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, to the movies that Netflix 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.

This report aims to explore the background and key context behind recommender systems that are used in commercial marketplaces, where data-sparsity is a defining issue. The majority of the report will apply these concepts across three datasets of varying size under three different technical algorithms, using a collaborative approach. These datasets are the Movie Lens Dataset, the Yahoo Music Dataset and the Restaurant Rec Dataset. The final part of the report aims to develop and explore a novel approach that improves upon the identified downsides of the previous algorithms, in addition to discussing possible further explorations. The novel algorithms proves to be x times quick and y times more accurate than the next best algorithm z.

Acknowledgements

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

This project stemmed from an internal need from a start-up that I co-founded called Carbon Codes. Carbon Codes is aimed at improving the sustainability of the foodservice industry by connecting consumers to discounted eco-conscious food from our partner restaurants. The initial aim of the project was to improve the recommendations of restaurants to users. The project subsequently broadened to more general marketplaces. An additional thanks goes to the Carbon Codes Team for the support during the project as well as for providing critical AWS resources for increased compute power.

Contents

1 Introduction

1.1 Project Aim

The aim of this project is 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. Both the contextual and technical background will be 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) and then the multi-objective optimisation (MOO) to produce a final ranking based on all stakeholder interests.

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.

A key part 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 collaborative filtering that will be explored in this report. We'll be looking into three key 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 doesn't 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'll be focusing on collaborative filtering with the Restaurant Rec dataset which is 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'll be using the movie lens dataset which is large but 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'll be focusing on a rating-based collaborative approach which is generally the preferred method. We'll be using the Yahoo Music dataset which is medium in size and very sparse ($\sim$0.35%).

To summarise, we will focus on exploring collaborative filtering since this the most used and effective technique for the applications discussed above. This is predominantly for optimising for user interests, other parts of the system will use other non-ML, 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' will be used interchangeably. The three key stakeholders are the product user who views the recommendation list (e.g. Spotify user), the vendor that is supplying the item (e.g. music artist), and finally, the business that owns and supports the marketplace (e.g. Spotify). In these examples the item is a Spotify track. Providing a final recommendation list that takes into account the interests of all of these three stakeholders is a complex problem that involves two key elements: machine learning (for the collaborative filtering) and multi-objective optimisation (for the final ranking).

In this report, the user, and sometimes the vendor, have objectives that are optimised via collaborative filtering, whilst the businesses have an objective that is generally solely dependent on the items (e.g. certain products will maximise revenue). It's 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 will extend 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 will be 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 [1], and more than 80% of Netflix TV shows that are watched are discovered via recommendations [2]. 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 [3]. This extends past Netflix to almost any consumer-facing internet-based product, which are all in essence a marketplace – either for information, services or for physical goods. A marketplace is an intermediary that helps facilitate economic interaction between two or more agents.

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 in its simplist form can be described 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 supervied 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.

- **Collaborative filtering (CF)** compares the past interaction history of a user to thee interaction history of other users to generate recommendations. This is unsupervised learning. CF relies on a ratings matrix where the items are along the top, users are along the left, and the entries are the ratings. As discussed previously, this is the primary approach we will be exploring 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). **Group-based** recommendations are focused on recommends items to a group of users as opposed to a single user.
- 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. 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 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.

In most cases, there'll be a combination of content-based recommendation and collaborative filtering, combined with even more context and maybe even a hybrid multi-armed bandit approach – in the ML-world and therefore the RecSys-world, data is key. 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 [4]. 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 off of 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.

Some important details around collaborative filtering:

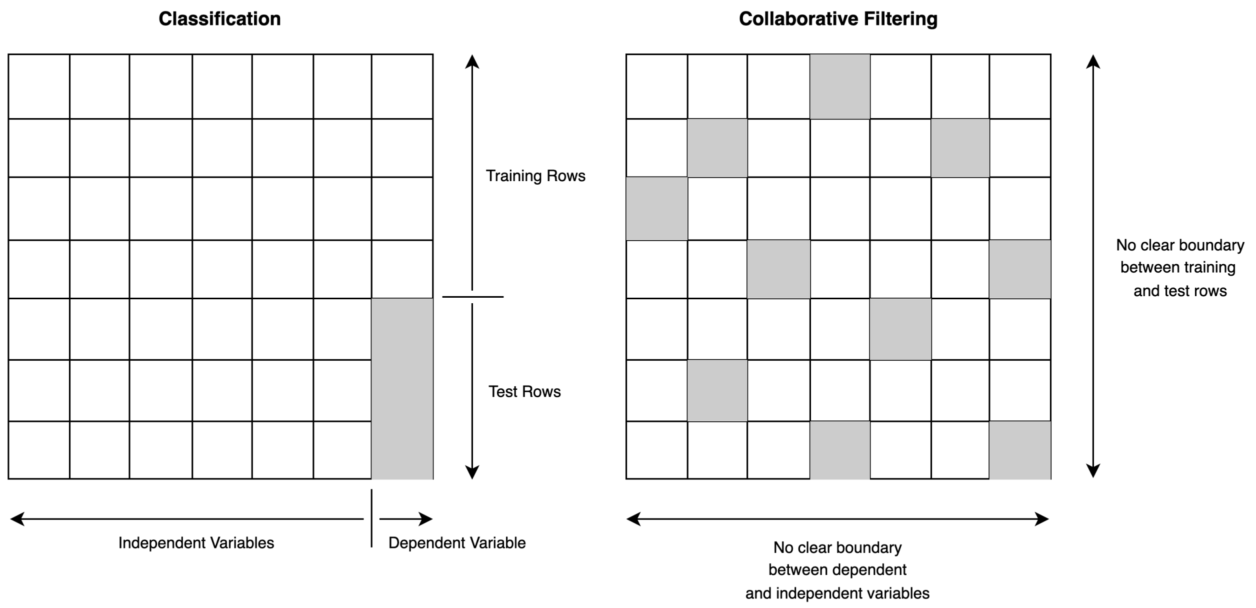

Classification
Collaborative Filtering
Training Rows
Test Rows
Independent Variables
Dependent Variable
No clear boundary between training and test rows
No clear boundary between dependent and independent variables

Figure 1: Comparison between classification and collaborative filtering. *Adapted from [4], Figure 1.4.*

- **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. 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. This report will cover 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. The ratings we will be dealing with this in this report are explicit ratings.
- **Ratings prediction vs. ranking prediction.** Ranking prediction can be achieved directly or can be based off of the explicit ratings prediction (i.e. forming a ranked list by ordering the ratings for a user).
- **User-based vs. item-based approach**. A user-based approach considers users and relation-

ships between rows, whereas an item-based approach considers items and relationships between columns. This will become more mathematical evident in section []. The most effective methods use a combined approach.

1.3.3 Common Collaborative Filtering Algorithms

Some common collaborative filtering methods include:

- **K-Nearest Neighbours (K-NN).** The top *K* most similar rows/columns are used to form a prediction for user *i* on item *j*.
- **Naive Bayes.** Bayes theorem is used to maximise the posterior probability of the rating for user i on item j taking a certain value.
- **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. Specific examples include SVD++.
- **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.
- **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. These algorithms are overly simplistic and are usually ineffective.

This report explores K-NN (Section []), Naive Bayes (Section []) and LFMs (Section []), before developing a novel algorithm (Section []).

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. The aim of this report is to demonstrate these issues.

- **Data Sparsity.** Only some users have rated some items (the cold-start problem). This is thee case for all marketplaces nowadays and is thus a defining feature of the datasets used in this report. Sparsity is calculated using Equation 1.
- **Scalability.** Often datasets are very large and algorithms need to be able to keep up with this scale.

- **Population Bias.** Popular items are recommended more than long tail items, making them even more popular and dragging out the long tail
- Other problems:
  - 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 batman 2 lack diversity).

$$\text{sparsity} = \frac{\text{no. of rated entries}}{\text{total no. of entries}} * 100 \tag{1}$$

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. 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's not about just 'doing' multi-objective optimisation, it's about choosing the best strategy. Furthermore, optimising for several complimentary objectives actually improves each metric independently compared to just optimising that metric (proof). Whilst optimising for competing metrics isn't necessarily a zero-sum game [].

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). Pareto optimality will also be explored in this report, where weightings are unknown.

2 Literature Review

There is much literature and research around single-objective Recommender Systems which are user-centric, however, multi-stakeholder RecSys still seems to be a relatively novel field.

2.1 Recommender Systems, The Textbook

A summary of the field of recommender systems in general and proves as a sound basis for the CF algorithms used in this paper.

2.2 Multi-Stakeholder Recommendation: Applications and Challenges

Multi-stakeholder view on recommender systems. The challenges that they face and the primary applications of effective MSRS.

2.3 Popularity Bias in Multi-Stakeholder Recommendation

Multi-stakeholder recommender systems with a focus on tackling popularity bias – items in the long tail aren't recommended often, which means that the long-tail gets worse.

2.4 Multi Objective bandits for Music Streaming

The use of multi-armed bandits to recommend Spotify tracks to users is an interesting exploration and a potential extension to this report.

3 System Overview

3.1 Introduction

3.1.1 Process

There are two key 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 key 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 aim of exploring well-known algorithms is to understand their effectiveness within multi-stakeholder RecSys, as well as initiating insights into how to develop a novel algorithm which improves upon any downfalls within a constrained, but commercially applicable, context.

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.
- The vendor objective which originates from the vendor's interests.

It's worth noting 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. As a result, much of this chapter will be focussed on the collaborative filtering part of the system.

In MOO objectives can either be complementary or competing. Optimising for complementary objectives has shown to improve the performance of each individual objective [], and in the case of marketplaces, because the end goal is to keep the user happy, the objectives tend to be complementary. In some commercial cases, there can be side-stakeholders in addition to the three main ones e.g. the Deliveroo [] rider for Deliveroo.

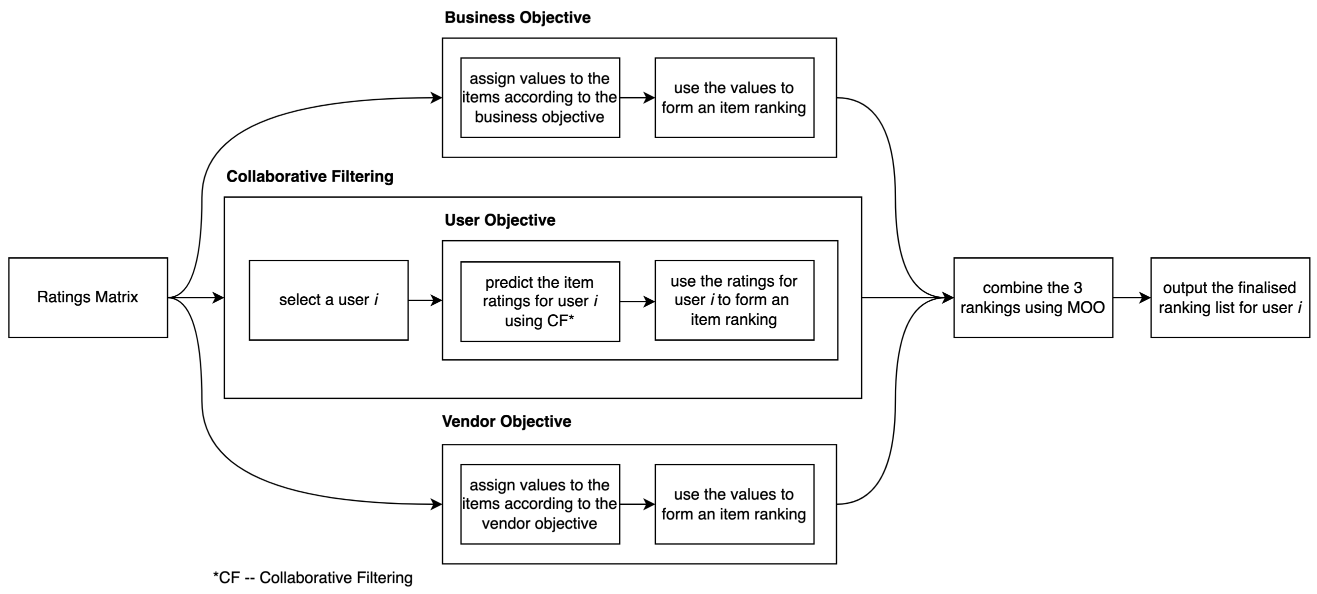

Business Objective
assign values to the items according to the business objective
use the values to form an item ranking
Collaborative Filtering
User Objective
Ratings Matrix
select a user i
predict the item ratings for user i using CF*
use the ratings for user i to form an item ranking
combine the 3 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 approach fits together.

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

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

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. There are various different ways of combining these rankings, the key ones that are explored are weighted methods and ϵ-constraint methods. Weighted methods are straightforward if the weights of the objective function are known. We will 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. 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. 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 will 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 aren't 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.
- Population Bias: the frequency at which ratings appear, which normally forms a long tail.
- 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
Naive 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: Collaborative filtering input parameter/s and output result/s of each 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 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, population 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

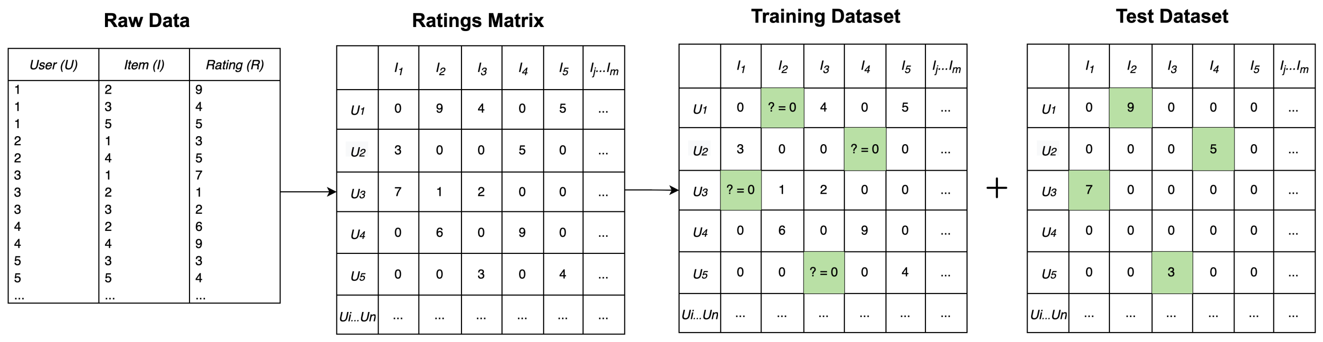
The datasets that are used are described as ratings matrices – users are represented along the rows, and items are represented by the columns. A specific entry in the ratings matrix (R_{ij}) is user i's rating of item j. The ratings matrix is formed from instances as shown in Figure 3, where n is the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

The ratings matrix is then split into a training and test dataset by entries using cross validation. The key 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.

It is important to 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. These features can be seen in Table 2. Note that the original



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
...	...	...

	I_1	I_2	I_3	I_4	I_5	$I_j ... I_m$
U_1	0	9	4	0	5	...
U_2	3	0	0	5	0	...
U_3	7	1	2	0	0	...
U_4	0	6	0	9	0	...
U_5	0	0	3	0	4	...
$U_i ... U_n$	...	...	...	...	...	...

	I_1	I_2	I_3	I_4	I_5	$I_j ... I_m$
U_1	0	? = 0	4	0	5	...
U_2	3	0	0	? = 0	0	...
U_3	? = 0	1	2	0	0	...
U_4	0	6	0	9	0	...
U_5	0	0	? = 0	0	4	...
$U_i ... U_n$	...	...	...	...	...	...

	I_1	I_2	I_3	I_4	I_5	$I_j ... I_m$
U_1	0	9	0	0	0	...
U_2	0	0	0	5	0	...
U_3	7	0	0	0	0	...
U_4	0	0	0	0	0	...
U_5	0	0	3	0	0	...
$U_i ... U_n$	...	...	...	...	...	...

Figure 3: Diagram of how the training and the test datasets are formed from the raw data.

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
Ratings	1,2,3	1,2,3,4,5,6,7,8,9,10	0.1 - 10 (in varying increments)
No. of unique ratings	3	10	43
Sparsity (%)	6.4716	3.4149	0.3532
No. of rated entries	1,161	49,175	883
No. of rows with 1 or 0 nonzero values	0	29	253
No. of columns with 1 or 0 nonzero values	0	277	419

Table 2: Environment features of the three datasets.

There are four key environment features which will be varied 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: depending on the order of complexity of different algorithms, some scale well with increased size and some do not – in this case increasing size means increasing the matrix dimensions, n and m.

- Population 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 population bias.
- Sparsity: some algorithms are able to handle sparsity well, whereas others aren't able to handle sparsity and produce false entries and a high RMSE.
- 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.

3.2.3 Restaurant Rec Dataset

The Restaurant Rec dataset 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 also shows that all the items are rated at least 3 times and even though a population bias is present, it is less so compared to the other datasets. The x-axis of each graph contains the ranked item number and the y-axis contains the frequency of those items. Note that the item number is just a popularity rank and doesn't actually refer to the item number from the ratings matrix.

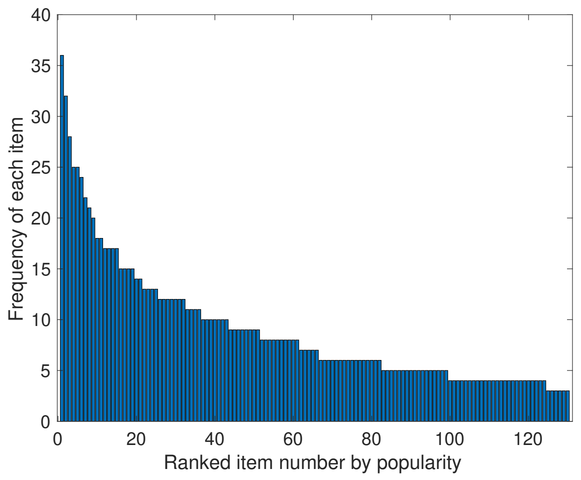


Figure 4: Restaurant Rec Dataset Long Tail

Table 3 shows the three different objectives that the final ranking will have to optimise for – user satisfaction, novelty and sustainability. Novelty is aimed at ensuring restaurants that haven't 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 gCO_2-eq [] equivalent 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: Restaurant Rec dataset objectives.

3.2.4 Movie Lens Dataset

The Movie Lens dataset 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 population bias in Figure 5. 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 will have 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 aims to favour longer content in the final ranking.

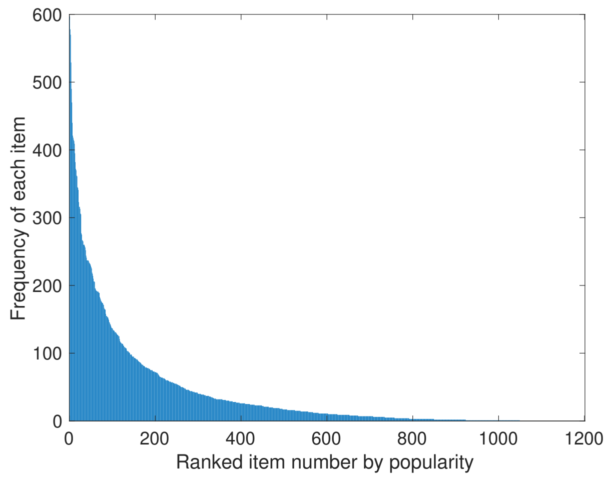


Figure 5: Movie Lens Dataset Long Tail

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.
Business	Revenue	Assign movies a length of time.	Rank based on maximising the potential time listened.

Table 4: Movie lens dataset objectives

3.2.5 Yahoo Music Dataset

The Yahoo Music dataset 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's 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 6 also shows how it is heavily skewed towards 50 items, whilst the remaining items have few ratings – a perfect example of heavy population bias.

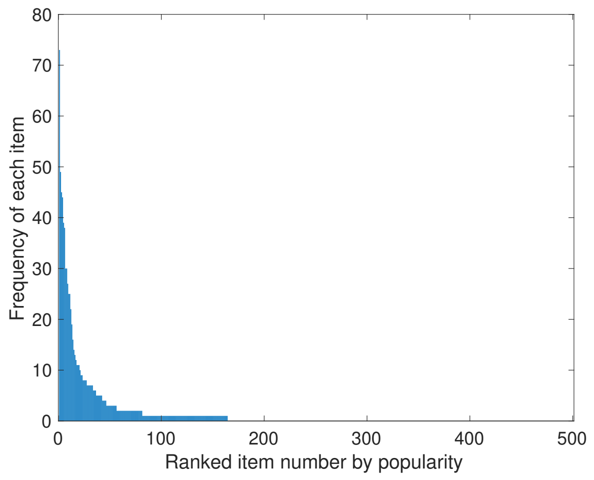


Figure 6: Yahoo Music Dataset Long Tail

Table 5 shows the three different objectives that the final ranking will have 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**	**Methods**	**Technical Objective**
User	User Satisfaction	Accuracy of rating prediction.	RMSE
Vendor	Serendipity	Non-obvious items from the population bias graph are ranked higher.	Rank based on assigned population bias ranking.
Business	Time Listened	Assign songs a length of time.	Rank based on maximising the potential time listened.

Table 5: Yahoo Music dataset objectives.

3.3 Collaborative Filtering

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

3.3.1 Introduction

Notation

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

$$(i, j) \text{ : row-column co-ordinates}$$

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

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

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

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

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

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 is calculated using Equation 4, where e_{ij} is calculated in Equation 3.

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

3.3.2 K-Nearest Neighbour

The k-NN 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 isn't taken advantage of, and the disadvantage of using an item based approach is that any user correlation information isn't taken advantage of. Hence, I'll be looking to assess both separately, as well as a combined version, over the datasets. K-NN 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 model cannot be applied. Contains an offline and an online phase.

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's 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's 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
5         rows_matrix = [ rows_matrix ; nn_entry comp_entry]; % assign to a matrix that
     contains all the paired nonzero entries
6     end
7 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 calculated using Equation 6.The row means of rows u and v are calculated in Equation 5.

$$\mu_u = \frac{\sum_{j \in I_u \cap I_v} r_{uj}}{|I_u|}, \mu_v = \frac{\sum_{j \in I_v \cap I_v} r_{vj}}{|I_v|} \tag{5}$$

$$\text{pearson}(u,v) = \frac{\sum_{j \in I_u \cap I_v} (r_{uj} - \mu_u) \cdot (r_{vj} - \mu_v)}{\sqrt{\sum_{j \in I_u \cap I_v} (r_{uj} - \mu_u)^2} \cdot \sqrt{\sum_{j \in I_u \cap I_v} (r_{vj} - \mu_v)^2}} \tag{6}$$

In its simplist form, the item-based similarity can be calculated by taking the user-based similarity of the transpose of the ratings matrix, R^T. In the implmentation of...

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 prediction is made using Equation 7.

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

An equivalent for the item-based prediction can be made where instead...

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 Naive Bayes

The Naive 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 []. The Naive 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. Equation 8 shows how the posterior is calculated using Bayes Theorem, where $\{r_{uj}, \forall j \in I_u\}$ denotes the set of nonzero ratings for user u.

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

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 (Equation 9).

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

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: Naive 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 naive assumption which is based on conditional independence between the ratings [4], and can be mathematically described by Equation 10.

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

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);
8 % divide the numerator by the denominator
9 cond_prob_val = cond_prob_val_num ./ cond_prob_val_den;
```

Listing 5: Naive 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.This is described by Equation 11.

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

4. Form the prediction ratings matrix.

In a similar fashion to k-NN, 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.

Something important to note is that the Naive Bayes algorithm predicts discrete integer ratings whereas the k-NN algorithm predicts any value within a range. Naive 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 [4].

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, as shown in Equation 12. R_{tr} refers to the training dataset.

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

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 removing 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 shown in Equations 13 and 14.

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

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

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 V are outlined below. It is important to note that the values are initially stored in an intermediary value (U_{next} and V_{next}), so that the current iteration doesn't 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 matrixes.

```
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'. Using an implicit user factor matrix 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.

λ is the predetermined regularisation term to prevent against overfitting. 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. Equations 15 and 16 are adapted equations from the previously described stochastic, whilst Equation 17 outlines the updates to the normalised implicit ratings matrix, Y.

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

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

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

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

Once the iterations have terminated, the matrixes can be recombined to find $\tilde{R}$. For unconstrained matrix factorisation using batch and stochastic gradient descent, Equation 18 can be used. For SVD++, Equation 19 can be used.

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

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

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 further developed.

K-NN 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. Equation 20 shows how the weights are calculated, where v_j is the number of ratings for item j [4]. w_j is inserted into each summation from Equation 6.

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

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

The **Naive 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 matrixes (U and V). The implementation and maths behind this will be explored in the novel algorithm in Section [].

Combining some of these algorithms to form a hybrid algorithm can prove to be an effective way to utilise the positives from multiple different algorithms. This will be explored, along with the other extensions outlined, when developing the novel algorithm in Section [].

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 isn't 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 aim now becomes to just 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 7. The final ranking is evaluated in two ways, as discussed in section []. 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 will somewhat just be a reflection of the MOO strategy taken, the aim 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 [5].

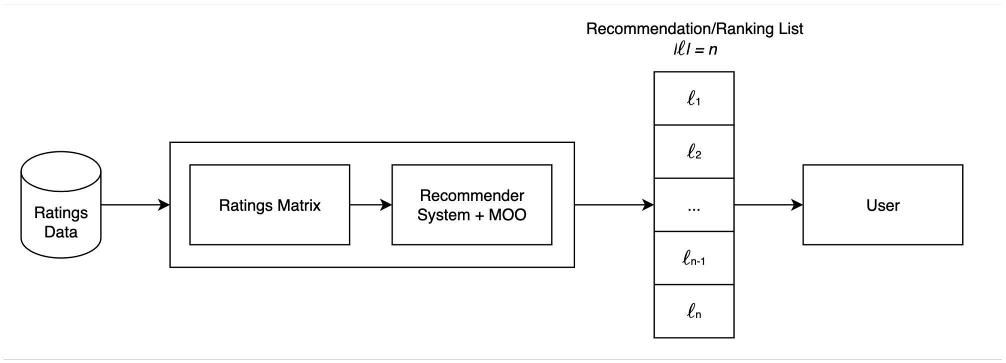


Figure 7: Flow chart of how the final ranking list is produced. *Adapted from Figure 2.1 [6].*

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**. The objectives are combined using a weighted equation, where the weights (w_k), i.e. the parameters, are known. Equation 21 shows how the final rank is produced for each item j, once each item has its new rank, an overall rank is formed as the final output rank. k is the objective number.

$$rank_j^{output} = \sum_{k=1}^{3} w_k \cdot rank_{jk} \tag{21}$$

The second approach is to use **ϵ-constraint methods** which essentially aim to optimise one objective whilst treating the other objectives as constraints with predetermined upper bounds [7]. 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. This is what Equation 22 describes, where j represents an item and k represents an objective. 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.

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

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 will be 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.

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 [8].

4 Exploration of Well-Known Algorithms

4.1 Restaurant Rec Results & Analysis

4.1.1 CF Results

Table 6 shows the collaborative filtering results of running the various different algorithms under their optimised input parameters. 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 was taken.

CF Algorithm		**Input Parameter/s**	**RMSE**		**Computation Time**		**Other**
			Av	SD	Av	SD	Av
K-NN		k = 35%	0.7840	0.0143	0.2384	0.0283	False entries = 78.67
Naive Bayes		alpha = 0.2	0.8988	0.0459	0.8381	0.0101	False entries = 0
Matrix Factorisation	Batch	step = 0.001 k = 6 CC = 0.000001	0.7881	0.0077	0.2683	0.0679	% Δ RMSE = 32.27 Iterations till convergence = 196.3
	Stochastic	step = 0.001 k = 8 CC = 0.000001	0.9891	0.0421	0.3435	0.0706	% Δ RMSE = 0.0700 Iterations till convergence =162.3
	SVD++	step = 0.001 k = 3 lambda = 0.05 CC = 0.000001	1.385	0.1089	2.120	0.3327	% Δ RMSE = 0.1200 Iterations till convergence = 451.7

Table 6: Collaborative filtering results for the Restaurant Rec dataset.

Before moving onto multi-objective optimisation, the best collaborative filtering algorithm needs to be chosen, given the results in table 6. When comparing the K-NN algorithm and the Naive Bayes algorithm it is clear that K-NN is both quicker and has a lower error by more than 0.1. However, the downside of this algorithm is the presence of false entries where there wasn't enough data (due to the sparsity) for the algorithm to predict an entry. In these cases, the correlation between rows or columns couldn't be calculated and the algorithm produced a nan output. On average 78.87 entries were false. It's worth noting that these false entries were ones that both the user and item based K-NN methods couldn't predict. On average, around 50% more item-based entries were false compared to user-based. In commercial applications there can be simple methods to tackle this, such as averaging all the present entries in the row and the column – this is an overly simplistic example.

When comparing the K-NN method to matrix factorisation methods, the only contender is the Batch method, which produced a slightly higher error and had a marginally higher computation time. Both the stochastic and SVD++ methods performed poorly, especially SVD++. This is partly due to their inability to descend – on average, stochastic algorithms only reduced the initial error by 0.07% whilst SVD++ only reduced the initial error by 0.12%. Batch reduced the initial error by 32.27%. This is due to stochastic gradient descent (using in both algorithms) being unable to deal with sparsity [] – these algorithms become increasingly worse as sparsity increases as can be seen in the subsequent sections. 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, the Batch Matrix Factorisation collaborative filtering algorithm works the best. Although the performance of K-NN is marginally better, over 27% of the entries are false entries and cannot be predicted. The time to generate a single prediction for an entry is still 2.120s since the entire ratings matrix is optimised when latent factor models are used.This is the algorithm that will be used for multi-objective optimisation.

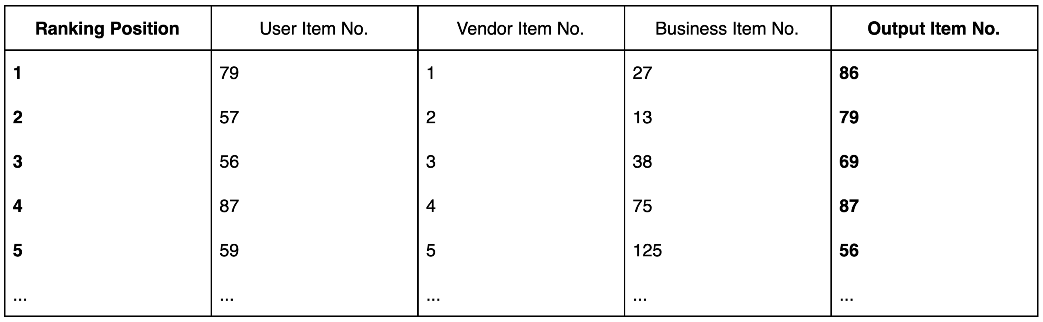
4.1.2 Multi-Objectiveness

The user ranking was based off of 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 ranking values were then 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 it isn’t fair to compare that error to the rest of the objective ranking errors.

	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: Restaurant Rec multi objective optimisation results, averaged over 3 runs.

Figure 8 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.).



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

4.1.3 Summary

When analysing the effectiveness of a collaborative filtering approach, the error is the most important aspect. Depending on the approach, the error varies a great deal. The speed of producing rankings is also an important factor when deciding on the collaborative filtering approach – batch matrix factorisation was the best in the case of the Restaurant Rec dataset, both by error and speed. The defining feature of this dataset is its small size. This use case is reflective of an early stage marketplace e.g. Deliveroo in its early days.

The experiments prove that 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. In the case of marketplaces, this must be the user objective. Yet, optimising using all the objectives will inevitably produce a better system outcome since all needs of the stakeholders are taken into account on some level.

4.2 Movie Lens Results Analysis

4.2.1 CF Results

CF Algorithm		Input Parameter/s	RMSE		Computation Time (s)		Other
			Av	SD	Av	SD	Av
K-NN		k = 25%	2.0422	0.0091	104.0	18.75	False entries = 55
Naive Bayes		alpha = 0.2	2.5301	0.0189	559.2	48.05	False entries = 0
Matrix Factorisation	Batch	step = 0.0001 k = 25 CC = 0.0001	2.0366	0.0035	4.471	0.0921	% D in RMSE = 20.13 Iterations till convergence = 113.3

Table 8: Collaborative filtering results for the Movie Lens dataset.

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 Naive Bayes algorithm has the largest error and the largest computational time – it is evident that this algorithm scales poorly with increased dataset sizes. The K-NN algorithm has a much lower percentage of false entries compared to the Restaurant Rec dataset, of 0.45%. This is due to more data being present since this dataset is nearly 50 times larger. Batch Matrix Factorisation produced the lowest error. Both stochastic and SVD++ weren't able to converge with a convergence criterion of 0.0001, increasing this to 0.0000001 led to a large RMSE for stochastic of 5.53 with only a 0.0315% reduction in RMSE over 3103 iterations – SVD++ performed even poorer.

Although Batch has a lower error, its computation time per entry of 4.471 seconds is very high – in commercial settings, results need to be produced quickly. Per entry, K-NN has a computation time of

0.0085s – this is significantly lower. Hence, for this dataset, the K-NN collaborative filtering algorithm works the best and this is the algorithm that will be used for multi-objective optimisation.

In the context of recommending movies to users, basic contextual information can be used to help improve post processing predictions of false entries for K-NN. 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 25% to reflect the increased preference, if it is December – the ranking of this item will then reflect accordingly. This is still an overly simplistic approach and only works in this case because the percentage of false entries is so small.

4.2.2 Multi-Objectiveness

The user ranking was based off of the collaborative filtering predictions, where each item was ranked after the unrated entries were predicted. Since K-NN produces false entries, these 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 business ranking tended towards the user ranking using gradient descent until the $target\_error_{max}$ for the user ranking error 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 ‘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).

	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	440.6	2.781

Table 9: Movie Lens multi objective optimisation results, averaged over 3 runs.

The $target\_error_{max}$ was set at 70 leading to an average computation time of 1.125s – this is long for the user and would lead to a bad experience. The error is also high both for the user and the business, especially when comparing them to the corresponding restaurant Rec ranking errors.

Figure 9 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.

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

4.2.3 Summary

The best collaborative filtering approach was K-NN 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.

Figure 10 shows how the business ranking error changes as the $target\_error_{max}$ is increased. It operates in a mildly linear fashion with a few steps (e.g. at (300,250)) due to the vendor ranking.

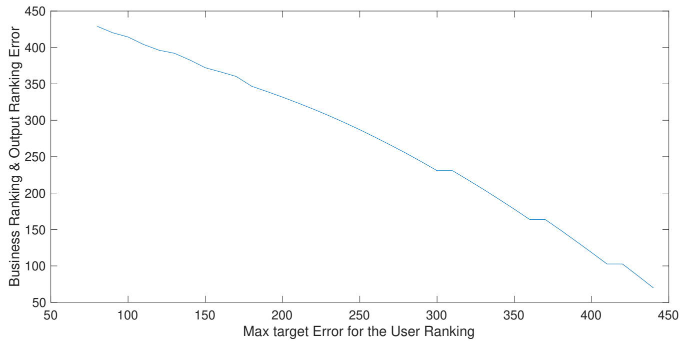


Figure 10: Varying the threshold for the $user\_ranking\_error$ when fixing the business ranking and conducting ϵ-constraint.

The use of ϵ-constraint MOO showcased worse overall ranking errors. However, this is due to the Movie

Lens dataset which is larger and is far sparser compared to the Restaurant Rec dataset. In fact, this presents a better strategy for combining the different rankings since the $target\_error_{max}$ parameter can be learned as a single parameter.

4.3 Yahoo Music Results Analysis

4.3.1 CF Results

CF Algorithm		Input Parameter/s	RMSE		Computation Time (s)		Other
			Av	SD	Av	SD	Av
K-NN		k = 40%	3.0184	0.0481	0.3528	0.0770	False entries = 102.3
Naive Bayes		alpha = 0.15	3.0402	0.1996	3.0301	0.1733	False entries = 0
Matrix Factorisation	Batch	step = 0.001 k = 35 CC = 0.0001	2.1567	0.2465	0.3697	0.0985	% D in RMSE = 27.35 Iterations till convergence = 52
	Stochastic	step = 0.001 k = 25 CC = 0.0001	3.0416	0.1187	0.6620	0.0605	% D in RMSE = 0.256 Iterations till convergence = 36

Table 10: Collaborative filtering results for the Yahoo Music dataset.

The prediction difficulty with the Yahoo Music dataset is clearly demonstrated. In this scenario the entries may not be explicit user ratings but could be, for example, a length of listening time of a user for a particular song, which is then scaled and put in buckets. The values are more continuous, i.e. 43 discrete rating values from 0.1 to 10, in varying increments. The sparsity is also very high for this dataset at 0.3

Both the K-NN algorithm and the Naive Bayes algorithm performed poorly by having large errors in addition to the Naive Bayes algorithm having a high computation time – the Naive Bayes was also out performed by Stochastic (SGD) Matrix Factorisation. Due to the tough environment features of this algorithm, the K-NN algorithm had a 46% false entries rate – this is extremely poor and renders this algorithm ineffective. SVD++ performed poorly, with SVD++ not iterating at all as it terminated on the convergence criterion immediately – hence results were excluded from table [].

The best collaborative filtering algorithm is batch (BGD) matrix factorisation for its low error and low computation time of 0.3697. Note that as mentioned previously latent factor models (i.e. matrix factorisation) explored thus far in this report have to predict the whole matrix – this is not an issue as long as the computation time for the whole matrix doesn't hinder the end user experience (less than $\sim$ 0.5s []).

4.3.2 Multi-Objectiveness

To be completed.

4.3.3 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). The novelty will focus on this algorithm for this very reason – sparse marketplaces are common, yet algorithms find difficulty in producing accurate rankings.

4.4 Summary

It is evident that the collaborative filtering algorithm of choice can heavily affect the RMSE, the predicted ratings, and hence the rankings and the end satisfaction of a user. 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. Batch dealt the best with increased sparsity and increased scale. The lower errors that batch matrix factorisation produced is testament to its ability to deal with the incumbent long tail that was present in the data.

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.

The next section will look at a novel approach on how to develop matrix factorisation to be even more effective. Whilst also factoring in contextual information to reduce the computational time by limiting the number of items that are ranked for an individual. The focus will be on the Yahoo Music dataset as it was so sparse and presents the toughest environment – if an algorithm can be effective on a tough environment, it should also be effective on an easier environment, such as the Restaurant Rec dataset.

5 Exploration of a Novel Algorithm

5.1 Technical Approach

5.1.1 Aim

5.1.2 Bayesian MF/NN

5.2 Analysis

5.2.1 Results

5.2.2 Analysis

5.3 Further Improvements

6 Conclusion

References

[1] 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.

[2] 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.

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

[4] Charu C. Aggarwal. *Recommender Systems: The Textbook*. Springer, 2016.

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

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

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

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