Building a Context-Aware Recommender System

1st Andre Hui
Department of Computer Science
Durham University
Durham, UK
tsun.m.hui@durham.ac.uk

***Abstract*—This paper compares multiple approaches to building a context-aware recommender system (CARS), and builds one itself with techniques from existing approaches. It further evaluates the CARS against existing benchmarks.**

***Index Terms*—context-aware, recommender system, CARS**

Introduction

Recommender systems are useful for generating relevant and helpful recommendations to users in all types of applications, from music apps like Spotify, to online shopping websites such as Amazon. Adding context-awareness to such systems allow us to further refine these recommendations, making them more specific to the users' needs. Such systems may make use of explicit data, such as user ratings or mood selection, as well as implicit data, such as current location, time of day and weather.
This paper will be briefly summarising existing context-aware recommender systems, and building a CARS based on these techniques, capable of generating recommendations with a single context. We will then evaluate our system against pre-existing benchmarks.

Related Work

CARSs may be categorised into three paradigms:

1) Contextual Pre-filtering: context is used to select a set of rating data relevant to the context, before generating recommendations;
2) Contextual Post-filtering: recommendations are generated, and reordered afterwards with regards to some context;
3) Contextual Modelling: contextual information is used in generating ratings.

[1] takes a pre-filtering approach, where it's proposed to split the set of ratings into subsets, according to the value of a contextual variable, and assign new fictitious items to each set, given that there is statistical evidence implying differences in ratings between the contextual conditions. [2] pre-filters by grouping similar contextual situations, determining their similarity by comparing their influence on the rating data. Such approaches may suffer from data sparsity and overly specific contexts.
[3] takes a post-filtering approach, exploiting association rules to identify the greatest correlations among context and item characteristics, using them to filter the prediction formed by traditional recommender systems. By adjusting rankings after generating the recommendations, we can avoid the data sparsity problem.
[4] uses a contextual modelling approach, integrating both user preferences and contextual conditions in a common vector space. [5] models the context by extending a Support Vector Machine (SVM), adding axes of context to its feature space. [6] looks at multiple approaches based on matrix factorization, to incorporate context into recommendation generation, and [10] builds upon this, looking at the full user-item-context interactions. [7] looks at using deep neural networks to generate recommendations. With these approaches, training time and cold start becomes a factor, however they should generate the fastest responses.

Methods

Data Type and Source

The dataset referred to in the rest of this paper will be the TripAdvisor dataset from [8], in the format of a .csv file. The pandas module in python is used to import and manage the dataset.
The dataset contains entries corresponding to UserID, ItemID, Rating, UserState, UserTimeZone, ItemCity, ItemState, ItemTimeZone and TripType. In this CARS, the UserID (U), ItemID (I), Rating (R) and the context TripType (C) will be considered. Other candidate contexts include UserState/TimeZone, and ItemCity/State/TimeZone. However, TripType is the one recommended by the authors of [8], and amongst all possible candidates, TripType is most likely to have an effect on the rating.

Feature Extraction and Selection Methods

The only useful user data given is the specific rating objects submitted by the users, consisting of UserID, ItemID, Rating and Context. Using the set of ratings, however, the most likely context condition can be approximated for each given user, by counting the instances of a context condition within a users ratings. This can be used to generate initial predictions that are more relevant to the user, without gathering explicit information such as number of travellers.
Other data considered include User/Item State/TimeZone/City. The only likely candidate for use as another filter is Item State/City, which indicates any bias a user has towards a certain city, and base the initial recommendations around that city. However, this requires a larger dataset; many users have only

5 rated items, and this is typically insufficient to determine a modal trip destination for each user. Furthermore, questions are raised as to whether recommendations only within the locations a user has visited are desirable, or if exploration should be encouraged. Perhaps, if the dataset included city characteristics, previous destinations could be used as a basis for recommending new destinations.
To determine the context of a user outside of initial recommendations, the number of travellers are considered, as explicitly informed by the user. This information can then be used to infer the trip type: if there is a single traveller, it is more likely to be a SOLO- or BUSINESS- type trip. If there are two travellers, it is more likely to be a COUPLE-type trip. If the number of travellers is greater than 2, it is more likely to be a FRIENDS- or FAMILY- type trip. With this, unneeded contexts are filtered out from the predictions for each user, making recommendations more catered/specific to the user.

User Profiling and Prediction Methods

A contextual-modeling approach is taken, implementing a matrix factorization model, based on ICAMF-I from [10].
Contextual modelling is chosen over pre-filtering due to the possibility of over-filtering our specific dataset, which is considerably small in size and sparse. Post-filtering is chosen against for similar reasons, where a post-filtered solution could filter out too many irrelevant items.
Amongst contextual-modeling approaches, ICAMF-I is selected as opposed to tensor factorization due to the faster training times of a matrix factorization approach. Amongst the existing context-aware matrix factorization approaches, ICAMF-I was chosen over the CAMF models in [6] due to its consideration for user-item-context interaction, as well as item and user bias. ICAMF-I was chosen over ICAMF-II as recommended in [10], where ICAMF-I was found to perform better with less influential contexts. This is ideal in with the dataset in [8], where the context of trip type is unlikely to have a significant influence over the rating.

Evaluation Methods

To evaluate the success of our CARS, Mean Absolute Error (MAE) across the entire training dataset is calculated at the end of training, as well as the accuracy of the recommendations when generating existing predictions. We choose N items for the CARS to recommend to a given user u. Let the lowest rating in the N recommendations be r_{min}. If a test item for u is contained within those predictions, with a rating greater than r_{min}, we determine such results to be true positives. If a test item is not contained within the predictions, but the test item has a rating lower than r_{min}, these are considered true negatives. If a test item is contained in the predictions, with a rating lower than r_{min}, they are considered false positives. If a test item is not in the predictions, but has a rating greater than r_{min}, these are false negatives. r_{min} is rounded to the nearest integer to account for the discrete values of a rating.
Using the true/false positives/negatives, the precision and recall of our recommender system may be calculated. Due to the sparsity of the dataset, choice of users with large amounts of ratings is limited. Ultimately, a fifth of the ratings for each user was randomly hidden from training. These ratings are then used to test the accuracy of the CARS.

Implementation

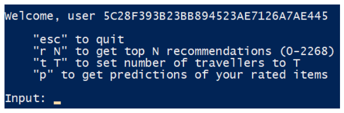
We implement ICAMF-I in Python, using the pandas module to parse the dataset file, and numpy to do much of the training. A class MF is implemented, which takes a 3-dimensional array, an integer representing the number of features f per user/item/context condition, and the regularization hyperparameter λ_1. To initialize the class, let the number of users be u, number of items i, and number of context conditions c. We generate 3 arrays, U, I and C, of size (u, f), (i, f) and (c, f) respectively to represent the features corresponding to each user/item/context condition, as well as 2 arrays of size (u) and (i) representing the user/item biases. Predictions are made using the following equation:

$$\hat{r}_{uic} = \bar{r} + b_u + b_i + I_i.U_u + I_i.C_c + U_u.C_c$$

where $\hat{r}$ is the mean of all observed ratings in the dataset, b_u and b_i are the learned biases of the user and item respectively, and U_u, I_i and C_c representing the features of user u, item i and context condition c respectively.
The loss function is as follows:

$$\min_{I_*, U_*, C_*, b_*} \sum_{r_{uic} \in R} [L(r_{uic} - \hat{r}_{uic}) + \frac{1}{2}\lambda_1(b_u^2 + b_i^2 + ||U_u||^2 + ||I_i||^2 + ||C_c||^2)]$$

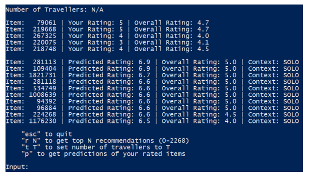
where R is the dataset of ratings, and $L()$ is the loss function. In this case, Absolute Error is used.
As in [10], features and biases are trained on the above function, using Stochastic Gradient Descent, with a learning rate of 0.001, and $\lambda_1 = 0.02$ works well for this dataset. We iterate through the dataset 1000 times.
User input/output is done through a basic console. Users will



```
Welcome, user 5C28F393B23BB894523AE7126A7AE445

    "esc" to quit
    "r N" to get top N recommendations (0-2268)
    "t T" to set number of travellers to T
    "p" to get predictions of your rated items

Input: _
```



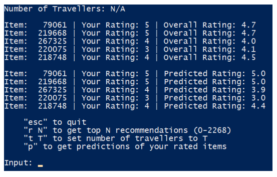
```
Number of Travellers: N/A

Item:   79061 | Your Rating: 5 | Overall Rating: 4.7
Item:  219668 | Your Rating: 5 | Overall Rating: 4.7
Item:  267325 | Your Rating: 4 | Overall Rating: 4.0
Item:  220075 | Your Rating: 3 | Overall Rating: 4.1
Item:  218748 | Your Rating: 4 | Overall Rating: 4.5

Item:  281113 | Predicted Rating: 6.9 | Overall Rating: 5.0 | Context: SOLO
Item:  109404 | Predicted Rating: 6.9 | Overall Rating: 5.0 | Context: SOLO
Item: 1821731 | Predicted Rating: 6.7 | Overall Rating: 5.0 | Context: SOLO
Item:  281118 | Predicted Rating: 6.6 | Overall Rating: 5.0 | Context: SOLO
Item:  534749 | Predicted Rating: 6.6 | Overall Rating: 5.0 | Context: SOLO
Item: 1008639 | Predicted Rating: 6.6 | Overall Rating: 5.0 | Context: SOLO
Item:   94392 | Predicted Rating: 6.6 | Overall Rating: 5.0 | Context: SOLO
Item:   96884 | Predicted Rating: 6.6 | Overall Rating: 5.0 | Context: SOLO
Item:  224268 | Predicted Rating: 6.6 | Overall Rating: 4.5 | Context: SOLO
Item: 1176230 | Predicted Rating: 6.5 | Overall Rating: 4.0 | Context: SOLO

    "esc" to quit
    "r N" to get top N recommendations (0-2268)
    "t T" to set number of travellers to T
    "p" to get predictions of your rated items

Input:
```



```
Number of Travellers: N/A

Item:   79061 | Your Rating: 5 | Overall Rating: 4.7
Item:  219668 | Your Rating: 5 | Overall Rating: 4.7
Item:  267325 | Your Rating: 4 | Overall Rating: 4.0
Item:  220075 | Your Rating: 3 | Overall Rating: 4.1
Item:  218748 | Your Rating: 4 | Overall Rating: 4.5

Item:   79061 | Your Rating: 5 | Predicted Rating: 5.0
Item:  219668 | Your Rating: 5 | Predicted Rating: 5.0
Item:  267325 | Your Rating: 4 | Predicted Rating: 3.9
Item:  220075 | Your Rating: 3 | Predicted Rating: 3.0
Item:  218748 | Your Rating: 4 | Predicted Rating: 4.4

    "esc" to quit
    "r N" to get top N recommendations (0-2268)
    "t T" to set number of travellers to T
    "p" to get predictions of your rated items

Input: _
```

be prompted to login as one of 2371 users. After logging in, users will be able to set the number of travellers, and get a list of top N rated items, as well as see the user's previously rated items. For demonstration purposes, users will also be able to see the predicted ratings of any items the user has previously rated. Updating the context will change the top N predicted items.

Evaluation Results

	100-Recs	Boundary
Precision	.510	.786
Recall	.022	.817
MAE	.096	

TABLE I
RESULTS FOR ICAMF-ALT

This ICAMF-I implementation (denoted as ICAMF-Alt in the tables of results) was configured to train over 1000 iterations of the dataset, where the dataset is randomly split such that each user has at least 1 rating in the test dataset, and approximately one fifth of their ratings in the test dataset. As a result, some items were initiated in the matrix with no ratings, hence some items may fail to be recommended properly. Given a sufficiently large dataset, this should not happen.

In gathering the confusion matrix, it was found that the predictions are heavily skewed positively, with many predicted ratings exceeding 5. Although these predicted ratings could be constrained to 5, it does not change the fact that many items end up highly recommended, hence a large number of predictions are required before any true positives could be detected. Hence, a secondary test was ran to generate a confusion matrix, using a rating boundary rather than a fixed number of recommendations. Ratings above said boundary would correspond to items that the user should find useful, and those below the boundary are useless recommendations. The boundary selected for this test is a rating of 3.5, and results can be seen in II. A confidence matrix for N predictions is also given in III. Full results for ICAMF-Alt can be seen in I The MAE measured from ICAMF-Alt is very promising, implying that all predictions on learnt ratings are within .1 of the actual ratings. However, the recall of ICAMF-Alt is lacking, in that it requires a significant N before recommending an item that was hidden from training. Perhaps if we filtered recommendations according to a user-defined location variable, this result would improve significantly. The precision is also not impressive at $N = 100$, with an approximately 50% precision rate, showing mediocre ability to distinguish between good and bad recommendations. However, setting a rating boundary and getting predictions for all items does show that positively rated items are generally predicted as good recommendations, and vice versa.

	+ve Recommendation	-ve Recommendation
Rated	1697	381
Not Rated	462	100

TABLE II
CONFUSION MATRIX OF RESULTS, USING A FIXED BOUNDARY

	+ve Recommendation	-ve Recommendation
Rated	26	1164
Not Rated	25	1425

TABLE III
CONFUSION MATRIX OF RESULTS, USING A FIXED NUMBER OF RECOMMENDATIONS, N

Conclusion

This paper has briefly surveyed the different CARSs currently available, and has implemented, with reasonable success, a working CARS, capable of making recommendations with regards to a single context. As mentioned throughout the paper, the chosen dataset was limited in the number of ratings each user/item had, resulting in poorer modelling of certain users and items.

Future Developments

This particular implementation, ICAMF-Alt, can be improved by supporting more than a single context. In such a development, a different dataset will be used for training. Given a sufficient amount of time, a significantly larger dataset, such as MovieLens or #nowPlayingRS, could be used. Other developments may include supporting online usage, where new users/items are being added at all times. One specific fault to investigate with the current system is the predictions falling outside of the 1-5 range.

References

[1] L. Baltrunas and F. Ricci. Context-Dependent Items Generation in Collaborative Filtering. ACM RecSys'09. 2009.

[2] V. Codina, F. Ricci and L. Ceccaroni. Distributional Semantic Pre-filtering in Context-Aware Recommender Systems. User Modeling and User-Adapted Interaction. 2015.

[3] P. Cremonesi, P. Garza, E. Quintarelli and R. Turrin. Top-N recommendations on Unpopular Items with Contextual Knowledge. CARS-2011. 2011.

[4] I. Fernandez-Tobias, P.G. Campos, I. Cantador and F. Diez. A Contextual Modelling Approach for Model-Based Recommender Systems. Advances in Artificial Intelligence. Lecture Notes in Computer Science, pp. 42-51. 2013.

[5] K. Oku, S. Nakajima, J. Miyazaki and S. Uemura. Context-Aware SVM for Context-Dependent Information Recommendation. MDM'06. 2006.

[6] L. Baltrunas, B. Ludwig and F. Ricci. Matrix Factorization Techniques for Context Aware Recommendation. RecSys'11. 2011.

[7] A. Livne, M. Unger, B. Shapira and L. Rokach. Deep Context-Aware Recommender System Utilizing Sequential Latent Context. RecSys'19. 2019.

[8] Y. Zheng, B. Mobasher and R. Burke. Contexts Recommendation Using Multi-label Classification. WI 2014. 2014.

[9] G. Adomavicius, R. Sankaranarayanan, S. Sen and A. Tuzhilin. Incorporating Contextual Information in Recommender Systems Using a Multidimensional Approach. ACM Transactions on Information Systems. 2005.

[10] J. Li, P. Feng and J. Lv. ICAMF: Improved Context-Aware Matrix Factorization for Collaborative Filtering. 2013 IEEE 25th International Conference on Tools with Artificial Intelligence. 2013.