add-to-cart*. This suggests that in the multi-feedback sampling scheme, it is important to optimise the sampling weights for the positive levels and to sample the less preferred item from the user's unobserved items with a high proportion. In particular, to improve upon our treatment of levels for the RetailRocket dataset, one might consider *view* as a positive level instead of a negative level. For the Amazon datasets, a proposed improvement could be to treat the sampling weights for the positive level ratings as hyperparameters to be optimised.

In what follows, we summarise our verdicts of the four pairwise ranking models (BPR-MF, NCR, NCR-LDA and NCR-W2V) based on inter-comparisons between them and with the baseline model, MF. Whilst we have attempted to keep experimental conditions consistent to facilitate fair comparisons between models, the varying

architectures of the models necessitate differences in the initialisation methods of model parameters and choices of SGD Optimisers. We remark that these factors may contribute to the differences in the test set performance between models.

Our simplest pairwise ranking approach, BPR-MF, outperforms MF on all datasets. This demonstrates that the pairwise BPR objective is a more effective optimisation criterion for the task of ranking item recommendations than the quadratic loss function. Thus, we conclude that BPR-MF is a more effective model than MF, which aligns with the conclusion in the original paper [6]. This is in spite of its slightly longer runtime, which we hypothesise is due to the following reasons: (a) our implementation of BPR-MF is not optimised compared to the *surprise* package implementation of MF, (b) for each user, BPR-MF solves a classification problem on $O\left(|I|^2\right)$ items whereas MF solves a regression problem on $O\left(|I|\right)$ items and (c) in BPR-MF, significant time is required to convert explicit feedback into feedback levels and to sample the training examples for SGD using the multi-feedback sampling scheme.

The three NCR models which optimise a pairwise ranking criterion also outperform the baseline model, Matrix Factorisation (MF), across the Amazon datasets. This demonstrates the potential of pairwise ranking approaches compared to pointwise approaches, supporting the growing research interests into pairwise ranking for recommender systems.

NCR generalises BPR-MF and slightly outperforms it on the Amazon datasets, whereas BPR-MF outperforms NCR on the RetailRocket dataset. NCR has the capability of learning more complex interactions between latent user and item features that underpin pairwise preference structures. However, it therefore risks overfitting to the validation set, as highlighted by the fact that across the datasets, the difference between the validation AUC and test AUC is much higher for NCR compared to MF-BPR across the datasets. Furthermore, as NCR learns a more complicated model with a larger number of parameters, it takes significantly longer to train. In addition, we hypothesise that *negative sampling* in NCR contributes to better performance on datasets with sparser positive interactions. Therefore, whether NCR is preferred over BPR-MF depends on the complexity of the pairwise preference structures in the dataset, the computational budget and the sparsity of positive interactions.

NCR-LDA outperforms NCR on *Amazon: Groceries*, although the reverse holds on *Amazon: Home & Kitchen*. This suggests that the topics derived from *Amazon: Groceries* may be informative about a user's preference towards an item, whilst the topics

derived from *Amazon: Home & Kitchen* may not be. On the other hand, the results may be inconclusive, as the inconsistency between the two datasets may arise as a result of variance in the generalisation performance on the test sets. It is deemed that a conclusion about whether using LDA features in NCR leads to an improvement in generalisation performance requires further investigation on different datasets. Considering that LDA is quick to train and that concatenating LDA features onto the embeddings in NCR does not add significant computational load in training the model, we consider NCR-LDA to be a competitive alternative to NCR.

In contrast with NCR-LDA, NCR-W2V performs significantly worse than both NCR and NCR-LDA on the Amazon datasets. We hypothesise that this is due to information lost from the word embedding vectors through dimensionality reduction and averaging. Furthermore, we note that the Word2Vec model requires significant time to both train and look-up embedding vectors. We conclude that the approach of using averaged Word2Vec embeddings as item features in NCR is ineffective.

7.1 Further Work

The results of our investigation open up several directions for further research. Firstly, as the pairwise BPR criterion significantly improves upon the traditional pointwise objective for the MF model, it is promising to apply BPR to more complex pointwise recommender system models. Secondly, we have demonstrated that the quality of textual features used in NCR strongly influences the accuracy of predictions of pairwise preferences between items. Therefore, using more specialised NLP techniques may lead to improvements in recommendation performance: for example, an LSTM approach may derive better quality features from the item descriptions than our bag-of-words approaches.

Finally, I have consciously limited parts of this work. For example, the models frameworks that we have developed are only capable of leveraging a small subset of the available features in the datasets: user-item interactions and item descriptions. The work in this dissertation supports the growing trends that pairwise ranking and deep learning are competitive approaches for recommender systems. Therefore, it may be lucrative to extend our model frameworks: for example, to utilise other types of features, to address the challenges facing recommender systems (such as cold-start) and to investigate the potential of their applications in other domains.

Bibliography

[1] C. Speier, J. S. Valacich, and I. Vessey, "The effects of task interruption and information presentation on individual decision making," in *Proceedings of the Eighteenth International Conference on Information Systems*, Atlanta, GA, USA, 1997, pp. 21–36.

[2] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, "Recommender systems handbook," 2010.

[3] M. Marshall, "Venture beat article," *Accessed:*, vol. 8, 2006. [Online]. Available: http://venturebeat.com/2006/12/10/aggregate-knowledge-raises-5m-from-kleiner-on-a-roll/

[4] T.-Y. Liu, "Learning to rank for information retrieval," *Found. Trends Inf. Retr.*, vol. 3, pp. 225–331, 2009.

[5] J. Hu and P. Li, "Decoupled collaborative ranking," in *Proceedings of the 26th International Conference on World Wide Web*. Republic and Canton of Geneva, Switzerland, 2017, pp. 1321–1329.

[6] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bayesian personalized ranking from implicit feedback," in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, Arlington, Virginia, United States, 2009, pp. 452–461.

[7] L. Lerche and D. Jannach, "Using graded implicit feedback for bayesian personalized ranking," in *Proceedings of the 8th ACM Conference on Recommender Systems*, New York, NY, USA: ACM, 2014, pp. 353–356.

[8] S. Jiang, X. Wang, C. Yuan, and W. Li, "Mining user preferences for recommendation: A competition perspective," in *Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data -12th China National Conference, Suzhou, China*, 2013, pp. 179–189.

[9] Y. Fang and L. Si, "A latent pairwise preference learning approach for recommendation from implicit feedback," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, New York, NY, USA: ACM, 2012, pp. 2567–2570.

[10] R. van Meteren and M. van Someren, "Using content-based filtering for recommendation," 2000.

[11] "Collaborative filtering advantages and disadvantages." [Online]. Available: https://developers.google.com/machine-learning/recommendation/collaborative/summary

[12] B. Lika, K. Kolomvatsos, and S. Hadjiefthymiades, "Facing the cold start problem in recommender systems," *Expert Syst. Appl.*, vol. 41, pp. 2065–2073, 2014.

[13] J. Wilson, S. Chaudhury, and B. Lall, "Improving collaborative filtering based recommenders using topic modelling," in *Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)* - Washington, DC, USA, 2014, pp. 340–346.

[14] C. Musto, G. Semeraro, M. de Gemmis, and P. Lops, "Word embedding techniques for content-based recommender systems: An empirical evaluation." in *RecSys Posters*, ser. CEUR Workshop Proceedings, P. Castells, Ed., 2015.

[15] A. v. d. Oord, S. Dieleman, and B. Schrauwen, "Deep content-based music recommendation," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, USA: Curran Asso-iates Inc., 2013, pp. 2643–2651.

[16] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th International Conference on World Wide Web*, Republic and Canton of Geneva, Switzerland, 2017, pp. 173–182.

[17] X. Liu, C. C. Aggarwal, Y. Li, X. Kong, X. Sun, and S. Sathe, "Kernelized matrix factorization for collaborative filtering," in *Proceedings of the 2016 SIAM International Conference on Data Mining, Miami, Florida, USA,* 2016, pp. 378–386.

[18] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA,* 2016, pp. 191–198.

[19] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah, "Wide & deep learning for recommender systems," *arXiv:1606.07792*, 2016.

[20] J. Bennett and S. Lanning, "The netflix prize," in *In KDD Cup and Workshop in conjunction with KDD*, 2007.

[21] Y. Koren, R. M. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *IEEE Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[22] R. Salakhutdinov and A. Mnih, "Probabilistic matrix factorization," in *Proceedings of the 20th International Conference on Neural Information Processing Systems*, USA: Curran Associates Inc., 2007, pp. 1257–1264.

[23] Hecht-Nielsen, "Theory of the backpropagation neural network," in *International 1989 Joint Conference on Neural Networks*, 1989, pp. 593–605.

[24] R. Grosse, "Lecture: Exploding and vanishing gradients," vol. 15. [Online]. Available: http://www.cs.toronto.edu/~rgrosse/courses/csc321 2017/readings/L15%20Exploding%20and%20Vanishing%20Gradients.pdf

[25] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA,* 2011, pp. 315–323.

[26] J. Mount, "The equivalence of logistic regression and maximum entropy models," 2011.

[27] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016.

[28] J. C. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011.

[29] G. Hinton, "Video Lecture: Neural networks for machine learning," 2012. [Online]. Available: https://www.coursera.org/course/neuralnets

[30] C. Cortes and M. Mohri, "AUC optimization vs. error rate minimization," in *Advances in Neural Information Processing Systems 16, 2003*, pp. 313–320.

[31] A. Herschtal and B. Raskutti, "Optimising area under the ROC curve using gradient descent," in *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*, 2004.

[32] B. Loni, R. Pagano, M. Larson, and A. Hanjalic, "Bayesian personalized ranking with multi-channel user feedback," in *Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, September 15-19, 2016*, 2016, pp. 361–364.

[33] J. Hu and P. Li, "Decoupled collaborative ranking," in *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, 2017, pp. 1321–1329.

[34] B. Song, X. Yang, Y. Cao, and C. Xu, "Neural collaborative ranking," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26, 2018*, 2018, pp. 1353–1362.

[35] I. Nusrat and S. Jang, "A comparison of regularization techniques in deep neural networks," *Symmetry*, vol. 10, no. 11, p. 648, 2018.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, 2015, pp. 1026–1034.

[37] D. M. Blei, "Probabilistic topic models," *Commun. ACM*, vol. 55, no. 4, pp. 77–84, 2012.

[38] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, 2003.

[39] A. Bhowmick, U. Prasad, and S. Kottur, "Movie recommendation based on collaborative topic modeling," 2014.

[40] J. Chang, J. L. Boyd-Graber, S. Gerrish, C. Wang, and D. M. Blei, "Reading tea leaves: How humans interpret topic models," in *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009*, pp. 288–296.

[41] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems 26: 2013*, pp. 3111–3119.

[42] J. Shlens, "A tutorial on principal component analysis," *CoRR*, vol. abs/1404.1100, 2014.

[43] V. Raunak, "Effective dimensionality reduction for word embeddings," *CoRR*, vol. abs/1708.03629, 2017.

[44] B. J. Wilson and A. M. J. Schakel, "Controlled experiments for word embeddings," *CoRR*, vol. abs/1510.02675, 2015.

[45] A. N. Aizawa, "An information-theoretic perspective of tf-idf measures," *Inf. Process. Manage.*, vol. 39, no. 1, pp. 45–65, 2003.

[46] P. Cremonesi, Y. Koren, and R. Turrin, "Performance of recommender algorithms on top-n recommendation tasks," in *Proceedings of the 2010 ACM Conference on Recommender Systems, RecSys 2010, Barcelona, Spain*, 2010, pp. 39–46.

[47] "Amazon homepage." [Online]. Available: https://www.amazon.co.uk/

[48] M. Feurer and F. Hutter, "Hyperparameter optimization," in *Automated Machine Learning - Methods, Systems, Challenges*, 2019, pp. 3–33.

[49] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," J. *Mach. Learn. Res.*, vol. 13, pp. 281–305, 2012.

[50] D. R. Jones, "A taxonomy of global optimization methods based on response surfaces," *J. Global Optimization*, vol. 21, no. 4, pp. 345–383, 2001.

[51] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems 24: 2011*, pp. 2546–2554.

[52] M. Nabil, "Random projection and its applications," *CoRR*, vol. abs/1710.03163, 2017.