# Context-Aware Recommender Systems

1st Louis Robinson
*Computer Science Department*
*University of Durham*
Durham, UK
louis.c.robinson@durham.ac.uk

***Abstract*—In this report I outline what a context-aware recommender system is, and compare two examples of context-aware recommender systems on a subset of the #NowPlaying song dataset [1] on the metrics: precision, recall, MAE, RMSE, ROC, novelty and coverage. The first recommender system technique I demonstrate uses Singular Value Decomposition (SVD) on the users-items ratings matrix and then post filtering to incorporate the context. The second technique is a type of contextual modelling, the rating prediction is an implementation of tensor factorisation (TF) [2] to model the users-item-context ratings matrix. In this second recommender system the recommendation function select items using a Determinantal Point Process (DPP) [3] the sampling process accounts for quality and diversity of the items. My code to accompany this report can be found on my GitHub [4].**

## I. Introduction

Context is regarded as an important factor to consider for Recommender Systems (RS), but there is no standardised way to incorporate it. The methods to include context fall into the categories: contextual modelling, pre-filtering or post-filtering. Generally contextual modelling and pre-filtering are viewed as more promising, however less research has focused on post-filtering [5]. I will outline a contextual modelling and a post-filtering approach and compare them.

### A. Domain

I experiment on a dataset in the music domain, #NowPlaying. This is a large dataset of listening events, from Spotify users, combined with information in the hashtag from tweets from each user. Spotify's API allows contextual factors to be extracted with the listening event such as location and time etc.

### B. Related work review

Work in context-aware recommender systems such as collaborative filtering techniques is well summarised up to 2017 by Villegas [5]. In my report I create a similar SVD model to that in [6] for my first RS which I will refer to as SVD-PF. The second RS technique is based on Tensor Factorisation [2] for rating prediction, and the recommend function uses a Determinantal Point Process (DPP) [3] to sample items for recommendation.

### C. Purpose / Aim

To be able to research novel context aware recommendation techniques and demonstrate the strengths and weaknesses for different aspects of the recommender systems.

## II. Methods

I use $u \in \{1, ..., n\}$, $t \in \{1, ..., m\}$, $c \in \{1, ..., p\}$ to index users, items and a context. Let $r \in \{0, 1\}^{n \times m}$ or $r \in \{0, 1\}^{n \times m \times p}$ be the user-item or user-item-context interaction matrix corresponding to the rating each user has given each item in each context.

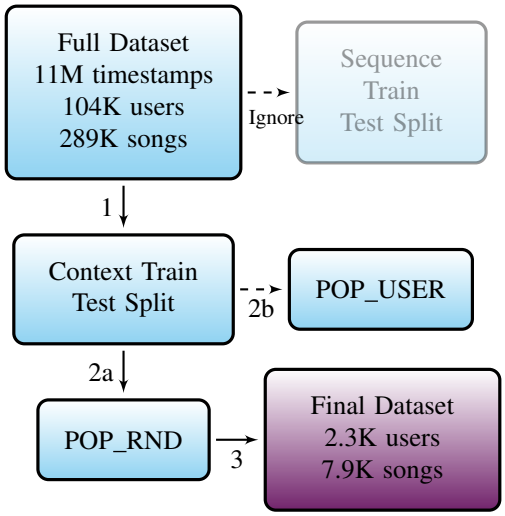


Fig. 1. Process the make the final dataset.

### A. Data type, source

The full datasets have been scrapped from APIs collecting tweets and Spotify listening events. This provides data for the context of a user, such as their location, time and mood (in the hashtag). To clean the data, [1] has first filtered for users and songs with over 10 listening events as well as removing hashtags without mood information (e.g. #NowPlaying or #ListeningLive etc.) see step 1 in Figure 1. The next step involved creating positive and negative examples from the implicit dataset, [1] decided to create the two datasets, 'POP_RND' and 'POP_USER'. Creating 'POP_RND' (step 2a in Figure 1) gave the positive listening events a rating of 1 and for each positive example added 9 tracks a user had not listened given a rating of 0.

To make the final dataset I use, only requires the user, track, location, month and rating columns for each listening event. This is shown in step 3 in Figure 1. I select a subset of the listening events in 'POP_RND' so that there is at least one user from each location, and set this as my final dataset (see file 'WriteFinalDataSet.py' on my GitHub [4]). I include the 'POP_USER' in Figure 1 as it is a dataset produced by for each positive example adding negative examples for the same track in contexts the user hasn't been in (step 2b) it would be interesting to test on this dataset.

### B. Feature extraction and selection methods

I selected the contexts location and month as these can easily be obtained implicitly from Spotify users (however in

my GUI the context location and month must be entered explicitly). Location and time have been shown to be effective features [5]. Another context shown to be a good feature is mood, in this case hashtag, however it was only available for a small proportion of listening events and therefore doesn't help learn about the majority users who don't tweet, so I chose not to include it to reflect the proportions in the original dataset. Since there is a large amount of data for listening events, not much user data and restricted item data (tempo, acousticness, key etc.) I choose collaborative filtering techniques instead of content based methods.

### C. User profiling and prediction methods

Both techniques use collaborative filtering, learning how to predict values in the matrix of ratings. Technique 1, involves Singular Value Decomposition to represent the interaction of a user and item ... to predict the rating a user would give to an item and then I use post-filtering to .

$$\hat{r}_{ut} = \mu + b_t + \beta_u + U_{u\cdot} \cdot T_{t\cdot} \,.$$

which require learning $nd_n + md_m + n + m + 1$ parameters.

Technique 2 is a contextual modelling technique which uses a multidimensional users-items-context representation called Tensor Factorisation [2]. I show the equations for adding a single contextual variable, adding more is trivial. Let $A = S \times_U U$ be the tensor product $A_{ujk} = \sum_{i=1}^{d_n} S_{ijk} U_{ui}$.

$$\hat{r}_{utc} = S \times_U U_{u\cdot} \times_T T_{t\cdot} \times_C C_{c\cdot} \,. \tag{1}$$

So the ratings matrix is factorised into 3 matrices $U \in \mathbb{R}^{n \times d_n}, T \in \mathbb{R}^{m \times d_m}, C \in \mathbb{R}^{p \times d_p}$ and one central tensor $S \in \mathbb{R}^{d_n \times d_m \times d_c}$, see Figure 2.

For both methods, to learn the parameters I use stochastic gradient descent (SGD) with the squared loss function. Define,

$$\ell(\hat{r}_{ut}, r_{ut}) = \frac{1}{2}(\hat{r}_{ut} - r_{ut})^2 \,.$$

To perform SGD to learn parameters we take a step in the opposite direction to the gradient (gradient of the loss with respect to the parameter we want to update) proportional to the loss value we achieved. For example the update for updating $U$ in SVD is,

$$\nabla_U \ell(\hat{r}_{ut}, r_{ut}) = (\hat{r}_{ut} - r_{ut}) \nabla_U \hat{r}_{ut} = (\hat{r}_{ut} - r_{ut}) T_{t\cdot} \,, \tag{2}$$

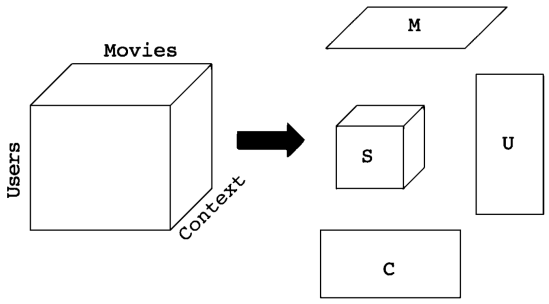


Fig. 2. Tensor Factorisation [2] (replace 'Movies' with 'Songs').

with a similar calculation for $\nabla_T \ell(\hat{r}_{ut}, r_{ut})$ and even simpler for the derivatives with respect to $\mu, b_t, \beta_u$. For the tensor factorisation equations the relevant gradients are $\nabla_{U_{u\cdot}} \ell(\hat{r}_{utc}, r_{utc}), \nabla_{T_{t\cdot}} \ell(\hat{r}_{utc}, r_{utc}), \nabla_{C_{c\cdot}} \ell(\hat{r}_{utc}, r_{utc})$ and $\nabla_S \ell(\hat{r}_{utc}, r_{utc})$. I present two for clarity, $\otimes$ it the tensor outer product,

$$\nabla_{U_{u\cdot}} \ell(\hat{r}_{utc}, r_{utc}) = (\hat{r}_{utc} - r_{utc}) S \times_T T_{t\cdot} \times_C C_{c\cdot} \,, \tag{3}$$

$$\nabla_S \ell(\hat{r}_{utc}, r_{utc}) = (\hat{r}_{utc} - r_{utc}) U_{u\cdot} \otimes T_{t\cdot} \otimes C_{c\cdot} \,, \tag{4}$$

the gradients with respect to $T$ and $C$ are similar to U. I have outlined how the prediction methods work and the parameter updates to improve the accuracy. In both techniques the user profile is a learnt row vector in the matrix $U$, $U_{u\cdot} \in \mathbb{R}^{d_n}$ (also combined with the element $\beta_u$ for SVD).

### D. Evaluation methods

The recommendation algorithms can be tested offline by splitting the 973K listening event examples in the dataset into 80% training examples and 20% testing examples. I let SVD train with a learning rate $\alpha = \min(0.3, \frac{10^4}{t})$, for $10^7$ steps (the loss usually drops to a reasonable number around $10^6$ steps $\sim$ 1 min) where a step is a single example whether a user liked an item. For TF I use $\alpha = \min(1, \frac{5 \times 10^4}{t})$, for $5 \times 10^6$ steps (about 20 mins). However these parameters can be varied and the algorithm will usually still converge to a reasonable minima. This allows us to measure the rating prediction accuracy metrics MAE (Mean Absolute Error), RMSE (Root Mean Squared Error) see Table I. The accuracy of usage predictions (whether RS accurately predicts that a user will view the recommendations) can be calculated: recall, precision and Receiver Operating Characteristic (ROC) Curves see Figure 4.

To determine properties of the recommendations that are output I select a single user and a single context and continuously recommend items to the user, this allows the measurement of novelty and coverage. Novelty is desirable in a RS as the ability to recommend items that are less popular/unseen by others allows all items to be presented. Catalog coverage [7] shows how well the recommendations output cover the total items available.

## III. Implementation

### A. Recommendation algorithm

Here I outline the recommend functions I use for the SVD and TF recommenders. For SVD I choose to use post filtering. This is done by preparing a list of potential items to recommend (using the SVD predicted ratings), and then using a function to select a subset of the recommendations that are the most promising. For a new user I do this by selecting popular songs in the current context they are in (requires storing an array for each context with the popularity of each item in that context), as well as a selection of novel items chosen with a probability $\mathbb{P} \propto \frac{1}{1+p_t}$, where $p_t$ is the total popularity of track $t$. For a user with some ratings I return a few tracks which they haven't listened to but the model predicts that they would rate highly (similar users liked that song), some tracks they have previously liked in their current context (if none some track they have previously liked in the next most similar context) and finally some novel items. The number that these different categories (novel/similar users/previously liked) of items are presented is adaptively changed depending on which of the categories they have previously liked (e.g. if they tend to like more novel items more novel items are presented).

The second technique generates recommendations by sampling a Determinantal Point Process (DPP) [3] on the set of

unseen items and a subset of size 10 of the seen items (for a new user it recommends in the same way as as the above technique). A DPP is a method for sampling a set of items from a base set which can encode negative correlation between items, so if an item is picked, we are less likely to also pick similar items in the sample. The similarity of items for the DPP is modelled by a (unit) diversity vector associated with each of the items, how similar those items are is equal to the dot product of their diversity vectors. We may want to favour certain items, this is done by associating each item with a quality (constant which scales that item's diversity vector). I use the latent representation of the items as the diversity vectors (rows of T) and the predicted rating $\hat{r}$ plus the normalised item popularity for the quality.

### B. Output (recommendations/predictions) presentation

The GUI I prepared can be run from the terminal, it first gives the option to select the text size and colour scheme, this is because some users may have difficulty reading smaller text or colour blindness. Finally it asks which recommender system you would like to use SVD-PF or TF-DPP. The GUI is loaded and the user can then enter their username into the input box and explicitly enter their context from two drop down boxes. If they are a returning user, the associated user profile is loaded and recommendations will be based on their history of likes and dislikes. If the username is not in the database, a new user profile is made and the recommendations are made as detailed above. User profiles are updated based on explicitly provided likes and dislikes. Since this is an app for a computer, I provide 10 recommendations as default, however if the user generally reviews only 1 or 2 items then only 5 recommendations are presented (if the start to review more it goes back up again). Each recommendation contains a note saying why that particular recommendation was presented.

## IV. Evaluation results

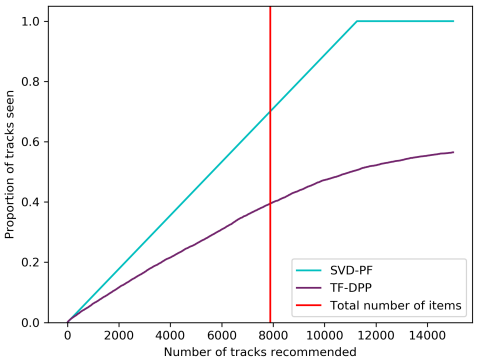


Fig. 3. Catalog coverage, given a user and a context, continuously recommend items and record the proportion of all the items that have been recommended.

It is desirable for a RS in the music domain to be able to recommend all songs, even those without many ratings (unpopular items) Figure 3 shows how well each recommender system covers all of the possible items. In [8] they use session-based recommendation algorithms on the #NowPlaying dataset and achieve a catalog coverage of 0.580. Note that the reason SVD-PF does so well is because the post filtering component forces unseen its to be recommended on average $\frac{7}{10}$ items are unseen per recommendation (some highly rated but unseen, some chosen because they're novel). I use predicted coverage as defined in [7], the number of items a recommender can make predictions about divided by the total number of items. In both methods the matrices are initialised to random values close to 0 and so a prediction can always be made, however if there is no data for an item the prediction will be inaccurate. By definition the predicted coverage is 1 for both methods (in some cases CF approaches set a minimum number of ratings for an item to make predictions for it). The user-space coverage is 1 all users can be recommended to, because the matrices are initialised close to 0, and as for predicted coverage, the ratings for that user will not be accurate without data. However in this case recommendations are based on popular items in the context that the new user is in, as well as suggesting a smaller number of more novel items.

TABLE I
RESULTS

| Recom. System | Ratings prediction | | Usage prediction | | Other |
|---|---|---|---|---|---|
| | *MAE* | *RMSE* | *Recall* | *Precision* | *Novelty* |
| SVD-PF | 0.00769 | 0.0877 | 0.9721 | 0.9972 | 5.86 |
| TF-DPP | 0.00137 | 0.0370 | 0.9438 | 1.0 | 1.74 |

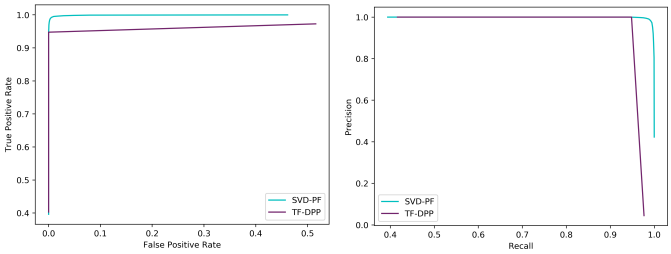


Fig. 4. ROC curves (left), precision-recall curves (right) for SVD and TF-DPP. Note the x, y axes are not 0-1.

## V. Conclusion

I have shown two ways collaborative filtering methods can incorporate context into a recommender system.

### A. Limitations

I have used methods that have linear complexity in the number of users and items, however not in the number of contexts. Other approximations/methods may allow scalable extensions in the number of contexts. Another adaptation that can be made is instead of the dot product in the SVD approach or the tensor products with $S$ in the TF approach non-linear functions can be learnt e.g. a neural network.

### B. Further developments

The catalog coverage results, Figure 3, show that post filtering has control over the types of items presented in the recommendations which is usually desirable. However this is not the full picture and maybe this is detrimental to the quality of recommendations, a user may not want to stay if it is recommended too many novel items and too little relevant items. In the post filtering I have allowed the recommendations to adaptively change the number of novel items. More research is needed to determine the most effective type of feedback.

## REFERENCES


[1] A. Poddar, E. Zangerle, and Y.-H. Yang, "#nowplaying-rs: A new benchmark dataset for building context-aware music recommender systems," in *Proceedings of the 15th Sound & Music Computing Conference*, Limassol, Cyprus, 2018, code at https://github.com/asmitapoddar/nowplaying-RS-Music-Reco-FM. [Online]. Available: http://mac.citi.sinica.edu.tw/ yang/pub/poddar18smc.pdf

[2] A. Karatzoglou, X. Amatriain, L. Baltrunas, and N. Oliver, "Multiverse recommendation: N-dimensional tensor factorization for context-aware collaborative filtering," in *Proceedings of the Fourth ACM Conference on Recommender Systems*, ser. RecSys '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 79–86. [Online]. Available: https://doi.org/10.1145/1864708.1864727

[3] A. Kulesza, "Determinantal point processes for machine learning," *Foundations and Trends® in Machine Learning*, vol. 5, no. 2-3, p. 123–286, 2012. [Online]. Available: http://dx.doi.org/10.1561/2200000044

[4] L. Robinson, *DPPsForRL*, 2020, https://github.com/Louiii/RecommenderSystems.

[5] N. M. Villegas, C. G. Sánchez, J. Díaz-Cely, and G. Tamura, "Characterizing context-aware recommender systems: A systematic literature review," *Knowl. Based Syst.*, vol. 140, pp. 173–200, 2018.

[6] Y. Koren, "Collaborative filtering with temporal dynamics," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 447–456. [Online]. Available: https://doi.org/10.1145/1557019.1557072

[7] M. Ge, C. Delgado, and D. Jannach, "Beyond accuracy: Evaluating recommender systems by coverage and serendipity," 01 2010, pp. 257–260.

[8] M. Ludewig and D. Jannach, "Evaluation of session-based recommendation algorithms," *CoRR*, vol. abs/1803.09587, 2018. [Online]. Available: http://arxiv.org/abs/1803.09587