A First Principles Approach to Trust-Based Recommendation Systems

Paras Stefanopoulos (u7300546)

November 2023

Research Report
Course Code: COMP3770
Supervisor: Dr. Ahad N. Zehmakan

Abstract. This paper is an exploration of trust-based recommender systems, dissecting various algorithms and evaluating their effectiveness and robustness across varying contexts, including adversarial environments. By investigating different information types in social networks, such as item-rating and intra-item information, and the contexts in which they are effective, we gain an understanding of how these distinct information forms combine to be constructive in preference prediction. Item-rating information was identified as a pivotal feature in determining similarity between users, significantly outperforming other information types in a collaborative filtering approach. Despite its lower mean absolute error (MAE) performance, the trust graph was revealed to formulate recommenders that are notably more robust to network adversarial attacks, as they leverage the hard-to-manipulate trust structures embedded in the network. Intra-item information, although sub-optimal in isolation, enhanced the consistency of predictions and improved lower-end performance when fused with other information forms. The paper introduces the "Weighted Average" (WA) framework, a versatile and demonstrably effective approach that enables the construction of recommendation systems around any user-to-user similarity metric. Our apex recommender, the Jaccard Item Jaccard WA, amalgamates trust-based and item-based Jaccard indexes to generate user similarities, achieving a mean MAE of 0.72 across validation data sets, and outperforming our other recommenders in all test cases. Furthermore, we navigate through the challenges of the cold start problem and propose further research into the resilience of recommender systems against various adversarial strategies, potential improvements to popular models in literature and recognizing the potential for numerous computational optimizations and enhancements.

Contents

1 Introduction

1.1 Background

Recommendation systems have become an essential component of online platforms, helping users navigate vast amounts of information and make decisions based on personalized suggestions. In social networks, these systems play a role in suggesting friends, pages, products, or content to users based on their preferences, behaviours, and social connections. Social networks provide a rich source of information on user interactions and preferences, which can be leveraged to create more accurate and personalized recommendations. This type of recommendation system not only considers the past behaviours and preferences of users but also the behaviours and preferences of their connections or friends in the network.

Trust is a crucial concept in social networks and plays a vital role in recommendation systems [15]. The abstraction of social connections as 'trust' is essential because it allows the system to consider implicit behaviours and preferences of users and their connections. For example, a user liking or sharing content, or having many mutual connections, can indicate a level of trust or shared interest, even if the users are not explicitly connected or have not directly interacted. This abstraction helps in capturing the nuances of social interactions and enables the recommendation system to provide more personalized and accurate suggestions. Additionally, the concept of trust incorporates a user's social circle's opinions and preferences, making recommendations more robust and comprehensive [17]. In this paper, we investigate trust connections, and the implications they have on a user's preferences and the robustness to adversarial attacks.

There are several types of recommendation systems, including content-based, collaborative filtering, hybrid, and others. Content-based recommendation systems suggest items based on the characteristics of the items and the preferences of the users. On the other hand, collaborative filtering recommends items based on the preferences and behaviours of similar users [18]. Hybrid systems combine both approaches. Trust-based collaborative filtering is particularly interesting because it not only considers the preferences of similar users but also incorporates the trust relationships between users. This approach helps in mitigating some of the limitations of traditional collaborative filtering, such as the cold start problem and data sparsity, by leveraging the trust network to generate recommendations even when there is limited interaction data available [19]. This is a technique we will explore in this paper, but with a first principes approach to understand how to best utilise these different methods and information forms.

One challenge in recommendation systems is the "cold start" problem. Which occurs when new users or items are added to the network and there is insufficient interaction data to generate accurate recommendations [4]. For new users, the system does not have enough information about their preferences, and for new items, there is a lack of user interaction data to determine their relevance to different users. This problem is particularly pronounced in trust-based recommendation systems, as new users or items may not have established trust relationships with existing entities in the network. As a result, the recommendation system struggles to provide meaningful recommendations for these new entities [12].

While the cold start problem is a significant challenge in trust-based recommendation systems, there are other challenges as well. For example, data sparsity is a common issue, as not all users rate or interact with many items. Additionally, the scalability of the recommendation system is another challenge, as the system needs to handle a large number of users and items while providing recommendations in real time. Finally, the presence of adversarial attacks, where malicious users try to manipulate the recommendation system, is a challenge that needs to be addressed to ensure the robustness of the recommendation system.

1.2 Related Work

Over the years, various approaches have been proposed to enhance the performance of trust-based recommendation systems. These approaches can be broadly categorized into the following groups:

1. **Traditional Approaches**

   Traditional trust-based recommendation systems mainly focus on incorporating explicit and implicit trust relationships into the recommendation process. For example, some approaches consider the ratings or preferences of a user's trusted friends when generating recommendations. Massa and Avesani [14] delved into a trust-aware collaborative filtering approach. Their method distinctively pivots on explicit trust relationships. This means that the system directly harnesses expressed trust values (like friendship connections) to modulate the influence of a user's peers on their recommendations.

   Other methods incorporate trust propagation mechanisms to infer trust relationships between users who are not directly connected. The idea is grounded in the notion that trust can be transitive to some degree [16]. For example, if Alice trusts Bob and Bob trusts Carol, even if Alice hasn't interacted with Carol, there might be a certain level of implicit trust Alice has for Carol due to the mutual connection. Numerous methods have been developed based on this trust transitivity principle. Some models utilize matrix factorization techniques to propagate trust across the user-item matrix, while others employ graph-based algorithms to trace the spread of trust across a network of users. Such propagation methods attempt to fill the gaps in the trust matrix, offering a more comprehensive and interconnected trust landscape that can be leveraged for more informed and nuanced recommendations [9].

2. **Cold-Start Approaches**

   Addressing the "cold start" problem remains a pivotal challenge for trust-based recommendation systems. The "cold start" issue surfaces when a new user joins the platform or a new item is added, and there's minimal to no interaction data available for meaningful recommendation generation. In such scenarios, traditional collaborative filtering methods often fall short, as they rely predominantly on historical interaction data. A trust network, in essence, is a web of relationships that capture which users trust which other users [10]. The assumption here is that even if a user hasn't interacted with many items, their trusted connections might have. By leveraging this network, systems can extrapolate the potential interests of the new user based on the preferences of their trusted peers.

   One particularly noteworthy approach to leverage the trust network against the "cold start" problem was conceived by Jamali and Ester [8] (the same authors as the matrix factorization paper!). They introduced TrustWalker, a random walk model grounded in the trust network. It commences its journey from the target user and propagates through their trusted connections, and with some probability, swapping to other similar items that may have more information. In this way, TrustWalker harnesses both explicit and implicit trust connections along with skipping to other items, which lets "cold start" users with little information still get reasonable results.

3. **Machine Learning Approaches**

   Machine learning, over the past decade, has revolutionized numerous domains with its ability to extract patterns from vast data sets. Traditional methods of collaborative filtering were primarily linear. However, the dynamics of user preferences, trust relationships, and item attributes often exhibit nonlinear patterns that require more sophisticated models for accurate representation. An area where ML approaches excel [13].

Neural collaborative filtering models emerged as a response to this. These models can capitalise on these non-linear patterns. One notable contribution in this domain is the paper titled "Neural Collaborative Filtering" by Xiangnan He et al. This research underscores the potential of deep neural networks in handling the collaborative filtering challenge on implicit feedback data sets. They propose three different ML-based models for developing recommendation systems and show the models compete with other state-of-the-art approaches [7].

In parallel, the application of graph neural networks (GNNs) in recommender systems has witnessed a significant surge, establishing GNNs as a state-of-the-art approach in this area. [3] by Chen Gao et al. offers a comprehensive review of the literature centred around GNN-based recommender systems. The motivation to incorporate GNNs into recommender systems primarily stems from their ability to capture the higher-order structure of the graph. The survey delves into the intricate challenges associated with graph construction, embedding propagation/aggregation, model optimization, and computation efficiency.

4. **Adversarial Defense Approaches**

Recommendation systems are vulnerable to adversarial attacks subverting their performance. These adversarial attacks can manifest in various forms: fake user profiles, strategically crafted reviews, or even swarming behaviours intended to boost or suppress particular items. A robust recommendation system isn't just about accurate recommendations; it's also about ensuring its resilience against these adversarial intrusions [2].

Recent research has been addressing this challenge. For instance, [1] undertook an exhaustive survey on the robustness of deep-learning-based recommenders, particularly in the context of social networks. Their findings illuminated the susceptibility of even the most advanced models to certain adversarial tactics. However, the silver lining is the evolution of defence mechanisms. Techniques such as adversarial training, where the model is trained with perturbed data, have shown promise in enhancing model robustness.

Another study focused on how to accurately portray the representative rating for an item to mitigate the risk of an adversary adding fake ratings. The paper found that placed weighting on network-related factors leads to increased robustness to adversaries [11]. Defensive techniques are a particular idea we will further investigate in this report and an area we think we can provide insight into.

1.3 Our Contribution and Motivation

Recommendation systems play a crucial role in the online experience of users, influencing their decisions on what products to buy, which movies to watch, or whom to connect with. As we encounter information overload online, the importance of providing accurate and personalized recommendations cannot be overstated. Efficient recommendation systems not only help users make better decisions but also increase user engagement and satisfaction, which are critical for the success of online platforms.

The presence of adversaries in social networks poses a significant challenge to the effectiveness of recommendation systems. Adversaries can manipulate the recommendation system by creating fake profiles, generating fraudulent reviews, or establishing deceptive trust relationships. Such manipulations can lead to inaccurate recommendations, diminished user trust in the platform, and ultimately, decreased user engagement [1]. Imagine an adversary wants their product to be maximally recommended to users without paying for advertising, this could be achieved by exploiting a non-robust recommendation system. Moreover, the "cold start" problem, where new users or items in the network have insufficient interaction data to generate accurate recommendations, further complicates the recommendation process.

Therefore, it is essential to develop robust recommendation systems that can resist adversarial attacks, address the "cold start" problem, and provide accurate recommendations despite these challenges. This study aims to approach the challenges and opportunities associated with trust-based recommendation systems in social networks by building up from first principles. We are motivated by the potential of trust-based recommendation systems to enhance the online experience of users by providing more accurate and personalized recommendations. Furthermore, by addressing the challenges associated with the sparsity and dynamics of trust relationships, the presence of adversaries, and the "cold start" problem, we hope to contribute to the ongoing development of more robust systems.

As mentioned, the goal of this research is to take a first principles approach to building a recommendation system. We will build up recommendation systems that compete with current methods and understand how different parts of the data contribute to producing a recommendation. By doing this we will also be able to dive deeper into how different information in the graph contributes to the robustness of recommendation systems to adversaries.

We will develop novel approaches to combining different data forms in recommendation systems, such as a simple, but effective approach for determining item-similarity and an adaptable framework for designing recommenders based on user similarities.

We hope that by understanding exactly what contributes to the accuracy of these systems under different circumstances and in different contexts we could provide a basis for further improvements in the field. We also identify some key aspects that should be further researched and simple extensions on previous STOA models that may prove of value.

2 Methodology

2.1 Common Equations/Approaches

Jaccard Index

The Jaccard Index is a computable statistic that is used to determine the similarity between two sets. It is used throughout this report in different formats, sometimes to determine the similarity between users, and other times to determine the similarity between items.

Let S_1, S_2 be two sets, then we have:

$$Jaccard(S_1, S_2) = \frac{|S_1 \cup S_2|}{|S_2 \cap S_2|}$$

Weighted Average

The weighted average framework for designing recommendation systems appears repeatedly in this report. This approach came about from experimentation and proved to be a reliable and useful tool throughout the project. Hence, many of the recommenders are based on this.

Let us imagine we want to predict a rating for item i, for user u_1. We have computed a similarity metric for all pairs of users. Hence, we know how similar u_1 is to every other user.

- We set up 5 "buckets". Each bucket represents a rating, i.e. bucket 1 $rightarrow$ rating 1, bucket 2 $rightarrow$ rating 2, etc.
- We then iterate through all *other* users in the graph and compute their similarity with u_1. Let us say u_1 as a similarity of k with u_j.
- If u_j has rated item i, we contribute k to the corresponding rating. I.e. if u_j rates the item a 4, bucket 4 is incremented by k.
- We repeat this for all users. At the end, each bucket has a score. We take the weighted average of these buckets with respect to their score as our final prediction.

The pseudo-code for this algorithm can be found in the appendix section.

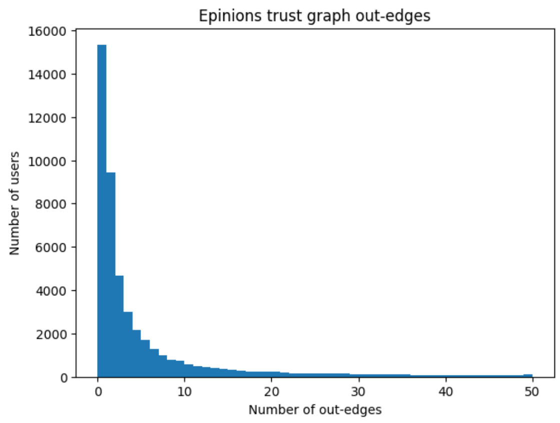
2.2 Training Data Set

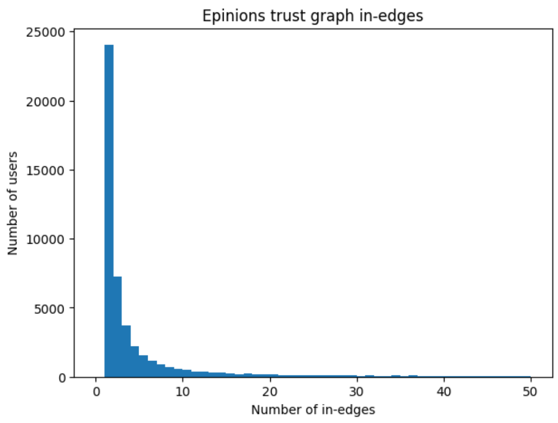
As previously mentioned, the approach taken in this research was to break down the problem into its first principles. Understand the data available, and build an optimal recommendation system from the ground up. For the sake of iteration in our experiments, we used a single data set for developing the algorithms and then validated the results with others.

In identifying a data set for this project, it was noted that the Epinions data set commonly appeared in related papers. Here we do a preliminary investigation into the Epinions data set to understand its structure and how items are rated.

Epinions was a consumer review site, where users could place reviews on items, and "trust" other users. User A trusting user B meant that user A would be recommended highly-rated products from user B. Ratings were based on stars and are hence in the range of 1-5.

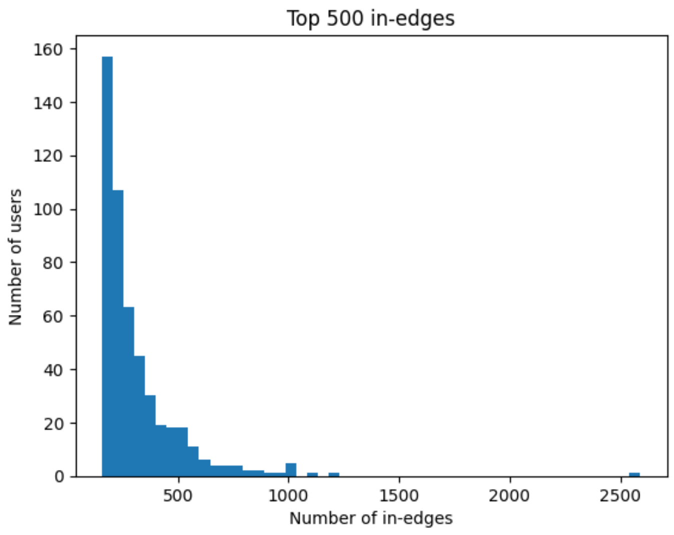
The trust network contains 49k users, with 480k "trust" edges between them. There are also 150k items, with 660k ratings between them.





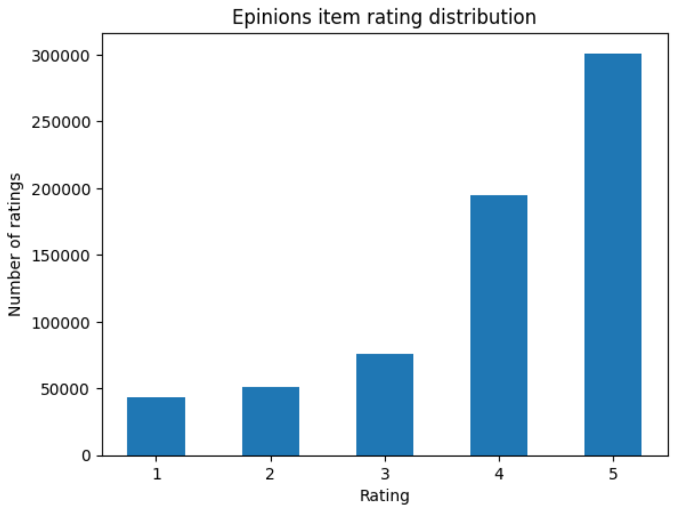
What is interesting about this data set is that it seems that it only contains nodes that have at least 1 “in-edge”. Meaning, they’re trusted by at least one other user. All in-edges come from another one of the 49k nodes, hence it seems this data set is an example of a strongly connected component.

We can also observe that this exponential distribution of in-edges continues at the limits. The following figure depicts the top 500 nodes, ranked by in-edges. We can consider these nodes “celebrities” as many nodes trust their reviews.

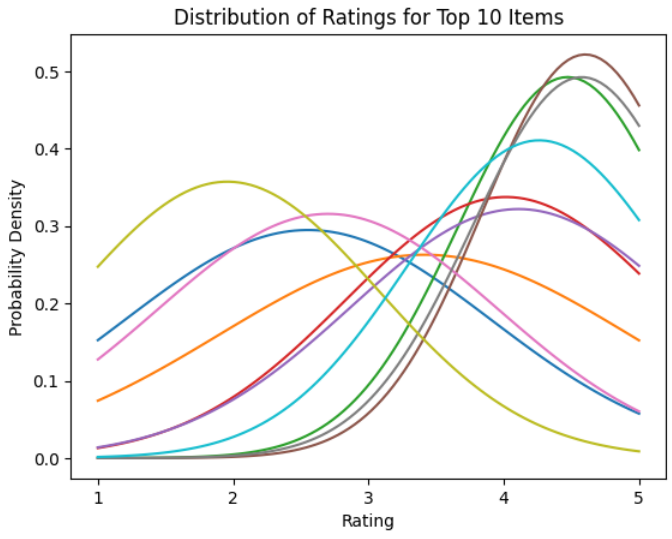


As such, it seems this network is distributed similarly to other social networks, where node popularity exponentially decays, where there are a few trusted authorities who provide powerful and centralised opinions.

When investigating the ratings items received, I wanted to first understand if there is an underlying bias in the ratings, which may influence the performance of a recommendation system.



It was evident by observing these ratings that people's ratings are negatively skewed, i.e. users had a positive bias on their ratings relative to the 5-start scale. I then wanted to understand how ratings are distributed for different items. As there are over 150k items, I decided to only focus on the top 10, being those rated by the largest numbers of users to provide a sample.



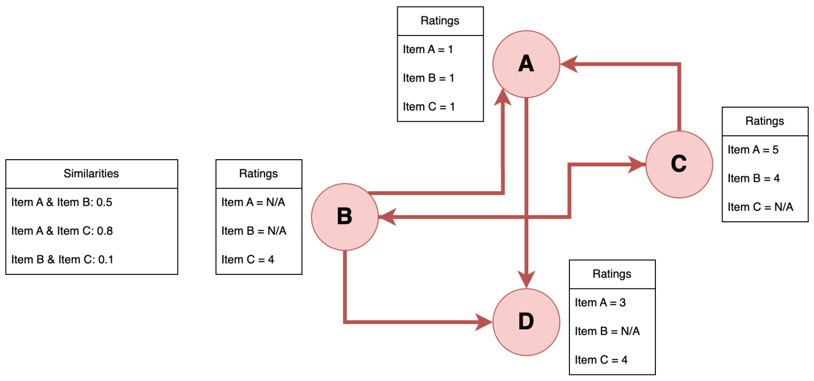
As is evident in these top-10 items, there is some negative skew, however, there are also more contested items. Hence, being able to accurately provide a recommendation will require interpolation from the graph data and is not easily solved by simply assuming high ratings.

2.3 Subsets of Data

To structure the development of these recommendation systems, the data was split into three subsets. The idea is, that if we can build recommendation systems that are limited to only these fundamental elements of “information”, we can then take what we learn to produce more effective recommenders that operate on combinations of these subsets.

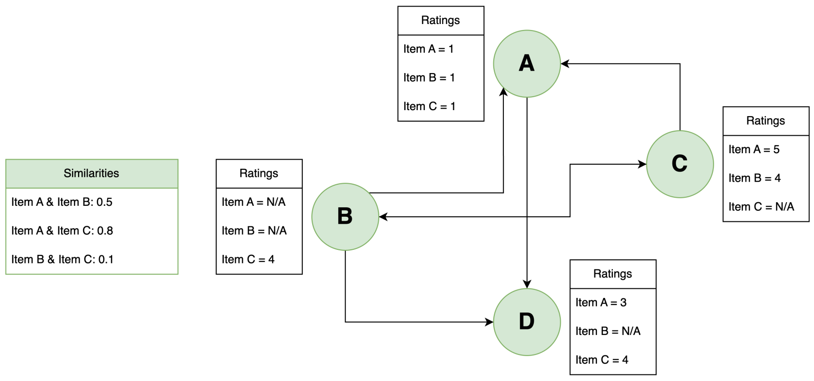
1. Trust Graph

   The trust graph is simply the nodes and the trust connections between one another. Recommendation systems built upon this sub-set of the data only concern themselves with the structure of the social network and not with the ratings on items. I.e. similarity between nodes would be reliant on metrics such as Jaccard Index. Recommendation systems based on the trust graph are optimal for contexts when a data set has no information on the user's ratings or opinions of other items other than the current. Also in contexts when a node is yet to rate any items of its own and must rely entirely on the information provided by others.



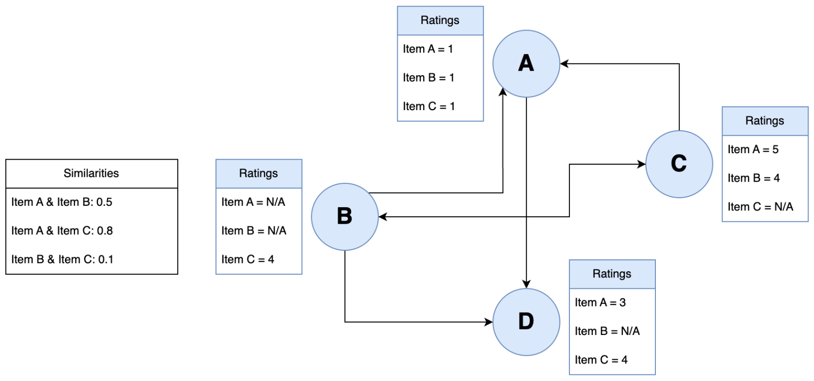
2. Intra-Item information

   The intra-item information is a subset of the data that contains the intra-item similarity, essentially just a table, providing similarity scores between pairs of items. Along with each node's neighbours. This information is useful in contexts when not much information exists about the current item, we can look to other items and try to interpolate.



3. Item-Rating information

The Item-Rating information is a subset of the data that only has nodes, items and their ratings. Recommendation systems built upon this subset draw similarities between nodes based on how they've jointly rated items, not considering the actual social connection between them. Also, not considering how different items are related to one another.



By splitting the data up in this way we can investigate which algorithms perform the best in certain scenarios. For example, a "cold start" user can be defined as one who has not rated any items and has only added a few friends. This can be categorised by any algorithms which only use the trust and intra-item information. I.e. information about social connections and a table relating how similar items are to one another. In this way, the algorithms don't consider a user's ratings. Thus we can identify the algorithms that perform best for cold-start users, and separately for all users.

2.4 Evaluating recommendation systems

Standardising our approach to evaluating the efficacy of recommenders was important such that we could contrast different approaches on an even playing field. We also wanted to be able to evaluate a recommender relatively quickly, with such large graphs and so many items, an ambitious evaluation procedure would prove slow and not let us iterate.

So what information would be needed by all recommendation systems? Nodes/users, directed edges representing a connection, a collection of items, and ratings by users on these items. The problem is, that these data sets prove to be quite sparse. With so many items, and often ratings concentrated on only popular items, computing recommendations for every item, is very costly.

To make recommendations quick, and to replicate an environment where there is substantive data, it was decided to reduce the data sets to a subset with the following algorithm:

- Take the top k items, as ranked by number of ratings
- Only keep the nodes that rated the most popular item

We are now left with a subset that contains active users, who all have at least one item in common (the most popular item), and it is likely that they also share a number of the other top k items due to their popularity. This graph is much smaller than the original full graph, contains the right data for our recommenders and lets us evaluate recommenders and iterate quickly.

So how do we use this graph to quantify performance? Arbitrarily, we settled on a "test-ratio" of 15%.

- Take 15% of the users at random, call these “test users”.
- For each item, i, in the graph:
  - Remove the rating for item i for each test user.
  - Compute the recommendation for item i.
  - Compute the binary accuracy (correct discrete rating or not) and mean absolute error across all the test users.
  - Repeat 5 times to accurately sample the performance of random-based recommenders.

Metrics

As per our evaluation procedure, for each recommender, we trial its performance on different sets of “test users”, and on multiple items. So for every trial, we record the MAE. With this set of MAE scores for each recommender, we can compute the standard deviation to gain an idea of how robust the recommender is/how consistently it provides good ratings. We assume the recommender’s performance models a normal distribution.

Control Recommenders

To maintain a reference point for our recommenders we designed two “control” recommenders.

1. Random

   The random recommender simply provides a random rating of 1-5 for each item.

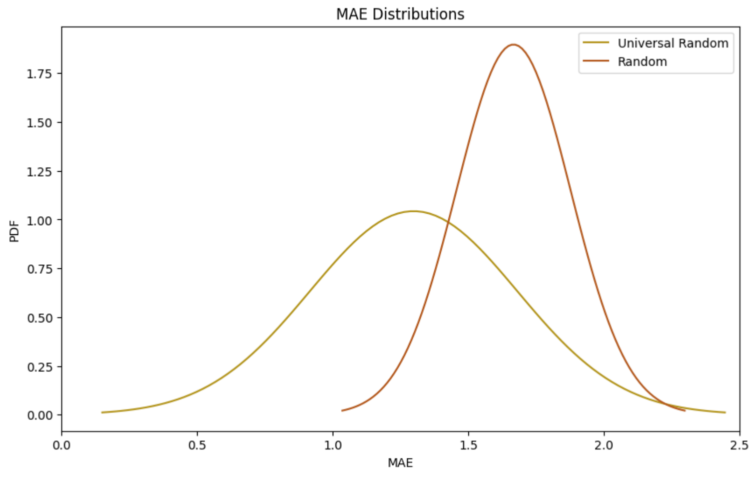
2. Universal Random

   The universal random recommender takes the proportion of ratings given to the item, for example; 20% of users gave a rating of 1, 36% a 2, etc. Based on this proportion we generate a random rating, i.e. 20% chance of 1, 36% chance of 2, etc.

   The idea is that this universal random recommender will give us a form of “random” that compensates for the distribution of ratings for a given item.

Table 1: Mean and Standard Deviation of MAE for Baseline Algorithms

Algorithm	μ_{MAE}	σ_{MAE}
Random	1.67	**0.21**
Universal Random	**1.30**	0.38

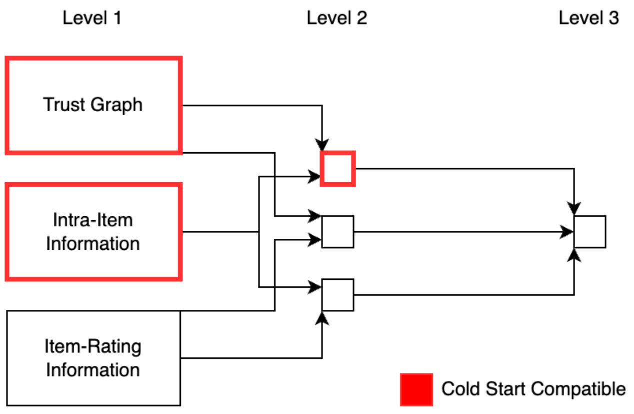


We can see that on the Epinions data set, both recommenders give reasonable results and will be good points of reference for future recommenders.

2.5 Making Our First Principles Approach Explicit

We plan to build many recommendation systems, starting by only accessing one type of data at a time. This will let us then evaluate these recommendation systems and understand how derivatives of different types of data act in different contexts.

For each level in the following figure, we will create recommendation systems. Our definition of a "cold start" node is one that has made connections but has yet to rate anything. Hence, the areas outlined in red in the following graph show us where we can create recommendation systems that are compatible to be used with cold start nodes. This also demonstrates the benefits of taking this approach.



3 Level 1: Investigating the Trust, Item and Intra-Item Information

In this section, we tried to build the best recommendation systems we could, confined to these information sub-sets. The aim is to later combine what we learn to form a recommender that best introduces all the available information.

3.1 Trust Graph

3.1.1 Similarity Metrics

One of the most popular similarity measures between users in related research is the Jaccard index. Thus, we decided to compute the Jaccard index between all pairs of users and use this similarity score in a variety of contexts.

With this standard jaccard implementation, we found there would often be nodes with many neighbours, but with a jaccard value of zero to all other nodes. Thus, the Jaccard score wouldn't provide any information. To encode the information of two nodes being neighbours, the Jaccard index was altered as follows:

$$J'(u_i, u_j) = Jaccard(N(u_i), N(u_j)) + \begin{cases} \frac{1}{|N(u_i) \cup N(u_j)|} & u_j \in N(u_i) \\ 0 & \text{otherwise} \end{cases}$$

See the 'Methodology' section for more information on the Jaccard Index.

With this information, multiple recommenders were designed and evaluated. These recommenders included a Monte-Carlo Random-Walk, Jaccard-Weighted Model, and a novel method we call "Jaccard Majority of Majorities".

3.1.2 Neighbour Constrained Algorithms

The most simple approach to take with the trust graph is, given a node, investigate it's neighbours. From the ratings of a node's neighbours, we can infer the current node's rating in a variety of ways. For the most part, it comes down to the three M's, mean, median and mode.

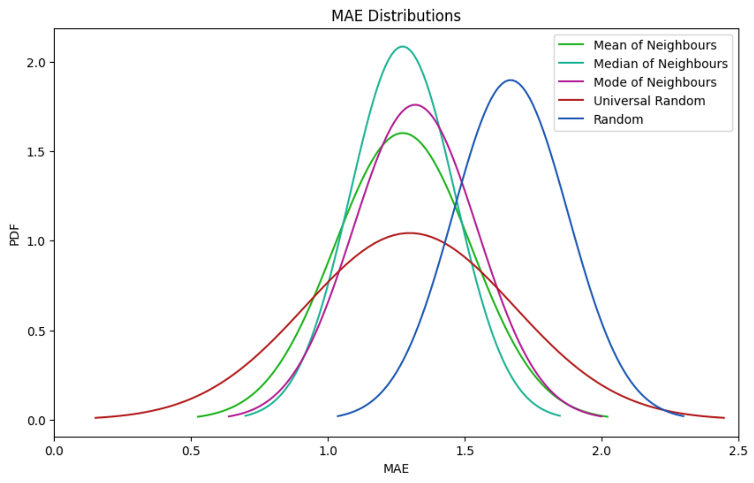
Mode/Median/Mean of Neighbours

Let's say we are rating item $\boldsymbol{i}$, we go through a node's neighbours, i.e. those who the node has an outgoing edge to, and take the mean, median or mode of their ratings for item $\boldsymbol{i}$ (if it exists). If no ratings for $\boldsymbol{i}$ exist, we return a random rating. Note, this algorithm mutates the underlying data source, whereby a recommended value for a node, can be used by another node in its rating inference. The pseudo-code for these algorithms can be found in the appendix.

The performance of these recommenders can be seen below.

Table 2: Mean and Standard Deviation of MAE for Neighbourhood Algorithms

Algorithm	μ_{MAE}	σ_{MAE}
Mean of Neighbours	**1.27**	0.25
Median of Neighbours	**1.27**	**0.19**
Mode of Neighbours	1.32	0.23
Universal Random	1.30	0.38
Random	1.67	0.21



As we can observe from the performance, the mean and median of neighbours were the best algorithms in this range. The mode version did not perform as well. I believe this is because 'mode' cannot produce decimal-precision ratings it also defaults to randomness when a mode can't be determined. I.e. let's take [1, 5] as an example, the median will return 3.0, whereas mode will throw an error and so a random rating will be returned.

Jaccard-Weighted Model

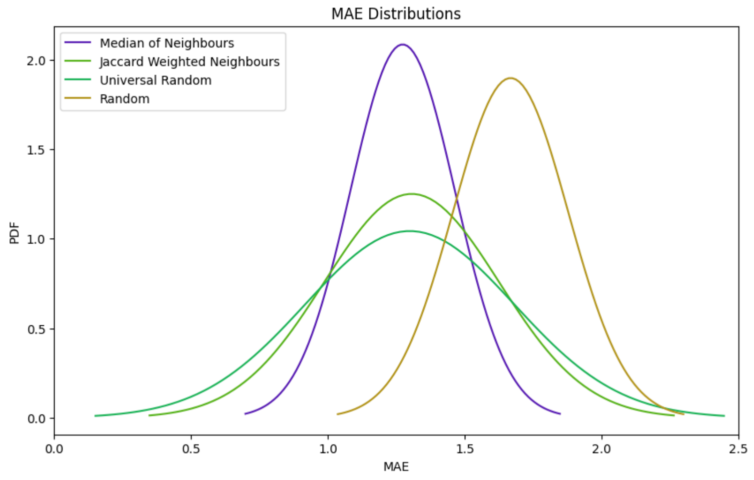
The Jaccard-Weighted Model extends the neighbour-based models from before. Effectively, given the neighbours of node A, their rating of an item is considered based on their Jaccard Index with node A. Note we use the modified Jaccard Index (J') mentioned earlier in this section.

So we initialise a length 5 array, called "potentialRatings". As we enumerate the neighbours, we contribute their Jaccard index to the corresponding ratings bucket. By taking the argmax, we identify the rating that received the maximum "score".

The performance of this algorithm, along with the previous Median of Neighbours model can be seen below.

Table 3: Mean and Standard Deviation of MAE for Jaccard Weighted Neighbours

Algorithm	μ_{MAE}	σ_{MAE}
Median of Neighbours	**1.27**	**0.19**
Jaccard Weighted Neighbours	1.31	0.32
Universal Random	1.30	0.38
Random	1.67	0.21



We can observe that the Jaccard-Weighted model performs quite similarly to the Median of Neighbours model when it comes to mean accuracy. However, the distribution is wider and the peak lower, implying the algorithm is less robust.

3.1.3 Full-Graph Algorithms

We then investigated recommendation systems which considered the entire trust-graph. These are the Monte-Carlo Random Walk model and the Jaccard Majority of Majorities. This is because Jaccard values are computable for any two nodes in a graph. There is no reason to be constrained to neighbours.

Monte-Carlo Random Walk

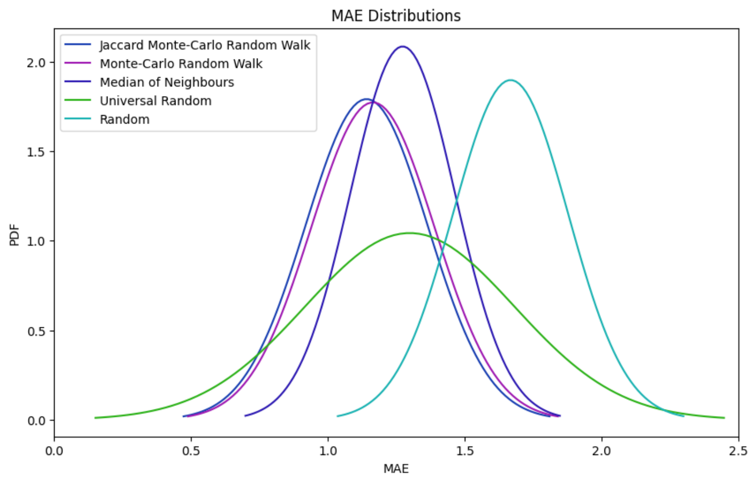
The MC Random Walk model works by starting at the node we are trying to produce a rating for and then performing a random walk. As the distance from the original node, d, increases, so does the probability that a random rating is returned. This random walk continues until a node with a rating is reached, or this random rating is produced. We run this experiment k times for each node and take the mean rating from the nodes reached via random walk.

We then extended this algorithm by choosing the neighbour, at each step of the random walk based on a weighted random incorporating the Jaccard index.

The performance of both MC algorithms, as compared with the others can be seen following.

Table 4: Mean and Standard Deviation of MAE for Monte-Carlo Algorithms

Algorithm	μ_{MAE}	σ_{MAE}
Jaccard Monte-Carlo Random Walk	**1.14**	0.22
Monte-Carlo Random Walk	1.16	0.23
Median of Neighbours	1.27	**0.19**
Universal Random	1.30	0.38
Random	1.67	0.21



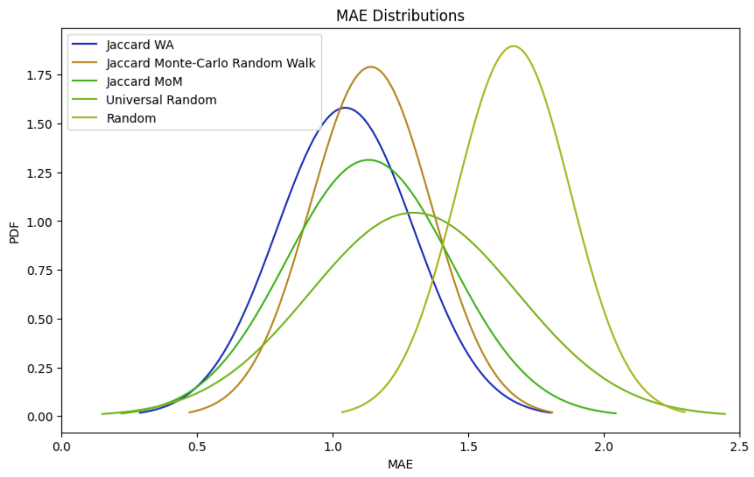
We can observe that the Monte Carlo algorithm outperforms the previous models. Interestingly, the Jaccard weighted version performed better. This implies to us that Jaccard indexes seem to hold more valuable information than mere trust connections.

Jaccard Majority of Majorities (JMoM) and Weighted Average (Jaccard WA)

These recommenders work by considering every other node, which has rated item i, in the graph. We then perform a summation of the Jaccard values for every node corresponding to each rating value. In the Majority of Majorities version, we then take the rating that has received the highest contribution of Jaccard scores. However, in the Weighted Average version, we instead take the weighted average of these ratings, weighted by the contribution of Jaccard scores. We will call these techniques MoM and WA moving forward. Psuedo-code for this 'WA' recommendation framework can be found in the appendix.

Table 5: Mean and Standard Deviation of MAE for Monte-Carlo Algorithms

Algorithm	μ_{MAE}	σ_{MAE}
Jaccard WA	**1.05**	0.25
Jaccard Monte-Carlo Random Walk	1.14	0.22
Jaccard MoM	1.13	0.30
Universal Random	1.30	0.38
Random	1.67	**0.21**



The JMoM performed reasonably well. Its mean accuracy being better than the random-walk. However, it was less robust. The WA version performed very well significantly beating both the MoM version and the random walk. By overcoming the constraint of only considering direct neighbours, the JMoM and Jaccard WA models are able to produce decent results, with a very simple algorithm.

The other upside is the underlying MoM and WA approaches can be used with any user-to-user similarity metric.

3.2 Item Information

3.2.1 Similarity Metrics

For the item rating information, we came up with two measures to quantify the similarity between nodes based on item ratings.

1. Jaccard-Item Index
2. Item-Rating Difference

Jaccard-Item Index is a form of the Jaccard index, but instead of relating users using their neighbours, we relate them based on the items they have rated.

$$J_I(u_i, u_j) = Jaccard(I(u_i), I(u_j))$$

Where J_I is the Jaccard-Item Index and $I(n)$ is the items rated by node n. On the other hand, the Item-Rating difference compares two nodes as follows:

$$R_D(u_i, u_j) = 1 - \frac{\sum_{i \in I(u_i) \cap I(u_j)} |R(u_i, i) - R(u_j, i)|}{4 \cdot |I(u_i) \cap I(u_j)|}$$

Although the summation may seem complex at first glance, we are effectively enumerating the items that two users have both rated. For each item, we take the absolute difference in their rating. We then normalise by dividing by 4 (maximum difference in ratings) and obtaining the average by dividing by the magnitude of the intersection.

With these two metrics, we performed similar algorithms to those based on the Jaccard Index.

3.2.2 Algorithms

Item-Jaccard WA

In this model, we took a similar approach to the successful Jaccard WA model from the previous section.

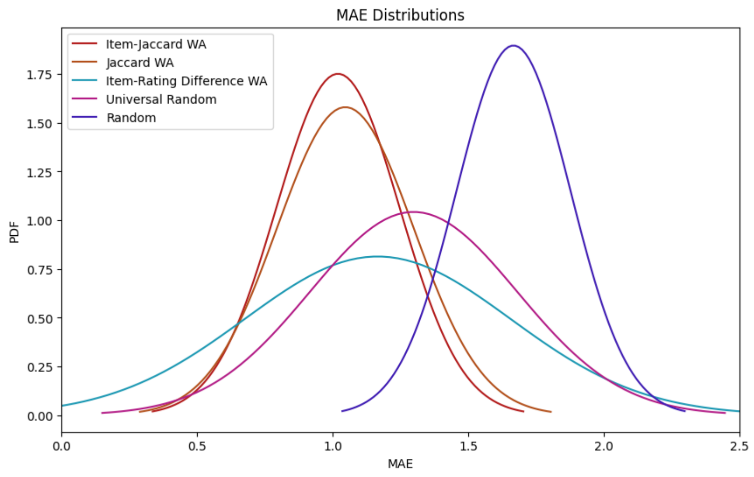
However, instead of using the J' form of Jaccard Index, we used this alternative form which is based on items rated.

Item-Rating Difference WA

We again use the weighted average framework approach but with the Item-Rating Difference. The performance of these alternative WA models can be seen below:

Table 6: Mean and Standard Deviation of MAE for Monte-Carlo Algorithms

Algorithm	μ_{MAE}	σ_{MAE}
Item-Jaccard WA	**1.02**	0.23
Jaccard WA	1.05	0.25
Item-Rating Difference WA	1.17	0.49
Universal Random	1.30	0.38
Random	1.67	**0.21**



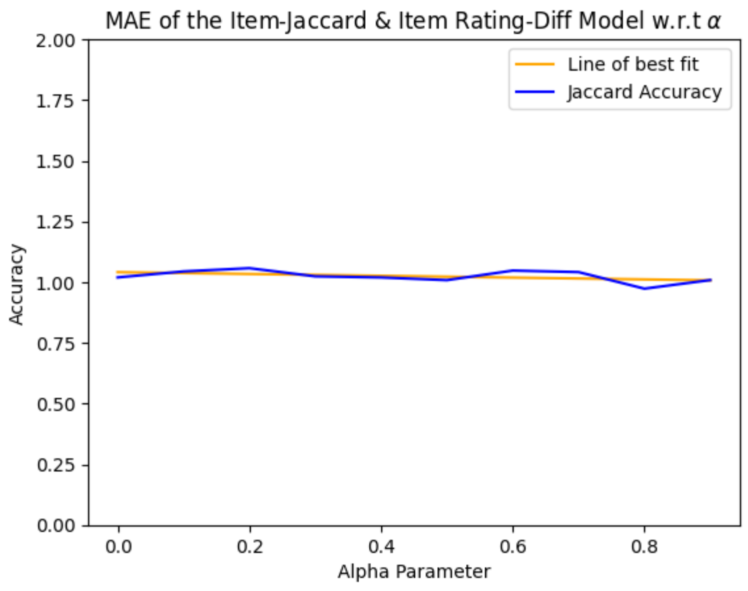
Again, very interesting results. We can see that the Item-Jaccard form of WA outperforms the previous Jaccard WA. Item-Rating Difference doesn't seem to perform very well and is quite random in its performance. A potential reason for the Item-Jaccard WA model performing so well is that a user's social circle may not be the best predictor for who is similar to them, rather relating people by their items themselves does a better job.

Combined Item-Jaccard and Item-Rating Difference Models

As an experiment, we attempted to combine both the previous similarity scores. As both metrics are values in the range [0, 1], we could use either multiplication or addition. As values can often approach 0, it was decided that an addition with an adjustable alpha parameter would be better.

Thus, the metrics would be combined as:

$$I_S(u_i, u_j) = \alpha \times R_D(u_i, u_j) + (1 - \alpha) \times J_I(u_i, u_j), \alpha \in [0, 1]$$



A simple iterated search over the parameter α proved to show that this combination model would not outperform the other models regardless. So no more attempt at this combination was made. This can be seen in the above figure.

3.3 Intra-Item Information

The Intra-Item Information concerns itself with the relationships between items themselves. With an understanding of item similarity, the hope is we can develop algorithms that capitalise on this information.

3.3.1 Similarity Metrics

Pearson Correlation

One similarity metric used to determine the relationship between two items is the Pearson correlation of the ratings given to them. This was proposed by [8].

The similarity function between two items, i and j, works as follows:

$$\rho(i,j) = \text{set of users who have rated both item } i \text{ and } j.$$

$$corr(i,j) = \frac{\sum_{u\in\rho(i,j)} (R(u,i) - \overline{R}(u))(R(u,j) - \overline{R}(u))}{\sqrt{(\sum_{u\in\rho(i,j)} (R(u,i) - \overline{R}(u))^2)(\sum_{u\in\rho(i,i)} (R(u,i) - \overline{R}(u))^2)}}$$

We then want to skew this correlation, such that if $|\rho(i,j)|$ is large, we are more confident. So, the final similarity metric, which is equivalent to that given in [8], is:

$$sim(i,j) = \frac{1}{1 + e^{-\frac{|\rho(i,j)|}{2}}} \times corr(i,j)$$

The sigmoid function is used to balance how much we favour the magnitude of ρ and to ensure our similarity metric is in the range $[0, 1]$.

Thus, for any pair of items, using just the Intra-Item graph, we can determine a quantified value that indicates how "related" the pair of items is. -1 being actively unrelated, 1 being the same item.

Intra-Item Jaccard

In this similarity metric, to relate items we use Jaccard. i.e. the number of users who have rated i and j divided by the union of the users who rated both i, j. So the similarity between the two items is:

$$J_{II}(i,j) = Jaccard(p(i,j), p(i,i))$$

3.3.2 Algorithms

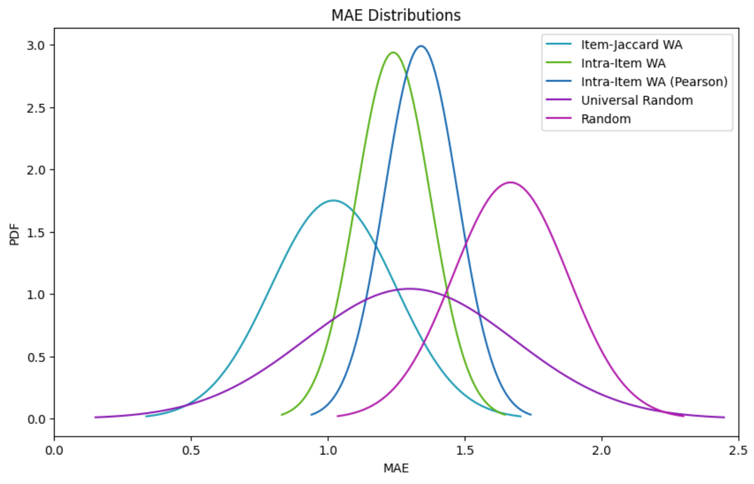
Intra-Item WA

In a similar fashion to the previous algorithms, we manipulated the WA approach to work with Intra-Item similarity data. Let's say we are predicting a recommendation for user u, for item i. The algorithm works by iterating through u's neighbours. We store 5 buckets, one for each rating such as the previous WA implementations. For each neighbour, we iterate through all of the items they have rated. Let's call the current item in the iteration, j. We then contribute $sim(i,j)$ to the bucket relating to j's rating. This is slightly different to the WA framework as we iterate through each user's items rather than using a user-user similarity metric.

The performance of the algorithms, one based on J_{II} and one based on the Pearson sim, can be seen below.

Table 7: Mean and Standard Deviation of MAE for Monte-Carlo Algorithms

Algorithm	μ_{MAE}	σ_{MAE}
Item-Jaccard WA	**1.02**	0.23
Intra-Item WA	1.24	0.14
Universal Random	1.30	0.38
Intra-Item WA (Pearson)	1.34	**0.13**
Random	1.67	0.21



It is evident both of the intra-item-based models underperform our previous models. The intra-item similarity using the Jaccard approach outperforms the Pearson correlation approach. This is a very interesting result, potentially this could be a better metric for determining the similarity of items and could be applied in [8]. It is evident the intra-item similarity conveys some information as we perform better than random, hence this could be useful in combination models in the coming sections.

3.4 Summary

By splitting the data up in the way we did, we now understand how algorithms focused on the different types of information perform. In a way, we know the value of the different types of information. Our item-based similarity metrics resulted in better-performing algorithms than the trust-based metrics. This implies that comparing users based on how they've rated similar items gives a better metric of their similarity than comparing based on trust or the Jaccard index. We also know that intra-item similarities can convey important results and that using a Jaccard-based approach outperforms the Pearson correlation as used in the TrustWalker paper[8] on downstream tasks.

4 Level 2: Combining the different information types

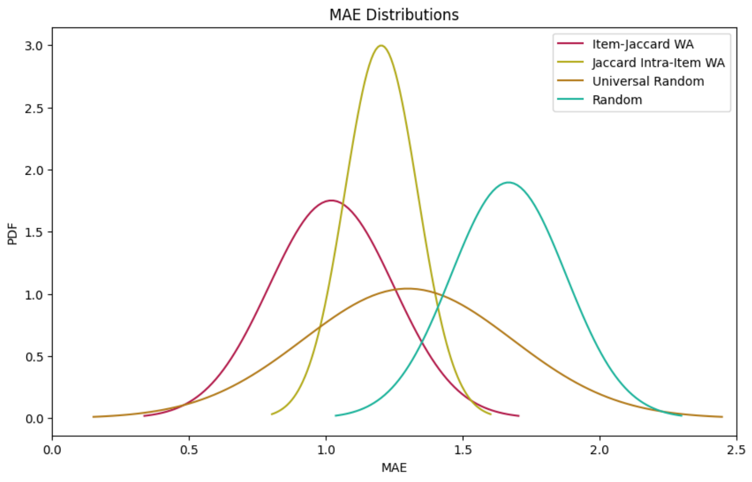
4.1 Trust Information, Intra-Item

Jaccard Intra-Item WA

As previously discussed, combining both Trust and Intra-Item information means these algorithms are effective on cold start users. In an attempt to combine both these modes of data, we developed an intersection between the Intra-Item WA (J_{II}) which operates on neighbours, and the Jaccard WA which operates on the entire graph. The idea of this algorithm is for every (user, item) pair, we contribute the Jaccard index of the user multiplied by the item's similarity to our rating bins. The performance can be seen below, pseudo-code in the Appendix.

Table 8: Mean and Standard Deviation of MAE for Jaccard Intra-Item WA Algorithm

Algorithm	μ_{MAE}	σ_{MAE}
Jaccard Intra-Item WA	1.20	**0.13**
Item-Jaccard WA	**1.02**	0.23
Universal Random	1.30	0.38
Random	1.67	0.21



Combining these two forms of information seemed to produce a very consistently performing recommender as is characterised by the large peak. However, it did not perform very well relative to our current best, Item-Jaccard WA in terms of mean MAE.

4.2 Intra-Item, Item-Rating

Weighted Item-Rating Difference WA (WIRD WA)

This model combines the Intra-Item Graph with the Item-Graph. Thus combing information about how nodes rated individual items and how the items are interrelated. The general idea was to slightly alter the item rating similarity metric, R_D, to provide a bias taking into account the similarity of items to the item we are recommending for.

The letter K will represent the item we are recommending for in the following equations.

$$R_D(u_i, u_j) = 1 - \frac{\sum_{i \in I(u_i) \cap I(u_j)} |R(u_i, i) - R(u_j, i)|}{4 \cdot |I(u_i) \cap I(u_j)|}$$

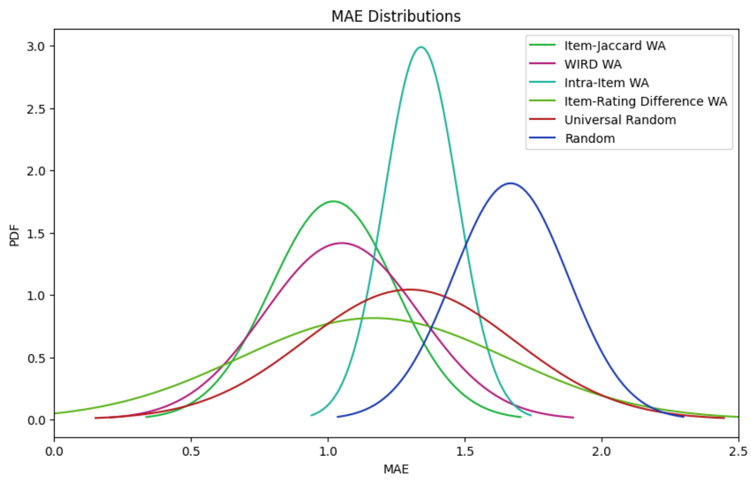
$$\downarrow$$

$$WR_D(u_i, u_j, K) = 1 - \frac{\sum_{j \in I(u_j)} J_{II}(K, j) \times |R(u_i, K) - R(u_j, j)|}{4 \cdot |I(u_j)|}$$

Note for intra-item similarity we use J_{II} rather than sim as we found it was more performant in the previous section. The results can be seen in the figure below:

Table 9: Mean and Standard Deviation of MAE for WIRD WA Algorithm

Algorithm	μ_{MAE}	σ_{MAE}
WIRD WA	1.05	0.28
Item-Jaccard WA	**1.02**	0.23
Intra-Item WA	1.34	**0.13**
Item-Rating Difference WA	1.17	0.49
Universal Random	1.30	0.38
Random	1.67	0.21



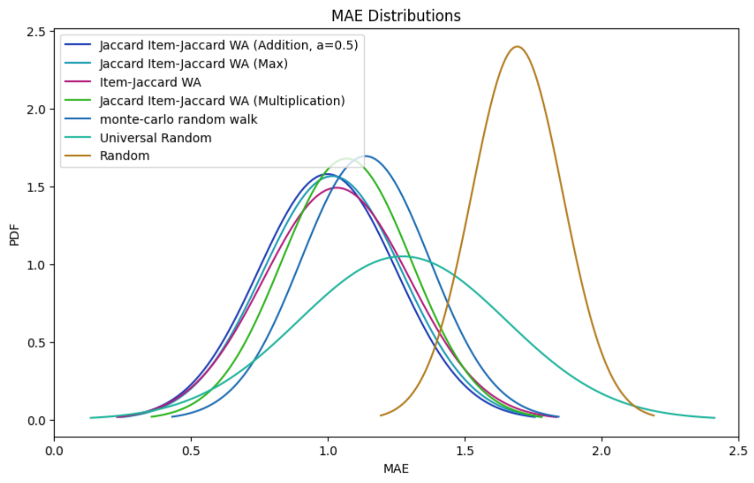
The WIRD model performed quite well. Slightly under performing the Item-Jaccard WA model. However, it performs better than just the intra-item information or item-rating difference WA alone. Hence, we have successfully combined the intra-item and item-rating information to achieve a better result.

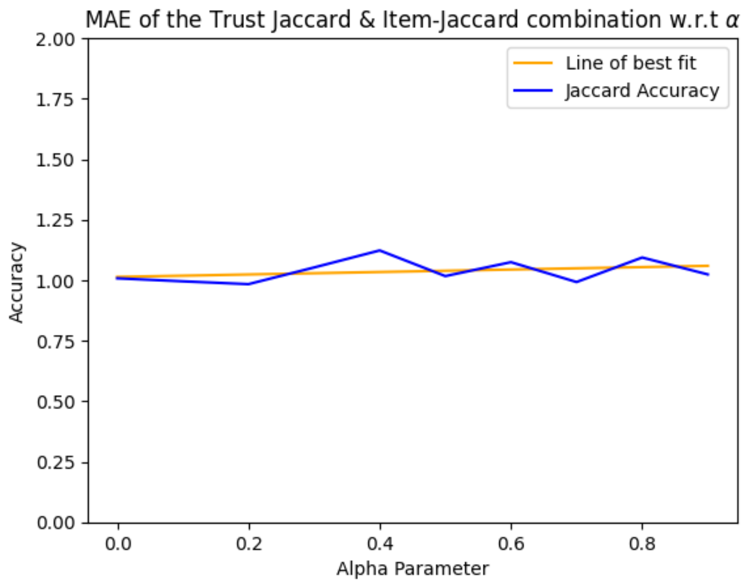
4.3 Trust, Item-Rating

Jaccard Item-Jaccard WA

In this model, we combine the Trust information with the Item-Rating information. To do so, we explored different ways of combining the Jaccard WA and Item-Jaccard WA models. Both similarity metrics are in the range $[0, 1]$, so we attempted linear combinations, multiplying them and using the max function.

Here is the performance of the linear combination of the two, with $\alpha = 0.5$, as well as an exploration of the impact of alpha.

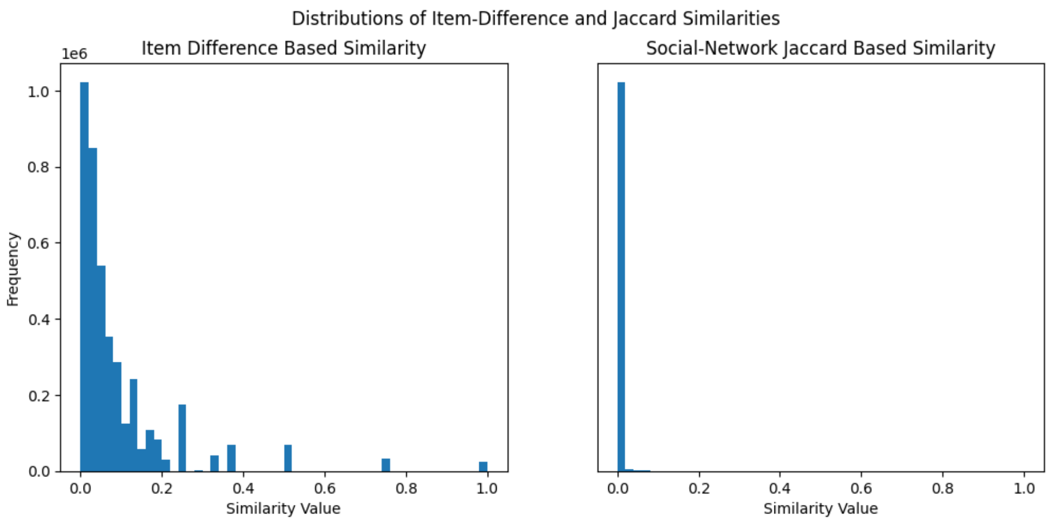




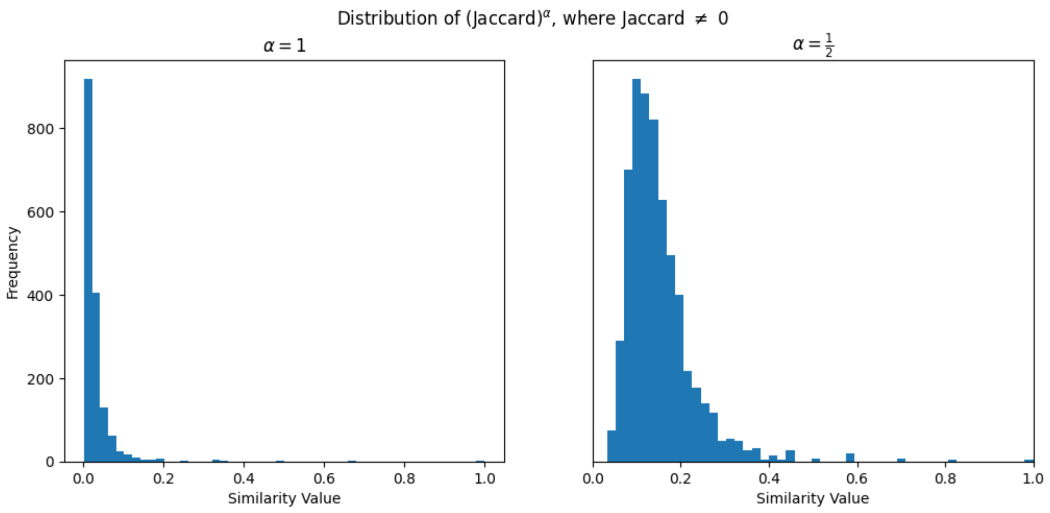
Inspecting the graph, it seems the weighting on the linear combination makes a very minimal difference to the model's MAE. What this leads me to believe is that the model performs well as there is a form of "backup". When the trust-based jaccard is 0, the item-jaccard is used and when the opposite is true, the trust-based jaccard is used.

Looking at the multiplication-based model, it produced an interesting result which was mirrored in the Jaccard and J_{II} combined from the Trust and Intra-Item section. The accuracy was essentially a blend between the Jaccard model and the Item-Difference one, however, the distribution is taller, implying it is potentially more robust and consistent in its accuracy.

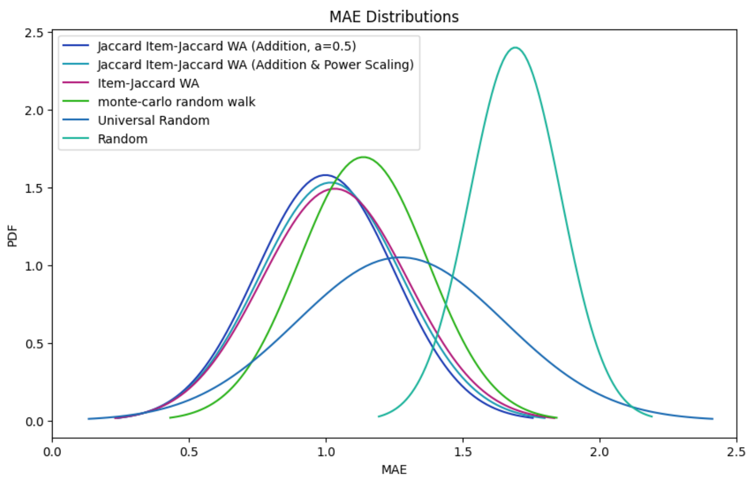
The *max*-based model performed well but not as well as addition. This implies that even small values of Jaccard or Item-Jaccard contribute information to the recommender that the addition model captures.



These distributions were quite striking. It seems that the item-rating distribution provides value for many more nodes than the Jaccard index. Almost all node pairs have a Jaccard index of 0. I figured that if we could augment that distribution, such that values near 0 were more comparable to those of the item-diff values then their combinations may be better. To do this, applying powers in the range $(0, 1)$ seemed like a good option, values near 0 would be stretched higher, and values near 1 not be impacted as much. Thus I experimented with a simple square root.



Great, as expected the distribution of non-zero Jaccard indexes was shifted to the right. I then measured the accuracy of the *addition* combination model, with the stretched augmented Jaccard index.



Unfortunately, this power-scaling didn't seem to improve results. Rather our original addition-based Jaccard, Item-Jaccard WA combination model is still our best performer!

5 Level 3: Combining all information types

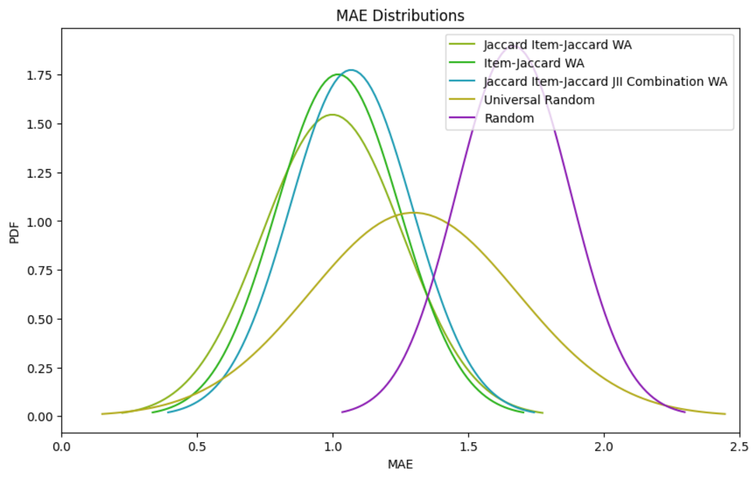
Intra-Item, Trust and Item-Rating Full Combination

We are now ready to combine all the data we have in an attempt to build our best-performing model.

From previous experiments, we found that the Jaccard cross Item-Jaccard Weighted Average model performed the best. This incorporates the trust information and the rating information. How can we extend this to incorporate intra-item information? To do this, we defined the similarity between two given users as $U_{sim} = J(u_1, u_2) + I_j(u_1, u_2)$. Let's say we are predicting a rating for u_1, item I. We would iterate through all other nodes in the graph, and for each of the items they've rated, contribute $U_{sim} \times J_{II}(I, j)$ where j is the other item. Essentially, other users influence our user's rating by;

1. how similar they are based on the social network structure
2. how similar they are based on the items they've rated
3. how similar each of their items is to the one we are predicting

The result was as follows:



Interestingly, the combination model under-performed our best model in terms of mean MAE. However, it featured the highest peak of any of the distributions short of random. This implies it is the most consistent model, performing similarly regardless of the user's request, or item asked to predict.

6 Validation With Other Data Sets

Throughout this report, we have designed our models and tested them using the Epinions data set. This is a data set popular in literature and gives us a good playground. However, we must ensure we don't overfit, so we have tested our algorithms on a range of data sets. These being:

1. Epinions

   The Epinions data set as mentioned previously was a consumer review site, where users could place reviews on items, and "trust" other users.

2. FilmTrust [6]

   Not much information circulates online about FilmTrust, but from what I can understand it was a movie review platform, where users could 'trust' one another's reviews. This data set was scraped from the site in 2011 and is commonly used in recommendation system papers.

3. CiaoDVD [5]

   Again, little information exists on this site, however, it seemed to be another movie review website, with a similar structure to FilmTrust. This data set was scraped in 2013.

Table 10: Mean of MAE for All Algorithms on Different Data Sets with Overall Mean

Algorithm	Data Sets			
	Epinions	FilmTrust	CiaoDVD	μ_{MAE}
Jaccard Item-Jaccard WA	**1.00**	**0.66**	**0.53**	**0.73**
Item-Jaccard WA	1.02	0.67	0.58	0.76
WIRD WA	1.05	0.67	0.72	0.81
Jaccard Item-Jaccard JII Combination WA	1.07	0.69	0.63	0.80
JWIRD WA	1.09	0.67	0.72	0.83
Item-Rating Difference WA	1.17	0.67	0.79	0.88
Universal Random	1.30	0.89	0.72	0.97
Jaccard WA	1.05	1.14	1.73	1.31
Jaccard Intra-Item WA	1.20	1.07	1.75	1.34
Jaccard MoM	1.13	1.19	1.77	1.36
Intra-Item WA	1.24	1.20	1.64	1.36
Jaccard Monte-Carlo Random Walk	1.14	1.20	1.81	1.38
Monte-Carlo Random Walk	1.16	1.20	1.82	1.39
Mean of Neighbours	1.27	1.25	1.67	1.40
Jaccard Weighted Neighbours	1.31	1.22	1.66	1.40
Mode of Neighbours	1.32	1.23	1.67	1.41
Intra-Item WA (Pearson)	1.34	1.21	1.67	1.41
Median of Neighbours	1.27	1.26	1.76	1.43
Random	1.67	1.34	1.82	1.61

It appears the Jaccard Item-Jaccard WA recommendation algorithm performed best across all data sets with the lowest Mean Absolute Error (MAE) value. The algorithm consistently outperformed other algorithms in each of the data sets - Epinions, FilmTrust, and CiaoDVD - with the

MAEs of 1.00, 0.66, and 0.53 respectively. The overall mean MAE (μ_{MAE}) for this algorithm was 0.73, which was significantly lower than other algorithms.

Several observations can be made from the results in Table 10:

- The Random algorithm had the highest MAE on every experiment. This was expected, but helps us in validating our models didn't feature any significant bugs.
- The performance of algorithms varied across data sets. For instance, the Jaccard WA algorithm had a relatively low MAE for Epinions but performed poorly on the FilmTrust and CiaoDVD data sets. This emphasizes the importance of testing recommendation algorithms on multiple data sets to get a comprehensive understanding of their performance.
- Algorithms that combined different recommendation techniques, like Jaccard Item-Jaccard JII Combination WA, did not necessarily guarantee better performance than their simpler counterparts.

7 Adversarial Attacks

We will now investigate how our recommendation systems respond to adversarial attacks on the network. To conduct this experiment we must construct an attack method and mutate the graph to represent it.

False Account Attack

The attack we will simulate is that of an actor who wants to artificially reduce the ratings of certain items on the system. The actor can create fake "bot" accounts and befriend an adversarial celebrity.

Here is how it will work:

1. Identify the adversarial celebrity, we will arbitrarily choose this as the 10th most popular node by in-edges.
2. Create k fake nodes, who rate every item a 1/5.
3. All of the fake nodes will trust the celebrity and the celebrity will in turn trust the fake nodes.

Adversarial Robustness Testing

To test the robustness of our algorithms. We will use the "epinions" data set and use 150 adversarial accounts (around 5% of the graph by node size). We deem this to be quite a significant attack on the integrity of the network. These are the algorithms we tested:

- Jaccard Item-Jaccard WA
- Item-Jaccard WA
- Jaccard WA
- WIRD WA
- Jaccard Item-Jaccard JII Combination WA
- Jaccard Monte-Carlo Random Walk

Table 11: Algorithm Performance Metrics Under Adversarial Conditions

Algorithm	Normal		Adversarial		Reduction (%)
	μ_{MAE}	σ_{MAE}	μ_{MAE}	σ_{MAE}	μ_{MAE}
Jaccard Item-Jaccard WA	**1.00**	0.26	1.25	**0.19**	24.93
Item-Jaccard WA	1.02	0.23	1.24	0.20	21.77
Jaccard WA	1.05	0.25	**1.17**	0.22	12.12
WIRD WA	1.05	0.28	1.32	0.23	25.53
Jaccard Item-Jaccard JII Combination WA	1.07	**0.22**	1.23	0.20	15.58
Jaccard Monte-Carlo Random Walk	1.14	0.22	1.21	**0.19**	**6.39**

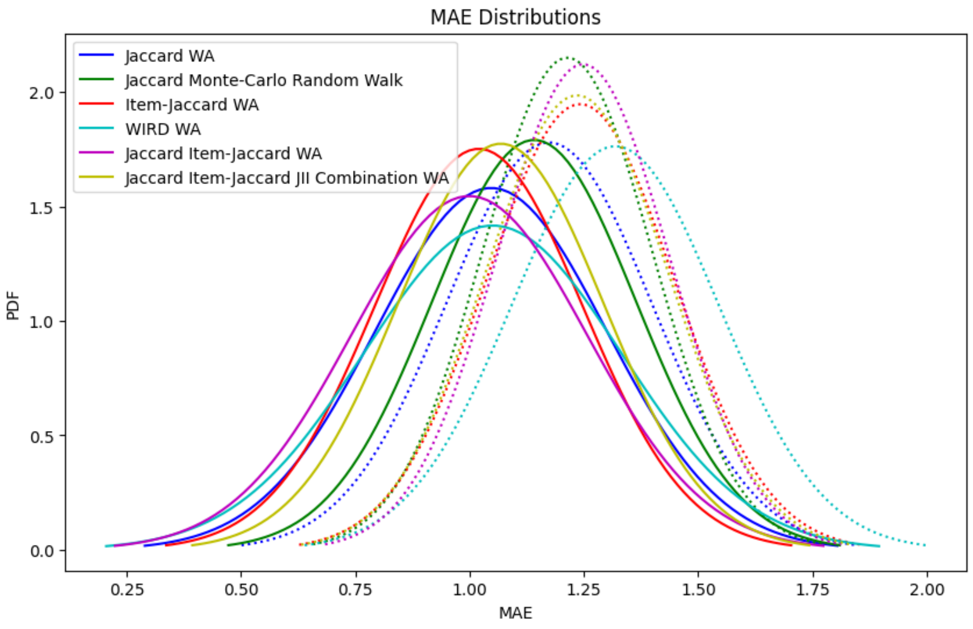


Figure displaying the MAE distributions for different recommender algorithms. The dotted lines depict the performance with an adversarial network and the solid lines show the same recommender under normal conditions.

In these results, we can observe that the minimally impacted algorithms were those that relied on the trust information. These are the Jaccard WA model and the Jaccard Monte-Carlo Random Walk. The most impacted were those that relied on Item-Rating and Intra-Item information, i.e. Item-Jaccard WA and WIRD WA. Interestingly, the combination model, which made use of all forms of data, seemed to not suffer as big a reduction in performance.

8 Discussion

We successfully broke down the concept of a recommender system to its core principles, starting with the information it ingests. As our algorithms had access to more information we found that we could achieve higher accuracy of predictions as evidenced by our best model being a combination of trust information and item information.

Through breaking down the different information forms, we noticed that the most important feature when assigning the similarity between two users for their preferences is the way they've rated other items and which items they have engaged with. This was evident in the Item Information section. The trust and intra-item based algorithms led to recommenders which scored $> 1.3\ \mu_{MAE}$ as is evidenced in Table 10.

All other models included the item-rating information in some way and ranged from $0.73 - 0.97$ μ_{MAE}, which is a huge difference. This result makes logical sense as it's more likely we are similar in opinion to a person who we share interests with than a person who we share friends with. The downside to this is that the item-rating information is the information that we lack when cold-starting a user. When a user first joins a system, they can add a few friends but don't produce enough of a profile via their item ratings to allow this information to be utilised.

We defined a cold start user as one who has not rated any items and has only added a few friends. We then inferred that algorithms that only use the Trust and Intra-Item information would be usable in these situations (see 2.4). The unfortunate outcome of the previous result is that these algorithms clearly perform much worse. The one conclusion we could make was that using the Jaccard WA approach was the lesser of evils, and provided the best performance given a bad environment, outperforming random walk and intra-item-based approaches. A further idea that could be explored is simply using an approach like the Universal Random, which was one of our baseline recommenders. This recommender performed reasonably when averaged across our data sets and predicts preferences based on the distribution of current ratings for said item. This makes sense, and in cases where we have no information on our user, this is an approach that yields results better than others operating on the same constraints.

When we started combining information we found it was quite difficult to outperform the Item-Jaccard-based recommender, which already performed well and achieved a μ_{MAE} of 0.76. As the other information, such as the Jaccard Index seems to be of a lower signal/noise ratio, combining the Item-Jaccard with anything else usually added noise and reduced its performance. The one case where we managed to improve upon the base Item-Jaccard model was when we performed addition with the Jaccard Index. So the similarity between the two users is based on the sum of the Jaccard Index and Item-Jaccard Index. As per the distributions of Jaccard and Item-Jaccard shown in section 4.3, the Jaccard Index is usually zero. We think that the performance of the recommender is increased as the Jaccard provides some marginal information which can impact the ratings in the case when a user trusts a celebrity. That way the user has a non-zero jaccard index with many other nodes who also trust said celebrity, hence, slightly shifting the rating prediction in the direction of these users.

We attempted combination models of various forms, combining the similarity scores with multiplication, with summation and with a maximum function. We also investigated scaling the Jaccard Index values so the distribution was more similar to the Item-Jaccard distribution. The idea behind this was to make the two scores easier to combine. However, we discovered that scaling the distributions had minimal impact on the resulting performance. We also concluded that simply using an equally weighted summation of the two scores leads to the simplest and most effective model.

Hence, the highest-performing recommendation model combined trust information and item rating information. It is not suitable for cold-start users. The model is based on the Weighted Average framework and computes user similarities with the following function:

$$JIJ_{sim}(u_1, u_2) = Jaccard(N(u_1), N(u_2)) + Jaccard(I(u_1), I(u_2))$$

In the above function, $N(u_i)$ returns a set of the neighbours of u_i and $I(u_i)$ returns a set of the items rated. For information on the weighted average framework or the Jaccard function see section 2.2. For pseudo-code of the weighted average framework see the appendix.

We also found that recommenders based on the WA framework outperformed random-walk methods. By reviewing Table 10, it is evidenced that random walk methods actually perform quite poorly when their results are averaged across the diverse data sets. We also found that the WA framework outperforms a similar approach called Majority of Majorities (this is covered in section 3.1.3). We believe this approach to collaborative filtering is effective because it is not constrained by the structure of the graph (as is a random walk) and can produce fine-grained recommendations that can be non-integer values. Whereas, the MoM approach forces a specific integer rating to be assigned and hence is not as representative of the user's preference.

The WA framework is very effective, adaptable to any user-based similarity and a great starting point for developing a recommendation system. The downside to the framework in our implementations is its time complexity, to compute the rating for a single user we have a complexity of $O(n*s)$, where n is the number of users in the entire graph and s is the complexity of computing similarity between each user. Something like a Random Walk on the other hand is constant in complexity and can be used on arbitrarily sized networks. There are ways to optimise this. For example, for bi-directional similarity scores, the similarity between u_i and u_j should only be computed once. In terms of parallelising, each user prediction is independent, so this should be a straightforward task. Large optimisations can be found in vectorising certain parts of the algorithm, i.e. finding abstractions that facilitate the use of matrix operations.

When it came to the intra-item information, we found it performed quite poorly from the perspective of μ_{MAE}. However, an interesting observation, as per Table 12 in the appendix, was that the algorithms that included intra-item information were the most consistent in their performance. Featuring the lowest σ_{MAE} across all data sets. Thus it can be deduced that the intra-item information is additive from a stability perspective, making a recommender perform with similar accuracy for all users.

This idea was reinforced when we created the fully combined model, which utilised all information types. This was the "Jaccard Item-Jaccard JII Combination WA" model. We found this model performed worse than the Jaccard Item-Jaccard WA model in μ_{MAE} (0.80 vs 0.73), however, it performed better in mean σ_{MAE}, where the recommenders achieved 0.19 and 0.21 respectively. Hence, in a scenario where providing consistently good predictions for all users is of importance, the introduction of intra-item information could facilitate this.

Through our experimentation with opposing intra-item similarity metrics (see 3.3.1), we determined that the Intra-Item Jaccard approach outperformed the Pearson Correlation metric on downstream tasks. This was determined by building models upon these metrics and comparing the resulting accuracy. When further testing with the Intra-Item Jaccard similarity metric was undertaken, the resulting models performed better than random, and in combination led to stable models as described earlier.

A recurring theme in these experiments was the very impressive performance of the Jaccard Index when used in a variety of applications. It was the go-to for drawing similarity scores using all information forms and is a simple but logical way to reason about the similarity of sets. We do believe that Pearson Correlation could potentially perform better on larger data sets, as it seemed to be effective in [8]. An interesting further step would be to trial models that have relied on this similarity metric in the past with our proposed “Intra-Item Jaccard”, as from our results it seems to be more performative.

Testing the performance of our algorithms on an adversarial network provided some interesting insights. The most obvious of which is that item rating information is highly susceptible to an attack with fake accounts. The trust graph is relatively robust to adversaries as one must influence individuals to shift the dynamics of this information. This is represented in the data by the recommendation systems involving the item rating information suffering from the most severe reductions in their MAE. We found that the least impacted algorithm was the MC Random Walk. As the algorithm propagates from a user, through the network to derive the rating, it will arrive at a legitimate node with a rating before venturing into the adversaries. The only case where the adversaries have an impact is when no ratings are found and the randomness means a rating is taken uniformly from the network. In these cases the volume of ratings from adversaries impact results.

The next best algorithm for being robust in the face of adversaries was Jaccard WA. This is a Weighted Average based entirely on the Jaccard Index. This algorithm performs better due to the value it places on the trust graph, very rarely do nodes have a non-zero Jaccard similarity with the bad actors hence the high performance. We also noted that the full combination model suffered less of a performance reduction than its Jaccard Item-Jaccard and Item-Jaccard counterparts. This is of note as it seems Intra-Item information assists in the consistency of algorithms in both adversarial and non-adversarial environments.

A conclusion we can draw from this information is that reliance on the trust graph is paramount when there can be adversaries. In these cases, we have seen that Random Walk based recommenders are the most robust, with purely trust-based recommenders like Jaccard WA being second best. If we can imagine a sliding scale between item rating information and trust information, we achieve the best performance in organic environments from the item information and as we shift to the trust information we gain models that are more robust to adversarial attacks as they are upheld by hard-to-cheat structure of the social network.

9 Conclusion and Future Research

Through our ground-up approach to trust-based recommendation systems we have successfully gained an understanding of how to use different information types present in social networks, the contexts they perform in and how they combine to be useful in preference prediction. We found that item-rating information was the most important form of information when determining similarity between users. When used in a collaborative filtering approach to recommendation systems, it outperformed the other information types. The trust graph did not perform as well as item-rating information for the average MAE but led to recommenders that were more robust to adversarial attacks on the network as they were reliant on the hard-to-cheat trust structures in the network. Intra-item information performed quite poorly on its own, however, when combined with other forms of information it made recommenders more consistent in their predictions and led to better lower-end performance. We determined that the more constrained a recommender is to trust information the less it is influenced by adversaries. The best approach in these scenarios is Random-Walk recommenders as they make use of the graph structure in a way as to rarely be influenced by nodes with little network interaction.

We also proposed a framework for designing recommendation systems we call the "Weighted Average" approach. This approach allows us to use any user-to-user similarity metric and construct a recommendation system around it. It is simple and as evidenced by our results, highly effective. Based on this framework, our highest-performing recommender resulted from a combination of the trust information and item information. It was the Jaccard Item Jaccard WA recommender, which combines the trust-based Jaccard index along with the item-based Jaccard index to produce user-to-user similarities that when used in the WA context, achieved a mean MAE of 0.72 across our validation data sets, out-performing all of our other recommenders in MAE in all cases. Another algorithm we propose, dubbed Jaccard Item Jaccard JII Combination WA, includes intra-item information alongside this recommender, which improves the standard deviation of results by 10% whilst leading to an 11% reduction in average MAE. Meaning it provides a trade-off of average accuracy for consistency in the context that this is of importance.

In the case of the cold start problem, where users have recently joined the system, have few friends and are yet to cast any ratings, we found either Jaccard WA or Universal Random was the best approach. However, this was the lesser of evils and nothing performed exceptionally in these scenarios.

When investigating intra-item information we discovered that an "Intra-Item Jaccard" metric was more effective than Pearson Correlation in determining our similar two items were. This was evidenced by recommenders performing better when using the former rather than the latter. This result prompts further exploration into the well-renowned Trust-Walker paper, it is possible that introducing this similarity metric could lead to improved performance [8].

Another area of further research is further improving the WA approach in its computational complexity/efficiency. The upside to random-walk methods is their constant run-time, however WA based recommenders must iterate over all nodes in the graph, which leaves room for many potential optimisations such as vectorising the algorithm or caching similarities between users which are equal in both directions instead of recomputing.

We also believe that further research into the robustness of various recommender approaches to adversarial networks is of importance. In this paper, we considered the case where an adversary can make fake accounts with fake ratings. However, there are other forms of adversaries, such as bribing popular nodes to change their opinions or even masking popular nodes through adversarial censorship. These different attacks may lend themselves to different recommenders and approaches and are worth investigating.

10 Appendix

10.1 Performance Incl. Std-Deviation

Table 12: Mean and Standard Deviation of MAE for All Algorithms on Different Data Sets

Algorithm	Epinions		FilmTrust		CiaoDVD	
	μ_{MAE}	σ_{MAE}	μ_{MAE}	σ_{MAE}	μ_{MAE}	σ_{MAE}
Jaccard Item-Jaccard WA	**1.00**	0.26	**0.66**	0.08	**0.53**	**0.28**
Item-Jaccard WA	1.02	0.23	0.67	0.08	0.58	0.32
Jaccard WA	1.05	0.25	1.14	0.08	1.73	0.36
WIRD WA	1.05	0.28	0.67	**0.04**	0.72	0.37
Jaccard Item-Jaccard JII Combination WA	1.07	0.22	0.69	0.06	0.63	0.30
JWIRD WA	1.09	0.27	0.67	0.07	0.72	0.40
Jaccard MoM	1.13	0.30	1.19	0.09	1.77	0.37
Jaccard Monte-Carlo Random Walk	1.14	0.22	1.20	0.08	1.81	0.49
Monte-Carlo Random Walk	1.16	0.23	1.20	0.08	1.82	0.35
Item-Rating Difference WA	1.17	0.49	0.67	0.08	0.79	0.41
Jaccard Intra-Item WA	1.20	**0.13**	1.07	0.06	1.75	0.32
Intra-Item WA	1.24	0.14	1.18	0.08	1.70	0.45
Median of Neighbours	1.27	0.19	1.26	0.08	1.76	**0.28**
Mean of Neighbours	1.27	0.25	1.25	0.07	1.67	0.43
Universal Random	1.30	0.38	0.89	0.11	0.72	0.58
Jaccard Weighted Neighbours	1.31	0.32	1.22	0.10	1.66	0.44
Mode of Neighbours	1.32	0.23	1.23	0.09	1.67	0.43
Intra-Item WA (Pearson)	1.34	**0.13**	1.21	0.07	1.67	0.50
Random	1.67	0.21	1.34	0.07	1.82	0.31

10.2 Algorithm List

There are many algorithms and names thrown around in this report so it is hard to keep track of which algorithm does what. This section provides a table of algorithm names and a short description of their action.

Algorithm Name	**Information Type**	**Description**
Mode, Median, Mean of Neighbours	Trust	This class of algorithms simply takes the neighbours ratings for the item we are predicting for if they exist and aggregates them to produce a rating via either mode, median or mean.
Jaccard Weighted Neighbours	Trust	We initialize five rating buckets to 0. We iterate through the neighbours, if a neighbour has rated the current item we contribute the jaccard index between the current user and this neighbour to the corresponding rating bucket. We then perform a weighted average of these rating buckets. This is a simplified form of the Weighted Average algorithm.
Monte-Carlo Random Walk	Trust	A Monte-Carlo random walk. Starts at the node we are predicting for and traverses along trust edges. After each step the probability of returning a random rating increases
Weighted Average	-	This is a method of producing rating with a given similarity metric. First we maintain 5 rating ``buckets". For each user, we compute the similarity with every other user and contribute the similarity to the rating bucket corresponding to the other user. We then compute the weighted average of these rating buckets.
Jaccard WA	Trust	Uses Jaccard Index as the similarity metric for a weighted average computation as described above.
Item-Jaccard WA	Item-Rating	Uses Item-Jaccard index as the similarity metric for a weighted average computation. Item-Jaccard for two users is the intersection of the items they've rated divided by the union.
Item-Rating Difference WA	Item-Rating	Computes the average amount that two user's ratings differ from one another based on the intersection of the items they've rated. Then uses this similarity metric to perform WA.

Continued on next page

Table 12: (Continued)

Intra-Item WA	Intra-Item	Computes the similarity of two items based on the intersection of users who have rated both items, divided by the union of those who have rated either. To produce a rating we iterate through a user's neighbours, and for each neighbour iterate through the items they've rated. We then compute the similarity between the item we are predicting for and this rated item and contribute its similarity to the bucket corresponding to what the rating provided by the neighbour.
WIRD WA	Intra-Item Item-Rating	This combined both the intra-item information and the item information. To determine the similarity between two users, we take a similar approach to to the Item-Rating Difference but instead we consider every item u_2 has rated. For each of these items we multiply the similarity of the item to the one we are predicting for and the difference in our ratings. We use $J_I I$ for intra-item similarity. This similarity metric for two users is then used in a WA context.
Jaccard Item-Jaccard WA	Trust Item-Rating	This model combines the Jaccard and Item-Jaccard similarity metrics to produce a single similarity score for any pair of users. In the report we discovered the best way to combine these metrics is via equally weighted addition. Using this combined similarity metric we then use the WA algorithm to produce a rating.
Jaccard Item-Jaccard JII Combination WA	Trust Item-Rating Intra-Item	This model combines all of the information types. We start off with the similarity between our current user and another user u_2. We then iterate through u_2's rated items and contribute the user similarity multiplied by the item similarity between the item we are predicting for and the item we are looking at to the rating buckets. We then use WA to produce a final rating.

10.3 Algorithm and Metric Complexities

N number of users in graph, n number of neighbours for a given user, i is the number of items reviewed by a user, s is the complexity of a similarity metric.

Metrics (computing metric for one pair of items/users)

- Jaccard Index: $O(n)$
- Item-Jaccard Index: $O(i)$
- Item Rating Difference: $O(i)$
- Intra-Item Jaccard: $O(N)$

Recommenders (computing recommendations for a single user)

- Weighted Average: $O(N * s)$
- M's of Neighbours: $O(n)$
- Random Walk Monte-Carlo: $O(\frac{1}{\alpha})$

10.4 Metric Psuedocode

Item Rating Difference

```
function itemRatingDifference($graph$, $ratings$, $user1$, $user2$)
    $rating \leftarrow 0$
    $user1Items \leftarrow set(ratings[user1].keys())$
    $user2Items \leftarrow set(ratings[user2].keys())$
    $itemIntersection \leftarrow user1Items \cap user2Items$
    for $item$ in $itemIntersection$ do
        ratings $+= 1 - |ratings[user1][item] - ratings[user2][item]|/4$
    end for
    return $rating/max(|itemIntersection|, 1)$
end function
```

Item-Jaccard Index

```
function itemJaccardIndex($graph$, $ratings$, $user1$, $user2$)
    $rating \leftarrow 0$
    $user1Items \leftarrow set(ratings[user1].keys())$
    $user2Items \leftarrow set(ratings[user2].keys())$
    return $\frac{user1Items \cap user2Items}{user1Items \cup user2Items}$
end function
```

10.5 Algorithm Pseudocode

Mode of Neighbours

Algorithm 1 Mode of Neighbours

```
function modeOfNeighbours($graph$, $ratings$, $item$, $user$)
    $surroundingRatings \leftarrow []$
    for $v$ in $graph[user]$ do
        if $item$ in $ratings[v]$ then
            $surroundingRatings$.push($ratings[v][item]$)
        end if
    end for
    if $surroundingRatings.length == 0$ then
        return randomInteger(1, 5)
    else
        return mode($surroundingRatings$)
    end if
end function
```

Weighted Average Framework

Algorithm 2 Weighted-Average

```
function weightedAverage($node$, $item$, $graph$, $ratings$, $similarityFunction$)
    $buckets \leftarrow Array(5)$
    for $otherVertex$ in $graph$ do
        if $otherVertex$ in $ratings$ && $otherVertex \neq node$ then
            $idx \leftarrow ratings[otherVertex][item] - 1$
            $buckets[idx] \leftarrow buckets[idx] + similarityFunction(current, otherVertex)$
        end if
    end for
    if $potentialRatings.sum == 0$ then
        return $ratings$ randomInteger(1, 5)
    end if
    $WA \leftarrow 0$
    for $i$ in $range(5)$ do
        $WA \leftarrow WA + buckets[i] * (i + 1)$
    end for
    return $WA$
end function
```

Monte Carlo Random Walk

Algorithm 3 Monte Carlo Random Walk

```
function monteCarloRandomWalk($node$, $item$, $graph$, $ratings$, $k$)
    function traverse($node$, $distance = 0$, $\alpha = 0.05$)
        if $node$ in $ratings$ then
            return $ratings[node]$
        end if
        if random(0, 1) $< d \times \alpha$ then
            return $traverse(random.choice(ratings)])$
        end if
        $next \leftarrow graph[node]$
        if $|next.neighbours| == 0$ then
            return None
        end if
        return $traverse(random.choice(next.neighbours)), d + 1)$
    end function
    $results \leftarrow []$
    for $i$ in range($k$) do
        $sampleRating \leftarrow traverse(node)$
        if $sampleRating == None$ then
            $results.push(random(1, 5))$
        else
            $results.push(sampleRating)$
        end if
    end for
    return $mean(ratings)$
end function
```

References

[1] Vito Walter Anelli, Yashar Deldjoo, Tommaso DiNoia, and Felice Antonio Merra. Adversarial recommender systems: Attack, defense, and advances. *Recommender Systems Handbook*, page 335–379, Nov 2021.

[2] Yashar Deldjoo, Tommaso Di Noia, and Felice Antonio Merra. A survey on adversarial recommender systems. *ACM Computing Surveys*, 54(2):1–38, Mar 2021.

[3] Chen Gao, Yu Zheng, Nian Li, Yinfeng Li, Yingrong Qin, Jinghua Piao, Yuhan Quan, Jianxin Chang, Depeng Jin, Xiangnan He, and Yong Li. A survey of graph neural networks for recommender systems: Challenges, methods, and directions. *ACM Transactions on Recommender Systems*, Jan 2023.

[4] Lesly Alejandra Gonzalez Camacho and Solange Nice Alves-Souza. Social network data to alleviate cold-start in recommender system: A systematic review. *Information Processing Management*, 54(4):529–544, Jul 2018.

[5] G. Guo, J. Zhang, D. Thalmann, and N. Yorke-Smith. Etaf: An extended trust antecedents framework for trust prediction. In *Proceedings of the 2014 International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2014.

[6] G. Guo, J. Zhang, and N. Yorke-Smith. A novel bayesian similarity measure for recommender systems. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2619–2625, 2013.

[7] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[8] Mohsen Jamali and Martin Ester. Trustwalker: A random walk model for combining trust-based and item-based recommendation. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’09, page 397–406, New York, NY, USA, 2009. Association for Computing Machinery.

[9] Mohsen Jamali and Martin Ester. A matrix factorization technique with trust propagation for recommendation in social networks. *Proceedings of the fourth ACM conference on Recommender systems - RecSys ’10*, 2010.

[10] Xuan Nhat Lam, Thuc Vu, Trong Duc Le, and Anh Duc Duong. Addressing cold-start problem in recommendation systems. *Proceedings of the 2nd international conference on Ubiquitous information management and communication - ICUIMC ’08*, 2008.

[11] Xinyuan Lian and Dengji Zhao. A network-based rating mechanism against false-name attack. *Springer eBooks*, page 216–227, Jan 2022.

[12] Blerina Lika, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. Facing the cold start problem in recommender systems. *Expert Systems with Applications*, 41(4, Part 2):2065–2073, 2014.

[13] Benjamin Marlin. *Collaborative filtering: A machine learning perspective*. University of Toronto Toronto, 2004.

[14] Paolo Massa and Paolo Avesani. Trust-aware recommender systems. *Proceedings of the 2007 ACM conference on Recommender systems - RecSys '07*, 2007.

[15] John O'Donovan and Barry Smyth. Trust in recommender systems. *Proceedings of the 10th international conference on Intelligent user interfaces - IUI '05*, 2005.

[16] Oliver Richters and Tiago P Peixoto. Trust transitivity in social networks. *PloS one*, 6(4):e18384, 2011.

[17] Wanita Sherchan, Surya Nepal, and Cecile Paris. A survey of trust in social networks. *ACM Computing Surveys*, 45(4):1–33, Aug 2013.

[18] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009, 2009.

[19] Bo Yang, Yu Lei, Jiming Liu, and Wenjie Li. Social collaborative filtering by trust. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(8):1633–1647, Aug 2017.