Collaborative Context-aware Preference Learning

Alexandros Karatzoglou
Telefonica Research
Barcelona, Spain
alexk@tid.es

Linas Baltrunas
Telefonica Research
Barcelona, Spain
linas@tid.es

Matthias Böhmer
German Research Center for AI
Kaiserslautern, Germany
matthias.boehmer@dfki.de

Abstract

Preference learning methods work by exploiting patterns in the data that relate users to items. Preference data often includes information such as the context of a recommendation (e.g. time/date, location). Leveraging this data (e.g. click logs, purchase/usage data) can significantly improve the relevance and quality of the recommendation. In this work we introduce a novel scalable context-aware collaborative filtering approach that is based on Tensor Factorization where additional information is represented by additional dimensions in the Tensor. The algorithm is tested on data from the Android mobile application recommendation service *appazaar*[1] and from the *Last.fm*[2] music service where it compares favorably with state-of-the-art collaborative filtering methods.

1 Introduction

Computing devices are becoming ever more mobile with the new generation of smartphones and tablets set to replace the personal computer as the computing device of choice and the main tool for on-line information access for most of the population. Mobile devices offer access to the context of the user which can be used for better (context-aware) recommendations. Moreover smartphones have the ability to collect a large amount of preference data and context variables without been intrusive.

Early work on context-aware preference learning techniques has shown that contextual factors such as (*e.g.* time, location, social context, activity, weather, emotional state, social network, etc.) influence heavily the recommendation needs of users [1]. The influence of context can be quite obvious e.g. travel and vacation recommendations influenced by season, but can be also more subtle such as e.g. mood influencing the type of music one would prefer.

In the data from the Android mobile application recommendation service *appazaar*, we found that time of the day plays a role in the type of app usage. While during the day mobile devices are mainly used as communication devices and much more in terms of general usage time, during the night the use of mobile apps is more diverse. And while news apps for instance have their highest share in the morning, social apps have the highest share in the evening. Location has been also shown to be very important for mobile app usage [7].

Some CF models have been built that incorporate the temporal dimension [9], but they do not personalize the effects of the temporal variable but rather model global time effects. In our past work [8] we introduced a Context-Aware Collaborative Filtering model for explicit data (i.e. ratings) that is based on Tensor Factorization (TF) which outperforms state-of-the-art context aware collaborative filtering methods. However, as mentioned previously CF models for explicit data (i.e. ratings) are not adequate for the case of implicit data (i.e. counts). Moreover in this work we use a new Tensor

[1] http://www.appazaar.net
[2] http://Last.fm

model that allows the use of both implicit feedback data (i.e counts) and explicit data (i.e. ratings) and thus supports faster optimization.

In this work we generalize efficient factor model approaches to the Context-Aware case in a compact way, train the model with a fast and straightforward algorithm that scales linearly to the number of available implicit data, that can include any number of contextual variables into the model itself.

2 Tensor-decomposition Model

A Tensor is a generalization of a matrix to multiple dimensions. N-dimensional TF extends CF to N-dimensional data where the additional dimensions can represent the context of the recommendation. Tensor Factorization (TF) can be used to add any number of variables to a CF-based Recommender System by means of additional dimensions to the tensor.

Notation For the sake of simplicity, we will describe the model for a single contextual variable C, and therefore the tensor Y containing the ratings will be a 3-dimensional tensor. In the following we denote the tensor of count observations by $Y \in \mathbb{N}^{n \times m \times c}$, where n are the number of users, m the number of items, and c where $c_i \in \{1, \ldots, c\}$ the number of contextual variables. Typically, counts are represented in integer values scale and thus $Y \in \mathbb{N}^{n \times m \times c}$, where the value 0 indicates that a user did not purchase/interact with an item. In this sense, 0 is special since it does *not* necessarily indicate that a user dislikes an item but rather that there was no interaction. Finally, we denote by U_{i*} the entries of the i_{th} row of matrix U.

Candecomp-Parafac decomposition The Candecomp- Parafac (CP) [5] model, is a tensor decomposition model shown in where e.g. a 3-dimensional tensor Y is decomposed into three matrices $U \in \mathbb{R}^{n \times d}$, $M \in \mathbb{R}^{m \times d}$ and $C \in \mathbb{R}^{c \times d}$ and the decision function for a *user i, item j, context k* is given by

$$F_{ijk} = \langle U_{i*}, M_{j*}, C_{k*} \rangle = \sum_{l=1}^{d} U_{il} M_{jl} C_{kl} \tag{1}$$

One of the advantages of the CP decomposition model compared to other tensor decomposition models (HOSVD, etc) is its simplicity which will allow us to use an analytical expression for solving the decomposition problem and achieve linear scalability. We thus propose the following tensor-based latend factor model to model context-aware preferences in a collaborative manner:

$$\begin{aligned} \min_{U,M,C} \sum_{i}^{n} \sum_{j}^{m} \sum_{k}^{c} [w_{ijk} \left(p_{ijk} - \langle U_{i*} M_{j*} C_{k*} \rangle\right)^2 \\ + \frac{\lambda}{n} ||U_{i*}||^2 + \frac{\lambda}{m} ||M_{j*}||^2 + \frac{\lambda}{c} ||C_{k*}||^2] \end{aligned} \tag{2}$$

here p_{ijk} signals the existance or not of $user - item - context$ interactions that is $p_{ijk} = 1$ if an interaction exists $p_{ijk} = 0$ otherwise. The term $\frac{\lambda}{n}||U_{i*}||^2 + \frac{\lambda}{m}||M_{j*}||^2 + \frac{\lambda}{c}||C_{k*}||^2$ is required for regularization. Note that we scale the regularization parameter with the dimensionality of each factor matrix. This is particularly important in the case of Tensor Factorization for Context-Aware Collaborative Filtering since typically the context factor matrix C is much smaller than the factor matrices U and M since there are usually less contextual states (e.g. time of the day, state of the weather, activity) than users or items in the dataset. Thus the contribution of the Frobenius norm of the factor matrices, needs to be scaled. The actual value of the λ parameter can be found using tuning techniques and cross-validation.

Note moreover that we minimize a weighted version of the squared error loss where w_{ijk} is a weight that is used on the interaction between user i and item j that we define as follows:

$$w_{ijk} = \begin{cases} \alpha log(1 + Y_{ijk}) + log(1 + \frac{m}{m_i + 1}) & Y_{ijk} > 0 \\ 1 & Y_{ijk} = 0 \end{cases} \tag{3}$$

where m_i is the number of items used by user i and α a parameter which we set to 10 for all our experiments. The first term in eq. 3 reflects the confidence regarding items that are often being consumed while the second term reflects the fact that confidence in items should be high if only very few items are used.

The reason for the use of weighted loss function is to deal with the bias introduced by the fact that our data is based exclusivly on observed interactions between *users* and *items* and *context* that is whe only know which items the user interacts with but we do not have a clear indication of which items the user *dislikes*. From the data it is unclear if a users choose not to interact with an item because he does not like it or bechause he is not aware of it thus we cannot consider all missing items from a users profile as negative feedback. At the same time unless we add some kind of negative feedback in the model we are going to end up with an positivly biased estimator that will predict interactions between most *users* and *items* something that is not supported by the data. For this reason we need to take into account the non-observed 0 entries of the counts tensor Y. To this end we use a weighted loss function where the weights of the entries are given by equation 3. Using a naive optimization procedure over the whole tensor Y would not be scalable, we thus show how to deal with this large problem in the next section where we introduce an optimization procedure that scales linearly to the number of observed entries. Note that state-of the art factor models (e.g. [11], [10]) that deal with rating input do not take the non-observed entries into account.

2.1 Optimization

We optimize the objective function for Tensor Factorization 2 using Alternating Least Squares. When trying to optimize over a single factor matrix while keeping the remaining factor matrices fixed we observe that the cost function becomes quadratic and that there is an analytic solution. We first illustrate how to optimize the objective with respect to the user factor matrix U. We differentiate the objective function and set the derivative to zero. Solving with respect to a single user i factor vector gives:

$$U_{i*} = \left(\sum_{j}^{m} \sum_{k}^{c} \left[M_{j*} \odot C_{k*} \right]^T w_{ijk} \left[M_{j*} \odot C_{k*} \right] + \frac{\lambda}{n} I \right)^{-1} \times \sum_{j}^{m} \sum_{k}^{c} \left[M_{j*} \odot C_{k*} \right]^T w_{ijk} p_{ijk} \tag{4}$$

where $\odot$ is the Hadamar or element-wise product and I the identity matrix of size d. Directly computing this expression would scale $O((nc)^2 d)$ which would be prohibitively expensive even for relatively small datasets. We can achieve a very significant speedup by writing the expression by rewriting this expression as:

$$\sum_{j}^{m} \sum_{k}^{c} \left[M_{j*} \odot C_{k*} \right]^T w_{ijk} \left[M_{j*} \odot C_{k*} \right] = \tag{5}$$

$$\sum_{j}^{m} \sum_{k}^{c} \left[M_{j*} \odot C_{k*} \right]^T \left[M_{j*} \odot C_{k*} \right] + \sum_{j}^{m} \sum_{k}^{c} \left[M_{j*} \odot C_{k*} \right]^T \left(w_{ijk} - 1 \right) \left[M_{j*} \odot C_{k*} \right] \tag{6}$$

The first part $\sum_{j}^{m} \sum_{k}^{c} \left[M_{j*} \odot C_{k*} \right]^T \left[M_{j*} \odot C_{k*} \right]$ is now independent of the user and can be precomputed at the beginning of the iteration over each user factor matrix. In the second part $\sum_{j}^{m} \sum_{k}^{c} \left[M_{j*} \odot C_{k*} \right]^T \left(w_{ijk} - 1 \right) \left[M_{j*} \odot C_{k*} \right]$ the expression $(w_{ijk} - 1)$ is 0 for all the non-observed values in the tensor Y which are the vast majority of entries in the Tensor and we thus can compute it over the non-zero values S_i^{U+} of user i in the Tensor Y. This vastly improves computation time and scalability to $O((d)^2 n^{U_i+} + d^3)$ where n^{U_i+} the number of positive feedback items for user i and assuming cubic scalability for the matrix inversion. Note that more scalable matrix inversion techniques would provide limited benefits in this case since the dimension of d is

typically small $10 - 40$. We can thus rewrite this expression as:

$$\sum_{j}^{m}\sum_{k}^{c}\left[M_{j*}\odot C_{k*}\right]^{T} w_{ijk}\left[M_{j*}\odot C_{k*}\right] = \tag{7}$$
$$\sum_{j}^{m}\sum_{k}^{c}\left[M_{j*}\odot C_{k*}\right]^{T}\left[M_{j*}\odot C_{k*}\right] +$$
$$\sum_{j,k\in S_{i+}^{U}}\left[M_{j*}\odot C_{k*}\right]^{T}\left(w_{ijk}-1\right)\left[M_{j*}\odot C_{k*}\right]$$

Since we need to compute this over each user the scalability becomes $O(d^2N + d^3n)$.

Lemma 1 *We show that:*

$$\sum_{j}^{m}\sum_{k}^{c}\left[M_{j*}\odot C_{k*}\right]^{T}\left[M_{j*}\odot C_{k*}\right] = M^{T}M\odot C^{T}C \tag{8}$$

Proof

$$\sum_{j}^{m}\sum_{k}^{c}\left[M_{j*}\odot C_{k*}\right]^{T}\left[M_{j*}\odot C_{k*}\right] =$$
$$\sum_{j}^{m}\sum_{k}^{c}\left[M_{j*}\otimes M_{j*}\right]\odot\left[C_{k*}\otimes C_{k*}\right]$$

$$\sum_{j}^{m}\left[M_{j*}\otimes M_{j*}\right] = M^{T}M \quad \sum_{k}^{c}\left[C_{k*}\otimes C_{k*}\right] = C^{T}C$$

where $\otimes$ is the outer product. ■

Lemma 1 is used to speedup computations since matrix libraries handle this type of operations (matrix multiplications and element-wise products) very efficiently. Once the user factors U are computed we can compute the item M and context factors C in a similar manner. The optimization procedure is repeated over each factor matrix until convergence. The whole algorithm scales $O(d^2N + d^3(n + m + c))$ and usually $N \gg (n + m + n)$. We typically run 10 iterations of the optimization procedure over the factor matrices.

3 Experiments

We evaluate the methods on two datasets:

appazaar *appazaar* is a recommender system that suggests mobile applications to its users that is available on the Android Market Store. The relevance of an application strongly depends on the user's current context. Context-aware Recommendation is thus particularly relevant for *appazaar*. At the time of writing there are more than 150,000 applications available for Android smart phones. *appazaar* traces mobile application usage in parallel with available context information as a basis for context-aware recommendations [2]. In total, the *appazaar* dataset contains 3,260 users and 18,205 items and 3.7 million records about the usage of applications. The features that can be extracted in addition to the user and item id's are:

- **Moving**: Whether the user was moving with walking speed (3), faster (4) or standing still (2); or this information is not available (1).
- **Location**: A heuristic whether the user is at home, at work or elsewhere. We have set the most frequent place from 6am to 6pm as home (1), the most frequented place from 6pm to 6am as home (2), and defined all other locations as elsewhere (3).

- **Time of day**: The time of the day in blocks of 2 hours, from 12pm-2am (1) to 10pm-12pm (12).
- **Day of the week**: From Sunday (1) to Saturday (7).
- **Number of times used**: The number of times which the application was used by the user with regard to the other parameters.
- **Total time used**: The accumulated time which the application was used by the user with regard to the other parameters.

Note that we have 6 context variables (including user and app id) we thus model the data with a 6-dimensional Tensor.

Last.fm data *Last.fm* is a music website with over 40 million active users where users can create playlists and can listen to audio track's from *Last.fm's* music library on demand. The data we use is based on the data that was made available by [3]. While the original data contains information on the song listened we modify the data in order to provide info on the album based on the album that the song belongs to. Moreover we extract date and time information from the time stamp while also including the users gender information. In total the *Last.fm* data contains: contains 884 users and 193218 items and 4901416 listening records. The context features (in addition to user and album id) that can be extracted are as follows.

- **Gender**: The gender of the user
- **Month**: The month the album was listened
- **Day**: The day of the week the album was listened
- **Hour**: The hour of the day of listening

3.1 Evaluation Protocol

We temporally order the data and split it with the first 80% of the data forming the training set and the remaining 20% the test set. The data was then aggregated for each contextual combination found in the data set. For example, user u used application i while being still at home, between 6pm to 6am on Weekend 25 times and used it in total for 49 minutes. In order to facilitate the comparison to non-context aware methods we filtered the test set so that for each $user - item$ combination only one context combination is in the test set.

For the testing procedure we adopt a similar strategy to [4]: We first randomly select 1000 additional items that the user did not use/listen. We can assume that most of them will not be of interest to user u. We predict the scores for the test item j and for the additional 1000 items. We form a ranked list by ordering all the items according to their predicted scores. We then form ranked lists of items and compute the Mean Average Precision (MAP)

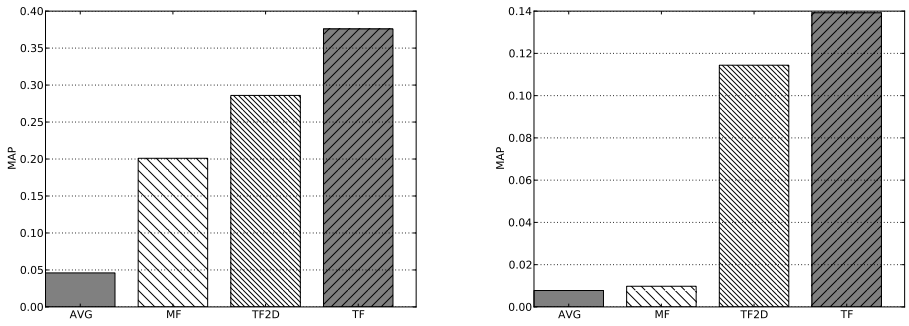
3.2 Methods in Comparison

We compare our method (denoted TF in all the Figures) to 2-dimensional TF (*TF2D*), which takes into account only users and items and ignores the additional context in the data but treats the 0 values of the non-observed $user - item$ pairs as negative feedback. This method is essentially matrix factorization for implicit feedback data and corresponds to the method proposed in [6]. We also compared the results against a standard Regularized Matrix Factorization method (*MF*) (based on Simon Funk's[3] approach) where the 0 values are ignored and a regression is performed on the p_{ijk} values derived from eq. 3. Naturally, *MF* also ignores the contextual information. As a baseline we used the overall popularity of the apps for recommending items to the users (*AVG*). The popularity is computed by averaging the usage counts for each application, over all the users and each context.

3.3 Experimental Results

We first compute the MAP of all the methods on the data after tuning. Results are shown in Figure 1. TF outperforms the other methods in particular in the *appazaar* data. Interestingly, *MF* shows poor

[3] `http://sifter.org/~simon/journal/20061211.html`



(a) appazaar (b) Last.fm

Figure 1: Ranking performance of the methods measured in MAP for the *appazaar* 1(a) and the *Last.fm* 1(b) data.

performance. More noticeable, TF improves *MAP* over the non-context aware method *TF2D* by 31% for the *appazaar* data and 19% for the *Last.fm* data.

References

[1] G. Adomavicius and A. Tuzhilin. *Recommender Systems Handbook*, chapter Context-aware Recommender Systems. Springer, 2010.

[2] M. Boehmer, M. Prinz, and G. Bauer. Contextualizing mobile applications for context-aware recommendation. In *Adj. Proc. of Pervasive 2010*, 2010.

[3] O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.

[4] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 39–46, New York, NY, USA, 2010. ACM.

[5] F. L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6, 1927.

[6] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM '08: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, pages 263–272, Washington, DC, USA, 2008. IEEE Computer Society.

[7] Eija Kaasinen. User needs for location-aware mobile services. *Personal Ubiquitous Comput.*, 7:70–79, May 2003.

[8] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *RecSys '10: Proceedings of the fourth ACM conference on Recommender systems*, pages 79–86, New York, NY, USA, 2010. ACM.

[9] Yehuda Koren. Collaborative filtering with temporal dynamics. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 447–456, New York, NY, USA, 2009. ACM.

[10] N. Srebro, J. Rennie, and T. Jaakkola. Maximum-margin matrix factorization. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, Cambridge, MA, 2005. MIT Press.

[11] Gabor Takacs, Istvan Pilaszy, Bottyan Nemeth, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research*, 10:623–656, 2009.