Social Computing COMP3208

Group Report

Kimathi Nyota (kn3g17) and Fernando Winardi (fw1u17)

ABSTRACT

A recommender system is a tool used to filter information based on a personal profile. The purpose of this report is to go more in depth on building recommender systems mainly using collaborative filtering techniques. Moreover, our initial implementation has been tuned by experimenting with different parameters to produce two recommender systems: one that generates the most accurate predictions, and another that aims to generate accurate predictions while prioritising speed. Finally, we have compared our final recommender systems to other benchmark systems that use different approaches and we have evaluated our system to find out how it can be improved.

Users can have a hard time making decisions such as making a purchase, choosing a movie to watch, etc due to the sheer abundance of options to choose from. To tackle such problems, a recommender system can be built. It is an algorithm that aimed at suggesting relevant items to users based on previous experiences (Rocca. B, 2019).

A recommender system can be built using different types of filtering such as content based filtering or collaborative filtering. However, for this project we will look solely into item-based collaborative filtering. Item-based collaborative filtering works on the assumption that users are likely to give a similar review for similar items. Our recommender system aims to produce a set of predicted ratings for a testing set based on the training set provided.

Furthermore, we will also be testing different parameter combinations within the recommender system (i.e. dealing with cold start) to come up with the best predictions. Finally, we will compare our recommender system with other ones that uses different approaches and consider how our system can be improved.

1 Design and Description of Recommender System

1.1 Representation

Given data with tuples in the form of UserID, ItemID, Rating, Timestamp, the goal of the system is to generate the missing predicted rating field for each tuple. To achieve this, we first store the inputs into a MySQL database table, the reason for choosing MySQL instead of other alternatives such as SQLite is mainly because we had prior experience with this RDBMS. Two hash maps were created for accessing data within the recommender system. One maps an item to a map of users to ratings. This was used to speed up the access of users that have rated both items, which is required to produce a similarity matrix. The other is a map of users to a list of item records (in the form of ItemID, Rating and Timestamp) and the total rating. This was used to increase the speed of calculating the average user rating. Additionally, we chose to use Item Based Collaborative Filtering for our recommender system mainly because the number of items in the dataset is much less compared to the number of users so this approach would require a much smaller similarity matrix.

1.2 Similarity Measure

1.2.1 General

For calculating similarity, we used the adjusted cosine similarity formula (shown at appendix A). Despite many other alternatives, we chose this similarity measure because it is commonly used in recommender systems and it can easily (and cheaply) be implemented.
The calculated similarity is stored as two one-dimensional arrays: one array for storing all of the similarity values and another array for storing the indexes for each item allowing for easy access of similarity in the form of (item, user) shown in figure 2. The previous approach was chosen to save memory since using a two dimensional array (e.g. sim[item][user]) wastes memory by disregarding the symmetrical nature of similarity matrices. The calculated similarity matrix was stored as a .ser file object in secondary storage. This approach was chosen instead of saving it directly into a new MySQL table or storing it in main memory as hash maps because the former approach takes too long to store and the latter is fast but isn't possible due to the Java heap space limit.

Figure 1. A code extract showing how the similarity matrix is generated and stored

```
public void saveSimilarityMatrix(int max) throws Exception{
        double[] sim = new double[199970001];
        // primitive arrays of exact size required chosen to save space
        int[] index = new int[20000];
        int ind = 0;
        int count =0;
        for(int item1=1; item1<=max; item1++) {
            for (int item2 = item1 + 1; item2 <= max; item2++) {
                // symmetrical so: item1 > item2 <= max
                sim[count] = this.calculateSimilarity3(item1, item2);
                // sim calculated using general adjusted cosine similarity
formula
                count += 1;
            }
            // index(item) = starting position of item in sim array
            index[ind] = count;
            ind += 1;
        }
        // code below saves sim and index to secondary storage
        saveArrays(sim, index);
    }
```

Figure 2. The equation used for fast access, O(1), of data in the one-dimensional representation of the similarity matrix in the form of (item, user)

$$getSim(x, y, sim_{arr}, indexes_{arr}) = sim_{arr}\left[(y - x - 1) + \begin{cases} indexes_{arr}[x-2], & x > 1 \\ 0, & otherwise \end{cases}\right]$$

$$where\ x = item1, \quad y = item2, \quad y > x, \quad sim_{arr} = 1D\ similarity\ matrix\ array$$

1.2.2 Dealing with cold start problem

A cold start problem occurs when there are too few ratings between two items to calculate the similarity. In this case, instead of returning null, our system returns 0 because this is a neutral value (given that similarity values range from -1 to +1) that is neither similar nor dissimilar.

1.3 Neighbourhood Selection

The predicted score for a given user and item will be calculated from a neighbourhood set of items that are similar to the input item. In our implementation, the neighbourhood set will consist of the k most similar (adjusted-cosine similarity) items.

Figure 3. The specific algorithm used for determining the neighbours is shown below

```
function getNeighbours(item, k, sim, indexes)
    bestItems: Priority-Queue[max=k] // only stores a max of k items
    // stores (Item, Similarity) in order of Similarity
    // head of queue is the entry with the lowest similarity
    For i from 1 to 19999
        If i != item
            sim = getSim(min(item, i), max(item, i), sim, indexes)
            bestItems.add(i, sim)
    return bestItems
end function
```

1.4 Prediction

Figure 4 This code calculates the predicted score using the standard weighted sum formula (shown in appendix B)

```
1.      public int prediction2(int user, int item, PriorityQueue neighbours){
2.          // PriorityQueue with entries in the form of (Item, Similarity)
3.          double predNum = 0;
4.          double predDenom = 0;
5.          double temp;
6.          // Standard weighted sum formula applied below
7.          for(Pair<Integer, Double> pair: neighbours){
8.              temp = pair.getValue(); // similarity
9.              /* userRating(user, item) = rating of user of item
10.                returns itemAverage(item) if user hasn't rated item */
11.             predNum += temp * userRating(user, pair.getKey());
12.             predDenom += temp;
13.         }
14.         // Handling sparsity/cold start problem:
15.         if (predDenom == 0){
16.             int r = (int) Math.round(itemAverage(item));
17.             return r;
18.         }
19.
20.         double pred = predNum / predDenom;
21.         return Math.max((int) Math.round(pred), 1);
22.     }
```

1.4.1 Dealing with cold start (sparsity) problem

In lines **15 to 18** (figure 4), a denominator value of 0 indicates that there isn't enough data available for making a prediction using the standard weighed sum formula. These cases are referred to as the cold start or sparsity problem and this problem is solved in our implementation by using the average rating for the item as the predicted rating. This method was chosen instead of using a median value of 2 or 3 since it takes into consideration cases where a consensus has been achieved by most users e.g. if almost all users agree with the rating of 0 for a faulty product, the average is a more likely prediction than using 2.

2 Evaluation and discussion of Recommender System

2.1 Gathering parameters for tuning

The following parameters were gathered for the purpose of tuning our recommender system.

2.1.1 K-value

The first parameter that we considered for tuning is the number of neighbours selected during neighbourhood selection. The reason for doing this is to compare the outcome of using different value of K ranging from fewer neighbours to more neighbours.

2.1.2 Neighbour Metrics

The second parameter is the neighbour metrics which is split it into 3 options (Highest Similarity, Most Recent and Most Recent and Highest Similarity). With highest similarity,

the k items with the highest similarity were selected as neighbours. With most recent, the k items that were rated in the most similar time period were chosen, since we assumed that time period has an effect on an item's rating (i.e. trends). Finally, we tuned our system against both Most Recent and Highest Similarity to see if the combination of both metrics will result in the best outcome.

Figure 5 A summary of all of the options for the neighbourhood metrics parameter

#	**Neighbourhood Metric**
0	Highest similarity
1	Most Recent
2	Highest and Most Recent

2.1.3 Cold Start

Cold start occurs when a prediction value could not be reached due to a limited number of ratings to produce a prediction. To overcome this problem, we came up with 2 methods, using the median value of the ratings, in this case 2 and 3, and using the item average rating as the predicted rating. The reason for specifically using the median value is that it has a higher likelihood of getting closer to the correct prediction compared to when using lower bound and upper bound values. Using item average instead of median value will result in a larger overhead as it has to go through the whole set of ratings for an item to find its average rating, however, it is more intuitive than using a particular value. An example of such case would be when an item is faulty and so has a low average rating, it would be more accurate to predict the average rating rather than median value. Finally, if an item hasn't been rated at all, the median value is used.

Figure 6. A summary of all of the options for the cold start parameter

Approach	#	Description
Using item average	0	Item average with default = 2
	1	Item average with default = 3
Using median value	2	Median value = 2
	3	Median value = 3

2.1.4 Similarity Range

The last parameter to be tuned is the similarity range. We will be using 2 different similarity ranges. The first would be the default similarity ranging from -1, which is the least similar, to 1 being the most similar. The second similarity range, produced via shifting from the previous one, is 0, being the least similar, to 1, being the most similar. The reason for doing this is to remove negative values as we made an assumption that when the denominator of the prediction is 0, there are too few ratings to accurately produce a predicted rating, and it is possible for negative similarity values to incorrectly trigger this assumption.

2.2 Designing the Experiment

The aim of this experiment is to find the configuration of parameters that will have the best outcome. The outcome will be based on 2 accuracy metrics.

2.2.1 MSE (Mean Squared Error)

The first accuracy metric is Mean Squared Error. It refers to the squared difference of the predicted values and the actual value. We chose this approach as it is a commonly used metric.

2.2.2 Number of Exact Matches

The second accuracy metric is the Number of Exact Matches that refers to the number of correct predicted ratings. The reason for this approach is that it is more intuitive than MSE and also easier to calculate. In addition to this, it is also more restrictive than MSE therefore higher values of this metric are highly indicative of prediction accuracy.

2.2.3 Running the experiment

The parameters required to run this experiment are the metrics mentioned in the earlier section. We then iterate through each of the first 5000 entries in the training set and calculate and compare the predicted ratings to the actual ratings. Finally, these ratings are used to output the parameters, MSE value and Number of Exact Matches to a CSV file.

2.3 Results

Figure 8. A table showing the average MSE value for each cold start method (#) in **figure 6**

# Cold Start	Average MSE
0	1.057
1	1.057
2	1.057
3	1.057

The average MSE value is the same for each method of dealing with the cold start problem. This shows that the cold start method choice is inconsequential and so any of these approaches can be chosen for a data set consisting of the first 5000 entries of the training set.

Figure 9. A table showing the average MSE and exact match counts for each neighbourhood metric (#) in **figure 5** for both larger and smaller neighbourhood sets

	K >= 10,000		K < 10,000	
# Metric	Average MSE	Average Exact Match	Average MSE	Average Exact Match
0	1.0472	2096	1.0567	2091
1	1.0667	2065	1.0580	2987

2	1.0606	2077	1.0541	2091
Average	**1.0582**	**2079**	**1.0563**	**2090**

The results show that the most accurate predictions are made using the highest similarity as the neighbourhood metric (option 0) with larger neighbourhood sets as these parameters have the lowest average MSE and the highest average exact match. However it's more accurate to use smaller neighbourhood sets (especially when using the similarity-recent combination metric, #2) instead of using larger sets with the other metrics (#1 or #2).

Figure 10. A table showing the average MSE and exact match counts for each similarity range option for both larger and smaller neighbourhood sets

	K >= 10,000		K < 10,000	
Range	Average MSE	Average Exact Match	Average MSE	Average Exact Match
-1 to +1	1.0538	2087	1.0559	2090
0 to 1	1.0625	2072	1.0566	2090
Average	**1.0582**	**2079**	**1.0563**	**2090**

The results show that the most accurate predictions were made when a similarity range of -1 to +1 was used with larger neighbourhood sets. Using larger neighbour sets with the other similarity range led to the most inaccurate predictions.

Figure 11. A table below shows the most accurate entries. Left: Shows the entries with the lowest MSE values in ascending order. Right: Shows the entries with highest exact match counts in descending order. The corresponding numbers (#) can be found in **figures 5** for neighbourhood metric.

Lowest MSE Values				**Highest Exact Match Counts**			
K	# Metric	# Similarity Range	MSE	K	# Metric	# Similarity Range	Exact matches count
19999	0	-1 to +1	0.9964	19999	0	-1 to +1	2147
19750	0	-1 to +1	1.021	19750	0	-1 to +1	2137
19500	0	-1 to +1	1.0318	40	1	0 to 1	2117
19250	0	-1 to +1	1.0368	19500	0	-1 to +1	2115
19000	0	-1 to +1	1.0388	40	1	-1 to +1	2113
17500	0	-1 to +1	1.0446	19250	0	-1 to +1	2110
30	1	-1 to +1	1.0448	20	1	-1 to +1	2108
40	1	-1 to +1	1.045	20	1	0 to 1	2105
20	1	0 to 1	1.0466	19000	0	-1 to +1	2103

Note: The max number of neighbours (k) is 19,999 so this value corresponds to using all items as neighbours. Also, cold start approach has been excluded to avoid redundant entries.

These results show that the most accurate predictions are made, when using MSE as the accuracy metric, with larger neighbour sets with k values greater than 17500 using the highest similarity neighbourhood metric and a similarity range of -1 to +1. When using highest exact match counts as the accuracy metrics in **figure 11**, the best predictions are still made with larger neighbourhood sets (k >= 19750),-1 to +1 similarity range, and highest similarity as the neighbourhood metric. However, a greater number of entries with smaller k values are closer to the top, with an entry with a k value of 40 being the 3rd most accurate.

2.4 Conclusions

Accuracy isn't the only important attribute in evaluating a recommended system; processing speed is also just as important since highly accurate recommender systems that take too long to be produced are not very useful to users. As shown in the results, the most accurate predictions are formed with configurations using larger neighbourhood sets. Since these large sets are iterated through when calculating the predictions, the recommender systems using larger k values will take longer to compute. In industry there may be certain problems where accuracy is more important than speed or vice versa; for example, in cases where real time recommendations are required speed takes precedence, however accuracy is more important in places where parts of the system can be processed in batches.

2.4.1 Most Accurate Recommender System

The final parameters chosen for the most accurate recommender system are shown below

K	Similarity Range	Neighbourhood metric	Cold start method
19999 (ALL)	-1 to +1	Highest Similarity	Item average with a default value of 2

Under both accuracy metrics, the most accurate predictions were made using the above parameters. As mentioned earlier, the choice of method for handling cold starts has been shown to be inconsequential for the data set used in the experiment. Although there is an overhead in computing the average, compared to using the median value, this choice has only been shown to be inconsequential for this particular data set and so to err on the side of caution, we decided to predict the item average, with a default value of 2, in cold-start cases where there isn't enough information to form a prediction. Specifically a default value of 2 instead of 3 was chosen because we assumed that users are more likely to trust a system that underestimates their score compared to one that overestimates it.

2.4.2 Fastest Recommender System

The final parameters chosen for the fasters (but still accurate) recommender system:

K	Similarity Range	Neighbourhood metric	Cold start method
40	0 to 1	Most Recent	Default value of 2

An entry with the above parameters is among the top 5 entries with the highest exact match counts and among the top 8 entries with the lowest MSE value. Of all the entries with k values less than 10000, the above one can be considered to be the most accurate one for the experimental data set.

2.5 Evaluation

2.5.1 Comparison to standard benchmarks

Figure 12. A table showing a comparison of the prediction accuracy between the two above tuned recommender systems and certain benchmarks. Each recommender system was run for 5000 records in the training set.

Recommender System	Description	MSE	Exact Match Count
Accurate	Tuned system in section 2.4.1	0.9964	2147/5000
Fastest	Tuned system in section 2.4.2	1.045	2113/5000
Average	System that predicts, for a given item and user, the average of the mean item rating and the mean user rating.	1.0874	2060/5000
Random	System that predicts a random rating	4.4634	827/5000

2.5.2 Improving recommender systems

2.5.2.1 Limitations

The main limitations of our recommender system are the following:

- **Accuracy**: The accuracy of our predictions can be improved. Although both of our tuned systems are more accurate (under both metrics) than the average and random benchmarks, the difference between them and the average recommender system is not as large as we would want.
- **Robustness**: Our solution to the cold start (also known as the sparsity) problem needs to be improved.
- **Single criteria**: The system only works with a single score for a user rating. Often in industry a user will rate an item given multiple criteria so our system can be extended to accommodate this (Jhalani et al, 2016) .

2.5.2.2 Regression

Incorporating regression is one method of improving the accuracy of our recommender system. In our current implementation, the prediction of a user's rating of an item is calculated as the average of the ratings for similar items, weighted by the similarity for each item. One problem of this approach is it is possible for some items to be highly similar but have a low adjusted cosine similarity resulting in some bad predictions. Regression approaches attempt to circumvent this problem by approximating the actual ratings using a regression model (Sarwar et al, 2001).

2.5.2.3 Matrix factorisation

These systems will find clusters of user and item ratings in order to discover latent factors. These factors will be used to represent the user to item rating matrix as two smaller matrices: a matrix of item vectors, representing how connected an item is to each factor, and a matrix of user vectors, representing how connected a user is to each factor. The predicted score for a given user and item is determined by taking the dot product of the item and user vector.

These systems typically optimise the predictions for a given training set by minimising the error between the predicted and actual rating (Koren et al, 2009).

By switching to a matrix factorisation method, both the accuracy and robustness of the solution will be improved. This is because these methods are better at handling cold-start cases of limited information since a user that has not rated a particular item may have rated items that are similar through particular latent factors that can be used to form the prediction. The most memory and time intensive process for our recommender system is calculating and storing the similarity matrix. Using matrix factorisation can improve the efficiency of our system since by shortening the initial matrix to just two matrixes with fewer dimensions the calculation is simpler to compute and we can avoid having to store an extremely large similarity matrix.

References

1. Rocca, B., 2019. *Introduction To Recommender Systems*. [online] Medium. Available at:<https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada>
2. Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. Computer8 (2009), 30–37
3. Jhalani, T., Kant, V. and Dwivedi, P., 2016. A Linear Regression Approach to Multi-criteria Recommender System. Data Mining and Big Data, pp.235-243.
4. Sarwar, B., Karypis, G., Konstan, J. and Reidl, J., 2001. Item-based collaborative filtering recommendation algorithms. Proceedings of the tenth international conference on World Wide Web - WWW '01,.
5. Vucetic, S. and Obradovic, Z., 2005. Collaborative Filtering Using a Regression-Based Approach. Knowledge and Information Systems, 7(1), pp.1-22.

Appendix

Appendix A: The adjusted cosine similarity

$$sim(i1, i2) = \frac{\sum_{u \in U}(r_{u,i1} - \bar{r}_u) \cdot (r_{u,i2} - \bar{r}_u)}{\sqrt{\sum_{u \in U}(r_{u,i1} - \bar{r}_u)^2} \cdot \sqrt{\sum_{u \in U}(r_{u,i2} - \bar{r}_u)^2}}$$

where:

- *i1* and *i2* are *items* and $r_{user,item}$ = rating of item by user, $\bar{r}_u$ = average user rating
- *U* is the set of users who have rated both *i1* and *i2*

Appendix B: The standard weighted sum prediction formula

$$pred(u, i) = \frac{\sum_{i' \in N} sim(i,i') * r_{u,i'}}{\sum_{i' \in N} sim(i,i')} \qquad N = neighbourhood\ set\ of\ items$$

Technical Resources

Resource	Source
MySQL	https://www.mysql.com/
MySQL Work Bench	https://www.mysql.com/products/workbench/