

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Recommender System With Explanations

Frederico Portugal Pinho Rocha



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Ana Paula Rocha

July 14, 2021

Recommender System With Explanations

Frederico Portugal Pinho Rocha

Mestrado Integrado em Engenharia Informática e Computação

July 14, 2021

Abstract

With the approaching dawn of the Artificial Intelligence (AI) Revolution, deemed the successor to the Industrial Revolution of the 18th century and characterized by the mass adoption of AI capital into the production process, more and more commercial platforms are adopting actually-existing narrow AI into their solutions, in order to ease consumer interaction, increase consumer retention and handling of Big Data, ultimately bolstering the value of their end-product. One of the types of narrow AI that has been enjoying a surge in popularity in the last decade among tech giants has been Recommender Systems.

Recommender Systems serve to analyse patterns in user and item profiles, in a given environment or domain, and draw predictions for yet unexplored suitable user-item pairs. Research into this subject began intensifying after the famous Netflix Prize contest in the late 2000s, which challenged contestants to create an algorithm that could predict user ratings for films, based only on previous given ratings, with the most accuracy; ever since, variations and improvements on the original Funk Matrix Factorization model, which originated from this contest, started to be proposed to the scientific community, with one of them being the relatively simple and popular application of Single Value Decomposition in a Recommender System. Alongside this boom in mathematical intensive research of recommender systems, there grew a branch of research into recommender systems that focused more on the psychological and emotional aspect of end-users, positing that an engine that could be able to explain transparently the reasons for the recommendations given would increase their overall effect on the user's decisions.

With this dissertation, we provide a practical model for implementing explanations into recommender systems, by proposing an item-influence based solution and applying it to a generic recommender system, created over an item-item similarity collaborative filtering model. This influence based solution can also have the versatility of calculating not only a specific item's influence, but also the influence of items' characteristics on the recommendation, thus transposing into a content-based explanation model built over a collaborative filtering recommendation model. It is also hypothesised a user-influence based solution for generating explanations, in conjunction with the item-item similarity engine; it unfortunately suffers greatly from cold start and higher relative processing power, as well as the arbitrary constraints for selecting users to base the explanation on, as with the dataset used we had the luxury of users being able to be categorised as friends and elite.

Keywords: Recommendation systems, collaborative filtering, item similarity, explanations, influence, SVD, big data, artificial intelligence, computational advertising, information systems applications

Resumo

Com o início da Revolução de Inteligência Artificial (IA) aproximando-se, considerado o sucessor à Revolução Industrial do século 18 e caracterizada pela adoção em massa de capital inteligente no processo produtivo, cada vez mais plataformas comerciais estão a adotar *narrow AI* já existente nas suas soluções, com o intuito de facilitar interações com o consumidor, aumentar a retenção de consumidores e o tratamento de *Big Data*, fundamentalmente potenciando o valor do seu produto final. Um dos tipos de *narrow AI* que tem vindo a ganhar popularidade na última década com as grandes empresas de tecnologia tem sido os Sistemas de Recomendação.

Sistemas de Recomendação são capazes de analisar perfis de utilizadores e de itens, num determinado ambiente ou domínio, e retirar conclusões quanto a possíveis pares de utilizador-item ainda não explorados que sejam adequados. O estudo desta matéria intensificou nos finais dos anos 2000s, com o famoso concurso Netflix Prize; este desafiava as equipas participantes a criar algoritmos que pudessem prever o rating que utilizadores dariam a filmes da plataforma, com base apenas nos ratings já dados, com a maior certidão. Desde então, variações e melhorias foram vindo a ser descobertas para o modelo Funk MF original, que surgiu durante o concurso, sendo uma delas a relativamente simples e eficaz aplicação de *Single Value Decomposition* em sistemas de recomendação. Em paralelo a este *boom* de pesquisa de sistemas de recomendação com elevado foco no lado matemático, desenvolveu-se também um novo ramo de pesquisa de sistemas de recomendação mais focado no aspeto psicológico e emocional dos utilizadores finais, levantando a tese de que um sistema que fosse capaz de explicar os resultados obtidos com uma dada transparência e clareza iria afetar positivamente o nível de aceitação do utilizador à recomendação obtida.

Com esta dissertação, apresentamos um modelo prático de implementar explicações num sistema de recomendação, propondo uma solução baseada na influência que um item tem na recomendação. Esta proposta foi construída sobre um sistema de recomendação genérico, que usa um modelo *collaborative filtering* com uma matriz de similaridade item-item. Esta solução baseada em influências pode ser aplicada também a outras áreas do domínio, além de apenas itens, podendo-se usar esta solução para calcular a influência de determinadas características de itens na recomendação, transpondo então deste modo o modelo de explicação para um modelo *content-based* sobre um modelo de recomendação que usa *collaborative filtering*. Foi também hipotetizado a aplicação da solução ao cálculo da influência de utilizadores na recomendação, não obstante o facto do sistema de recomendação ser construído na base de similaridades item-item. Infelizmente esta aplicação da solução sofre não só de *cold start* e de um elevado encargo no processador relativamente às outras hipóteses, mas também da dependência de existir apriori categorias para facilitar a seleção de grupos de utilizadores, já que o *dataset* utilizado neste projeto já continha agrupamento possível de utilizadores em amigos e elites.

Acknowledgements

This masters dissertation was initially planned to have been worked on a year prior, in association with Beihang University; it was unfortunately delayed a full year and, ultimately, the cooperation was cancelled due to the global pandemic. This project is thus concluded after 18 peculiar months, in big part due to the help of everyone who believed in me, advised me, and kept me focused.

My family, for the unflinching support throughout.

Ana Paula Rocha, for the countless emails and the help provided in completing this project.

Richong Zhang, for the initial availability of being co-supervisor.

Rui Silva, António Cunha and the Sysnovare family, for the patience and confidence put onto me during times of uncertainty, and allowing me to grow alongside them.

My friends, near and far, for always keeping me sane and grounded.

Frederico Portugal Pinho Rocha

*“Approach it and there is no beginning;
follow it and there is no end.
You can’t know it, but you can be it, at ease in your own life.
To realize where you come from: that is the essence of wisdom.”*

Laozi

Contents

Abstract	i
Resumo	iii
Aknowledgements	v
Abbreviations	xv
1 Introduction	1
1.1 Context	1
1.2 Motivation	1
1.3 Objectives	2
1.4 Dissertation Structure	2
2 State of the Art	3
2.1 Recommender Systems	3
2.1.1 Evaluating Recommender Systems	4
2.1.2 Data Preparation	5
2.2 Collaborative Filtering	6
2.2.1 Matrix Factorization	9
2.3 Content Based Filtering	10
2.4 Other Methods	11
2.4.1 Constraint Based RS	11
2.4.2 Context Aware RS	12
2.5 Explanations	12
2.6 Related Works	14
2.6.1 Hybrid Location-based Recommender System for Mobility and Travel Planning [35]	14
2.6.2 A Semantically Aware Explainable Recommender System using Asym- metric Matrix Factorization [1]	15
2.6.3 Explaining Recommendations: Satisfaction vs. Promotion [4]	15
2.6.4 Personalized Explanations for Hybrid Recommender Systems [19]	16
2.6.5 Yelp Recommendation System Using Advanced Collaborative Filtering [12]	17
3 Recommender System	19
3.1 Dataset	19
3.1.1 Files	19
3.1.2 Constraints	20
3.2 Pre-Processing	20

3.2.1	Data Filtering	20
3.2.2	Data Sampling	21
3.3	Evaluation Metrics	22
3.4	Building the Recommender System	23
3.4.1	Similarity Matrix	23
3.4.2	SVD	26
3.4.3	Prediction Model	29
4	Explanations	31
4.1	Goals & Evaluation Metric	31
4.2	Types of Explanations	32
4.3	Post-Recommendation Explanation	32
4.3.1	Individual Item Influence	33
4.3.2	Category Influence	34
4.3.3	Other Influence Factors	34
4.4	Explanation-Based Recommendation	34
4.4.1	Item-Based Recommendation	35
4.4.2	Friends-Based Recommendation	35
4.4.3	Category-Based Recommendation	36
4.5	Discussion	37
5	Conclusions	39
5.1	Future Work	39
	References	41
A	Pseudo Code	45

List of Figures

2.1	Collaborative Filtering	7
2.2	SVD Matrices	10
2.3	Content-based Filtering	11
3.1	Toronto Dataset Classes	22
3.2	System Pyramid	24
3.3	AMSD 1s-Removed Accuracy	26
3.4	Learning Rates Variation by Epoch	28
4.1	Category-Based Recommendation	36

List of Tables

2.1	Functions of explanations	14
2.2	Related Works Characteristics	17
3.1	Toronto Dataset Dimensions	22
3.2	Similarity Models Accuracy	25
3.3	Similarity 1s-Removed Models Accuracy	26
3.4	Comparison with other projects that used the Yelp Dataset	29

Abbreviations & Symbols

ACOS	Asymmetrical Cosine Similarity
AMSD	Asymmetrical Mean Squared Difference
CB	Content-based Filtering
CF	Collaborative Filtering
CFU	Cool Funny Useful
COS	Cosine Similarity
ICF	Item-Item Collaborative Filtering
ISE	Influence Style Explanation
KSE	Keyword Style Explanation
MAE	Mean Absolute Error
MF	Matrix Factorization
MPD	Standard Prediction Deviation
MSD	Mean Square Difference
NSE	Neighbourhood Style Explanation
RMSE	Root Mean Squared Error
RS	Recommender System
SVD	Single Value Decomposition
UCF	User-User Collaborative Filtering

Chapter 1

Introduction

1.1 Context

In the current era of Internet of Things overflowed with Big Data, it has become increasingly complex to deliver relevant information to users. For this purpose alone, search engines already serve a limited solution; however, they do not offer personalized filtering of data, which is where Recommender Systems (RS) come in. RS have the purpose of providing a personalized suggestion of items to users. These are typically implemented in digital marketplaces, where the offered suggestions are products from the marketplace; and web applications, where the recommended items are themselves elements of the applications, such as YouTube videos, Netflix shows, and Facebook friends. These systems work on the basis of incorporating data analysis with machine learning methods and algorithms. The data collected is related to the users' activity in the digital environment, which can include things such as buying a product, visiting a page, and browsing for a type of product; the algorithms use this data to draw correlations between this user's profile and other users', as well as between this user's profile and other items. In other words, recommender systems belong to a class of information filtering that aims to predict the rating given to a certain item by a user, or the likelihood that the user likes such an item.

Starting with the Netflix Prize in the late 2000s, research into Recommender Systems traditionally focused more on mathematical accuracy measures, tied into the matrix factorization methods implemented. Recommender Systems thus operated as a defacto system, only announcing the recommendation it had arrived at, without offering any clarification as to how they achieved such result. However, it is now becoming increasingly common for high-end enterprises to equip their recommendation engines with the ability to provide meaningful explanations for the recommendations generated.

1.2 Motivation

Artificial intelligent systems are usually perceived as black boxes, where the input and output are easily understood, but the processes that return the output consider too many variables and are too

complex to be condensed into a simple equation. As such, while mechanisms like collaborative filtering are very effective in proposing a recommendation, they're poor in clarifying why certain items are being suggested, beyond declaring generic formulas such as 'users who bought item A also bought item B'. It's becoming consensus [27] that evaluating RS only through accuracy metrics and runtime computation complexity is not enough to get the full picture. User satisfaction, trust and result diversity has gained increasing weight when evaluating these systems. It's been proven [2] that, by providing a simple and coherent explanation for a suggestion, a RS is making itself more believable, and its results become easier to be accepted by the user, as it creates an element of transparency and trust. With this, the user is expected to be more active in the environment related to the RS, especially with the items it has been suggested, and are more likely to make a return to it. The main motivation for this project lies in the fact that RS with explanations have limited state of the art documentation of their praxis.

1.3 Objectives

The objectives of this project are twofold. First, it is to implement an illustrative Recommender System, able to pinpoint relevant points of interest and establishments in a city for all kinds of users (tourists and locals). The second objective is to be able to elucidate them by providing clear and effective explanations for the suggestions given. This last objective is the focal part of the project, to achieve meaningful and well understood explanations in a generic RS.

1.4 Dissertation Structure

This dissertation is divided into five sections. The current chapter 1 covered the context into which this project is inserted, as well as the importance of the thesis proposed herein. In chapter 2, we provide a discussion on the state of the art, pinpointing the genesis of this area of studies, the most prominent methodologies applied in creating recommendation systems, the current state of explanations in RS, and an analysis of works related to the problematic at hand. Chapter 3 presents the process used in creating a generic recommendation engine with the existing constraints; along with the data processing, matrix factorization, collaborative filtering, and testing techniques used. Chapter 4 presents the solutions reached for implementing explanations into a RS. In the final chapter 5, we offer the general conclusions reached with this research, detailing possible improvements and further uses for this solution.

Chapter 2

State of the Art

This chapter introduces the main theme of the thesis, recommender systems, as well as the typical steps that take to make one, including data preparation and some common model building methods. It is also explored the state of the art of explanations in recommender systems, such as the contexts in which they should be implemented and to what purposes, as well as the way it's been theorised for them to be integrated in recommender systems.

Finally, some works related to the theme of this thesis are explored, detailing how they built their own recommender system and, in one case, how explanations were integrated into the recommender engine.

2.1 Recommender Systems

Recommender Systems are software tools and techniques that analyse users in an environment and match them with existing items that might be of use. 'Item' is a general term that can encompass any and everything available in a platform that can be recommended to its users. A RS is most commonly applied in: e-commerce, where the users are customers and the items are the products available either for purchase or for clicking; and social media, where the items recommended to users are typically ones that increase the users' interaction levels within the platform, such as other users, groups, topics, events, etc. As such, items are characterized by complexity and utility. The complexity level is correlated to the amount of variables that define an item, where books are low complexity items and financial investments are a higher complexity. The cost associated with them is defined by the acquisition cost, which incurs the real monetary cost, and the cognitive cost of searching for and interacting with the item. User profiles are created based on their behavioural patterns, meaning their shopping history in an e-commerce platform for example, and contextual facts such as their location and age.

When using a platform that has a good recommender engine, a user is more likely to be able to find new interesting items and operate in a different space of options, as well as better narrowing down and solidifying their overall choices. The platform itself benefits from implementing a RS because it allows it to offer a better and more personalized service to the customer. It can increase

short-term sales and customer interaction due to the accurate item suggestions, as well as long-term profit due to the increased trust and loyalty from the user, resulting in bigger user retention. It also allows the system to attain better knowledge about its users preferences.

RS take in a user's profile, including their past activity and maybe some additional context, and compute a relevance score for each recommendable item; this relevance score is based on the predicted utility that each item will have for the user. Recommendation computation is typically modelled around a function of utility $R(u,i)$, where a real valued score R is attributed to the pair user u and item i . The main task of a RS then becomes calculating $R'(u,i)$, where R' denotes the estimated value of R , essentially predicting the R score for all relevant pairs. In cases where the utility function is Boolean, a RS will no longer estimate the utility value for a user-item pair, but will instead determine whether an item is useful or not through the application of heuristic models.

When designing a RS, it must be taken into account which of the item costs are more important to reduce for the user when generating recommendations. In situations where the items recommended are videos, the cognitive cost is the only one that must be reduced, whereas when recommending a car the monetary cost is the one that should take most consideration.

2.1.1 Evaluating Recommender Systems

At their genesis, Recommender Systems were solely evaluated on their prediction power, on how accurate their suggestions were relative to the user's real choices; as the knowledge in the field increased, it became standard to include more metrics for evaluating a good RS. Consumers seek more from a recommendation engine than just an accurate anticipation of their tastes; as such, a system designer must also take into account the user's interest in discovering a diverse array of products, maintaining their privacy, and having a fast, consistent and seamless interaction with the RS. Additionally, an element of trust and transparency between the system and the user can serve to increase the overall experience of the latter.

There exist three main methods for testing a recommender system, each with different overall costs and different focuses. The simplest one requires using existing datasets and user behaviour protocols, to mostly test the system's accuracy power. Alternatively, in order to test for how the system fares with realistic user interactions, a small group user study can be ran, where test subjects are made to perform tasks in the system's environment and then answer a set of questions related to their experience. Lastly, the RS can be tested in a real world application with everyday user interactions. Although this method poses risks to the testing platform, it's the most reliable when evaluating the true value of the recommendation system, because it allows direct measurement of system goals like profit generated and user approval. Additionally, this method allows for different algorithms to be tested at the same time in the same environment, as users can be randomly redirected to a certain recommendation engine and have their various interactions recorded and compared.

The main properties that can define the quality of a RS are: accuracy, confidence, user trust, utility, risk, privacy, scalability and adaptability. [22] Accuracy is most commonly measured using a standard 80/20 cross-validation split for training and testing, where the engine is trained on 80%

of the known data and then made to predict the values for the remaining 20% of the known data; the prediction quality is measured by how low the average Root Mean Square Error (RMSE) (2.1) or Mean Absolute Error (MAE) (2.2) are after a certain number of rounds of training & testing with random batches of known data. Confidence can be defined as the system's trust in its own recommendations; while it can be measured the same way as accuracy, it must also include whether there is over-fitting or under-fitting in the model, as well as how sparse the data is. While confidence is measured by the system's trust in the predictions, user trust is measured by the user's trust in the predictions. Typically measured through questionnaires, this user trust can be increased by the RS occasionally recommending "safe picks" to the user, or by providing explanations. Utility can be measured twofold: one focusing on the utility for the platform implementing the RS, which can be measured by the overall increase in revenue or click; the other focusing on the user's utility from the RS, which can be measured with user questionnaires asking whether the user ended up enjoying or not the recommendation. An assumption that can be made for user utility is the fact that, with two items having around the same predicted value, a recommendation of a 5 star item will incur higher utility for the user than from a 4 star item. Risk accounts for the likelihood that the user will end up losing something with the recommendation; it's a prominent factor in risk-sensitive systems, such as financial advice platforms, and can be generally measured by taking into account the expected utility of the recommendation and the utility variance. The privacy question refers to the user's data remaining undisclosed to third-parties, as well as leak avoidance. Adaptability is measured by how fast a RS can adapt to shifting trends and interests in the platform, in other words, the amount of recent data it needs to make a recommendation. Scalability is related to the overall time and space complexity of the system, or the total resource consumption; it is typically tested by evaluating the variance of speed and resources through different sized datasets.

$$\text{RMSE} = \sqrt{\frac{\sum_{t=1}^T (R_t - \tilde{R}_t)^2}{T}} \quad (2.1)$$

$$\text{MAE} = \frac{\sum_{t=1}^T |R_t - \tilde{R}_t|}{T} \quad (2.2)$$

T is the total sample size

R_t is the real value for t

$\tilde{R}_t$ is the predicted value for t

2.1.2 Data Preparation

When faced with large data sets while building a recommendation system, it becomes necessary to reduce them into smaller, yet relevant and representative, subsets; for such a task, sampling methods are commonly used. Random sampling simply extracts random items from the main dataset into the subset, while stratified sampling first partitions the dataset along particular features

before proceeding to sample these partitions randomly. Stratified sampling is generally used when possible, as it allows for more representative subsets to be generated by reducing sampling bias.

Data sparsity and datasets with high dimensional space are a common issue when handling data for a RS. The number of samples needed to estimate an arbitrary function accurately and the effort exerted when searching for similarities grow exponentially with the number of added variables; for such an issue, dubbed as the curse of dimensionality, it becomes a borderline necessity to apply dimensionality reduction techniques to the datasets used when constructing a RS. Matrix Factorization, the most common method for reducing dimensionality of a dataset, is approached later in this section.

Data noise, introduced into the datasets involuntarily by users, can cause non-negligible effects on the performance of a RS. Natural noise can be reduced by asking users to re-evaluate their choices and ratings, however, striving to completely eliminate noise in a dataset can lead to over-fitting of the engine; as such, it becomes important to almost always include a healthy dose of data noise in the system.

Analysis of the processed data starts with the classification of the different features available. For a restaurant recommender system, a classifier can be implemented so that it predicts a restaurant to be ‘good’ or ‘bad’ according to its describing features.

A classifier is a data mining method capable of mapping labels to feature spaces. It can have one of two types of classifications, depending on the context of the system: in supervised classifications, the set of labels is already known prior to the classification process and will accompany a training set with labelling examples; with unsupervised learning classifiers, the labels are unknown and it’s the classifier’s job to organize the data elements into fitting groups, through methods like clustering. In order to improve scalability and increase system efficiency, clustering can also be further applied to the processed dataset, at the cost of a likely decrease in final accuracy.

In order to test for accuracy, it is necessary to use cross validation with training and testing sets. Sampling can be used on the original dataset to achieve an 80/20 distribution between training and testing sets. The engine is first trained on the training set, and subsequently tested on the labelled examples of the testing set. This process is repeated a relevant number of times, with different generations of training and testing sets, in order to achieve a representative performance value for the engine through the average of all performances. The accuracy of the model can then be calculated as the error of the results between the testing and the training sets.

2.2 Collaborative Filtering

When creating a recommender system, there are mainly two different approaches for handling and filtering the data and reaching predictions. Collaborative Filtering (CF) bases its predictions on the behaviours of the user relative to the other users in the system and the degree of similarity between activity. This methodology [2.1](#) is cemented on the assumption that if certain users share the same behavioural patterns, regarding their interaction with certain items in the system, they are likely to agree about other items; it is thus believed that aggregating the opinions of users similar

to the main user can provide a reasonable prediction of their preferences. Typically, CF algorithms first calculate the similarity value between users, and then calculating the predicted rating for each item using the user similarity values.

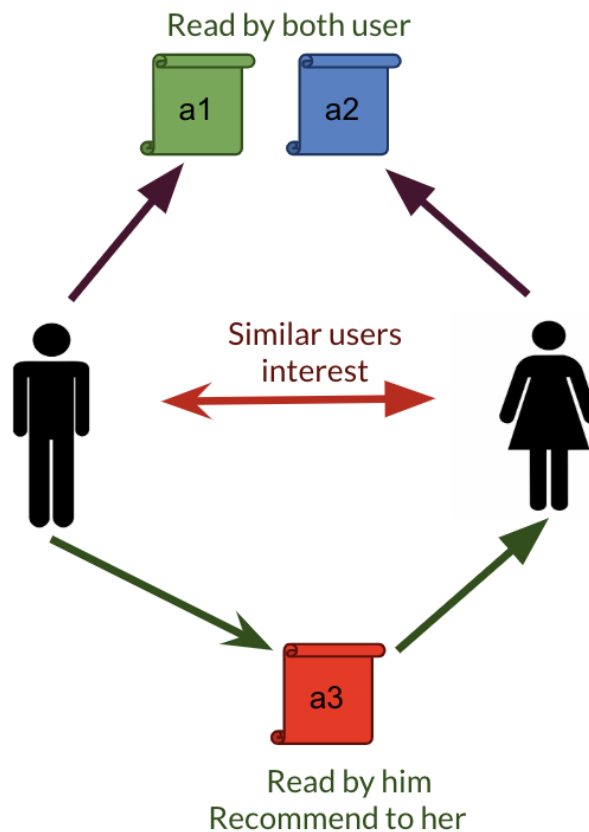


Figure 2.1: Collaborative Filtering

User-User Collaborative Filtering (UCF) is the most straightforward approach to collaborative filtering; it finds other users whose activity profile is most similar to the current user and extrapolates the value the user would attribute to certain items it has not interacted with. The model starts by calculating the user similarity matrix, applying a similarity function for every user pair; it then predicts $R(u, i)$ by comparing the ratings all users a gave to i and the similarity between u and a . There are many variations to this $R(u, i)$ equation, where some include other elements of the environment into the final calculation, but all have this comparison as its core.

The similarity function, which produces the similarity value between a pair of users, is one of the most important steps in building a CF engine. The most studied functions are as follows:

- Pearson Correlation; computes the statistical correlation between two users' common ratings. It suffers from allowing high values for user pairs with very few co-rated items, which is commonly alleviated by setting a co-rated threshold

- Constrained Pearson Correlation; varies from the regular method by assigning an absolute numerical reference, allowing it to compute absolute like/dislike instead of a relative deviation
- Spearman Rank Correlation; like the standard Pearson Correlation model, except ranks are used instead of ratings. The rated items are ranked starting at 1 as the highest rating and lower rated items are assigned higher ranks
- Cosine Similarity; instead of a statistical approach it uses a vector-space approach based on linear algebra. User profiles are represented as n dimensional vectors and the similarity values are generated through the cosine distance between the two vectors in question
- Mean Square Difference; calculates the similarity between user profiles based on the difference between common rated items, much like the Manhattan Distance method

These similarity functions will all generate symmetrical similarity matrices, meaning the similarity between user a and u and vice-versa will be the same; there are, however, asymmetrical alternatives studied, such as the Asymmetrical Cosine Similarity, which present a variation to the formula that, while sacrificing a little more computational power and space, tend to achieve better results. [34]

K-Nearest Neighbour User CF is a greedy variation of the standard UCF that only accounts for the top k users most similar to the user in question when calculating the prediction values; the number of neighbours considered in a neighbourhood can be found according to a similarity value threshold or a set neighbourhood size. This variation is suitable for reducing calculation times, but the neighbourhood size must be reached in a way that the quality of the recommendations remains mostly unaffected. [32]

Item-Item Collaborative Filtering (ICF) was developed to tackle the scalability issues of the UCF method and is nowadays the most commonly deployed collaborative filtering technique. It works by computing the similarity between rated patterns of items; two items are considered very similar if they tend to be liked and disliked by the same users. It differs from content-based filtering due to the fact that, in this case, item similarity is calculated through user preference patterns instead of extracted item data. In UCF, the user's neighbourhood is volatile to change as new ratings are constantly being supplied; however, in ICF, especially when an item already has a substantially big rating matrix, the addition of a new rating by a user is unlikely to greatly affect the similarity matrix between this item and others. As with UCF, there exist a variety of methods for calculating the similarity value between items in ICF:

- Cosine Similarity; calculates the cosine distance between item rating vectors. Is the most popular for ICFs due to being lightweight and relatively accurate
- Conditional Probability; for engines with unary ratings, it calculates the conditional probability between two items belonging to a user's purchase history

- Pearson Correlation; can also be applied for item-item CF problems, but never seems to perform as well as Cosine Similarity
- Mean Square Difference; applies a combination of common ratings ratio, proportion of common ratings (Sorensen Index) and mean square deviation of absolute ratings. These extra calculations make it perform better than cosine similarity in most cases, albeit sacrificing some computation speed

2.2.1 Matrix Factorization

In both UCF and ICF, there are possibilities of representing the user similarity matrix and the item similarity matrix, respectively, as a high-dimensionality vector space. There is expected to be an equally high level of redundancy in these dimensions and missing or empty values; because of this, it becomes necessary to apply dimensionality reduction techniques before proceeding with the prediction calculations. If a model were to simply use only the few known entries in the sparse similarity matrices, it would be highly prone to causing over-fitting.

Matrix Factorization first got introduced into the field of Recommender Systems during the Netflix Prize of 2006-09, proposed by Simon Funk and his Funk MF algorithm. This algorithm decomposed the user-item ratings matrix into two lower dimensionality matrices, the first having a row for each user and n columns, and the second having a column for each item and n rows; each user row and item column described as a latent factor, these would each represent their respective user and item as a n -dimension vector. The calculation for $R(u, i)$ would then be the vector multiplication of row(u) of the first matrix with column(i) of the second matrix. The objective function to be minimized in the original Funk MF model was as follows:

$$\operatorname{argmin}_{H, W} \|R - \tilde{R}\|_F + \alpha \cdot \|H\|_F + \beta \cdot \|W\|_F \quad (2.3)$$

$\|\cdot\|_F$ denotes the Frobenius norm (2.4)

R is the existing ratings matrix and $\tilde{R}$ is the predicted matrix

H and W are the user and item latent factor matrices respectively

α and β are regularization terms added to avoid over-fitting

$$\|A\|_F = \sqrt{\sum_{i=1}^I \sum_{j=1}^J |A_{ij}|^2} \quad (2.4)$$

A variation born from the Funk MF is the Single Value Decomposition (SVD). SVD consists in the factorization of a matrix into three constituent matrices, instead of two. In the case of a ratings matrix R , its SVD would result in the matrices $U\Sigma T^T$, with U representing the users' characteristics across n latent features, and T the items' relevance for each n latent feature 2.2. Σ is a $n \times n$ matrix representing the weights for the user-item preferences across the system. SVD

becomes particularly useful when applied to recommender systems because it can provide the best low-rank approximation of the original ratings matrix while retaining only the first $k < n$ latent features in Σ ; this method, dubbed Truncated SVD, is analogous to only considering the k most relevant features in the system. Currently, the SVD of a complete matrix can be easily achieved in the Python programming language, using libraries such as numpy, scikit or tensorflow; however, there currently exists no library capable of calculating the SVD for incomplete matrices, which is the case for all rating or similarity matrices in recommender systems, requiring machine learning computations of gradient descents.

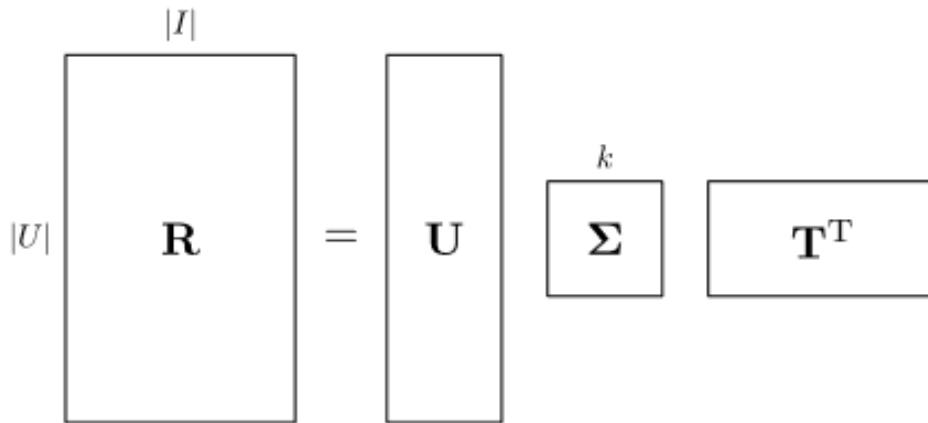


Figure 2.2: SVD Matrices

2.3 Content Based Filtering

The implementation of a Content-Based filtering (CB) [2.3](#) approach on a recommendation system revolves around analysing the descriptions or individual characteristics of items previously rated by a user, followed by building a model of user interests based on those features. The process of singling out items to recommend to the user consists in matching up the attributes of a user's activity profile against the attributes of the existing items in the system.

The basic architecture of a content-based information filtering system consists of three main engines: a content analyser, a profile learner, and a filtering component. The content analyser handles the pre-processing step, usually using feature extraction methods to convert the original data structure into a target one. The profile learner receives this new data structure and employs machine learning techniques to generalise the data into a user profile. The filtering component is the engine that matches the user profile against the items' attribute profiles, resulting in a ranked list of potentially interesting items for the user, ordered by their similarity value with the user's profile.

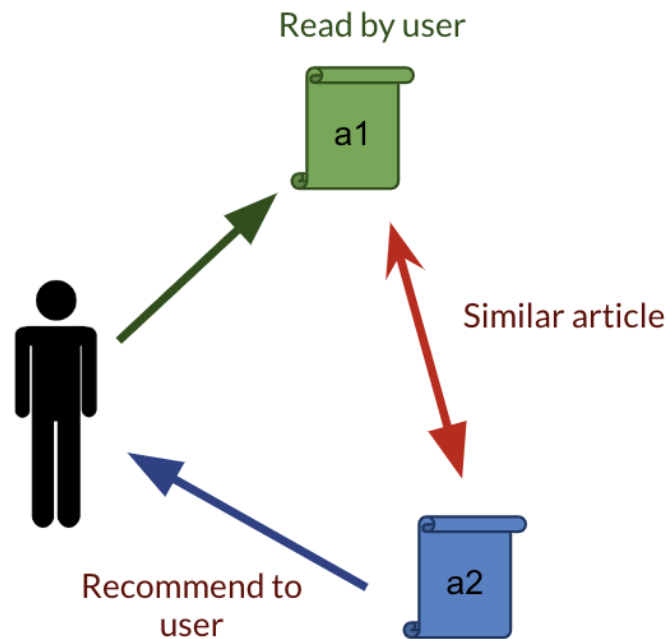


Figure 2.3: Content-based Filtering

As it isn't reliant on the users having relatively complete profiles already built from their interactions with the platform, this method is very good at handling Cold Start issues that would otherwise arise from the use of the Collaborative Filtering method; due to this reason, creating recommendation systems with a hybrid model that makes use of both Content-Based and Collaborative Filtering methods quickly became an attractive option in domains where it's possible to easily differentiate items by their characteristics [24]. There are many ways to approach the construction of a hybrid model, as the combination of both methods must be applied such that the prediction quality of the overall model doesn't suffer great losses. The 3 simplest approaches are described as: weighting each model's predictions and combining them into one prediction; switching between which singular recommendation technique is used depending on contextual criteria; presenting a mix of recommendations from both methods.

2.4 Other Methods

2.4.1 Constraint Based RS

The two traditional recommendation approaches are well suited for recommending products that revolve around quality and taste, like books, films, phones, etc; however, in more complex recommendation scenarios where retrieving substantial ratings becomes impossible, for items such as apartments and financial services, a knowledge-based approach becomes necessary. This approach works by instead exploiting explicit user requirements and deep knowledge on the product's domain.

An example of this approach is constraint-based recommenders, which exploit predefined rules about the relationships between user requirements and item features. These two are typically understood as the variables in a recommender knowledge base, whereas three specific sets of constraints are used to filter user-item pairs into recommendation. The first two sets restrict the possible instantiations of user and item properties, respectively, while the third set defines the possible relationships between both users and items. This approach does not suffer from cold start like collaborative filtering, however, pre-processing the knowledge from domain experts into executable representations is generally an arduous task, known as knowledge acquisition bottleneck.

2.4.2 Context Aware RS

It is commonly argued that context comes heavily into play when it's necessary to provide recommendations on an environment and that a third entity, contextual information, should accompany the traditional user+item model; as an example, temporal context is very necessary for recommending vacation plans, since winter and summer vacations destination vary greatly.

A common approach for creating a Context Aware RS is, for each relevant context, modelling different profiles for the same user. For a video RS environment like YouTube, for example, it could be created a user profile for the afternoon and one for the late night, as user activity is expected to vary for both contexts. Another example, in an environment like Spotify, would be the modelling of user profiles for Fridays and Mondays, as the user's mood is expected to be very different for both days of the week; as such, the classic model for predicting utility, $R(u, i)$, would become $R(u, i, c)$, where c represents the specify context where the prediction is being calculated on. This approach is called Contextual Pre-Filtering, as the context is applied to the data before the prediction filtering is done. Conversely, Contextual Post-Filtering represents the method where context is applied to the generated recommendations, as a means to further filter the generated list into a final list of recommendations.

While Context-Aware RS are able to provide more personalised recommendations, it suffers from lack of flexibility. In order to avoid hard-wired and pre-determined contexts into the system, it would be required for the user to be asked to input their situational context into the system every time they interact with it. Additionally, while the recommendations are more personalised, the existence of various types of contexts leads into increased levels of complexity in the system, which can become redundant and create less accurate predictions.

2.5 Explanations

Advances in Artificial Intelligence (AI) and the adoption of AI systems across many different industries came alongside the continuous increase in these models' complexity. The general ambiguity surrounding these systems has become a roadblock in their mass integration with more critical domains, where the reaped benefits of the integrations are large but the cost of malfunction could be fatal, as is the case with AI in healthcare; as such, the study of Explainable AI has gained great importance over the course of the last decade. Explainable AI started very early with

white-box AI models, such as decision-trees, as despite their lacking performances compared to more complex systems, the explanations were very easily achieved. Explainable AI related to the more complex black-box models began in the 1970s, where simple models built around heuristic-based systems were capable of presenting which of the rules had the biggest contribution to the systems' decisions [6] [10]. Nowadays, non-linear black-box models such as neural networks require more complex approaches for reaching the same level of explainability, like the LRP method proposed in [21], which determines the input vectors' features that contributed the most to the neural network's final output; or LIME [37], which trains a linear model in the neighbourhood of the prediction made by the underlying model, by removing features of the original instance and comparing the perturbed instance's result to the original one. In recent times, explainability in Recommender Systems specifically has gained importance due to the increasing recognition of user satisfaction and interaction as an evaluation metric for these systems.

Explanations have many simple functions in a system 2.1. Users tend to return to systems they find trustworthy and transparent; for this effect, explanations can make more apparent to the user what the heuristic and reasoning behind a recommendation are, which in turn can make them more forgiving of recommendations with poor accuracy. Explanations can help the user in better understanding the system behind the suggestions, making it easier for them to correct behaviours they find are resulting in bad recommendations.

The most sought after qualities from explanations is effectiveness and efficiency. Effective explanations are ones that help the user make better decisions, discarding irrelevant options while helping them recognise more relevant ones. An efficient explanation allows the user to better comprehend the relation between competing options, turning their decision-making process into a faster and easier one.

The way explanations are done varies greatly with how the recommended items are being presented and with how the recommender system is built. When the recommendation is the top item or n-items, the explanation can be based around the process of selection, eg. the user's history: "because you've been interested in item or topic x". When the item is chosen through a content-based system, an explanation around feature similarity might be useful, as in "because you bought x, we recommend y". Similarly, when the item is chosen through collaborative filtering, a different explanation might be more useful, such as "users who bought x also bought y". A different methodology towards approaching explanations is structuring them around other items' features trade-offs, for example "phone x has a good camera, while phone y has a good processor"; this approach makes more sense in expert recommender systems with pre-existing knowledge databases.

The evaluation of explanations is commonly done through user questionnaires. When testing for the level of transparency, for example, a user can be asked if they more or less understood how the process of finding a suggested item works. Testing for effectiveness and persuasiveness can be achieved by questioning the user on its interest in consuming an item before and then after receiving the corresponding explanation. Other qualities, such as scrutability and efficiency, are better evaluated through task-based scenarios; respectively, the user can be asked to make the

Table 2.1: Functions of explanations

<i>Function</i>	<i>Definition</i>
Transparency	Explain how the system works
Scrutability	Allow to inform the system that it's wrong
Trust	Increase the confidence in the system
Effectiveness	Help make good decisions
Efficiency	Help make decisions faster
Persuasiveness	Be convincing
Satisfaction	Increase ease of usability

system stop recommending certain types of items, or test for how long it takes for the user to locate a desired product in the platform.

2.6 Related Works

For related works, the first one studied proposes an improvement for collaborative filtering when building a travel recommender system. The second one proposes a collaborative filtering matrix factorization method with explanations based around item features. The last one proposes three different explanation methods for content based and collaborative RS.

2.6.1 Hybrid Location-based Recommender System for Mobility and Travel Planning [35]

This study offers a take on POI and travel recommender systems, using collaborative filtering alongside an ensemble based co-training method with swarm intelligence algorithms.

This ensemble approach was taken in hopes of handling data sparsity in the user-item matrix, as well as the common cold start problem that follows collaborative filtering. The co-training method works by extrapolating from at least two views of labeled data in a dataset, and making predictions for the unlabeled data. The different views are generated by the swarm intelligence algorithms, where each algorithm clusters the original data and uses its result, combined with the results of the other algorithms, to learn from and continuously improve its clustering model.

The result of this ensemble is the original data with added predictions for unlabeled pairs, which allows the collaborative filtering algorithm to work with more complete initial data.

2.6.2 A Semantically Aware Explainable Recommender System using Asymmetric Matrix Factorization [1]

This study attempted to combine the Semantic Web platform with a black box collaborative filtering RS, built around matrix factorization, in order to make recommendations that can be easily explained. The Semantic Web is a platform of linked data, which has been used before in the Web Mining field and in building meaning-aware user profiles. The semantic data from the SW can be a source of knowledge for the system about both users and items, which can then be used to create meaningful explanations from the relations between both.

In order to keep the explanations consistent with the recommendations, it was built a matrix factorization model that, not only, included the user-item ratings, but also the potentially meaningful explanations for those ratings. This MF method consists of two steps, knowledge foundation and model building.

Using latent matrix factorization, it first measures if a semantic feature belongs to an item; in our case, a restaurant category such as serving vegetarian food. It then measures the number of times a semantic feature appears in items liked by the user; in our case, it could measure the average rating items with the vegetarian category have with the user. This project considers simply a binary like/dislike rating system, while ours has ratings ranging from 0 to 5. The two matrices generated are then combined into an explainability score matrix, which indicates the average rating the user would give an item, based on the semantic features it might possess and how they are preferred by the user.

In the model building step, this semantic latent space model is factorized, into asymmetrical user and item spaces, and used to update the, similarly factorized, original ratings matrix. In this step, weak explainability scores are overtaken by the known ratings, while the opposite happens with high explainability scores. In the end, the model is able to provide recommendations, and base its explanations on the user's interest in the item's semantic properties.

2.6.3 Explaining Recommendations: Satisfaction vs. Promotion [4]

For this project, it was developed three different approaches for generating explanations on a content-based and collaborative RS.

The first one, a Keyword Style Explanation approach, analyses an item and finds the strongest match between its features and the user's preferred features. This method synergises with content-based filtering, as it is based around calculating item similarities. The second method, Neighbour Style Explanation, analyses the active user's neighbourhood of similar users and generates an explanation around how those users rated the active item. This method synergises with collaborative filtering, as it needs to have user similarity calculations done beforehand.

The last method, Influence Style Explanation, presents to the active user a list of items, which the user has already rated, that most influenced the current recommendation. To calculate the influence of an item, in a content based system, the item's rating is first removed from the user's ratings matrix, then its rating is predicted by the system using the new user matrix. The numerical

influence is then taken as the difference between the item's real rating and its predicted rating. This method can, thus, work with both types of filtering algorithms, as it functions independently of what prediction algorithm is applied to the new user matrix, in every item case. In a Content-boosted Collaborative Filtering hybrid system, the influence of an item is measured as the average influence for said item, after running both the content-based and collaborative algorithms separately.

In order to evaluate each explanation method, an online user study was conducted, where each user was presented a recommendation alongside its explanation and asked to rate the recommendation. Afterwards, the user was asked to try the recommended item and rate it again. A good explanation would, in this case, minimize the difference between both ratings.

According to their results, the Neighbour Style Explanation method performed poorly, as it caused users to overrate an item, which could not be perceived as a bad thing in some scenarios; the Keyword Style Explanation and Influence Style Explanation explanation allowed users to accurately estimate an item's value.

2.6.4 Personalized Explanations for Hybrid Recommender Systems [19]

This study implements a recommendation system for songs based on probabilistic soft logic [18], that uses a template set of logic rules to determine the probability of a user listening to an item, and generates different styles of explanations based on the most dominant rule. This set of rules makes use of user similarity and item similarity calculated through collaborative filtering, item similarity based on content-based filtering and popularity information and friend relationships; as such, this RS is able to operate on a hybrid of data from different sources and filtering methods.

The explanation styles implemented are based on: user similarity, item similarity, item content, social, and item popularity. As each explanation style directly correlates to one of the logic rules, the recommendations generated will have personalised explanations according to the most determining rule in the recommendation. As an example, the user-based explanation style takes root in the rule 2.5, which means that if during the inference stage the predicted LISTENS value from this rule is superior to the ones from other rules and the item is recommended, the explanation that follows is one based on the users most similar to them.

$$\text{SIM}(U_1, U_2) \wedge \text{LISTENS}(U_2, A) \longrightarrow \text{LISTENS}(U_1, A) \quad (2.5)$$

Finally, a user study was conducted on 198 test users to figure out which explanation styles worked better, taking into consideration users' personality traits, and whether the number of explanations presented influenced the overall persuasiveness. It was also tested for visual vs. textual presentation of explanations. As for the styles persuasiveness scores, explanations based on similar users and friends with similar tastes ranked the lowest, while collaborative item similarity and item content similarity scored the highest. With regard to personality traits, only the ones belonging to the larger group of neurotic traits seemed to have the best receptiveness to explanations,

with calm people responding better to popularity-based explanations and anxious people responding better to explanations from collaborative item similarity.

2.6.5 Yelp Recommendation System Using Advanced Collaborative Filtering [12]

This project was based on the Yelp public dataset, with the aim of predicting the ratings that each user would give to all restaurants using collaborative filtering in conjunction with some variations of K-Nearest Neighbours techniques.

The algorithm was based on user-user similarity, which was accomplished using a simple Euclidean distance calculation for users' common restaurant ratings. In order to solve for data sparseness, restaurants that served similar types of food (eg. Pizza, Vegetarian, Burgers, etc) were aggregated into clusters and treated as only one; users would now have their similarity measured in regards with other users by their average ratings for each food type cluster. This model achieved better results than the standard baseline model. An improvement upon this model was made by creating an additional layer for clustering, by now also including the restaurants style characteristics (eg. casual, classy, trendy, etc) alongside the type of food they served. This created more clusters, which managed to achieve better results in favour of more sparsity.

Table 2.2 presents the condensed information for each work studied in this chapter, mainly the filtering algorithm used in the creation of their RS and the major finding of the paper.

Table 2.2: Related Works Characteristics

	<i>Method</i>	<i>Particularity</i>
Logesh et al., 2019 [35]	Collaborative Filtering	Co-Training Method with Swarm AI to handle data sparsity
Alshammari et al., 2018 [1]	Collaborative Filtering	Using matrix factorization on the items' Semantic Features and ratings matrix to reach explainability scores
Bilgic et al., 2005 [4]	Collaborative & Content-Based Filtering	Introduction of different styles of explanations, such as Item Influence, for different filtering algorithms
Kouki et al., 2019 [19]	Collaborative & Content-Based Filtering Hybrid	Probabilistic RS model able to take in different data sources and allowing for various styles of explanations
Chee Hoon Ha, 2014 [12]	Collaborative Filtering with KNN	K-Nearest Neighbours clusters of item characteristics as the basis for user-user similarity collaborative filtering algorithm

Chapter 3

Recommender System

This chapter starts by introducing the dataset used in this project and its characteristics, as well as the constraints associated with it. It is presented the pre-processing steps taken to adapt the original dataset into a format suitable to be used by the recommender system, and which responds to its desired functionalities. Having pre-processed the data, we explain the process of building the recommender system, including the generation of the similarity matrix and the calculations that use said matrix to reach the predicted rating values. This chapter ends with a conclusion on the building of the recommender system, by analysing the evaluation metrics and the values reached when testing.

3.1 Dataset

The dataset used is the one publicly provided by Yelp [49]; it is comprised of 3.6 GBs of .json files, as well as a function for converting them to .csv files. The three types of .json files that will be useful for creating this recommender system are named `business.json`, `user.json`, and `review.json`. Respectively, these files contain data over each business and user existing in the system's environment, as well as the data related to the interactions between users and businesses. Each entry in these files is marked by its own unique ID.

3.1.1 Files

An entry in the `business.json` file contains data regarding the business name, location: city, address, state, postal code, latitude and longitude; average stars rating, number of reviews, attributes: if it has takeout or parking, etc; categories: restaurant, vegan, nightclub, etc; and opening and closing hours.

`User.json` contains the users' name and account creation date, their average ratings score, the total number of reviews, a list of their friends in the platform, a list of years in which the user had the elite status, the number of different types of compliments ('cool', 'funny' and 'useful' or CFU) the user sent out and received.

In each entry in the Review.json file we have the user's ID (reviewer), the ID of the business reviewed, the rating given from 1 to 5 stars, the date of the review, the review text, and how many CFU votes the review got from other users in the platform.

3.1.2 Constraints

Many of the features of each type of data are ultimately irrelevant for the recommender engine; although a robust system could be built using the 'cool', 'funny' and 'useful' votes feature, for example, the RS built in this project has no such aspirations. Despite that, features such as business categories can become important later when implementing explanations into the RS; it's important to be mindful of such nuances when preparing the dataset.

The businesses included in this dataset hail from various cities in the USA and Canada. The cities with most businesses present are Las Vegas, USA; and Toronto, Canada. In terms of user-item ratio in the dataset, the full dataset contains 2 million user accounts, 200,000 businesses, and 8 million reviews; meaning, there is a 10-1 user to item ratio. These entry numbers must be reduced, as the recommender engine would take too much time in its calculations otherwise; however, the way in which the sampling is done is very important to the overall quality of the resulting reduced version of the dataset, as the user-item ratio can be greatly altered from the original dataset and the number of interconnected users can become very low. By interconnected user we can understand as a pair of users which has rated the same item.

3.2 Pre-Processing

As previously stated, the dataset must be subject to some pre-processing so it can be usable by the recommender engine. In this section, we explore the pre-processing steps followed in this project, as well as ones which were considered as alternatives but ultimately put aside.

3.2.1 Data Filtering

The data filtering step consisted in selecting only the necessary features in each element type of the dataset, in a way that reduced processing speed while retaining enough information for the RS to extrapolate on. In terms of user entries, as identification measure it was kept the user ID and the name. For future calculations, it made sense to keep the average stars rating as well. Furthermore, the elite years feature, as list which kept account of the years in which the user was an elite user, was transformed into a counter of how many years the user had been elite, and kept for posterity in the explanation phase. For this purpose as well, it was also kept the lists of the users' friends' IDs. In the business entries, like with the users, it was kept the alphanumeric business ID and name. It was also kept the business' stars rating in the platform for the prediction calculations, and the list of categories in which the business belongs to, as a possible means expanding the explanations. Additionally, to facilitate the SVD related calculations, it was created a numeric

Matrix ID parameter for each business, starting with 0. With the reviews entries, it was only kept the business and user IDs and the stars given in the review.

With regards to both the reviews and the users profiles, since the 'cool', 'funny', and 'useful' attributes are expected to share some linear correlation between themselves, it was hypothesised to be applied dimensionality reduction in the way of Principle Component Analysis to the User-CFU matrix and Review-CFU matrix, in order to perhaps extract some importance of the user's opinion or the review's value. In the end, this approach was scrapped and, with the users, replaced with the far simpler elite years counter approach.

3.2.2 Data Sampling

In order to further reduce the size of the dataset, it was used data sampling techniques. The aim was to reduce the near 8 million review data points into a more manageable number; regardless, since the first action on the dataset was to be the creation of the item-item similarity matrix, the aim was ultimately to reduce the 200,000 businesses into a more manageable amount, although the reviews number reduction remained a good metric for comparison with other benchmarks.

One of the sampling techniques that was first tried for being the most intuitive, but later rejected, was random data sampling. It consisted in first selecting only a percentage of random businesses from the dataset, followed by selecting all the review entries done about those businesses, and finally selecting all the users who made those reviews. This approach was also tried starting with random user selection, followed by their reviews and then the businesses reviewed by them; however this method quickly showed flaws by the quality of the sample datasets generated. The balance of the user-item ratio in the sample datasets was completely inverse to the ratio in the original dataset, with a consistent 1-8 user to item ratio; this ratio would suggest a user-user similarity matrix approach instead, but the main problem with the randomly sampled dataset was a clear low number of interconnected users, which would make a user-user similarity matrix extremely sparse. This can be explained by the dataset containing business entries in various cities, thus a random selection would necessarily end up containing users in different cities with no common business interactions between them.

To solve this problem, a stratified sampling approach was followed. This approach started by selecting all businesses from a city, followed by the reviews of those businesses and then the users who made those reviews. This approach was tested in three cities of differing sizes (number of business entries), with Las Vegas, Toronto, and Mississauga, in descending order. By centralising the user activity to a single city, the user interconnectivity was brought up to a maximum, with the user-item ratio being even bigger than in the original dataset. After building the item-item similarity matrix, applying SVD extrapolation to it and testing for its accuracy with the datasets of all 3 cities, it was concluded that the reduced size of the Mississauga dataset did not incur any penalties to the model's accuracy, so the city's sampled dataset was initially chosen. This is explained in greater detail in section 3.4 of this chapter. However, since the number of reviews was relatively low compared to other projects' benchmarks, at only 60k, another dataset sampling approach was tried. It was tried the same stratified sampling approach expect with the

condition that only users with more than 50 reviews made in the city would be selected; this generated a dataset for the city of Toronto with around the same number of businesses as the previous Mississauga dataset, but with 4 times more users and with almost 400k review entries. When tested for accuracy, the model generated with this dataset greatly outperformed the original Mississauga dataset model, with negligible performance time increase. Thus, this new restrictive Toronto dataset was selected for the recommender system, with its attributes seen on Table 3.1 and Figure 3.1.

Table 3.1: Toronto Dataset Dimensions

Users	97 511
Items	2 846
Reviews	388 188

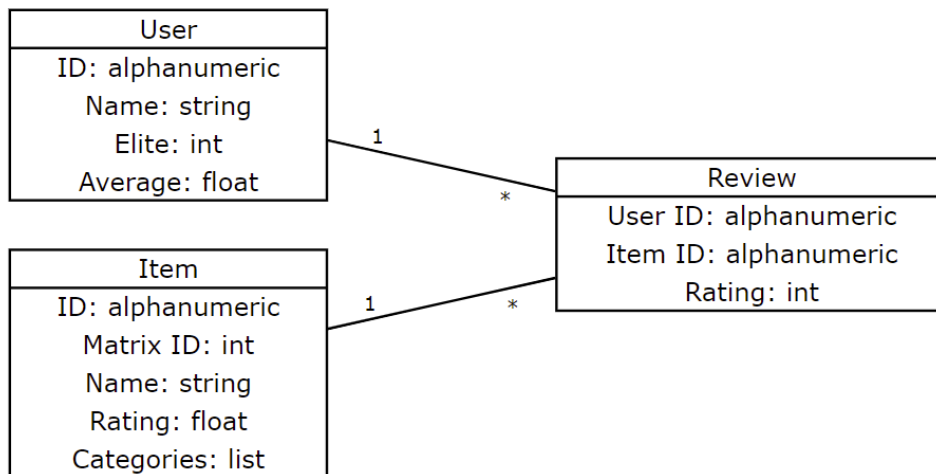


Figure 3.1: Toronto Dataset Classes

3.3 Evaluation Metrics

The system will be tested with state of the art benchmarks, mainly for their prediction strength, coverage, diversity, and confidence. The system's evaluation will be sectioned into two parts: evaluating the quality of the item-item similarity matrix's SVD and evaluating the quality of the recommender model's ratings predictions.

With regards to the SVD, the accuracy will be achieved through the RMSE between the testing results and the training data when applying the SVD to the similarity matrix. The confidence will be measured by checking if the model is either overfitting or underfitting when measuring this

accuracy; good confidence would imply the difference between the training error and validation error isn't too drastic and that the model is generalising well.

With regards to the recommender model, the coverage will be measured as the proportion of users which the system can confidently give recommendations to. Diversity of recommendations will be measured by inspecting the recommended items' categories and comparing them to the user's most liked items'; good diversity in recommendations would suggest the user isn't being recommended exclusively items from the same categories as his favourite items. For accuracy, the model will be tested by predicting known ratings for a meaningful percentage of the user base and comparing the results with the actual ratings; this size limitation is imposed due to the time and processing power it'd take to predict the value of every item rated by every user. This evaluation will use the metric RMSE (3.1), which will provide the absolute value comparison between the real ratings and the predicted ratings; this metric was the one used to evaluate the results in the famous Netflix contest and is known to penalise poor predictions more than MAE, as the errors are squared, thus making it a good candidate to be used in this project.

$$\text{RMSE} = \sqrt{\frac{\sum_{u,i} (R_{u,i} - \tilde{R}_{u,i})^2}{T}} \quad (3.1)$$

R and $\tilde{R}$ represent the real and predicted ratings respectively

T represents the total number of user-item pairs tested

3.4 Building the Recommender System

The approach to building this recommender system is one of building a pyramid, where a building block is sustained by the previous one, shown in Figure 3.2. In this case, the system is composed of three major building blocks, where each serves as the base for the next one. The first building block is the incomplete item-item similarity matrix; by applying the SVD algorithm to it we get the next building block in the complete item-item similarity matrix; finally, the prediction model uses this complete matrix to achieve its ratings predictions. As an extension to this metaphor, the explanations model is the tip of the pyramid, as it is based on the results from the prediction model.

3.4.1 Similarity Matrix

As discussed in chapter 1, the recommender system was to be built with collaborative filtering techniques. In the subsection 3.2.2 of this chapter, it was presented the reasons as for why an item-item similarity approach was most suited compared to user-user similarity, with regards to the dataset being used: a significantly lower amount of items than users makes the item similarity matrix more manageable, and allowing for better processing times and lower sparsity than a possible user similarity matrix.

Literature suggests a few different ways of calculating the similarity values between items; for this project, it was experimented and measured for accuracy and speed the following methods:

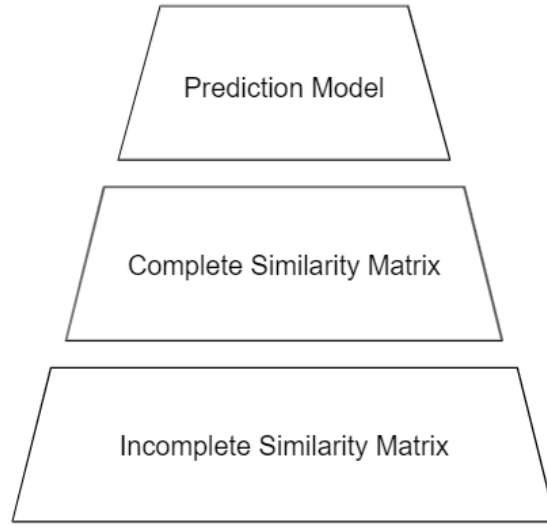


Figure 3.2: System Pyramid

cosine similarity (COS), asymmetrical cosine similarity (ACOS), and asymmetrical mean squared difference (AMSD). Cosine similarity, for a given pair of vectors, can be described as the dot product between each vector divided by the sum of every element in the vectors (3.2). In the case of this project, a vector is represented as all user reviews for a single item. The dot product between two item vectors will be 0 if neither item has mutual user ratings between each other.

$$\text{COS}(a, b) = \frac{I_a \cdot I_b}{\|I_a\| + \|I_b\|} \quad (3.2)$$

I_a represents the user reviews for item a

$\|I_a\|$ represents the number of user reviews for item a

The two asymmetric item similarity techniques are an adaptation of the asymmetric user similarity techniques presented in [34]. To make the traditional cosine similarity equation become asymmetric, it is introduced the ratio of common ratings and the Sorensen Index (3.3). The first element inherently introduces asymmetry by normalizing the mutual user ratings by only one of the item's ratings; this however does not account for the total number of common ratings between users, only the proportion, so the latter element is introduced.

$$\text{ACOS}(a, b) = \frac{I_a \cdot I_b}{\|I_a\| + \|I_b\|} \cdot \frac{\|I_a \cap I_b\|}{\|I_a\|} \cdot \frac{2 * \|I_a \cap I_b\|}{\|I_a\| + \|I_b\|} \quad (3.3)$$

$\|I_a \cap I_b\|$ represents the number of mutual user who reviewed both item a and b

The last similarity technique is based on the mean square difference (3.4) of the common ratings between two item vectors, with the introduction of the previous two asymmetry elements and a balancing element L equal to the maximum possible squared difference between ratings (3.5). If L were smaller than the biggest possible rating difference square, 16 in this case, it would also serve as a threshold, discarding item pairs with poor similarity; this approach was not taken, as to not increase matrix sparsity. With regards to processing speed, all three techniques had relatively similar times when applied to the same datasets, despite ACOS and AMSD incurring some extra calculations compared to COS.

$$\text{MSD}(a, b) = \frac{\sum_{u \in \|I_a \cap I_b\|} (I_{a,u} - I_{b,u})^2}{\|I_a \cap I_b\|} \quad (3.4)$$

$$\text{AMSD}(a, b) = \frac{L - \text{MSD}(a, b)}{L} \cdot \frac{\|I_a \cap I_b\|}{\|I_a\|} \cdot \frac{2 * \|I_a \cap I_b\|}{\|I_a\| + \|I_b\|} \quad (3.5)$$

Finally, after applying the matrix factorization algorithm to the three similarity matrices and testing for accuracy, all matrices performed very similarly when it came to the RMSE, being stuck in this local minimum of 0.025. However, as the models were built using different similarity equations, the resulting similarity values are different orders of magnitude despite all being in the same $[0,1]$ scale; thus the RMSE has to be contextualised. For this case, measuring the ration between the RMSE and the mean of the values produced, the asymmetric methods proved to be substantially better than the regular cosine similarity method, as seen on Table 3.2.

Table 3.2: Similarity Models Accuracy

<i>Method</i>	<i>RMSE</i>	<i>Mean Similarity</i>	<i>Error-Mean Ratio</i>
COS	0.02523	1.16e-6	21750
ACOS	0.02524	6.01e-4	41.99
AMSD	0.02497	1.17e-3	21.34

However, following the steps of some examples in the literature, it was also tested the accuracy for similarity models where the similarity values for the same items were never calculated, as these would always result in 1 and could lead the model to underfit. As shown in the Table 3.3, most methods performed better under this approach, except for ACOS. As such, the item similarity matrix built with the AMSD 1s-Removed method was chosen for this project. Figure 3.4 shows the learning rate variation for the chosen similarity method across 100 epochs of training. The appendix section 1 shows the steps required to recreate the AMSD method through pseudo-code.

Table 3.3: Similarity 1s-Removed Models Accuracy

<i>Method</i>	<i>RMSE</i>	<i>Mean Similarity</i>	<i>Error-Mean Ratio</i>
COS	8.45e-4	1.74e-4	4.843
ACOS	1.58e-5	1.95e-7	81.37
AMSD	8.96e-4	4.94e-4	1.813

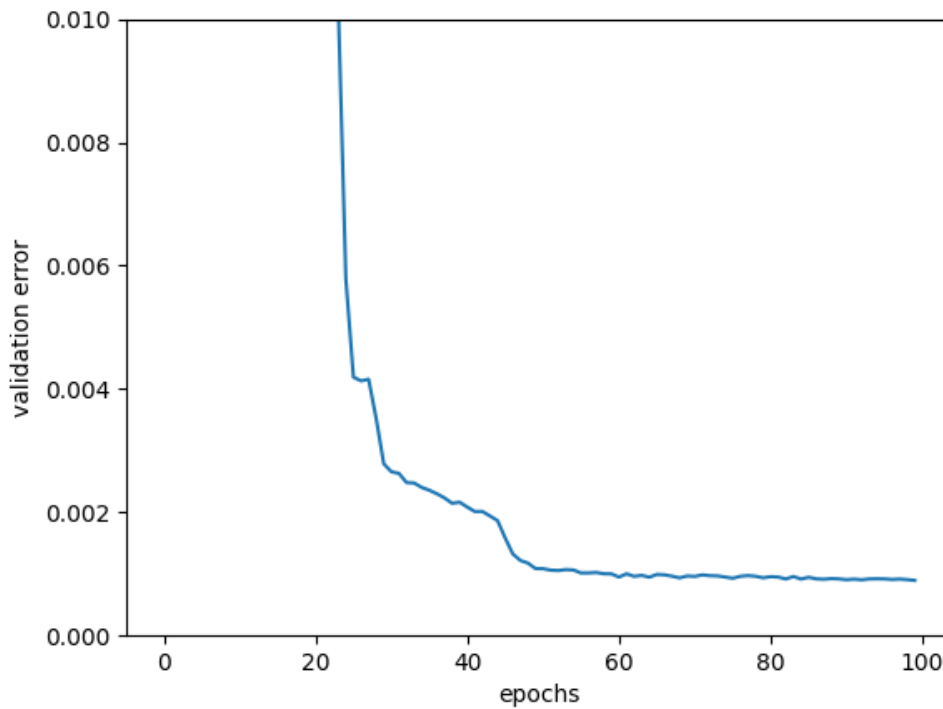


Figure 3.3: AMSD 1s-Removed Accuracy

3.4.2 SVD

The second step in building this recommender system was turning the incomplete and sparse item similarity matrix, built in the previous step, into a complete matrix, to be later used as a base to calculate rating predictions. As discussed in the state of the art, chapter 2, matrix factorization methods such as Single Value Decomposition are very often used in recommender systems, as they can produce a faithful low-rank approximation of a sparse ratings matrix or, in this case, item similarity matrix.

The SVD algorithm used in this project is an adaptation of the one from [20], using the Tensorflow machine learning environment. Tensorflow allows us to perform the derivative calculations of an incomplete matrix factorization algorithm with automatic differentiation, meaning it builds a computational graph which tracks the data flow through the operations, automatically calculating

their derivatives and filling in the derivatives of each parameter when backpropagating the graph; this greatly expedites the process and ultimately means we only have to write the inference and optimization equations, marking which variables and embeddings should be tracked.

The main difference in this adaptation lies in the cost and optimization functions, which had to be adapted since the input matrix was different, while the base of the algorithm remains somewhat faithful to the original; throughout testing, the hyper parameters were repeatedly tweaked in order to achieve the best results. The algorithm consists of epochs, each one containing a training and a testing phase; at the start of each epoch, a random 80% section of the dataset is assigned to the training phase, while the remaining 20% is for testing afterwards; the training set is then divided into batches, which are used sequentially in place of the whole training set during the training phase. Both these measures are to avoid overfitting the model. Only after every batch from the training set has been exhausted does the algorithm test the inference equation on the entire testing set.

Since Tensorflow performs automatic differentiation, we only need to program the inference part, which includes the inference matrix and the optimization function. As detailed in chapter 2, SVD consists of decomposing a high-dimensionality matrix into two lower-dimensionality matrices and a square latent factors matrix. When applying this logic using derivative calculation, the equation becomes one such as the one from (3.6). When comparing it to 2.2, I_U and I_T correspond to the item's latent feature matrices U and T ; since the item similarity matrix is asymmetrical, $I_{U,1}$ represents the $\text{sim}(1, x)$ latent features of item number 1, while $I_{T,1}$ represents the $\text{sim}(x, 1)$ latent features of item number 1. The latent feature weights square matrix Σ is represented by the bias arrays b_U and b_T and the global bias value μ . Each bias array has an entry for each item in the respective matrices. After the model has been trained, $\tilde{S}(a, b)$ gives us the predicted similarity value for the pair (a, b) , even if the item pair has no previous user interactions in common. The size of the I_U and I_T matrices is dependent on the number of items in the dataset and a determined number of dimensions, representing the number of latent features per item; if the model appears to be underfitting, the number of latent features and the size of the batches can be increased accordingly. For this case, what appeared to offer the best results was 20 latent feature dimensions and batch size corresponding to 30% of the training set size, throughout 100 epochs.

$$\tilde{S} = \mu + b_U + b_T + I_U \cdot I_T \quad (3.6)$$

The objective of the algorithm is to minimize the optimization function result (3.9), or model cost, while solving for the inference equation. The optimization function is characterized the elements: L2Loss Function, Frobenius Norm of the two Latent Feature Matrices, and a Zeroes Penalty function.

$$\text{L2_Loss} \leftarrow \frac{\sum_{i=1}^I |S_i - \tilde{S}_i|^2}{2} \quad (3.7)$$

$$\text{Zeroes_Penalty} \leftarrow \left\| \frac{|\tilde{S}| - \tilde{S}}{2} \right\|_F \quad (3.8)$$

$$\text{Cost} = \text{L2_Loss} + \lambda \|I_U\|_F + \lambda \|I_T\|_F + \text{Zeroes_Penalty} \quad (3.9)$$

The L2 Loss Function (3.7) can be calculated as the squared difference between the real and the predicted values, or the Euclidean Norm of the difference, which is the main objective for minimization in the optimization function. The Frobenius Norms (2.4), along with the penalty λ , act as regularizers to let the model generalise well and prevent overfitting. The additional Zeroes Penalty (3.8) function serves as a deterrent from the model producing negative similarity values, which should theoretically be impossible to exist based on the similarity functions from previous steps; this became necessary as the original similarity values calculated were very close to 0 and the model wouldn't know to avoid negative numbers otherwise, and can be defined as the Frobenius Norm of the Latent Feature Matrices where all positive values are turned into 0.

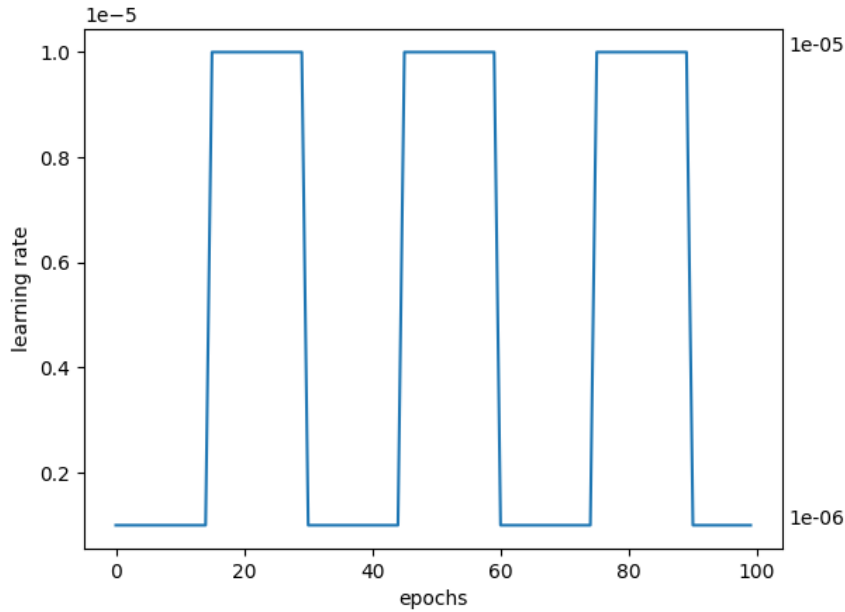


Figure 3.4: Learning Rates Variation by Epoch

For this project, the model cost was plugged into the Adam Optimizer function provided by the Tensorflow environment, which is an implementation of the Adam stochastic gradient descent method; this method is suitable for optimizing the similarity matrix as it deals well with sparse gradients and shows computational efficiency [17]. As for the learning rate, it was chosen the approach similar to the one proposed in [41], in which the learning rate varies between minimum and maximum bounds over intervals. This method is said to facilitate the gradient descent from avoiding saddle points, by offering higher learning rates periodically. While this method is based on continuously updating the learning rate every epoch between the min and max bounds, the method implemented in this project only updated the learning rate directly from the min to the max bound and vice-versa once every few epochs, displayed in Figure 3.4, as it was shown to

provide better results for this specific project. The complete pseudo-code for the SVD algorithm used can be found on the appendix section 2

3.4.3 Prediction Model

The last step in the generic recommender system is calculating the predicted ratings for the items that the user in question hasn't yet rated, and presenting the better rated ones. Following the method proposed in [9], the $\tilde{R}$ value can be calculated (3.10) by making use of the item similarity matrix. Pseudo-code for this step is presented in the appendix section 3.

$$\tilde{R}(a, i) = \frac{\sum_{j \in I_a} \tilde{S}(j, i) \cdot R_{a, j}}{\sum_{j \in I_a} \tilde{S}(j, i)} \quad (3.10)$$

In this prediction calculation, a weighted sum is applied to the similarity values between the item in question and all other j items, multiplied by the rating that the user in question gave to j ; as such, j must belong to I_a , i.e. the set of items previously rated by the user. Since it is a weighted sum and it lacks ways to treat nonpositive similarity values properly, the approach of thresholding the I_a set to only items with positive similarities to i was taken. Literature [17] suggests only a certain neighbourhood k of j items need be considered when calculating $\tilde{R}$, according to their relation to i ; in this project we consider the full neighbourhood of j items, but, taking it in consideration and after testing, it was decided to limit the prediction algorithm to users with at least 10 rated items, in order to avoid low confidence predictions.

Lastly, the model was tested for its prediction accuracy as detailed in 3.3. Due to dataset size constraints, the model predicted the rating of every item for 25% of all eligible users. The final RMSE reached was 0.996, which is around the benchmark for an acceptable error value when comparing it to other projects using the same base Yelp dataset for creating recommender systems, detailed in Table 3.4. When tested with the similarity matrix built using the COS method, the model achieved a worse RMSE of 1.034.

Table 3.4: Comparison with other projects that used the Yelp Dataset

<i>Method</i>		<i>RMSE</i>
Collaborative Filtering with AMSD & SVD		0.996
Wang et al., 2018 [48]	Collaborative Filtering with Gradient Descent	0.877
Chee Hoon Ha, 2014 [12]	KNN with Restaurant Clustering	1.124
LeXu et al., 2015 [16]	K-Means	0.917

Chapter 4

Explanations

This chapter concludes the detailing of the project, by introducing the main explanation model and how it fits into the generic recommender system built prior. It delineates the main goals for the explanation model and how it will be evaluated, based on what was already presented in [2.5](#). It further presents the different types of explanations contained within the explanation model, how they respond to the model's goals and how they differ in regards to how they are implemented. A general influence calculation model is introduced, for determining which factors had the most influence in a recommendation and, thus, serve as the basis for the explanations. The last section presents applications of the influence model, as well as alternatives that can forego the influence model entirely while still accomplishing other explanation goals.

4.1 Goals & Evaluation Metric

The main function of a recommendation explanation is to provide clarity to the user and to dispel the perceived magic behind the system that generated the recommendation. The explanations given must be meaningful, in the sense that they help the user make sense of why a specific new item has been recommended, and be relevant in regards to the item that's being recommended. As such, the explanation model must be able to determine not only which factor had the most impact on the recommendation, but also the best way to present that information.

Due to the nature of the static test dataset in which the recommendation system is inserted into, it becomes impossible to evaluate the efficacy of the explanations through simple user questionnaires; as an alternative approach, the user questionnaires must be simulated for random users in the database. This process begins by individually studying the profile of some test users, simulating recommendations and explanations for them, and evaluating the explanations following the metrics from the table [2.1](#). An example of a failed explanation would be justifying a recommendation of an item that is a hotel by stating the user tends to like seafood restaurants; an acceptable explanation would be considered if it stated the user tends to rate hotels highly, or even airports.

4.2 Types of Explanations

Regarding the explanation model, it's important to recognise the two distinct phases where it can insert itself into the recommender system, and that for each phase there are multiple ways of generating explanations, depending on the features of the dataset.

The most intuitive phase, from an implementation standpoint, comes after the recommended item has been found, wherein the explanation model attempts to uncover the factors that motivated that recommendation, which we can call Post-Recommendation Explanation. A simple example for this method is finding which item already rated by the user had the most impact in the recommendation calculations. The other alternative phase comes before the prediction algorithm is ran and a recommendation is found; in this scenario, the explanation model first finds a specific preference pattern surrounding the user's profile, and then recommends around that pattern; we can call this method Explanation-Based Recommendation. An example for this method would come in first finding which items the user's friends tend to like the most, finding the most recommendable item for the user in that item pool, and presenting it on the basis that the user's friends enjoy the item.

Due to the nature of the Post-Recommendation Explanation method, it is expected that it would satisfy the user's need for clarity and transparency, as its purpose is to attempt to explain how the system works. On the other hand, as the explanations provided using the Explanation-Based Recommendation method are more tailored to the user, and go hand-in-hand with the generated recommendation, it is expected that the user will be more persuaded in believing in the recommendation.

As both methods for generating explanations vary in how they're applied and in which features of the dataset they can more accurately reach, so do their results vary. When implementing an explanation model, it is important to have it provide a wide variety of different types of explanations, as it will result in more transparency for the end-user; however, it is crucial to have it give preference to the types of explanations that will give the best results for the specific users and datasets. Thus, the explanation model must show flexibility in how it chooses which types of explanations to give and in its ability to adapt with user feedback.

4.3 Post-Recommendation Explanation

At the core of the Post-Recommendation Explanation model is finding out which factors had the most influence in the system's recommendation; once the most impactful factor has been determined, the explanation for the recommendation can be presented with that factor as its base. The way to assert this influence can be simplified, for a chosen factor, as the difference in rating prediction with and without that factor being included in the calculations [4]. This influence score (4.1) is best taken as the percent difference between the real prediction and the factor-less

prediction, as it allows for different types of factors to be compared interchangeably and further compared with influence benchmarks.

$$\text{Influence}(\text{Factor}, \tilde{R}) = \frac{\tilde{R} - \tilde{R}_{\setminus \{\text{Factor}\}}}{\tilde{R}} \quad (4.1)$$

$\tilde{R}_{\setminus \{\text{Factor}\}}$ is the prediction score calculated without the factor

A positive influence score for a factor indicates that it was responsible for the increase of the predicted rating of the item, while a negative influence score means the factor is one of the reasons the system thinks the user might not like the item as much. An influence score closer to zero simply implies that the factor had close to no influence on the recommendation.

4.3.1 Individual Item Influence

Upon analysing the rating prediction equation for $\tilde{R}(a, i)$, it becomes obvious that the simplest factor to determine influence for is the individual items belonging to the list I_a of items rated by the user. In this case, in order to determine the influence that an item x belonging to I_a had in the recommendation, the factor-less prediction is calculated by never considering x as a possible j . This item influence score is not interchangeable with the similarity score between items x and i , as the similarity scores do not take into consideration the user's appreciation of the items.

Finally, after determining which of the user-rated items had the biggest influence score, the explanation can be constructed by presenting the most influential items. However, a flaw in this procedure is that the influence can come from items that the user liked or disliked. In order to present the explanation in the format "Because you liked such and such items..." it becomes necessary to first isolate the items that the user liked before following this method. An item can be defined as being liked by the user if it received a rating above the average user rating score; thus, only the top 50% liked items by the user are considered for the influence calculations. Alternatively, by singling out the bottom 50% of liked items by the user, the explanation could be presented as "Because you disliked such and such items..."; however impactful user dislikes are to the recommendation system, this explanation method should be avoided, as it loses out on the basic functions of explanations by focusing on the negative aspects of the user's experience. The explanation presentation should include the top-k most influential items in the recommendation, as to maximize the explanation's effectiveness. An example of an explanation that was generated while testing this method with the dataset, for a recommendation of a bar establishment, the most influential items were shown to be mostly other bars and nightclubs that the test user had visited and rated favourably. A pseudo-code demonstration for this method is available in the appendix section 4.

4.3.2 Category Influence

Commonly, with recommender systems, the items in the dataset can be assigned tags or categories to distinguish their most basic features. With the dataset provided by Yelp, items have a list of features called Categories, which can range from generic attributes, such as Restaurant and Food, to more specific distinctions, such as Seafood and Asian Fusion, to name a few examples related to the food section of the businesses in the dataset.

Following the logic from the Individual Item Influence method, individual categories can too be considered for their influence in the item recommendation. This method can be achieved by first singling out all the categories belonging to the items liked by the user; then, for an individual category c , calculate the factor-less prediction rating by removing all items from I_a that belong to that category. For the explanation presentation, it can be shown to the user the top-k most influential categories for the recommendation; additionally, the explanation can filter out either the generic or the more specific categories, in order to increase user readability.

4.3.3 Other Influence Factors

While not explored in this project, there remains the possibility of going much further in how creatively factors are determined, depending on the limitations imposed by the dataset and the platform in which the recommended system is inserted. As an example, a factor can be items recently reviewed by the user, where, following the category influence method, items are singled out if they've been reviewed by the user in the past month or year; for this example, if the influence score for recent items is enough, the explanation can be presented as "Because of your recent activity...".

In a collaborative filtering system built around user similarity, the individual item influence method discussed above 4.3.1 could be modified to calculate the individual influence of other users in the recommendation. While somewhat intrusive for this case, since the user is being presented speculations on other users' consumer patterns, this method could be explored for other platforms such as Facebook, where the user might be more interested in following the general public opinion. Regardless, the more general recommender systems based on user similarity would attain better results by following methods similar to the one discussed in 4.3.1, as it better preserves the other users' anonymity and focuses more on their consumer patterns' characteristics.

4.4 Explanation-Based Recommendation

The alternative to the Post-Recommendation Explanation model is completely foregoing the influence calculations and the factors' influence as the base for the explanation; instead, the explanation is built into the recommendation before the recommendation itself is even known. This explanation model is, in contrast to the previous one, more intuitive for the end-user, due to the information generally being presented in simpler terms. As with the other explanation model, this one too has a vast array of applications, dependant on the dataset's limitations and the designer's creativity.

This model works by first identifying a desirable pattern in the user's activity profile, then filtering out items that do not belong to or do not match up with this pattern; afterwards, the recommendation process is continued as normal, following the process shown in the appendix section 3, except the recommendable item pool is limited to only those items filtered in the previous step. The result is a list of the most recommendable items that belong to the desirable pattern.

The pattern detection process is dependent on context, provided by a knowledge base, in order to select the most appropriate pattern or patterns for the context in which the user is situated in the platform; for example, in a platform like Yelp, when the user is viewing an item's page, the most appropriate pattern for this context would be this specific item (this type of pattern is explored in 4.4.1). These context-pattern associations are to be manually determined and stored for each platform that the recommendation system is inserted into.

4.4.1 Item-Based Recommendation

The pattern for this type of explanation is based on an individual item. There are plenty of ways an item can be qualified to be chosen to be the base of an explanation; for this project, it was experimented on items qualifying to be chosen by the following criteria: an item that is liked by the user, and the item that is currently being viewed by the user. Depending on the qualifying aspect of the individual item, the explanation generated and the functions performed by the explanation will be different.

For the first type of item qualifying aspect tested, it was determined that a liked item was one that was rated by the user above their average scores given; once determined the list of items most liked by the user, the qualifying item is chosen at random. For the second method, the item in question is already known prior. The item filtering step is then performed by excluding the bottom 75% of total items from the item pool, based on their similarity to the chosen item. Finally, the recommendation engine is ran and the most recommendable items from the remaining ones in the item pool are presented.

For the first type of qualifying aspect, the explanation is presented the list of items with the subtext of "Because you liked item <chosen item>: ...". For the other qualifying aspect, the list of recommendable items is presented with the subtext of "Items similar to this one: ...". The first type is expected to increase user retention in the platform, as it recommends items specifically based on items the user already enjoyed; the second type is expected to also increase user retention in the platform, as it presents different options to the user while they themselves are browsing the platform.

4.4.2 Friends-Based Recommendation

The pattern for this type of explanation is based on the user's friends' preferences. Considering all items not yet rated by the user, the determining factor for items to pass the filtering stage of this method is whether or not they are liked by the majority of the user's friends. The criteria for a friend liking an item is the same as the one for the user liking an item. In order to avoid outliers,

only items that are rated by at least 3 friends are considered for this stage. As such, an item is considered for the final item pool if, for all the user's friends that have rated it, the majority of them favour it. Afterwards, the recommender engine is ran for the final item pool and the most recommendable items are presented to the user with the subtext of "Your friends seem to like these items: ...".

This type of explanation is presented in a similar manner as the liked item based explanation from 4.4.1, while also introducing the social aspect of the user's friends. This added variety and reach is expected to increase the overall effectiveness of the explanations, as the user is given more diverse options to choose from.

4.4.3 Category-Based Recommendation

Upon tallying the user's most liked items, it is possible to then extract these items' categories and find out the user's most preferred categories; a category is more liked the more liked items it appears in. The pattern for this type of explanation is based on item's categories, more specifically the user's most liked categories. The basic steps for this can be seen on Figure 4.1.

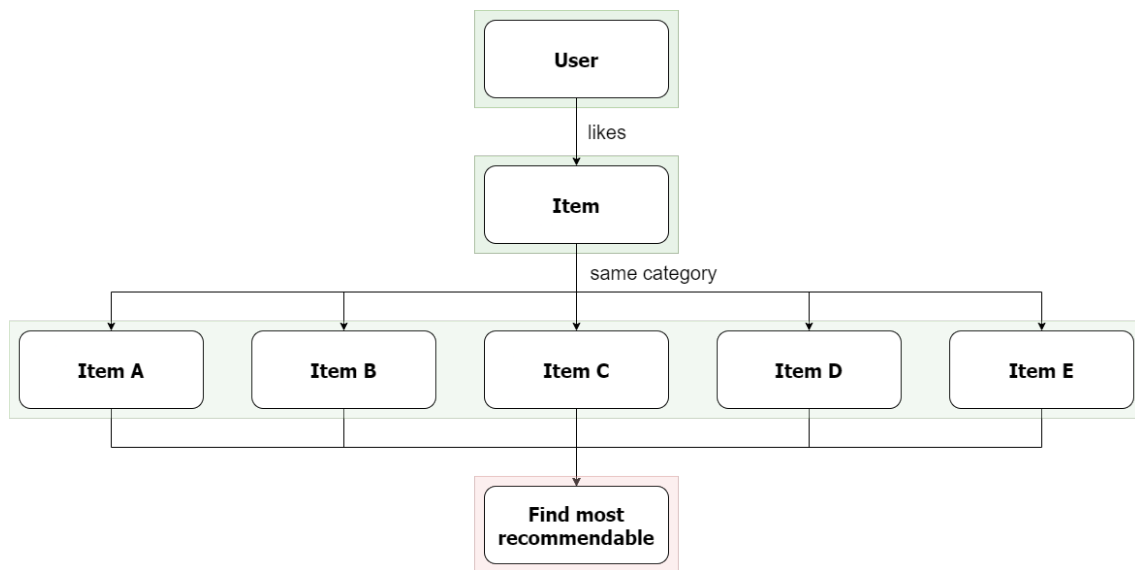


Figure 4.1: Category-Based Recommendation

The category chosen for a given explanation is based on a weighted random pick, from the list of liked categories and the times each category shows up in the liked items list. Then, the filtering stage is done by filtering out, from the total pool of items unrated by the user, those that do not belong to the chosen category. Finally, the recommendation engine is ran on this item pool and the most recommendable items are found and the explanation is presented as "Because you like <chosen category>: ...". This explanation type is more restrictive than the rest, as it forces all recommendable items to belong to a certain category; however, it is expected to result in a more

efficient recommendation, as it's presenting safe options for the user and thus making the decision making process go faster, and satisfies the recommendation system condition of presenting relatively safe recommendations on occasion, as a way to keep the user's trust in the system high.

4.5 Discussion

In summary, this chapter explained the different ways in which an explanation model can be constructed to perform alongside a generic recommendation system, and to satisfy the myriad of standards held up by the end-users. There were exposed two distinct methods of integrating explanations into a RS, their strong points and examples for deriving explanations from within those two methods. As mentioned previously, the testing phase for the explanation model had to forego the use of user questionnaires, due to the somewhat abstract environment in which the recommendation system inserted itself. As such, each explanation method was tested manually for their hypothesised strong points, thus remaining questions regarding their full efficiency in a volatile and live environment. Regardless, as each method, in a way, managed to mimic results obtained in fulfilled and well known platforms, like Youtube and Netflix, the eye test results can be given a margin of trust.

Chapter 5

Conclusions

In this study, it was detailed the creation of a practical model for integrating explanations into a generic recommendation system. Along the creation of the generic recommendation system, it was always prioritised that its overall quality fit against established benchmarks, while retaining the basis of a generic engine and maintaining a degree of simplicity that would allow it to later be replaced with a more complex engine without compromising the work done with the explanation model. When conceiving the explanation model, it was made sure that it was capable of creating explanations that would satisfy all the necessary functions expected from explanations in a recommendation system.

5.1 Future Work

Analysing the work done, the most glaring aspect to be improved on is the application of the explanation model, theorised and built around a generic and static recommendation system, into a live platform, for further verifying its results. Furthermore, it would be interesting to apply the explanation model and its methodologies into more complex recommendation systems built around different types of datasets, as it would allow for the experimentation on other types of influence factors and other methods for conceiving explanations as base for recommendations.

References

- [1] Mohammed Alshammari, Olfa Nasraoui, and Behnoush Abdollahi. A semantically aware explainable recommender system using asymmetric matrix factorization. pages 268–273, 09 2018. DOI:[10.5220/0006937902680273](https://doi.org/10.5220/0006937902680273).
- [2] Linas Baltrunas, Bernd Ludwig, Stefan Peer, and Francesco Ricci. Context-aware places of interest recommendations for mobile users. volume 6769, pages 531–540, 07 2011.
- [3] Linas Baltrunas, Bernd Ludwig, Stefan Peer, and Francesco Ricci. Context-aware places of interest recommendations for mobile users. volume 6769, pages 531–540, 07 2011. DOI:[10.1007/978-3-642-21675-6_61](https://doi.org/10.1007/978-3-642-21675-6_61).
- [4] Mustafa Bilgic and R.J. Mooney. Explaining recommendations: Satisfaction vs. promotion. *Beyond Personalization (workshop at IUI)*, pages 13–18, 01 2005.
- [5] Nick Bostrom. *Superintelligence: paths, dangers, strategies*. 01 2014.
- [6] J. Brown, R. Burton, and Johan Kleer. Pedagogical, natural language and knowledge engineering techniques in sophie i, ii and iii. *Intelligent Tutoring Systems*, 01 1982.
- [7] Evangelia Christakopoulou and George Karypis. Local item-item models for top-n recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys ’16, page 67–74, New York, NY, USA, 2016. Association for Computing Machinery.
- [8] Nick Dyer-Witheford, Atle Kjosén, and James Steinhoff. *Inhuman Power: Artificial Intelligence and the Future of Capitalism*. 06 2019. ISBN 9780745338606.
- [9] Michael Ekstrand, John Riedl, and Joseph Konstan. Collaborative filtering recommender systems. *Foundations and Trends in Human-Computer Interaction*, 4:175–243, 01 2011. DOI:[10.1561/1100000009](https://doi.org/10.1561/1100000009).
- [10] Lawrence Fagan, Edward Shortliffe, and Bruce Buchanan. Computer-based medical decision making: from mycin to vm. 3, 01 1984.
- [11] Alexander Felfernig, Nava Tintarev, Thi Ngoc Trang Tran, and Martin Stettinger. Designing explanations for group recommender systems. 02 2021. [arXiv:2102.12413v1](https://arxiv.org/abs/2102.12413v1).
- [12] Chee Hoon Ha. Yelp Recommendation System Using Advanced Collaborative Filtering. <http://cs229.stanford.edu/proj2014/Chee%20Hoon%20Ha,%20Yelp%20Recommendation%20System%20Using%20Advanced%20Collaborative%20Filtering.pdf>, 2014. Accessed: June 2021.
- [13] Marcel Hildebrandt, Swathi Sunder, Serghei Mogoreanu, Mitchell Joblin, Akhil Mehta, Ingo Thon, and Volker Tresp. A recommender system for complex real-world applications with

- nonlinear dependencies and knowledge graph context. *Lecture Notes In Computer Science*, pages 179–193, 05 2019. DOI:[10.1007/978-3-030-21348-0_12](https://doi.org/10.1007/978-3-030-21348-0_12).
- [14] Sarika Jain, Anjali Grover, Praveen Singh Thakur, and Sourabh Kumar Choudhary. Trends, problems and solutions of recommender system. *International Conference on Computing, Communication & Automation*, 2015. DOI:[10.1109/CCAA.2015.7148534](https://doi.org/10.1109/CCAA.2015.7148534).
- [15] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. Recommender systems - an introduction. 2010.
- [16] Yingran Xu Jia Le Xu. Recommendation System Using Yelp Data. http://cs229.stanford.edu/proj2015/301_report.pdf, 2015. Accessed: June 2021.
- [17] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.
- [18] Pigi Kouki, Shobeir Fakhraei, James Foulds, Magdalini Eirinaki, and Lise Getoor. Hyper: A flexible and extensible probabilistic framework for hybrid recommender systems. 09 2015.
- [19] Pigi Kouki, James Schaffer, Jay Pujara, John O’Donovan, and Lise Getoor. Personalized explanations for hybrid recommender systems. pages 379–390, 03 2019.
- [20] Vaibhav Kumar. Singular value decomposition (svd) & its application in recommender system. <https://analyticsindiamag.com/singular-value-decomposition-svd-application-recommender-system/>, 2020. Accessed: June 2021.
- [21] Sebastian Lapuschkin, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS ONE*, 10:e0130140, 07 2015.
- [22] James Le. Recommendation System Series Part 8: The 14 Properties To Take Into Account When Evaluating Real-World Recommendation Systems. <https://towardsdatascience.com/recsys-series-part-8-the-14-properties-to-take-into-account-when-evaluating-real>, 2015. Accessed: June 2021.
- [23] James Le. Recommendation System Series Part 4: The 7 Variants of Matrix Factorization For Collaborative Filtering. <https://towardsdatascience.com/recsys-series-part-4-the-7-variants-of-matrix-factorization-for-collaborative-f>, 2020. Accessed: June 2021.
- [24] George Lekakos and Petros Caravelas. A hybrid approach for movie recommendation. *Multimedia Tools Appl.*, 36:55–70, 01 2008.
- [25] Ninghao Liu, Yong Ge, Li Li, Xia Hu, Rui Chen, and Soo-Hyun Choi. Explainable recommender systems via resolving learning representations. pages 895–904, 10 2020.
- [26] Jiazhen Lu, Lili Ye, and Jing Dong. Applied singular value decomposition method in transfer alignment and bias calibration. *IET Radar, Sonar & Navigation*, 14, 01 2020.
- [27] Sean McNee, John Riedl, and Joseph Konstan. Being accurate is not enough: How accuracy metrics have hurt recommender systems. pages 1097–1101, 04 2006. DOI:[10.1145/1125451.1125659](https://doi.org/10.1145/1125451.1125659).

- [28] David Mcsherry. Explanation in recommender systems. *Artificial Intelligence Review*, 24:179–197, 10 2005.
- [29] Saaransh Menon. Stratified sampling in Machine Learning. <https://medium.com/analytics-vidhya/stratified-sampling-in-machine-learning-f5112b5b9cfe>, 2020. Accessed: June 2021.
- [30] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267, 06 2017.
- [31] Narjes Nikzad Khasmakhi, Mohammad Balafar, and Mohammad Reza Feizi Derakhshi. The state-of-the-art in expert recommendation systems. *Engineering Applications of Artificial Intelligence*, 82:126–147, 06 2019.
- [32] Youngki Park, Sungchan Park, Sang-goo Lee, and Woosung Jung. Fast collaborative filtering with a k-nearest neighbor graph. In *2014 International Conference on Big Data and Smart Computing (BIGCOMP)*, pages 92–95, 2014.
- [33] P. Jonathon Phillips, Carina Hahn, Peter Fontana, David Broniatowski, and Mark Przybocki. Four principles of explainable artificial intelligence. 08 2020. DOI:[10.6028/NIST.IR.8312-draft](https://doi.org/10.6028/NIST.IR.8312-draft).
- [34] Parivash Pirasteh, Dosam Hwang, and Jason Jung. Exploiting matrix factorization to asymmetric user similarities in recommendation systems. *Knowledge-Based Systems*, 83, 07 2015.
- [35] Logesh R., Subramaniaswamy V, Vijayakumar Varadarajan, Siguang Chen, A. Karmel, and Malathi Devarajan. Hybrid location-based recommender system for mobility and travel planning. *Mobile Networks and Applications*, 04 2019. DOI:[10.1007/s11036-019-01260-4](https://doi.org/10.1007/s11036-019-01260-4).
- [36] Sanjog Ray and Rajesh Sharma. Explanations in recommender systems: an overview. *International Journal of Business Information Systems*, 23:248, 01 2016.
- [37] Marco Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. pages 1135–1144, 08 2016.
- [38] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. *Recommender Systems Handbook*. Springer, 2010. ISBN 978-0-387-85819-7.
- [39] Francesco Ricci, Giovanni Semeraro, Marco de Gemmis, and Pasquale Lops. Decision making and recommendation acceptance issues in recommender systems. In *Advances in User Modeling*, volume 7138, pages 86–91, 07 2011. DOI:[10.1007/978-3-642-28509-7_9](https://doi.org/10.1007/978-3-642-28509-7_9).
- [40] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. *Proceedings of ACM World Wide Web Conference*, 1, 08 2001.
- [41] Leslie Smith. Cyclical learning rates for training neural networks. In *IEEE Winter Conference on Applications of Computer Vision*, pages 464–472, 03 2017.
- [42] Guocong Song. Tensorflow-based Recommendation Systems. <https://github.com/songgc/TF-recomm>, 2017. Accessed: June 2021.

- [43] Nava Tintarev. Explanations of recommendations. *ACM Conference on Recommender Systems*, pages 203–206, 01 2007. DOI:[10.1145/1297231.1297275](https://doi.org/10.1145/1297231.1297275).
- [44] Nava Tintarev and Judith Masthoff. Effective explanations of recommendations: user-centered design. In *ACM Conference on Recommender Systems*, pages 153–156, 01 2007.
- [45] Nava Tintarev and Judith Masthoff. A survey of explanations in recommender systems. *2007 IEEE 23rd International Conference on Data Engineering Workshop*, 2007. DOI:[10.1109/ICDEW.2007.4401070](https://doi.org/10.1109/ICDEW.2007.4401070).
- [46] Ankur Tomar. ITEM-ITEM Collaborative Filtering Recommender System in Python. <https://medium.com/@tomar.ankur287/item-item-collaborative-filtering-recommender-system-in-python-cf3c945fae1e>, 2017. Accessed: June 2021.
- [47] Manolis Vozalis and Konstantinos G. Margaritis. Applying svd on generalized item-based filtering. *International Journal of Computer Science Applications*, 3, 01 2006.
- [48] Yaxin Wang, Minmin Zhu, Jiaxi Wu, Zonghao Li, and Tianze Yue. YELP Recommendation System Based on Collaborative Filtering. <https://github.com/zonghao1/YELP-Recommendation-System-Based-on-Collaborative-Filtering>, 2018. Accessed: June 2021.
- [49] Yelp. Dataset Challenge. <https://www.yelp.com/dataset>, 2014. Accessed: June 2021.
- [50] Markus Zanker and Daniel Ninaus. Knowledgeable explanations for recommender systems. volume 1, pages 657–660, 2010. DOI:[10.1109/WI-IAT.2010.131](https://doi.org/10.1109/WI-IAT.2010.131).

Appendix A

Pseudo Code

This appendix section contains samples of the project's code presented in pseudo-code format. The full project's code is available at <https://github.com/dfreddy/Recommender-System>

Algorithm 1 AMSD

```
1: Items  $\leftarrow$  list of all items in the dataset
2: SimilarityMatrix  $\leftarrow$  Items x Items empty matrix
3: L  $\leftarrow$  16
4: for all a  $\in$  Items do
5:   Ratings_a  $\leftarrow$  list of all user ratings for item a
6:   for all b  $\in$  Items do
7:     SKIP: a is b or SimilarityMatrix[a][b] already exists
8:     Ratings_b  $\leftarrow$  list of all user ratings for item b
9:     MutualUsers  $\leftarrow$  list of users that rated both items a and b
10:    UsersRatings  $\leftarrow$  matrix of users and their item ratings
11:    MSD  $\leftarrow$  0
12:    for all u  $\in$  MutualUsers do
13:      MSD  $\leftarrow$  MSD + (UsersRatings[u][a] - UsersRatings[u][b])2
14:    end for
15:    MSD  $\leftarrow$   $\frac{MSD}{size(MutualUsers)}$ 
16:
17:    SimilarityMatrix[a][b]  $\leftarrow$ 

$$\frac{L - MSD}{L} * \frac{size(MutualUsers)}{size(Ratings_a)} * \frac{2 * size(MutualUsers)}{size(Ratings_a) + size(Ratings_b)}$$

18:
19:    SimilarityMatrix[b][a]  $\leftarrow$ 

$$\frac{L - MSD}{L} * \frac{size(MutualUsers)}{size(Ratings_b)} * \frac{2 * size(MutualUsers)}{size(Ratings_b) + size(Ratings_a)}$$

18:   end for
19: end for
```

Algorithm 2 SVD

```

1:  $MaxEpochs \leftarrow$  max number of epochs (100)
2:  $Data \leftarrow$  item similarity matrix
3: for  $EpochCounter=0$  to  $MaxEpochs$  do
4:    $TrainingData \leftarrow$  random 80% of the item similarity matrix
5:    $TestingData \leftarrow$  remaining 20% of the item similarity matrix
6:    $TrainingError \leftarrow 0$ 
7:   for all  $Batch \in TrainingData$  do
8:      $Prediction \leftarrow Inference(Batch)$       {generates the inference matrix for the batch by
      minimizing the cost equation}
9:      $TrainingError \leftarrow TrainingError + (Prediction - Batch.values)^2$ 
10:  end for
11:   $TrainingError \leftarrow \sqrt{\frac{TrainingError}{length(TrainingData)}}$       {RMSE}
12:   $TestPrediction \leftarrow Inference(TestingData)$ 
13:   $TestingError \leftarrow \sqrt{\frac{(TestPrediction - TestingData.values)^2}{length(TestingData)}}$       {RMSE}
14:  update learning rate according to  $EpochCounter$ 
15: end for
16: return  $Inference(Data)$ 

```

Algorithm 3 Prediction

```

1:  $ItemsList \leftarrow$  list of all items in the dataset
2:  $UserRatedItems \leftarrow$  list of all items already rated by the user
3:  $PredictedScores \leftarrow$  list of the predicted score for all items
4: for all  $item \in ItemsList \setminus \{UserRatedItems\}$  do
5:    $a \leftarrow 0$ , weighted sum top
6:    $b \leftarrow 0$ , weighted sum bottom
7:   for  $j \in UserRatedItems$  do
8:      $a \leftarrow a + Similarity(item, j) * UserRating(j)$ 
9:      $b \leftarrow b + Similarity(item, j)$ 
10:  end for
11:  if  $b == 0$  then
12:     $PredictedScores[item] \leftarrow 1$ 
13:  else
14:     $PredictedScores[item] \leftarrow \frac{a}{b}$ 
15:  end if
16: end for
17: return top-k values from  $PredictedScores$ 

```

Algorithm 4 Item Influence

```

1: UserLikedItems  $\leftarrow$  list of items liked by the user
2: InfluenceList  $\leftarrow$  list of items and the influence they had on the recommendation
3: for all  $i \in \textit{UserLikedItems}$  do
4:   RemainingLikedItems  $\leftarrow \textit{UserLikedItems} \setminus \{i\}$ 
5:   InfluenceList[ $i$ ]  $\leftarrow \textit{Influence}(\textit{RemainingLikedItems})$ 
6: end for
7: return top-k values from InfluenceList
8:
9: Influence():
10: Item  $\leftarrow$  the item that was recommended
11:  $a \leftarrow 0$ , weighted sum top
12:  $b \leftarrow 0$ , weighted sum bottom
13: for  $j \in \textit{ItemsList}$  do
14:    $a \leftarrow a + \textit{Similarity}(\textit{item}, j) * \textit{UserRating}(j)$ 
15:    $b \leftarrow b + \textit{Similarity}(\textit{item}, j)$ 
16: end for
17: if  $b == 0$  then
18:   PredictedScore  $\leftarrow 1$ 
19: else
20:   PredictedScore  $\leftarrow \frac{a}{b}$ 
21: end if
22: return  $\frac{\textit{OriginalScore} - \textit{PredictedScore}}{\textit{OriginalScore}}$ 

```
