Collaborative Recommender Systems for Multi-Stakeholder Data-Sparse Marketplaces

Fourth Year Project Report

Rhim Shah

Supervised by Michael Osborne (Main) and Edwin Lock (Secondary)

Department of Engineering Science

University of Oxford

May 2022

Abstract

Marketplaces exist in many different commercial settings, from the recommendation list that users have to pick songs from for Spotify [1], to the movies that Netflix [2] recommends. Recommender systems (RecSys) are broadly used across marketplaces to ensure that not only are users receiving the best item recommendations but also that the interests of the other stakeholders involved are also being taken into consideration. Within the more general RecSys problem, several issues exist, most notably, data sparsity – only some users interact with some items (an item is e.g. a song for Spotify).

Current literature mostly focuses on single-objective RecSys and, some literature focuses on multi-stakeholder RecSys in non-sparse environments, but there is little research that analyses a full multi-stakeholder recommender system. This report establishes the background and key context behind recommender systems that are used in commercial marketplaces, where data-sparsity is a defining issue. The majority of this report applies these concepts across three datasets of varying size under five different technical algorithms, using a collaborative approach (historic ratings are compared between users using a ratings matrix to generate predictions). These datasets are the Restaurant Rec dataset [3], the Movie Lens Dataset [4] and the Yahoo Music Dataset [5]. Three objectives are combined using multi-objective optimisation (MOO) to create a full recommender system. Matrix factorisation with stochastic gradient descent was concluded as the best algorithm for highly sparse datasets, with Pareto optimality being the optimal multi-objective optimisation approach.

Acknowledgements

I would like to express my great appreciation for my direct supervisor, Michael Osborne, for supporting me throughout this self-initiated project and providing me with highly informative and constructive advice. I would also like to thank Edwin Lock for his helpful insights.

This project stemmed from an internal need from a start-up that I co-founded called Carbon Codes [6]. Carbon Codes' mission is to improve the sustainability of the food-service industry by connecting consumers to discounted eco-conscious food from our partner restaurants. The initial focus of the project was to improve the recommendations of restaurants to users. The project subsequently broadened to more general marketplaces, increasing impact and addressing a more widely applicable problem. A final thanks goes to the Carbon Codes team for providing AWS [7] resources for increased compute power.

Contents

1 Introduction

1.1 Project Aim

The aim of this project was to explore well-adopted approaches for recommending items to users in commercial marketplaces, where data-sparsity is prevalent, and to understand the downfalls of these approaches. The primary focus of this report is to provide a novel outlook on the best multi-stakeholder recommender systems in sparse environments – something that current literature does not consider. Both the contextual and technical background are set in depth before diving into the experimentation. The two key areas that are to be explored in the experimentation are the rating predictions (for user interests) using collaborative filtering, and multi-objective optimisation (referred to in this report as MOO) to produce a final ranking based on all stakeholder interests, all within the context of sparsity. Collaborative filtering (CF) predicts user preferences (i.e. ratings) in item selection based on the known historic item ratings from users [8].

1.2 Commercial Background

1.2.1 Applications

Recommender systems (RecSys) are widely used across all spaces of technology, from Spotify's song suggestions to Netflix's movie recommendations, and even to restaurant recommendations on Deliveroo. These important commercial use-cases have one thing in common – they serve as marketplaces where several stakeholders are involved, all with differing interests and objectives. Before diving into the more technical aspects of how the recommender system problem is structured, it is important to fully understand the commercial scope of the task at hand.

The focus of this project is replicating real-life applications in order to demonstrate the commercial applicability of this research. In particular, data-sparsity is an important aspect of recommender systems that is explored in this report, which current state-of-the-art approaches fail to address properly. This report looks at three applications:

- Restaurant recommendations. This applies to a broad range of use cases, from Deliveroo to Carbon Codes (a platform for matching consumers with sustainable discounts at local restaurants). A context-aware collaborative approach often supersedes other methods as little information is required from the user – e.g. the user does not have to identify their labels via a long questionnaire that would normally be required to assess their preferences for a content-based matching approach. We focus on collaborative filtering with the Restaurant Rec dataset [3] which is comparatively small in size but is much less sparse compared to the other datasets ($\sim$6.5%).

- Movie recommendations. Improving the quality of movie recommendations for users of Netflix, and other similar SVoD (Streaming Video on Demand) platforms. In general, SVoD movie recommendations are based on a user's past watch history. We use the Movie Lens dataset [4] which is the largest dataset but also very sparse ($\sim$3%), i.e. many users have only rated a few movies from a large catalogue of movies.
- Music recommendations. Although the most obvious example is Spotify, there are many other applications such as Youtube Music and Amazon Music. Often deep analysis of songs (danceability, energy, tempo etc.) is used to pair similar songs to the ones a user has listened to, as opposed to just the ratings. However, we focus on a rating-based collaborative approach, which is generally the preferred method. We use use the Yahoo Music dataset [5] which is comparatively the middle dataset by size but it is the most sparse ($\sim$0.35%).

To summarise, we focus on exploring collaborative filtering since this the most used and effective technique for the applications discussed above where there are no user profiles are not created [8]. This is predominantly for optimising for user interests, other parts of the system use other non machine learning, more simplistic, approaches to rank items for the other stakeholder objectives.

1.2.2 Stakeholders

Across all of the applications discussed in the previous section, there are three stakeholders involved each with their own objective. Note that throughout this report 'multi-stakeholder' and 'multi-objective' are used interchangeably. The three stakeholders are:

1. the product user who views the recommendation list (e.g. Spotify user),
2. the vendor (e.g. music artist) that is supplying the item (e.g. a Spotify track), and finally,
3. the business that owns and supports the marketplace (e.g. Spotify).

Providing a final recommendation list that takes into account the interests of all of these three stakeholders is a complex novel problem, especially in the presence of sparsity, that involves two key elements: machine learning (for the collaborative filtering) and multi-objective optimisation (for the final ranking).

In this report, the user has an objective that is optimised via collaborative filtering, whilst the businesses and vendors have objectives that are generally solely dependent on the items (e.g. certain products maximise revenue). It is important to note that the end goal for both the business and the vendor is to produce recommendations that keep the user happy as this extends the life time value of the user. Hence, user satisfaction is the most important objective to optimise, represented by user ratings – the focus of this report is on collaborative filtering and the overall RecSys approach of combining the objectives to form a final ranking through MOO.

1.2.3 Commercial Value

We have understood the applications from a qualitative viewpoint, it is also important to quantify the impact of recommender systems on the success of commercial businesses. For a commercial business, optimising the recommendation approach holds significant value. 35% of Amazon.com's revenue comes from behavioural recommendations [9], and more than 80% of Netflix TV shows that are watched are discovered via recommendations [10]. This is the primary reason why Netflix held the Netflix Challenge – an opportunity for researchers around the world to improve Netflix's recommendation algorithm for a prize of 1 million USD [11]. This extends past Netflix to many consumer-facing internet-based product, which are marketplaces, even where you do not initially expect them to be, either for information, services or for physical goods. A marketplace is an intermediary that helps facilitate economic interaction between two or more agents [12].

As can be seen, the need to explore RecSys not only from a research perspective, but also from a business angle, is evident. An important distinction to make is that we are not considering search-based recommendation problems – we are looking at passive RecSys, not active where e.g. a search query is used to initiate the recommendations.

1.3 Technical Background

1.3.1 User-Centric RecSys Approaches

The RecSys problem can be summarised as: how can we best recommend an item j to user i? Many different data points, algorithms and methods are combined to answer this question to provide the best recommendations nowadays. The most common approaches include:

- **Content-based** recommender systems, users and items are matched via descriptive labels and attributes. For example, knowing that user *i* likes Italian food means that a restaurant recommender system can suggest other Italian restaurants. This is an example of supervised learning. The biggest disadvantage of content-based recommender systems is that the system always needs to know attributes of users and items, which can be impossible for new users and items [13].

- **Collaborative filtering (CF)** compares the past interaction history of a user to the interaction history of other users to generate recommendations [8]. This is unsupervised learning. CF relies on a ratings matrix where the items are the columns, users are the rows, and the entries are the ratings. As discussed previously, this is the primary approach explored in this report given the applications.
- Other approaches include **knowledge-based** recommendations which utilise precise search based attributes (e.g. user *i* wants restaurants that are Italian, medium price range and within 3 miles) [13]. **Group-based** recommendations are focused on recommending items to a group of users as opposed to a single user [14].
- The final RecSys approach worth mentioning is the use of **multi-armed bandits (MABs)**. MABs are an online evaluation method that require sequential updates to the system in order to provide a recommendation to a user [15]. MABs are effective at tackling sparsity since they can explore new options, as well as being able to exploit data that may already be available. The problem associated with extreme sparsity is known as the ‘cold start problem’ – how do you identify similar users/items if a user/item has little or no interaction history?

Most commercial systems combine content-based recommendation and collaborative filtering to create a hybrid system [13], in addition to using other context (e.g. location) – in the ML-world and therefore the RecSys-world, all the data available is used. Thinking more granularly, the RecSys approach that is selected depends on many factors, some of which are: requirement for processing accuracy or processing speed, dataset data sparsity, dataset size, objectives to be optimised and number of objectives. Therefore, although the underlying context is consistent (i.e. data-sparse marketplaces), the best approach and the best algorithm may vary depending on the specific application.

1.3.2 Collaborative Filtering

Collaborative filtering can be seen as an unsupervised generalisation of the classification problem [8]. Figure 1 shows that whilst classification has clear boundaries between the training and test rows and between the independent variables and the dependent variable, there are no clear boundaries in collaborative filtering and learning is based on entries.

In collaborative filtering, ratings matrices tend to be very large but sparse, and prediction can only be based on rated entries (so that an error can be measured). Algorithms are assessed by predicting the ratings of randomly selected entries which are hidden during the training phase.

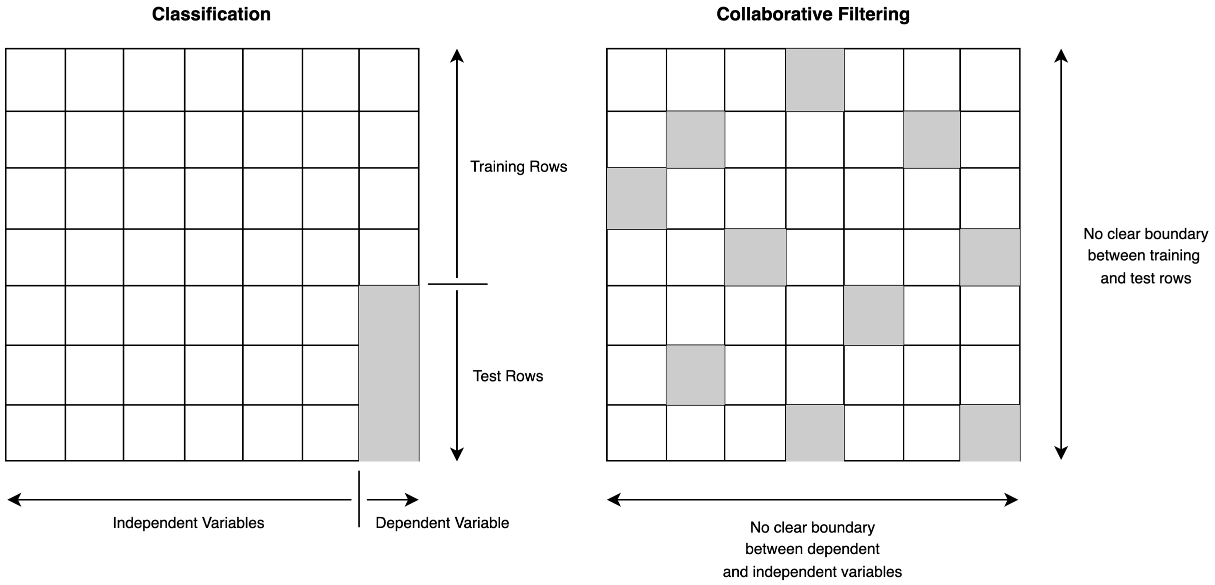

Classification
Collaborative Filtering
Training Rows
Test Rows
Independent Variables
Dependent Variable
No clear boundary between training and test rows
No clear boundary between dependent and independent variables

Figure 1: Comparison between how datasets are used in classification versus collaborative filtering.

Some important details around collaborative filtering:

- **Memory-based vs. model-based**. Model-based methods aim to form a model that is learned from a training dataset which can then be applied to a test dataset, in a similar way to the standard approaches for regression-based classification [8]. However, memory-based methods are instance based – e.g. k-nearest neighbours makes a prediction based on the top-*k* most similar users/items. Memory-based methods, as can be imagined, can be more computationally intensive [8]. This report covers a mix of both model-based and memory-based.
- **Explicit ratings vs. implicit ratings**. Explicit ratings can have any value across a range and unrated entries have the value of ‘0’. Implicit ratings, also known as unary ratings, have a value of either ‘1’, or ‘0’ for unrated entries e.g. a ‘1’ represents that the user clicked that item from a recommendation list. Implicit ratings can also be drawn from an explicit ratings matrix where any valued rating → ‘1’, and ‘0’s remain ’0’s. Explicit ratings are treated as preferences whilst implicit ratings are treated as confidences [16]. The ratings that are dealt with in this report are explicit

ratings.

- **Ratings prediction vs. ranking prediction.** Ranking prediction can be achieved directly or can be based on the explicit ratings prediction (i.e. forming a ranked list by ordering the ratings for a user) [8].
- **User-based vs. item-based approach**. A user-based approach considers users and relationships between rows, whereas an item-based approach considers items and relationships between columns. The most effective methods use a combined approach [17].

1.3.3 Common Collaborative Filtering Algorithms

Some common collaborative filtering methods include:

- **K-Nearest Neighbours (KNN).** The top *k* most similar rows/columns are used to form a prediction for user *i* on item *j* [17].
- **Naïve Bayes.** Bayes' theorem is used to maximise the posterior probability of the rating for user i on item j taking a certain value [18].
- **Latent Factor Models (LFMs).** These are highly efficient models that factorise the original ratings matrix into two latent factor matrices, before optimising these matrices and then recombining them [8]. Specific examples include SVD++. These methods are also known as matrix factorisation.
- **Rule-based.** Association rules work by clearly outlining a set of rules that can dictate what items a user may like, given that they are in a particular bucket [19].
- **Evolutionary Algorithms (EAs).** EAs work by evolving a population of random solutions, usually by combining solutions that are deemed better than the rest, until they are accurate [20]. These algorithms are overly simplistic and are usually ineffective [21].

This report explores KNN (Section 3.3.2), Naïve Bayes (Section 3.3.3) and LFMs (Section 3.3.4).

1.3.4 Common Collaborative Filtering Problems

There are three main problems that define whether recommendations are effective or not, especially associated when dealing with marketplaces. This report addresses these issues.

- **Data Sparsity.** Only some users have rated some items [17]. This is the case for all marketplaces nowadays and is thus a defining feature of the datasets used in this report. Sparsity is calculated using,

$$\text{sparsity} = \frac{\text{er}}{\text{et}} * 100, \tag{1}$$

where ’er’ is the number of rated entries and ’et’ is the total number of entries.

- **Scalability.** Often datasets are very large and algorithms need to be able to keep up with this scale.
- **Popularity Bias.** Popular items are recommended more often than other items, making them even more popular and dragging out the long tail [22].
- **Grey Sheep.** Recommendations can be hard to generate for users with specific/different profiles.
- **Shilling Attacks.** False data may affect the recommendation process and needs to somehow be filtered out.
- **Diversity.** Recommendations need to be able to maintain a varied set of results to keep users happy (e.g. Batman 1 and Batman 2 lack diversity).

1.3.5 Multi-Objectiveness

So far we have discussed predicting the user ratings using collaborative filtering, however, in the context of marketplaces, other objectives relating to the other stakeholders have to be considered – this is a large research gap that this report focuses on. These objectives vary and could be optimised on their own from using collaborative filtering, or by simply generating an item-dependent ranking (e.g. profits for an item). However, the way in which these objectives are combined, optimised, and then converted into a final ranking is important. Most of the RecSys space is targeted towards a single objective but multi-objectiveness is becoming increasingly more important as commercial applications move away from user-centric models to multi-stakeholder models. The MOO strategy matters – it is not about just ’doing’ multi-objective optimisation, it is about choosing the best strategy. Furthermore, optimising for several complementary objectives actually improves each metric independently compared to just optimising that metric (proof). Optimising for competing metrics is not necessarily a zero-sum game [12].

Common approaches for MOO include basic weighting methods (the objectives are combined into one objective by a weighted sum) and ϵ-constraint methods (choose one objective to optimise and treat the other objectives as constraints with predetermined upper bounds) [23]. Pareto optimality is also explored in this report, where weightings are unknown.

2 Literature Review

As part of this project, an in depth literature review was undertaken to understand Recommender Systems (RecSys) and what the current state-of-the-art is surrounding multi-objective RecSys that are adapted for commercial marketplaces that are sparse. The general conclusion was that there is much literature surrounding single-objective RecSys, which is user-centric, however, multi-stakeholder RecSys still seems to be a relatively novel field. Furthermore, sparse marketplaces seem to be less of a focus, despite their commercial importance.

2.1 User-Centric Recommender Systems

There is a wealth of research around different user-centric RecSys approaches. These vary from collaborative filtering methods, to content based, to more niche approaches such as knowledge-based systems. Zhang *et al.* [8] summarises these approaches well.

Whilst these methods are effective, they lack the commercial insight to deal with sparse marketplaces – multiple objectives and little data to use for optimisation. An example is the Netflix Prize [11] which used a dataset of sparsity greater than 1.2% [24]. This report explores datasets with sparsities of around 6.5% (Restaurant Rec), 3% (Movie Lens), and most importantly, 0.35% (Yahoo Music) – nearly 3.5 times more sparse than the Netflix Prize dataset.

2.2 Multi-Stakeholder Recommender Systems

There have been several papers (in particular [25], [22] and [20]) which outline different methods for dealing with multi-stakeholder recommendation, in addition to outlining problems that these algorithms face.

Although these approaches correctly approach the problem by dealing with it from a multi-objective viewpoint, they have two primary downfalls, which this report focuses on. Firstly, they lack a specific focus on sparse datasets. Secondly, they do not explore how the effectiveness of different algorithms changes depending on the dataset features (i.e. size, sparsity etc.) – there is no ’silver bullet’ algorithm.

The following three papers explore multi-stakeholder recommendation, and illustrate the current research gaps that this report tackles.

2.2.1 Paper 1: *’Multi-Stakeholder Recommendation: Applications and Challenges’*

Zheng [25] focuses on the challenges that Multi-Stakeholder Recommendation systems face and their primary applications. Reciprocal recommendations are used to illustrate that the users and the items in a system both have interests that need to be satisfied and hence they may have the same utilities – this may help create better results in the recommendation process. The term ’utility’ is used instead of ’objective’. Group recommendations can help to improve recommendations by bundling user’s who are perceived as being similar. The final research topic is the exploration of correlation among utilities – the value in one utility may affect the value in other utilities. Within all these ideas, a few applications are given, in particular social networks and *Tripadvisor.com*. Zheng *et al.* builds on this by discussing exact implementation using scalarization methods, evolutionary algorithms and Pareto optimality in a tutorial-based paper [17].

Zheng excludes an explicit analysis of different algorithms and only broadly talks about different concepts without fully testing them across different datasets. Furthermore, marketplaces which are sparse are not considered. This report uses data and results to assess different algorithms in the presence of sparsity – this report directly tackles a research gap.

2.2.2 Paper 2: *’Popularity Bias in Multi-Stakeholder Recommendation’*

Himan [22] takes an in-depth look at how popularity bias affects RecSys, in particular with multi-stakeholder recommendation. General challenges of recommendation are highlighted, in particular, ensuring different interests of the stakeholders are met, diversity of results and the long tail (i.e. popularity bias). Although a couple of medium sparse datasets are used to conduct experiments with sparsity 5.4% and 1.7%, both of these are significantly higher than the Yahoo Music dataset which has a sparsity of around 0.35%, which is what is one of the datasets analysed in this report. Himan considers k-nearest neighbours, RankALS and a single matrix factorisation algorithms.

Although the problems of sparsity and popularity bias are related, they are still separate issues. Tackling popularity bias creates fairer rankings which predominantly benefits the vendor (whilst also helping the user get more varied recommendations), whereas tackling sparsity improves the raw rating predicting capacity of a Recommender System in order to ultimately improve user satisfaction. Himan does not compare the effectiveness of different algorithms on different dataset environments. This report directly fills this important research gap by ensuring algorithms are tested in environments of differing sparsity in addition to other features (such as size, range of ratings etc.).

2.2.3 Paper 3: *’A Novel Multi-Objective Evolutionary Algorithm for Recommendation’*

Cui *et al.* [20] develops and tests a novel multi-objective evolutionary algorithm for multi-objective RecSys. Cui *et al.* analyses the algorithm in a similar manner to the methods set out in this report. The algorithm is tested on the Movie Lens dataset [4], but the dataset is adapted to have a sparsity of 6.3% where each user has rated at least 20 items.

There are some evident research gaps. The first one being that Cui *et al.* does not consider the multi-**stakeholder** approach and only considers objectives that affect the user. These objectives tend to be more complementary – taking different **stakeholder** interests into account is more challenging since the interests, and hence the objectives, are more likely to compete. In addition to this, evolutionary algorithms are typically overly simplistic and produce worse results compared to other algorithms [21]. Finally, Cui *et al.* only looks at evolutionary algorithms in comparison and disregards other types of algorithms, such as latent factor models, which are known to be highly effective [26]. This report looks at five different sophisticated algorithms which vary in technical approach.

2.3 Summary

This report addresses the problem of how a recommender system can effectively operate in the context of a sparse marketplace based on the interests of multiple stakeholders, using collaborative filtering techniques. The current literature focuses more broadly on marketplaces which are not sparse enough to provide issues with common collaborative filtering algorithms. Although single-objective RecSys matrix factorisation is explored there is not much literature on the use of matrix factorisation techniques on sparse multi-stakeholder systems. Multi-stakeholder (or multi-objective) RecSys algorithms are briefly explored, but no in depth assessment is provided. Current literature does not outline how the selection of the MOO strategy greatly affects not only the output results, but also the system usability.

This report establishes a full multi-stakeholder recommender system which can effectively tackle sparsity as well as assessing what types of systems are best for specific dataset environments, to create a final output ranking. Multi-Objective Optimisation methods are compared along with an in-depth analysis of the objectives for the business and vendor, as well as the user. Finally, this report discusses novel extensions to improve the algorithms and the system approach even more.

3 System Overview

3.1 Introduction

3.1.1 Process

There are two parts of the multi-stakeholder RecSys approach – the collaborative filtering part, and then the multi-objective optimisation part which is used to form a final ranking. The distinction between these parts is critical to understanding how the whole system fits together. Figure 2 shows how the whole RecSys approach fits together to form a whole system – the collaborative filtering part is in the middle where the user objective ranking is formed. The goal of experimentally exploring algorithms is to understand their effectiveness within multi-stakeholder RecSys, initiating insights into the best algorithms to use under different settings. This is all in the presence of sparsity. The final ranking is formed by applying multi-objective optimisation to the three different objectives from each of the stakeholders:

- the business objective which originates from the business's interests,
- the user objective (user satisfaction) which originates from the user's interests, and finally,
- the vendor objective which originates from the vendor's interests.

Note that the MOO is achieved post collaborative filtering since only the user objective is wholly user-dependent (i.e. user ratings matter). The diagram in Figure 2 highlights this by showing that both the business and vendor objectives are user **independent**. As discussed previously, the most important system outcome is ensuring the user is happy, as this maximises long term value of the customer, which automatically keeps the business and vendor happy [17]. As a result, much of this chapter focuses on the collaborative filtering part of the system.

In MOO, objectives can either be complementary or competing. In some instances, optimising for complementary objectives has shown to improve the performance of each individual objective [12]. In some commercial cases, there can be side-stakeholders in addition to the three main ones e.g. the Deliveroo rider for Deliveroo [27].

3.1.2 Algorithmic Approach

Although the algorithmic approaches for both the collaborative filtering and the MOO part are important, the most important objective is user satisfaction – ensuring ratings are accurately predicted. As a result

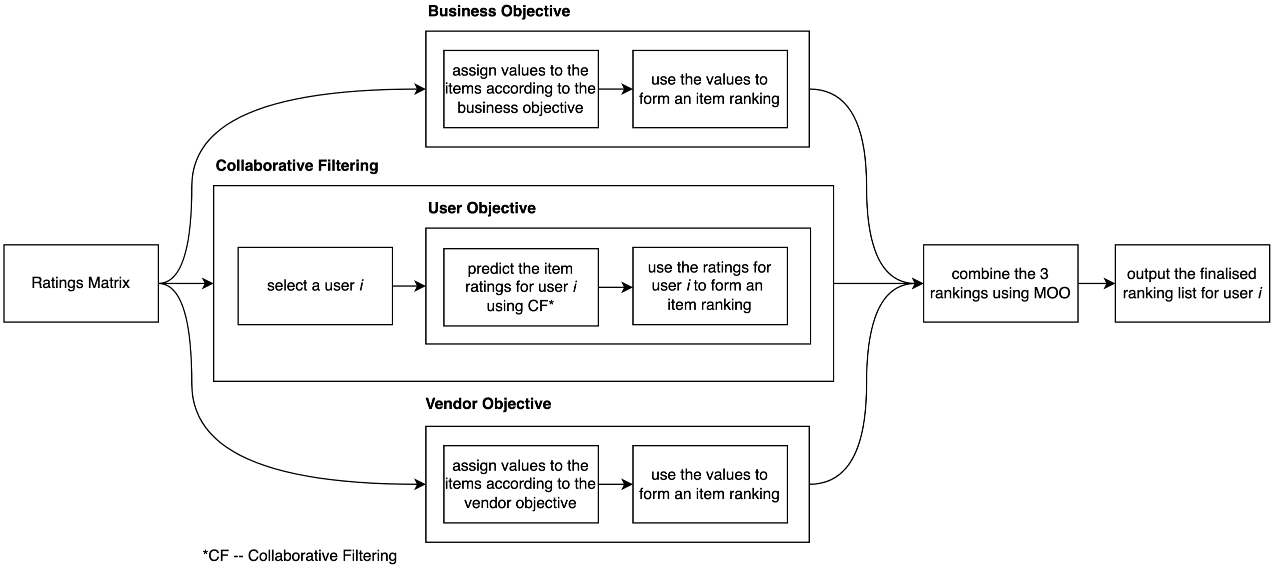

Business Objective
assign values to the items according to the business objective
use the values to form an item ranking
Collaborative Filtering
User Objective
Ratings Matrix
select a user *i*
predict the item ratings for user *i* using CF*
use the ratings for user *i* to form an item ranking
combine the 3 rankings using MOO
output the finalised ranking list for user *i*
Vendor Objective
assign values to the items according to the vendor objective
use the values to form an item ranking
*CF -- Collaborative Filtering

Figure 2: Flow chart of how the overall multi-stakeholder RecSys system fits together, highlighting the three different objectives that are combined to form an output ranking for user *i*.

much of this report, including the technical analysis, focuses on this objective, and hence, focuses on collaborative filtering.

There are three different **collaborative filtering** algorithms that are compared – k-nearest neighbours, matrix factorisation and naïve Bayes. K-nearest neighbours (KNN) is a neighbourhood based algorithm where the *k* most similar user rows/item columns are compared to the row/column of the entry that is being predicted, and then combined to form a prediction [13]. The naïve Bayes-based model algorithm is a generative approach that uses Bayes' theorem [18]. The final algorithm is an industry-wide adopted approach known as matrix factorisation which uses latent factor models (LFMs) [28]. Within matrix factorisation, unconstrained batch, stochastic and SVD++ are analysed. All the collaborative filtering algorithms are discussed in more depth as the report progresses.

Multi-objective optimisation involves ranking a set of items for a specific user, for each objective, and then combining them in the most optimal way. This is briefly touched on by Zheng [25], but this report draws insight into combining methods to form an effective hybrid MOO system. There are various different ways of combining these rankings, the ones that are explored are weighted methods and ϵ-constraint methods [23]. Weighted methods are straightforward if the weights of the objective function are known. We assume that they are, but also discuss further considerations around when they are not known.

3.1.3 Collaborative Filtering Evaluation

Evaluating the collaborative filtering approach depends on three parts – the input parameters, the environment features and the output results [8]. The input parameters affect the output results across different environment features, for different algorithms. Input parameters are what we can change, these are algorithm dependent, such as the regularisation term for matrix factorisation [28]. Environment features are the characteristics that can be altered, such as the dataset sparsity. This is achieved by using three different datasets with differing environment features. Output results are the criteria that we can measure in order to assess the effectiveness of the algorithms, such as the root mean squared error (RMSE). The output results are the mathematical objectives that should be optimised in some way.

The collaborative filtering part of the system is evaluated offline. This involves using historical data to form a sparse ratings matrix, where only some users have rated some of the items, which is then split into a training (D_training) and test dataset (D_test). The training dataset is used to predict the same values that are present in the test dataset, forming a new prediction dataset (D_pred). The root mean square error (RMSE) is then measured between D_pred and D_test. The RMSE is essentially a technical proxy for measuring user satisfaction – if ratings can be predicted more accurately, then rankings more closely match the ground truth of what a user prefers, improving their satisfaction. Equation 4 indicates how the RMSE is calculated. Offline evaluation is common practice for assessing algorithms in closed contexts, however it has two main limitations – human factors are not taken into account and offline datasets are innately imperfect. Each dataset can be described by the following environment features:

- Dataset size: the number of total entries.
- Popularity Bias: the frequency at which ratings appear, which normally forms a long tail [22].
- Ratings: the range of ratings that a user can assign to an item.
- Sparsity: number of entries of the ratings that are rated (or nonzero) as a percentage of the total number of ratings in the ratings matrix.

Table 1 shows the input parameters that can be tuned and the output results that can be measured for each algorithm. Often in machine learning approaches, especially probabilistic based ones, the input parameters are learned through a validation dataset, however, in this report they were optimised by iteratively updating each parameter.

Algorithm	**Input Parameter/s**	**Output Result/s**
K-Nearest Neighbours	*k* top rows/col	Computation time, RMSE, no. of false entries
Naïve Bayes	Alpha (additive smoothing parameter)	Computation time, RMSE, no. of false entries
Matrix Factorisation	Step size, *k* (dimensions of U/V), Lambda (regularisation term)	Computation time, RMSE, no. of iterations till convergence

Table 1: A table showing the collaborative filtering input parameter/s and output result/s for each type of algorithm.

3.1.4 System Evaluation

The evaluation methods for the system are based around trying to replicate the exact scenario in which a real life user might interact with a marketplace. In such a case, under an instance-based online evaluation, there is no point predicting all the unknown values for all the users (or generating a ranking for all users) – only the unknown items for the user i are of interest.

In a similar fashion to evaluating the collaborative filtering element, the overall system has three key elements – the input parameters, the environment features and the output results. These elements differ compared to just the collaborative part of the system. These are all listed below and discussed in further depth as the report progresses.

Environment Features:

- Number of items: the number of items that need to be ranked for user i.
- Number of users: the amount of data that is present to help generate the ranking for user i.
- Additional collaborative filtering features: range of ratings, popularity bias, sparsity.

Input Parameters:

- User objective input parameters for each algorithm, as described in Table 1.
- Vendor objective input parameters.

- Business objective input parameters.

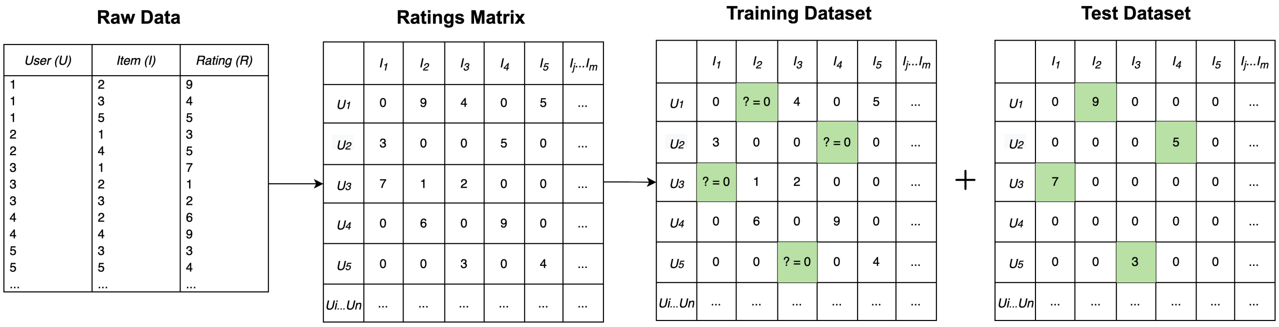
Output Parameters:

- Computation time: speed of final ranking generation.
- Criteria met for each objective: how close is the output ranking to each individual objective ranking (this is a direct reflection of the MOO strategy).
- Interpretability of results: can we tell the user why they are seeing a suggested result?

3.2 Datasets

3.2.1 Introduction

The datasets that are used are described as ratings matrices – users are represented along the rows, and items are represented by the columns. A specific entry in the ratings matrix (R_{ij}) is user i's rating of item j. The ratings matrix is formed from instances as shown in Figure 3, where n is the total number of users and m is the total number of columns – **the ratings matrix is hence an *n x m* matrix**. Note that unrated entries have the value of '0', and rated entries have an integer value greater than '0'.



Raw Data

User (U)	Item (I)	Rating (R)
1	2	9
1	3	4
1	5	5
2	1	3
2	4	5
3	1	7
3	2	1
3	3	2
4	2	6
4	4	9
5	3	3
5	5	4
...	...	...

Ratings Matrix

	I_1	I_2	I_3	I_4	I_5	$I_j...I_m$
U1	0	9	4	0	5	...
U2	3	0	0	5	0	...
U3	7	1	2	0	0	...
U4	0	6	0	9	0	...
U5	0	0	3	0	4	...
Ui...Un	...	...	...	...	...	...

Training Dataset

	I_1	I_2	I_3	I_4	I_5	$I_j...I_m$
U1	0	? = 0	4	0	5	...
U2	3	0	0	? = 0	0	...
U3	? = 0	1	2	0	0	...
U4	0	6	0	9	0	...
U5	0	0	? = 0	0	4	...
Ui...Un	...	...	...	...	...	...

+

Test Dataset

	I_1	I_2	I_3	I_4	I_5	$I_j...I_m$
U1	0	9	0	0	0	...
U2	0	0	0	5	0	...
U3	7	0	0	0	0	...
U4	0	0	0	0	0	...
U5	0	0	3	0	0	...
Ui...Un	...	...	...	...	...	...

Figure 3: A diagram of how the training and the test datasets are formed from the ratings matrix which is in turn formed from the raw data.

The ratings matrix is split into a training and test dataset by entries using cross validation. The main difference between how cross validation is conducted in classification versus collaborative filtering is that collaborative filtering maintains matrix dimensions across the folds and is conducted on entries, as opposed to rows. In Figure 3 the 12 rated entries have been split into a training dataset of 9 and a test dataset of 4. The entries that are removed from the training dataset have been labelled with a question mark, and become '0's since they are seen by the system as being unrated as these are the entries that need to be predicted. Once the collaborative filtering algorithm has predicted these values, the error is measured with the ground truth values from the test dataset.

Note the sparsity that is present in this example – using Equation 1, the ratings matrix has a sparsity of 48%. The datasets used in the actual analysis are far sparser in order to replicate real life problems – only some users have rated some items.

3.2.2 Environment Features

The differing environment features of each dataset enable the algorithms and general approaches to be tested in a variety of different settings. The current literature surrounding multi-stakeholder RecSys fails to do this and hence does not select the best algorithms for different use cases. Furthermore, Himan *et al.* [22] does not consider highly sparse datasets, such as the Yahoo Music dataset [5]. These features can be seen in Table 2. Note that the original ratings from the raw data were shifted to allow ‘0’ to represent an unrated entry.

Feature	**Restaurant Rec**	**Movie Lens**	**Yahoo Music**
No. of Users	138	1,200	500
No. of Items	130	1,200	500
Total No. of Entries	17,940	1,440,000	250,000
No. of Unique Ratings	3	10	43
Ratings	1,2,3	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	0.1 - 10 (in varying increments)
Sparsity	6.4716	3.4149	0.3532
No. of Rated Entries	1,161	49,175	883
No. of Rows with a Single or No Rated Entries	0	29	253
No. of Columns with a Single or No Rated Entries	0	277	419

Table 2: A table showing the different environment features for all three datasets.

There are four key environment features which vary in order to assess the system and the algorithms.

These features are innately changed by using different datasets as described in Table 2. The problems that are being assessed within these features are:

- **Scale**: some algorithms scale well with increased size and some do not – in this case increasing size means increasing the matrix dimensions, n and m.
- **Popularity Bias**: often collaborative filtering algorithms favour the items that are more popular (i.e. rated more), despite less popular items potentially being better recommendations. This leads to a slippery slope where only a subset of items are recommended to users, forming a long tail. The optimal RecSys approaches work against popularity bias. Although Himan discusses this [22], the literature fails to look at datasets with vastly varying degrees of popularity bias (as well as avoiding sparsity, a more important problem). Figure 4 shows the scaled long tails of the three datasets used – all of them have varying degrees of popularity bias.
- **Sparsity**: some algorithms are able to handle sparsity well, whereas others are not able to handle sparsity and produce false entries and a high RMSE. All current literature does not work with highly sparse datasets, such as the Yahoo Music dataset, which has a sparsity of 0.35%.
- **Range of ratings**: the range and the continuity (i.e. how close the ratings are to being continuous) of the ratings can affect an algorithm's performance. This again provides a novel dimension for this report, where current literature fails to analyse ratings that appear more continuous – this is more challenging to predict for and again the Yahoo Music dataset displays this challenge.

3.2.3 Restaurant Rec Dataset

The Restaurant Rec dataset [3] is the smallest dataset, but is also the least sparse. This would present favourably to most collaborative filtering algorithms and make the recommendation process easy for them. Although the limited range of ratings (1,2,3) should reduce the RMSE since they are less spread, this could also make ranking more difficult for exactly the same reason.

The long tail graph from Figure 4 shows that this dataset has the least severe case of popularity bias. The x-axis contains the ranked item number (scaled to 100) and the y-axis contains the frequency of those items (scaled to 100). Note that the item number is just a popularity rank and does not actually refer to the item number from the ratings matrix. For this dataset, all the items are rated at least 3 times.

Table 3 shows the three different objectives that the final ranking has to optimise for – user satisfaction,

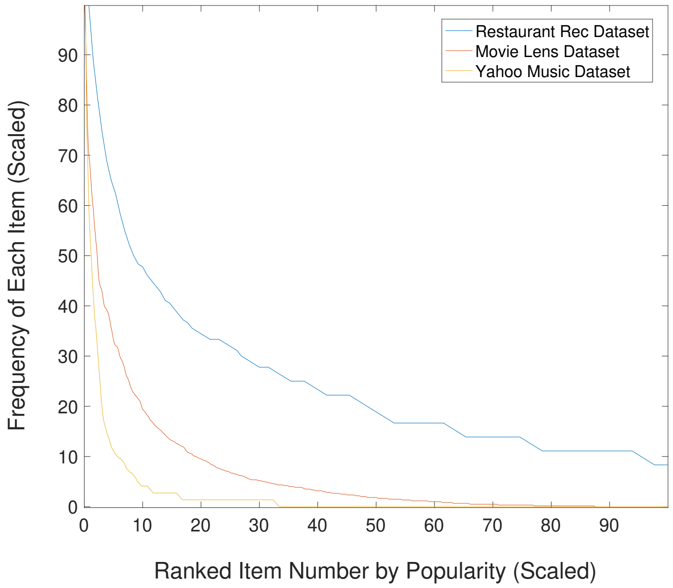


Figure 4: A graph of the combined long tails of the three different datasets, scaled along both axes but with the shape retained. The x-axis contains the item number (re-ranked by popularity) and the y-axis contains the frequency of each item. Yahoo Music clearly displaces the strongest long tail and hence the strongest popularity bias.

novelty and sustainability. Novelty is aimed at ensuring restaurants that have not been recommended previously to a user are favoured – this promotes equality across all the restaurants, who are the vendors in this case. The business objective of sustainability is formed from assigning random figures to each item (i.e. restaurant) which represent the average equivalent grams of CO_2 (gCO_2-eq) of a dish from that restaurant.

Stakeholder	**Metric**	**Method**	**Technical objective**
User	User Satisfaction	Accuracy of rating prediction.	RMSE
Vendor	Novelty	Assign weightings based on if an item is rated.	Higher weightings are given to items that are not rated by the user.
Business	Sustainability	av. gCO_2-eq number assigned to items (i.e. restaurants).	Minimise the av. gCO_2-eq.

Table 3: A table of the Restaurant Rec dataset objectives for each of the three stakeholders.

3.2.4 Movie Lens Dataset

The Movie Lens dataset [4] represents the most common form of marketplace data – a rating from 1 to 10, a large number of items and users and it is also quite sparse. This dataset also almost perfectly shows popularity bias in Figure 4. The Movie Lens dataset is the largest of the three and has the most rated entries at just under 50,000 entries.

Table 4 shows the three different objectives that the final ranking has to optimise for – user satisfaction, group fairness and revenue. Group fairness ensures that all the movie production companies (i.e. the vendors) are treated fairly. Within steaming video on demand (SVoD) marketplaces, such as the ones the Movie Lens datasets emulates, longer content tends to generate more revenue compared to shorter content i.e. Netflix would rather an individual watches a movie with total content length of 3hrs compared to a short movie of length 1hr. The business objective thus favours longer content in the final ranking.

Stakeholder	**Metric**	**Method**	**Technical objective**
User	User Satisfaction	Accuracy of rating prediction.	RMSE
Vendor	Group Fairness	Assign the items randomly to groups.	The top items should contain a single item from each group.
s Business	Revenue (Time Watched)	Assign movies a length of time.	Rank based on maximising the potential movie length watched.

Table 4: A table of the Movie Lens dataset objectives for each of the three stakeholders.

3.2.5 Yahoo Music Dataset

The Yahoo Music dataset [5] presents the hardest environment for collaborative filtering to accurately predict ratings for, and hence, also accurately predict rankings (the overall system). It is an extremely sparse dataset – around 10 times more sparse than the Movie Lens dataset and around 20 times more sparse than the Restaurant Rec dataset. It is a medium size dataset, yet has more columns and rows with only one or none entries in than the Movie Lens dataset, and fewer rated entries than the Restaurant Rec dataset as quantified in Table 2. Figure 4 also shows how this dataset is heavily skewed towards a few items ($\sim$50 items when unscaled), whilst the remaining items have few ratings – a perfect example of heavy popularity bias that creates a long tail.

Table 5 shows the three different objectives that the final ranking has to optimise for – user satisfaction, serendipity and time listened. Serendipity is another way of ensuring that vendors are treated fairly by showing users items that are non obvious. This is a perfect example of complementary objectives where although this objective has the vendor's interests in mind, it also benefits the user. The business objective of time listened is based on the principle that the longer a song is, the higher the potential listening time is for a user and hence the higher the potential revenues are (assuming increased revenue is a direct result of increased listening time).

Stakeholder	Metric	Method	Technical objective
User	User Satisfaction	Accuracy of rating prediction.	RMSE
Vendor	Serendipity	Non-obvious items from the popularity bias graph are ranked higher.	Rank based on assigned popularity bias ranking.
Business	Revenue (Time Listened)	Assign songs a length of time.	Rank based on maximising the potential time listened.

Table 5: A table of the Yahoo Music dataset objectives for each of the three stakeholders.

3.3 Collaborative Filtering

The following section outlines how the chosen collaborative filtering algorithms work mathematically, along with the Matlab [29] implementation. Note that nonzero entries and rated entries are used interchangeably.

3.3.1 Introduction

Notation

R : ratings matrix, $\tilde{R}$: prediction ratings matrix

(i, j) : row-column co-ordinates

$$(i,j) \in T : \text{the set of row-column co-ordinates of nonzero values in } R$$

$$(i,j) \in S : \text{the set of row-column co-ordinates in the test dataset (nonzero by definition)}$$

$$R_{ij}, \forall(i,j) \in S : \text{test dataset, } \tilde{R}_{ij}, \forall(i,j) \in S : \text{prediction dataset}$$

$$E_{ij} = \tilde{R}_{ij} - R_{ij}, \forall(i,j) \in S : \text{error matrix} \tag{2}$$

$$e_{ij} = \tilde{r}_{ij} - r_{ij}, (i,j) \in S : \text{error entry} \tag{3}$$

Cross Validation

`cross_validation.m` works by doing the following. Locate all the nonzero values in R using line 1 from Listing 1. Next, form a $|T|$ x 3 matrix containing the row position of each nonzero value, the column position of each nonzero value and then the nonzero value itself (line 2, Listing 1). Randomly generate a set of $|T|$ numbers as a list. Split the list into the specified number of folds, forming P new lists. Form P new matrices by using each list to index the nonzero values and place them in a new n x m zeroes matrix (`zeros(n,m)`), where each matrix has $|S|$ nonzero values in. Throughout the experimentation, 4 folds were used (i.e. $P = 4$).

```
1 [i,j,v] = find(input_ratings_matrix); % location of nonzero values
2 cross_val_info = [i j v]; % matrix of nonzero values, [row column value]
```

Listing 1: Cross validation

Training and Test Dataset Formation

Combine the first $1{:}P-1$ folds to form the training dataset. The P^{th} dataset is the test dataset. The folds are rotated P times, such that each fold matrix forms the test dataset once. The output results are then averaged.

RMSE

The root mean square error (RMSE) is calculated using,

$$\text{RMSE} = \sqrt{\frac{\sum_{(i,j)\in S} e_{ij}^2}{|S|}}, \tag{4}$$

where e_{ij} is the entry error and is calculated in Equation 3.

3.3.2 K-Nearest Neighbour

The k-nearest neighbour (KNN) algorithm relies on using the similarity function to understand how close the entry's row/column is to other rows/columns. A user-based approach looks at the rows, whilst an item-based approach looks at the columns. The disadvantage of using a user-based approach is that any item correlation information is not taken advantage of, and the disadvantage of using an item based approach is that any user correlation information is not taken advantage of. Hence, both are assessed separately, in addition to as a combined version, over the datasets. KNN is an instance based approach, as previously mentioned – each time a new prediction needs to be made for a new entry the entire algorithm needs to run, i.e. a pre-calculated model cannot be applied.

In order to form the prediction matrix, the prediction entry for each nonzero value in the test dataset has to be calculated. The algorithmic steps to predict each value ($r_{ij}, (i,j) \in S$) are as follows, for $i = u$ and $j = y$, predict r_{uy}:

1. ***Calculate the similarity.***

Firstly let us consider the user-based similarity. Calculating the similarity between two rows involves calculating the Pearson correlation. Other similarity functions can be used, such as the cosine similarity, however the Pearson correlation is more discriminative and produced better results on early experimental tests.

Let I_i be the set of column indexes for which user i has specified a rating (i.e. has a nonzero value). If user u is the target user of the entry being predicted and user v is the row that is being compared (which by definition must have it is y^{th} column entry rated), then the set $I_u \cap I_v$ contains the indexes of the entries that have been rated by both users – these are the values that are of interest when calculating the Pearson correlation. The implementation of this logic can be seen in Listing 2, line 2. If column index j is in set $I_u \cap I_v$, then the corresponding ratings, r_{uj} and r_{vj}, are added to a matrix: `rows_matrix`.

```
1 for n_entry_comp = 1:size(D_training,2) % iterate through the entries in the 2 rows being compared
2    if (D_training(n_row,n_entry_comp) ~= 0) && (D_training(row_pos,n_entry_comp) ~= 0) % check if both entries are nonzero
3        nn_entry = D_training(n_row,n_entry_comp); % the nearest neighbour entry
4        comp_entry = D_training(row_pos,n_entry_comp); % the entry being analysed
```

```
        rows_matrix = [ rows_matrix ; nn_entry comp_entry]; % assign to a matrix that contains all the paired nonzero entries
    end
end
```

Listing 2: Nearest Neighbour User-Based Similarity

Once this is complete and all the values r_{uj} and r_{vj} (where $j \in I_u \cap I_v$) have been compared, the Pearson correlation between rows u and v is,

$$\text{pearson}(u,v) = \frac{\sum_{j \in I_u \cap I_v} (r_{uj} - \mu_u) \cdot (r_{vj} - \mu_v)}{\sqrt{\sum_{j \in I_u \cap I_v} (r_{uj} - \mu_u)^2} \cdot \sqrt{\sum_{j \in I_u \cap I_v} (r_{vj} - \mu_v)^2}}, \tag{5}$$

where the means of rows u and v are,

$$\mu_u = \frac{\sum_{j \in I_u \cap I_v} r_{uj}}{|I_u|}, \mu_v = \frac{\sum_{j \in I_v \cap I_v} r_{vj}}{|I_v|}. \tag{6}$$

The easiest method of calculating the the item-based similarity is by taking the user-based similarity of the transpose of the ratings matrix, R^T. In the implementation, to avoid the additional complexity of transposing a large matrix, the item-based similarity was directly calculated from the columns.

2. *Calculate the prediction from the top-k similar rows/columns.*

Once the correlations have been calculated between u and all the other qualifying users, they are ordered from the highest correlation to the lowest. The top-*k* rows are then selected – let k be the set of these rows. The user-based prediction for entry u, y is,

$$\tilde{r}_{uy,user} = \frac{\sum_{v \in k} r_{vy} \cdot \text{pearson}(u,v)}{\sum_{v \in k} \text{pearson}(u,v)} \tag{7}$$

The equivalent for the item-based prediction of entry u, y can be calculated by again taking the transpose of the ratings matrix, R^T, and using Equation 7.

3. *Form the prediction ratings matrix.*

The final prediction ($\tilde{r}_{uy}$) is calculated by averaging the user and item predictions. This can be seen in line 2 of Listing 3. Listing 3 then shows how the predicted value is added to a blank matrix: `pred_test`. This happens for all values in S, that are valid, in order to form the prediction matrix. Some entries cannot be calculated due to a lack of information, especially with sparse datasets, and so these entries are left blank (logic in row 3), in addition to being counted (row 6). In an ideal world, the false entries counter is 0, however, marketplaces work with sparse datasets which can often result in too little information to calculate entries.

```
1  pred_test = zeros(n,m); % form a blank nxm prediction matrix
2  pred_entry = (user_pred_entry + item_pred_entry)./2 % calculate the prediction entry
3  if (isnan(pred_entry) == 0) && (isinf(pred_entry) == 0) % check the entry is valid
4     pred_test(row_pos,col_pos) = pred_entry; % add pred_entry to the prediction matrix
5  else
6     false_entries = false_entries + 1; % if the entry is not valid, increase the counter
7  end
```

Listing 3: Nearest Neighbour Forming the Prediction Matrix

From here, the RMSE can be calculated using Equation 4.

3.3.3 Naïve Bayes

The naïve Bayes model is an item-based generative model where the items can be treated as features and the users can be treated as instances, similar to classification [18]. The naïve Bayes algorithm relies on maximising the posterior probability of an entry having a certain value given the ratings in its row, using Bayes' theorem. In order to form the prediction matrix, the prediction entry for each nonzero value in the test dataset has to be calculated. Let $i = u$ and $j = y$, the goal is to predict r_{uy}. Let I_i be the set of column indexes for which user i has specified a rating (i.e. has a nonzero value). Let v_s indicate a possible discrete rating. The posterior is calculated using Bayes' theorem,

$$P(r_{uy} = v_s | \{r_{uj}, \forall j \in I_u\}) = \frac{P(r_{uy} = v_s) \cdot P(\{r_{uj}, \forall j \in I_u\} | r_{uy} = v_s)}{P(\{r_{uj}, \forall j \in I_u\})}, \tag{8}$$

where $\{r_{uj}, \forall j \in I_u\}$ denotes the set of nonzero ratings for user u.

Since we are trying to find the value v_s which maximises the posterior, this problem can be further simplified to just maximising the product of the prior and the likelihood,

$$P(r_{uy} = v_s | \{r_{uj}, \forall j \in I_u\}) \propto P(r_{uy} = v_s) \cdot P(\{r_{uj}, \forall j \in I_u\} | r_{uy} = v_s). \tag{9}$$

The algorithmic steps to predict each value ($r_{ij}, (i, j) \in S$) are as follows:

1. ***Calculate the prior for each possible rating value.***

The prior probability, $P(r_{uy} = v_s)$, is calculated by dividing the number of occurrences of v_s (line 1, Listing 4) by the total number of nonzero values (line 2), in column y .

```
1 col_ratings_value_count = sum(col(:) == n_item_val); % count the frequency of vs in the col
2 col_ratings_count = nnz(col); % count the total number of nonzero entries in the col
3 item_prior = col_ratings_value_count ./ col_ratings_count; % divide to get the prior
```

Listing 4: Naïve Bayes Prior Calculation

2. ***Calculate the likelihood for each possible rating value.***

The likelihood, $P(\{r_{uj}, \forall j \in I_u\}|r_{uy} = v_s)$, is slightly more tricky to calculate and relies on the naïve assumption which is based on conditional independence between the ratings [18],

$$P(\{r_{uj}, \forall j \in I_u\}|r_{uy} = v_s) = \prod_{j \in I_u} P(r_{uj}|r_{uy} = v_s). \tag{10}$$

Each conditional probability value is calculated by dividing the number of rows that have their j^{th} value equal to r_{uj} (line 2 Listing 5), by the number of rows that have their y^{th} value equal to v_s (line 5), only counting rows that have a rating in their j^{th} column. If no user has rated the entry r_{uj} when r_{uy} takes a particular value v_s then the conditional probability $P(r_{uj}|r_{uy} = v_s)$ = 0. This leads to the entire likelihood being 0, in addition to the posterior – this is often the case with sparse datasets. The use of an additive smoothing parameter, α, removes this issue by adding a small amount to every conditional probability in the likelihood such that no value is ever 0 – the likelihood will always be nonzero. This can be seen in line 3 and line 7 of Listing 5.

```
1 % calculate the numerator: find similar rows, then sum the generated array
2 comp_value_vector_num = ismember(n_rated_val_col(:,1),n_item_val_col(:,1));
3 cond_prob_val_num = sum(comp_value_vector_num) + alpha;
4 % calculate the denominator: find similar rows, then sum the generated array
5 comp_value_vector_den = ismember(n_nonzero_val_col(:,1),n_item_val_col(:,1));
6 n_alpha = size(comp_value_vector_den,1); % count the additive parameter multiple
7 cond_prob_val_den = sum(comp_value_vector_den) + (n_alpha*alpha);
```

```
% divide the numerator by the denominator
cond_prob_val = cond_prob_val_num ./ cond_prob_val_den;
```

Listing 5: Naïve Bayes Likelihood Calculation

The conditional probability values (`cond_prob_val`) are then multiplied together to get the likelihood.

3. Select the rating value which maximises the posterior.

In order to select the value v_s that maximises the posterior, the priors and the likelihoods for each value v_s must be multiplied together. Then locate the highest likelihood prior product – the equivalent v_s rating is the entry's prediction,

$$\tilde{r}_{uy} = \text{argmax}_{v_s}(P(r_{uy} = v_s) \cdot P(\{r_{uj}, \forall j \in I_u\}|r_{uy} = v_s)). \tag{11}$$

4. Form the prediction ratings matrix.

In a similar fashion to KNN, the predicted value is then added to a blank matrix to form `pred_test`, as in Listing 3. Again, this only happens for valid predicted values – the false entries counter counts the number of values that cannot be predicted. From here, the RMSE can be calculated using Equation 4.

Note that the naïve Bayes algorithm predicts discrete integer ratings whereas the KNN algorithm predicts any value within a range. Naïve Bayes would thus be less effective on data that is more continuous, such as the Yahoo Music dataset.

3.3.4 Matrix Factorisation

Three different forms of matrix factorisation are explored in this report – unconstrained with batch updates, unconstrained with stochastic gradient descent and finally SVD++ (which is also unconstrained). There are no constraints on the factor matrices U and V in unconstrained matrix factorisation. Although this has the disadvantage of being less interpretable, the solution quality tends to be much better, especially when dealing with sparsity [30].

Latent factor models (matrix factorisation) are considered to be a state of the art technique in RecSys due to their accuracy and speed, especially in sparsity. LFMs rely on the fact that the data is correlated in some-way and can hence be represented by a low rank matrix because it has built in redundancies. The basic idea of dimensional reduction methods is to rotate the axis system, so that pairwise correlations between dimensions are removed. Each column of U (or V) is referred to as a latent vector or latent component, whereas each row of U (or V) is referred to as a latent factor. U is an n x k matrix and V is an m x k matrix, where k can be tuned and is related to the rank of R. u_{iq} and v_{jq} denote the entries, where $q \in \{1..k\}$. The basic idea of matrix factorisation is to use gradient descent to minimise the Frobenius norm of the difference between the training dataset and the recombined UV^T, which is essentially the error,

$$\text{minimise } J = \frac{1}{2}||R_{tr} - UV^T||^2 = \frac{1}{2}\sum_{(i,j)\in S} e_{ij}^2, \tag{12}$$

where R_{tr} refers to the training dataset.

1. *Randomly initialise the factor matrices, U and V.*

Before optimising U and V via gradient descent, they need to be randomly initialised – the implementation of this is shown in listing 6.

```
U = rand(size(D_training,1),rank_k); % initialise U
V = rand(size(D_training,2),rank_k); % initialise V
```

Listing 6: Initialise U and V

2. *Minimise the error using gradient descent.*

Gradient descent works by subtracting the gradient from the current values of u and v in order to converge on a solution, where the gradient is the partial derivative of J with respect to the decision variable. For basic batch gradient descent the gradients are,

$$\frac{\partial J}{\partial u_{iq}} = \sum_{j:(i,j)\in S} (e_{ij})(-v_{jq}),\ \forall i \in \{1...n\},\ \forall q \in \{1...k\}, \tag{13}$$

$$\frac{\partial J}{\partial v_{jq}} = \sum_{i:(i,j)\in S} (e_{ij})(-u_{iq}),\ \forall j \in \{1...m\},\ \forall q \in \{1...k\}. \tag{14}$$

Gradient descent stops after a certain number of iterations, or whenever the error ($E_{ij} = \tilde{R}_{ij} - R_{ij}, \forall (i,j) \in S$) gets within a certain defined limit. The methods for updating U and V are outlined below. Note that the values are initially stored in an intermediary value (U_{next} and V_{next}), so that the current iteration does not work with future predicted values. The step size for the gradient descent is a predetermined value represented by α and `step`.

For unconstrained **batch** matrix factorisation, all the entries can be updated at the same time using matrix representation, by working with U and V as opposed to u_{iq} and v_{jq}. This is shown in Listing 7.

```
U_next = U + (step*E*V); % calculate the next U
V_next = V + (step*E.'*U); % calculate the next V
```

Listing 7: Batch Gradient Descent

Stochastic gradient descent decomposes the linear error updates from the previous batch method to entry by entry updates. Listing 8, lines 1 and 3 show the entry by entry updates, and lines 2 and line 4 show the intermediary updates to the matrices.

```
1 u_next_entry = u_entry + (step * error_entry * v_entry); % calculate the next u entry
2 U_next(i,q) = u_next_entry; % assign new u entry to U_next
3 v_next_entry = v_entry + (step * error_entry * u_entry); % calculate the next v entry
4 V_next(j,q) = v_next_entry; % assign new v entry to V_next
```

Listing 8: Stochastic Gradient Descent

SVD++ takes into account the implicit ratings which are derived from explicit ratings – the fact that an entry is rated is treated as a '1' whilst unrated entries remain as '0'. The implicit user factor matrix is FY, where F is the normalised implicit matrix and Y is an m x k matrix that is to be optimised along with U and V [30].

λ is the regularisation term to prevent against over-fitting. The optimal λ is found by iterating through different values and finding the value that minimises the error. This is done generally and an extension would be to instead use another validation dataset split to learn λ for the specific cross validation split. The gradient descent equations to update u_{iq} and v_{jq} are adapted from the stochastic approach (equations adapted from [30] and [31]),

$$u_{iq}^{next} = u_{iq} + \alpha(e_{iq} \cdot v_{jq} - \lambda \cdot u_{iq}), \forall q \in \{1...k\}, \tag{15}$$

$$v_{jq}^{next} = v_{jq} + \alpha\left(e_{iq} \cdot \left[u_{iq} + \sum_{h \in I_i} \frac{y_{hq}}{\sqrt{|I_i|}}\right] - \lambda \cdot v_{jq}\right), \forall q \in \{1...k\}, \tag{16}$$

where I_i is the set of nonzero (i.e. rated) values for user i, as in previous collaborative filtering methods. h is the row number of matrix Y and in this case, $h \in I_i$. Finally, y_{hq} is a target entry in matrix Y.

The entries for the normalised implicit ratings matrix Y, y_{hq}, are updated using,

$$y_{hq}^{next} = y_{hq} + \alpha \left(\frac{e_{ij} \cdot v_{jq}}{\sqrt{|I_i|}} - \lambda \cdot y_{hq} \right), \forall q \in \{1...k\}, \forall h \in I_i. \tag{17}$$

3. Recombine the factor matrices to form the prediction matrix.

Once the iterations have terminated, the matrices can be recombined to find $\tilde{R}$. For unconstrained matrix factorisation using batch and stochastic gradient descent,

$$\tilde{R} = UV^T. \tag{18}$$

For SVD++,

$$\tilde{R} = (U + FY)V^T. \tag{19}$$

The prediction matrix can then be calculated by selecting entries that satisfy $(i, j) \in S$. The RMSE can then be calculated using Equation 4.

3.3.5 Further Extensions

It is worth understanding how these algorithms could be tweaked to improve their efficacy.

KNN can be extended by using inverse user frequency (IUF) which reduces the impact of the long tail by using specified weightings during the recommendation process. The weights are calculated using,

$$w_j = log(\frac{n}{|v_j|}), \forall j \in 1...n, \tag{20}$$

where v_j is the number of ratings for item j [32]. w_j is inserted into each summation from Equation 5.

Furthermore, any false entries can be roughly predicted using some approximation, such as by averaging all the other users' entries. This ensures that the entry is at least included somewhere in the final rankings.

The **Naïve Bayes** algorithm can be improved by incorporating a user-based approach and then taking a final prediction as a weighted function of the item-based and user-based approaches. This would help to use any of the information and correlation stored between the rows.

Matrix factorisation (SVD++ specifically) can be improved to include user (o_i) and item (p_j) biases. User biases account for general user behaviours e.g. a particular user who rates many items, and rates them highly, must be treated differently to a pessimistic user who rates items very harshly. A similar analogy can be used for item biases. These biases are held in additional columns in the factor matrices (U and V).

Combining algorithms to form a hybrid algorithm can prove to be an effective way to utilise the different advantages of the individual algorithms, whilst mitigating their disadvantages [13].

3.4 Multi-Objective Optimisation (MOO)

3.4.1 Ranking

Figure 2 shows how the three rankings which correspond to the three objectives, are formed and then combined to create the final output ranking. The best collaborative filtering algorithm is chosen from the initial collaborative filtering analysis. In order to form the ranking for the user objective, the approach differs from the analysis conducted to choose the optimum collaborative filtering algorithm. Primarily, cross validation is not conducted on the dataset, instead, a random user i is selected and then a ranking is formed for the user objective. Instead of rated items being predicted and then assessed, the goal now becomes to predict the unrated values (i.e. the ones with a value of '0' in the ratings matrix). Once the user has been selected, these values are predicted using the selected collaborative filtering algorithm, combined with the already rated items, sorted by value (line 2 Listing 9), and then turned into a ranking. This is what Listing 9 shows.

```
1 unordered_ranking = [1:size(pred_user_row,2) ; pred_user_row].'; % assign an item number
2 ordered_ranking = sortrows(unordered_ranking,2,'descend'); % sort the list by the value
3 user_ranking = [1:size(pred_user_row,2) ; ordered_ranking(:,1).'].'; % generate the final
      ranking list, [ranking item]
```

Listing 9: Forming the User Rank

This ranking is then combined using MOO with the other two rankings to form the final output ranking as a recommendation list to the user, Figure 5. The final ranking is evaluated in two ways. The first of which is by measuring the computation time. The second is by measuring how the final ranking fits with each individual stakeholder objective ranking. Although this is somewhat just a reflection of the MOO strategy taken, the goal is to in fact have the final ranking be better than expected since the objectives are complementary – optimising for multiple objectives is better for each individual metric compared to just optimising it directly [12].

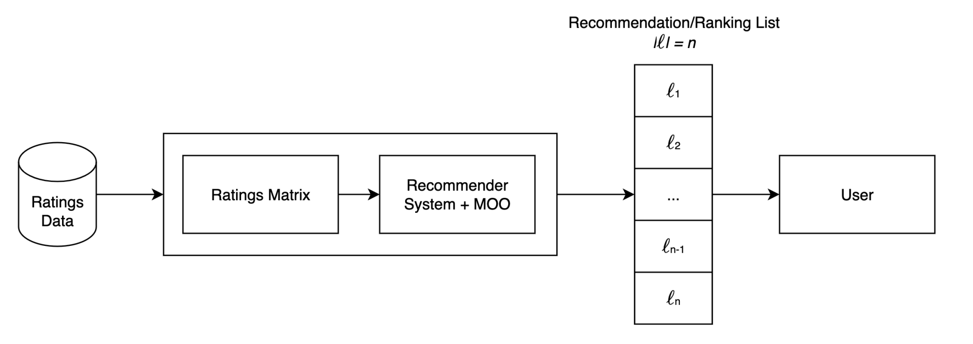


Figure 5: A flow chart of how the final ranking list is produced for user *i* from the ratings data. *Adapted from Figure 2.1 [22].*

3.4.2 MOO with Known Parameters

When the parameters that dictate how much each objective should contribute to the overall ranking are known, the multi-objective optimisation is quite straightforward.

The first approach is a relatively simplistic, but widely used, multi-objective strategy that uses a **weighted method** [23]. The objectives are combined using a weighted equation, where the weights (w_k), i.e. the parameters, are known. The final rank is produced for each item j using,

$$\text{rank}_j^{output} = \sum_{k=1}^{3} w_k \cdot \text{rank}_{jk}, \tag{21}$$

where k is the objective number. Once each item has its new rank, an overall rank is formed as the final output rank.

The second approach is to use **ϵ-constraint methods** which optimises one objective whilst treating the other objectives as constraints with predetermined upper bounds [17]. In the case of recommendation systems and rankings, for example, the business objective could be the fixed objective whilst the user objective has a ranking error upper bound ($target\_error_{max}$). A ranking error in this case can be defined as the equivalent root-mean squared error between the ranking of an item across two different objectives,

$$rank\_error = \sqrt{\frac{\sum_{j=1}^{m}(\text{rank}_{jk}^{fixed} - \text{rank}_{jk}^{constrained})^2}{m}}, \tag{22}$$

where j represents an item , k represents an objective and m is the total number of items. The constrained objective (rank$_{jk}^{constrained}$) can be treated as the ground truth, whilst the fixed objective (rank$_{jk}^{fixed}$) can be treated as the predicted value. The ranking terminates as soon as the rank error falls below the $target\_error_{max}$ – this ranking then becomes the final output ranking.

3.4.3 MOO with Unknown Parameters

If the parameters, such as the weightings or the $target\_error_{max}$, are not known, then there are several logical approaches that can be used. In this report both MOO methods where parameters are known and methods where parameters are unknown are analysed.

One potential method to use involves Pareto optimality – different combinations of the parameters are trialled and solutions are plotted along a 3 axis plot, forming a Pareto front. The core idea of a Pareto front is that optimising one objective will always result in the other objective being made worse off. So some method of choosing a solution along the front must be implemented which forms a compromise between the objectives. Non-dominated solutions sit along the front, whilst dominated solutions are the unoptimised points which sit after the curve. A Pareto front makes selecting a solution much more straight forward.

Another class of methods could be to learn the parameters through learned optimisation – this is especially effective for online systems which can gradually learn the right parameters over time. One specific approach could be to use empirical risk minimisation through a loss function [33].

4 Experimental Results

4.1 Implementation

All the results were generated using algorithms that were my own Matlab [29] implementation – no machine learning, recommendation or related libraries were used. In total 51 scripts (i.e. files) were used to produce the three collaborative filtering algorithms and the MOO ranking algorithms, totalling just over 2100 lines of code. All code was my own work [34]. Thanks to Edwin Lock, my secondary supervisor, for providing two brief code reviews on two different scripts to aid with minor bug fixing.

4.2 Restaurant Rec Dataset Results & Analysis

4.2.1 Collaborative Filtering Results

Table 6 shows the collaborative filtering output results of running the various different algorithms under their optimised input parameters, for the Restaurant Rec dataset [3]. Where ’CC’ refers to the convergence criterion, ’Av’ refers to the average value and ’SD’ refers to the standard deviation. Each algorithm was run 3 times and the average of each output result was taken.

Collaborative Filtering Algorithm		**Input Parameters (s)**	**Output Results**				
			RMSE		**Computation Time (s)**		
			Av	**SD**	**Av**	**SD**	**Per Row Av**
KNN		*k* = 35%	0.8694	0.0143	0.2714	0.0283	***0.0020***
Naïve Bayes		alpha = 0.2	0.8988	0.0459	0.8381	0.0101	0.0061
Matrix Factorisation	**Batch**	step = 0.001 *k* = 6 CC = 0.000001	0.7881	0.0077	0.2683	0.0679	0.2683
	Stochastic	step = 0.001 *k* = 8 CC = 0.000001	0.7863	0.0226	***0.1768***	0.0290	0.1768
	SVD++	step = 0.001 *k* = 3 CC = 0.000001 λ = 0.2	***0.7510***	0.0092	0.9758	0.0687	0.9758

Table 6: A table showing the collaborative filtering input parameter values and output results for the Restaurant Rec dataset, across the five different algorithms.

Before moving onto multi-objective optimisation, the best collaborative filtering algorithm needs to be chosen, given the results in Table 6. When comparing the KNN algorithm and the naïve Bayes algorithm it is clear that KNN is both quicker and has a lower error. The downside of KNN is the presence of false entries where there is not enough data (due to the sparsity) for the algorithm to predict an entry. In these cases, the correlation between rows or columns could not be calculated and the algorithm produced a *nan* output. On average, 78.87% of the output entries were false. On average, around 50% more item-based entries were false compared to user-based entries – false entry outputs were ones where both the item-based and the user-based outputs were false.

Although all three matrix factorisation methods are more accurate than KNN and naïve Bayes, especially SVD++ which has an error of only 0.7510, they have longer computation times per row. The per row computation time is a measure of how long it would take to create rating predictions for a single user and hence how long it would take to produce a ranking list for a user. For matrix factorisation the entire matrix is estimated and then the required row is extracted – this means that the per row computation time is the same as the overall computation time. Stochastic gradient descent was shown to initially have the lowest error, by increasing *k*, however this was at the expense of stability – the standard deviation of results increased and in some instances, the algorithm failed to converge. This is unwanted in commercial systems which must be reliable and hence a lower value of *k* was chosen to increase stability. SVD++ had the highest percentage change from the initial RMSE to the final RMSE of 49.2% on average, and the largest number of iterations of 418 on average. For SVD++, the regularisation term decreased the error by 3%. Stochastic gradient descent was the quickest overall algorithm and the quickest matrix factorisation algorithm by row time computation. Across all three matrix factorisation approaches, the convergence criterion was kept consistent so that the algorithms were measured in a fair setting.

For this dataset, **SVD++ matrix factorisation** works the best since it has the lowest error. Although the computation time is more than the other algorithms, it is still not unreasonable, and in fact it could be made quicker through pre-processing and having an offline phase which stores predictions.

4.2.2 Multi-Objectiveness

The user ranking was based on the collaborative filtering predictions, where each item was ranked after the unrated entries were predicted. The vendor ranking, aimed at favouring novel items to the user, preferenced items that were unrated by the user. A ranking was formed where each novel item contributed a rank of ‘1’ whilst items that were already rated contributed a rank of ‘2’. The business ranking was based on generating random gCO_2-eq figures and then ranking them from the least to the highest. These rankings were all combined for each item using **weighted MOO**, and then re-ranked based on the combined ranking figure.

Table 7 shows the results where the time to produce a final ranking was relatively quick at 0.3848s. Since user satisfaction is the most important metric, its weighting contribution was 0.8 whilst the other objectives had a weighting contribution of 0.1 each. This is why the user ranking error is much less than the business ranking error. Note that the vendor ranking error was just measuring the error compared to the fixed rankings of ‘1’ and ‘2’ and thus, as a number, it should not be compared to the ranking errors of the other objectives. The business ranking also has a significantly higher standard deviation compared to the other objectives.

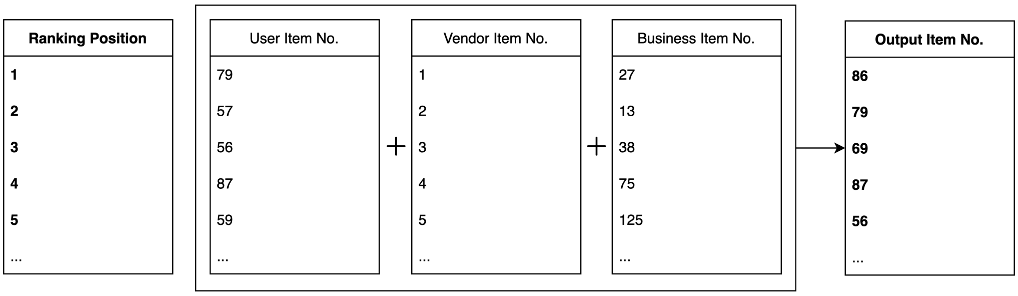
	Average	**Standard Deviation**
Computation Time (s)	0.3848	0.09495
User Ranking Error	4.547	0.1208
Vendor Ranking Error	74.58	0.007336
Business Ranking Error	48.25	1.054

Table 7: A table showing the Restaurant Rec multi-objective optimisation ranking results, averaged over 3 runs. Where the user ranking had a weighting of 0.8 and the vendor ranking and business ranking had weightings of 0.1 each.

Figure 6 shows an example set of rankings using a weighted MOO strategy (only the top 5 results are shown). The user ranking contributes heavily to the output rankings, which makes sense given its weighting contribution – 3 of the top 5 items in the user ranking (User Item No.) are included in the top 5 output ranking (Output Item No.).

4.2.3 Summary

The Restaurant Rec dataset presents the nicest environment for the collaborative filtering algorithms to predict ratings for – medium number of entries to analyse, low sparsity and a small range of ratings. SVD++ was the most effective since it utilises stored implicit information. In commercial applications, methods should be used to decrease the per row computation time, such as storing results or



Ranking Position	User Item No.	Vendor Item No.	Business Item No.	Output Item No.
1	79	1	27	86
2	57	2	13	79
3	56	3	38	69
4	87	4	75	87
5	59	5	125	56
...	...	...	...	...

Figure 6: An example of the top 5 Restaurant Rec multi objective optimisation rankings, where $w_{user} = 0.8$, $w_{vendor} = 0.1$ and $w_{business} = 0.1$.

pre-processing the offline stage.

The ranking approach was effective in combining the different objectives. The final output ranking list mostly contains items from the user ranking list – this is wanted since user satisfaction is the most important objective. However, although objectives may seem complementary on the surface, they will always have slightly different rankings and hence the MOO strategy must favour a particular objective. The required objective weights should be set by the system designer (or the business), however, a better strategy would be to learn the weights. Note that although user satisfaction is the most important objective, optimisation using all the objectives inevitably produces a better system outcome since all needs of the stakeholders are taken into account. This use case is reflective of an early stage marketplace e.g. Carbon Codes [6].

4.3 Movie Lens Dataset Results & Analysis

4.3.1 Collaborative Filtering Results

Table 8 shows the collaborative filtering output results of running the various different algorithms under their optimised input parameters, for the Movie Lens dataset [4]. All of the algorithms had larger errors for the Movie Lens dataset – this dataset is much harder to predict due to its larger scale, increased sparsity and larger range of ratings.

The naïve Bayes algorithm has the largest error and the largest computational time – it is evident that this algorithm scales poorly with increased dataset size. The KNN algorithm has a much lower percentage of false entries compared to how it performed on the Restaurant Rec dataset, of 0.45%. This is due to the dataset being nearly 50 times larger and hence more data is present. Batch matrix factorisation produced the lowest error but also had a large per row computation time. SVD++ proved ineffective on this dataset

Collaborative Filtering Algorithm		Input Parameter/s	Output Results				
			RMSE		Computation Time (s)		
			Av	SD	Av	SD	Per Row Av
KNN		*k* = 25%	2.0622	0.0091	124.0	4.3936	***0.1033***
Naïve Bayes		alpha = 0.2	2.5301	0.0189	559.2	13.20	0.4660
Matrix Factorisation	**Batch**	step = 0.0001 *k* = 25 CC = 0.0001	***2.0166***	0.0035	3.4714	0.0921	3.4714
	Stochastic	step = 0.001 *k* = 17 CC = 0.0001	2.0381	0.0183	***0.8630***	0.1426	0.8630
	SVD++	step = 0.001 *k* = 3 CC = 0.0001 λ = 0.05	2.9969	0.0180	22.36	1.3947	22.36

Table 8: A table showing the collaborative filtering input parameter values and output results for the Movie Lens dataset, across the five different algorithms.

since the increased sparsity renders the use of an implicit ratings matrix ineffective. Stochastic gradient descent had the largest change in RMSE of 44.5% over only 21.3 iterations, on average.

Although Batch has a lower error, its computation time per row of 3.4714 seconds is high. Per row, KNN has the lowest computation time of 0.1033s and an error comparable to batch matrix factorisation. Hence, for this dataset, **KNN** works the best and this is the algorithm that is used for the multi-objective optimisation. Figure 7 shows that a value for *k*, the percentage of top similar rows/columns selected, of 25% is optimal.

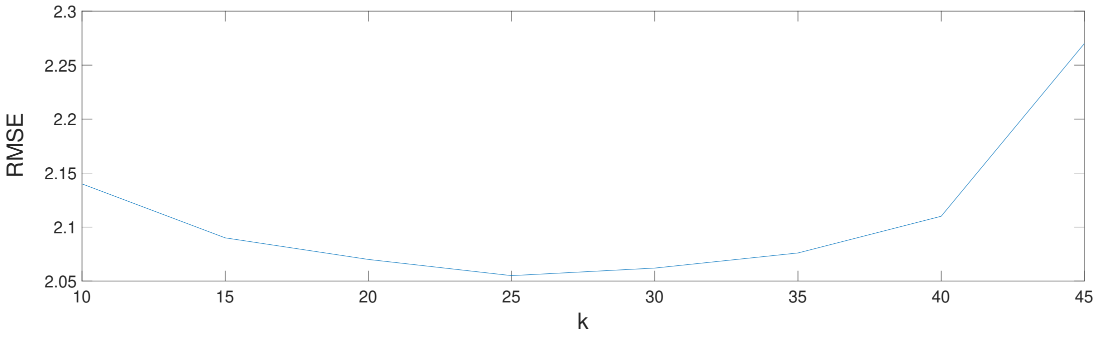


Figure 7: Graph of how the RMSE varies as *k* (the percentage of top similar rows/columns) is varied for the KNN algorithm, applied to the Movie Lens dataset. The x-axis contains the value of *k* and the y -axis contains the RMSE. *k* = 25 clearly minimises the RMSE.

In the context of recommending movies to users, basic contextual information can be used to help improve predictions of false output entries for KNN. For example, if a system knows that users prefer Christmas movies in December, then a false entry for a Christmas movie can be initially averaged over its row and column, and then the rating can be increased by a percentage that reflects the increased preference, if it is December – the ranking of this item will then reflect accordingly. This is an overly simplistic approach and only works in this case because the percentage of false entries is so small – commercial systems can use much more effective context-aware approximation methods.

4.3.2 Multi-Objectiveness

The user ranking was based on the collaborative filtering predictions, where each item was ranked after the unrated entries were predicted. Since KNN was used, any false entries were removed and the average of all the values in the row was used instead. The user and business objectives were combined using ϵ-constraint MOO, where the max user ranking error was fixed ($target\_error_{max}$), and the business ranking gradually tended towards the user ranking using interpolation until the $target\_error_{max}$ for the user ranking error, i.e. the convergence criteria, was met. The vendor ranking was based on group fairness – all of the items were equally assigned to 5 groups. Once the user and business rankings had been combined (call this the 'new ranking'), the first value from the new ranking in group '1' was given a ranking of '1', the second value in group '2' was given a ranking of '2' and so on, until all five groups had their highest ranking item assigned the corresponding rank. All the remaining values were given a rank of '6'. This new list was then combined with the output ranking between the user and business objectives using an equal weighted method (i.e. both had a weighting of 0.5).

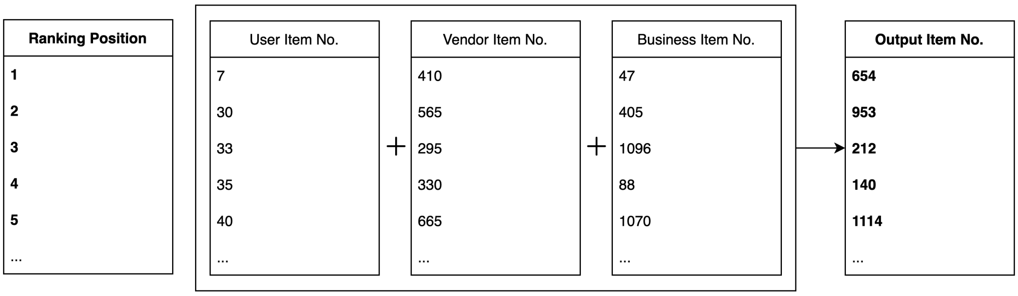
Table 9 shows the objective errors and the computation time of the multi-objective optimisation. The $target\_error_{max}$ was set at 70 leading to an average computation time of 1.125s. This is higher than expected given the per row computation time of only 0.1033s for KNN, and is caused by the gradient descent for ϵ-constraint MOO.

On average, bounce rate from a web page increases from 6% to 11% if the load time is longer than 2s (but less than 3s) [35] – 1.125s is acceptable, but less than 0.5s is ideal. The high vendor error highlights the conflicting rankings between the user ranking and the vendor ranking. The business ranking again has a very high standard deviation between repeats compared to the other objectives.

	Average	**Standard Deviation**
Computation Time (s)	1.125	1.006
User Ranking Error	69.25	0.2372
Vendor Ranking Error	690.7	0.04713
Business Ranking Error	48.25	2.781

Table 9: A table showing the Movie Lens multi-objective optimisation results, averaged over 3 runs. Where the $target\_error_{max}$ = 70 and the 'new ranking' and business ranking had equal weightings.

Figure 8 shows that no items from the top 5 user ranking (User Item No.) were included in the top 5 final output ranking (Output Item No.). This indicates that the $target\_error_{max}$ should be set lower to maximise user satisfaction – this is at the expense of the vendor ranking error which would increase further.



Ranking Position	User Item No.		Vendor Item No.		Business Item No.		**Output Item No.**
1	7		410		47		**654**
2	30		565		405		**953**
3	33	+	295	+	1096	→	**212**
4	35		330		88		**140**
5	40		665		1070		**1114**
...	...		...		...		...

Figure 8: An example of the top 5 Movie Lens multi objective optimisation rankings, where $target\_error_{max} = 70$, the number of vendor groups is 5.

4.3.3 Summary

The best collaborative filtering approach was KNN which had a low percentage of false entries – these could easily be removed by averaging the row values. An extension of this could be to average column values as well to create a better prediction for the false entries. Overall the error was still quite large with this dataset across all the algorithms – this is due to the increased sparsity.

The Movie Lens rankings showcased worse errors compared to the Restaurant Rec rankings – this is predominantly due to the Movie Lens dataset being larger and far sparser compared to the Restaurant Rec dataset, as opposed to any differences in MOO methods. In fact, ϵ-constraint MOO presents a better strategy for combining the different rankings since the $target\_error_{max}$ parameter can be learned as a single parameter.

4.4 Yahoo Music Dataset Results & Analysis

4.4.1 Collaborative Filtering Results

Table 10 shows the collaborative filtering output results of running the various different algorithms under their optimised input parameters, for the Yahoo Music dataset.

Collaborative Filtering Algorithm		**Input Parameter/s**	**Output Results**				
			RMSE		**Computation Time (s)**		
			Av	**SD**	**Av**	**SD**	**Per Row Av**
KNN		*k* = 40%	3.0184	0.0481	0.3528	0.077	***0.0007***
Naïve Bayes		alpha = 0.15	3.0402	0.1996	3.0301	0.1733	0.0061
Matrix Factorisation	**Batch**	step = 0.001 *k* = 35 CC = 0.001	2.1567	0.2465	0.3697	0.0985	0.3697
	Stochastic	step = 0.001 *k* = 29 CC = 0.001	***2.0466***	0.1309	***0.2444***	0.0791	0.2444
	SVD++	step = 0.0001 *k* = 10 CC = 0.001 λ = 0.25	3.3885	0.1959	4.1990	0.3199	4.1990

Table 10: A table showing the collaborative filtering input parameter values and output results for the Yahoo Music dataset, across the five different algorithms.

The prediction difficulty that all the collaborative filtering algorithms face with the Yahoo Music dataset is highlighted by the increased errors. This is due to the the increased sparsity ($\sim$ 0.3%) and the rating values appearing more continuous, i.e. 43 discrete rating values from 0.1 to 10, in varying increments. The values are not explicit user ratings but are lengths of listening time of a user for a particular song, which is then scaled and put in buckets – they are still referred to as ratings for the purposes of consistency.

Both the KNN algorithm and the naïve Bayes algorithm performed poorly by having large errors. Due to the tough environment features of this algorithm, the KNN algorithm had a 46% false entries rate – this is extremely poor and renders this algorithm completely ineffective for this dataset. SVD++ performed similarly to the Movie Lens dataset where both the error and the computation time were high. SVD++ gradient descent had the largest percentage change in RMSE of 45.1%, this can be seen in Figure 9.

The best collaborative filtering algorithm is therefore **stochastic gradient descent** due to its low error and quick per row computation time. Figure 9 shows the descents of the three different matrix factorisation algorithms – stochastic is clearly the best with the fewest number of iterations yet the smallest final error.

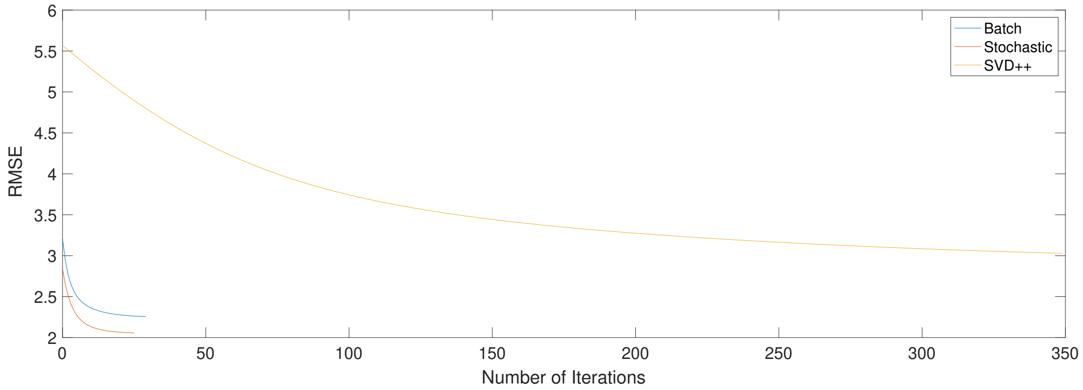


Figure 9: Graph of the descent of the three different Matrix Factorisation algorithms applied to the Yahoo Music dataset. The x axis contains the number of iterations and the y-axis contains the RMSE. Stochastic gradient descent produced the lowest error over the fewest number of iterations, on average.

4.4.2 Multi-Objectiveness

The MOO strategy explored for this final dataset focuses on parameter selection – how can all the data be used to better predict the right MOO parameter value? Pareto optimality considers how changing the parameter value varies each objective's ranking error. The user ranking is based on stochastic gradient descent collaborative filtering. The business objective generates random song lengths and ranks an item higher if the song is longer – this generates more revenue for the business. The vendor objective looks at serendipity – recommending non obvious items to the user and hence creating fairer recommendations. This is achieved by reverse ranking the items based on the long-tail graph.

In the approach taken here, ϵ-constraint generates the Pareto optimal solutions (i.e. the non-dominated solutions) which lie along a front. This is achieved in a similar way to the Movie Lens MOO method, where the $target\_error_{max}$ is the variable parameter and sets the max user ranking error along the x-axis.

The business and vendor rankings are initially combined equally using a weighted strategy to form a new ranking, this ranking gradually tends to the user ranking using interpolation, until the user ranking error hits the convergence criterion set by $target\_error_{max}$. Figure 10 shows how the business ranking error and the vendor ranking error change as the user ranking error ($target\_error_{max}$) increases, to create a Pareto front of the non-dominated solutions. Generating a Pareto front allows the trade-off in

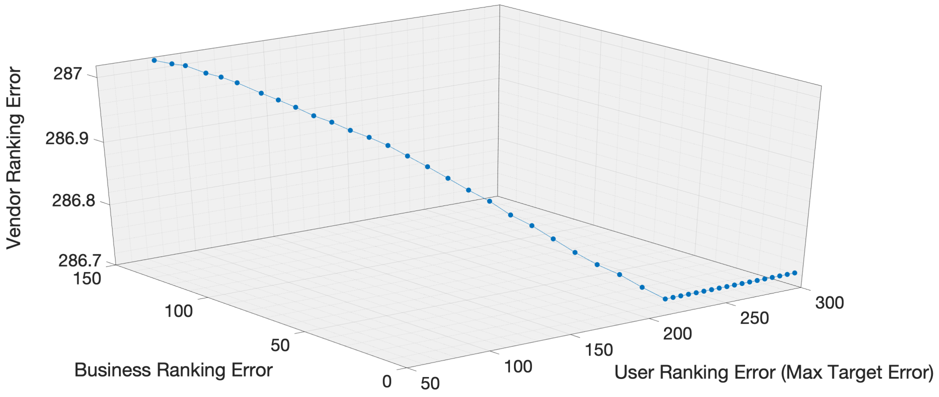


Figure 10: A graph showing the different objective ranking errors when the threshold for the user ranking error is varied by varying $target\_error_{max}$ through ϵ-constraint optimisation. The $target\_error_{max}$ should always be set lower than 230.

objectives to be visualised and illustrates how choosing a certain parameter value, i.e. $target\_error_{max}$, affects the solutions. In this scenario after $target\_error_{max}$ was greater than 230, any further increase in $target\_error_{max}$ did not decrease the vendor ranking error nor the business ranking error – hence, the user ranking error should always be set lower than 230. The vendor ranking error did not change much and only varied from 286.9 to 287.1. However, the business ranking error varied significantly. Based on this information, a value for $target\_error_{max}$ could be chosen.

An alternative approach to help learn the best parameter value could be to vary the parameter value and then to use a further product metric (e.g. revenue, daily active users, bounce rate etc.) to select the best value across that product metric. Note that a full Pareto surface could be created by further separating out the vendor and business rankings and then by varying two parameters instead of one, however, there is not much need for this since really we only want to set the user satisfaction i.e the user ranking error.

The method outlined in this Section uses a combination of many different approaches – weighted methods, ϵ-constraint and Pareto optimality. This is the optimum way to achieve MOO out of all of the three MOO methods since it allows for ease of parameter value visualisation, and hence selection.

4.4.3 Analysis

Overall this dataset was the toughest for the collaborative filtering algorithms to handle, including batch – this can be seen from the high errors (Table 10). In general, the errors were higher as a result of this and in fact SVD++ proved to be completely ineffective. The best collaborative filtering approach was stochastic gradient descent. Although the non-matrix factorisation methods (i.e. KNN and naïve Bayes) had significantly lower per row average computation times, stochastic gradient descent still had a reasonable time.

Utilising Pareto optimality, combined with other MOO methods, provides a good visualisation of the objective trade-offs and enables smoother parameter selection.

Since the Yahoo Music dataset is extremely sparse and has a wide range of possible ratings – the whole system in general (both collaborative filtering and MOO) performed worse. For this reason, the novel exploration focuses on generating better approaches for datasets and marketplaces of a similar level of difficulty.

4.5 Summary

It is evident that the collaborative filtering algorithm of choice can heavily affect the RMSE, computation time, the predicted ratings, and hence the rankings and the end satisfaction of a user. The Literature Review at the beginning of this report outlined how current research fails to holistically assess different algorithms for different dataset environments for multi-stakeholder RecSys. This section has not only analysed many different algorithms, but has also demonstrated the best algorithm when working with highly sparse datasets as stochastic gradient descent matrix factorisation.

Figure 11 summarises all the RMSE results from the collaborative filtering part of the experimentation. There was often a significant difference in both RMSE and computation time between the different algorithms. Overall matrix factorisation methods displayed lower errors and were generally quicker – they are known in industry as being very effective collaborative filtering solutions [26]. For less sparse datasets, i.e. Restaurant Rec, SVD++ proved to be very effective. However, SVD++ should not be used where sparsity is prevalent since the implicit matrix adds little information and in fact hinders the process – this was seen in the Yahoo Music dataset. Furthermore, SVD++ produced computation time results with a high standard deviation between runs i.e. the computation time was unpredictable. The best algorithms can be matched with dataset environment types:

- The Restaurant rec dataset: smaller, less sparse, few possible ratings – batch gradient matrix factorisation descent is optimal.

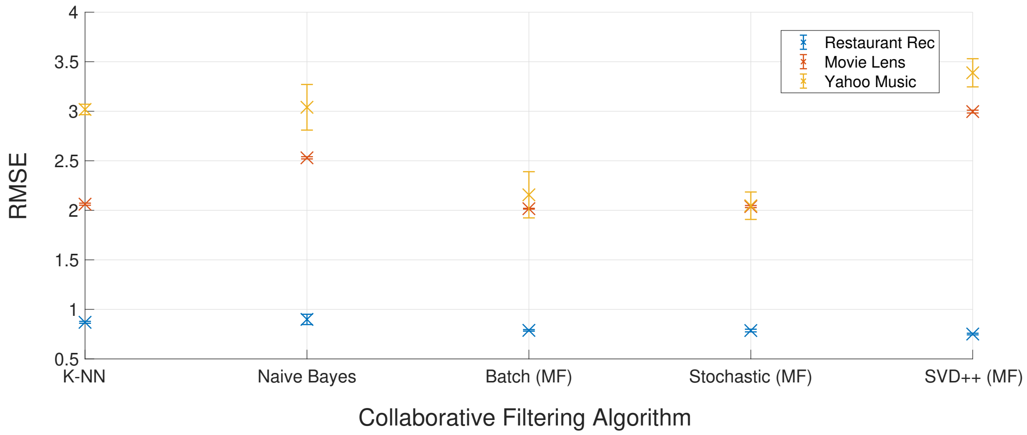


Figure 11: A scatter plot with error bars summarising the five different collaborative filtering algorithms used across the three different datasets. Each dataset has a different optimal algorithm.

- The Movie Lens dataset: larger, medium sparse, medium number of possible ratings – KNN is optimal.
- The Yahoo Music dataset: medium, very sparse, lots of possible rating – stochastic gradient descent matrix factorisation is optimal.

The multi objective optimisation strategy matters a great deal and the chosen strategy should vary depending on the form of the objectives. Although the parameters for the MOO should be learned, it can be challenging deciding on the method of learning – which stakeholder should be prioritised? Since the end goal is to maximise the long term value of a user and thus, their immediate satisfaction, ϵ-constraint is a very effective strategy where the max user error can be set whilst the other constraints are varied. This parameter can be learned in an online system through different trials.

5 Collaborative Filtering Extensions

5.1 Aim

A particularly important aspect of this report has been experimentally outlining and analysing novel approaches to applying collaborative filtering to solve the multi-objective RecSys problem. Since collaborative filtering contributes the greatest towards the final recommendation list as user satisfaction is the most important objective, this section discusses methods of further improving the collaborative filtering algorithms.

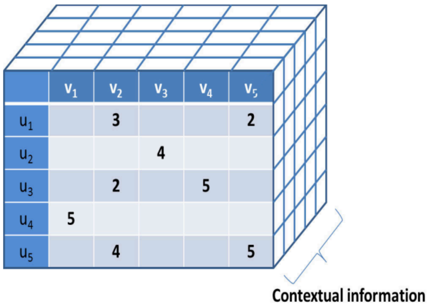
For the Yahoo Music dataset, the optimal collaborative filtering algorithm was stochastic gradient descent. Although this algorithm comparatively produced a lower error than the other algorithms, there are some challenges that remain that motivate further explorations – the error for the Yahoo Music dataset was still quite high compared to the restaurant Rec dataset on account of the harsher environment features. This can be seen in Figure 11 – the algorithm needs to have a **lower error**. The algorithm also needs to be made **quicker** in order to generate a ranking list for the user faster. Finally, an additional problem of **even higher matrix sparsity** needs to be addressed. An example of this could be the case of a brand new user entering the system who has no historic data – this is commonly known as the **cold start** problem.

5.2 Technical Approach

5.2.1 Online Evaluation

Commercial systems generate recommendations online where data is constantly increasing, as opposed to the predominantly offline approach (or online emulated from offline as seen during the MOO) used in this paper. Replicating this from a research perspective holds much value, in particular, being able to assess context-aware algorithms and multi-armed bandits (MABs).

Creating a context aware algorithm helps to factor in additional information making the job of the collaborative filtering algorithm easier. For example, if it is December a system may know to favour Christmas movies. Or, for example, if system were to learn when an individual likes to exercise then it can recommend more upbeat exercise tracks. The applications of this are plenty – the more context we have around a user and around the items, the more likely the recommendations will be relevant [36]. However, as can be imagined, this is reliant on data sources and systems outside of the recommender system and is thus difficult to evaluate offline. Figure 12 shows how the ratings matrix would change to introduce a third dimension under a context aware system [37]. This could be user ratings under different months of the year for example, where the z-axis represents a month of the year.



	v_1	v_2	v_3	v_4	v_5
u_1		3			2
u_2			4		
u_3		2		5	
u_4	5				
u_5		4			5

Figure 12: A diagram of how the ratings matrix would change under a context aware recommender system, where a third dimension is introduced to store the contextual information [37].

Multi-armed bandits help to generate online predictions where there is little to no data by balancing exploration against exploitation. When a user initially enters a system, exploration can be used to learn their preferences, before exploitation is used to take advantage of information learned [15]. This approach helps to tackle sparsity, and in the extreme case, the cold start problem.

5.2.2 Other

Other extensions include applying mini-batch gradient descent on the Yahoo Music dataset to understand its efficacy versus stochastic gradient descent. With mini-batch gradient descent, the training data is divided into smaller groups that are utilised to identify model error and update model coefficients [38].

Further to the bias reduction method discussed in Section 3.3.5 where additional columns store the user and item biases, the ratings can be mean centred to remove bias [17]. In real life, no two users are exactly the same and everyone has biases in some form – some users can be more, or less, optimistic in their ratings. Mean centering is an approach to remove these biases, and works by removing the row average from each rating. However, this does not work with all algorithms, and in particular complicates naïve Bayes.

These methods help to decrease the error and increase the speed of the collaborative filtering approach, making the overall system perform better.

6 Conclusion

There is plenty of literature and research that outlines user-centric recommendation approaches ([8], [18]) and some literature that outlines multi-stakeholder approaches ([22], [25], [20]). However, there is no literature that assesses many different collaborative filtering algorithms to holistically establish a combined multi-stakeholder recommender system that can operate in highly sparse environments.

This report has established that for highly sparse ratings matrices, such as the Yahoo Music dataset [5], stochastic gradient descent matrix factorisation is optimal. For other less sparse datasets batch gradient descent matrix factorisation and k-nearest neighbours proved to be effective. When combining objectives to form a user ranking output, the best multi-objective optimisation strategy is to use ϵ-constraint combined with Pareto optimality. Finally, extending the system to online evaluation and adding in contextual information, in addition to a multi-armed bandit to deal with cold starts, will make a novel multi-stakeholder recommender system that is highly effective at handling extremely sparse environments.

References

[1] Spotify as a recsys example. https://www.spotify.com/uk/.

[2] Netflix as a recsys example. https://www.netflix.com/.

[3] Restaurant rec dataset. https://www.kaggle.com/uciml/restaurant-data-with-consumer-ratings.

[4] Movie lens dataset. https://grouplens.org/datasets/movielens/25m/.

[5] Yahoo music dataset. https://webscope.sandbox.yahoo.com/catalog.php?datatype=r.

[6] Carbon codes. https://carboncodes.co.uk/.

[7] Amazon web services (aws). https://aws.amazon.com/.

[8] Ruisheng Zhang, Qi dong Liu, Chun-Gui, Jia-Xuan Wei, and Huiyi-Ma. Collaborative filtering for recommender systems. 2014.

[9] 35% of amazon.com's revenue comes from behavioural recommendations. https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers. Accessed: 29/02/22.

[10] More than 80% of netflix tv shows that are watched are dis- covered via recommendations. https://www.wired.co.uk/article/how-do-netflixs-algorithms-work-machine-learning-helps-to-predict-what-viewers-will-like. Accessed: 29/02/22.

[11] The netflix prize. https://www.thrillist.com/entertainment/nation/the-netflix-prize. Accessed: 29/02/22.

[12] Personalising explainable recommendations. https://www.youtube.com/watch?v=KoMKgNeUX4k&ab_channel=MLconf. Accessed: 29/03/22.

[13] Khalid al Fararni, Badraddine Aghoutane, Jamal Riffi, Sabri Abdelouahed, and Ali Yahyaouya. Comparative study on approaches of recommendation systems. 2020.

[14] Yilena Pérez-Almaguer, Raciel Yera, Ahmad A.Alzahrani, and Luis Martínez. Content-based group recommender systems: A general taxonomy and further improvements. 2021.

[15] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalised news article recommendation. 2012.

[16] Implicit and explicit feedback recommenders. https://resources.bibblio.org/hubfs/share/2018-01-24-RecSysLDN-Ravelin.pdf.

[17] Yong Zheng and David (Xuejun) Wang. Multi-objective recommendations: A tutorial. 2021.

[18] Priscila Valdiviezo-Diaz, Fernando Ortega, Eduardo Cobos, and Raúl Lara-Cabrera. A collaborative filtering approach based on naïve bayes classifier. 2019.

[19] B. Apolloni, Simone Bassis, Marco Mesiti, Stefano Valtolina, and Francesco Epifania. A rule based recommender system. 2020.

[20] Laizhong Cui, Peng Ou, Xianghua Fu, Zhenkun Wen, and Nan Lu. A novel multi-objective evolutionary algorithm for recommendation systems. 2016.

[21] Evolutionary algorithms aren't good. https://www.quora.com/Why-didnt-evolutionary-algorithms-become-as-popular-as-deep-learning.

[22] Himan Abdollahpouri. Popularity bias in recommendation: A multi-stakeholder perspective. 2020.

[23] Nyoman Gunantara. A review of multi-objective optimization: Methods and its applications. 2018.

[24] Netflix prize dataset features. https://paperswithcode.com/dataset/netflix-prize.

[25] Yong Zheng. Multi-stakeholder recommendation: Applications and challenges. 2017.

[26] Matrix factorisation is highly sought after. https://www.mygreatlearning.com/blog/matrix-factorization-explained/.

[27] Deliveroo as a recsys example. https://deliveroo.co.uk/.

[28] Introduction to matrix factorization for recommender systems. https://osf.io/pnd5w/download.

[29] Matlab. https://uk.mathworks.com/products/matlab.html.

[30] Rajeev Kumar, B. K. Verma, and Shyam Sunder Rastogi. Social popularity based svd++ recommender system. 2014.

[31] Shijie Wang, Guiling Sun, and Yangyang Li. Svd++ recommendation algorithm based on backtracking. 2020.

[32] Sonia Ticha, Azim Roussanaly, Anne Boyer, and Khaled Bsaies. Feature frequency inverse user frequency for dependant attribute to enhance recommendations. 2013.

[33] Ritesh Noothigattu, Nihar B. Shah, and Ariel D. Procaccia. Loss functions, axioms, and peer review. 2021.

[34] Github repository containing 4yp code. https://github.com/rhimshahcc/4YP. Author: Rhim Shah, Year: 2022.

[35] Website load time statistics: Why speed matters in 2022. https://www.websitebuilderexpert.com/building-websites/website-load-time-statistics/. Accessed: 06/04/22.

[36] G. Adomavicius, B. Mobasher, F. Ricci, and A. Tuzhilin. Context-aware recommender systems. 2011.

[37] Xin Liu and Karl Aberer. Soco: A social network aided context-aware recommender system. 2013.

[38] Si Thin Nguyen, Hyun Young Kwak, Si Young Lee, and Gwang Yong Gim. Featured hybrid recommendation system using stochastic gradient descent. 2021.

Department of Engineering Science

Supplementary Questions for 4th Year Project Students

Factor	Answer	Things to Consider	Record details here
Has the checklist covered all the problems that may arise from working with the VDU?	☒ Yes □ No		
Are you free from experiencing any fatigue, stress, discomfort or other symptoms which you attribute to working with the VDU or work environment?	☒ Yes □ No	Any aches, pains or sensory loss (tingling or pins and needles) in your neck, back shoulders or upper limbs. Do you experience restricted joint movement, impaired finger movements, grip or other disability, temporary or permanently	
Do you take adequate breaks when working at the VDU?	☒ Yes □ No	Periods of two minutes looking away from the screen taken every 20 minutes and longer periods every 2 hours Natural breaks for taking a drink and moving around the office answering the phone etc.	
How many hours per day do you spend working with this computer?	□ 1-2 ☒ 3-4 □ 5-7 □ 8 or more		
How many days per week do you spend working with this computer?	□ 1-2 □ 3-5 ☒ 6-7		
Please describe your typical computer usage pattern	I tend to use my computer for periods of 1-2hrs at a time before taking a break wherein which I then ensure not to look at a screen for at least 15 minutes. In total I use aim to use my computer for this project for ~ 4hrs/day on average, however this figure may fluctuate.		

Student Declaration and Academic Approval

Student Declaration:

I have completed the DSE Workstation Checklist and the Supplementary Questions for my computer-related risk assessment for 4YP Project Number indicated below:

4YP Project Number: 12500

4YP Student's Name *(please print)* Rhim Shah

4YP Student's Signature: RShah

Academic Approval

I confirm my approval of this 4YP DSE Risk Assessment.

Academic Supervisor's Name: *(please print)*

Michael Osborne

Academic Supervisor's Signature: