Experiments in Sparsity Reduction: Using Clustering in Collaborative Recommenders

Derek Bridge and Jerome Kelleher

University College, Cork

Abstract. The high cardinality and sparsity of a collaborative recommender's dataset is a challenge to its efficiency. We generalise an existing clustering technique and apply it to a collaborative recommender's dataset to reduce cardinality and sparsity. We systematically test several variations, exploring the value of *partitioning* and *grouping* the data.

1 Collaborative Recommenders

Given a set of items $i = 1 \ldots m$ and a set of users $u = 1 \ldots n$, a collaborative recommender will predict the rating $p_{au,ai}$ that active user au will assign to an item ai using the ratings that the other users have assigned to the items $r_{u,i}$.

Our experiments use the MovieLens dataset (`www.grouplens.org`). In this dataset, there are 100000 ratings by 943 users on 1682 items (movies) and the data has been cleaned up to include only users who have rated at least 20 movies. This dataset has been the subject of an extensive empirical investigation [2]. We can adopt the design decisions that [2] shows to give the best results.

Specifically then, our collaborative recommenders will work as follows:

- The similarity $w_{au,u}$ between the active user au and each other user u is computed using Pearson Correlation [2]. Following [2], the Pearson Correlation is weighted by a factor of $\frac{s}{50}$ where s is the number of co-rated items, to decrease similarity between users who have fewer than 50 co-rated items.
- After computing the similarity between au and each other user u, the 20 nearest neighbours are selected, i.e. the 20 for whom $w_{au,u}$ is highest.
- The predicted rating $p_{au,ai}$ is computed from the neighbours' ratings of ai as follows:
$$p_{au,ai} \hat{=} \bar{r}_{au} + \frac{\sum_{u=1}^{k}(r_{u,ai} - \bar{r}_u)w_{au,u}}{\sum_{u=1}^{k} w_{au,u}} \tag{1}$$
where k is the number of neighbours (in our case, 20). This is essentially a weighted average of the neighbours' ratings for ai (weighted by their similarities). But it normalises the ratings (by using deviations from mean ratings) to counter the effects of 'niggardly' or 'immoderate' users.

In collaborative systems, many possible items will be available to many users, most of which the users will not have rated. The high cardinalities give efficiency problems but also, if ratings are too sparse, the prediction accuracy can be lower than that of a non-personalised recommender [2][3]. We investigate one possible solution to these problems: clustering the dataset.

2 The Clustering Algorithm

The clustering algorithm which we have chosen is used in [1] to cluster users. The algorithm recursively invokes the k-means clustering algorithm.

In the k-means algorithm, elements are repeatedly assigned to clusters on the basis of their similarity to the cluster centre. We use Pearson Correlation again to compute similarity. The centres are initially seeds, provided as parameters. As the algorithm iterates, the centres become the centroids of the existing clusters, where a centroid is a new 'dummy' element formed by averaging ratings within the cluster. We consider the algorithm to have converged when the size of all clusters is the same on two successive iterations. We found empirically that convergence usually takes 15 iterations. To ensure that the algorithm always terminates, even on pathological data, we imposed a complementary stopping criterion in the form of an iteration limit of 20.

The RecTree clustering algorithm [1] calls k-means to split successive datasets into child clusters. It returns a binary tree of clusters, where the root represents the whole dataset. The algorithm splits the dataset if its size is greater than the maximum cluster size. It selects seeds using the 'two mutually well separated centres policy' [1]. The first seed is the element within the dataset whose distance is greatest from the dataset centroid. The second seed is the element which is most distant from the first seed.

The algorithm terminates when the sizes of all leaf nodes are less than or equal to the maximum cluster size and so no more subdivision is required. In some cases however, when data is not suitably distributed, growth in the tree can be 'lopsided'. This is where a very small and a very large cluster are created at each invocation. In this case we prevent the construction algorithm from over-partitioning the data by limiting the number of internal nodes (in our case to 2000). Because of this the algorithm makes no guarantee about the size of any leaf cluster. If the algorithm terminates normally, all leaf clusters will have at most the maximum cluster size number of elements. Of course, some might have very few members; if they have fewer than a certain threshold (in our case 10), we call the node an *outlier node*.

3 The Role of Clustering

We have developed and experimented with four collaborative recommender systems. In this paper, only three of the systems will be described, although the graphs show the results for all four systems. Where clustering is used in different systems, it is used for different purposes: *partitioning* and *grouping*.

Naive. Our Naive system acts as a baseline comparison system. It operates as described in Section 1: it exhaustively computes all similarities, picks the 20 nearest neighbours, and computes a prediction from these.

Clustered Users. We use clustering to form *partitions* of users. This reduces

	110	111	112	113	114	115	116	117
Jim	3		3				3	
Kelly		2	5		3		2	
Ray					4	4		3
Kim	4		3				2	

(a) Raw Data

	S1={110,111,112}	S2={113,114,115}	S3={116,117}
Jim	3		3
Kelly	3.5	3	2
Ray		4	3
Kim	3.5		2

(b) Schematic Illustration of the Effect of Item Clustering

Table 1. Clustering Items into Superitems

the *effective* cardinality of the dataset as only users from within a particular partition are considered as potential predictors. To produce a prediction for au, we apply the Naive algorithm only to the partition containing au. However, if a user is a member of an outlier node (Section 2), then we ascend up the RecTree and use the value from the centroid of the parent. In effect what we are doing here is using a non-personalised recommendation in cases where we do not have enough cluster members to provide an accurate collaborative recommendation.

Clustered Items. We use clustering to form *groups* of items ('superitems'). A superitem's rating will be the average of its member ratings. By grouping items into superitems, we reduce the cardinality, and hence the sparsity, of the dataset; see Tables 1(a) and 1(b). To cluster items we use Pearson Correlation but we compare items over co-rating users (instead of comparing users over co-rated items). Once we have clustered items in the dataset into superitems, we can produce predictions. To predict Jim's rating for item 113, for example, we find the superitem to which 113 belongs, S2, and then we make a prediction for S2 by applying the Naive algorithm to the superitems data.

4 Experimental Methodology

In each experiment, the MovieLens dataset is split into two disjoint sets, the training set (80%) and the test set (20%). All results are subject to five-fold cross validation, each time using a different 80/20 split.

To give an implementation-independent measure of *efficiency*, we count Total Rating Comparisons (TRC), i.e. the number of rating comparisons carried out

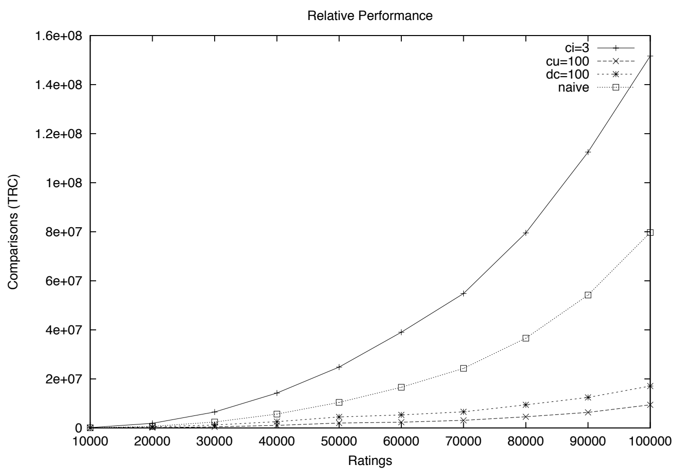

Fig. 1. Efficiency Results

by the system in making a prediction. To measure prediction *accuracy*, we use Mean Absolute Error (MAE) (the average of the absolute difference between the system's predicted rating and the user's actual rating). *Coverage* is measured by counting the number of times the system fulfils a prediction request and reporting this as a percentage of the overall number of prediction requests made.

One parameter of the experiments that we had to set was the maximum cluster size. To estimate the best values to use for this parameter, we ran some experiments in which we varied the maximum cluster sizes as we increased the number of ratings from 10000 to 100000. There is no room to plot or discuss the results from these experiments here, but we summarise our findings below.

For *Clustered Users*, we found that a maximum cluster size of 100 users gave a good trade-off between accuracy and running time; in our graphs this system is denoted by the line cu = 100. For *Clustered Items*, we found that, of the sizes tried, a maximum cluster size of 3 was consistently best for efficiency, accuracy and coverage; in our graphs this system is denoted by the line ci = 3.

5 Results

5.1 Efficiency (Figure 1)

Clustered Users carries out the lowest number of comparisons. The reason for this is the reduced number of potential advisors for each user. This gives a significant improvement over Naive in which all users in the system are potential advisors.

Clustered Items is by far the least efficient. This may seem counter-intuitive, but it is explained by realising that grouping items decreases sparsity, which increases the number of co-rated (super)items to be compared. This is shown

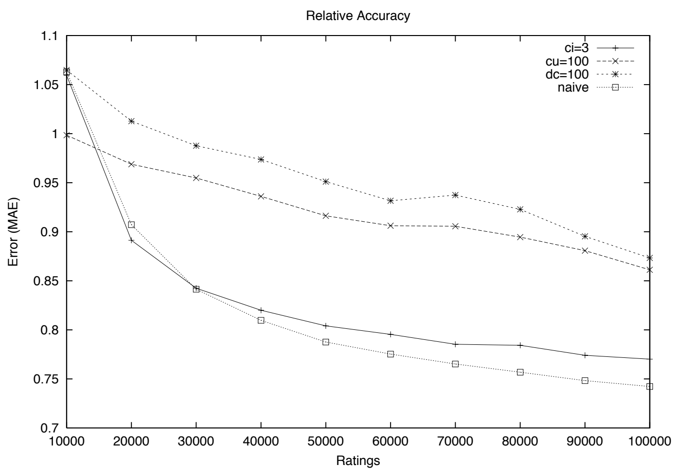

Fig. 2. Accuracy Results

in Tables 1(a) and 1(b) where we reduce the cardinality from eight to three by clustering on contiguous item identifiers. Suppose we want to predict Jim's rating for item 117. In the raw data, Ray is the only person in a position to provide a prediction. But, when we produce a prediction for Jim using the clustered items, we find that item 117 is a member of cluster (superitem) S3. All of the other users have rated item S3 (because they all rated one or more of its constituent items), so we must consider each of these other users as a predictor, so we must compare Jim for similarity with *three times* more users.

5.2 Accuracy (Figure 2)

Naive is generally the most accurate of the systems as it always finds the optimal advisors for each request. But, when the number of ratings in the system is small, Clustered Users has the lowest error. This reflects a well known property of collaborative recommender systems: when the number of ratings is low, a non-personalised recommender is more accurate than a collaborative one. Recall that Clustered Users resorts to non-personalised recommendations for outlier nodes. However, the accuracy of Clustered Users is soon surpassed by Naive and Clustered Items, because the optimal advisors for a prediction on an item are not generally members of the same partition as the active user. Clustered Items is almost as accurate as Naive, but some additional error is introduced due to inappropriately clustered items.

5.3 Coverage (Figure 3)

Clustered Users has by far the worst coverage. By partitioning users, we are making useful ratings inaccessible to users in other partitions. Clustered Items

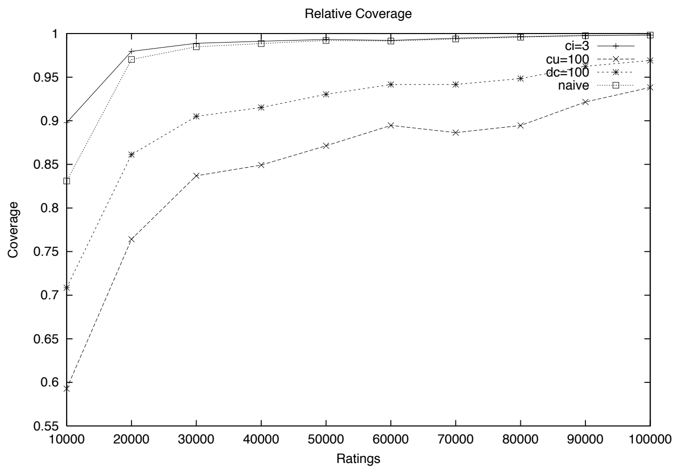

Fig. 3. Coverage Results

has very good coverage, sometimes higher than Naive. This is easily explained by an example. In Table 1(a), we cannot obtain a prediction for Kim on item 113: no-one has rated item 113. However, in Table 1(b), item 113 is a member of cluster S2. Both Kelly and Ray have ratings for superitem S2, so we can provide a prediction for Kim on superitem S2, and hence for item 113.

6 Conclusions

We presented results from some of our experiments on different systems. The *partitioning* given by Clustered Users results in by far the best efficiency. Accuracy is highest in the Naive system as it always finds optimal advisors for every prediction request, but the *grouping* given by Clustered Items yields surprisingly low error. Clustered Users suffers a severe accuracy penalty as the partitioning distributes item ratings unevenly among the partitions. Clustered Items has the further advantage of increasing coverage in cases where data is very sparse but the efficiency implications of this approach are very severe.

References

1. Chee, S.H.S.: *RecTree: A Linear Collaborative Filtering Algorithm*, M.Sc. Thesis, Simon Fraser University, 2000.
2. Herlocker, J.L.: *Understanding and Improving Automated Collaborative Filtering Systems*, Ph.D. Thesis, University of Minnesota, 2000.
3. Resnick, P., N.Iacovou, M.Suchak, P.Bergstrom & J.Riedl: GroupLens: An Open Architecture for Collaborative Filtering of Netnews, in *Procs. of ACM CSCW'94 Conference on Computer-Supported Cooperative Work*, pp.175–186, 1994.