Expanding Trust-Based Avenues for Social Recommendation

Viswajith V, *CS11B028*
Ayush Bhargava, *CS11B056*
Sriram V, *CS11B058*

***Abstract*—Recommender systems have been around for a few years now, to help suggest to users relevant information, and bring to their notice products which are likely to be of interest to them. Up until recently, such systems, which most commonly employ Collaborative Filtering, have always worked under the assumption of users being uncorrelated, except for information about their ratings of various products. We now show how we can use external information – namely, information about social connections between users – to aid our recommendation process, and help us build recommendation systems that perform better. Based on information provided by each user about who all he or she trusts, we extract connection information between users, and factor this into our predicted rating. Moreover, we make our recommender system discriminate by category – different users are likely to provide more helpful ratings for different categories of products, and we capture this information using a measure that we term "experience". We built a model that incorporated all this information, and tested it empirically on the freely available Ciao dataset, and evaluated our performance based on standard metrics employed in this domain.**

***Keywords*—Recommender systems, Social Network Analysis, Trust, Expertise, Collaborative Filtering**

I. Introduction

RECOMMENDER systems have been doing the rounds for quite some time now; in parallel with the rapid growth of the internet-savvy population, there has been a surge in online shopping, and sites that sell products to their users need a way of pushing content to the users. Since the time that each user spends on a website is limited, websites have very few shots at presenting products to users, and it is of crucial importance to them that a reasonably large fraction of the products they present to a particular user are those of interest to him or her. There are two major approaches to recommender systems that have traditionally been taken: content-based recommendation and collaborative filtering. Collaborative filtering requires users to rate products; it uses this information to infer which users show similar preferences, and makes appropriate recommendations.

With the proliferation of social networks, however, there is now an external source of information which is of great utility for the recommendation process: information about social connections between users. In the Ciao dataset we experimented on, users were allowed to mark certain other users as 'trusted'. If a user explicitly states that he or she trusts another user's ratings, then collaborative filtering need look no further to identify similar users. However, such trust relations are likely to be sparse, as they require effort from individual users, but that's not the end of the story – by examining transitive links and giving them importance based on their length and the extent of reinforcement (the idea being that multiple paths of trust link between two users means their trust value is higher).

However, a problem with both the traditional user similarity matrix and the trust-based similarity matrix is that they are both very sparse. We propose a method of normalising and combining the two measures consistently, so that if they are both present for a particular pair of users, we incorporate them both. If only one of them is present, of course, we're obviously forced to use just that one – this is still better than nothing, and the user coverage for the combined measure will be significantly larger, being the union of that in the individual cases.

Another aspect to keep in mind, which we propose to incorporate into our model, is that different users will, for the most part, know more about certain domains than others. In the specific example of the Ciao dataset we used, products are split into *categories*. Based on the available data, we define and calculate the experience that a particular user brings to products of a particular category, and show how to use this information to form better predictions.

The rest of this report is organised as follows. Section II looks at recommender systems and the technique of collaborative filtering, elucidating on its formal definition and common evaluation metrics used in this domain. Section III talks about bringing social information into the framework of recommender systems. Section IV is a brief summary of existing models from the literature that we have implemented to provided a baseline for comparison. Section V is a detailed description of the enhanced model we propose. Section VI is a report on the experiments conducted on the dataset and a summary of the results obtained. Finally, section VII concludes our report, and lists out directions for future work.

II. Recommender Systems and Collaborative Filtering

AS we mentioned in our discussion above, recommender systems aim to identify products that are likely to be of relevance to a particular user. The two main approaches that they have traditionally taken are:

- **Content-based:** This approach looks at the products that a user has shown a preference for in the past, and suggests products that are similar as computed by an appropriate measure. The major hurdle to such an

approach is identifying how similar products are – in most domains, such judgements are very subjective, and there is often no clear or consistent way to define it mathematically.

- **Collaborative Filtering:** The more commonly used approach, this requires users to explicitly express opinions on products by *rating* them. The recommendations are then produced based on the ratings of users similar to a given user – a similarity that is much more well-defined and readily computable than the content similarity required in content-based recommender systems. However, it requires every user to contribute with his or her ratings, failing which collaborative filtering has nothing to go on. Fortunately, users in social networking sites are quite willing to contribute to the community, which means that this is definitely feasible in practice; further, empirical studies are definitely facilitated by the fact that there are plenty of available datasets enumerating users and the rating they assigned to various products.

We look at collaborative-filtering based models, and propose extensions to the same.

A. Formal Definition

The formal definition of collaborative filtering-based recommender systems is fairly straightforward.
We have a user base $\mathbb{U}$, consisting of N users, and a product listing $\mathbb{P}$, consisting of M products:

$$\mathbb{U} = \{u_1, u_2, \ldots, u_N\}$$
$$\mathbb{P} = \{p_1, p_2, \ldots, p_M\}$$

We have a discrete set of K rating values that a user can rate products into – for example, it could be a five point scale.

$$\mathbb{S} = \{r_1, r_2, \ldots, r_K\}$$

The input to our recommender system $\mathbb{RS}$ is, then, an $N \times M$ rating matrix, $\mathbb{R}_{N \times M}$, whose each element is either from $\mathbb{S}$ or is a special number z to indicate that the user has not rated the particular product. That is:

$$r_{ij} = \begin{cases} r \text{ such that } r \in \mathbb{S} & \text{if } u_i \text{ has rated } p_j \\ z & \text{otherwise.} \end{cases}$$

The task of our recommender system is to use this matrix to suggest to a given user u_i a list of products that he or she is likely to find relevant. For ease of mathematical understanding, we can chalk out the following largely equivalent task to our recommender system. Our recommender system functions as a black box, which takes in as input the matrices mentioned above, and outputs, for every user u_i and product p_j such that $\mathbb{R}_{ij} = z$, a *predicted rating* for the given user-product combination. Once this is done, the interface presenting content to the user has left only the simple task of picking out the products for which the user's predicted ratings are highest, and recommending them to the user.

B. Evaluation Metrics

The following standard metrics are used to evaluate the performance of recommender systems, and it is in these terms that we have presented the performance of our model. We must keep in mind that we are evaluating the model here based purely in an *offline* manner on our training set. In practice, there will always be available online test feedback in the form of user ratings on recommended products. However, since the only ratings we have are the ones we train our model on (namely, the elements of $\mathbb{R}$ such that $\mathbb{R}_{ij} \neq z$), we have only them to estimate our error on. However, we are likely to obtain a good enough approximation of the actual error, because our metrics have been designed to be robust to this to the extent possible.

1) Mean Absolute Error: This is a leave-one-out technique. It involves training a model on the entire training set, and using it to predict ratings. The MAE is then evaluated by averaging the absolute error between the predicted and actual rating on the set of available ratings in R.

$$PR = \text{Predicted rating matrix}$$
$$A = \text{Set of available ratings} = \{(i, j, \mathbb{R}_{ij}) | \mathbb{R}_{ij} \neq z\}$$
$$MAE = \frac{1}{|A|} \sum_{(i,j,\mathbb{R}_{ij}) \in A} (|PR_{ij} - \mathbb{R}_{ij}|)$$

2) Mean Absolute User Error: In the specific context of recommender systems, a frequent observation is that the errors tend to be smaller for *heavy raters* (users who have rated many items), and significantly larger for *cold start users*, who are new to the website or environment in question, and have not rated many items. But since heavy raters provide many ratings, their small errors will be counted many times in our average, and will make the MAE an underestimate of the actual extent of error produced. Thus, optimising for MAE may end up making our system predict near perfectly for a few heavy rater users, while performing abysmally for everyone else. From the point of view of the website, which seeks to maximise profit, such a system will not be very effective.
Hence, we also evaluate our models on an alternative metric, MAUE, where we calculate the MAE per user and average this out, so that our estimate is not biased towards the error of heavy raters.

$$\text{MAE}_i = \text{MAE for user } u_i \text{ for } i = 1, 2, \ldots, N$$
$$\text{MAUE} = \frac{1}{N} \sum_{i=1}^{N} \text{MAE}_i$$

3) Users Coverage: In the same spirit as MAUE, this metric is based on the observation that recommender systems are required to get accurate predictions for as many users as possible. To this end, user coverage is used – it is defined simply as the portion of users for which the recommender system is able to predict *at least* one rating.

C. Disadvantages

Collaborative Filtering-based recommender systems as discussed above have a few major drawbacks:

- **Sparseness of $\mathbb{R}$:** The matrix of ratings is very sparse, as each user would have rated at most only a handful of products. The only way for us to identify similar users is if there are a minimum number of products that they have *both* rated – with how sparse the typical dataset is, however, there will be few enough users even if we set the threshold for the minimum number of common products as 1!
- **Cold start problem:** This is related to the above issue, but is specific to new users – until a user has rated a sufficiently large number of items, we will not have very much information about him or her as we cannot identify similar users, and our recommender system will perform poorly for all users, at least the first few times they use the site.
- **Non-discrimination between products:** An advantage of content-based systems is that they identify intrinsic similarity between products, something which collaborative filtering-based methods do not do directly at all. This means that these systems are completely ignoring a very crucial aspect of user ratings: different users are likely to be better at rating different kinds of products.

To sum up, most of the issues with collaborative filtering-based systems are because they have only one source of information – product ratings by different users. They do not incorporate intrinsic similarities between products or between users, which could possibly be obtained through external means. In the following section, we examine the latter – exploiting social information to identify similar users that wouldn't have been captured by just looking at common products they rated.

III. Incorporating Social Information

SOCIAL networks have proliferated over the last few years, and users have spent a lot of time online. Different sites provide different mechanisms for users to concretize their interpersonal relationships online – these range from the Facebook friend relation and the Twitter follow relation to the "trust" relation in the Ciao dataset which we ran our experiments on.

Incorporating this information into our collaborative filtering framework is quite straightforward – where earlier, we computed similarities between users by looking at instances where they both rated the same product, we now use the user-provided information itself, which is likely to provide a much better estimate of their similarities!

This is not a magic solution to all our problems, however. In particular, if we utilise only user-provided relationships directly, the sparseness problem is even starker than before – since identifying other users they trust is effort on the part of the user, we cannot expect individual users to mark too many others. Fortunately, it is possible to do better than this with our available data. In particular, let us look at the trust relationship in the Ciao dataset, and how we can leverage it to extract as much utility as possible for our recommender system.

A. Understanding Trust Relationships

In the Ciao dataset, to facilitate better recommendation, users are asked to identify other users whom they trust – or, to quote the website, "reviewers whose reviews and ratings they have consistently found to be valuable". This is an asymmetric, very distributed relationship, but the size of connected components is likely to be very large – we empirically verified in our dataset that there is a giant strongly connected component of trust relationships, even though each user provides only a handful of other users that he or she trusts. Now, the question comes up: how can we exploit this?

B. Extended Trust Networks

Let's say we have three users, A, B and C. A says he trusts B, who says he trusts C. These are the only explicit trust relations that are available to us. Intuitively, it seems likely (and it has been verified empirically) that A will transitively trust C. It is also less likely, but nevertheless worth thinking about, that the reverse relations will hold – namely, that B trust A and that C trusts B. If we incorporate both the extensions, then we could even conclude that C trusts B. We can differentiate the different edges we've added by *weighting* them – obviously, that the $A \to B$ and $B \to C$ edges should be given the highest weights since they are explicit. Transitive and symmetric induced relations can be given lower weights, the exact magnitudes of which must be determined empirically.

1) Propagating Trust: Propagating trust to form an extended trust network is computationally fairly simple – take a trust matrix $\mathbb{T}_{N\times N}$, where N is the number of users, whose each element is given by:

$$\mathbb{T}_{ij} = \begin{cases} 1 & \text{if } u_i \text{ has stated that he or she trusts } u_j \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

Based on elementary graph theory, it follows that $\mathbb{T}^n_{ij}$, the element at (i, j) of the matrix $\mathbb{T}$ multiplied by itself n times, is equal to the **number of trust paths between users u_i and u_j**. Thus, we simply need to multiply $\mathbb{T}$ by itself n times, and check for non-zero values in the matrix to test for transitive relations at a distance n. We can keep adding relations up to a maximum distance N, and assigning them different weights.

2) Assigning weights: Assigning weights is difficult – although there are many intuitively appealing methods, it is computationally expensive to evaluate our model with different weights, and there is no guarantee that what works well for one model works well for another. The heuristics we followed kept in mind the following things:

- While there is a clear argument for the inclusion of transitive edges, it is difficult to estimate just how far this will hold: if u_1 trusts u_2, who trust u_3 and so on up to u_n for a large value of n, can we say with any sort of certainty that u_1 is likely to trust u_n? Definitely not. The effectiveness of the transitive link we have drops off rapidly with the length of the path; exactly how fast is unclear. We experimented with two approximations, which gave similar performance on our dataset. First, we approximated it to be linear, but cut off transitive

links at a maximum distance of 4 edges – this means, our weights assigned are $1, 0.75, 0.5$ and 0.25. We also tried allowing links at larger distances than 4 (this will, if nothing else, make our final trust matrix considerably less sparse), but dropping off weights in accordance with an inverse polynomial law.

- Instead of merely checking for non-zero values, we could also examine their exact magnitude – for example, if, for a particular pair of users u_i and u_j, if $\mathbb{T}_{ij} = 0$ but $\mathbb{T}^2_{ij} = 20$, it means that there are 20 users who u_i trusts, who trust u_j. This means the transitive edge we induce must be given significantly more weight than another edge whose entry was 1 instead of 20. We incorporated this multiplicatively into our model, but cut off our scaling at 5 – that is, we treated all values above 5 as 5 itself. This was based on the intuition that the effect just described is likely to plateau out at some level, and there is no point looking at the exact magnitude – which was doing more harm than good, because some extremely high values ended up dwarfing out the rest in comparison.

3) Is trust (approximately) symmetric?: How likely is it that u_j trust u_i, given that u_i trust u_j? This question is an interesting research problem by itself, and exploring it to the full extent possible was not within the scope of our project. However, its incorporation was not effected because it was found that our trust propagation procedure largely rendered it unnecessary – *most* symmetric relations in our dataset were in fact filled in through transitive edges. This works better than an unguided approximation on our part because it weights different back-relations with as good a measure as any, and completely ignores some others (which – again, purely on the basis of intuitony – makes sense).

IV. Baseline Models

KEEPING in mind all the definitions built up in the preceding sections, we present two collaborative filtering-based models from existing literature, which we implemented for our dataset to give us a baseline for comparison. Before going on to model-specific details, we must formalise how our system will predict ratings.

A. The Rating Predictor

As we formally defined earlier, collaborative filtering-based models have as input a set of users, a set of products, and some information on what certain users have rated certain products. We also assume that we have as input a similarity matrix, $\mathbb{S}$, whose computation will be model-specific. The matrix will have a number which is 0 if there is no information available about the users, or a weight that indicates how similar they are. The neighhbours of a user can immediately be defined as the set of users with whom a particular user has non-zero similarity. With all this in hand, we can predict the rating of user u_i for product p_j as:

$$PR_{ij} = \bar{r}_i + \frac{\sum\limits_{u \in \text{neighbours}(i)} \mathbb{S}_{iu} \left(r_{uj} - \bar{r}_u\right)}{\sum\limits_{u \in \text{neighbours}(i)} \mathbb{S}_{iu}} \tag{2}$$

where $\bar{r}_i$ is the average rating of user u_i across all the products he or she has rated and r_{ij} is the rating of user u_i to product p_j.

Now, we can look at our specific models, which differ only in how they produce the similarity matrix that they feed to the predictor.

B. User Similarity-based Model

This is based on the classic collaborative filtering approach – users are similar if they have shown similar tendencies on products they have rated. As discussed before, this is likely to be horribly sparse, as it requires users to rate a reasonably large set of identical products. Nevertheless, it is not uneffective, and in the absence of external information, it is probably as well as we can do. There are several proposed techniques to compute user similarity, the most common of which is the Pearson Correlation Coefficient, which is computed on overlapping items. Let there be m such items; then the similarity between users u_i and u_j is computed as:

$$\mathbb{S}_{ij} = \frac{\sum_{p=1}^{m} \left(r_{ip} - \bar{r}_i\right)\left(r_{jp} - \bar{r}_j\right)}{\sqrt{\sum_{p=1}^{m} \left(r_{ip} - \bar{r}_a\right)^2 \sum_{p=1}^{m} \left(r_{jp} - \bar{r}_j\right)^2}} \tag{3}$$

C. Trust-based Model

In this model, we simply feed in the Extended Trust matrix obtained by taking the trust relationships in the data and extending them to include transitive edges and weight them accordingly. The details of how this is done was discussed in the section on incorporating social information.

V. Proposed Enhanced Model

WE now discuss the new *expanded trust-based recommender system* we proposed and implemented. Our experiments were conducted largely on the Ciao and Epinions datasets.

A. Proposed Improvements and Motivation

Our model builds on the two existing baseline models proposed above, and improves on them in the following ways:

- We modified the trust-based similarity matrix to account for the number of paths, and modified the method of weighting we followed, as dicussed in the earlier section.
- Incorporation of **both** the user similarity matrix and the trust-based similarity matrix, after suitable normalisation consistent with their distributions. The motivation behind this is that both measures are effective in their own right, and we will, by combining them, do at least as well as by using the better of the two on its own. A

more convincing argument for doing this is that both matrices are by themselves sparse, and the non-zero similarity values we have now will be the union of the two earlier; since our predictor is dependent heavily on the neighbours of a particular user, sparseness reduction measures which consequently result in an increase in the number of neighbours will significantly improve our performance.

- Incorporating information about similarity of products, which was hitherto restricted to content-based models. We did this by splitting products up by category, using the information already available in our dataset. For each user, we define and evaluate a new term, namely "experience". We then augmented the trust-based similarity matrix with this experience information, because although the trust information does not incorporate categories, in real life, **users don't trust everyone with everything**.

B. Formal Description

1) User-based similarity matrix: This matrix was computed exactly as in the baseline model, based on the Pearson Correlation Coefficient.

2) Initial Trust-based similarity matrix: This matrix was based on the computation of the extended Trust-based network; the exact weights mentioned here are for the linear approximation case cutoff at a transitive link length of 4, although, as mentioned above, different schemes, including an inverse power law, were also tried out.

Formally, given a matrix $\mathbb{T}_{N\times N}$, where each element is given as in (1), we generate the output matrix $S_{\mathbb{T}}$ by the following procedure:

1) $S_{\mathbb{T}} = \mathbb{T}$ initially, since all direct links are given a weight of 1.
2) Multiply our running product matrix $\mathbb{T}'$ by $\mathbb{T}$. All elements which were zero until now, but which have become non-zero are given a weight in $S_{\mathbb{T}}$ of $1 - \frac{i}{4}$ multiplied by a factor based on how large the non-zero value is (as mentioned earlier, we cut off at 5, and ensure that link length is given highest priority by our weighting scheme – for example, even a value of 1 at link length 2 is given more weight than a value of 5 at link length 3), where i is the number of the iteration.
3) Repeat step 2) till we have iterated 4 times (or however many iterations we wish to have).

3) Experience Measure and Augmenting the Trust-based similarity matrix: It is quite likely that different users with different domains of expertise are likely to perform better when rating different kinds of products. Thus, we need to cluster the content – this can be done in multiple ways, depending on the availability of data. For instance, if the content is in the form of text or images, it can be converted to a machine-parsable representation and appropriate clustering algorithms can be employed. In the experiments performed as part of this project, we had a readily available clustered dataset available – products were classified into categories – and therefore, an examination of clustering techniques which would work well in such scenarios is beyond the scope of this project. In our dataset, every rating was also associated with a "helpfulness", a number based on user feedback, which was an integer from 1 to 7.

Using all this information, we defined a notion of experience in this domain (note that this is not, therefore, applicable in general to all datasets, but the intuition is simple, and constructing analogues based on data availability should be possible in most cases). The experience of user u_i for products in category c_m is computed as:

$$\mathbb{E}_{im} = \frac{\sum_{r\in\mathbb{R}_{im}} \text{Helpfulness}_r}{AH_m \times |P_m|} \tag{4}$$

where $\mathbb{R}_{im}$ is the set of ratings by u_i for products in category c_m, AH_m is the average helpfulness of ratings by *all* users in category m, and P_m is the set of products in category m.

The idea behind this experience measure is that, the higher the fraction of products in a particular category that a given user rates, the more likely he or she is to be an expert in that category. Things are not that simple, though – some users might just rate a lot even though they're bad. This is where we leverage the information linked to helpfulness that is available to us in the dataset; we are summing not the number of ratings, but their experience, and we are dividing it by the average helpfulness of ratings in that category. The term $|P_m|$ in the denominator also plays the added role of accounting for the fact that if a user has rated a given number of products, his experience is more valuable if the total number of products in the category is small.

We are now ready to augment our trust-based similarity matrix, and making it discriminate by category. We propose a simple multiplication to get our new measure of **experience-weighted trust**. Specifically, the experience-weighted trust of user u_i to user u_j in category c_m is given by:

$$\mathbb{ET}_{ijm} = \mathbb{T}_{ij} \times \mathbb{E}_{jm}$$

4) Normalising the matrices and combining them: A very tricky issue is combining the two similarity matrices we have. Their values are along different scales, and therefore, simply adding them does not make sense – they have to be normalised first. Normalising is itself a tricky proposition, however, as we must somehow preserve the properties of the distribution. Lastly, there is the question of how we combine the values when they are *both* present for a pair of users, and in the special cases (which are quite likely) where exactly one is present.

- **Normalisation:** It has been empirically determined that these similarity matrices are likely to follow a power-law distribution. Two immediately obvious normalisation solutions for general distributions are standardisation (subtract the mean from each element and divide it by the standard deviation) and min-max normalisaton (wherein we subtract the minimum element from each one and then divide it by the range). However, neither is particularly appropriate for a power law distribution.

Thus, we followed a bin-based approach, based on the following procedure:

1) Choose a large number of bins, B, which is in the order of the range of the variable.
2) Create B bins in the original range, and similarly B bins in our target range $[0, 1]$. There is now a one-to-one mapping between both.
3) For each value, figure out which bin in the original range it belongs to, and map it to the min or max of the appropriate bin in our target range.

Note that, for the augmented trust matrix, $\mathbb{ET}$, we normalise *category-wise*, because the power lab distribution is followed by the distributions *conditioned on the category*, and it would be incorrect to do a bin-based normalisation for the overall distribution and then split it up by category.

- **Combining them:** We take the normalised matrix values, and, to compute the final effective similarity between users u_i and u_j, we do the following:
  - If neither of the matrices have a similarity value, we don't have a final effective simlarity value either.
  - If only one of the values is present, we use that one.
  - If both the values are present, say, s from the user similarity matrix and t from the augmented trust-based similarity matrix, we linearly combine them to get effective similarity = $\lambda t + (1 - \lambda)s$. After empirically varying λ between 0 and 1 in intervals of 0.1, we found, for our dataset, that the best results were obtained for $\lambda = 0.8$.

C. Computational Complexity

Our models fair quite badly as far as complexity is concerned, as they involve pairwise measures between users. While operations like intersection could potentially be optimised by using specialised data structures, we did not do so in our experiments as our goal was the examination of the new model based on the performance of its recommendation; the code written, however, is quite easily modifiable, and can be made to work with alternate datasets without too much difficulty. As we have implemented them, though, our models are either $O(n^2)$ or $O(n^3)$ in terms of both time and space complexity, where n is used as an umbrella variable for all the three dimensions (users, products and categories) our data can be spread across. However, it is probably impossible to make these algorithms run in linear or sublinear time, which limits their scalability. Two immediate consequences of this are:

- Our model training itself, that is, the computation of all these matrices from the available rating data, has to be done in an *offline* manner – we cannot update our model incrementally as new ratings flow in; we will have to have a trained model to use for predicting ratings, which can be updated on occasion.
- Our predicted ratings cannot be all computed and stored feasibly, as it would require immense storage space – we have, after all, added a third dimension to the whole system. Therefore, these ratings also have to be computed *lazily*, as and when required. Since computing *all* the predicted ratings for a user before deciding on the content to offer him or her might be infeasible, we could look into picking the top products picked by a cheaper rating system (say, from a baseline model), and compute the more accurate rating that our enhanced model predicts for this subset.

VI. Experiments on Dataset

OUR experiments are based on the Ciao and Epinions datasets[1], both of which are information scraped from www.epinions.com. **Epinions.com** is a general consumer review site that was established in 1999, during the dot-com bubble, which permits visitors to read and write reviews on a variety of products.

A. Description of Dataset

The dataset consists of two matrices:

- **rating.mat:** Each data point (row) in this matrix corresponds to the rating of a particular product by a particular user. There are five columns, namely userid, productid, categoryid, rating, helpfulness. The rating is on a five-point scale, with 1 being worst and 5 being best.
- **trustnetwork.mat:** This has a row for every trust relation in the network, i.e, if u_i trusts u_j, then there will be a row with (i, j) in this matrix.

1) Size: The **rating.mat** file downloaded from the site contained 284086 instances of ratings, with 7375 unique users rating 105114 unique products belonging to 28 unique categories (which are tabulated in table I). The **trustnetwork.mat** file contained 111781 instances of trust relations between users (note that, with 7375 users, the theoretical maximum number of trust relations is $7375 \times 7374 = 54383250$ which is a coverage of 0.19%, or a sparsity of 99.81% – which seems quite detrimental to our cause, but this is as good as one can hope for in practice).

2) Preprocessing Performed: Since our experiments required us to frequently run code on our dataset after trying out various tweaks, we preprocessed the dataset as follows:

1) Since the crux of our new findings are in the information exploited from product similarity – namely, the categoryid provided in the dataset – we discarded categories which had very few instances in the rating matrix.
2) Since products which are rated by too few users are likely to make little difference to our results (our model will not be able to predict anything new for them), we discarded all ratings involving products without a threshold.

[1] you can find these at http://www.public.asu.edu/ jtang20/datasetcode/truststudy.htm

TABLE I. LIST OF CATEGORIES IN THE DATASET

Online Stores & Services
Games
Movies
Books
Music
Personal Finance
Electronics
Home and Garden
Computer Hardware
Hotels & Travel
Restaurants & Gourmet
Magazines & Newspapers
Software
Media
Cars & Motorsports
Education
Sports & Outdoors
Wellness & Beauty
Kids & Family
Musical Instruments
Business & Technology
Pets
Computers & Internet
Web Sites & Internet Services
Gifts
Preview Categories
Photo & Optics

3) Over these two steps, we lost quite a few users, and we proceeded to remove all rows involving them from the trust relations matrix.

At the end, our dataset was trimmed down considerably, but care was taken to ensure that the points we let go of were points that would not have done our training process very much good anyway. The final version used for our experiments, after preprocessing, had 216272 instances of ratings with 5008 unique users rating 78626 unique products belonging to 11 unique categories. The trust matrix had 91979 instances (with an analogous calculation to that used in the original matrix, the coverage works out to 0.36%, or a sparsity of 99.64%, a drastic improvement over what we had before, although this was not a consideration at all during our preprocessing. Why such an improvement was witnessed can probably be traced to the fact that users who have rated fewer products are likely to be newer or less active on the site, and are consequently likely to have identified fewer users as users that they trust).

B. User Similarity

The user similarity-based model was implemented by analysing products rated in common by the two users and evaluating the Pearson Correlation Coefficient (as in (3))[2].

[2]The code can be found in the file **userSimilarity.m**

C. Trust-based Similarity

The trust-based similarity was implemented[3] by computing transitive links in the provided trust network, based on the algorithmic procedure outlined in the section describing our proposed enhanced model.

D. Predictor

The predictor[4] takes as input the rating matrix and a similarity matrix from any source, and gives as output a matrix of predicted ratings, one for each input. The values are, in general, floats, which can for prediction purposes be rounded off to a whole number for ratings. However, we don't round them off initially, for two reasons:

- The ultimate function of the system we build is to suggest items to users, not predict ratings; thus, with more exact information on predicted ratings, we will know which items to recommend first, rather than arbitrarily picking one of the highest-rated items.
- It will give us better estimates of errors, thereby aiding our evaluation.

E. Enhanced Model

The enhanced model works by feeding in the combined similarity matrix with experience information into our predictor.

1) Incorporating category-based experience: The experience metric by category, as defined in (4) is computed in a straightforward manner[5]. Given the rating matrix, this function outputs an experience matrix, which has an $N \times C$ matrix, with each row detailing the experience of a particular user in various categories. The augmented matrix, which can theoretically be obtained by doing a pointwise multiplication of the trust-based similarity matrix and this experience matrix, is not precomputed and stored as it is too large – the mechanism for normalisation we employ is (or more specifically, is forced to be) *lazy* as detailed below.

2) Normalisation: As explained earlier, to combine the matrices, we had to normalise the values we obtained while somehow preserving their distribution, and for this, we followed the bin-based approach outlined earlier[6]. Since the sheer size of the combined matrix involved would be too large to precompute the entire thing (it would be three-dimensional, with $N \times N \times C$ elements, where N is the number of users and C is the number of categories, the code is set up to compute similarity *lazily*, and pass it on to the predictor as and when asked for.

3) Combining: Based on the straightforward rules for combining mentioned earlier, we linearly combine the values we obtain from different sources to get the final augmented similarity value for user u_i with user u_j for a product in

[3]The code can be found in the file **get_trust_matrix.mat**
[4]The code can be found in the file **rating_predictor.mat**
[5]The code can be found in the file **experience_mine.m**
[6]The code can be found in the file **power_law_normalisation.m**

TABLE II. EVALUATION METRICS FOR TRADITIONAL BASELINE MODELS

Metric	Model	Value
Ratings Coverage	User Similarity	0.43
	Trust	0.54
MAE	User Similarity	0.795
	Trust	0.767
MAUE	User Similarity	0.912
	Trust	0.874

TABLE III. EVALUATION METRICS FOR OUR ENHANCED MODEL

Metric	λ	Value
Ratings Coverage	N/A	0.71
MAE	0.25	0.784
	0.5	0.772
	0.75	0.759
MAUE	0.25	0.894
	0.5	0.876
	0.75	0.861

category c_m:

$$\mathbb{AS}_{ijm} = \begin{cases} \lambda(\mathbb{T}_{ij} \times \mathbb{E}_{jm}) + (1-\lambda)\mathbb{S}_{ij} & \text{When both are present.} \\ \mathbb{T}_{ij} & \text{When only } \mathbb{T}_{ij} \text{ is present} \\ \mathbb{S}_{ij} & \text{When only } \mathbb{S}_{ij} \text{ is present} \\ \text{Not present} & \text{When neither is present} \end{cases}$$

F. Results Obtained

Table II shows the results for the traditional baseline models for us to perform our comparison with, while table III shows the results obtained in our model for various values of the linear combination-influencing parameter λ. It is clear that, of the two baseline models, the model

VII. CONCLUSION AND SCOPE FOR FUTURE WORK

AS discussed above, the advancements that our model makes over previous models is two-fold:

- We combine two different kinds of similarity matrices using a novel normalisation procedure – the user similarity computed from their common ratings, and the trust-based similarity metric.
- We bring in a crucial aspect of content-based filtering which was hitherto absent from collaborative filtering models – we look at intrinsic similarity between products themselves, and make our recommender system discriminate user effectiveness based on cluster or category information.

We evaluated this new model on the Ciao and Epinions datasets, There are, however, several areas where our model can be improved on; some of them have been discussed inline in the report, but we explicitly list them out here for easy reference:

- **Working in the absence of category information:** In the absence of readily available category information like in the Ciao dataset, we can look into inferring categories from the data by representing it in a machine-readable format (there are established methods to do this for text and images, but not for all domains), and clustering the representation (again, there are a plethora of techniques available to do this).
- The baseline models we used are quite dated – of late, several models incorporating matrix factorisation and other more recent advances has come up. We ran our experiments on these old models for simplicity, since the aim of our assignment was not to implement the recommender system that solves all problems, but to examine the effects that the dual advancements we proposed have. The inherent assumption in this way of working is, of course, that the techniques for normalisation and combining, and incorporation of category information that we proposed, will extend quite naturally to more recent models as well.
- **When are trust relations symmetric?** This question came up when we were in the process of formalising the extended trust-based similarity matrix, and is an interesting research problem by itself. Given that u_i trusts u_j, what can we infer about the reverse relation? What external information can we use to aid this inference, how confident can we be about it, and what weight do we assign the reverse edge?
- **Extending this to other domains:** Our experiments are very closely tied to the specific dataset we used; it is also possible for us to extend this in general to social networks. For example, for orkut[7] to suggest communities to users, it could extract similarity measures of users based on communities they've liked, and by analysing the structure of the social network containing them (this will involve extensions of our trust-based similarity).
- **Figuring out optimal schedules for offline computation:** Due to the computational complexity of our model, it cannot work in an *online* manner, and it must keep training itself on available data at regular intervals, as opposed to updating itself as and when new ratings and trust relations are established. Figuring out good schedules for these periodic training phases, to get best performance while minimising computation, could be an interesting problem per se.

ACKNOWLEDGMENT

The authors would like to thank...

REFERENCES

[1] P. Massa and P. Avesani, *Trust-aware collaborative filtering for recommender systems*.

[2] Jamali, M., Ester, M., *Trustwalker: a random walk model for combining trust-based and item-based recommendation*

[3] Cho, J., *The mechanism of trust and distrust formation and their relational outcomes.*

[7] With specific reference to the dataset at http://snap.stanford.edu/data/com-Orkut.html

[4] Ma, H., Lyu, M.R., King, I., *Learning to recommend with trust and distrust relationships.*

[5] Victor, P., Cornelis, C., De Cock, M., Teredesai, A.M., *A comparative analysis of trust-enhanced recommenders for controversial items.*

[6] Victor, P., De Cock, M., Cornelis, C., *Trust and recommendations.*