Alleviating the User Cold-Start Problem in Recommendation Systems using Social Network Information

Ciarán Miceal Johnson

Master of Informatics

School of Informatics
University of Edinburgh

2020

Abstract

Recommendation systems have been used as a successful solution to the information overload problem faced by many e-commerce and music streaming platforms. However, there are several issues that surround the ability of recommendation systems to effectively alleviate the information overload problem. In this dissertation, I will address the user cold-start problem in a data sparse scenario while ensuring that the recommendation system is attack-resistant. I address these problems using social information gained from the user upon signing into the platform with a social network account. The concepts of homophily and social influence were applied to address this problem. I find homophily to be the most beneficial when tackling these problems, and offer a recommendation system, the FriendRec system, based on this concept.

Acknowledgements

I would like to acknowledge my parents for supporting me as well as my supervisor, Professor Stephen Gilmore, who read countless drafts and was always willing to offer his time and feedback.

Table of Contents

Chapter 1

Introduction: The Problem with Recommendation Systems

Recommendation systems have been used as a successful solution to the information overload problem faced on many e-commerce and music streaming platforms [57, 58]. For example the music streaming platform Spotify has over 50 million songs and 271 million users [11], making it difficult for users to search and discover interesting music.

1.1 Flaws in Recommendation Systems

There are several problems that surround the ability of recommendation systems to filter and correctly select items to recommend to users. Four of these problems will be briefly discussed here, while attempts will be made to find solutions for only three; these are the sparsity and cold-start problems, attacks from malicious users and user privacy, trust and ethics [34].

1.1.1 Sparsity

The sparsity problem is caused by an insufficient amount of user feedback data. This significantly increases the difficulty in distinguishing users with similar interest which can heavily affect popular recommendation systems such as the collaborative filtering systems [20]. The sparsity rating of recommendation systems can be calculated as the ratio of observed to total ratings [39] with the sparsity problem occurring around 99% sparsity [27].

1.1.2 Cold-start

The cold-start problem is a version of the sparsity problem which occurs when a lack of preference information negatively affects the recommendations produced by a system. The cold-start problem takes place when a new item, user or even system is created

for the first time. The user cold-start problem is the focus of my research, this problem arises when a new user is added to the system and due to the lack of meaningful feedback provided by the user the system is unable to generate a set of valid recommendations. In general the cold-start phase is considered to be when the user has rated less than five items [27]. However this number may vary between systems as the type of feedback provided by the user also affects the required quantity.

1.1.3 Attack resistant

Recommendation systems have the effect of increasing the sales of a company upon receiving good ratings [21, 37] and therefore they have become the target of attacks by malicious users. Such attacks have not only been directed at the average inexperienced company but also towards giants in the e-commerce industry such as Amazon and Ebay. This paper will focus on shilling attacks which is when malicious profiles are inserted into the recommendation system to either push or nuke ratings for particular items.

1.1.4 Privacy, Trust and Ethics

A fourth concern that has been brought to the attention of the public in recent years is that of user privacy, trust and ethics in recommendation systems. Recommendation systems have the ability to shape an individual's experience of digital platforms and their social interactions [18, 46]. Therefore the ethical implications of these recommendation systems are drastic. A notable mention of when recommendation systems were used to manipulate users is the Cambridge Analytica scandal. Cambridge Analytica used Facebook's third party API to gather significant data on Facebook users and their friends [17] in order to sway their political preferences in the 2016 US presidential elections [46]. This study will however not be addressing the privacy, trust and ethical implications of the implemented recommendation systems.

Generating personalised recommendations for new users is an important tool in consumer retention [60, 37] by securing consumer loyalty [20] as valid recommendations improve user satisfaction and engagement. The validity and integrity of these recommendations must be kept intact as shilling attacks may cost the user time and money by recommending bad items [37].

1.2 State-of-the-Art

The state-of-the-art is disputed in the field of recommendation systems, with varying datasets and multiple foci. However a few recommendation systems stood out when researching the issues I discussed earlier.

DropoutNet [68] looks at the use of Deep Neural Networks (DNN) to learn user preference by treating the problem as the missing data problem. By applying dropout to the units in the network, each unit in a specified layer is removed with the probability p. They were able to generalise the DNN to predict the missing input. This DNN-

based latent model performed on par with what they considered to be state-of-the-art for warm-start users and outperformed state-of-the-art in cold-start situations.

Merge2 [28] uses trusted users who are similar to the cold-start user to predict ratings for unseen items. This synthetic rating data is then used to find other similar users through collaborative filtering, finally the system uses both the trusted and similar users to generate recommendations. This can be also used for warm-users to alleviate the effects of data sparsity.

VarSelect SVD [45] implements a Singular Value Decomposition (SVD)-based collaborative filtering model as well as a SVD-based shilling detection algorithm to remove the negative impact of shilling attacks on their recommendation system. The proposed system is broken into two phases: a detection and removal phase and a recommendation model building phase. The detection phase uses dimensionality reduction to remove malicious users. Shilling profiles tend to be highly correlated, and since highly correlated variables add little discriminative information they will be eliminated during dimensionality reduction.

1.2.1 Suitability of State-of-the-Art

Latent models perform well when an abundance of preference information is available but start to degrade in highly sparse settings [68]. Therefore Deep Learning approaches did not appear to be an appropriate method for handling this scenario.

Most cold-start systems that also handle data sparsity use trust values in social networks to tackle these problems effectively. Trust values are either implicit or explicit, with explicit trust values providing exceptional results in both accuracy and coverage. Measuring implicit trust, on the other hand, has proven to be a difficult task, in some cases impossible, with limited gains [16] and therefore implicit trust was not used in this dissertation.

The majority of recommendation systems that cater to new users in sparse data settings are vulnerable to shilling attacks due to their implementation of collaborative filtering [45]. In order to ensure these systems are protected from such attacks, a malicious user detection phase must be incorporated before calculating trust and then eventually generating recommendations. Each sub-process is important in tackling the various problems faced when generating recommendation lists for new users in a sparse data setting. However each sub-process is time and computationally expensive.

1.3 Motivation and Research Approach

The focus of my research will be to find a relatively accurate, attack-resistant recommendation system to cater to new users upon their first time using the system. Consequently, recommendations should be generated in a time efficient manner to ensure the user is provided with accurate recommendations immediately and can therefore begin to use the system.

The aim of my research is to find the most accurate recommendation system possible

in a data sparse cold-start situation. After finding the optimal systems I will perform a shilling attack to determine the stability and robustness of these recommendation systems. Finally I will determine the average time required to generate recommendations in these systems.

My hypothesis is that social information is the key to providing accurate recommendations during a user's cold-start. Friendship information may also be the key to protecting recommendation systems from attacks. This assumption is founded on the fact that malicious profiles inserted to attack the recommendation system will not appear in this user's social network prior to using the system.

My contribution is a fast attack-resistant recommendation system that provides users with valid recommendations upon first creating an account.

1.4 Outline

This paper has the following structure Chapter 2, Background Research, introduces the topics necessary to understand the rest of this paper as well as the existing research in this field. Chapter 3, Analysing Recommendation Systems, discusses the various metrics used to compare recommendation systems. Chapter 4, Design, will give a brief overview of the recommendation systems implemented during the study with Chapter 5, Implementation, going into a greater detail on how these systems were implemented. Finally Chapter 6, Experimental Results, will lay out the setup and experiments that took place, Chapter 7, Evaluation, will discuss the results achieved in Chapter 6 and Chapter 8, Conclusion, will review the project as a whole and propose further areas of research.

Chapter 2

Background Research

This section provides a basic introduction into the field of recommendation systems, with the intent to provide enough information to understand the following chapters. Subsequently this section will begin with a very broad overview of the field as a whole and will later narrow the focus delving deeper into aspects necessary to understand the work in this dissertation. Finally I will review the dataset used to test my hypothesis and the related literature surrounding the topics of user cold-start, data sparsity and the use of social network information in recommendation systems. This section will cover:

- Types of Recommendation Systems
- Types of User Feedback
- Social Information and Recommendation Systems
- Shilling Attack
- Dataset
- Related Work

2.1 Recommendation Systems

There are two broad categories in collaborative recommendation systems: Content-based and Collaborative Filtering (CF) [35, 52, 41]. Content-based systems select items based on the correlation between the content of items and the users' preferences. The other is CF which is considered to be the most successful [35] and therefore popular recommendation system [38]. Collaborative recommendation systems are in fact so successful that they are implemented by companies such as Amazon.com and LinkedIn [26, 69, 40].

2.2 Content-based Recommendation Systems

The first of the two approaches is the content-based recommendation system. This approach recommends users items similar to that which they liked in the past [55]. Content-based systems owe their success to the creation of user and item profiles to generate recommendations [16, 41]. These item profiles are manufactured by providing feature descriptions which are used to annotate the items. The recommendation system is therefore only as good as the feature descriptions provided for these items [60, 41]. In order to provide interesting recommendations the system must be able to differentiate between items, requiring feature descriptions to be discriminatory. The item profiles of highly rated items are then used to form user profiles in order to gain an insight into the user's preferences [41].

Content-based models scale very well when applied to systems with a large number of users as they do not require any information about other users in the system [1]. Since content-based systems only use the target users profile to make recommendations, the model is capable of capturing the niche interests of each user [1] and will recommend items that would not appeal to the user population as a whole.

The content-based approach, however, suffers from three major drawbacks. Firstly, the model is only able to recommend items that fit the user's profile, it is unable to make novel recommendations [41]. Secondly, as stated previously, the recommendation system is only as good as the item profiles. Poorly annotated items may lead to poor recommendations [60, 41]. Finally, when introducing a new user the cold-start problem appears as the system does not have enough information on the new user to create a profile and accurately recommend new items [13].

2.3 Collaborative Filtering Recommendation Systems

The second approach is known as Collaborative Filtering (CF). CF is based on the idea of crowd-sourcing recommendations to people with similar interests. This can also be broken down into:

- Memory-based
- Model-based

I will briefly look at each approach to gain an understanding of the different ways in which CF recommendation systems can be implemented.

2.3.1 Memory-based Collaborative Filtering

Memory-based CF systems are motivated by the understanding that compatible users trust each others' recommendations [70]. Memory-based systems can yet again be broken into two categories these are: User-based and Item-based.

User-based systems recommend items by identifying the users in the system with the highest similarity score to the target user. The top K users are then used to generate

recommendations. By multiplying the similarity score of each user with their item ratings, predicted ratings can be calculated for items the target user has yet to rate. These scores are then used to create recommendation lists. User-based recommendation systems can be characterised using the following formula:

a = an active user
i = item which is not yet rated by a
$p_{a,i}$ = a's predicted rating for i
$r_{u,i}$ = is the rating of user u for item i
$\bar{r}_a$ and $\bar{r}_u$ = the mean ratings of users a and u
$w_{a,u}$ = the similarity weight between users a and u

$$p_{a,i} = \bar{r}_a + \frac{\sum_{u=1}^{N}(r_{u,i} - \bar{r}_u)w_{a,u}}{\sum_{u=1}^{N} w_{a,u}} \tag{2.1}$$

The computational cost of user-based systems increases with the number of users in the system. Since recommendation systems are required to produce high quality recommendations for a large number of users in very short periods of time, this poses a scalability problem [50]. Current systems are able to search tens of thousands of potential neighbours in real time, however, with the increasing popularity of these e-commerce companies they are now required to search tens of millions of potential neighbours [56]. A solution to this problem is the item-based system.

Item-based systems work in a similar way to the user-based recommendation system, however, instead of finding the similarity between users we find the similarity between items. The rating attributed to a test item is calculated by multiplying the user's rating for each item by the similarity of that item and the test item. Item-based recommendation systems can be characterised using the following formula:

$\bar{r}_i$ = mean rating of item i
$w_{i,k}$ = similarity weight between items i and k

$$p_{a,i} = \bar{r}_i + \frac{\sum_{k=1}^{M}(r_{a,k} - \bar{r}_k)w_{i,k}}{\sum_{k=1}^{M} w_{i,k}} \tag{2.2}$$

The item neighbourhood is fairly static, which means that we can pre-compute the similarity matrix offline. There will therefore be very little online computation, resulting in a very high online performance [56]. However this approach will not be used in this dissertation as it requires user rating data in order to find similar items. This rating data is not available in the user cold-start scenario.

2.3.2 Model-based Collaborative Filtering

Model-based CF builds machine learning models to recognise complex patterns in data. These models are then used to predict ratings for unseen items. Well-known model-based CF techniques include Bayesian belief networks, clustering models, and latent semantic models [63]. Classification algorithms are used if the user ratings are categorical. Regression models and SVD methods are used if the ratings are numerical

[64]. One benefit of using model-based approaches is that they address the scalability problem of increasing users in the system [70]. In conjunction with an improved prediction performance, the model-based approach becomes an attractive option. These models perform well when an abundance of preference information is available but start to degrade in highly sparse settings [68] and therefore were not considered in this dissertation.

2.4 Types of User Feedback

In order to provide users with accurate personalised recommendations, feedback is required from users regarding items. There are two common forms of feedback used in recommendation systems, these are: implicit and explicit feedback [39].

2.4.1 Implicit Feedback

Implicit feedback is the inferred rating of an item based on the user's interactions with the system. Some examples of implicit feedback are selection, duration, repetition, saving and forwarding of items [49]. This list is not exhaustive and implicit feedback is open to interpretation with no designated rating attached to each action. The main benefit of using implicit feedback is it reduces the cognitive load on the user as they do not need to physically rate each item with which they interact [49]. As a result we can use this more abundant feedback to gain information on a wider range of items [30].

2.4.2 Explicit Feedback

Explicit feedback is when the user provides feedback to the system with information about an item. This is often in the form of a rating. Explicit feedback is the most common type of feedback in the research field of recommendation systems [30]. Explicit feedback can be provided via a plethora of methods such as like/dislike, ratings and text comments to name a few. The benefit of using explicit feedback is that analysing this information is relatively simple, providing more reliable and accurate recommendations [32]. However since explicit feedback is often sparse [33], using only this form of data would likely result in the data sparsity and cold-start problem.

2.5 Social Networks and Recommendation Systems

Social network information can be manipulated to improve recommendation systems [29]. These networks consist of nodes (users) and edges (links) between these nodes which represent relationships between users. Social correlation theories such as homophily and social influence have shown to be good indicators of user preference [66].

Homophily is the assumption that people form strong positive social ties with people of similar interests [44]. Therefore popular items among close friends should be of interest to the user. This in turn will hypothetically result in valid recommendations for any user even in the absence of feedback for that user in the system.

Social influence reveals that users who are connected are more likely to have similar preferences [66]. In this dissertation Random Walk with Restart was implemented to find the largest influencers on a given user in a social network.

2.5.1 Random Walk with Restart

Random Walk with Restart is an algorithm which provides a relevance score between two nodes and can be used to capture the structure of a given graph [67]. The Random Walk with Restart algorithm is similar to the basic Random Walk algorithm which begins with a starting node and at each step may move to an adjacent node in the social network. The probability of moving to each adjacent node in the network is decided by the weight applied to the links between nodes. Random Walk with Restart is essentially the same however it introduces a probability at each step of returning to the starting node.

The long term probability associated with being at each node can be obtained by recursively applying the algorithm until convergence [35]:

x = starting node
a = probability at each stage to restart at x
$\mathbf{p}^{(t)}$ = a column vector of probability at stage t
$p_i^{(t)}$ = the probability that random walk at stage t is at node i
q = column vector with the element corresponding to the starting node set to 1
S = column normalised adjacency matrix of the graph
$S_{i,j}$ = probability of j being the next state given that the current state is i

$$\mathbf{p}^{(t+1)} = (1-a)\mathbf{S}\mathbf{p}^{(t)} + a\mathbf{q} \tag{2.3}$$

These stationary probabilities are the expected frequency of return to each node given a starting node. A larger probability of restart forces the algorithm to return to the starting node more frequently, resulting in higher probabilities for nodes nearer to the starting node.

2.6 Shilling Attacks

Due to the economic benefits of recommendation systems [23] they have become the target of attacks. In order to influence the frequency in which users are recommended certain products, malicious profiles known as shills are inserted into the recommendation system. These malicious profiles collude and insert ratings on items in order to alter the recommendations produced for certain users. Shilling attacks can be split into two categories: push and nuke attacks [45]. Push attacks work by rating a particular item highly and therefore pushing the item on other users. Nuke attacks rate a certain item poorly reducing the chances of the item appearing in other users' recommendation lists.

This dissertation will carry out a segment attack, a form of push attack, on the recommendation systems. A segment attack works by focusing on a small demographic of users and inserting malicious profiles to match this segment's interests [47]. The malicious profiles will then all rate a certain item highly, increasing the likelihood of that item being pushed on all users in the segment. An example of a shilling attack was in June 2001 when Sony Pictures admitted to creating fictitious movie critics to promote their new released films [37].

2.7 Dataset

To test the hypothesis put forward in the previous section, I will be using the Last.Fm dataset which was released for the 2nd International Workshop on Information Heterogeneity and Fusion in Recommendation Systems (HetRec 2011) [12]. This dataset contains information about the users of the music social network, Last.Fm. Last.Fm develops a user's profile based on their music taste by recording the tracks the user listens to across multiple platforms. On the website users are also able to follow friends in order to create a social network. This dataset contains the following information.

Stats:

- Number of Users: 1,892
- Number of Artists: 17,632
- Bi-directional user friend relationships: 12,717
- Artist-User connections: 92,834
- Number of Tags: 11,946
- Tag Assignments: 186,479
- Sparsity rating: 99.997%

This dataset contains the top 50 most listened to artists for each user, which resulted in an average of 49.1 artists per user. While this shows that not all users have listened to the maximum 50 artists, it is nevertheless a sufficient sample size to conduct valid experiments. The use of social tagging allows users to describe artists using the words they feel are most representative. The benefit of this approach is that I can begin to understand how the user views certain items as well as where they believe their interests lie. With an average of 98.6 tag assignments per user and 14.9 tag assignments per artist, social tagging is clearly a benefit in the Last.Fm dataset, which I will utilize in my recommendation system. There are on average 13.4 friend relationships per user, these connections are bi-directional meaning that friendships are mutual. Finally since the dataset has a data sparsity rating above 99% this dataset is sparse [27] and can therefore be used when attempting to tackle this problem.

2.8 Related Work

The use of social information to overcome the cold-start problem has been extensively studied in recommendation systems. With varying agendas and the diverse feedback available to recommendation systems, an assortment of approaches have been proposed to handle this problem.

Sedhain et al.[60] uses an item-based social CF recommendation system to great effect when tackling the cold-start problem. The paper assesses the use of Facebook page likes, demographic information and Facebook friends when recommending items. They concluded that cosine similarity in conjunction with Facebook page like information resulted in the most accurate recommendations. The Facebook page like recommendation system showed a six-fold improvement in precision and recall when compared with the popular item, demographic and Facebook friend recommendation systems for cold-start users.

Prando et al.[53] was the first attempt at overcoming the cold-start problem by generating recommendations using only social network data to match products from a real e-commerce website using a content-based approach. By combining three social media elements gathered from Facebook and Twitter; direct users posts, content likes and page likes, the recommendation system was able to recommend relevant categories taken from the e-commerce website BestBuy. The paper proposes using Natural Language Processing techniques, tf-idf, to prepare data for the Naive Bayes Classifier. This classifier is then used to rank the categories in order of appeal to the new user. This approach achieved satisfactory results when tackling the cold-start problem.

Trust-aware Recommendation Systems (TARS) have also been a successful approach in alleviating the cold-start user problem in highly sparse data settings. Massa and Avesani [43] introduce a TARS designed to handle these problems called MoleTrust, due to the propagation of trust over the social network they are able to compute trust weights in more users than using CF on trust matrices. This alleviates the data sparsity problem by introducing the concept of transient trust. Through this system users are able to benefit from recommendations from all users in this trust network, not only the subset of users they directly trust. Recommendations can be computed even after only a small number of user ratings.

Another approach discussed in the introduction, which builds on Massa and Avensai's work, is Guo, Zhang and Thalmann [28] who proposed Merge2 in order to handle the data sparsity and user cold-start problem. This approach predicts the ratings of cold-start users by finding the ratings of all users that are trusted and similar to the target user. These approximated ratings are then used to find similar users using CF. Finally, all trusted users and similar users found during CF are employed to generate recommendations for the cold-start user. This approach can also reduce the effects of data sparsity for warm-start users.

Social information has not only been used to improve recommendations in user cold-start scenarios, but has also been known to provide valid improvements for warm-start users. An example is Sun et al.[65] which uses a social regularisation approach to recommend items to users. It looks at bi-clustering the friends of a given user into

two categories, relevant and irrelevant, in order to improve recommendations. This is based on the fact that certain friends may have different interests than the target user. For example, parents are often friends with their children on social media yet their taste in music may be irrelevant when it comes to providing them with music recommendations.

As shown above, a number of authors have addressed the topic of social information and recommendation systems in a systematic manner. In particular, Konstas, Starhopolis and Jose [35] identified a range of benefits to be derived from embedding social information into recommendation systems. Their use of the Random Walk with Restart algorithm along with their method for combining social information into a CF model provides the foundations for a lot of the work described in the remainder of this report.

Chapter 3

Analysing Recommendation Systems

In this dissertation I hypothesise that social network information is the key to improving recommendations during a user's cold-start period. It is, therefore, important to determine means of measuring improvements over the various systems. The following section distinguishes between online and offline metrics.

3.1 Online Analysis

Online analysis is the most effective method of judging the success of a recommendation system [61, 42]. This type of analysis entails directly observing a user's interaction with the system while online. A common form of online analysis is called A/B testing, where users are split into two groups with each group implementing a different recommendation system. A/B testing is an effective way to evaluate and compare recommendation systems and can be used to measure the direct impact of the system on the end user [13]. However this requires a large scale system with many users to gain meaningful real-time information about recommendations from users [25, 42]. In this method of analysis, the quality of recommendations is typically judged using the click-through rate [15], which is the number of times a recommendation is selected.

3.2 Offline Analysis

Offline analysis is one of the most popular forms of analysis for recommendation systems[13]. Offline analysis uses historic datasets to judge the performance of recommendation systems. Due to the lack of user feedback on recommendations, assessing the validity through Offline analysis proves difficult. Nevertheless, there are several methods that have been developed to assess the performance of recommendations using historic datasets which are discussed below.

3.2.1 Accuracy

Accuracy has for many years been the of metric of choice when performing offline analysis on recommendation systems [24]. There are several approaches to finding

the accuracy of a recommendation system, for example the (root) mean squared error can be used to compare the accuracy of the predicted ratings of items with their actual ratings. In this dissertation, however, I will be using Precision@K in order to determine the accuracy of the recommendations generated. Precision@K uses the number of relevant items in the top K recommended items as the accuracy score for the recommendation system [62]. Nevertheless accuracy is not always the best metric when assessing recommendation systems as it can provide an incomplete picture [13]. Characteristics of a good recommendation system are subjective and therefore in order to compare different recommendation systems it is often advised to use some secondary metrics. Common secondary metrics are:

3.2.2 Coverage

Coverage is the ability, or lack thereof, to recommend a certain item to any user or a specific number of items to a certain portion of users [24]. Insufficient coverage in recommendation systems can often be attributed to the sparsity of the ratings matrix [13]. There are two types of coverage: item-space coverage and user-space coverage.

User-space coverage is the percentage of users that can receive at least n item recommendations from the system. This occurs through a lack of user ratings, which can result in the inability to find similar users due to a limited number of ratings on mutual items. User-space coverage is calculated by finding the fraction of users without a full list of recommendations.

Item-space coverage is the percentage of items for which we are unable to predict the rating for at least n users. I will not be considering this metric as I am generating recommendation lists rather than predicting the users' ratings for an item.

3.2.3 Confidence and Trust

Confidence is the system's faith in the recommendations it provides to the user. Recommendation systems that can accurately recommend items at a smaller confidence interval tend to be desirable as they bolster the user's trust in the system [13].
Trust, on the other hand, is the user's faith in the recommendations provided by the system. The most simple method of gauging trust in a system is by conducting a survey that allows users to directly express their level of trust in the system [13]. The utility of the recommendation should also be taken into account as recommending an already purchased item may boost the user's trust in the system, but would ultimately be useless. Additionally, with a lack of participants, it is impossible to measure the users' trust in the system.

3.2.4 Novelty

Novelty is the concept of providing the user with recommendations of which they are unaware or for which they have not yet assigned feedback. This can also be viewed as a method of self discovery by making the user aware of their likes and dislikes. The simplest way of calculating the novelty of a system is through an online analysis

approach where users are asked if they were aware of the item before the recommendation. However novelty can also be calculated using offline datasets provided that the rating data is timestamped. This method takes the dataset at a certain point in time and uses the past recommendations to build the recommendation system. Since novelty is essentially predicting what items the user will like in the future, novelty can be calculated by taking the remaining (future) rating data and counting the number of future data points recommended by the system at this timestamp. Since our dataset only provides timestamps for tags this metric will not be used to measure improvements in the recommendation systems.

3.2.5 Serendipity

Serendipity is the occurrence of events by chance which result in a beneficial outcome [2], therefore in the context of recommendation systems, we can view this as stumbling across an unexpected item which in turn is highly rated by the user. This metric is different from novelty as novelty only requires the user to be simply unaware of the item recommended whereas serendipity requires the recommendation to be surprising. Serendipity can be calculated by creating a basic recommendation system that will be used to predict obvious recommendations. The number of valid recommendations produced from the non-serendipitous recommendation system which also do not appear in the recommendations produced by the serendipitous recommendation system is the serendipity score. Increasing the serendipity in a system can result in important long term improvements however it may negatively impact the accuracy of the system [36].

3.2.6 Diversity

Recommending a variety of different genres increases the likelihood that the user will find an item in the recommendation list that appeals to them; the difference between the items in the recommendation list is known as diversity [31]. The negative effect of recommending all the same genre is that if the user dislikes one product they are likely to dislike all the items recommended [13]. The diversity score can be calculated by finding the mean of the pairwise similarities of all the items in the recommendation list [19].

3.2.7 Robustness and Stability

Robustness and stability ensures that the system is not susceptible to attacks and will not be radically affected by large changes to the dataset [13]. An example of a common attack to a recommendation system is when malicious users generate fake ratings for certain items. The recommendation system must be able to prevent this from significantly affecting genuine users' recommendation lists. Several measurements have been proposed to calculate the stability and robustness of a recommendation system against various attacks. In this dissertation a segment attack will be applied to all recommendation systems in order to test their robustness and stability.

3.2.8 Scalability

As the number of users and variety of feedback is ever increasing, and the range of items ever expanding, it is paramount that recommendation systems are able to scale to meet this demand. There are a combination of different factors such as training time, prediction time and memory requirements to measure the feasibility and scalability of a system [13]. These three metrics are self-explanatory with the desired outcome being lower training and prediction time as well as smaller memory requirements. Prediction time will be recorded as the time required to generate a recommendation list for each target user and this will represent the scalability of the systems in this dissertation.

3.3 Going Forward

To summarise, the metrics that will be used to measure improvements in the recommendation systems, will be:

- Accuracy
- Serendipity
- Diversity
- Robustness and Stability
- Scalability

Chapter 4

Design

4.1 Overview

In this chapter I will give a brief high level overview of the three recommendation systems implemented in the course of this dissertation. These systems are:

- Combined Collaborative Filtering (CCF)
- The Social Network's Influencers (SNI)
- Friend Recommendation System (FriendRec)

Each system's description is also accompanied with a diagram in order to aid the visualisation of the system. These diagrams are slightly simplified and are here for descriptive purposes.

4.2 Combined Collaborative Filtering

The first system implemented was the Combined Collaborative Filtering (CCF) system which is used for all CF experiments and acts as a baseline. The Last.Fm dataset contains friendship, tagging and rating data. The purpose of the CCF system is to use all the information provided about a specific user in order to determine the Last.Fm users that are most similar to the user in question. I accomplish this task by breaking the system down into three parts: Rating, Friendship and Tagging similarity.

Once I have calculated these similarity scores I multiply each score by a predefined weight, ensuring the most relevant similarity score has the largest impact on the overall similarity of the two users. The system has also been designed with weights in order to quickly remove the effect of certain data on the overall similarity score. For example by setting $\alpha = 0$ we are able to remove the rating similarity score's effect on the overall similarity of the two users.

Upon generating a similarity score between the target user and all other users of the system, I then select the top K users with the highest similarity score. This is an impor-

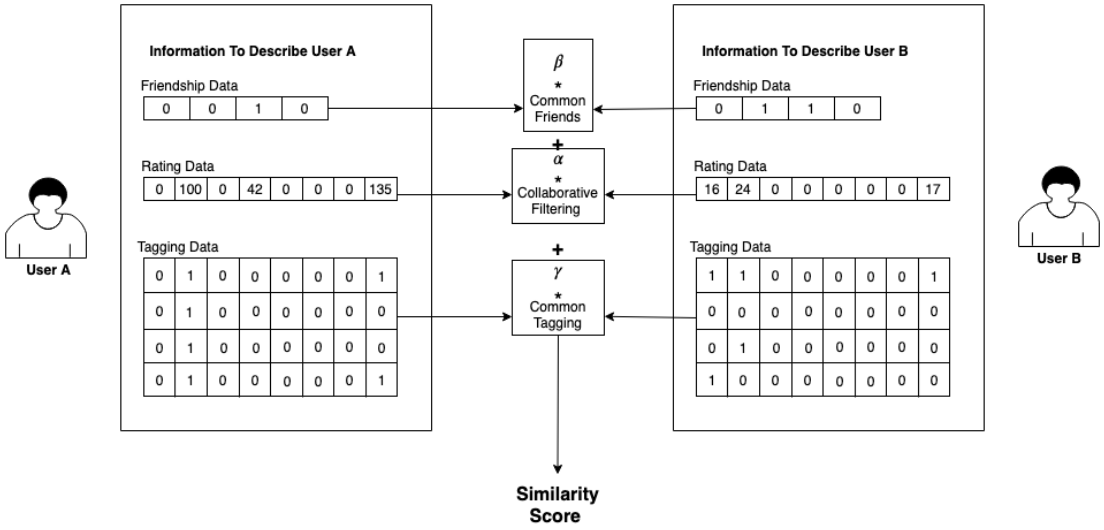

Figure 4.1: Collaborative Filtering system that combines: Rating, Tagging and Friendship data. The system combines weighted similarity scores for each type of data to generate an overall similarity score for the two users.

tant step in CF as the top K similar users will be used to generate a recommendation list.

4.3 The Social Network's Influencers

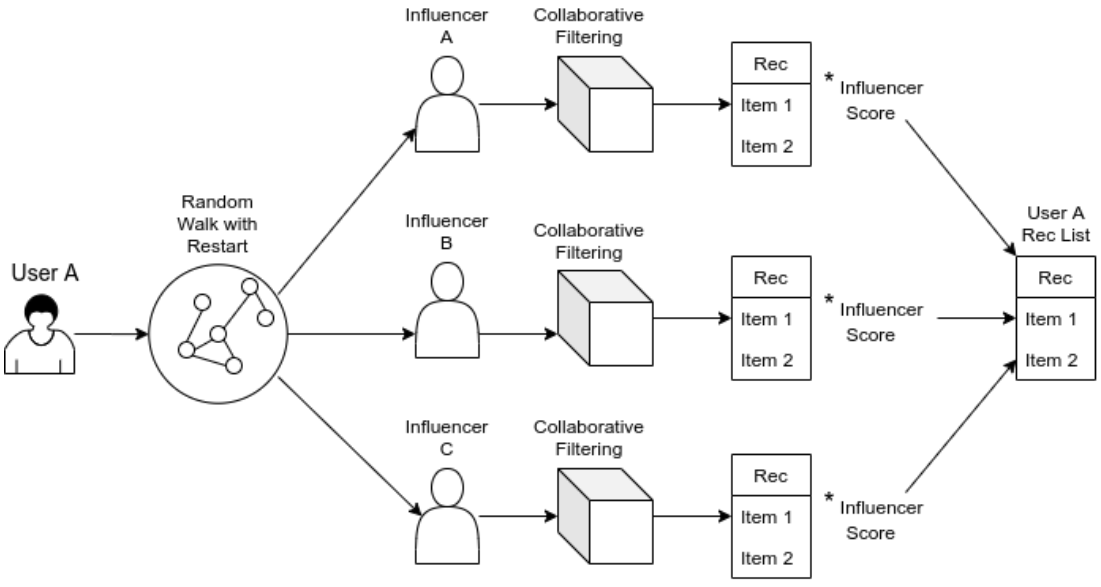

Figure 4.2: The Social Network's Influencers System, using Random Walk with Restart to find influencers and using CCF to generate recommendation lists.

The idea of using social network information in order to overcome the cold-start problem is a popular solution which has yielded impressive results [60, 53, 28].

The design of the system I implemented, which uses social network information to

overcome the cold-start problem, has been visualised in Figure 4.2. This system begins with the target user, in this case user A, and performs a Random Walk with Restart on the social network provided by Last.Fm. Random Walk with Restart finds the influence that each node in a network has on a given starting node. These influence scores are then used to determine the three most influential nodes in the target user's social network. The CCF system proposed in section 4.2 is then applied to each influencer in order to produce a recommendation list.

We then combine the three influencers' recommendation lists by multiplying each artist in a list with the individual's influence score, which was calculated during the Random Walk with Restart step. Each artists' new score was summed and the top 20 artists with the highest scores formed the recommendation list for the target user.

4.4 Friend Recommendation System

"*You're the average of the five people you spend most of your time with.*"

— Jim Rohn

The Friend Recommendation System (FriendRec) is a simplistic design, based on the idea that each individual is the average of their five closest friends. Since I found no effective way of calculating trust between users in the network, I decided that the target user would be the sum of all their friendships rather than their five closest friends. As shown in Figure 4.3 the solid lines represent the target user's friends whereas the dashed lines represent their friends' friendships. We can also see from the diagram that each friend has a list which represents that user's top 50 artists. These lists are used to form two lists for the target user: Artist Total Listen count and Artist count. The Artist Total Listen count is the sum of each individual friend's listen count for a given artist. This sum is equivalent to the total time that the target user's friends have listened to each artist. The second list is the Artist count which contains the artists' IDs along with the number of friends that have the artist present in their top 50 list on Last.Fm. This system has the option to be extended to account for not only the target user's direct friends but also their friends of friends.

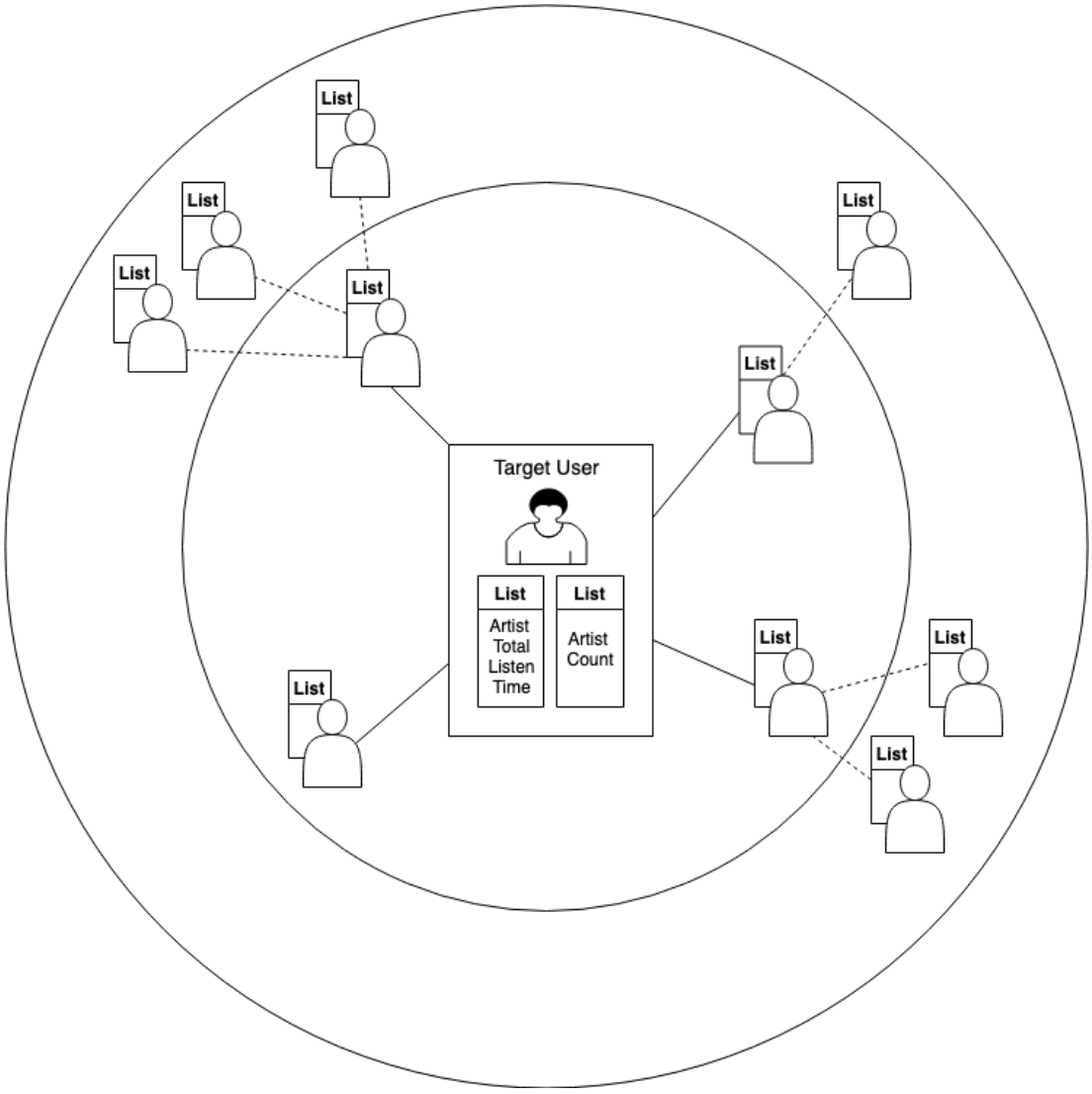

Figure 4.3: Friend Recommendation System using friends' listened to artists to generate recommendations for the target user.

Chapter 5

Implementation

5.1 Overview

This chapter will provide a deeper insight into the systems described in Chapter 4. I will describe the data structures, functions, and methods I employed in order to implement these systems. In addition, I will provide segments of my code to complement the description of these methods.

5.1.1 Data Structures

Panda's Series is a one-dimensional labeled array capable of holding any data type [4].

Panda's DataFrame is a size-mutable, two-dimensional labeled data structure, the columns of which can be of different types. It can be thought of as a dictionary of Panda's Series data structures [3].

SciPy's LIL Matrix is a row-based list of lists sparse matrix. This allows for the incremental construction of sparse matrices. Sparse matrices can be used in arithmetic operations: they support addition, subtraction, multiplication, division, and matrix power. The *lil_matrix* class supports basic slicing and complex indexing with a similar syntax to NumPy arrays [10]. By only displaying the non-zero elements, this makes visualising the sparse matrices easier than multidimensional arrays.

Compressed Sparse Row format is a method of storing sparse matrices to allow for efficient arithmetic operations, row slicing and fast matrix vector products [8].

Dictionaries are data structures which store key value pairs, with the requirement that the keys are unique [6]. Keys must be unique as they are used to index the data structure. Keys can be any immutable type, e.g. strings and numbers.

5.1.2 Functions

progress.bar.Bar is a progress reporting function for python, this is nonessential and used solely to gauge the progress of the system when generating recommendations [5].

scipy.stats.pearsonr is used to calculate the Pearson correlation coefficient and p-value for testing non-correlation [9]. The Pearson correlation coefficient measures the linear relationship between two datasets, with a value of +1 indicating perfect positive correlation and -1 indicating perfect negative correlation [48]. The calculation of the p-value relies on the assumption that each dataset is normally distributed. The p-value roughly indicates the probability that the strength of correlation produced by the system occurred by chance [14].

itertools.islice makes an iterator that returns selected elements from the iterable [7]. This allows me to index into a data structure and select the desired rows with minimal code, increasing the readability of my code.

5.2 Top K Users' Recommendations

In order to generate a recommendation list based on artists rated by the top K most similar users, I used:

- **Data Structures**
    1. Panda's DataFrame
    2. NumPy Array
    3. Dictionary
    4. List
- **Imported Function**
    1. progress.bar.Bar
    2. itertools.islice

Please reference Figure 5.1 during the following discussion.

5.2.1 Generating Recommendation Lists

To begin generating a recommendation list I started by taking a dictionary of the top K most similar users to a given target user. This dictionary's keys are the users' IDs and its values are the users' computed similarity scores.

The first step is to create another dictionary to store the artists' recommendation scores. These scores will be the sum of the top K users' weighted rating scores for that artist. For all of the top K users, I create a Panda's DataFrame of the artists rated by that user. This DataFrame is used to create a list containing all their rated artists. I then iterated over this list of artists normalising each of their listen counts using the following equation:

$$normalised_{ai} = \frac{count_{ai} - (min_a - 1)}{max_a - (min_a - 1)} \tag{5.1}$$

In this equation the max_a is the maximum and min_a is the minimum listen count of that user. $count_{ai}$ is the listen count of the artist for that user and $normalised_{ai}$ is

```
def recommendations(self, user_id, k_users):
    rating_dict = {}

    bar = Bar('Processing', max=len(k_users))
    for user in k_users:
        bar.next()
        # Get all the artists the user has rated
        user_ratings = self.user_artists.loc[self.user_artists['user_id'] == user]
        artist_list = user_ratings['artist_id'].tolist()

        # Find max, min ratings for the user and get the denominator for normalisation equation
        max_value = user_ratings['weight'].max()
        min_value = user_ratings['weight'].min() - 1
        denom = max_value - min_value

        for artist in artist_list:
            # Get the users ratings for the artist
            artist_weight = user_ratings.loc[user_ratings['artist_id'] == artist]['weight'].item()

            # Normalise the users rating
            artist_rating = (artist_weight - min_value)/denom
            user_score = artist_rating * k_users[user]

            # Add the users normalised rating for an artist to all other ratings for that artsit
            if artist in rating_dict:
                rating_dict[artist] = rating_dict[artist] + user_score
            else:
                rating_dict[artist] = user_score
    bar.finish()

    # Sort and select the top 20 artists with the highest combined rating to create the recommendations
    rating_dict = {k: v for k, v in sorted(rating_dict.items(), key=lambda item: item[1], reverse = True)}
    rec_list = dict(islice(rating_dict.items(), 20))

    hits = self.compare_list(user_id, rec_list)
    print(hits)

    return rec_list, hits
```

Figure 5.1: Top K User's Recommendations Implementation.

the normalised artist listen count for that user. This equation normalises the user's listen count so that very active users cannot hijack the recommendation system. The reason for subtracting one from the minimum listen count is due to the fact that the user's least listened to song should still have an impact on the recommendation system. Subtracting all listen counts by the minimum would remove this artist entirely.

After normalising the user's listen counts they are multiplied by the user's similarity score to the target user, the result of this multiplication is the artist's recommendation score for that user. This score is then summed with all the other users' recommendation scores for said artist and stored in the dictionary discussed earlier. Once all the artists' recommendation scores have been calculated the top 20 artists with the highest scores are then selected. These selected artists form the recommendation list for the target user.

5.3 Combined Collaborative Filtering

In order to implement the CCF system, I used:

- **Data Structures**
    1. Panda's DataFrame
    2. Panda's Series

3. SciPy's LIL Matrix
4. List
5. Dictionary
6. NumPy Array

- **Imported Function**
  1. progress.bar.Bar
  2. scipy.stats.pearsonr
  3. itertools.islice

5.3.1 Rating Similarity Score

In order to calculate the rating similarity score, I used:

- **Data Structures**
  1. Panda's DataFrame and Series
  2. NumPy Array
- **Imported Function**
  1. scipy.stats.pearsonr

Please reference Figure 5.2 during the following discussion.

```
def get_rating_weights(self, a_value, u_value):
    # Find the two users artist rating information
    user_a = self.user_artists.loc[self.user_artists['user_id'] == a_value]
    user_b = self.user_artists.loc[self.user_artists['user_id'] == u_value]

    # Create and initialise the array's to store the two users ratings
    len_artist = len(self.artists.index)
    a_rating = np.zeros(len_artist)
    b_rating = np.zeros(len_artist)

    # Fill the two array's
    for art_id in user_a['artist_id']:
        a_index = self.artists.index[self.artists['id'] == art_id].tolist()
        art_inp = user_a.loc[user_a['artist_id'] == art_id]
        a_rating[a_index] = art_inp['weight']

    for art_id in user_b['artist_id']:
        b_index = self.artists.index[self.artists['id'] == art_id].tolist()
        art_inp = user_b.loc[user_b['artist_id'] == art_id]
        b_rating[b_index] = art_inp['weight']

    # Calculate the Pearsons correlation coefficient of the two arrays
    pcc, p_value = pearsonr(a_rating, b_rating)
    return pcc
```

Figure 5.2: Rating Similarity Score Implementation.

Calculating the rating similarity score began by creating two DataFrames which stored the ratings provided by the target user and User B. I then created two NumPy arrays with the same length as the number of artists in the dataset and set the value stored at each index to zero. The listen count for all the artists in the target user's rating DataFrame was inserted into a single NumPy array at the correct index for that artist. This process was then repeated for User B. By inserting the rating data into the two NumPy arrays I was able to compute the Pearson correlation coefficient using SciPy's pearsonr function. This function returns the Pearson correlation coefficient and a two-tailed p-value. We are only interested in the Pearson correlation coefficient value and use this as the rating similarity score for the target user and User B.

5.3.2 Tagging Similarity Score

In order to calculate the tagging similarity score I used:

- **Data Structures**
    1. Panda's DataFrame and Series
    2. NumPy Array
- **Imported Function**: [None]

Please reference Figure 5.3 during the following discussion.

```
def get_fraction_tag(self, user_a, user_u):
    # Get the two users tags and combine them to find all the artists tagged by either user
    a_tags = self.user_taggedartists.loc[self.user_taggedartists['user_id'] == user_a]
    u_tags = self.user_taggedartists.loc[self.user_taggedartists['user_id'] == user_u]

    combined_tags = pd.concat([a_tags, u_tags])
    tagged_artists = combined_tags['artist_id'].unique()

    tag_weight = 0

    for artist in tagged_artists:
        # Find all the tags assigned to an artist
        art_tag_a = a_tags.loc[a_tags['artist_id'] == artist]
        art_tag_u = u_tags.loc[u_tags['artist_id'] == artist]
        denom_a = len(art_tag_a)
        denom_u = len(art_tag_u)

        # Find all the tags they have in common for a certain artist
        common_tags = len(set(art_tag_a['tag_id']) & set(art_tag_u['tag_id']))

        # If they have a tag in common for the artist increment the tag weight
        if denom_a > 0 and denom_u > 0 and common_tags > 0:
            tag_weight = tag_weight + (common_tags/denom_a * common_tags/denom_u)
    return tag_weight
```

Figure 5.3: Tagging Similarity Score Implementation.

I began calculating the tagging similarity score by creating two DataFrames which stored the social tagging information for the target user and User B. I then created a new DataFrame to contain both users' tagging information, this was achieved by appending User B's DataFrame onto the target user's. The purpose of this concatenation is to find the disjunction of artists that the two users have tagged. I achieved this through the use

of the DataFrames *unique* function which returns a NumPy Array of the set of artist IDs that have been tagged by the two users. This was done so that I only iterate over the artists that have been tagged by at least one of the users. I then create a variable called *tagweight* and initialise it to zero. For all artists in the NumPy Array I used the following equation:

$$tagweight = \sum_{i}^{I} (\frac{common_i}{tags_{ai}}) * (\frac{common_i}{tags_{bi}}) \tag{5.2}$$

Where *tagweight* is the tagging similarity score for the target user and User B, I is the NumPy array of artists. $common_i$ is the number of common tags assigned to artist i by the target user and User B. $tags_{ai}$ is the total number of tags the target user assigned to artist i and $tags_{bi}$ is the number of tags User B assigned to artist i. This calculation increases the similarity score for users whose tags match when assigning fewer tags. I then return this summation as the tag similarity score between the two users.

5.3.3 Friendship Similarity Score

In order to calculate the friendship similarity score I used:

- **Data Structures**
  1. Panda's DataFrame
  2. SciPy's LIL Matrix
  3. List
  4. NumPy Array
- **Imported Function**
  1. scipy.stats.pearsonr

Please reference Figure 5.4 during the following discussion.

```
def get_friend_weights(self, user_a, user_u, adj_matrix, user_list):
    # Find user and friends index in list in order to index into the sparse matrix
    user_ind = user_list.index(user_a)
    friend_ind = user_list.index(user_u)

    # Get user and friends rows from the sparse matrix and convert to arrays
    list_a = adj_matrix.getrow(user_ind)
    list_u = adj_matrix.getrow(friend_ind)
    array_a = list_a.toarray()
    array_u = list_u.toarray()

    # Calculate the Pearsons correlation coefficient of the two arrays
    pcc, p_value = pearsonr(array_a[0], array_u[0])
    return pcc
```

Figure 5.4: Friendship Similarity Score Implementation.

I began calculating the friendship similarity score by creating a list to store the IDs of all the users that had at least one friendship. Similar to the process of finding the tagging similarity score, this was done in order to limit iterations to only users that had friendship data. By using the LIL Matrix data structure I was able to create a sparse matrix to store the friendship information between users. The user's position in the list is their index in the sparse matrix, this was used to reduce the size of the matrix and the list was preserved for future referencing into the matrix.

I then find the index of the target user and User B in the list and use these indices to extract the correct row of information in the LIL Matrix. The rows were then converted into NumPy arrays and fed into SciPy's pearsonr function. This function returns the Pearson correlation coefficient which we use as the friendship similarity score between the two users.

5.3.4 Combining Similarity Scores

In order to combine all the similarity scores I used:

- **Data Structures**
    1. Panda's DataFrame
    2. SciPy's LIL Matrix
    3. List
    4. Dictionary
    5. NumPy Array
- **Imported Function**
    1. itertools.islice
    2. progress.bar.Bar

Please reference Figure 5.5 during the following discussion.

In order to combine the similarity scores I created a dictionary to store each users' ID and their similarity score with the target user. I then created a NumPy array of all the users with rating data, removing the target user from the array. This array was then iterated over calculating the similarity score for each of the metrics described previously: rating, tagging and friendship. In order to combine these similarity scores I performed the following calculation:

$$CombinedSimilarity = \alpha * rating + \beta * friendship + \gamma * tagging \tag{5.3}$$

Where $rating$, $tagging$ and $friendship$ represent their respective similarity scores and α, β and γ represent weights. The $CombinedSimilarity$ is the combined score for this user and is added to the aforementioned dictionary with the users' IDs as the key and the combined score as the value.

```
def combined_weights(self, alpha, beta, gamma, user_a, subset):
    print(self.user_artists)
    user_weights = {}

    # Subset variable is for testing purposes
    if subset:
        users = [922]
    else:
        # Get all the User IDs minus the target user's ID
        users = self.user_artists.loc[self.user_artists['user_id']!= user_a]['user_id'].unique()

    bar = Bar('Processing', max=len(users))
    friend_martix, user_list = self.get_adjacency_matrix()

    for user_u in users:
        bar.next()
        # Find the three similarity scores for the user and the target user
        friend_weight = self.get_friend_weights(user_a, user_u, friend_martix, user_list)
        rating_weight = self.get_rating_weights(user_a, user_u)
        tag_weight = self.get_fraction_tag(user_a, user_u)

        # Store the summed weighted similarity score with the User ID
        user_weights[user_u] = alpha * rating_weight + beta * friend_weight + gamma * tag_weight
    bar.finish()

    # Sort and select the top 20 users with the highest summed weighted similarity score
    sorted_weights = {k: v for k, v in sorted(user_weights.items(), key=lambda item: item[1], reverse = True)}
    top_users = dict(islice(sorted_weights.items(), 20))
    print(top_users)

    return top_users
```

Figure 5.5: The Combined Collaborative Filtering System Implementation.

Upon calculating a similarity score for all the users the top 20 users with the highest combined similarity scores are selected using itertools *islice* function. These 20 users were then passed to the Top K Users' Recommendations described in section 5.2 in order to generate a recommendation list for the target user.

5.4 The Social Network's Influencers

5.4.1 Finding the Influencers

In order to find the biggest influencers on the target user in their social network I used:

- **Data Structures**
    1. Panda's DataFrame
    2. SciPy's LIL Matrix
    3. Compressed Sparse Row
    4. List
    5. Dictionary
    6. NumPy Array
- **Imported Function**: [None]

Please reference Figure 5.6 during the following discussion.

I began by creating a list of all the users with at least one user-user connection. I then created a NumPy array full of zeros, this array was the same length as the previously created list. This array stores the respective probabilities of being at any given node at

```python
def random_w_r(self, user, restart_prob):
    # Get number of users with user-user connections
    user_list = self.user_friends['user_id'].unique().tolist()
    mat_size = len(user_list)

    # Create starting and previous probabilities vectors
    user_ind = user_list.index(user)
    s = np.zeros(mat_size)
    s[user_ind] = 1
    x = s
    x_prev = s = np.zeros(mat_size)

    # Get user-user connection matrix
    adj_matrix = self.get_adjacency_matrix()
    csr_adj_mat = adj_matrix.tocsr().transpose()

    # Preform Random Walk with Restart Algorithm
    while(not ((x == x_prev).all())):
        storage = restart_prob * s + (1 - restart_prob) * csr_adj_mat * x
        x_prev = x
        x = storage

    return x, user_list
```

Figure 5.6: Random Walk with Restart Implementation.

a specific time in the Random Walk with Restart algorithm. I then found the index of the target user in the the list of user-user connections and set the corresponding index in the array to 1 as this is the starting node and has a probability of 1 at the start. I then created a NumPy array full of zeros with the same length as the list and the other array to store the probabilities of the algorithm at the previous step, this is used to show convergence which in turn stops the algorithm. Following this, a sparse matrix was created using the LIL Matrix data structure, the method of filling this data structure is the same as the process used in subsection 5.3.3.

Once the LIL Matrix was generated, I converted it to a Compressed Sparse Row format which allowed me to transpose the matrix and apply the Random Walk with Restart algorithm described in subsection 2.5.1. The output of this algorithm is a new array containing the probabilities of being at each node at the next stage of the Random Walk. This is then compared to the previous step and if these NumPy arrays are identical then convergence has happened and the algorithm is stopped. Once the algorithm has stopped I have an array of values that correspond to the probability of visiting each node in the network given a start node and a probability of restart p. Those users with a larger probability are visited more often and therefore I assume they have a higher influence on the starting user. I then selected the top three users and find their corresponding IDs in the list created at the start. A dictionary is created to hold these IDs and their probabilities which I will now refer to as influence scores. This dictionary is then used to produce the target user's recommendation list.

5.4.2 Producing the Recommendation List

In order to produce the recommendation list for the target user given the three biggest influencers in their social network and their influence score, I used:

- **Data Structures**
    1. Panda's DataFrame
    2. Panda's Series
    3. SciPy's LIL Matrix
    4. List
    5. Dictionary
    6. NumPy Array
- **Imported Function**
    1. progress.bar.Bar
    2. scipy.stats.pearsonr
    3. itertools.islice

After finding the top influencers in the target user's social network and their influence score I then decided to use the CCF system on each of the influencers generating a recommendation list for each. Once I had the influencers' recommendation list, I then multiplied each artist's recommendation score with their influence score; I will call this value the artists' weighted score. I then create a list with the disjunction of artists in the influencers' recommendation lists by appending their lists together and taking the set of the artists' IDs. Then for every artist in this set I sum up the influencers' weighted score for the given artist and store the artists' IDs and summed weighted score in a dictionary. I then select the top 20 artists with the highest summed weighted scores from this dictionary to form the target user's recommendation list.

5.5 Friend Recommendation System

5.5.1 Direct Friendship

In order to generate a recommendation by using the information provided by the target user's direct friendships, I used:

- **Data Structures**
    1. Panda's DataFrame
    2. List
- **Imported Function**: [None]

Please reference Figure 5.7 and Figure 5.8 during the following discussion.

```
def handle_friend_rec(self, type_rec, user_u, count_only=True, remove_artist = []):
    if (type_rec == "fof"):
        print("Fof")

        # Get a list of all the users friends and the artists they listen to
        u_friends = self.user_friends.loc[self.user_friends["user_id"] == user_u]
        artist_pd = self.find_friends_artists(user_u)

        # Get the number of users that listen to each artist and the sum of their listen times
        count_artist, sum_artist = self.get_sum_count(artist_pd)
        max_rating = artist_pd['weight'].max()

        # Repeat the process for all the friends of the target users friends
        for user in u_friends['friend_id']:
            artist_pd = self.find_friends_artists(user)
            friend_count_artist, friend_sum_artist = self.get_sum_count(artist_pd)
            count_artist = count_artist.add(friend_count_artist, fill_value = 0)
            sum_artist = sum_artist.add(friend_sum_artist, fill_value = 0)

            friend_max_rating = artist_pd['weight'].max()
            if friend_max_rating > max_rating:
                max_rating = friend_max_rating
    elif(type_rec == "friends"):
        # Get just the target users friends recommendations
        artist_pd = self.find_friends_artists(user_u)
        count_artist, sum_artist = self.get_sum_count(artist_pd)
    else:
        print("Handle Friend Rec error: fof or friends")

    if count_only:
        return self.get_rec_list(count_artist, sum_artist, remove_artist)
    else:
        # Divide the sum of all artist by the highest listened to artist by a single user.
        # Creating a composite score for each artists
        ratio = sum_artist.divide(max_rating)
        friend_weights = count_artist.add(ratio)
    return friend_weights
```

Figure 5.7: The Friend Recommendation System Implementation.

```
def get_sum_count(self, artist_pd):
    sum_artist = artist_pd.groupby('artist_id')['weight'].sum()
    count_artist = artist_pd.groupby('artist_id')['weight'].count()
    return count_artist, sum_artist
```

Figure 5.8: Producing the Count and Sum DataFrames in the Friend Recommendation System.

I begin by creating a Panda's DataFrame containing all of the target user's direct friendships. This DataFrame is then used to create a list containing all the target user's friends' IDs. Another DataFrame is then created to store all the users in this list's rating information. Once the DataFrame contains all the friends' rating information I group by artist ID and create a further two DataFrames: one for the sum of each artist's listen counts and the other for the number of appearances of an artist in the original DataFrame. I then pass these two DataFrames to the system's recommendation list generator.

5.5.2 Friends of Friends

In order to generate recommendations by using the information provided not only by the target user's direct friendships but also their friends of friends, I used:

- **Data Structures**
    1. Panda's DataFrame
    2. List
- **Imported Function**: [None]

Please reference Figure 5.7 and Figure 5.8 during the following discussion.

I began by performing the same steps as in the section above regarding direct friendships, which produced the count and sum DataFrames. I then obtained a DataFrame of the target user's friendship information. I iterated over the targets user's direct friends and repeat the same steps as before producing count and sum DataFrames for the friends' direct friendships. These DataFrames are then added to the two original DataFrames producing a total sum and total count DataFrame for all the users in the friends of friends network. These two DataFrames were then passed to the system's recommendation list generator.

5.5.3 Similar Friends

In order to generate recommendations using the direct friends and friends of friends implementation while increasing the weight, and therefore impact, of similar friends I used:

- **Data Structures**
    1. Panda's DataFrame
    2. List
- **Imported Function**: [None]

Please reference Figure 5.7 and Figure 5.8 during the following discussion.

Another option of the Friend Recommendation System is increasing the impact of similar friends on the recommendations generated. I calculate similar friends through their use of social tagging. This can be done with either the direct friend or friends of friends implementation described previously. The major difference being that this method combines the sum and count DataFrames when generating recommendations. It does this by finding the largest single user rating in the network and divides the total listen count for each artist in the sum DataFrame by this value. This divided listen count is then added to the count DataFrame, resulting in a combined recommendation score for each artist.

After producing a DataFrame of combined recommendation scores, I created a list to store all users who had at least one tag in common with the target user. I then created a sum and count DataFrame from all the similar users, finally combining them as before into a single combined recommendation score DataFrame. Weights are then applied to the combined DataFrames and added together. The top 20 artists with the highest scores are then selected and form the target user's recommendation list.

5.5.4 Recommendation List Generation

In order to generate a recommendation list in the Friend Recommendation system, I used:

- **Data Structures**
  1. Panda's DataFrame
  2. List
- **Imported Function**: [None]

Please reference Figure 5.9 during the following discussion.

```
def get_rec_list(self, count_artist, sum_artist, remove_artist):
    rec = []
    i = 1
    while(len(rec) < 20):
        # Find all artists with the max count in the Count DataFrame
        max_count = count_artist.max()
        max_list = list(count_artist[count_artist == max_count].keys())
        # If adding these artists would make the rec list be greater than 20
        if(len(rec) + len(max_list) > 20):
            # Find the set of artist to add to the rec list
            k_left = 20 - len(rec)
            max_list = self.find_best_values(max_list, sum_artist, k_left)

        # This line is used for an experiment
        max_list = [i for i in max_list if i not in remove_artist]
        # Add artists to rec list and remove users with the max count from Count Dataframe
        rec.extend(max_list)
        count_artist = count_artist[count_artist != max_count]
        i = i+1
    print(rec)
    return rec
```

Figure 5.9: Friend Recommendation List Generator Implementation.

I began by creating a list to store the recommended artist IDs and as long as this recommendation list had less than 20 artist IDs, I repeated the following process. I found the max value in the count DataFrame, which was passed to the recommendation list generator, and created a list containing all the artists with a count equal to this max value. If this list plus the recommendation list had a size greater than 20, I found the number of spaces available in the recommendation list. I then filled the recommendation list with the artists that had the highest total listen count from the sum DataFrame. If, however, the number did not exceed 20 then I added all the artists with the max count to the recommendation list and removed them from the count and sum DataFrame, resulting in a lower max count in future iterations. Once the recommendation list contained all 20 artists the process stopped and the user was provided with these recommendations.

Chapter 6

Experimental Results

6.1 Overview

In this chapter I will discuss the experiments used to test and compare the systems described in Chapter 4 and Chapter 5.

6.2 Setup

In order to select the optimal hyperparameter (α, β and γ) values discussed in Chapter 4 and Chapter 5, I split the dataset into a training, validation and test set. Due to the time required to carry out each experiment I decided to only include 10 users in the validation and the test set. These users were selected as they possessed desirable criteria: more than three friendships and 50 rated artists. This was achieved by creating a list containing the IDs for every user with artist ratings. I then randomly selected a single user from this list and checked that the selected user had more than three friendships as well as the full 50 rated artists, which is the maximum number of ratings per user in this dataset. If the user matched this criteria then they were added to the list of validation or test users. This process was repeated until there were ten users in the validation and test set.

I chose these requirements due to the fact that I wanted to increase the probability of a recommendation list containing a hit. Therefore by ensuring that each user in the validation and test set had rated the full 50 artists I increased the potential for a valid recommendation. Users with less than the 20 artists required to fill a recommendation list would most likely provide less discriminatory results regarding the performance of each recommendation system. The decision to only include users with more than three user-user connections was due to the fact that these experiments look into the use of alternative information to provide valid recommendations with the main focus being on analysing individuals' social networks to provide recommendations. To ensure that the users were part of a social network, I set the threshold of user-user connections at three, this number was chosen as the SNI system described in section 4.3 requires a minimum of three users in the individual's social network to provide recommendations.

Validation Set		
User ID	No. of Tags	No. of friends
6	14	5
40	509	11
133	69	51
332	4	29
491	47	5
925	3	34
1084	16	6
1136	5	25
1301	10	33
1581	30	19

Table 6.1: Basic Information on randomly selected users in the validation set used to select the hyperparameters of each Recommendation System.

Test Set		
User ID	No. of Tags	No. of friends
3	12	7
12	788	40
128	13	31
478	4	4
582	191	14
912	114	10
1278	233	33
1375	11	7
1458	39	13
1509	13	10

Table 6.2: Basic Information on randomly selected users in the test set used to compare the performance of different Recommendation Systems.

The validation set which was selected through this process can be found in Table 6.1, while the test set can be found in Table 6.2.

When composing the recommendation lists, I used the *K*-Nearest Neighbours approach selecting the 20 nearest neighbours to the user and taking only these users' information into account when producing a list of 20 recommended artists for that user. I am using the recommendation lists to measure the accuracy and will be using Precision@20 to measure this. Precision@K is a valid metric commonly used to measure the accuracy of recommendation systems [51, 59] and is described in subsection 3.2.1.

6.3 Baseline

6.3.1 Baseline 1: Popular Artists

The first baseline I have decided to implement is a recommendation system that simply creates an impersonalised recommendation list full of the artists who have been rated

Artist Name	Artist ID	No. of listeners
Lady Gaga	89	608
Britney Spears	289	519
Rihanna	288	481
The Beatles	227	479
Katy Perry	300	470
Madonna	67	426
Avril Lavigne	333	414
Christina Aguilera	292	403
Muse	190	399
Paramore	498	398
Beyonce	295	395
Radiohead	154	391
Coldplay	65	368
Ke$ha	466	359
Shakira	701	317
The Killers	229	303
P!nk	302	302
Black Eyed Peas	306	301
Kylie Minogue	55	295
Miley Cyrus	461	284

Table 6.3: The most popular artists in the Last.Fm dataset after removing the validation and test sets rating information.

by the most users. These artists were found by removing all validation and test user information from the ratings dataset and finding the number of appearances of each artist. The top 20 most rated artists formed the recommendation list for this system, the full list can be found in Table 6.3.

6.3.2 Baseline 2: Friendship Collaborative Filtering

A baseline is the minimum starting point which is used for comparison. Since the cold-start user only has friendship information, my second baseline is a CF system using only user-user connections. This can be viewed as a new user who has signed up to your platform through a social media app such as Facebook.

This can be considered the typical cold-start user. They have yet to rate any artist and have not provided any tagging information on an artist. This can be achieved using the CCF system by setting both α and γ to zero.

6.4 Combined Collaborative Filtering

The CCF system works by taking the similarity score of two users for three data sources and producing a total similarity score. Since each data source does not contribute equally to generating valid recommendations for a user, the first experiment was to determine the weights to be applied to each sub-similarity score in order to achieve

USER ID	ALPHA ONLY	BETA ONLY	GAMMA ONLY
6	0.25	0.00	0.05
40	0.35	0.25	0.10
133	0.65	0.65	0.65
332	0.30	0.30	0.05
491	0.65	0.35	0.50
925	0.40	0.45	0.10
1084	0.80	0.30	0.45
1136	0.75	0.45	0.15
1301	0.55	0.45	0.10
1581	0.25	0.15	0.00
MEAN	0.495	0.335	0.215

Table 6.4: The first experiment ran, tested using the Precision@20 to see the effects of generating recommendation lists based solely on the information provided by one similarity score.

the highest accuracy.

For these experiments I randomly selected half of the validation and test set's artists' ratings and removed them from the dataset. This means that they would not be used in the CF system but would still count as hits if predicted by the recommendation system.

The first experiment conducted was to see the effects of selecting the top K users based solely on one similarity score. This was achieved by setting all weights to zero, bar the weight of the similarity metric in question. The results can be seen in Table 6.4. These show that as expected the rating data contained the most important information for predicting user ratings. The tagging and friendship CF systems performed worse with friendship data providing slightly more discriminative information than social tagging data.

After running these experiments I wanted to observe the effects of combining the similarity metrics in a variety of ways, resulting in the tests that can be found in Table 6.5. The first four experiments were decided before running any further experiments. The next three setups were decided upon receiving the results in Table 6.6. Finally the last three experiments were conducted to check that no further improvements could be achieved from incorporating all three similarity scores.

Upon performing the initial experiments into combining all three similarity scores in the CCF system, I realised that the best performing setup was still the alpha-only experiment where $\alpha = 1$ and $\beta = \gamma = 0$. Slightly surprised, I decided to investigate if there were any benefits that could be gained from using all three similarity scores. Therefore I conducted further experiments in order to refine the system's weights. Since in these initial experiments big-alpha performed better than big-gamma, which performs better than big-beta, I decided upon the next three experiments, the results of which can be found in Table 6.7.

By reducing the impact of tagging and friendship data on the recommendation system I was able to increase the total number of hits in the recommendation system. How-

Experiment Name	Alpha	Beta	Gamma
Big Alpha	0.5	0.25	0.25
Big Beta	0.25	0.5	0.25
Big Gamma	0.25	0.25	0.5
Even	1/3	1/3	1/3
γ-Six	0.6	0.1	0.3
γ-Seven	0.7	0.1	0.2
γ-Eight	0.8	0.05	0.15
β-Six	0.6	0.3	0.1
β-Seven	0.7	0.2	0.1
β-Eight	0.8	0.15	0.05

Table 6.5: The experimental setups designed to find the best weights to apply when combining all three similarity scores in the combined collaborative filtering system.

User ID	Big Alpha	Big Beta	Big Gamma	Even
6	0.25	0.00	0.15	0.10
40	0.10	0.10	0.10	0.10
133	0.65	0.65	0.65	0.65
332	0.30	0.30	0.35	0.35
491	0.50	0.50	0.50	0.50
925	0.25	0.40	0.40	0.40
1084	0.55	0.55	0.50	0.50
1136	0.65	0.55	0.60	0.60
1301	0.60	0.65	0.50	0.65
1581	0.20	0.15	0.15	0.25
Mean	0.405	0.385	0.390	0.410

Table 6.6: The initial tests to find the best alpha, beta and gamma values for the combined collaborative filtering system, using Precision@20. The weight values for these tests can be found in Table 6.5.

ever these results were still significantly lower than those achieved in the alpha-only experiment. Since friendship data produced a higher Precision@20 than social tagging data in the beta-only and gamma-only experiments, I decided to perform the final three experiments.

The results of the final three experiments can be found in Table 6.8, showing both β-Seven and β-Eight coming close to the alpha-only experiment's Precision@20. Since the CCF system is being used in a cold-start situation and in the absence of rating information, β-Eight was selected as the top performing cold-start setup and is used as the CCF weights from this point forward.

6.5 The Social Network's Influencers

In order to gain information about a user with only social network information I decided to apply Random Walk with Restart on the users' social networks. By setting the starting node to the target user I was able to find the users in the network with the

USER ID	ALPHA ONLY	γ-SIX	γ-SEVEN	γ-EIGHT
6	0.25	0.25	0.25	0.25
40	0.35	0.10	0.10	0.10
133	0.65	0.65	0.65	0.65
332	0.30	0.30	0.30	0.25
491	0.65	0.50	0.50	0.50
925	0.40	0.40	0.40	0.40
1084	0.80	0.55	0.65	0.85
1136	0.75	0.65	0.65	0.70
1301	0.55	0.65	0.65	0.55
1581	0.25	0.20	0.25	0.25
MEAN	0.495	0.425	0.440	0.450

Table 6.7: Refining the alpha, beta and gamma values used in the combined collaborative filtering system following the initial test results gained when incorporating all three similarity scores. The results are the Precision@20 for each experiment. The weight values for these tests can be found in Table 6.5.

highest influence on the target user. In the following experiments I opted to find the three most influential users on the target user.

Upon finding the three influencers I stored the converged probability of being at each of these nodes and used this as their respective influence scores. Since each influencer has rating, tagging and friendship data I performed CCF using all three similarity scores in order to generate three recommendation lists. These recommendation lists were then combined to generate a final recommendation list for the target user.

Random Walk with Restart requires a selection of the restart probability p. This value is the likelihood of the next node in the Random Walk being the initial node. The value of the restart probability was originally selected to $p = 0.8$ as this is the value recommended by [35]. Setting a higher restart probability forces the algorithm to return to the initial node more frequently. Therefore performing Random Walk in the neighbouring nodes results in a more personalised model. An experiment was also run with a smaller restart probability of $p = 0.2$. The reasoning behind this experiment was to observe the performance of a less personalised model. This was done in order to compare the recommendations produced from the influencers of a larger area of the network to the biggest influencers in close proximity to the target user.

The CCF parameters used for finding the recommendations in this experiment were set to $\alpha = 0.8$, $\beta = 0.15$ and $\gamma = 0.05$. By summing the multiplication of the relevance score of each artist on their influencer with the influencer score resulted in a composite score for each artist. This list was further reduced by finding the top 20 artists with the highest composite score.

As shown in Table 6.9, this implementation is not an effective solution for tackling the cold-start problem as the accuracy of the recommendation system is significantly smaller than both the popular recommendation systems and the friendship only CF system. In addition to producing poor recommendations, the average time required to generate a list for a single user was 870.17 seconds. This time is unacceptable for the

USER ID	ONLY ALPHA	β-SIX	β-SEVEN	β-EIGHT
6	0.25	0.25	0.25	0.25
40	0.35	0.15	0.15	0.15
133	0.65	0.70	0.70	0.60
332	0.30	0.30	0.40	0.30
491	0.65	0.50	0.55	0.70
925	0.40	0.25	0.40	0.40
1084	0.80	0.80	0.85	0.85
1136	0.75	0.65	0.65	0.65
1301	0.55	0.60	0.60	0.65
1581	0.25	0.25	0.25	0.25
MEAN	0.495	0.445	0.480	0.480

Table 6.8: Refining the alpha, beta and gamma values used in the combined collaborative filtering system upon deciding that alpha is the largest followed by beta then gamma. The results are the Precision@20 for each experiment. The weight values for these tests can be found in Table 6.5.

USER ID	POPULAR	CF BASELINE	SNI $p = 0.2$	SNI $p = 0.8$
6	0.00	0.00	0.00	0.00
40	0.00	0.25	0.00	0.05
133	0.70	0.65	0.20	0.10
332	0.30	0.30	0.00	0.00
491	0.00	0.35	0.05	0.00
925	0.25	0.45	0.25	0.10
1084	0.15	0.30	0.05	0.15
1136	0.50	0.45	0.00	0.05
1301	0.45	0.45	0.10	0.10
1581	0.00	0.15	0.05	0.05
MEAN	0.235	0.335	0.070	0.060

Table 6.9: The initial tests using Precision@20 to find the best restart probability p for the Random Walk with Restart algorithm in the Social Network's Influencers' system.

cold-start problem which has to generate recommendations quickly when a user first creates an account on the platform.

6.6 Friend Recommendation System

Since the SNI system performed significantly worse than users with similar user-user connections as the target user, I began to consider the fact that close friendships could be the key to providing valid recommendations to cold-start users. To begin my exploration into the impact of friendship on a cold-start user's recommendations, I decided to find all the artists rated by the target user's immediate friends. The concept of homophily is the assumption that people form strong positive social ties with people of similar interests [44]. If this is to hold true then this subset of artists will likely contain the majority of artists listened to by the target user. The task would then shift to

USER ID	POPULAR	CF BASELINE	FRIENDS	FOF
6	0.00	0.00	0.00	0.05
40	0.00	0.25	0.15	0.10
133	0.70	0.65	0.90	0.95
332	0.30	0.30	0.35	0.35
491	0.00	0.35	0.60	0.65
925	0.25	0.45	0.30	0.60
1084	0.15	0.30	0.50	0.15
1136	0.50	0.45	0.65	0.60
1301	0.45	0.45	0.65	0.65
1581	0.00	0.15	0.30	0.15
MEAN	0.235	0.335	0.440	0.425

Table 6.10: Experimentation into using the artists' count and total listen time of Friends and Friends of Friends (FoF) when generating recommendations for a target user compared to the two baselines. The results are the Precision@20 for each experiment.

finding a way of effectively and accurately reducing this group of artists to a list of recommendations for the target user.

The first method used was simply to find the artists that had been listened to most by the target user's direct friends and recommend them. Tiebreakers were settled by using the total listen time of each artist calculated by the target user's friends. This method is referred to as Friends in Table 6.10 and has proved to be an efficient and accurate system. The FriendRec system achieves a higher accuracy than both baselines for all users bar two in the validation set.

Despite the obvious benefits of using the FriendRec System certain users still had a low Precision@20 and in one case zero accurate recommendations. Therefore further tests were conducted to include rating information from friends of friends (FoF). This would be useful in the case where the target user had very few friendships. However it was uncertain if these users' recommendations should hold the same weight as the target user's direct friendships therefore different weights were applied to these connections. The following weights were applied; direct friendships, represented here as α, and indirect friendships, represented here as β:

- $\alpha + \beta$
- $2\alpha + \beta$
- $\alpha + 2\beta$

These experiments, however, showed a negligible difference to the recommendations generated, with marginally smaller Precision@20 than the direct friendship recommendation system and almost zero change when altering the weights. The conclusion is that for this validation set with the information provided there is very little gain to be made when introducing friends of friends into the recommendation system in this manner. The results of the simple $\alpha + \beta$ experiment can be found in Table 6.10, which gained a Precision@20 just shy of the Friends implementation.

USER ID	NEW CF BASELINE	FRIENDS
6	0.00	0.00
40	0.10	0.15
133	0.70	0.90
332	0.30	0.35
491	0.50	0.60
925	0.45	0.30
1084	0.40	0.50
1136	0.45	0.65
1301	0.45	0.65
1581	0.10	0.30
MEAN	0.345	0.440

Table 6.11: Comparing the new collaborative filtering baseline that incorporates friendship and social tagging data with the top performing recommendation system on the validation set using Precision@20.

6.7 New CF Baseline

Since the FriendRec system outperformed the two original baselines, I wanted to compare my recommendation system against a CF system that also incorporated social tagging information. The presence of social tagging information in a cold-start environment is not an implausible situation for a recommendation system. We can imagine this as the case where the new user has created a profile linked to a social networking account in order to gain access to the user's social network. The platform then asks the user to assign tags to their favourite artists before they may use this product. This is the use case for my new set of experiments.

Experiments were performed on the validation set, which can be found in Table 6.1, without rating information. This was done by setting the value of α to zero while the other weights remained the same as the optimal settings $\beta = 0.15$ and $\gamma = 0.05$ which were found in section 6.4. The results from using both social tagging and friendship data resulted in a slightly higher Precision@20 when compared to the original CF baseline.

6.8 Changing Recommendation Systems

In a cold-start situation, the proposed FriendRec system outperformed the CCF system. However during testing for the optimal values used in the CCF system I noticed that when supplied with 25 out of the 50 ratings, this system surpassed the accuracy achieved with the FriendRec system. This confirmed the assumption that provided sufficient information about the user, this system would be preferable. Further experiments were then created to decide how many positive ratings were required before switching from the FriendRec system to the CCF system. This swap in recommendation systems would signify the end of the cold-start phase.

The experimental setup is as follows; I generated the initial recommendation list by

USER ID	NO. VALID RATINGS
6	0
40	2
133	6
332	1
491	4
925	0
1084	1
1136	2
1301	6
1581	2
MODE	2

Table 6.12: The number of valid ratings required for each user in order to produce more accurate recommendations from the combined collaborative filtering system than the Friend Recommendation system.

using the FriendRec system. I then select the highest valid recommendations in the list, this corresponds to the artist that the recommendation system is most confident about. This artist's rating information is then added to the CCF system and removed from the FriendRec system. I then generate two recommendation lists, one for each of the new systems. If for a given user, the FriendRec system produces a list with a higher Precision@20 than the CCF system in conjunction with the Precision@20 being greater than zero, then this process is repeated until one of these clauses is false for that user. Once a clause is found to be false I stopped generating lists for the user. At this point the FriendRec system is no longer the better alternative.

The results from this experiment can be found in Table 6.12. In this table we can see that the majority of users require no more than two valid recommendations in order for the CCF system to overtake the FriendRec system. Since the most common number of valid recommendations is two, I suggest switching away from the FriendRec system after the user has given a positive rating for two items. The classification of a positive rating may vary between platforms, for example an explicit rating above a chosen number may indicate a positive rating or in my case a listen count above a certain threshold. For this system the cold-start phase ended at two positive ratings.

6.9 Attack Resistant Recommendation Systems

In order to test the resistance of all three recommendation systems from shilling attacks, I decided to create a Segment Attack. This attack is achieved by creating bots similar to segment market users in order to push a particular item within a relevant community. In order to increase the effect of the attack I decided to insert 50 malicious profiles matching the rating and tagging information for every user in the system that has an impact on generating recommendations for the target user. Each of these malicious profiles rated a fake artist, using the ID zero as it was not linked to an artist, and assigning a listen count equal to double the maximum listen count currently in the system. If the fake artist ID appears in the recommendation list, then the attack

USER ID	POPULAR	FRIEND BASELINE	NEW BASELINE	SNI $p = 0.2$	FRIEND
3	0.00	0.00	0.00	0.00	0.05
12	0.25	0.30	0.20	0.40	0.35
128	0.05	0.25	0.25	0.05	0.30
478	0.05	0.20	0.20	0.05	0.45
582	0.00	0.30	0.35	0.00	0.35
912	0.00	0.35	0.40	0.05	0.55
1278	0.55	0.35	0.50	0.05	0.50
1375	0.00	0.05	0.05	0.05	0.15
1458	0.45	0.15	0.60	0.15	0.10
1509	0.60	0.30	0.55	0.15	0.60
MEAN	0.195	0.225	0.310	0.095	**0.340**

Table 6.13: The test set's Precision@20 using: the impersonalised popular artist (Popular) recommendation system, the Social Network's Influencer (SNI) system, the Combined Collaborative Filtering (CCF) system and the Friend Recommendation (Friend) system.

has been successful. This attack was applied to all of the recommendation systems, the only two recommendation systems that were not affected by the malicious profiles were the FriendRec system and the baseline known as the friendship CF system, a.k.a the gamma-only experiment. The popular artists baseline, the improved CF baseline, the SNI system and the full CCF system with rating information, all fell victim to this segment attack with every user in the validation set being recommended artist zero as their most relevant recommendation.

6.10 Recommendation System Comparison

Throughout this chapter I have analysed a variety of different recommendation systems, tuning these systems' hyperparameters using the validation set and also providing an initial insight into which recommendation system may generate more accurate recommendations in a cold-start situation. However the only way to effectively compare these systems is through the use of the test set, which can be found in Table 6.2. In this section I will show the results gained from using this unseen data as well as delve deeper into analysing the recommendation systems' performance using not only the Precision@20 but also the secondary metrics discussed in Chapter 3.

6.10.1 Accuracy

Throughout this dissertation I have used Precision@20 as the score for accuracy and therefore I will continue to do so in order to get a general impression of the system's performance. Precision@K is a valid metric commonly used to measure the accuracy of recommendation systems [51, 59]. I have also added the total hit count of each system into Table 6.14 along with the Precision@20. This number is the total number of valid recommendations supplied to the users in the test set and can be divided by 200 to get the Precision@20.

System	Total Hits	Precision@20	Time	Serendipity	Diversity
Friend	68	0.34	0.03 ± 0.022	42	96
CCF	62	0.31	338.01 ± 137.4	36	102
SNI	19	0.095	870.17 ± 59.9	13	140

Table 6.14: Analysing the Combined Collaborative Filtering, the Social Network's Influencers and the Friend Recommendation systems' performance on the test set using a variety of metrics.

6.10.2 Serendipity

I have decided to use the popular artists recommendation system as the serendipitous recommendation system. I calculated serendipity as the number of valid recommendations in a recommendation list that is not in the popular artists recommendation list.

6.10.3 Diversity

Since no genres were assigned to each artist, diversity had to be calculated in a suboptimal manner. By finding the top tag assigned to each artist in the recommendation list for a user, I was able to group artists using these tags. Each group represented what would typically be a genre, with the number of different genres in the recommendation list representing the diversity score for that user.

Chapter 7

Evaluation

7.1 Overview

In this chapter I will analyse the three recommendation systems' results stated in the Experimental Results chapter. I will begin by discussing the CCF weight selection process, followed by why the SNI system failed to produce valid recommendations and finally a brief comment on the FriendRec system which was the top performing system in the cold-start scenario. After I will do a more in depth comparison of the CCF system and the FriendRec system using the primary and secondary metrics calculated in Section 6.10.

7.1.1 Combined Collaborative Filtering Weights

The alpha-only CCF experiment achieved the highest Precision@20 when selecting the weights to be used in this system. This was to be expected as the top-20 users selected to generate the final recommendation list were chosen solely based on them having artist ratings similar to the target user. Since the selected users had rated some, if not all, of the 25 artists in the target user's rating dataset these rated artists will most likely appear in the recommendation list generated for the target user. In sum, finding users that have similar artist ratings to the target user will result in a very accurate system that will increase the users' trust. However the majority of recommendations will be pointless as there will be very little novelty in the system's recommendations. This means the user is unlikely to discover new artists or explore and refine their music preferences, which is also a key feature of many recommendation systems.

Despite the alpha-only experiment gaining the highest hit count, its weights could not be used for future experiments. The alpha-only system requires rating data to generate recommendations and this information is not available to the system in the cold-start scenario. Since certain cold-start profiles may contain only tagging or friendship information, testing was focused on finding the best CCF system which incorporated all three similarity scores. This was done to accommodate both types of users and provide tailored recommendations.

I was originally surprised that the incorporation of tagging and friendship information

did not improve the Precision@20 of the CCF system. However I quickly realised the potential detrimental factors associated with using tagging or friendship information in CF systems and how these methods can incorporate noise into such systems. Firstly when incorporating tagging information it is important to understand that similar tags do not equate to a similar taste in music. The assumption that people who have taken the time to tag an artist have an interest in that artist is not always true as the users' purpose for tagging is often unclear [22]. Systems that use this assumption can often suffer from noise as two users are not required to have the same music preference in order to assign the same tag to an artist.

Secondly, friendship is a strong indicator of music preference, however, people who have similar friends to the target user may not share their taste in music. Although users have similar friends to the target user, they are not necessarily the target user's friends themselves. Even if they are direct friends, not all friends listen to the same music and due to the absence of a trust value between users in the system it is hard to determine if the target user trusts their recommendations. These factors can also add noise to the CCF system when generating recommendations.

Nevertheless I was able to find two experiments whose results are not far from those of the alpha-only Precision@20 of 0.495. Experiment β-Seven and β-Eight both gained Precision@20 of 0.480 just off the Precision@20 boasted by the alpha-only experiment. Satisfied with the accuracy of both these experiments, I had to select which weights setup to use going forward. I chose β-Eight as they increased the impact of the rating data while still incorporating tagging and friendship data.

7.1.2 SNI

The SNI system significantly fell short in all departments when analysing it with the various metrics. This system is considered to be ineffective when tackling the cold-start problem since the average time to generate recommendations for the target user was 870.17 seconds in conjunction with the low Precision@20. Usually, a higher diversity is a desired quality, however in this system's recommendation list it simply signifies the fact that the system was unable to learn the unique preferences of each user. The low Precision@20 and high diversity can be attributed to the fact that for each user their recommendation lists contains the same artists when $p = 0.2$. This is because the largest three influencers in each target user's social network is the same for all users given this restart probability; these users are likely the most influential users in the whole network. Since all recommendation lists are the same, the system can be considered an impersonalised recommendation system.

It became obvious that the influencers do not possess the discriminative information necessary to generate accurate recommendations for the target user. A possible improvement would be to use the influencers' rating data instead of their recommendations as this is a more accurate representation of the influencers' favourite artists.

7.1.3 FriendRec

The FriendRec system outperformed all systems using the validation set as well as when generalising to the test set. This system produced the highest accuracy of all the systems and with the relatively short period of time required to generate recommendations, this system appears to be the favourite. When analysing the different options provided by the recommendation system, the direct friends approach produced slightly better results than the friends of friends approach. However this was not the case for all users as six out of the ten users in the validation set had an equal or greater Precision@20 when it came to the friends of friends recommendation system. This means that given a potentially larger or different set of users the friends of friends recommendation system may prove to be the better alternative. Further experimentation into the optimal weights for the target user's friends and friends of friends could also result in significant improvements. However with no clear benefit in altering these values I decided to opt for the direct friend approach as it was computationally less expensive and provided the higher Precision@20 for the validation set.

7.1.4 Changing Recommendation System Experiment

There are three users which required significantly more positive ratings in order for the CCF system to outperform the FriendRec system. Nevertheless changing recommendation systems after two positive ratings would not significantly impact the quality of recommendations provided to these users. This is due to the CCF system producing 10 or more valid recommendations for these users given two positive ratings.

7.2 Comparing Recommendation Systems

In this section I will provide an in depth comparison of the FriendRec system and the improved CCF baseline system that incorporates both friendship and tagging information. The SNI system will not be involved in the comparison as it did not beat the original two benchmarks nor the improved benchmark.

7.2.1 Accuracy

Accuracy is the primary metric used to evaluate recommendation systems in this dissertation. Accuracy in this dissertation was measured using Precision@20 as the recommendation lists produced had a fixed length of 20.

The FriendRec system is slightly more accurate than the CCF system making it a more desirable recommendation system. However the margins are very slight with a difference in precision of 0.03. Therefore other attributes were required in order to conclusively determine the preferred recommendation system when tackling the cold-start problem.

7.2.2 Scalability

Psychologists believe that people have a cognitive bias which can lead them to misinterpret future information to support their hypotheses [54]. This confirmation bias is when a person uses seemingly ambiguous information to back up their opinions even if their beliefs are misinformed. Therefore if a user's first impressions of a recommendation system are that it is not accurate or impersonalised, they may be unlikely to trust that its recommendations are tailored to their specific tastes in the future. This belief may persist despite the generation of accurate recommendations and could result in the user changing to a competitor's platform.

In order for a recommendation system to be viable in a cold-start scenario it must be able to produce recommendations for a given user in an extremely short amount of time. Table 6.14 shows that in less than a second the FriendRec system is able to generate relatively accurate recommendations. The CCF system, on the other hand, requires 5 minutes and 38 seconds to generate recommendations for the target user; the user must be distracted for this period of time. As the common phrase goes "time is money", therefore the friend recommendation system would provide a clear financial advantage. It would be a waste to dedicate resources to performing CF on insufficient information, which generates invalid recommendations.

However as discussed in section 6.8, the FriendRec system should only be used until there is enough information for the CCF system to provide more accurate recommendations. By using the FriendRec system for the initial recommendations we give the platform time to generate valid recommendations without the need to either waste the user's time or provide impersonalised recommendations.

7.2.3 Serendipity

First impressions count and it is therefore desirable that the recommendation systems produce accurate and tailored recommendations from the start in a timely manner. Table 6.14 shows that the FriendRec system provides the highest Precision@20 of all the systems while also boasting the highest serendipity score. The serendipity score, like the majority of the secondary metrics, is not a true representation however it is a close approximate. Since there is no user rating information it is unknown if the user is already aware of these recommendations and if these recommendations would be considered surprising. As the serendipitous recommendation system is the popular artists recommendation system, I instead consider the serendipity score to be a measure of personalisation showing the degree to which the system understands the user's preferences.

Increased serendipity in a recommendation system can have a detrimental affect on accuracy but in the long run will usually provide better results [36]. A balance between accuracy and serendipity is therefore important, the goal is to increase serendipity while keeping accuracy as high as possible. In my experiments the FriendRec system had not only higher accuracy but also a higher serendipity score. The serendipitous recommendation system is supposed to be a dumb system [13], therefore I decided to make this system the popular artists recommendation system as it assumes everyone

will like the most popular artists. Therefore in my experiments the serendipity score in conjunction with the Precision@20 gives a good picture of the degree to which the recommendation system personalised their recommendations. This shows that the FriendRec system had a better understanding of the target users' music preference.

7.2.4 Diversity

Diversity is useful in recommendations as having a larger diversity means there is a higher chance that the user will be interested in at least one of the recommendations. In these experiments the CCF system produced a higher diversity score. With a relatively high hit count this is a desirable characteristic of the recommendation system.

7.2.5 Robustness and Stability

The only two recommendation systems not susceptible to attacks are the FriendRec System and the original CCF baseline known as the friendship CF system. This is due to the fact that they generate their rating data based solely on friendship information. It is therefore impossible for the malicious profiles inserted into the recommendation system to affect the target user's recommendation list without becoming friends with them or their direct friends. Since the malicious profiles are created to match the average user in the target user's segment, they will likely be selected as part of the K-nearest neighbours during CF and can therefore influence the recommendation list generated. All of the systems that implement CF with either rating or tagging information fell victim to the segment attack.

Referring back to the importance of first impressions of a system on a user, if the first recommendation list is filled with artists generated by bots in a segment attack, the user is unlikely to continue using the system for a variety of reasons. In a worst case scenario, controversial recommendations could result in users' immediate termination of their account. In addition, if the user continues to use the system the recommendations given to a user may depend solely on the malicious profiles. These likely poor, impersonalised, and in some cases unchanging, recommendations may deter the user from continued loyalty to the platform.

Chapter 8

Conclusion

To conclude when tackling the user cold-start problem the FriendRec system was the most accurate attack-resistant system being able to generate recommendations in less than a second. Despite the CCF system, which used social tagging and friendship data, providing comparable results in accuracy, serendipity and diversity the nail in the coffin for this system was the time to generate recommendations. The average time for the CCF system was 11,267 times greater than that of the FriendRec system. Due to the time restrictions of the user cold-start scenario and limited information available to the system this computational cost appeared unnecessary. There are methods in which CF can become attack-resistant as well as effective when handling the user cold-start problem in a data sparse scenario. However, to tackle these problems results in extra computation on top of the already excessive time required to generate recommendations. As the user does not expect the system to predict their preferences exactly from the start the more complex CF method would be an unnecessary burden on the system.

At the beginning of this dissertation I hypothesised that social information is the key to providing accurate recommendations during a user's cold-start as well as being attack-resistant. The SNI system was created based on the concept of social influence [66], which states that connected users are more likely to have similar interests. This premise did not prove robust when recommending artists to the cold-start user with this system. Homophily, on the other hand, is the idea that users form friendships with people who have similar preferences [44]. This was the assumption on which FriendRec was grounded. As shown, the concept of homophily proved an effective solution when tackling the user cold-start problem. The experiments conducted and the performance of the FriendRec system support my hypothesis.

The FriendRec system however is not meant to act as a fully functioning recommendation system in its own right. Rather it is an appendage to other more discriminative systems which require more time and information to provide valid recommendations to the user. As a complement to these systems, the FriendRec system allows platforms to be both accurate and time efficient upon a cold-start user's first visit.

8.1 Future Works

The validity of the experiments could have been improved by selecting a larger split for the validation and testing set. This would give more representative information about the dataset as a whole, instead of the small subset selected. Another drawback of this approach was that only users with more than three friends and at least 50 rated artists were tested. This is a strong assumption, and might bias the value of the system towards heavy users. As a subset these users' recommendations may be not be representative of the general user in the system. The recommendation system may only provide valid recommendations for social users who are likely to frequently use the system therefore using a wider range of individuals could also lead to more reliable results.

Attack resistant CF systems exist which perform malicious user detection using dimensionality reduction before CF. Due to the time taken to detect malicious users and then perform CF I would assume this method would have too high a time complexity to be efficient in a cold-start scenario. However this would be an interesting area for future research. Another parameter to adjust during future experimentation would be the size of the top K users selected to generate the recommendation list as well as the size of the list.

The simplicity of the FriendRec system may be defeated when incorporating more user data such as demographic information or trust between users into the system. Therefore further experiments should be run using a larger dataset with access to these forms of information. I began to create a FriendRec system which uses the tagging information in order to assign a higher weight to similar friendships. Given extra information this could lead to dramatic improvements when generating recommendations.

Bibliography

[1] Google developer recommendation system course. https://developers.google.com/machine-learning/recommendation/content-based/summary. Accessed:14-04-2020.

[2] Lexico dictionary. https://www.lexico.com/en/definition/serendipity. Accessed:14-04-2020.

[3] Pandas dataframe documentation. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html. Accessed:14-04-2020.

[4] Pandas series documentation. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html. Accessed:14-04-2020.

[5] Pypi documentation for progress. https://pypi.org/project/progress/. Accessed:14-04-2020.

[6] Python dictionary documentation. https://docs.python.org/3/tutorial/datastructures.html#dictionaries. Accessed:14-04-2020.

[7] Python documentation for itertools. https://docs.python.org/3/library/itertools.html#itertools.islice. Accessed:14-04-2020.

[8] Scipy compressed sparse row documentation. https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html. Accessed:14-04-2020.

[9] Scipy pearsonr documentation. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html. Accessed:14-04-2020.

[10] Scipy sparse data structure documentation. https://docs.scipy.org/doc/scipy/reference/sparse.html. Accessed:14-04-2020.

[11] Spotify company info. https://newsroom.spotify.com/company-info/?fbclid=IwAR0MygOZCDnDzDMnOR3ah_azzmrW3fDrQiPAgmzg8m7VD7NZH1KmNOGD-vI. Accessed:14-04-2020.

[12] *HetRec '11: Proceedings of the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, New York, NY, USA, 2011. Association for Computing Machinery.

[13] Charu C. Aggarwal. *Recommender Systems: The Textbook*. Springer Publishing Company, Incorporated, 1st edition, 2016.

[14] Haldun Akoglu. User's guide to correlation coefficients. *Turkish Journal of Emergency Medicine*, 18, 08 2018.

[15] Joeran Beel, Stefan Langer, Marcel Genzmehr, Bela Gipp, Corinna Breitinger, and Andreas Nürnberger. Research paper recommender system evaluation: A quantitative literature survey. In *Proceedings of the International Workshop on Reproducibility and Replication in Recommender Systems Evaluation*, RepSys '13, page 15–22, New York, NY, USA, 2013. Association for Computing Machinery.

[16] David Ben-Shimon, Alexander Tsikinovsky, Lior Rokach, Amnon Meisels, Guy Shani, and Lihi Naamani. Recommender system from personal social networks. volume 43, pages 47–55, 01 2007.

[17] Elena Boldyreva. Cambridge analytica: Ethics and online manipulation with decision-making process. pages 91–102, 12 2018.

[18] Christopher Burr, Nello Cristianini, and James Ladyman. An analysis of the interaction between intelligent software agents and human users. *Minds and Machines*, 28(4):735–774, 12 2018.

[19] Pablo Castells, Saúl Vargas, and Jun Wang. Novelty and diversity metrics for recommender systems: Choice, discovery and relevance. *Proceedings of International Workshop on Diversity in Document Retrieval (DDR)*, 01 2011.

[20] Yibo Chen, Chanle Wu, Ming Xie, and Xiaojun Guo. Solving the sparsity problem in recommender systems using association retrieval. *JCP*, 6:1896–1902, 08 2011.

[21] Paul-Alexandru Chirita, Wolfgang Nejdl, and Cristian Zamfir. Preventing shilling attacks in online recommender systems. In *Proceedings of the 7th Annual ACM International Workshop on Web Information and Data Management*, WIDM '05, page 67–74, New York, NY, USA, 2005. Association for Computing Machinery.

[22] Frederico Durao and Peter Dolog. A personalized tag-based recommendation in social web systems. *CEUR Workshop Proceedings*, 485, 03 2012.

[23] Daniel M. Fleder and Kartik Hosanagar. Recommender systems and their impact on sales diversity. In *Proceedings of the 8th ACM Conference on Electronic Commerce*, EC '07, page 192–199, New York, NY, USA, 2007. Association for Computing Machinery.

[24] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. Beyond accuracy: Evaluating recommender systems by coverage and serendipity. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, page 257–260, New York, NY, USA, 2010. Association for Computing Machinery.

[25] Alexandre Gilotte, Clément Calauzènes, Thomas Nedelec, Alexandre Abraham, and Simon Dollé. Offline a/b testing for recommender systems. In *Proceedings*

of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM '18, page 198–206, New York, NY, USA, 2018. Association for Computing Machinery.

[26] Carlos A. Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Inf. Syst.*, 6(4), December 2016.

[27] Guibing Guo. Improving the performance of recommender systems by alleviating the data sparsity and cold start problems. pages 3217–3218, 08 2013.

[28] Guibing Guo, Jie Zhang, and Daniel Thalmann. A simple but effective method to incorporate trusted neighbors in recommender systems. In *Proceedings of the 20th International Conference on User Modeling, Adaptation, and Personalization*, UMAP'12, page 114–125, Berlin, Heidelberg, 2012. Springer-Verlag.

[29] Ido Guy and David Carmel. Social recommender systems. pages 283–284, 01 2011.

[30] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272, Dec 2008.

[31] Neil Hurley and Mi Zhang. Novelty and diversity in top-n recommendation – analysis and evaluation. *ACM Trans. Internet Technol.*, 10(4), March 2011.

[32] F.O. Isinkaye, Y.O. Folajimi, and B.A. Ojokoh. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, 16(3):261 – 273, 2015.

[33] Gawesh Jawaheer, Martin Szomszor, and Patty Kostkova. Comparison of implicit and explicit feedback from an online music recommendation service. In *Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, HetRec '10, page 47–51, New York, NY, USA, 2010. Association for Computing Machinery.

[34] Shah Khusro, Zafar Ali, and Irfan Ullah. *Recommender Systems: Issues, Challenges, and Research Opportunities*, pages 1179–1189. 02 2016.

[35] Ioannis Konstas, Vassilios Stathopoulos, and Joemon M. Jose. On social networks and collaborative recommendation. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, page 195–202, New York, NY, USA, 2009. Association for Computing Machinery.

[36] Denis Kotkov, Shuaiqiang Wang, and Jari Veijalainen. A survey of serendipity in recommender systems. *Know.-Based Syst.*, 111(C):180–192, November 2016.

[37] Shyong K. Lam and John Riedl. Shilling recommender systems for fun and profit. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, page 393–402, New York, NY, USA, 2004. Association for Computing Machinery.

[38] Neal Lathia, Stephen Hailes, and Licia Capra. Trust-based collaborative filtering. In Yücel Karabulut, John Mitchell, Peter Herrmann, and Christian Damsgaard Jensen, editors, *Trust Management II*, pages 119–134, Boston, MA, 2008. Springer US.

[39] Joonseok Lee, Mingxuan Sun, and Guy Lebanon. A comparative study of collaborative filtering algorithms, 2012.

[40] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[41] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. *Content-based Recommender Systems: State of the Art and Trends*, page 73. 2011.

[42] Andrii Maksai, Florent Garcin, and Boi Faltings. Predicting online performance of news recommender systems through richer evaluation metrics. In *Proceedings of the 9th ACM Conference on Recommender Systems*, RecSys '15, page 179–186, New York, NY, USA, 2015. Association for Computing Machinery.

[43] Paolo Massa and Paolo Avesani. Trust-aware recommender systems. In *Proceedings of the 2007 ACM Conference on Recommender Systems*, RecSys '07, page 17–24, New York, NY, USA, 2007. Association for Computing Machinery.

[44] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 27(1):415–444, 2001.

[45] Bhaskar Mehta and Wolfgang Nejdl. Attack resistant collaborative filtering. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, page 75–82, New York, NY, USA, 2008. Association for Computing Machinery.

[46] Silvia Milano, Mariarosaria Taddeo, and Luciano Floridi. Recommender systems and their ethical challenges, 04 2019.

[47] Bamshad Mobasher, Robin Burke, Chad Williams, and Runa Bhaumik. Analysis and detection of segment-focused attacks against collaborative recommendation. In *Proceedings of the 7th International Conference on Knowledge Discovery on the Web: Advances in Web Mining and Web Usage Analysis*, WebKDD'05, page 96–118, Berlin, Heidelberg, 2005. Springer-Verlag.

[48] Mavuto Mukaka. Statistics corner: A guide to appropriate use of correlation coefficient in medical research. *Malawi medical journal : the journal of Medical Association of Malawi*, 24:69–71, 09 2012.

[49] Douglas Oard and Jinmook Kim. Implicit feedback for recommender systems. In *in Proceedings of the AAAI Workshop on Recommender Systems*, pages 81–83, 1998.

[50] Manos Papagelis, Dimitris Plexousakis, and Themistoklis Kutsuras. Alleviating the sparsity problem of collaborative filtering using trust inferences. In *Proceed-*

ings of the Third International Conference on Trust Management, iTrust'05, page 224–239, Berlin, Heidelberg, 2005. Springer-Verlag.

[51] Nikolaos Polatidis and Christos Georgiadis. A multi-level collaborative filtering method that improves recommendations. *Expert Systems with Applications*, 48, 12 2015.

[52] Alexandrin Popescul, David M. Pennock, and Steve Lawrence. Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, page 437–444, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[53] Alan Prando, Felipe Contratres, Solange N A Souza, and Luiz Souza. Content-based recommender system using social networks for cold-start users. pages 181–189, 01 2017.

[54] Matthew Rabin and Joel L. Schrag. First impressions matter: A model of confirmatory bias. 1997.

[55] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Recommender Systems Handbook*, volume 1-35, pages 1–35. 10 2010.

[56] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, page 285–295, New York, NY, USA, 2001. Association for Computing Machinery.

[57] J. Ben Schafer, Joseph Konstan, and John Riedl. Recommender systems in e-commerce. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, page 158–166, New York, NY, USA, 1999. Association for Computing Machinery.

[58] J. Ben Schafer, Joseph A. Konstan, and John Riedl. E-commerce recommendation applications. *Data Min. Knowl. Discov.*, 5(1–2):115–153, January 2001.

[59] Gunnar Schröder, Maik Thiele, and Wolfgang Lehner. Setting goals and choosing metrics for recommender system evaluations. 811, 01 2011.

[60] Suvash Sedhain, Scott Sanner, Darius Braziunas, Lexing Xie, and Jordan Christensen. Social collaborative filtering for cold-start recommendations. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, page 345–348, New York, NY, USA, 2014. Association for Computing Machinery.

[61] Guy Shani and Asela Gunawardana. *Evaluating Recommendation Systems*, volume 12, pages 257–297. 01 2011.

[62] Thiago Silveira, Min Zhang, Xiao Lin, Yiqun Liu, and Shaoping Ma. How good your recommender system is? a survey on evaluations in recommendation. *International Journal of Machine Learning and Cybernetics*, 10:813–831, 2019.

[63] Xiaoyuan Su and Taghi Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. Artificial Intellegence*, 2009, 10 2009.

[64] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009, January 2009.

[65] Zhoubao Sun, Lixin Han, Wenliang Huang, Xueting Wang, Xiaoqin Zeng, Min Wang, and Hong Yan. Recommender systems based on social networks. *Journal of Systems and Software*, 99:109 – 119, 2015.

[66] Jiliang Tang, Xia Hu, and Huan Liu. Social recommendation: a review. 2013.

[67] Tong, Hanghang, Faloutsos, Christos, Pan, and Jia-Yu Pan. Fast random walk with restart and its applications. 12 2006.

[68] Maksims Volkovs, Guangwei Yu, and Tomi Poutanen. Dropoutnet: Addressing cold start in recommender systems. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4957–4966. Curran Associates, Inc., 2017.

[69] L. Wu, S. Shah, S. Choi, Mitul Tiwari, and Christian Posse. The browsemaps: Collaborative filtering at linkedin. *CEUR Workshop Proceedings*, 1271, 01 2014.

[70] Kai Yu, Anton Schwaighofer, Volker Tresp, Xiaowei Xu, and Hans-Peter Kriegel. Probabilistic memory-based collaborative filtering. *IEEE Trans. on Knowl. and Data Eng.*, 16(1):56–69, January 2004.