UNIVERSITY OF CAPE TOWN

RESEARCH PROJECT

Collaborative Genre Tagging

Author:
James LESLIE

Supervisor:
Dr Miguel LACERDA

A project submitted in fulfilment of the requirements for the degree of M.Sc. Data Science

in the

Faculty of Science
Department of Statistical Sciences

February 4, 2020

Declaration of Authorship

I, James LESLIE, declare that this project titled, "Collaborative Genre Tagging" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a master's degree at this University.
- Where any part of this dissertation has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this project is entirely my own work.
- I have acknowledged all main sources of help.
- Where the project is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

University Of Cape Town

Abstract

Faculty of Science

Department of Statistical Sciences

M.Sc. Data Science

Collaborative Genre Tagging

by James LESLIE

Recommender systems (RS) are used extensively in online retail and on media streaming platforms to help users filter the plethora of options at their disposal. Their goal is to provide users with suggestions of products or artworks that they might like.

Content-based RS's make use of user and/or item metadata to predict user preferences, while collaborative-filtering (CF) has proven to be an effective approach in tasks such as predicting movie or music preferences of users in the absence of any metadata.

Latent factor models have have been used to achieve state-of-the-art accuracy in many CF settings, playing an especially large role in beating the benchmark set in the Netflix Prize in 2008. These models learn latent features for users and items to predict the preferences of users. The first latent factor models made use of matrix factorisation to learn latent factors, but more recent approaches have made use of neural architectures with embedding layers.

This master's dissertation outlines collaborative genre tagging (CGT), a transfer learning application of CF that makes use of latent factors to predict genres of movies, using only explicit user ratings as model inputs.

Acknowledgements

I would like to give a sincere thank you to my project supervisor, Dr Miguel Lacerda. Without his guidance and direction, the result achieved in this project would not have been possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Description

With the internet boom of the '90s, many industries have migrated much of their business to the web. Digital commerce is no exception, with Forbes magazine predicting that online sales will continue to grow to a total volume of $5.8 trillion by 2022. This number would make up over 17% of all business-to-consumer sales and would be achieved through annual compound growth rates of around 20% (McKee, 2018). The growth in this industry has presented shoppers with an overwhelming number of options to choose from, creating the need for tools to help users filter through the multitude of options (Francesco Ricci and Shapira, 2011).

The purpose of a recommender system is to help people make decisions in an area where they do not necessarily have a wealth of personal experience (Resnick and Varian, 1997). They have been developed to aid users with filtering through the wide range of options at their disposal to find those offerings most relevant to them. The output of a recommender system is usually in the form of a suggestion to the user, helping them to make decisions such as what items to buy, what music to listen to or what movie to rent (Francesco Ricci and Shapira, 2011).

As the number of options has grown, so too has the amount of user feedback data. Already in 2007, the online video subscription service Netflix had collected nearly 2 billion ratings from more than 11.7 million subscribers on over 85 thousand titles in its first 10 years of operation. Considering the number of similar streaming and online commerce services, there is indeed an overwhelming number of options for users to choose from – but also a wealth of data that can be used to make recommendations. More choices means more feedback; we now have more recorded interactions of what people do – and *don't* – like than ever before (Bennett, Lanning, et al., 2007).

Recommender systems can be divided into two categories: *collaborative filtering* (CF) and *content-based* models. CF models use only interactions between users and items to learn user preferences; they make recommendations based on what other similar users like. Content-based models make use of metadata such as user demographics or item genres; recommendations are based solely on the attributes of items, such as which actors appear in a movie or which genre it falls under. It has been shown that CF can be used to accurately predict user preferences and that, furthermore,

these models can provide more serendipitous suggestions to users than content-based models, without the need for any meta data (Yehunda Koren and R. Bell, 2011), (Lops et al., 2011), (Shani and Gunawardana, 2011). Figure 1.1 shows the different ideas behind the content-based and CF recommenders.

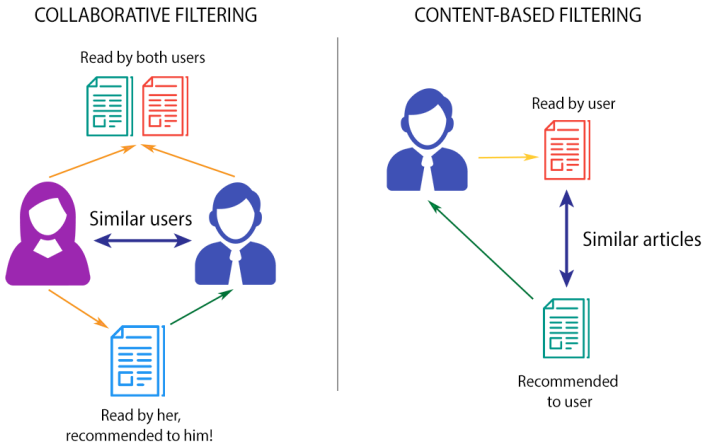

FIGURE 1.1: Content-based and collaborative filtering Recommender systems (Mohamed et al., 2019).

The job of a recommender system is to suggest relevant products to customers, but it is the customer's job to decide *which* suggestions are appropriate for their mood or needs at a given point in time. While meta data may not be needed to predict user preferences, attributes such as genres or other popular tags still hold value to users for assessing recommendations provided to them. Netflix goes to great lengths to populate their entire movie-base with what they term "microtags" in order to provide their users with prescriptive recommendations for every occasion. This requires trained experts to watch and tag every movie on record, which is a time-consuming and costly process (Madrigal, 2014).

1.2 Background to Research

The Netflix Prize contest in 2008 paved the way for large improvements to the predictive ability of collaborative filtering models (Bennett, Lanning, et al., 2007). The contest saw the progression from simpler k-nearest-neighbour models to the use of more advanced latent factor models, achieving a reduction of 10% in the root mean squared error compared to the previous best model (Yehuda Koren, 2009).

The winning submission to the contest used a matrix factorisation model to achieve state-of-the-art performance in predicting user ratings. This approach used matrix decomposition to factorise the ratings matrix into two smaller matrices, as shown in figure 1.2 The sizes of these matrices depend on the number of users and items in the dataset (Yehuda Koren, 2009).

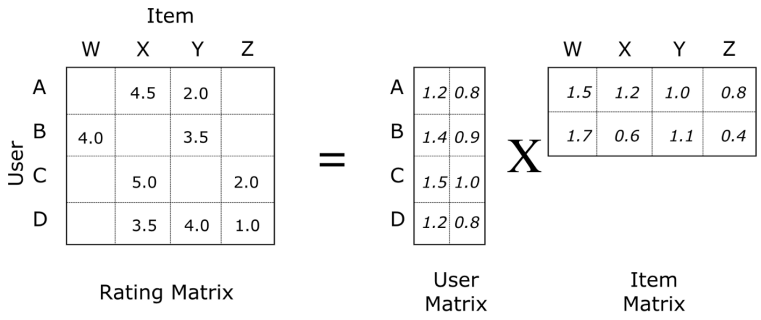

FIGURE 1.2: Matrix factorisation (Liao, 2018)

The values in each of the two matrices are learned through an alternating least squares method such that the cross product of the two matrices produces a matrix that is close to the original ratings matrix, but with all missing ratings filled in. Furthermore, it was shown that these latent features are descriptive of the attributes of the items in the dataset and that similar items will have similar latent feature vectors (Yehuda Koren et al., 2009).

Despite the advances made during the contest, these efforts were somewhat untimely as this was before the mainstream adoption of deep learning. Four years after the close of the Netflix Prize contest "AlexNet", utilising the enhanced computational capacity of GPUs for model training, showcased the predictive ability of neural networks with its record-breaking accuracy in the annual ImageNet image classification contest (Krizhevsky et al., 2012). Since then, the field of deep learning has continued to grow, with neural networks achieving state-of-the-art accuracy on a wide range of problems (Alom et al., 2018).

It has been shown that deep learning can also be applied to collaborative filtering problems. He et al. (2017), used a neural architecture capable of learning the same latent features in users and items as the matrix factorisation methods. The use of a neural network offered improvements on the previous matrix factorisation methods.

1.3 Purpose of the Research

CF is based on the idea that if two users, A and B, both rate the same set of movies similarly, then if user B has seen and enjoyed movies that user A has not seen, it is likely that user A will enjoy them too. While there has been much literature on the subject of predicting user ratings using CF models, I am not aware of any published investigation into the potential use of CF for predicting item metadata.

CF models, by definition, do not make use of metadata for predicting user preferences. Indeed, this is the domain of content-based models. However, metadata such as item genres are still useful to users for assessing recommendations provided to them. Without the complement of metadata users are expected to trust recommendations provided to them, without having some sort of description of the recommended item. While CF has shown its ability to provide recommendations to users

without using content attributes, descriptive features may still be useful for users to make informed decisions on *which* recommendations to choose.

This minor dissertation explores the potential for using collaborative filtering to predict categorical attributes of items. The method, dubbed collaborative genre tagging (CGT), is a transfer learning application of CF that makes use of latent factors for predicting genres of movies, using only explicit user ratings as model inputs.

1.4 Layout of the Minor Dissertation

Chapter 2 provides an overview of the relevant literature on recommender systems. The chapter covers the methodology surrounding the evaluation of recommender systems and outlines the two main types of models, namely content-based and collaborative filtering models. Particular attention is given to collaborative filtering models applied to explicit feedback data, leading to an in-depth discussion of the models that have been developed to perform this task. These models are visited chronologically, starting with early attempts, then moving onto the notable efforts from the Netflix Prize competition, before outlining some of the more recent approaches using deep learning.

Chapter 3 outlines the architecture of the CGT model and lists the hyperparameters that need to be tuned for optimal performance. Additionally, the framework for tuning hyperparameters and assessing model performance is detailed in this chapter.

Chapter 4 provides an overview of the datasets used in this dissertation, including descriptions of their rating scales, summary statistics, and profiles of their distributions in terms of numbers of users, items and ratings.

Chapter 5 covers the results of hyperparameter tuning and model evaluation across all of the datasets listed in Chapter 3.

Conclusions and recommendations for further research are outlined in Chapter 6

Chapter 2

Literature Review

This chapter provides a review of all related works in the field of recommender systems. The chapter first focuses on the background and origin of recommender systems and outlines their main applications. This is then followed by an overview of the methods used to evaluate the accuracy and effectiveness of recommender systems. Then, the two main types of recommender systems are discussed, namely *content-based* and *collaborative filtering* systems. Finally, the chapter is concluded by evaluating the top submissions to the Netflix Prize, a contest that was held in 2008 and was a major contributor to the progression of collaborative filtering algorithms.

2.1 Introduction to Recommender Systems

When trying to decide what movie to watch or what album to listen to, people are presented with a myriad of options that can, oftentimes, be overwhelming. To help people make decisions such as these, inspiration is often taken from the word of mouth of friends, reviews from critics and general surveys. Recommender systems provide an automated means for helping people discover new items (Resnick and Varian, 1997).

The goal of a recommender system can be defined as predicting the response of a user for new items based on historical information, and suggesting novel items for which the user's response is likely to be positive (Desrosiers and Karypis, 2011).

Approaches to Recommender Systems

Recommender systems can be divided into two broad categories. **Content-based** recommender systems make use of meta data for generating recommendations. Examples of item meta data include the genre of a musical artist or song, or the director of a movie. For users, meta data such as demographic information might be used. This meta data is used to match users to items based on matching tags. For example, the recommendations for a user who likes work of a certain author might include other books by the same author or books of the same genre (Di Noia et al., 2012).

There have been a number of successful content-based recommender systems. Many web recommenders use a keyword-based technique to match web pages based on

the frequency of words on the page. In movie and book recommenders, a common approach is to use similarities between the text descriptions to match products. (Lops et al., 2011)

One major obstacle to content-based approaches is the lack of sufficient meta data, which can be time-consuming and costly to collect (Hu et al., 2008).

An alternative approach, known as **collaborative filtering** (CF), uses only the historical interaction behaviour between users and items to make recommendations (Su and Khoshgoftaar, 2009). This is the approach that will be investigated in this dissertation and will be described in more detail below.

2.2 Evaluating Recommender Systems

Before expanding on the intricacies of collaborative filtering, it is first necessary to discuss the methods and metrics that are commonly used to evaluate recommender systems. This is the most subjective aspect of the process, since the success of the system depends on the objectives of its designer. For example, the intent could be to stretch customer spend, or it could be to encourage users to discover new products. In each of these cases, success would be measured using different yard sticks (Herlocker, Konstan, and Riedl, 2004).

There are three main approaches for evaluating the success of a recommender system. The first approach is to evaluate the system offline, without any form of user testing. The other two approaches involve user testing; either using small groups of subject matter experts or more large-scale, online user testing. While each recommender system can be evaluated subjectively based on its ability to meet certain outcomes, there are certain measurable properties that can be used to compare different systems (Shani and Gunawardana, 2011). The intended outcomes of the recommender system will determine whether it can be evaluated offline, or if it will require expert or user group online testing.

2.2.1 Offline Evaluation

Most recommender systems are scored and assessed based on their ability to predict known user ratings. Such models are trained to predict the numerical values of ratings that users will assign to items. The predicted values can be compared to the true ratings using common regression metrics such as mean absolute error (MAE) or root mean squared error (RMSE) (Ge et al., 2010). If we use $\hat{y}_i$ and y_i to denote the predicted and observed user ratings for the ith user-item interaction, and n to represent the total number of user-item interactions, then the accuracy of the recommender system can be measured as

$$\text{MAE} = \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{n} \tag{2.1}$$

or

$$\text{RMSE} = \sqrt{\sum_{i=1}^{n} \frac{(y_i - \hat{y}_i)^2}{n}}. \tag{2.2}$$

In some cases, a recommender system could achieve high levels of accuracy, but still struggle to recommend new items to users. Herlocker, Konstan, and Riedl (2004) used the example of a recommender system that suggests bananas to shoppers in a grocery store: *"Statistically, this recommendation is highly accurate: almost everyone buys bananas. However, everyone who comes to a grocery store to shop has bought bananas in the past, and knows whether or not they want to purchase more. Further, grocery store managers already know that bananas are popular, and have already organised their store so people cannot avoid going past the bananas."*

It should be noted, however, that a recommender system should not *only* recommend unexpected items. As Swearingen and Sinha (2001) showed, non-novel recommendations help to develop trust from users in the system and should be combined with more unexpected suggestions to provide an enjoyable overall user experience.

Serendipity is a metric that has been used to assess both a recommendation's unexpectedness **and** its usefulness (Maksai et al., 2015). Serendipity is more of a conceptual evaluative measure, rather than a fixed formula like RMSE. One approach to capture serendipity numerically has been suggested by Herlocker, Konstan, and Riedl (2004). For recommender systems that produce expected probabilities for a given user for every item in the system, for example, one could divide each probability by the probability score of the average user to produce re-weighted scores. The resulting probabilities would represent a measure of how much more likely a given user is to like items than the average user.

In the case where ratings are not available – that is when only a list of user-item interactions is given – the task of providing recommendations is often transformed into one of providing a set number of items to a user. Using P to denote the set of items that are suggested to the user and O to denote the set of items that the user interacted with, this type of approach can then be evaluated using **precision** and **recall**, computed as

$$\text{precision} = \frac{|P \cup O|}{|P|} \tag{2.3}$$

and

$$\text{recall} = \frac{|P \cup O|}{|O|} \tag{2.4}$$

(Desrosiers and Karypis, 2011).

2.2.2 Online Testing

Testing recommender systems in an online manner, be it user studies or large-scale trials, requires a large enough audience that can be split into test and control groups. Ideally, the two user groups would be identical apart from the fact that the test group would receive suggestions from the recommender system, while the control group

would not. The two groups would then be assessed on their behaviour over a trial period, allowing for the recommender system to be assessed in various areas such as uplift in incremental sales or exploration of the item space (Maksai et al., 2015).

An example of online testing can be found in the work done by Swearingen and Sinha (2001), where users were asked to evaluate a number of internet recommender systems and compare their suggestions to those of their friends. Users were asked to classify which recommendations they received were good, and then to further distinguish between **useful** recommendations they had never seen before and **previously liked** recommendations. It was observed in this study that recommender systems tended to produce more suggestions of completely new items than friends, who tended to recommend items of which users were already aware.

Online testing assesses the extent to which the recommender system changes user behaviour. Indeed, recommender systems exist for the main purpose of suggesting items to users that they are predicted to like, but would otherwise not have known about. The influence of a recommender system cannot be assessed through back-testing. It can only be done through online trials (Desrosiers and Karypis, 2011).

Since each system needs its own independent test group, the size of the audience required increases with the number of recommender systems being compared, which makes online testing infeasible in the context of this minor dissertation. Therefore, all comparisons between recommender systems in this paper will be made using offline metrics.

2.2.3 Netflix Prize

A major event in the progression of recommender system methods, as well as their evaluation metrics, was the Netflix Prize which was run between 2006 and 2009. In 2019, Netflix is a well-known online video subscription service that provides content to subscribers through streaming. However, in 2006, Netflix was an online video subscription service that provided DVD rentals to subscribers via the mail. As part of their service, they encouraged users to rate movies that they watched, producing a dataset of some 1.9 billion ratings from 11.7 million subscribers across 85 thousand titles in under 10 years (Bennett, Lanning, et al., 2007).

This dataset was used by Netflix to create their own recommendation algorithm known as Cinematch. Cinematch used a variant of Pearson's correlation to determine a movie's similarity to all other movies. These similarities between movies were used to provide personalised recommendations to users based on the movies they had rated. The main method used to assess the performance of Cinematch was to compute the RMSE between the predicted and observed user ratings (Bennett, Lanning, et al., 2007).

In 2006, Netflix released a variation of this dataset to the data science community at large with the challenge of improving on the performance achieved by the existing Cinematch algorithm. The dataset provided for this challenge contained over 100 million ratings (and their dates) from over 480 thousand subscribers on almost 18 thousand movies. The data were collected between 1998 and 2005, and comprised a representative sample of all ratings captured by Netflix during this period. The

user ratings were measured on an integer scale from 1 to 5. Three million of the most recent ratings from those same subscribers across the same set of movies was withheld as a competition qualifying set. Half of the qualifying dataset was used to compute the RMSE of submissions on a running public leader board, while the other half, known as the "quiz" subset was used by Netflix to decide who won the competition (Bennett, Lanning, et al., 2007).

As a reference for competition contestants, Netflix reported that the Cinematch algorithm was able to achieve a performance of 0.9514 RMSE on the quiz subset, which was 9.6% lower than simply predicting using the average movie ratings (Bennett, Lanning, et al., 2007).

Figure 2.1 shows the relative distribution of the submissions' performances with respect to that of the Cinematch algorithm. The figure was created in 2007, at which point no team had achieved the goal of a 10% improvement over Cinematch. The numbers in red correspond to submissions in which the same value is used for every prediction, while the "wooden spoon" was the lowest score achieved by any of the entrants.

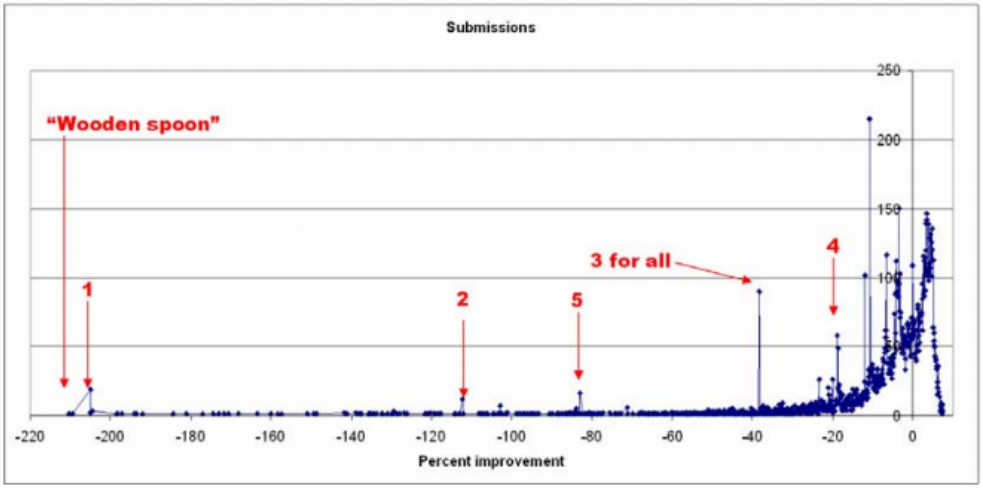

FIGURE 2.1: Netflix prize submissions ordered by improvement over Cinematch (Bennett, Lanning, et al., 2007).

In 2009, the competition reached its conclusion when a team known as "BellKor's Pragmatic Chaos" surpassed the 10% improvement level (Yehuda Koren, 2009).

2.3 Collaborative Filtering

The term "collaborative filtering" was first coined by D. Goldberg et al. (1992), who used it to refer to a system for suggesting relevant emails for a person from a selection of mailing lists. This system incorporated the reactions of other users in the filtering process.

Since then, the term has generally been used to describe a process through which known preferences of users in a group are used to predict the unknown preferences

of other users (Su and Khoshgoftaar, 2009). In most cases, the preferences of users within the group are indicated by a rating or score of some kind; for example, users might assign positive ratings to items they liked, and negative ratings to items they did not. The term *collaborative* refers to the fact that the users of the system improve its performance with each rating that they contribute (K. Goldberg et al., 2001), i.e., as users record more ratings of items, the ability of the system to *filter* accurate recommendations for other users improves.

Collaborative filtering techniques provide an algorithmic method for making recommendations to users based on the preferences of other similar users. The fundamental assumption that underpins CF techniques is that groups of users who have rated a set of items similarly will also rate other items similarly. For example, if user A likes movies m_1, m_2, m_3, and user B likes movies m_1, m_2, m_3, m_4, then user A is also likely to enjoy movie m_4 (Su and Khoshgoftaar, 2009).

One of the advantages of collaborative filtering is that it can be performed without any meta data relating to either the users or the items in the database. Whereas other predictive models make predictions for a response variable using the values of one or more predictor variables, CF models make predictions for ratings using only other ratings (Francesco Ricci and Shapira, 2011).

One issue with content-based approaches is their limited capacity to recommend novel, unexpected items. The system will typically recommend items that match highly to a user's known preferences, i.e. they are limited by the user's tendency for exploration. (Lops et al., 2011)

CF-based models, on the other hand, leverage the collective experiences of all users to recommend interesting new items to users. In contrast to content-based models, models that employ collaborative filtering are known to produce serendipitous recommendations that users would not typically expect (Herlocker, Konstan, and Riedl, 2002).

2.3.1 User Feedback

In collaborative filtering recommender systems, the inputs are provided by people (users) who rate items with which they have interacted (Jawaheer et al., 2010a). The function of a recommender system is to aggregate these ratings from users to make recommendations to other users. In some cases, ratings are provided to a recommender system explicitly by users, but in many other cases these ratings have to be inferred based on user-item interactions (Hu et al., 2008).

Explicit Ratings

Explicit feedback is captured when a user makes the conscious decision to *explicitly* rate an item. These ratings can be binary (e.g. like or dislike), numeric (e.g. 1-10 rating scale) or unstructured (e.g. annotated text). For example, a user can indicate that they like a certain song by giving it a "thumbs up" on a music streaming service (Jawaheer et al., 2010a).

This type of feedback directly captures user preferences toward items; however, it can be scarce. It has also been found that the rate at which users provide explicit feedback decreases over time and, furthermore, that leaving feedback has a negative effect on user behaviour overall (Jawaheer et al., 2010a). This is shown in figure 2.2 where the average percentage of loved tracks over all users is shown to decrease as playcount increases.

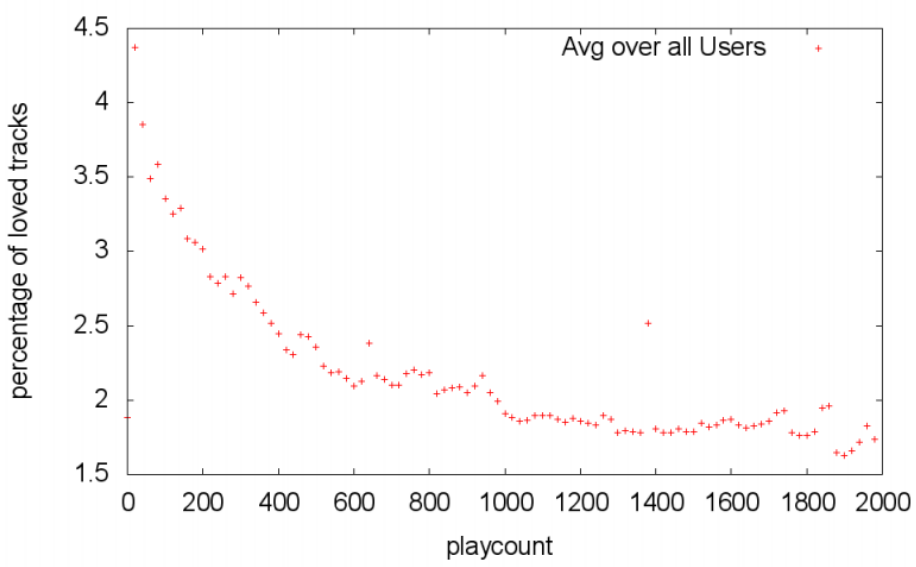

FIGURE 2.2: Percentage of loved tracks as playcount increases (Jawaheer et al., 2010a).

Implicit Feedback

Alternatively, the user might not explicitly rate items they like, in which case a recommender system would need to infer their preferences from their interaction history. Using the example of a music streaming service, many users do not choose to 'like' or rate songs; however, they will tend to listen to songs they like more often than those they do not. Jawaheer et al. (2010a), showed that there is a positive correlation between the number of 'likes' a track receives from users and its total play count. Therefore, it is a viable option to infer positive feedback from user interaction data. Other forms of implicit feedback include purchase history, browsing and search patterns, or mouse movements and click patterns (Yehunda Koren and R. Bell, 2011).

Whether the user feedback is implicit or explicit, the job of the recommender system is the same - to aggregate this feedback from users to aid others in discovering new items of interest. It has been shown that using a combination of explicit and implicit feedback can yield improved recommendation accuracy (Jawaheer et al., 2010b); however, it is the subtleties around *how* recommender systems identify all the signals from the available user feedback data that remains the greatest area of focus for improving accuracy.

2.3.2 Neighbourhood-based Recommender Models

There are two main techniques used in collaborative filtering recommender systems: *neighbourhood-based* and *latent factor* models (Yehunda Koren and R. Bell, 2011). As the name suggests, neighbourhood models employ a nearest-neighbours approach to recommending items to users.

Neighbourhood-based approaches were some of the most prevalent in the early years of recommender systems at the end of the 90s. These methods make use of similarity measures to select subsets – or neighbourhoods – of users or items (Hu et al., 2008). These neighbourhoods are used to produce weighted average ratings to generate predictions for users (Herlocker, Konstan, Borchers, et al., 1999).

User-based Similarity

The first types of neighbourhood models estimated unknown ratings using a user-oriented approach in which predictions were based on the known ratings of like-minded users (Hu et al., 2008).

To make a prediction for a given user a for item i, the first step is to measure their similarities to all other users of the system. Similarity between users a and b can be calculated using the Pearson correlation coefficient, $w_{a,b}$, computed as

$$w_{a,b} = \frac{\sum_{i=1}^{m}[(r_{a,i} - \bar{r}_a)(r_{b,i} - \bar{r}_b)]}{\sqrt{\sum_{i=1}^{m}(r_{a,i} - \bar{r}_a)^2 \sum_{i=1}^{m}(r_{b,i} - \bar{r}_b)^2}}, \tag{2.5}$$

where $r_{j,i}$ is the rating of item i by user j, $\bar{r}_j$ is user j's average rating and m is the number of items rated by both users.

This similarity can then be used to predict the rating for the active user a for item i as

$$P_{a,i} = \bar{r}_a + \frac{\sum_{b \in N(a)} w_{a,b}(r_{b,i} - \bar{r}_b)}{\sum_{b \in N(a)} w_{a,b}}, \tag{2.6}$$

where $N(a)$ is the set of user a's neighbours (Herlocker, Konstan, and Riedl, 2002).

This approach was modified by Shardanand and Maes (1995), who used a constrained Pearson correlation coefficient, calculated as

$$w_{a,b} = \frac{\sum_{i=1}^{m}[(r_{a,i} - 4)(r_{b,i} - 4)]}{\sqrt{\sum_{i=1}^{m}(r_{a,i} - 4)^2 \sum_{i=1}^{m}(r_{b,i} - 4)^2}}. \tag{2.7}$$

The value 4 was used as it was the midpoint of the seven-point rating scale on which their music recommendation system, dubbed "The Ringo", was based. The number of neighbours was limited by applying a correlation threshold, where only users

above the threshold could be considered for membership to the neighbourhood. Increasing the magnitude of this threshold resulted in greater accuracy, but limits the number of items for which the system can make recommendations (Herlocker, Konstan, and Riedl, 2002).

The most common alternative similarity measure to the Pearson correlation coefficient is the cosine similarity, which is calculated between users a and b as

$$\cos(r_a, r_b) = \frac{(r_a \cdot r_b)}{||r_a|| \times ||r_b||}, \tag{2.8}$$

where r_a and r_b represent the ratings vectors for each user, considering only items which have been rated by both users, $r_a \cdot r_b$ is the dot product of the two rating vectors and $||r_a|| \times ||r_b||$ is the product of their magnitudes (Amatriain et al., 2011).

This measure can then be used to make a prediction for user a and item i by selecting the n most similar neighbours and taking an average of their rating of the item, weighted by their cosine similarity.

Though the Pearson correlation and cosine similarity are the most popular measures used, it is possible to use any other common distance measure, such as the Euclidean or Minkowski distance (Amatriain et al., 2011).

Item-based Similarity

An alternative to neighbourhood models is an item-oriented approach, in which the known ratings of a user for similar items are used to make a prediction for a new item (Hu et al., 2008). When making a prediction for user a and item i, all known ratings of the user are used as the neighbourhood. The predicted rating is taken as the average of all known ratings, weighted by their corresponding item similarity with i. The similarity measure for this neighbourhood can be any of the same measures as used in the user-based approach, only in this case the similarities are measured between items instead of users (Desrosiers and Karypis, 2011).

2.3.3 Latent Factor Recommender Models

Latent factor models attempt to characterise both users and items using anywhere in the region of between 20 to 100 factors. These latent factors capture behavioural patterns between users and items in the ratings space (Yehuda Koren et al., 2009).

The most common version of latent factor model is the matrix factorisation (MF) method, used by R. M. Bell et al. (2008) to win the Netflix Prize. In this approach, the user/item interaction data is represented as a matrix which is then factorised as the product of two lower rank matrices.

Figure 2.3 shows the matrix representation of the explicit interactions between users and items. All blue values are known ratings submitted by users, while the white zeroes represent unknown values which need to be predicted. Each row thus holds the ratings made by a particular user and each column the ratings received by a

single item. The rating made by user i of item j can be found in the j^{th} column of row i.

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	0	3	0	3	0
User 2	4	0	0	2	0
User 3	0	0	3	0	0
User 4	3	0	4	0	3
User 5	4	3	0	4	0

FIGURE 2.3: Interactions between users and items represented as a matrix (Bailey, 2016).

If the ratings matrix consists of m rows and n columns, then any two matrices of dimensions m x d and n x d respectively, when cross multiplied together, would result in a matrix of the same dimensions as the original ratings matrix. The common dimension, d, is a hyperparameter that represents the number of latent factors, where $d \leq \min(m, n)$ (Bailey, 2016).

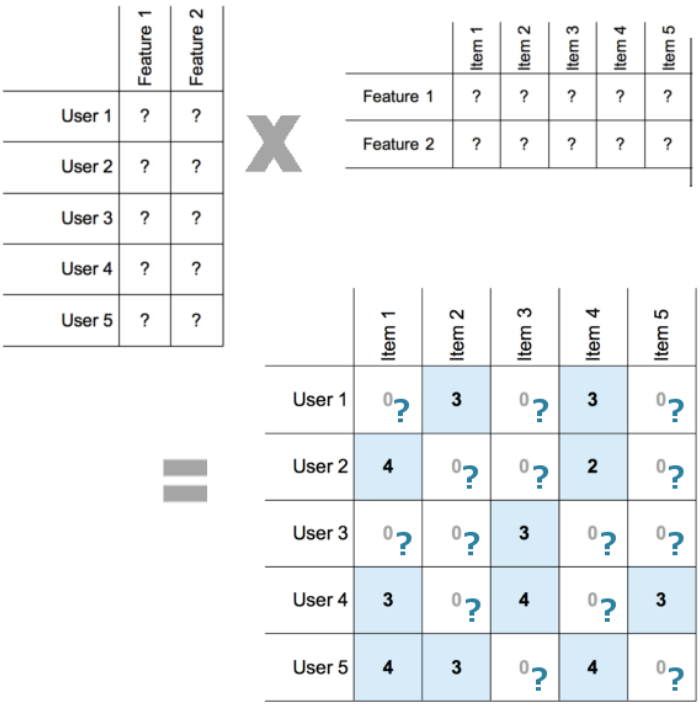

FIGURE 2.4: Ratings matrix factorised as product of two latent factor matrices (Bailey, 2016).

To get the rating for the first user of the first movie, one takes the dot product of the first row of the user matrix with the first column of the item matrix. More generally, the rating made by user u of item i is calculated as

$$\hat{r}_{ui} = p_u \cdot q_i^T, \tag{2.9}$$

where p_u is the u^{th} row of the user latent factor matrix and q_i is the i^{th} column of the item latent factor matrix.

Optimisation Techniques

The two latent factor matrices that are used in the MF approach need to be *learnt*. Their values can be solved through either stochastic gradient descent or alternating least squares (ALS), optimising the minimal distance between the original ratings matrix and the reconstructed matrix. The ALS optimisation technique was the favoured approach taken by R. M. Bell et al. (2008). This technique holds the values in one factor matrix fixed and then uses least squares to solve the values in the other and *vice versa*. The converged solution provides the closest reconstruction of the original ratings matrix, with the missing values filled in (Yehuda Koren et al., 2009).

The two resulting latent factor matrices consist of vectors that describe the users and the items respectively. For each user and item, there will be d latent factors that describe their characteristics. If the items are movies, for example, the latent factors might represent known dimensions, like genres such as comedy or romance. They might also represent deeper dimensions, like dialogue style or the amount of satire. The latent factors might also represent completely uninterpretable dimensions too. In the case of users, the latent factors represent their partiality to the corresponding item latent factor (Yehuda Koren et al., 2009).

Biases

The MF approach may be extended to include biases that compensate for systematic tendencies across both users and items. For example, some users tend to rate all items higher or lower than average, and some items might be generally over-rated by all users (Yehuda Koren et al., 2009).

When incorporating biases, equation 2.9 becomes

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u \cdot q_i^T, \tag{2.10}$$

where μ is the overall average rating and b_u and b_i are the biases of user u and item i.

Netflix Prize Winning Submission

The winning submission to the Netflix Prize made use of an ensemble model, achieving an RMSE of 0.8712 on the test set using a combined total of over 500 models, including matrix factorisation methods, restricted Boltzmann machines and nearest neighbour models. Despite this extensive range of models used in their winning ensemble, the team members noted that the most accurate standalone approach was matrix factorisation (Yehuda Koren, 2009).

Aside from the choice of algorithm(s), the winning team paid special attention to what they termed "baseline predictors". The purpose of these was to "encapsulate those effects which do not involve user-item interaction." Predicted ratings were decomposed into multiple parts in a similar manner to how equation 2.10 represents predicted ratings. In this equation, the first three terms can be considered baseline features, while the final term represents the specific user-item interaction. For example when predicting how a user, Alice, might rate the movie Inception, the prediction would first consider the average rating of the movie, as well as the average of all of Alice's ratings as the baseline predictors. Then, the specific user-item interaction based on the latent features is used to adjust the baseline prediction. In addition to user and item biases, temporal aspects were used to adjust the baseline. Factors such as time since a user's first rating and how many ratings a user made on a particular day were found to have a significant impact on user ratings. To capture temporal effects, the user and item biases were each calculated as a function of time.

When accounting for temporal components, equation 2.10 becomes

$$\hat{r}_{ui} = \mu + b_u(t_{ui}) + b_i(t_{ui}) + p_u \cdot q_i^T, \tag{2.11}$$

where b_u and b_i are functions that change over time (Yehuda Koren, 2009).

2.3.4 Uses of Deep Learning in Collaborative Filtering

Tremendous effort was required from all of the contestants of the Netflix Prize competition to push the boundaries of predictive ability in CF models. Ensemble models and intricate modelling of baseline predictors were used to eek out marginally better models; however, it was shown that the improvement in accuracy was largely due to the use of latent factors. Then, it was through the use of matrix decomposition that these latent factors were learnt; however neural networks can perform the same task through the use of embedding layers, with the added ability to learn non-linear patterns in the data.

The first notable use of deep learning for collaborative filtering was in 2015, when Sedhain et al. used an autoencoder framework for their *AutoRec* model. They opted for an item-based approach in which the input to the neural network is a partially observed vector containing all user ratings for a given item. Figure 2.5 shows the architecture of this model.

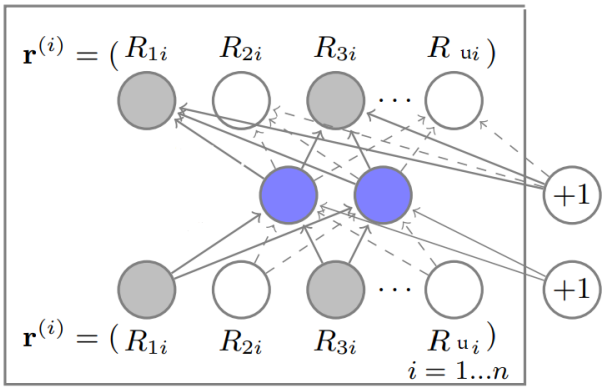

FIGURE 2.5: AutoRec uses an autoencoder architecture (Sedhain et al., 2015)

It can be seen in figure 2.5 that the input dimension of the AutoRec model is equal to the number of users, u. There are n copies of the network, one for each item. These copies are linked so that there is shared learning of weights corresponding to the same user. Since each input is sparse, only the parameters associated with the observed ratings are updated during backpropagation. This means that the trained model takes as input a sparse item ratings vector, representing all available user ratings for that item, and outputs a complete ratings vector containing all predicted user ratings for the item (Sedhain et al., 2015).

Evaluation of this model was done using the MovieLens 1M, 10M and Netflix datasets using 5 separate random 90%-10% train-test splits, with 10% of the training data being used for hyperparameter tuning. Test accuracy was taken as the average of these 5 random splits and users or items in the test set without training observations were given a default rating of 3 stars. They reported test accuracies of 0.831, 0.782 and 0.823 on the MovieLens 1M, 10M and Netflix datasets respectively (Sedhain et al., 2015).

Figure 2.5 shows the item-based AutoRec model. Sedhain et al. (2015) experimented with a user-based model, taking n-dimensional user rating vectors as inputs to m copies of the network. However, they found this model achieved reduced accuracy, likely due to the fact that there are more items per movie than per user in the datasets they used.

A neural architecture was used in the "Neural Collaborative Filtering" (NCF) model created by He et al. (2017). This particular effort focused on the CF scenario where only implicit feedback is used; however, the architecture used is equally suited to using explicit feedback. Figure 2.6 shows the architecture of a NCF model, with X hidden layers and k latent factors.

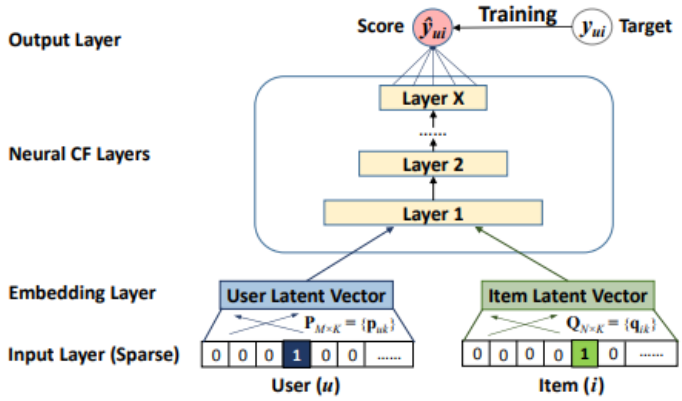

FIGURE 2.6: Neural Collaborative Filtering architecture (He et al., 2017)

The NCF model generalises MF through the use of an embedding layer, which combines both user and item latent factors. The embedding layer serves the same purpose as the latent factor matrices in the MF approach, while the hidden layers allow for non-linear combinations of these latent factors when predicting ratings.

Although the results achieved by Sedhain et al. represented an improvement of more than 5% on the winning submission to the Netflix Prize, it should be noted that their method of evaluation on the Netflix set was not equivalent to the true holdout set used in the competition.

2.4 Transfer Learning

The literature that has been reviewed in the previous sections of this chapter has covered the topic of recommender systems, with specific focus on tasks related to predicting explicit ratings using CF. While questions have been raised regarding the reproducibility of some results that claim to have improved on the state of the art, (Dacrema et al., 2019), there can be no questioning the effectiveness of latent factor models for learning otherwise-unknown features of the user and item sets.

The aim of this minor dissertation is to predict genre labels of items, such as movies or books, that have been rated explicitly by users. These genre labels are predicted using latent features learnt through the task of predicting ratings. These latent features can then be re-used in a second model to predict genres. This process is named collaborative genre tagging (CGT) This process of adapting the learnings of one model in order to train another places CGT under the domain of transfer learning.

Ruder (2019) describes transfer learning as *"a means to extract knowledge from a source setting and apply it to a different target setting"*. Its use has allowed for image classifiers to be created by fine tuning convolutional neural networks, achieving state-of-the-art performance in a wide variety of computer vision tasks while using significantly less data for training (Sharif Razavian et al., 2014). Any machine learning scenario

in which knowledge is gained from one task and re-used for another is considered transfer learning, as can be seen in figure 2.7.

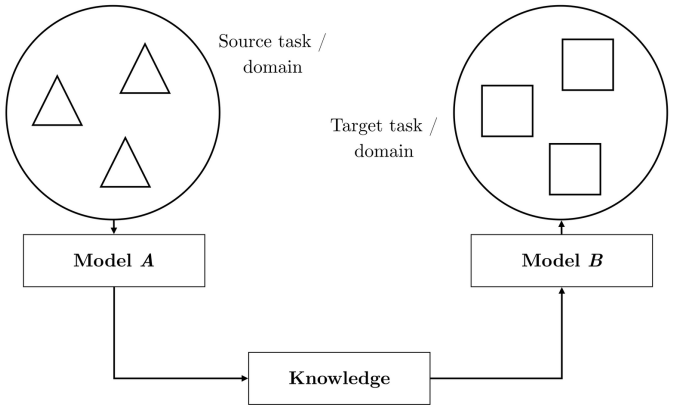

FIGURE 2.7: The transfer learning scenario (Ruder, 2019)

Collobert et al., 2011 demonstrated the effectiveness of using word embeddings for a variety of natural language processing tasks. Word embeddings can be learnt through trainable layers in a neural network that extract a set of features for each word. For each word, a set of features is stored in a lookup table. These features can be trained using unlabelled datasets, such as the entire corpus of the English Wikipedia. By removing words from sentences in the corpus and then training a language model to predict the missing word, word embeddings can be learnt such that they capture the syntactic and semantic information associated with the words themselves. These word embeddings can then be re-used in a sequence model to perform transfer learning on tasks such as sentiment analysis or part-of-speech tagging.

Chapter 3

Methodology

This chapter details the methods used in this minor dissertation. Specifically, the focus is on collaborative genre tagging (CGT), a sequential transfer learning approach that applies collaborative filtering to predict genres of items such as movies or books. First, CGT is outlined and compared to more common uses of collaborative filtering on datasets such as MovieLens and Goodbooks. Then, the model architecture of CGT is detailed, along with the key hyperparameters. Finally, the framework for hyperparameter tuning and model evaluation is described.

3.1 Collaborative Genre Tagging

Most of the previous work on collaborative filtering (CF) tackled the problem of predicting user preferences using their interaction history. The most successful methods all involve the use of latent factor models that are capable of learning otherwise hidden properties of the users and items in CF datasets. Recently, this approach has been adapted from matrix factorisation models to neural architectures that capture latent factors in embedding layers, while allowing for non-linearity in fully connected hidden layers.

The general idea behind CF is that users with similar preferences will provide similar ratings for a given movie. The idea behind CGT is that similar movies will be rated similarly by users who share a preference for that type of movie. This concept is illustrated in figure 3.1. Just as user rating patterns have been used to infer user preferences, they can also be used to infer information about the items being rated by users.

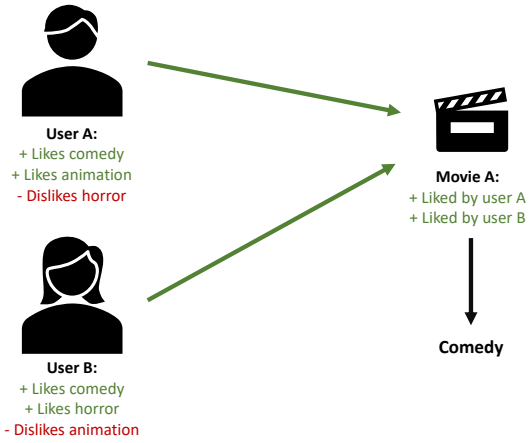

FIGURE 3.1: CGT uses patterns in user ratings to predict genres.

CGT involves two distinct learning tasks:

1. Learn user and item latent factors through the task of predicting user-item ratings, and

2. Learn to decode item latent factors to genres.

The neural architectures associated with each of the tasks listed above are shown in figure 3.2. The first model, referred to as the rating model, uses latent factors to predict user ratings and is similar in design to the Neural Collaborative Filtering model used by He et al. (the architecture of which is shown in figure 2.6). The second model, called the genre model, uses the embedding layer of the first network as the input to a hidden layer that maps item latent factors to genres.

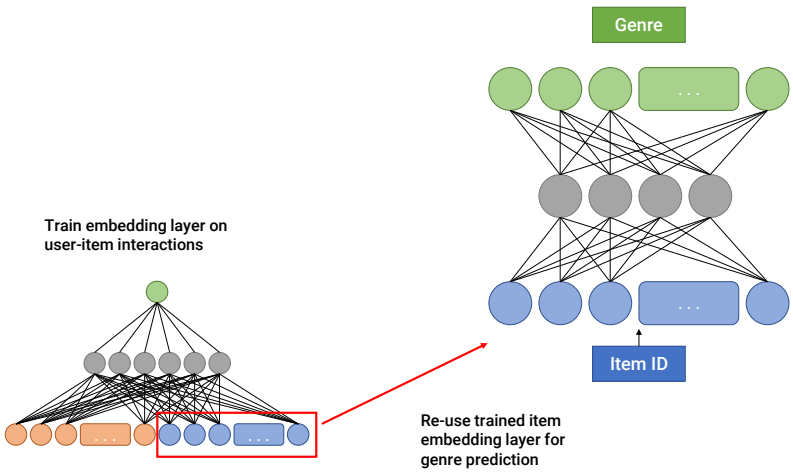

FIGURE 3.2: CGT model consists of two neural networks which share a common item embedding layer.

The rating model takes a pair of user and item IDs that are each mapped to embedding layers (represented by the orange and blue nodes in figure 3.2) and then concatenated together. The output of this model is a predicted rating for a given user-item pair. The purpose of this model is to train the embedding layer to produce latent factors associated with every item in the ratings dataset.

The genre model takes only an item ID as input, which maps to the same item embedding layer used by the rating model. The output is a predicted genre label.

The models are trained sequentially, with the shared embedding layer being frozen when training the second model. This approach takes its inspiration from the language models used in text sentiment analysis that are trained before the sentiment classification model to produce stronger word embeddings (Howard and Ruder, 2018).

3.2 Rating Model

The rating model is the base of the CGT model. The input to this model is a user-item ID pair, and the output is an explicit rating. The IDs in the input pair are connected to an embedding layer of $k \times 2$ length, where k is the number of latent factors for both users and items. The embedding layer then feeds to a fully connected layer and finally into a single output node that contains the predicted rating.

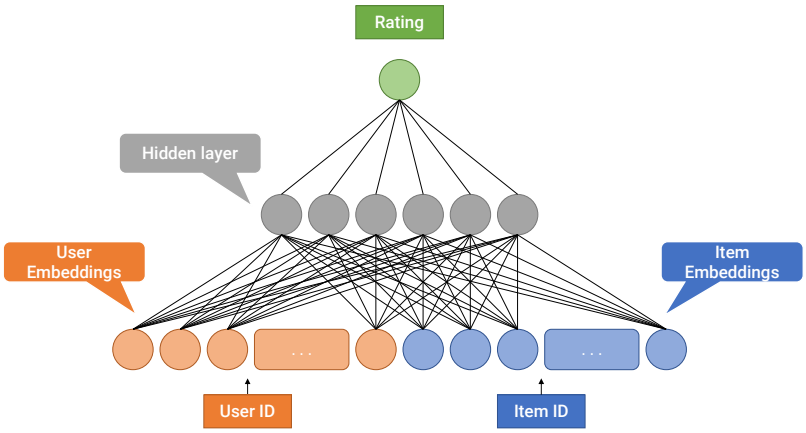

FIGURE 3.3: Rating model uses embeddings to capture latent factors of items and users

3.2.1 Embedding Layer

The first layer of the rating model is a concatenated embedding layer. This layer is comprised of two separate embedding matrices that are essentially the same as the latent factor matrices used in the matrix factorisation method popularised by Yehuda Koren et al. (2009). These embedding matrices hold the latent factors of all

users and items respectively. The user matrix is of dimension $m \times k$ while the item matrix is $n \times k$, where m and n are the number of distinct users and items respectively, and k is the number of latent factors.

Figure 3.4 illustrates the combination of latent factor matrices to form the concatenated embedding layer. The embedding layer takes a user-item ID pair as its input. Each of these IDs is passed to its respective embedding matrix (step 2 in figure 3.4) as a lookup to obtain two k-dimensional latent factor vectors. These two latent factor vectors are then concatenated together to form the first layer of the neural network. Since each latent factor vector is of length k, the concatenated first layer is of dimension $k \times 2$. The size of k is a hyperparameter that will need to be tuned.

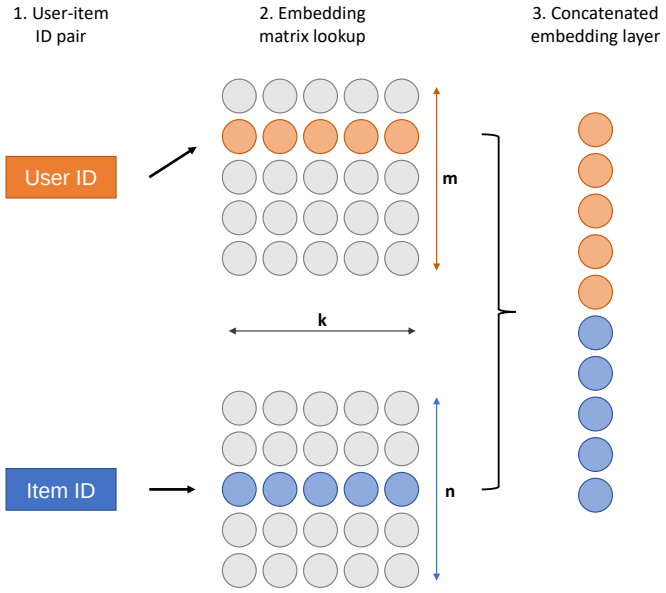

FIGURE 3.4: Embedding layer is created as the concatenation of two separate embedding matrix lookups.

The values of the latent factor vectors in each embedding matrix are randomly initialised and then learned during the process of training the rating model.

3.2.2 Rating Hidden Layer

The concatenated embedding layer feeds forward into the fully connected hidden layer of the model with h nodes. This hidden layer allows for the model to learn non-linear patterns in the concatenated latent factor vector through the addition of non-linear activation functions. The activation function can be any mathematical function used to transform the linear combination of inputs to a node. The use of non-linear activation functions distinguishes the CGT rating model from vanilla matrix factorisation. A popular activation function is the rectified linear unit (ReLU) defined as $f(x) = \max(0, x)$. ReLU is fast to compute and has shown success in a number of neural network applications (Nair and Hinton, 2010; Krizhevsky et al., 2012).

During training of the model, dropout may also be added to this layer as a means of regularization. This technique ignores a randomly chosen proportion of nodes at each training step; the proportion of ignored nodes is known as the dropout rate. This has been shown to prevent overfitting in neural networks (Srivastava et al., 2014). The number of hidden layers, the number of nodes in each layer, the choice of activation function and the dropout rate are all tunable hyperparameters. Figure 3.5 shows the overall architecture of the rating model.

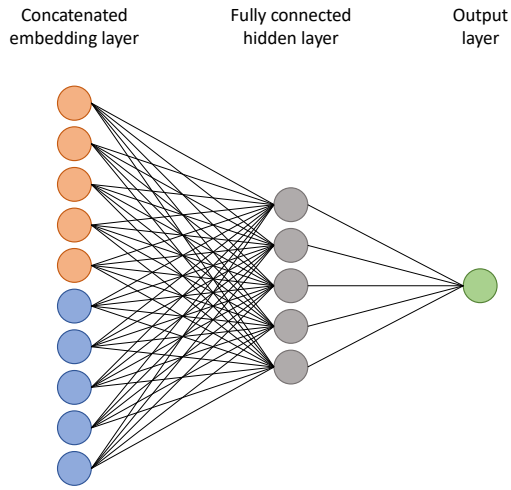

FIGURE 3.5: Hidden layer is fully connected to concatenated embedding layer and feeds into the output node.

3.2.3 Rating Output Layer

As discussed in section 3.2, the model takes two inputs: a user and an item ID. The output of the model is a predicted rating of the input movie by the input user.

The network as it is shown in figure 3.5 does not include any adjustments for user or item biases of any kind. Yehuda Koren et al. (2009) stated that *"much of the observed variation in rating values is due to effects associated with either users or items, known as biases or intercepts, independent of any interactions."* To handle biases inherent in rating data, they adjusted their matrix factorisation model as described in equation 2.10.

A similar approach has been taken in the CGT model, which adjusts the model output through the addition of the global mean rating, μ, and a specific user-item baseline prediction, b_{ui} for user u and item i. This adjustment is illustrated in figure 3.6.

To calculate the baseline prediction for any user-item combination, one needs to compute the average biases for both the user and the item. The bias for user u, b_u, is calculated as

$$b_u = \frac{\sum_{i=1}^{n_u}(r_{ui} - \mu_i)}{n_u}, \tag{3.1}$$

which is the average amount by which user u's rating differs from the average rating of an item, μ_i, taken across all n_u items rated by that user.

Similarly, the bias for an item, i, is

$$b_i = \frac{\sum_{k=1}^{n_i} r_{ki}}{n_i} - \mu, \tag{3.2}$$

which is the difference between the average rating of item i, over all n_i ratings of that item, and the global mean rating, μ.

The baseline prediction b_{ui} is then calculated as the average between the bias of user u and item i, defined as

$$b_{ui} = \frac{b_u + b_i}{2}. \tag{3.3}$$

The inclusion of b_{ui} and μ allows the model to decompose a rating into a portion that is attributable to biases and a portion that is attributable to interactions. The latter is captured in the embedding and hidden layers. Figure 3.6 illustrates the addition of baseline predictors to the CGT rating model.

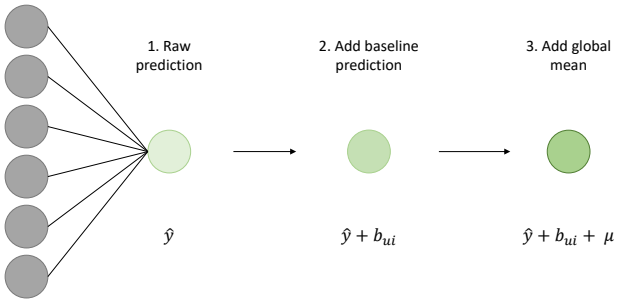

FIGURE 3.6: Model output is adjusted through the addition of baseline predictors.

The baseline predictors are not learnt during model training. Instead, the values of b and μ are calculated before model training using only the ratings from the training set. Users or items that appear in the holdout evaluation set but not the training set are assigned a bias of μ.

3.3 Genre Model

The genre model is the second component of the CGT model. It re-uses the trained item embedding layer as its input layer and attempts to learn to decode latent factors

to genres that are commonly used by users to describe items. The input layer has dimension k, following by a hidden layer with j nodes. Figure 3.7 illustrates the architecture of the genre model.

In this minor dissertation, the model was trained as a one-versus-all binary classifier, using only one output node with a sigmoid activation function used to predict the probability that an item belongs to one specific genre, such as drama. The sigmoid activation function is shown in blue in figure 3.9.

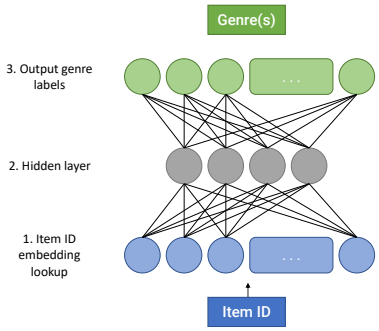

FIGURE 3.7: Genre prediction model re-uses the item embedding layer from the base ratings prediction model

Alternatively, the output layer can support multi-label classification, in which case there will be as many nodes as the total number of genre labels, g, with a softmax activation function to predict the probability that an item belongs to each of the g genres.

3.4 Tuning and Evaluation Framework

The standard approach to hyperparameter tuning of rating models in the past has been to use 5-fold cross validation (CV) on 90% of the available data, with the remaining 10% used as a true holdout set for model evaluation (Sedhain et al., 2015). Both the rating and genre model were tuned together, using 5-fold CV on 90% of the sample, with 10% used for testing. Figure 3.8 illustrates the splitting of the dataset for training, validation and testing.

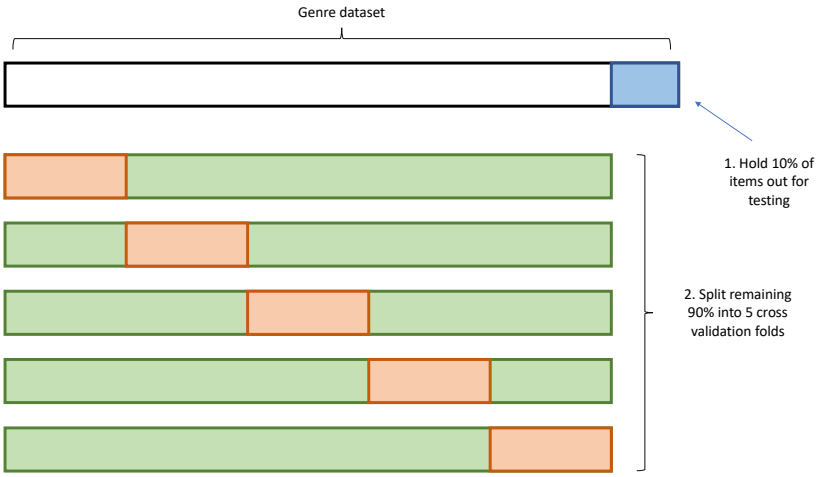

FIGURE 3.8: 10% of the sample (blue) was removed from the dataset *before* performing cross validation. In each fold, the model was trained on 80% (green) and validated on the remaining 20% (orange).

For a given configuration of the model, 5 separate instances of training need to be performed, using a different fold each time. The model is then evaluated by taking the average holdout performance across the 5 folds. Training in this way allows for a more robust measure of performance than a single 80/20 split, while enabling the use of all available data for training.

3.4.1 Hyperparameter Tuning

Tuning the hyperparameters of the two models was done using a grid search over many different combinations of the hyperparameters. For each combination of hyperparameters, the two models were trained sequentially. The rating model was trained on all available rating data, before the genre model was trained and assessed on 5 CV splits. The performance of each configuration was taken as the average validation fold loss of the genre model. The hyperparameters chosen in the rating model were those that led to the best performance in the subsequent genre model. In this way, the rating model was tuned with the objective of maximising the descriptiveness of the embedding layer, rather than the ability to predict user ratings.

The order of operations for the grid search is outlined below:

1. For every given combination of hyperparameters:
   (a) Train rating model on all rating data
   (b) Freeze item embedding layer
   (c) Create five sequential 80/20 CV folds (as shown in figure 3.8)
   (d) Train and assess genre model on each CV fold

(e) Record performance as average validation loss across all folds

2. Choose best performing combination of hyperparameters

3.4.2 Summary of Rating Model Hyperparameters

Table 3.1 summarises the tunable hyperparameters in the rating model. These hyperparameters were tuned and evaluated with respect to their influence on the accuracy of the final genre prediction.

Symbol	Description	Type	Range
k	Number of user and item latent factors	Discrete	(1, ∞]
$l1$	Number of hidden layers	Discrete	(0, ∞]
h	Number of nodes in hidden layer	Discrete	(0, ∞]
$dr1$	Dropout rate in hidden layer	Continuous	(0, 1)
$f1$	Activation function in hidden layer	n/a	n/a

TABLE 3.1: Summary of tunable hyperparameters in CGT rating model, in the case of 0 hidden nodes, the model reproduces matrix factorisation.

The range of values that were tested for k is defined by $k \in \{200, 500, 1000, 1500, 2000\}$. The rating model was tested with no hidden layer (i.e. $l1$ equal to zero) and with one hidden layer ($l1 = 1$), the number of hidden nodes defined by $h \in \{0, 50, 100, 200, 500\}$. In the case where $l1 = 0$, the rating model did not contain a hidden layer, and thus reproduced simple matrix factorisation. Dropout rates in the range $dr1 \in \{0.0, 0.1, 0.15, 0.2, 0.25, 0.3\}$ were tested, while four different activation functions were tested, namely Rectified Linear Unit (ReLU), Softplus, Hyperbolic tangent (tanh) and Linear. Figure 3.9 shows a plot of these activation functions. The linear function (not shown in figure 3.9) is simply the straight line represented by $f(x) = x$, i.e. no transformation is applied.

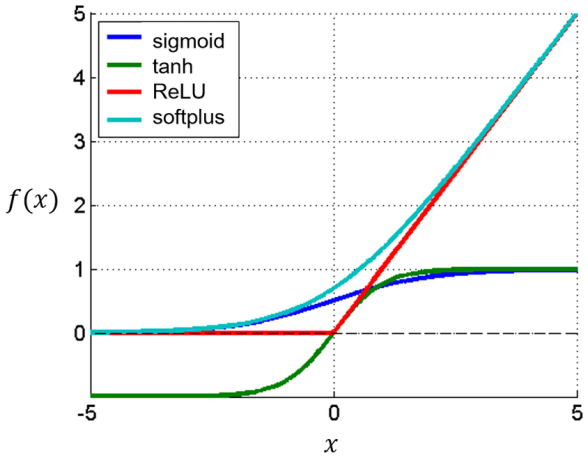

FIGURE 3.9: Plot of the sigmoid function, hyperbolic tangent, rectified linear unit layer, and the softplus function. It can be seen that the softplus function is the smooth version of the rectified linear unit layer (Musiol, 2016).

The hyperparameter values were chosen to minimise the mean squared error (MSE) on the training set.

3.4.3 Summary of Genre Model Hyperparameters

Table 3.2 summarises the tunable hyperparameters in the genre model. k, the number of latent factors, will be the same as the rating model as the two models share this layer.

Symbol	Description	Type	Range
$l2$	Number of hidden layers	Discrete	$(0, \infty]$
j	Number of nodes in hidden layer	Discrete	$(0, \infty]$
$dr2$	Dropout rate in hidden layer	Continuous	$(0, 1)$
$f2$	Activation function in hidden layer	n/a	n/a

TABLE 3.2: Summary of tunable hyperparameters in CGT genre model.

The genre model was tested with just one hidden layer ($l2 = 1$), with the number of hidden nodes defined by $j \in \{50, 100, 200, 500\}$. The same dropout rates and activation functions tested for the rating model were tested for the genre model.

3.4.4 Genre Model Loss Function

The log loss function was used to train the weights of the genre model. This loss function is defined as

$$\text{Log Loss} = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i), \tag{3.4}$$

where y_i is the true label (e.g. 1 for drama, 0 for *not* drama) of item i and $\hat{y}_i$ is the predicted probability that $y_i = 1$, for all n items. Using the log loss, as opposed to simply taking the squared difference between $\hat{y}_i$ and y, imposes a larger penalty on predictions close to 0.5 and therefore causes the model to "pick a side" when making predictions (Shreedhar and Chakraborty, 2019).

A threshold is used to convert probabilities into predicted genre labels, whereby all probabilities above the threshold are given positive labels and all below are given negative labels. In this minor dissertation, a threshold of 0.5 was used, but this can be adjusted to optimise for fewer false positives, depending on the use case.

The results of hyperparameter tuning and evaluation are detailed in chapter 5. This includes an overview of the range of hyperparameter values that were tested as well as the configurations of hyperparameters.

3.5 Software

The code for CGT is publicly available on GitHub at the URL: github.com/James-Leslie/deep-collaborative-filtering. All code is written in Python 3.7, with TensorFlow 2.0 as the deep learning framework. All code uses a fixed random seed for weight initialisation. The results provided in the next chapter are therefore reproducible.

Chapter 4

Data Description

This chapter provides an overview of the four datasets used in this minor dissertation, all of which contain explicit ratings made by users across various item sets.

4.1 Datasets

The MovieLens (ML) datasets are used frequently in CF research efforts. GroupLens research at the University of Minnesota is responsible for maintaining these data. Three stable ML datasets were used in this project, each containing a different number of ratings: ML 100k (100 thousand ratings), ML 1M (1 million ratings) and ML 10M (10 million ratings). The first two datasets use a 5 star, integer-only rating scale, while the 10M dataset contains ratings from 0.5 – 5 stars, in increments of a half (Harper and Konstan, 2016).

The Goodbooks-10k dataset contains 10 thousand books, with a total of just under 6 million ratings made by over 50 thousand users (Zajac, 2017).

All of these datasets were compiled in the United States of America. I am not aware of any suitable South African datasets.

4.2 Number of Users, Items and Ratings

Each dataset has a different profile in terms of the ratios between numbers of users, items and ratings. The **density** is calculated as the number of ratings provided as a proportion of the total number of possible user-item interactions. ML 100k – the dataset with the fewest ratings – has 943 unique users and 1 682 unique movie titles. This would allow a total of 1 586 126 possible user-movie ratings. Since there are only 100 000 ratings in this dataset, its density is thus $100000/1586126 = 6.30\%$. Table 4.1 provides a summary of the four datasets.

Name	Rating Scale	Users	Items	Ratings	Density
ML 100k	1–5, stars	943	1 682	100 000	6.30%
ML 1M	1–5, stars	6 040	3 706	1 000 209	4.47%
ML 10M	0.5–5, half-stars	69 878	10 681	10 000 054	1.34%
Goodbooks-10k	1–5, stars	53 424	10 000	5 976 479	1.12%

TABLE 4.1: Summary of the datasets used in this project

ML 100k, 1M and the Goodbooks-10k sets all use an integer rating scale between 1 and 5 stars. The ML 10M set also uses a 5 star scale, but allows half star ratings.

In terms of size, ML 100k is the smallest with 100 thousand ratings, while the other sets all range between 1 and 10 million ratings. ML 10M contains the most items, albeit only by a slight margin over Goodbooks-10k.

4.3 Distribution of Ratings

Table 4.2 provides a summary of the distribution of ratings across users and items in each of the datasets.

All three MovieLens datasets have a minimum of 20 ratings per user, while some of the movies have only 1 rating.

The Goodbooks-10k dataset is similar to the MovieLens datasets in terms of its distribution across items; however, every book has at least 8 ratings as opposed to the 1 rating minimum of the MovieLens sets.

Name	Min User	Max User	Avg User	Min Item	Max Item	Avg Item
ML 100k	20	737	106	1	583	60
ML 1M	20	2 314	166	1	3 428	270
ML 10M	20	7 359	143	1	34 864	937
Goodbooks-10k	19	200	112	8	22 806	598

TABLE 4.2: Summary of the distribution of ratings in each dataset

4.3.1 Number of Ratings per User

All three MovieLens datasets have a similar right-tailed distribution of the number of ratings made per user. The majority of users have made a small number of ratings – between 1 and 100 – with a small number of users having rated a very high number of movies.

The Goodbooks-10k dataset user ratings follow a bell-shaped distribution, with the majority of users having rated between 75 and 130 books. No user has rated fewer than 19, or more than 200, books.

The histograms in figures 4.1 and 4.2 show the distributions of the number of ratings made per user in ML 10M and Goodbooks-10k. MovieLens 1M and 100k have similar shapes, albeit each with fewer users than the ML 10M shown in figure 4.1.

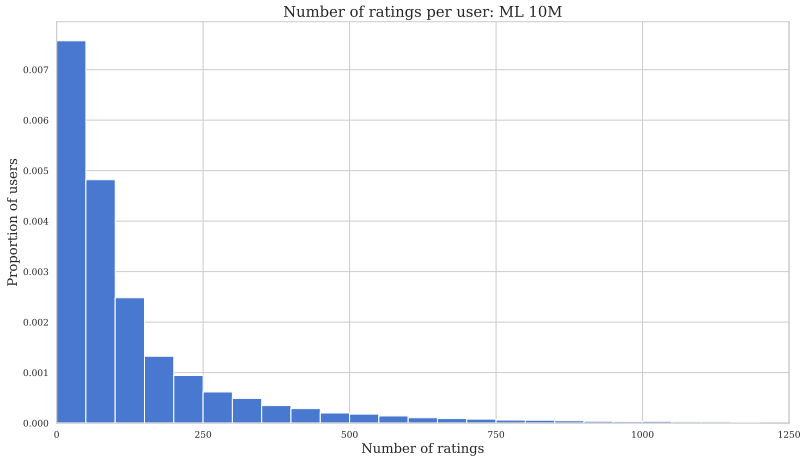

FIGURE 4.1: ML 10M number of ratings per user. Most users have rated a small number of movies.

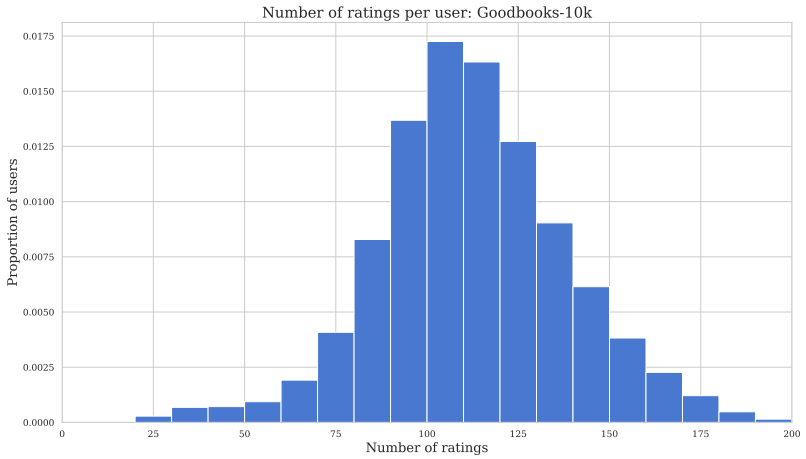

FIGURE 4.2: Goodbooks-10k number of ratings per user. Number of ratings per user is roughly normally distributed.

4.3.2 Number of Ratings per Item

The number of ratings per movie in the MovieLens datasets follow a similar right-tailed distribution to the number of ratings per user; however, since there are fewer movies than users, the average number of ratings per movie is higher.

The number of ratings per book in the Goodbooks-10k dataset does not follow a symmetric distribution, as is the case with ratings per user. Instead, the number of ratings per book in this dataset follows a similar right-tailed pattern to the MovieLens datasets.

Figures 4.3 and 4.4 show the distributions of the number of ratings per item in the ML 10M and Goodbooks-10k datasets respectively. As is the case with the distributions of the number of ratings per user, the three MovieLens datasets all have similar distributions across number of ratings per item and so only the largest of the datasets is shown below.

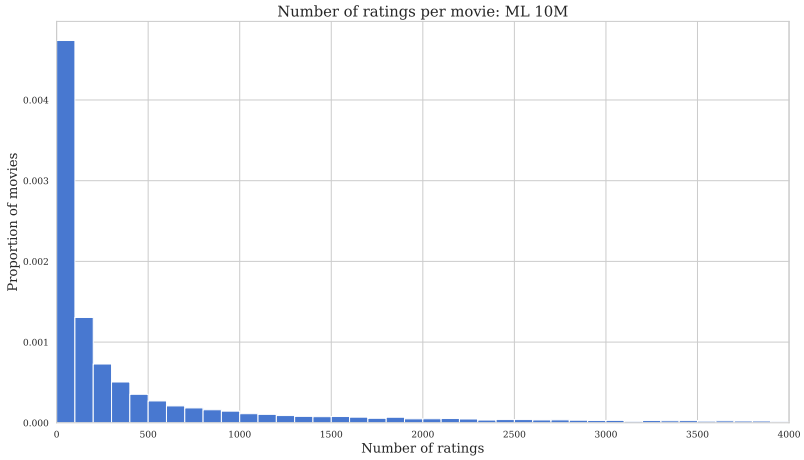

FIGURE 4.3: ML 10M number of ratings per movie.

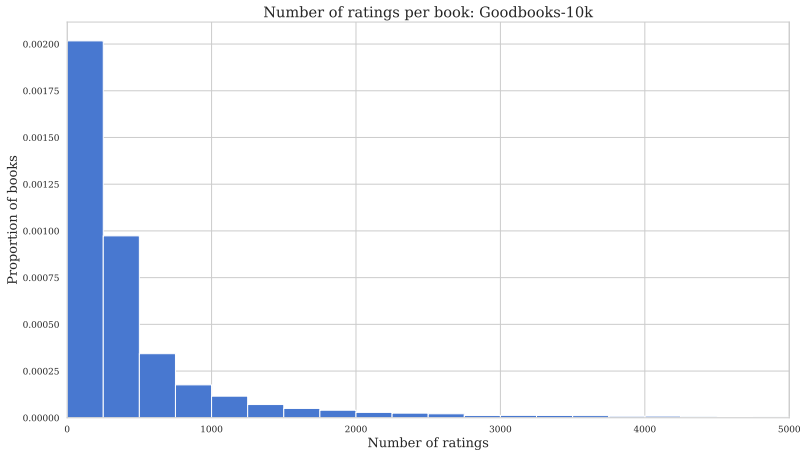

FIGURE 4.4: Goodbooks-10k number of ratings per book.

4.3.3 Rating Scales

In all of the datasets, the data are skewed in favour of positive ratings. For example, MovieLens 100k uses a 5-point integer-only rating scale. The middle value of this scale is 3 stars out of 5; however, the mean user rating is 3.53 and the median is 4. In each of the four datasets, both the median and mean values are higher than the middle point of the respective rating scale. Table 4.3 shows the 5-number summaries of each dataset.

Name	Min rating	Q2	Median	Q4	Max rating	Mean rating
ML 100K	1	3	4	4	5	3.53
ML 1M	1	3	4	4	5	3.58
ML 10M	0.5	3.0	4.0	4.0	5.0	3.51
Goodbooks-10k	1	3	4	5	5	3.92

TABLE 4.3: Summary of the distribution of ratings in each dataset

4.4 Distribution of Genres

All MovieLens datasets, as well as Goodbooks-10k, include item meta data. In the case of MovieLens, movies have been tagged with at least one of a total of 18 genres. Books in Goodbooks-10k have been tagged with at least one of 10 genres.

Table 4.4 shows the 18 genre categories used in the MovieLens datasets, as well as the proportion of movies that were assigned that genre.

Genre	ML 100k	ML 1M	ML 10M
Action	0.149	0.134	0.138
Adventure	0.080	0.076	0.096
Animation	0.025	0.028	0.027
Children's	0.073	0.067	0.049
Comedy	0.300	0.314	0.350
Crime	0.065	0.054	0.105
Documentary	0.030	0.030	0.045
Drama	0.431	0.403	0.500
Fantasy	0.013	0.018	0.051
Film-Noir	0.014	0.012	0.014
Horror	0.054	0.091	0.095
Musical	0.033	0.030	0.041
Mystery	0.036	0.028	0.048
Romance	0.147	0.124	0.158
Sci-Fi	0.060	0.074	0.071
Thriller	0.149	0.131	0.160
War	0.042	0.038	0.048
Western	0.016	0.018	0.026

TABLE 4.4: Summary of the distribution of ratings in each dataset

From table 4.4, one can see that the categorical variable genre is highly unbalanced. Of all the categories, drama has the greatest representation, with half of the movies in ML10M falling under this genre. The second most well-represented category is that of comedy, with 35% of the movies in ML10M falling under this genre. Since the items in all four of these datasets may be labelled with one or more genres simultaneously, attempting to predict their genres could be approached as a multi-label classification problem; however, due to the class imbalance in these datasets, it was approached as a binary classification task in this dissertation.

Chapter 5

Findings

This chapter details the results of tuning and evaluating the CGT model. All available user-item ratings were used to tune the rating model, while 90% of the movie genre labels were used to train and tune the genre model. As a proof of concept, the genre model was trained as a binary classifier, predicting whether items belonged to one particular genre or not. In the case of the MovieLens datasets, the genre model predicted whether movies fell under the drama category, since this category had the largest representation in each of the three datasets. For Goodbooks-10k, the "adult-fiction" category was used, as almost half of all books fall under this genre. The chapter concludes with an exploratory analysis of the item latent factors.

5.1 Hyperparameter Tuning

A number of searches was undertaken to tune the hyperparameters of the CGT model, using the cross validation framework outlined in section 3.4. Due to the different sizes of the MovieLens (ML) datasets, the smallest dataset, ML100k, was used to test the widest range of hyperparameters. Then, a subset of the best-performing hyperparameters was tested again on the ML1M dataset. No hyperparameter tuning was done on ML10M or Goodbooks-10k due to their size – grid searching each of these datasets would require a number of days computing time. The best hyperparameters from ML1M were used for both of these.

In each hyperparameter search, the best parameter values were defined as those that minimised the average cross-validated log loss of the genre classifications.

When initialising the weights of each model, the same random seed was used for reproducibility.

5.1.1 MovieLens100k

ML100k was used to search the widest range of hyperparameters. First, k – the number of latent factors, h – the number of hidden nodes in the rating model and j – the number of hidden nodes in the genre model, were tested using a grid search. Then, using the best values of k, h and j, the dropout rate and choice of activation function were tested.

Number of Latent Factors and Hidden Neurons

The range of values that were tested for k, h and j are defined by the three sets $k \in \{200, 500, 1000, 1500, 2000\}$, $h \in \{0, 50, 100, 200, 500\}$ and $j \in \{50, 100, 200, 500\}$. In the case where $h = 0$, the rating model did not contain a hidden layer, and thus reproduced simple matrix factorisation. While tuning these parameters, the dropout rate of the hidden layers, $dr1$ and $dr2$ were both arbitrarily fixed at 0.2 and the ReLU activation function was used. Different dropout rates and activation functions were tested in separate grid searches. A total of 48 different combinations were tested in this grid search, and the top 5 models are shown in table 5.1.

k	h	j	**Avg. CV log loss**	**Avg. CV acc.** %
1000	0	200	0.5983	66.89
1000	0	100	0.5999	65.90
200	0	500	0.6039	63.98
1000	0	50	0.6054	65.30
200	0	200	0.6060	64.11

TABLE 5.1: Results of first grid search performed on ML100k. Average CV log loss and accuracy are shown for different combinations of k, h and j.

The best performing set of hyperparameters in the first grid search was $k = 1000$, $h = 0$ and $j = 200$. It is notable that all of the top five used a value of 0 for h, i.e. CGT performed best when the rating model did not have a hidden layer. Thus, it would seem that the addition of a hidden layer to the rating model makes it more difficult for the genre model to decode the item latent factors.

Number of Epochs

Training the model for too many epochs could result in overfitting on the training set. Therefore, it was necessary to find the optimal number of epochs for both the rating and genre model. To do this, each model was trained for a large number of epochs. In each epoch, 20% of the training data was randomly held out as a validation set and the training and validation loss recorded at the end of the epoch.

First, the rating model was trained for 10 epochs. The optimal number of epochs was chosen as the point where the validation MSE was at a minimum. Figure 5.1 shows the training (blue) and validation (orange) MSE of the rating model over 10 epochs.

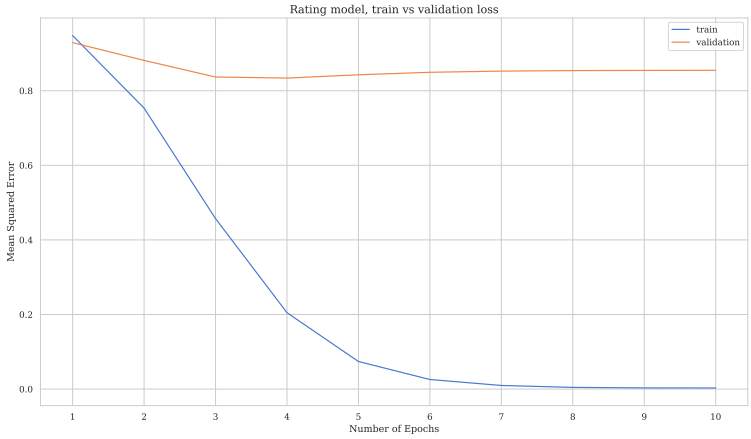

FIGURE 5.1: Training vs validation loss of rating model, training for 10 epochs on ML100k.

One can see in figure 5.1 that the validation MSE stops decreasing after 4 epochs, while the training MSE continues to converge towards zero. This is a sign of overfitting, so it was decided to train the rating model for 4 epochs.

The rating model was re-initialised and trained for 4 epochs so the item embedding layer could be used in the genre model, which was then trained for 10 epochs. Figure 5.2 shows the training and validation log loss of the genre model over 10 epochs.

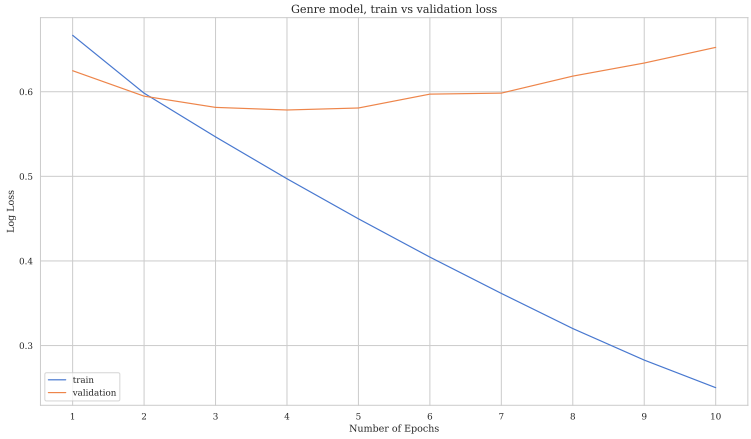

FIGURE 5.2: Training vs validation loss of genre model, training for 10 epochs on ML100k.

Similarly, the validation log loss of the genre model stops decreasing after 4 epochs, while the training log loss converges towards zero. To avoid overfitting, 4 was chosen as the number of epochs for which to train this model too.

Dropout Rates

Since the CGT model performed best with no hidden layer in the rating model, only $dr2$, the dropout rate in the genre model, needed to be tuned. Eight dropout rates in the set $dr2 \in \{0.0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$ were tested. The performance of the top 5 values of $dr2$ is shown in table 5.2.

$dr2$	**Avg. CV log loss**	**Avg. CV acc. %**
0.0	0.5938	66.69
0.05	0.5941	66.62
0.1	0.5943	66.62
0.15	0.5946	66.62
0.2	0.5949	66.69

TABLE 5.2: Results of third ML100k hyperparameter search which tested different values of$dr2$.

The best performing model did not use any dropout in the hidden layer of the genre model. Indeed, dropout did not appear to improve out-of-sample performance.

Activation Functions

Finally, four different activation functions were tested in the hidden layer of the genre model using the best parameters from the results above. The CV performance of each activation function is shown in 5.3.

Activation function	**Avg. CV log loss**	**Avg. CV acc. %**
ReLU	0.5938	66.69
Tanh	0.6066	66.36
Linear	0.6068	66.36
Softplus	0.6072	66.09

TABLE 5.3: Comparison of activation functions

ReLU showed the best performance of the five activation functions tested.

5.1.2 MovieLens1M

A second set of hyperparameter searches was performed on MovieLens1M, using a subset of the best performing values tested on MovieLens100k. The parameters tested on ML1M included the number of latent factors, number of hidden neurons and the dropout rate.

Number of Latent Factors and Hidden Neurons

The number of latent factors, k, was tested with values of 1000 and 200 as these showed the best performance on ML100k. No hidden layer was used in the rating model, i.e. h was kept at 0. The number of hidden nodes in the genre model, j, was tested at 100, 200 and 500. This totalled six different combinations, all of which are shown in table 5.4.

k	j	**Avg. CV log loss**	**Avg. CV acc. %**
1000	200	0.5099	73.16
1000	500	0.5123	72.98
1000	100	0.5146	72.68
200	500	0.5265	72.62
200	200	0.5343	72.05
200	100	0.5475	71.36

TABLE 5.4: Best values of k and j on MovieLens 1M dataset.

As was the case with ML100k, the best values for k and j for ML1M were 1000 and 200 respectively.

Dropout Rates

Dropout rates of 0, 0.05 and 0.1 were tested on ML1M, the results are shown below in table 5.5. As there was no hidden layer in the rating model, only $dr2$ needed to be tested.

$dr2$	**Avg. CV log loss**	**Avg. CV acc. %**
0.0	0.5099	73.16
0.05	0.5114	73.04
0.1	0.5119	73.04

TABLE 5.5: Performance for three dropout rates tested on ML1M.

Again, the lowest CV log loss was achieved with no dropout.

5.2 Holdout Testing

After tuning the hyperparameters, CGT was tested on the 10% holdout set for each of the three MovieLens datasets and the Goodbooks-10k dataset. For ML10M and Goodbooks-10k, the model was trained using the best hyperparameters from the ML1M model, i.e. without tuning any hyperparameters.

5.2.1 Classification Metrics

Performance on the holdout set was evaluated using four different classification metrics, namely accuracy, precision, recall and the F1 score. The test results across all four datasets are shown in table 5.6. "Positives" refers to the proportion of items in the holdout sets that had a positive class label. For example, the holdout set in ML100k consisted of a total of 169 movies, of which 73 were dramas, while 495 of the 1000 books in the Goodbooks-10k dataset belonged to the adult-fiction class.

Dataset	Positives	Accuracy	Precision	Recall	F1 score
ML100k	73/169 (43%)	0.63	0.57	0.60	0.59
ML1M	144/371 (39%)	0.74	0.66	0.67	0.66
ML10M	517/1068 (48%)	0.70	0.68	0.72	0.70
GB 10k	495/1000 (50%)	0.71	0.72	0.68	0.70

TABLE 5.6: Classification performance of CGT on holdout data.

One can see in table 5.6 that the performance improves with the size of the dataset. ML100k has the worst performance with an F1 score of 0.59. ML10M and Goobooks-10k, which are of similar size and both have nearly even splits between positive and negative labels (e.g. drama vs not drama), show the best performance, each achieving an F1 score of 0.70.

For each of the four datasets listed above, performance is better than naively assigning the positive label to all items. In the case of Goodbooks-10k, precision was as high as 0.72, meaning that 72% of the movies identified by the model as "adult-fiction" were done so correctly. The recall value of 0.72 for ML10M means that 72% of the dramas in this dataset were correctly identified as such.

5.2.2 Number of Ratings per Item

Looking at table 4.2, one can see that ML10M has the highest average number of ratings per item, 937, compared to only 60 ratings per item in ML100k. Intuitively, the better performance on the dataset with more ratings per movie makes sense as each user-item interaction provides an additional training observation for training item latent factors. Goodbooks-10k, which saw the best performance, had the second-highest number of ratings per item, with the additional property that no item was rated fewer than 8 times.

The relationship between the number of user ratings and classification accuracy was investigated by binning movies by rating count and then taking the average accuracy for each bin. The result of this investigation is shown in figure 5.3. Each bar represents a single bin's mean classification accuracy. Blue bars are for training data and orange represents testing data. The ranges for the bins were chosen to ensure a minimum of 20 test observations in each bin.

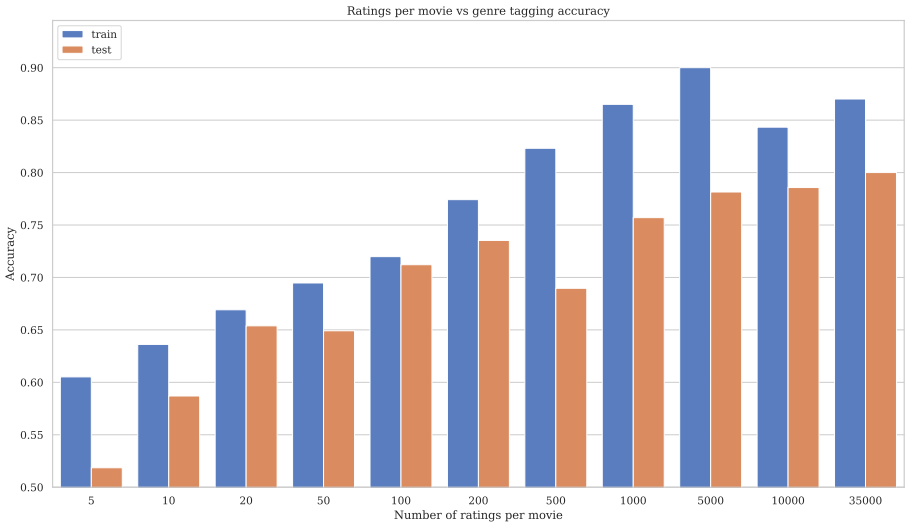

FIGURE 5.3: Movies with more ratings tend to be more accurately classified.

Figure 5.3 shows that as the number of ratings per movie increases, the accuracy with which CGT can predict its genre increases too. For movies with 1000 or more ratings, the average classification accuracy is over 75%. To investigate this relationship further, a subset of ML10M was created in which only movies and users with at least 100 ratings were selected. Applying this filter reduced the size of the dataset by just over 20% to 7 799 240 total ratings. The number of users and movies in this subset were reduced to 26 502 and 5 897 respectively – down from 69 878 users and 10 677 movies in the original set. This modified dataset was then split 90/10 into training and holdout sets. Table 5.7 shows the performance of CGT on the holdout portion of the modified subset of ML10M.

Dataset	Class weight	Accuracy	Precision	Recall	F1 score
ML10M*	279/590 (47%)	0.79	0.78	0.78	0.78

TABLE 5.7: Classification performance of CGT on holdout data of the modified ML10M dataset (ML10M*); only users and items with 100 or more ratings were included in this dataset.

It is clearly evident in table 5.7 that CGT performs better when trained on ratings made by users with more ratings. The implications of this are that if one were to use CGT for attribute tagging, one should form a subset of ratings made by the most active users, as each of their ratings hold more inherent meaning than, say, a user who has only rated a single item.

5.3 Latent Factor Analysis

Some investigation was carried out to understand *how* the latent factors capture descriptive attributes of items. The item embedding matrix of ML10M was used, as

each movie in this set had more ratings on average.

5.3.1 Dimensionality reduction

To understand the relationship between the latent factors in the embedding layer and genres in ML10M, the item embedding matrix was reduced from 1000 dimensions to 2. Dimensionality reduction was done using principal component analysis, in which the 1000-dimensional space of the embedding matrix was decomposed into 2 mutually orthogonal dimensions – or components – that captured the maximum amount of variance from the original space. While this reduction in dimensionality inevitably leads to a loss of information, it allows the movies to be visualised in the latent factor space.

5.3.2 Distribution of Genres

Figure 5.4 shows the distribution of two genres, "Children" and "Documentary", in the first two principal components of the latent factor space. Out of the total 10 000 movies in ML10M, 528 are classified in the "Children" genre and 479 are documentaries. There are only 2 movies which are labelled as both a children's film and a documentary.

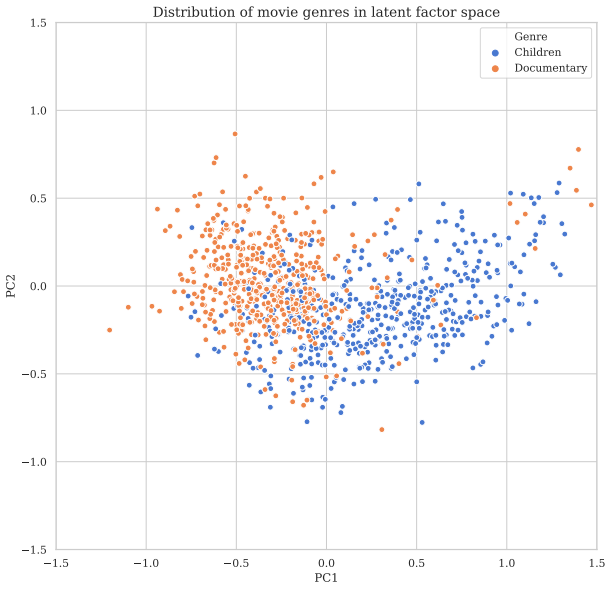

FIGURE 5.4: Children's movies and documentaries shown in the first two principal components of the latent factor space.

Although the genre model was not trained to predict either of these genres, this figure serves to illustrate the inherent ability of the latent factors to capture descriptive

features of movies. While not entirely separable, the difference in principal component values between these two genres can be seen in just two dimensions. The geometric means of each genre are given in table 5.8.

Genre	PC1	PC2
Children	0.257	-0.128
Documentary	-0.295	0.033

TABLE 5.8: Geometric means of "Children" and "Documentary" genres in first two principal components of latent factor space.

It is evident from the table above that a typical children's movie has a positive value for the first principal component and a negative value for the second component, while the opposite is true for the "Documentary" genre.

5.3.3 Distribution of Ratings

Figure 5.5 provides a visualisation of the average rating of all 10 000 movies in the latent factor space. This hex bin plot groups movies into hexagons with the boundaries defined in terms of the first two principal components. The value of each hexagon is calculated as the mean rating of all movies that fall inside its boundaries.

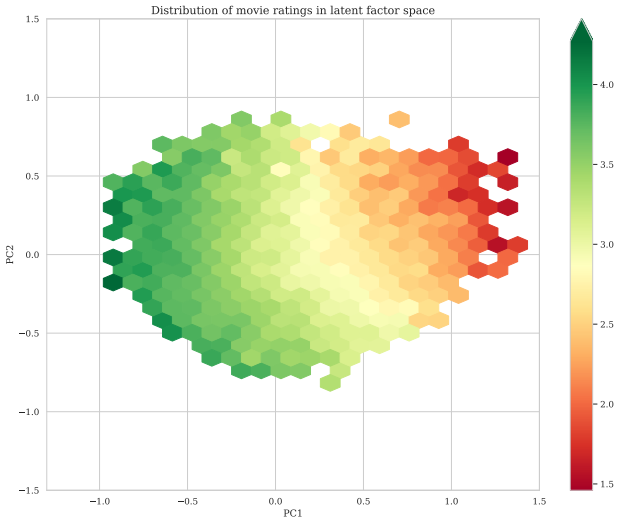

FIGURE 5.5: Hex bin plot shows average rating within each hexagon. Movies in the bottom left receive higher ratings on average than those in the top right.

There is a clear pattern in figure 5.5, with higher rated movies having larger negative values of the first principal component and lower rated movies having larger positive values along this dimension. Since the embeddings were trained for the purpose

of predicting user ratings, it makes sense that the first principal component would correlate with the movie ratings. One possible interpretation of figure 5.5 is that the first principal component simply captures the quality of a movie. The remaining principal components then capture the more specific nuances of movies, allowing the rating model to predict user-specific preferences. The genre model then learns to interpret the many dimensions of the latent factor vector to predict the genre category.

5.3.4 Classification Accuracy

Finally, the relationship between latent factors and classification accuracy was explored. A similar hex bin plot to figure 5.5 was used with the mean classification accuracy displayed in each hexagon. This is shown in figure 5.6.

It is interesting to note that higher accuracies were observed at the more extreme values of the first two principal components. This suggests that movies with more extreme latent factor values are more descriptive and therefore possibly easier to predict. If this is indeed the case, then the goal of the rating model in CGT should be to train latent factors that are maximally dissimilar to each other.

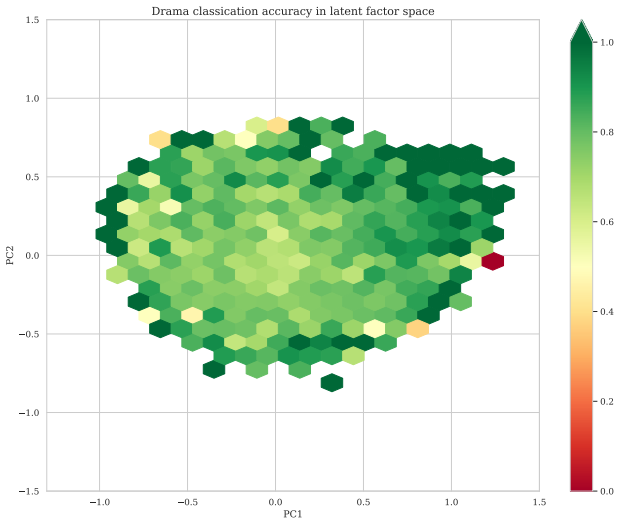

FIGURE 5.6: Classification accuracy across PC1 and PC2 of latent factor space. Movies with more extreme latent factors tend to have more extreme accuracy scores.

Chapter 6

Conclusions and Recommendations

6.1 Collaborative Genre Tagging

In this minor dissertation, it was shown that the collaborative genre tagging technique can be used to classify the genres of items such as movies and books using nothing more than explicit user ratings as inputs. Although neural architectures have been the popular choice in more recent collaborative filtering efforts (Zheng et al., 2016; He et al., 2017), CGT performs best when the rating model employs simple matrix factorisation without any hidden layers. CGT performs best on items that have a sufficiently large number of ratings. When trained on a subset of ratings from ML10M where no users or items with fewer than 100 ratings were included, the classification accuracy for the drama category was as high as 79%.

6.2 Implications of this Research

This work serves as a proof of concept that user feedback can be used to predict category labels of items. Labels such as the genre of a movie or book are useful to users when assessing recommendations made to them. CGT could serve as a complement to the recommender systems that have become ubiquitous in online retail and media today, capable of helping users to understand *why* certain suggestions are shown to them.

6.3 Future Work

While some grid searches were performed to find the optimal hyperparameters for the CGT model, these searches were by no means exhaustive of the space of all possible model architectures. There remains room for further research to be done in refining the model architecture of CGT.

CGT was only trained as a binary classifier, predicting the probability that an item belongs to one particular category at a time. Since items such as movies and books

can fall under multiple genres simultaneously, there is still room for CGT to be tested as a multi-label classification model. This would require further research into appropriate performance metrics for this type of setting.

Furthermore, the categories corresponding to "positive" labels (that is, drama for movies and adult-fiction for books) were chosen specifically to obtain a roughly equal representation of the positive and negative classes. There would be value in learning to identify more niche categories which, by definition, fall in the minority of items. This is exemplified by Netflix's use of what they term "microtags" – highly personalised labels that they use to help users select what to watch. The findings that have been outlined in section 5.3 could be expanding further to investigate the use of unsupervised techniques, such as clustering, for discovering new genres in rated items.

Bibliography

Alom, Md Zahangir et al. (2018). "The history began from AlexNet: a comprehensive survey on deep learning approaches". In: *arXiv preprint arXiv:1803.01164*.

Amatriain, Xavier et al. (2011). "Data Mining Methods for Recommender Systems". In: *Recommender systems handbook*. Springer, pp. 30–72.

Bailey, Katherine (2016). *Matrix Factorization with Tensorflow*. URL: https://katbailey.github.io/post/matrix-factorization-with-tensorflow/.

Bell, Robert M, Yehuda Koren, and Chris Volinsky (2008). "The bellkor 2008 solution to the netflix prize". In: *Statistics Research Department at AT&T Research* 1.

Bennett, James, Stan Lanning, et al. (2007). "The netflix prize". In: *Proceedings of KDD cup and workshop*. Vol. 2007. New York, NY, USA., p. 35.

Collobert, Ronan et al. (2011). "Natural language processing (almost) from scratch". In: *Journal of machine learning research* 12.Aug, pp. 2493–2537.

Dacrema, Maurizio Ferrari, Paolo Cremonesi, and Dietmar Jannach (2019). "Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches". In: *arXiv preprint arXiv:1907.06902*.

Desrosiers, Christian and George Karypis (2011). "A Comprehensive Survey of Neighborhood-based Recommendation Models". In: *Recommender systems handbook*. Springer, pp. 107–143.

Di Noia, Tommaso et al. (2012). "Linked open data to support content-based recommender systems". In: *Proceedings of the 8th International Conference on Semantic Systems*. ACM, pp. 1–8.

Francesco Ricci, Lior Rokach and Bracha Shapira (2011). "Introduction to Recommender Systems Handbook". In: *Recommender systems handbook*. Springer, pp. 7–29.

Ge, Mouzhi, Carla Delgado-Battenfeld, and Dietmar Jannach (2010). "Beyond accuracy: evaluating recommender systems by coverage and serendipity". In: *Proceedings of the fourth ACM conference on Recommender systems*. ACM, pp. 257–260.

Goldberg, David et al. (1992). "Using Collaborative Filtering to Weave an Information Tapestry". In: *Commun. ACM* 35.12. (Accessed on 10/10/2017).

Goldberg, Ken et al. (2001). "Eigentaste: A constant time collaborative filtering algorithm". In: *information retrieval* 4.2. (Accessed on 10/10/2017), pp. 133–151.

Harper, Maxwell and Joseph Konstan (2016). "The movielens datasets: History and context". In: *Acm transactions on interactive intelligent systems (tiis)* 5.4, p. 19.

He, Xiangnan et al. (2017). "Neural collaborative filtering". In: *Proceedings of the 26th international conference on world wide web*. International World Wide Web Conferences Steering Committee, pp. 173–182.

Herlocker, Jonathan, Joseph Konstan, Al Borchers, et al. (1999). "An algorithmic framework for performing collaborative filtering". In: *22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1999*. Association for Computing Machinery, Inc, pp. 230–237.

Herlocker, Jonathan, Joseph Konstan, and John Riedl (2002). "An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms". In: *Information retrieval* 5.4, pp. 287–310.

– (2004). "Evaluating collaborative filtering recommender systems". In: vol. 22. 1. ACM, pp. 5–53.

Howard, Jeremy and Sebastian Ruder (2018). "Universal language model fine-tuning for text classification". In: *arXiv preprint arXiv:1801.06146*.

Hu, Yifan, Yehuda Koren, and Chris Volinsky (2008). "Collaborative filtering for implicit feedback datasets". In: *2008 Eighth IEEE International Conference on Data Mining*. Ieee, pp. 263–272.

Jawaheer, Gawesh, Martin Szomszor, and Patty Kostkova (2010a). "Characterisation of explicit feedback in an online music recommendation service". In: *Proceedings of the fourth ACM conference on Recommender systems*. ACM, pp. 317–320.

– (2010b). "Comparison of implicit and explicit feedback from an online music recommendation service". In: *proceedings of the 1st international workshop on information heterogeneity and fusion in recommender systems*. ACM, pp. 47–51.

Koren, Yehuda (2009). "The bellkor solution to the netflix grand prize". In: *Netflix prize documentation* 81.2009, pp. 1–10.

Koren, Yehuda, Robert Bell, and Chris Volinsky (2009). "Matrix factorization techniques for recommender systems". In: *Computer* 8, pp. 30–37.

Koren, Yehunda and Robert Bell (2011). "Advances in Collaborative Filtering". In: *Recommender systems handbook*. Springer, pp. 145–184.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*, pp. 1097–1105.

Liao, Kevin (2018). *Prototyping a Recommender System Step by Step Part 2: Alternating Least Square (ALS) Matrix Factorization in Collaborative Filtering*. URL: https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square-als-matrix-4a76c58714a1.

Lops, Pasquale, Marco De Gemmis, and Giovanni Semeraro (2011). "Content-based recommender systems: State of the art and trends". In: *Recommender systems handbook*. Springer, pp. 73–105.

Madrigal, Alexis (2014). *How Netflix Reverse-Engineered Hollywood*. URL: https://www.theatlantic.com/technology/archive/2014/01/how-netflix-reverse-engineered-hollywood/282679/.

Maksai, Andrii, Florent Garcin, and Boi Faltings (2015). "Predicting online performance of news recommender systems through richer evaluation metrics". In: *Proceedings of the 9th ACM Conference on Recommender Systems*. ACM, pp. 179–186.

McKee, Jordan (2018). *Global Digital Commerce Sales To Near $6 Trillion By 2022*. URL: https://www.forbes.com/sites/jordanmckee/2018/09/11/global-digital-commerce-sales-to-near-6-trillion-by-2022.

Mohamed, Marwa Hussien, Mohamed Helmy Khafagy, and Mohamed Hasan Ibrahim (2019). "Recommender Systems Challenges and Solutions Survey". In: *2019 International Conference on Innovative Trends in Computer Engineering (ITCE)*. IEEE, pp. 149–155.

Musiol, Martin (2016). "Speeding up Deep Learning Computational Aspects of Machine Learning". In:

Nair, Vinod and Geoffrey E Hinton (2010). "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814.

Resnick, Paul and Hal R Varian (1997). "Recommender systems". In: *Communications of the ACM* 40.3, pp. 56–59.

Ruder, Sebastian (2019). *Matrix Factorization with Tensorflow*. URL: https://www.kdnuggets.com/2019/09/state-transfer-learning-nlp.html.

Sedhain, Suvash et al. (2015). "Autorec: Autoencoders meet collaborative filtering". In: *Proceedings of the 24th International Conference on World Wide Web*. ACM, pp. 111–112.

Shani, Guy and Asela Gunawardana (2011). "Evaluating Recommender Systems". In: *Recommender systems handbook*. Springer, pp. 257–294.

Shardanand, Upendra and Pattie Maes (1995). "Social information filtering: Algorithms for automating" word of mouth"". In: *Chi*. Vol. 95. Citeseer, pp. 210–217.

Sharif Razavian, Ali et al. (2014). "CNN features off-the-shelf: an astounding baseline for recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 806–813.

Shreedhar, Rajesh Bhat and Souradip Chakraborty (2019). *Why not Mean Squared Error(MSE) as a loss function for Logistic Regression?* URL: https://towardsdatascience.com/why-not-mse-as-a-loss-function-for-logistic-regression-589816b5e03c.

Srivastava, Nitish et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1, pp. 1929–1958.

Su, Xiaoyuan and Taghi M Khoshgoftaar (2009). "A survey of collaborative filtering techniques". In: *Advances in artificial intelligence* 2009. (Accessed on 10/10/2017).

Swearingen, Kirsten and Rashmi Sinha (2001). "Beyond algorithms: An HCI perspective on recommender systems". In: *ACM SIGIR 2001 Workshop on Recommender Systems*. Vol. 13. 5-6. Citeseer, pp. 1–11.

Zajac, Zygmunt (2017). "Goodbooks-10k: a new dataset for book recommendations". In: *FastML*.

Zheng, Yin et al. (2016). "A neural autoregressive approach to collaborative filtering". In: *arXiv preprint arXiv:1605.09477*.