Play list Recommender: An Ensemble approach

Himanshu Gupta
himgupt2@tamu.edu
Texas A&M University

Mahesh Avadhanam
maheshavadhanam@tamu.edu
Texas A&M University

Nishant Aditya
nishant.aditya@tamu.edu
Texas A&M University

Pranay Dhariwal
pranaydhariwal@tamu.edu
Texas A&M University

ABSTRACT

In this project we have implemented an ensembled approach to build a song recommender system. For a given user, we try to recommend a playlist to the user in the order of expected preference. We do this by using the principles of song popularity, user to user similarity, neural methods, and Bayesian Personalized Ranking. For our recommendation system, we use implicit information retrieved using song listen count for a user-song pair.

ACM Reference Format:
Himanshu Gupta, Mahesh Avadhanam, Nishant Aditya, and Pranay Dhariwal. 2019. Play list Recommender: An Ensemble approach. In *Proceedings of CSCE670 PROJECT*. College Station, TX, USA, 6 pages.

1 INTRODUCTION

Music is the common ground that brings a wide of people together. People share music taste across culture, religion and languages. In the modern era, internet and mobile devices are a big social equalizer. With the presence of internet and online vendors like Spotify, google music and savan, it is important to analyze how these systems work and how they can be improved further. Since the input space is too large with a lot of music options, we need a very robust and precise recommender system for users. We experiment with different approached and ensemble it to further improve the recommender's performance. In this project we consider models based on song popularity, collaborative filtering, SVD/ SVD++, Neural Networks for Collaborative Filtering, and Bayesian Personalized Ranking from Implicit Feedback. Using an ensemble approach will help us to incorporate the benefits of all the models and overcome the demerits of individual methods. We use Million Song Dataset[5] provided by Kaggle to learn about the users and their song preferences.

2 RELATED WORK

Early research in recommendation system depended largely on explicit feedback (like user ratings and explicit likes and dislikes) [3, 8] and collaborative filtering methods. In recent days, with a lot of data being accessible, researchers are more focused toward understanding and using implicit data like user clicks, time spent on a product or song listen count (as in our case). Using implicit data is more practical but challenging [7, 14].

Singular value decomposition has been proposed [4] to learn the feature matrices. But this approach is likely to be prone to overfitting. So, we need to have regularization to penalize the terms with higher coefficients. Now there has been a shift towards neural methods to learn about the data features and to build a model to predict unseen data samples. The most recent work is being done using the technique of autoencoders [10, 11, 13]. A user based AutoRec system tries to learn the most relevant features over the input data variations. This is achieved by training a multilayer neural network for training data with lesser nodes in the hidden layer. This method works well to identify the features in the item and user space. This method has been used recently in many works but mostly focused towards explicit ratings.

Another approach to make recommendations is using Bayesian Personalized Ranking [12] for implicit feedback. The important thing with implicit feedback is that it can handle items which are not rated or consumed by users. Every item has some confidence even if the user has never consumed the item. If a user consumes an item, then the confidence goes up. If a user consumes an item, it shows some preference for the item (regardless of rating). If the user does not consume a item, then there is no preference. BPR focuses to find personalized ranking for a user for all items by trying to maximize its posterior probability. We calculate the probability for a hypothesis with a given event. We calculate this by taking the prior probability of the hypothesis multiplied by the probability of the event. It can be seen as a posterior probability of hypothesis being true given the event. We update the probability of our hypothesis as more information becomes available.

While many existing systems use listen count, we weighed down this to log-factor and used the log-score as the rating. Many recommenders were built on traditional methods like collaborative filtering, SVD etc.,. Hence, we propose to use latest methods like Neural collaborative filtering and Bayesian Personalized Ranking and compare their performance to the traditional systems. Besides, many existing play list recommenders fail to address the cold start problem. To mitigate this, we use a song popularity model to recommend the most popular songs to a user even though no data is available to personalize the recommendations to him/her. To improve the performance further, we demonstrate an ensemble

CSCE670 PROJECT, SPRING '19, TAMU

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00

approach where we use the weighted outputs of these individual models to make a robust prediction/ recommendation.

3 DATA SET

To build this play list recommender, we used data provided by Million Song Data[5] Challenge hosted by Kaggle. This data was released by Columbia University Laboratory for the recognition and organization of speech and audio. Meta data of songs like audio content analysis is available for each song. However, we are exclusively using the listen count of a song by a user to build the recommender system. The Million Songs Data [5] is about 280GB and contains 48M of user-id, song-id, play count obtained through the listening histories of over one million users. Here, we only know about the listen count of a song by a user and since the users have not explicitly rated the songs, the play count is treated as implicit feedback. Since processing such large amount of data is highly memory and time intensive, we used a subset of above data containing data for 10,000 songs. It contains 2M triplets of 76,353 users and 10,000 songs. We also used meta data for 10,000 songs which consists of song-id, title, year, artist and release.

4 DATA PRE-PROCESSING

Since the user-id, song-id, play count triplets and song meta data is available in two different files, we merged the two files and created a master data structure which contains song meta data in addition to the 2M triplets. As listen count is an implicit feedback, it is not an absolute indicator of a userś preference. For example, if a user listens to song1 20 times more than song2, it cannot be assumed that user prefers song1 20 times to song2. However, what we can infer is that the user does like to hear song1 more than song2. Hence, to account for this, we propose to consider a weighted score 1 i.e.,.while building the recommender. In this way, we scaled down the listen counts by a log factor and used 80% of the data to train and the remaining 20% to evaluate the recommender.

$$w = 1 + log(listencount) \tag{1}$$

5 ARCHITECTURE

We implemented six models namely Popularity based recommender, SVD, SVD++, Collaborative filtering, Neural Collaborative Filtering and Bayesian Personalized ranking. We also developed an ensemble recommender 1 by finding the mean of the predictions of each model. While popularity-based recommender does not predict the listening score and it is not a personalized recommender, the other models perform well as a personalized recommender as they predict the listening score based on a user's play history. Each model is implemented as follows:

5.1 Popularity Based model

Play list recommenders often face with cold start problem for their new users. Since a user does not have a play count for songs, personalized recommendations can not be made to a new user. One possible approach to mitigate this problem is to recommend the most popular songs. In this model, we find out the popularity of each song by looking into the training data and calculating the total score of a song by summing up the weighted listening scores of

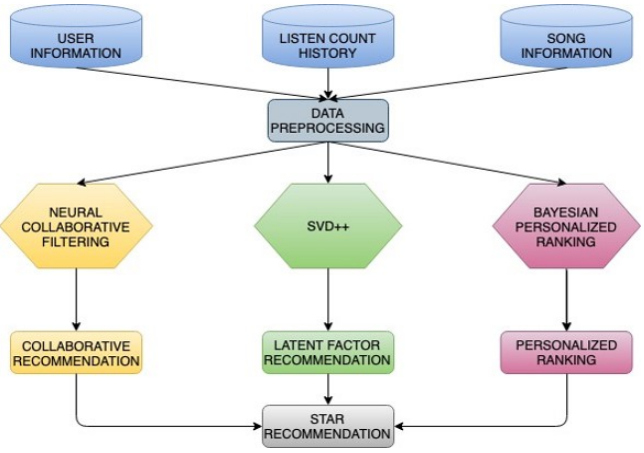

Figure 1: Architecture

that song across all the users. Songs are now sorted based on the popularity in descending order and top 10 songs are recommended for a new user's play list. The popularity of these songs can be changing over time and hence the most popular songs should be calculated from time to time. The disadvantage of this model is the absence of personalization.

5.2 SVD

Often, song preferences are influenced by factors specific to the domain like genre, artist, etc.,. These factors are however not obvious and it is hard to observe their correlations with the user preferences. Users and songs are characterized by these latent factors and they can be inferred from the data. Here, we build a matrix 2 which indicates the weighted listen score between a user and a song.

	Song1	Song2
User1		
User2		
User3		

Figure 2: Matrix M

SVD algorithm is implemented as follows: In the first step, we decompose matrix M (training data) into latent factor space that forms a relation between users and songs.

$$M = U \sum V, where \tag{2}$$

$$M \in R^{m*n}, U^{m*n}, \sum^{k*k}, V^{k*n} \tag{3}$$

Here, U corresponds to user factors and V represent item factors. Now, for each user, songs are recommended by ranking based on the predictions obtained from:

$$W_i = U_u^T . V_i \tag{4}$$

We used Surprise library [6] to perform the matrix factorization and predict the listening score for each pair of user and song available in the test data. Hence, in contrast to the popularity model, in this matrix factorization technique, we recommend the songs based

on the calculated listening score. For each user, we sort the songs in descending order of the predicted listening scores and top 10 songs in the list are augmented to the existing play list of the user. The predictions are evaluated using measures like RMSE, MAE, precision and NDCG as described in the evaluation section. Since, we are performing the recommendations based on the past play history, this model serves as a personalized recommendation system.

5.3 SVD++

The SVD++ is an extension of SVD and takes the implicit feedback into the account. The prediction is found by

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j \right) \tag{5}$$

Here, y_j terms are used to capture the implicit ratings. These terms just account if a user has listened to the song j but not the number of times the user has listened to the song. We used Surprise library[6] to calculate the predictions and here, the parameters are learned using SGD on the regularized square error objective. Once the ratings are calculated, we follow a similar process like SVD to recommend the songs. Songs in the test data are sorted in descending order for each user based on the predictions and top songs are added to the playlist of that user. Like SVD, SVD++ is also a personalized recommendation algorithm since the predictions are made based on the listening history of a user. Since the implicit feedback of whether user has listened to the song is also accounted in SVD++, this model performs better than a traditional SVD and the results also reflect the same.

5.4 Bayesian Personalized Ranking

BPR uses the user item pairs as the training data and ranks the item for a particular user in pairs as this is a better representation of the items which have not been visited by the user. Any item that has been rated by the user would be inferred as a preferred item over any non-rated item. Similarly, any two items which are both rated by the user, do not present any specific preference and same for any two non-rated items.

It presents a general optimization technique for personalized ranking using a Bayesian analysis of the problem. The users are assumed to be independent of each other and ranking of items among one user is independent from other users. The optimization criteria comes down to individual probabilities of users preferring an item i over item j. The algorithm introduces a general prior density which represents a normal distribution with zero mean and variance-covariance matrix. The maximization is done using gradient descent using bootstrap sampling of training triples. These triples are further decomposed into the difference of user item values. This is defined as below:

$$\hat{x}_{uij} = \hat{x}_{ui} - \hat{x}_{uj} \tag{6}$$

These user item values can now be predicted by matrix factorization. There are three regularization constants : user features, item features for positive and negative updates. We have the following posterior probability to maximize

$$p(\theta| >_u) \propto p(>_u |\theta) p(\theta) \tag{7}$$

θ represents a vector of an arbitrary model class (ex matrix factorization) [12]. This function can be further combined for all the users ranking for each user is independent from each other. This can be further broken down to the probability that a user prefers an item i over j as:

$$p(i >_u j|\theta) := \sigma(\hat{x}_{uij}(\theta)) \tag{8}$$

where σ is the logistic sigmoid [12]:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{9}$$

The prediction of user item values $\hat{x}_{ui}$ consists of finding the product of lower matrices called here[12] as W and H. This gives us the prediction formula for the user item values.

$$\hat{x}_{ui} = \langle w_u, h_i \rangle = \sum_{f=1}^{k} w_{uf} . h_{if} \tag{10}$$

where w and h are the row vectors in the lower matrices W and H.

The implementation used in this project is based on one of the popular github[9] implementation for BPR using Theano library. It is based on matrix factorization model where we have used 10 latent factors for this dataset. We first parse the data into raw values of user item interactions for both the training and testing dataset. The indexes for users and items are then mapped to indexes in training and testing sets. We then use the predicted songs and there scores for the evaluation. We also provide the AUC value which represents the quality of the predictions instead of their absolute values.

5.5 Collaborative Filtering

Collaborative filtering is a method of generating automatic recommendations/predictions about the interests of a user. This is done by collecting the preferences from many different users. It is equivalent to K-nearest neighbor approach. We implemented user- user CF. First, we calculate the similarities between the target users and all other users in our dataset. We calculate the expected rating of an item by using the user-to-user similarity and aggregate the scores by considering other users who have also consumed the same item. We do this for all the items of concern and then return the top items for recommendation generation. For calculating the similarities, we use Pearson correlation which takes care of the user to user dissimilarities by using average ratings of each users.

$$sim(x, y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x}) (r_{ys} - \overline{r_y})}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})^2} \sqrt{\sum_{s \in S_{xy}} (r_{ys} - \overline{r_y})^2}} \tag{11}$$

The above formula is used to calculate pearson similarity values among users. S_{xy} is the set of items consumed by user x and y. $\overline{r_x}$ $\overline{r_y}$ represents the average listen count by user x and y in logarithmic scale. r_{xs} and r_{ys} are the listen count for a particular song 's' in set S_{xy} . We calculate the similarities among all the pair of users and then aggregate the scores to calculate the expected rating which in

our case happen to represent expected listen count for a song. We aggregate the score using the following formula:

$$r_{c,s} = k \sum_{c' \in \hat{C}} sim\,(c, c') \times r_{c',s} \tag{12}$$

Some of the highlights of collaborative filtering approach are:

(1) it does not rely on feature selection about the item or user. It considers the available data to calculate the similarities which is used for predictions. So, it can be applied to a wide class of problems.
(2) CF can have a cold start when there arenŕ enough users in the system with a match.
(3) Recommending a new item is difficult, since there is no other user who would have rated it.
(4) CF is inclined towards popularity and does not work good for users with unique tastes.

Since collaborative filtering on 76,365 users consumes memory and computational power out of the scope, we implemented this model on a subset of users i.e., 5,000 users. Hence, these predictions were not included in the ensemble model. We observed that the precision of the CF model increases with an increase in the number of top recommendations made by the model.

5.6 Neural Collaborative Filtering

Neural network architecture is proven to be effective in several domains. Here, we build a neural network to tackle the problem of recommendation based on implicit feedback. The model explores the use of deep neural networks for learning the interaction function from data. We use listen count of a song by a user as an implicit attribute. Compared to explicit feedback (i.e., ratings and reviews), implicit feedback can be tracked automatically and is thus much easier to collect for content providers. However, it is more challenging to utilize, since user satisfaction is not observed and there is a natural scarcity of negative feedback.

The architecture comprises of a multi layer representation of user-item interaction. As it is a collaberative filtering approach, we use only the identity of a user and an item as the input feature, transforming it to a binarized sparse vector with one-hot encoding. The obtained user (item) embedding can be seen as the latent vector for user (item) in the context of latent factor model. The multi layer neural model takes the user embeddings and item embeddings as inputs and maps the latent vectors to prediction scores. The dimension of the last hidden layer X determines the modelĂŹs capability. The activation cell value of the final layer is the predicted score and minimize the point wise loss between predicted score and actual score.

We transform the input to one hot encoded vectors which are then mapped to a hidden space to generate embeddings. Now we train the GMF and MLP model, which is ensembled using a neural network to generate final predictions.In NCF, we allow the matrix and latent factors to be learnt from the data without uniform constraints. This creates another variant of matrix factorization with changing importance of features in the latent space. We use a non-linear sigmoid activation function for aout which generalizes better than a linear model. This approach is know as Generalized Matrix Factorization (GMF). Parallelly, these input embeddings are also fed into a multi-layer perceptron network to identify latent vectors and hence generate prediction scores. In each layer the model learns something new about the user item interaction. Now we have two well performing models, the GMF extracts the linear features and relationship where as MLP uses non-linear kernel to understand the data. We combine both these approach to make prediction of listen counts. We then use another neural ensemble approach to learn the weights for combining the outputs coming from GMF and MLP.

Layer (type)	Output Shape	Param #	Connected to
Song-Embedding-MLP (Embedding)	(None, 1, 10)	100010	Item[0][0]
User-Embedding-MLP (Embedding)	(None, 1, 8)	610832	User[0][0]
FullyConnected (Dense)	(None, 200)	3800	dropout_14[0][0]
Batch (BatchNormalization)	(None, 200)	800	FullyConnected[0][0]
Dropout-1 (Dropout)	(None, 200)	0	Batch[0][0]
FullyConnected-1 (Dense)	(None, 100)	20100	Dropout-1[0][0]
Batch-2 (BatchNormalization)	(None, 100)	400	FullyConnected-1[0][0]
Music-Embedding-MF (Embedding)	(None, 1, 3)	30003	Item[0][0]
User-Embedding-MF (Embedding)	(None, 1, 3)	229062	User[0][0]
FullyConnected-2 (Dense)	(None, 50)	5050	Dropout-2[0][0]
dropout_11 (Dropout)	(None, 3)	0	FlattenSongs-MF[0][0]
dropout_13 (Dropout)	(None, 3)	0	FlattenUsers-MF[0][0]
FullyConnected-3 (Dense)	(None, 20)	1020	FullyConnected-2[0][0]
Dot (Concatenate)	(None, 6)	0	dropout_11[0][0] dropout_13[0][0]
Activation (Dense)	(None, 1)	21	FullyConnected-3[0][0]
Concat-MF-MLP (Concatenate)	(None, 7)	0	Dot[0][0] Activation[0][0]
Combine-MF-MLP (Dense)	(None, 100)	800	Concat-MF-MLP[0][0]
FullyConnected-4 (Dense)	(None, 100)	10100	Combine-MF-MLP[0][0]
Prediction (Dense)	(None, 1)	101	FullyConnected-4[0][0]

Total params: 1,012,099
Trainable params: 1,011,499
Non-trainable params: 600

Figure 3: NCF architecture summary

3 shows the details of the architecture, where we have learned a total of 1,012,099 parameters. The user embeddings is a vector of size 1x8 and song embedding is a vector of size 1x10.

6 EVALUATION AND RESULTS

We used RMSE, MAE as parameters to evaluate our prediction scores as shown in 4. The best MAE values are obtained for neural collaborative filtering 1. Since the listen scores are not predicted for Popularity model, RMSE and MAE values were not reported. Apart from this, we calculated Precision[2] and NDCG[1] to evaluate our results. The results obtained for each model are tabulated in 1.

$$Precision = \frac{true\_positive}{(true\_positive + false\_positive)} \tag{13}$$

$$nDCG_p = \frac{DCG_p}{IDCG_p} \tag{14}$$

where

$$DCG_p = \sum_{i=1}^{p} \frac{rel_i}{log_2(i+1)} \tag{15}$$

$$IDCG_p = \sum_{i=1}^{|REL|} \frac{rel_i}{log_2(i+1)} \tag{16}$$

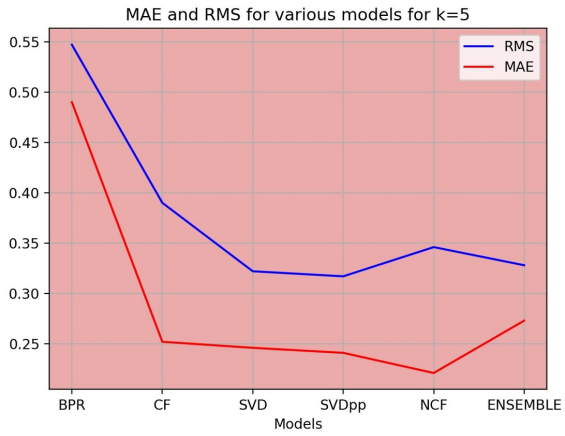

Figure 4: MAE and RMSE for k=5

To calculate precision@K, we predicted the topK songs for each user and compared our results with the actual topK songs from the test data. To take ranking of the songs into account, we used NDCG [1] to better evaluate the recommender. NDCG[1] penalizes the errors on the top of the predictions. In most models, we observed that precision and NDCG[1] values increase as K value increases 6. This is due to the size of the test data. Since our model recommends the songs within those listened by the user, as the K value increases, the songs predicted often match with the actual preference of a user.

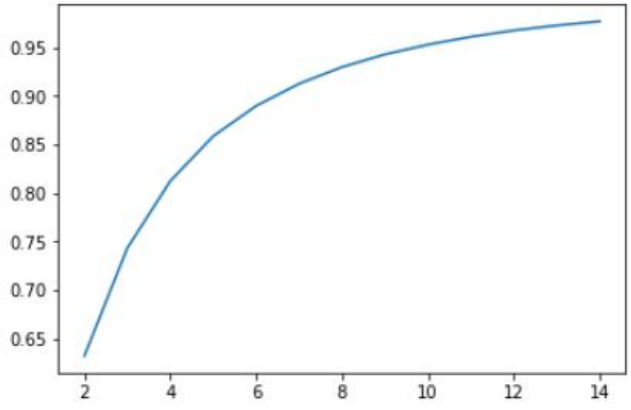

Figure 5: Precision vs K

The very low values of precision and NDCG for Popularity based model 1 is due to the lack of personalization. Since popularity model recommends the most listened songs in the test set, these songs often do not match with the songs listened by a user. We also

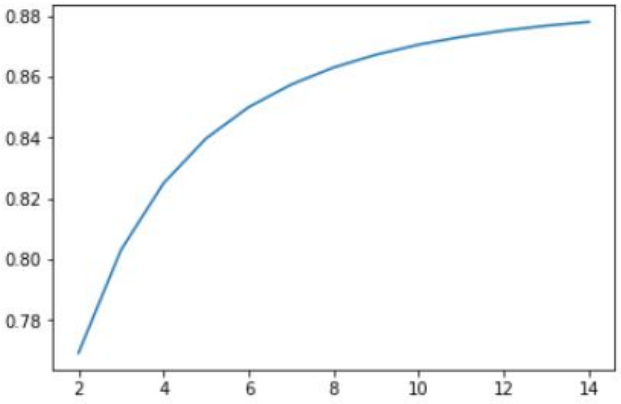

Figure 6: NDCG vs K

observe that SVD++ performs well than SVD 1. This is because accounts whether listened a song as implicit feedback and accounts that in the model.

BPR achieves a mean AUC of 0.86 and a precision value of 85% for top 5 results. The RMS and MAE errors of the BPR approach are on the higher side than SVD and NCF considering the different normalization technique that we used in this project. The scores of user item pairs are scaled in the range of 1-2 as we take a logarithmic scale of those scores. Initial values obtained for user item ratings were in a much larger range which was not comparable to other models directly. Although this does not changes our precision values for the model as we it consistently performs at par with different epoch values, the absolute error values are somewhat deviated. As consistent with other models we also plot a curve for precision at different k. It is evident from the graph that the model achieves very high precision for top 15 songs suggested to a particular user.

A low precision at small k value should not be considered as the only metric as the actual behavior of the user is more diverse. A user is expected to listen to about 10-15 songs on average and not just 2-4 songs. This opposes the fact that search results generally try to achieve high precision at small k values (like Google search) where top 1 or 2 results are very important as the user is very likely to click them.

BPR presents the Bayesian analysis of the optimization problem. The reason we see higher error values here is due to the different normalization of ratings used in the model. Predicted values are mapped in the range 1-2 and thus some users who have high listen count (>2) will contribute to higher RMS and MAE errors. The underlying methods used for matrix factorization used in SVD and BPR are very similar and therefore the precision values are very close. The AUC score achieved is 0.86486 at 10 latent factors which is at par with the Netflix dataset used in the original paper [12].

For the ensemble approach, we combine the predicted scores from 4 different approaches i.e., SVD, SVD++, neural collaborative filtering and Bayesian personalized ranking to give an ensemble

Model	RMSE	MAE	Precision@5	NDCG@5
Popularity based model	N/A	N/A	0.023	0.0339
SVD	0.3225	0.2468	0.8551	0.8384
SVD++	0.3172	0.2418	0.8575	0.8395
Collaborative Filtering	0.3388	0.2495	0.8468	0.8311
Bayesian Personalized Ranking	0.5473	0.4925	0.8501	0.8357
Neural Collaborative Filtering	0.3461	0.2218	0.8568	0.8392
Ensemble(SVD , SVD++, NCF, BPR)	0.3285	0.2731	0.8585	0.8397

Table 1: Results

approach for predicting the user item scores. Collaborative filtering was not included since this model was implemented only on 5,000 users because of the constraints on computing power and memory. Popularty based model was not embedded because here, we do not predict the listen score for a song. Instead, we just recommend songs based on their popularity. The ensemble approach gives a high precision of 85% at k value of 5. As outlined in the previous approaches we get a high precision value for top 10 results at about 95%. The overall RMS error for all the ensemble model is at 0.32 and mean absolute error at 0.27

7 CONCLUSION

We developed an ensemble recommender system to add songs to a userś play list. The base models used were SVD, SVD++, collaborative filtering, neural collaborative filtering and Bayesian personalized ranking.Using an ensemble approach helps to incorporate the benifits of each model and overcome the demerits of individual approaches. We evaluated the individual performance of each system and compared the results with one-another besides to the ensemble approach. We observed that latest approaches like neural collaborative filtering produced better results than traditional methodologies.

As future work, one can improve this approach by including content based recommendations using features like genre, artist, song hotness etc. However, one may find it challenging to gather the data for each song and implement the system given the volume of the Dataset. Besides, an ensemble recommender which learns weights of individual systems using existing machine learning methodologies can be developed to improve the over-all performance of the recommender. The recommender can also be implemented on entire data-set if one can use distributed systems to run parallel computations thereby reducing the run time.

REFERENCES

[1] 2018. Discounted cumulative gain. https://en.wikipedia.org/wiki/Discounted_cumulative_gain

[2] 2018. Precision and Recall. https://en.wikipedia.org/wiki/Precision_and_recall

[3] J. Konstan B. Sarwar, G. Karypis and J. Riedl. 2001. Item-based collaborative filtering recommendation algorithms. (2001).

[4] J. Konstan B. Sarwar, G. Karypis and J. Riedl. 2002. Incremental singular value decomposition algorithms for highly scalable recommender systems. *In Proceedings of the 5th International Conference in Computers and Information Technology* (2002).

[5] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. 2011. The Million Song Dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011).*

[6] Nicolas Hug. 2017. Surprise, a Python library for recommender systems. http://surpriselib.com.

[7] B. Kanagal I. Bayer, X. He and S. Rendle. 2017. A generic coordinate descent framework for learning from implicit feedback. (2017). https://arxiv.org/abs/1611.04666

[8] A. Mnih R. Salakhutdinov and G. Hinton. 2007. Restricted boltzmann machines for collaborative filtering. (2007).

[9] Yves Raimond. 2014. theano-bpr. https://github.com/bbc/theano-bpr.

[10] J. Kawale S. Li and Y. Fu. 2015. Deep collaborative filtering via marginalized denoising auto-encoder. (2015).

[11] S. Sanner S. Sedhain, A. K. Menon and L. Xie. 2015. Autorec: Autoencoders meet collaborative filtering. (2015).

[12] Zeno Gantner Steffen Rendle, Christoph Freudenthaler and Lars Schmidt-Thieme. [n. d.]. BPR: Bayesian Personalized Ranking from Implicit Feedback. ([n. d.]). https://arxiv.org/pdf/1205.2618.pdf

[13] F. Strub and J. Mary. 2015. Collaborative filtering with stacked denoising autoencoders and sparse inputs. (2015).

[14] M.-Y. Kan X. He, T. Chen and X. Chen. 2015. TriRank: Review-aware explainable recommendation by modeling aspects. (2015).