A Comparison of Collaborative Recommendation Algorithms Over Diverse Data Types*

John O'Donovan

June 16, 2003

Abstract

Information filtering techniques are becoming more widely used as available information spaces grow exponentially larger. New techniques for filtering information are being developed to tackle the information overload problem. This paper presents an assessment of the performance of three popular recommendation stratagem over a range of diverse data. Our aim is to show that the relative performance of these algorithms varies as they are applied to different data. We run performance tests on Item-Based Collaborative Filtering, Pure Collaborative Filtering, and an Association-Rule Based Filtering algorithm. These tests consider neighbourhood size, and density of the training set. Our empirical results show provisionally that there is a significant difference in the relative performance of these algorithms over our four experimental data platforms.

1 Introduction

Recommender Systems are designed to help tailor a users information space by suggesting items the system believes will be of interest to the user. An increasing number of online stores provide recommender systems on their sites, e.g. E-Bay, Amazon.com etc. There are two main bases upon which these systems operate: Collaborative[9] and Content-Based Filters[7]. A Content-Based approach to the recommendation task compares similarities in descriptions of content of items to user profiles to arrive at its goal. In the Collaborative approach, which is the focus of this paper, users provide ratings for items in a particular domain, and the system exploits similarities and differences between users based on these to compute its recommendations: If users A and B rate k items similarly, they share similar tastes and should rate other items similarly. There are many implementations and variants of collaborative filtering techniques in todays recommender systems, all of

*This work is part of the INTINN project, from which the support of Enterprise Ireland is gratefully acknowledged.

which are restricted by the same fundamental drawbacks. Collaborative Filtering (CF) techniques do not require items to be machine-analysable, and can arrive at serendipitous recommendations, that is, they can recommend relevant items that are completely different from those in a users profile. They also require little knowledge-engineering overhead.[8] CF techniques are all subject to two serious restrictions. *Sparsity Problem*: In any given case, it is unlikely that two users have co-rated many of the items in the system. Accurate similarity measurements depend on rich user profiles with high overlap which can be costly to attain. *Latency Problem*: This affects new, uncommon, or esoteric items. These items will not be recommended by a system until they are included (if ever) in a sufficient number of user profiles[8].

We investigate these performance difficulties in four different experimental datasets, using tests for density of the data, and neighbourhood size. Using these metrics, we aim to show that there is a difference in the relative performance of our recommendation algorithms based on the type of data they are used on, and hence realise suitability of particular technique(s) to individual data types. For example, we would like to show that as the number of neighbours looked at for similarity computation increases past some threshold on a dataset of jokes, and one of movies, predictive accuracy of our algorithms will fall faster on the jokes set, as there are smaller clusters of people with similar tastes in jokes, than movies, (Peoples tastes in jokes tends to be more erratic than in movies).

If for example, we can show that predictive accuracy for the Item-Based (IBCF) algorithm peaks at a higher point than pure CF on jokes data, and a lower point on movie data, we can hence assert that there may be a suitability towards that data type for the IBCF algorithm. Of course the density of the training data will have a serious effect on the accuracy of the predictions, so we introduce density tests in our evaluation of neighbourhood sizes, and normalise across all datasets by multiplying all parameters by the density of each dataset. Our overarching goal in this paper is to uncover these performance differences and essentially uncover links between performance of recommendation techniques and data types.

2 The AdRec System

This paper is based on the AdRec System, an ongoing research project into hybrid recommendation techniques. The AdRec system is an analytical framework wherein several recommendation techniques can be crossed to ascertain optimal recommendations over different datasets. AdRec implements a machine learning strategy over information filtering techniques to arrive at optimal configurations for a recommendation engine. In addition to specific implementations, AdRec develops an information filtering architecture, which is modular and allows new technologies to be added as they arrive.

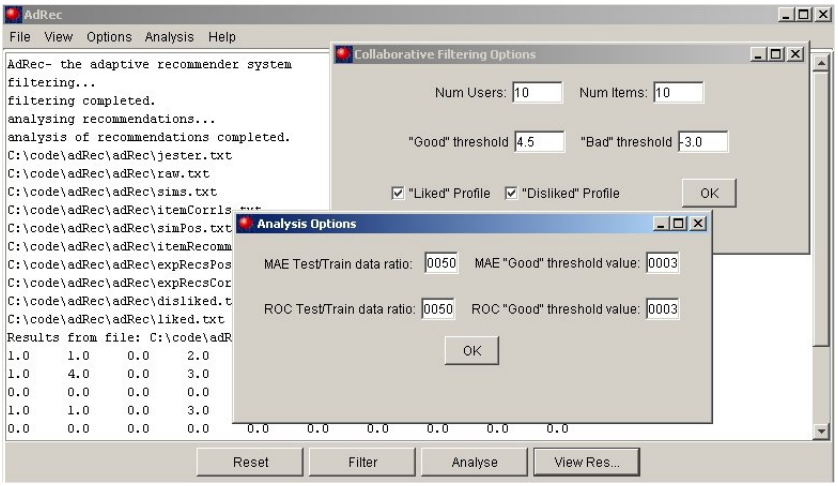

Figure 1: The AdRec Front-End

3 Pure Collaborative Filtering

This algorithm is the popular user-based CF, first introduced in 1989 by John Hey of LikeMinds, and furthered 1994 by the GroupLens[9] research group. Our User-Based CF is performed by calculating the Pearson correlation coefficient[4] of two vector representations of users, using this to define peer-groups, and suggesting items a users peers liked that has not yet been seen by the user. There are a range of methods for calculating similarity coefficients, such as Cosine-Similarity, Jaccard's coefficient etc. We use Pearson's as it is the most widely used in CF and allows us a more direct comparison with other existing systems. When the peergroup of users is chosen based on the similarity to an active user (the user receiving recommendations), a weighted combination their ratings is used to generate predictions, as seen in [7]. The following is a step-by-step outline of the CF algorithm.

1. Calculate the similarity between the active user and every other user.
2. Form the peer group by selection of the top-n most similar users.
3. Select items rated highly in the peers' profiles which are not in the active users profile.
4. Return a ranked list of these items as the recommendation set.

The following formula calculates the Pearson Correlation in the first step:

$$corr_{x,y} = \frac{\sum u \in U(R_{k,x} - \overline{R}_x)(R_{k,y} - \overline{R}_y)}{\sqrt{\sum u \in U(R_{k,x} - \overline{R}_x)^2 \cdot \sum u \in U(R_{k,y} - \overline{R}_y)^2}} \quad (1)$$

Where $corr_{x,y}$ is the Pearson correlation coefficient between user x and user y. $R_{k,x}$ is the rating of user x on item k, and R_x is the average item rating by user x.

Prediction calculation (step 3 above) is done using 2, which calculates the weighted average of deviations from the neighbours mean:

$$p_{x,i} = \overline{r}_x + \frac{\sum u \in U(r_{u,i} - \overline{r}_u) \times P_{x,u}}{\sum u \in U(P_{x,u})} \tag{2}$$

Where $p_{x,i}$ is the prediction for user x on item i, and $p_{x,u}$ is the similarity between users x, and u.

4 Item-Based Collaborative Filtering

The main difference between Item-Based CF (IBCF)[10] and the User-Based algorithm is that IBCF makes its predictions based on a model of Item similarity rather than user similarity. This algorithm looks at a set of items the active user has rated and computes their similarity to a target item i, in order to evaluate it for recommendation. The IBCF algorithm selects the k-most similar items $\{i_1, i_2, ..., i_k\}$ and their corresponding similarity values $\{s_1, s_2, ..., s_k\}$. The prediction is found by taking the active users ratings on these similar items, and weighting these by multiplying them by their similarity values. The algorithm boils down into two main phases: Phase 1 is the Model-Building and Phase 2 is the Prediction Calculation.

4.1 Model Building

The Model Building phase involves computation of similarity between items. This particular implementation uses a correlation-based similarity metric, calculated using Pearson correlation.

To make this algorithm as efficient as possible, we use the cases where items i and j have been co-rated by users. In 2, below, using Pearson's equation from section 3, the set U consists of only those users who have rated both item i and item j. In this case, users 0 and 3 This calculation results in an $n \times n$ matrix of item similarities, where n is the number of items in the dataset. Since we are generally only interested in the top-n most similar items to each other item, we can sort each item profile in non-increasing order, and store only the top-n from this list as our model. One serious advantage of this method is that the entire model-building phase is an off-line computation, resulting in rapid recommendations for online users. This method is also scalable to any number of users, as their recommendations are generated from a static item-item model. It will be interesting to see if the hidden relations captured by this algorithm result in better, or worse recommendation accuracy on our datasets.

4.2 Prediction Calculation

The prediction phase of the IBCF algorithm is done as follows:

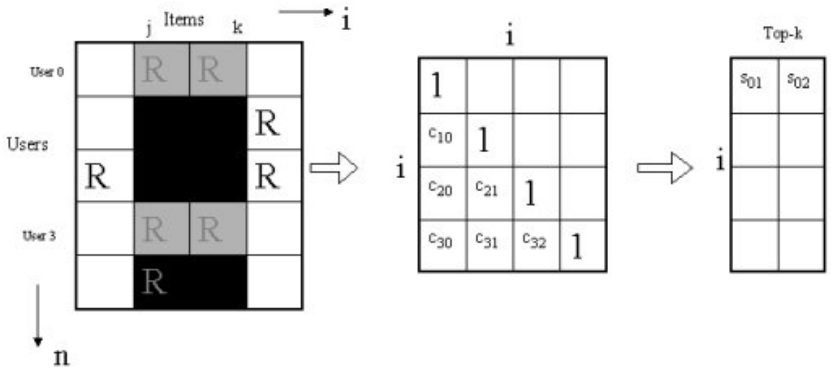

Figure 2: Offline Computation of the Item-Based Model

1. User rates a set U of items.
2. A Candidate set C of items is formed by taking the union of the k most similar items for each item $j \in U$. (Simply by taking the top-n from the ordered lists computed in the model-building phase)
3. Any duplicate items are removed. ie any $j \in (C \cap U)$.
4. Find the similarity between each item $c \in C$ and the set U. This is done by consulting the Item-Item correlation matrix from phase 1, and summing the correlation values between each $c \in C$ and every item in U.
5. C is then sorted in non-increasing order w.r.t these similarities.
6. The top-n items in C are the Recommendation Set.

5 Collaborative Filtering Using Association Rule Mining

This algorithm, as seen in [11], takes advantage of well known data mining techniques to compute similarity between items. The algorithm is in fact very similar to the IBCF algorithm described in the previous section. The APRIORI algorithm[1][3] discovers hidden relationships between items by generating rules of the form $A \Rightarrow B$, where A and B are itemsets. We can then use these rules to generate an item-item similarity matrix similar to that in 2 above. Association rules of this form have two important metrics: Rule Confidence and Rule Support. The confidence of a rule $A \Rightarrow B$ is the percentage of profiles containing A which also contain B. The support of a rule of this kind is the fraction of the total profiles in a database which uphold the rule:

$$support(A \Rightarrow B) = P((A \cup B) \subseteq T) \tag{3}$$

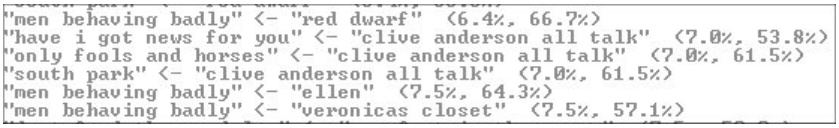

Figure 3: Sample Apriori Output for the PTV Dataset

$$confidence(A \Rightarrow B) = \frac{support(A \cup B)}{support(A)} \tag{4}$$

Taking each set of items in a profile as an itemset, the Apriori algorithm can derive item-item association rules and confidence levels. As can be seen in 3 below, these rules only incorporate binary-chaining, that is, they do not consider rules of the form $A \rightarrow B \rightarrow C$. This type of rule would identify further relations between the items in a dataset, and so can be forwarded as a method to combat the sparsity problem in CF. There is however, a trade-off with this approach: The more tenuous links become between items, the greater the risk of dissimilarity. Recommending a bad item is generally worse than failing to recommend a good item.

This association rule based algorithm is very similar to IBCF in that it contains a model-building phase and a prediction phase. The prediction phase of this algorithm is the same process as IBCF. The only difference between the two algorithms lies in the model-building. Item-Item similarity is derived directly from the association rules in order to compute the model. This is achieved by taking the confidence levels for rules generated by the Apriori algorithm, and using them as probabilities in the Item-Item similarity matrix. There are many ways to combine the probabilities with the support values to attain the Item-Item model. These are discussed in detail in [8]. For our testing, we simply use the rule support multiplied by its probability to arrive at a similarity value.

6 Experimental Evaluation

6.1 Experimental Data

For training and testing our algorithms, four experimental datasets are used: Jester[6] (an experimental dataset of jokes ratings, consisting of 21,800 users ratings of 100 jokes), EachMovie (73,000 users ratings of 1628 movies), PTV[5] (622 user ratings on TV programmes), and MovieLens[9] (100,000 user ratings in the movie domain.) It is hoped to also include a customer-product purchase database from an online sales company in the near future. For the purposes of our testing we selected subsets of 900 profiles from each of the above datasets comprised of the largest profiles: those users who had rated 20 items or more (with the exception of PTV which only contains 622 profiles).

For modularity in our system, these datasets were all parsed into the same format and stored in an SQL database.

6.2 Experimental Procedure

Common metrics for evaluating the accuracy of a prediction algorithm are in two main areas: Statistical Accuracy and Decision-Support metrics. Due to the number of algorithms and datasets, we focus on Statistical Accuracy only for our experimental evaluation. Our statistical accuracy is obtained by comparing a set of predicted values with a set of user-provided values (Test Data). MAE (Mean Absolute Error), is the average absolute difference between predicted ratings and actual ratings. We have simplified this somewhat further and measured predictive accuracy of our algorithms. For each dataset, if a users rating is beyond a certain threshold, the item is "liked" by that user. We predict the "liked" items for the unseen test data and record accuracy on each dataset. User profiles are split into training and test data. The training data is fed to each filtering component individually and each generates its own predictions for the unseen test data.

6.2.1 Density testing

In order to assess an algorithms performance with respect to density of a dataset:

$$\frac{number\,of\,nonzero\,entries}{number\,of\,total\,entries}$$

we vary the density of the set by running our accuracy tests over training ratios from 10% to 90% in increments of 10%. We should see increases in the accuracy of our recommendations as we increase the training set. There is a peak around 90% for the training set. This may be due to the fact that the test set becomes too small to be sufficient test sample after this threshold. For the rest of our testing we select this test-train ratio, as it should provide the most accurate results.

6.2.2 K-nn testing

It has been recorded[10] that there is a peak in the curve for the user-based and item-based algorithms at a neighbourhood size of around 30-40. We vary this neighbourhood-size between 10 and 100 in intervals of 10, in order to ascertain an optimal value for number of neighbours. We find that there is a difference in the relative performance of our algorithms over the datasets, most notably with the Jester dataset, where quality of recommendations drops dramatically after a threshold of approximately 40 with User-Based filtering, and 64 with Item-Based filtering. This may be attributed to the fact that groups of users who share similar tastes in jokes are in fact much smaller than those in a movie / TV-program domain.

6.2.3 The Zero-R algorithm

For comparison purposes we also test a primitive algorithm known as Zero-r. This simply predicts the majority class in the training data. This algorithm itself is not much use as a functional predictor, but serves well as a benchmark for the rest of our prediction algorithms. In one experimental case on the Jester dataset, IBCF and CF both perform worse than this basic algorithm. It has been suggested in [2] that this may be due to serious *overfitting*: Modelling of the training data so closely that the algorithm loses sight of the real picture.

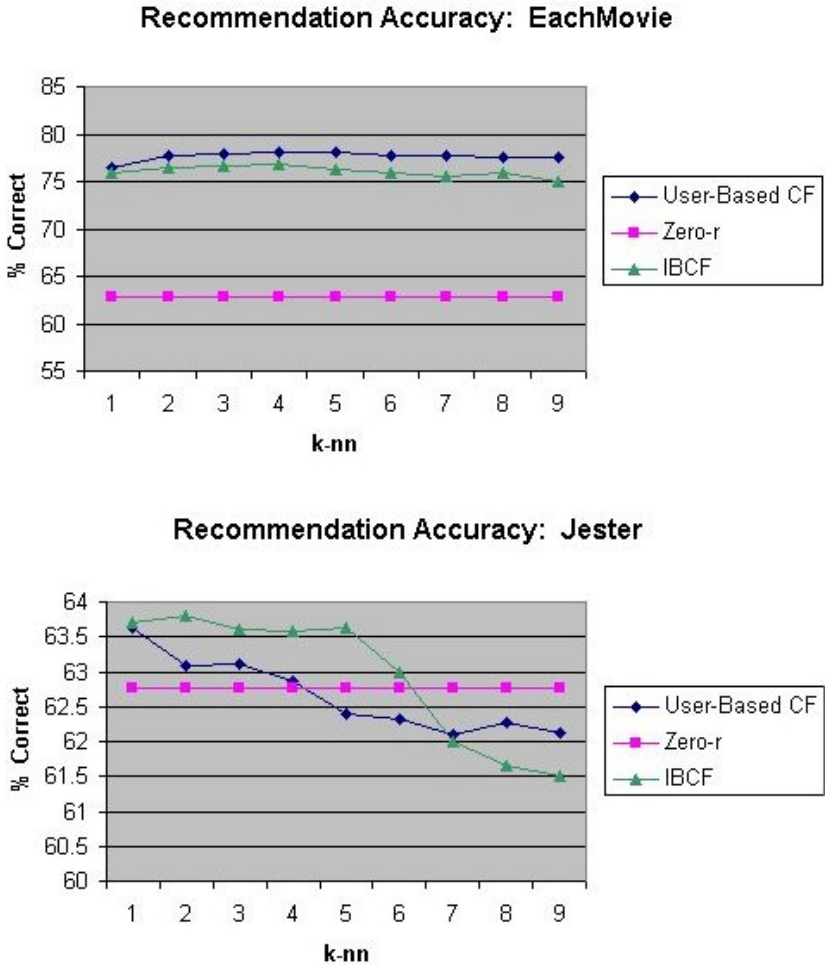

Figure 4: Testing neighbourhood sizes

6.3 Experimental Results

To recapitulate the aims outlined in section one, we aim to investigate relative performance differences of our recommendation algorithms over the different datasets we test them on. Our empirical results show that this is indeed the case. In our density evaluations, we varied train-test ratios in order to assess the effect of density of training data on our datasets. It was found across all sets that a train-test ratio of around 80% was the optimal value for the density, so this value was made static at 80% for all of the k-nn testing.

6.3.1 PTV

Our results do indicate that there is a relative difference in the performance of our algorithms with different data. Starting with the PTV dataset: At neighbourhood size k of 10, UBCF performs better than IBCF by 5%. The performance of the UBCF algorithm does not vary much with k, reaching a peak of 70.6% around $k = 50$. The performance of the IBCF algorithm improves to almost equal to UBCF at $k = 50$. This suggests that the neighbourhood size has a greater effect on the IBCF algorithm for the PTV dataset. All of our prediction algorithms outperform the Zero-r predictor, predicts the majority class for the PTV dataset at 62.5%. The best performance seen on this dataset throughout our tests is by the UBCF algorithm at a neighbourhood size of 50. The result in this dataset is somewhat different to [10], where IBCF outperforms UBCF at all k values.

6.3.2 Movielens

On the Movielens dataset, the results are very different. In this case, the UBCF algorithm is more affected by changes in neighbourhood size. At low k values, UBCF performs up to 9% worse than IBCF. However, unlike in the PTV set, this accuracy improves with textitk, and by $k = 60$ the UBCF algorithm outperforms IBCF. Interestingly, our baseline algorithm, Zero-r, performs better than the other prediction algorithms in this case. This is an odd result, which normally could be explained by examining the size of the majority class. However, the Zero-r algorithm has produced between 62% and 64% accuracy across all four of our test sets, which is the size of the majority class, meaning the Movielens set is not too dissimilar from the others in terms of this metric.

6.3.3 EachMovie

In the EachMovie dataset, there are further relative differences in performance between our algorithms with respect to data. Unlike all of the other test sets, the accuracy of IBCF and UBCF follow roughly the same trends, and are only marginally different at approximately 1%. The optimal recommender configuration for this set was found to be at a density of 30, and neighbourhood size of 40, using the UBCF

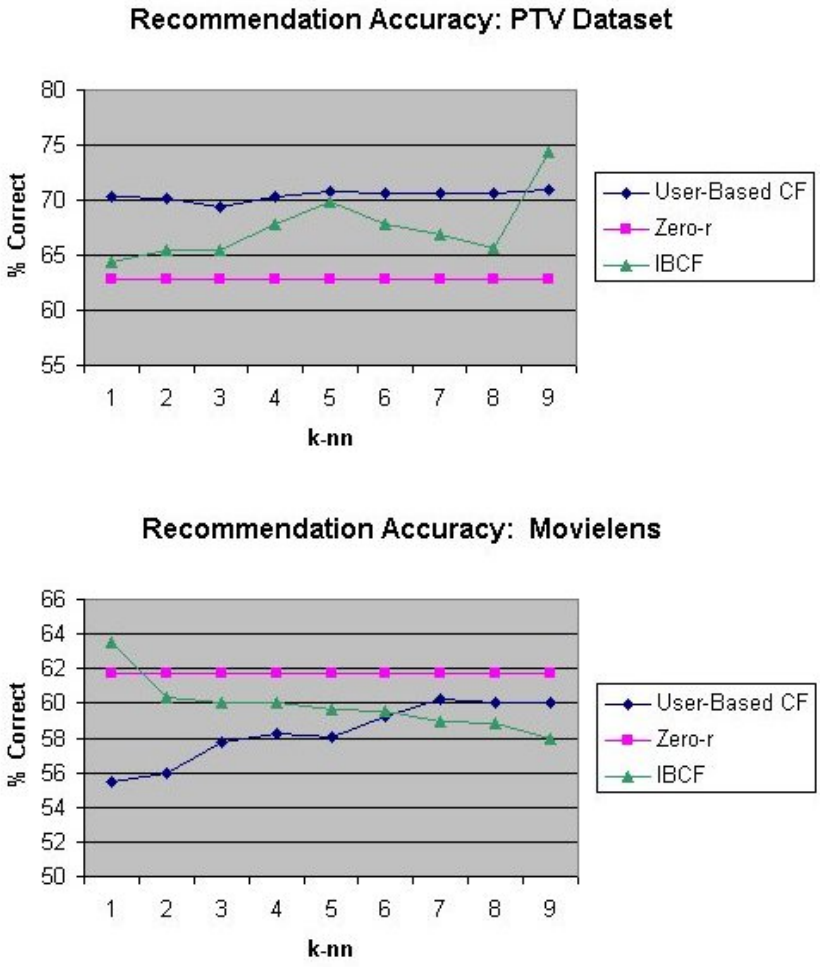

Figure 5: Testing neighbourhood sizes

algorithm. Our benchmark algorithm was outperformed across all k values by around 10%. These are the expected ranges for the other algorithms.

6.3.4 Jester

Jester shows more interesting results for our algorithms. The best performance here is given by the IBCF algorithm at low k values. This may be due to the fact that the clusters of people with similar tastes in jokes are smaller than those with similar tastes in movies, or TV programs. IBCF performs best up to a neighbourhood size of 50 before falling sharply with increasing k. At $k = 10$, UBCF performs as well , but falls sharply with increasing k. The Zero-r algorithm at

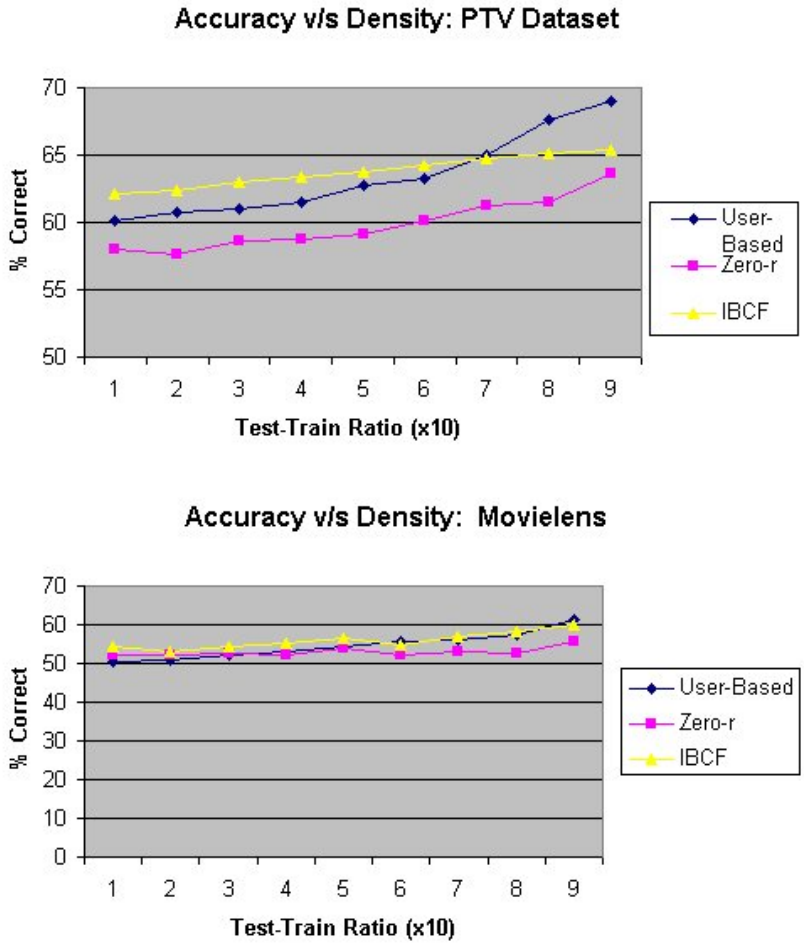

Figure 6: Density Testing

62.75% performs worse than the others, until neighbourhood reaches a max threshold (40 for UBCF and 64 for IBCF).

6.4 Recommendation Accuracy with Varying Density Levels

Density of the training data can be experimentally controlled by varying the test-train ratio of the experimental data. In 6 and 7 we can see the results of this variation. It is clear that across all of our datasets there an increase in predictive accuracy as the precentage of training data increases. This indicates that the algorithms are in fact learning well. We can see that the Item-Based algorithm seems to perform relatively better at lower test-train ratios, most notably in the Jester dataset, and is less affected overall by the change in the training set

size. This tells us something about the quality of the information captured by the Item-Based algorithm. The good performance of this algorithm at low test-train ratios may indicate that this filtering strategy may be employed as a method to solve the sparsity problem with User-Based collaborative filtering. Also, since better predictions can be made with less training data, this technique will save on memory (small model sizes) and will generate faster recommendations.

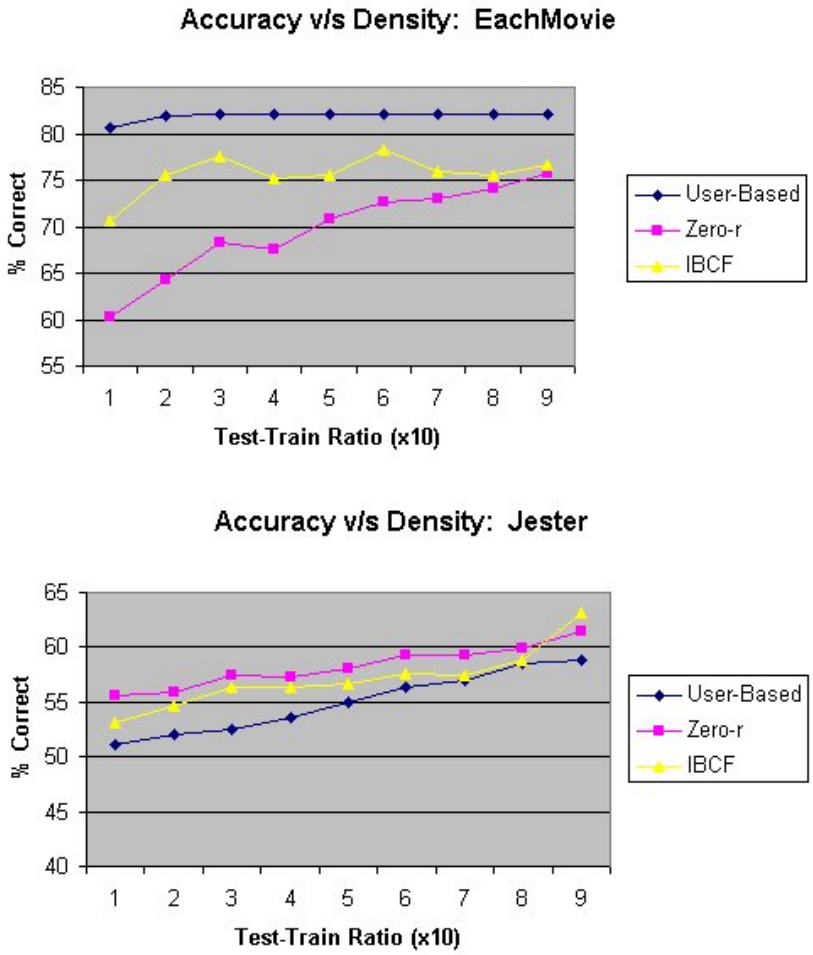

Figure 7: Density Testing

6.5 Summary of Results

The empirical results discussed here do show that performance varies across data types. It is hoped that we can capture this knowledge dynamically in a Machine Learning enviornment in order to adapt recommendation strategies to suit particular data types. These results

have also shown that the Item-Based algorithm has many advantages over the User-Based algorithm. This information will be valuable in the development of any hybrid stratageies.

7 Conclusion and Future Work

Collaborative recommendation techniques are powerful tools which enable the delivery of relevant information without users being forced to trawl through irrelevant material to find what they require. These techniques are relied on more and more as technology takes a stronger foothold in everyday living. These techniques also have great value to business as they can present a user/customer with items/products they are interested in, thereby generating more sales for the business with less hassle for the customer. Recommendation techniques are not without their flaws however, especially collaborative techniques. Problems such as sparsity of data, latency and early-rater discussed earlier lead to varying performance of recommendation techniques. In this paper we have evaluated three recommendation strategies on varying data. Our results indicate that there is a relative performance difference of these techniques over the datasets they were tested on.

Our future work will include extending the scope of test data to other domains, starting with a product-purchase database. It is also hoped to include new recommendation techniques in this analysis as they arrive. A machine-learning application is currently being developed to automatically learn the best recommendation technique (or hybrid technique) for a given dataset. This aspect of the AdRec system will be dealt with in later paper.

References

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.

[2] Syed S. Ali and Susan McRoy. Links: Java resource for artificial intelligence. *intelligence*, 11(2):15–16, 2000.

[3] Christian Borgelt and Rudolf Kruse. Induction of association rules: Apriori implementation. In *Proc. 15th Conf. on Computational Statistics (Compstat 2002, Berlin, Germany)*, Heidelberg, Germany, 2002. Physika Verlag.

[4] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In Gregory F. Cooper and Serafín Moral, editors, *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 43–52, San Francisco, July 24–26 1998. Morgan Kaufmann.

[5] Paul Cotter and Barry Smyth. PTV: Intelligent personalised TV guides. In *Proceedings of the 7th Conference on Artificial Intelligence (AAAI-00) and of the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-00)*, pages 957–964, Menlo Park, CA, July 30– 3 2000. AAAI Press.

[6] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[7] P. Melville, R. Mooney, and R. Nagarajan. Content-boosted collaborative filtering, 2001.

[8] Derry O'Sullivan, David C. Wilson, and Barry Smyth. Improving case-based recommendation: A collaborative filtering approach. In *Proceedings of the Sixth European Conference on Case Based Reasoning.*, pages LNAI 2416, p. 278 ff., 2002.

[9] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of ACM CSCW'94 Conference on Computer-Supported Cooperative Work*, Sharing Information and Creating Meaning, pages 175–186, 1994.

[10] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John Reidl. Item-based collaborative filtering recommendation algorithms. In *World Wide Web*, pages 285–295, 2001.

[11] B. Smyth, D. Wilson, and D. O'Sullivan. Improving the quality of the personalised electronic programme guide. In *In Proceedings of the TV'02 the 2nd Workshop on Personalisation in Future TV, May 2002.*, pages 42–55, 2002.