Last.fm New Artist Recommendation System (HW #2)

E 4571: Personalization Theory and Application

Tim Kartawijaya (tak2151), Charlene Luo (cl3788), Fernando Troeman (ft2515)

1. Objective

The goal of this project is to build a system that recommends top-k (i.e. top-20) artists for Last.fm users. These recommendations are personalized by each user's listening history and are new to each user. The business objective of this system is to increase user activity in Last.fm and user loyalty to the Last.fm brand by providing a service that allows users to discover new artists and have a more enjoyable listening experience.

2. Data

The dataset used in this project consists of 360,000 Last.fm users with their implicit feedback of their top 50 most played artists. Each observation has the user ID, artist ID, and number of times that user listened to that artist. The data was collected from the Last.fm API by Oscar Celma, who gathered the data in the Fall of 2018 (https://goo.gl/gxPTuK). For this homework, we worked with a subset of the dataset, containing a total of 9,000 users and 47,102 artists. In our initial analysis this dataset is about 99.09% sparse.

3. Model

In this project, we focused on using collaborative filtering models, specifically Alternating Least Squares Matrix Factorization (model-based) and K-Nearest-Neighbors (item-based). We chose to use Ben Frederickson's implementation of these models, using the *implicit* package (https://goo.gl/Xaf5Eb). This package is optimized for datasets with implicit feedback and runs significantly faster than equivalent implementations on Spark. Below are the different models that we implemented:

a. Most Popular Artists (Baseline)

In order to contextualize the results from Alternating Least Squares and K-Nearest Neighbors, we provided a baseline model in which every user is recommended the 20 most popular artists across all users. In this case, every user receives the same recommendation.

b. KNN

We opted to implement an item-based K-Nearest Neighbors system because these models are easier to scale than user-based models. Furthermore, item-based recommender systems provide more stability as items are added less frequently than users. The implicit package allows for item-based collaborative filtering on implicit data using 3 different distance metrics – Cosine, TF-IDF and BM25 (Best Matching 25). We have settled on BM25 as our go-to distance metric because it significantly outperformed Cosine and TD-IDF on our sample dataset.[1]

c. Model-Based (ALS)

First, we implemented the Alternating Least Squares (ALS) algorithm from the implicit package. Matrix factorization using ALS is a common collaborative filtering method. This algorithm allows for some customization through the number of latent factors and the regularization constant, which we will discuss in later sections.

4. Evaluation

To evaluate our models, we chose to use the holdout validation method. This method is appropriate for collaborative filtering settings, since during train-test splits, the training data will still contain user-specific data which are needed for proper training.

a. Train Test Split

First, we split our dataset into a training set and a test set on a per-user basis. For each user, we took the top-k artists that users have most listened to, then randomly sample a percentage of artists from said top-k set to create our test set for that user, i.e. the holdout set. Then, we create the training set by updating the values of those sampled user-pair artists to zero, effectively masking and "holding-out" data from training. To evaluate these models, we can then compare the holdout set with recommended items, since we would expect the holdout items to be included in the set of recommendations.

b. Metrics

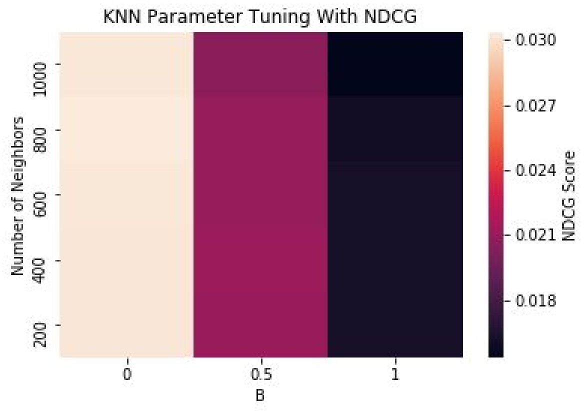
We chose to use recall and NDCG as our model accuracy metrics. We determine recall by calculating the ratio between the number of times a holdout item appears in the recommendations with the total number of holdout items (True Positives / Holdout Set Items). For NDCG, we calculate NDCG for each user, where values in the ranking are zero if the item is not in the holdout set. If it is a holdout item, then the number would be the number of plays as a proxy of the relevance score of that item. We then take the average across all users for reporting. We opted not to use error metrics like RMSE and MAE and precision-based metrics, since these metrics are not as insightful in top-k settings.

Furthermore, we also used catalog coverage as a metric of the diversity of our recommendation. A catalog coverage of 20% means that out of the 47,102 artists in our dataset, only 20% of them appear in at least one user's top-20 recommendations.

5. Parameter Tuning

To optimize the accuracy of these models, we tuned the parameters of each model to maximize NDCG. To do this, we held off a test set from the data and split the rest into k-fold training and tuning sets. This way, we were able to fit a range of parameters on each training set and select the parameters that produced the best NDCG on the respective tuning set. With k-folds, we were able to perform this k times.

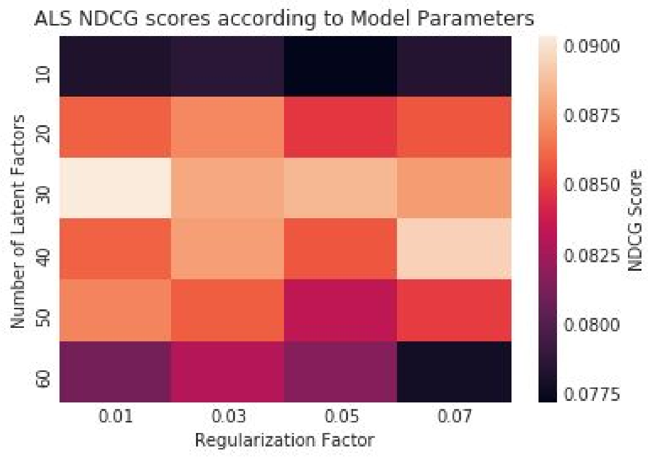
For KNN, the BM25 recommender gives us 3 parameters to work with: K, K1 and B. K refers to the number of neighbors computed by the model for each artist. K1 is a weighting function that decides whether our distance metric used is closer to that of Jaccard or Cosine similarity. B serves as a sort of bias towards or away from artists based on their popularity. A more detailed explanation of these parameters can be found in the Appendix.



The results of our hyper-parameter tuning is shown in the heatmap above. Setting K1 to the default value of 1.2 works sufficiently well for our recommender system, with any variation providing only negligible changes to our evaluation metrics. The first thing we noticed is that NDCG score is most heavily affected by B. NDCG is maximized when **B is set to 0**, which meant a heavy bias towards more popular artists - this makes sense in our case given the sparsity of our data. Thus, attaching a higher weight to popular artists makes our results more relevant. We have also found that the ideal value of **K is 400**.

For ALS Matrix Factorization, there were two factors to tune: the number of latent factors and the regularization factor. The number of latent factors determine the number of features/attributes that describe a user or an artist upon factorization. The regularization factor is a constant in the minimization function of this algorithm that makes sure there is no overfitting in the model.

As shown from the NDCG heatmap below, the optimal parameters are **30** for latent factors and **0.01** for the regularization. To little (~10) or too many (~60) latent factors determined whether the model was keeping not enough or too much information to predict the test set. The regularization factor seems to follow a less straightforward pattern, but 0.01 produced the optimal NDCG score when combined with 30 latent factors.



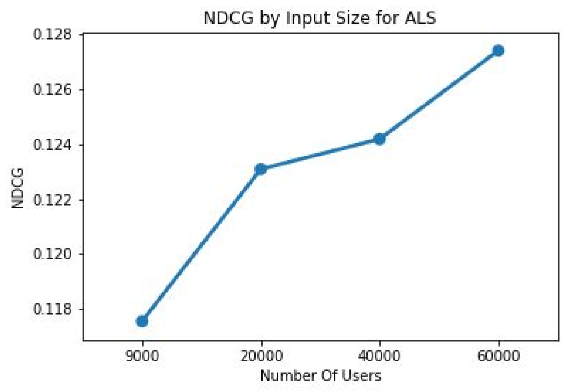
6. Results

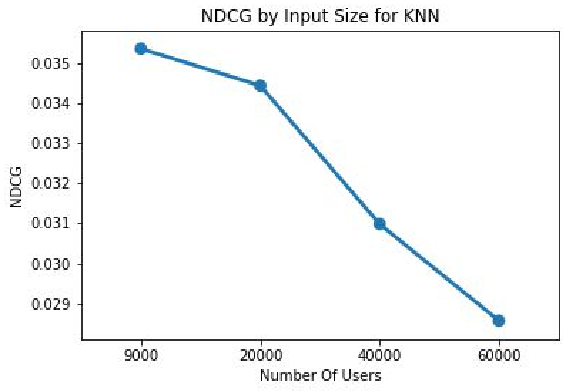
After training and testing the models, we come to the conclusion that the ALS method is superior in terms of recall in recommending new artists to users. As we can see in the table below, ALS has a higher recall and NDCG than both KNN and our baseline model. However, ALS has a lower catalog coverage than KNN (11.1% vs 16.9%). Given that our objective is to expand the scope of our users' music preferences, we should seek to improve the coverage of our ALS model. Looking at KNN, the model has a worse recall even compared to the baseline, at around 5%. We suspect KNN is performing poorly due to the high sparsity rate of our dataset (99.09%), thus the calculated distances between neighbors is not as accurate. For now, we will state ALS as the better model, since it does not sacrifice recall for coverage as much as KNN.

Model	**Recall**	**NDCG**	**Coverage**
Baseline	6.91%	4.74%	0.04%
KNN	5.06%	3.54%	**16.9%**
ALS	**18.0%**	**11.6%**	11.1%

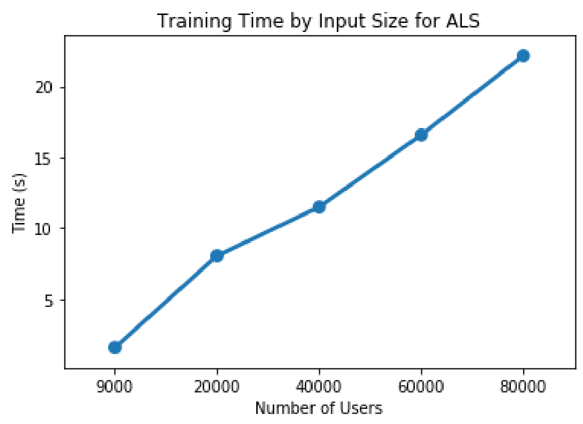
7. Scalability

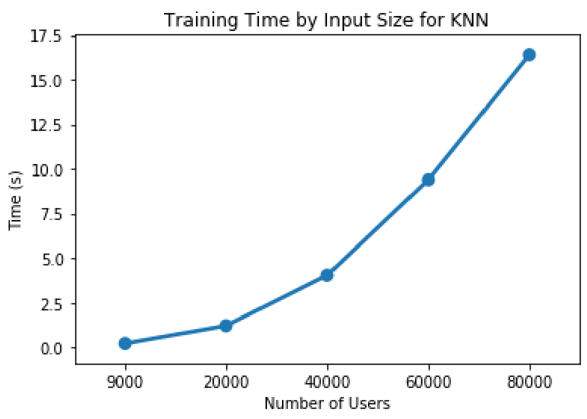
In addition to recall, we also explored how scalable these custom models are by increasing the input size i.e. the number of users included in the dataset. For reference, 100,000 users amount to about 5 million ratings. Note that we ran the models below on the Google Cloud Platform to run the code more efficiently, using an Ubuntu 18.04 LTS Virtual Machine with 8 vCPU and 52 GB of memory.





In regards to NDCG score, we see an opposing trend from the graph above between the NDCG of the ALS model and the KNN model. For ALS, NDCG improves with an increasing number of users. Conversely, the KNN model becomes worse as the number of users increase.





In regards to time complexity, the graphs above shows us that with increasing input size, training time for ALS grows linearly while for KNN it grows exponentially. This result would tell us that KNN is not as scalable in training compared to ALS. Either way, *implicit*'s implementation is very efficient, where even for 80,000 user data, the training time is under a minute.

8. Conclusion and Next Steps

In conclusion, the model-based collaborative filtering method is the best model in performing top-k artist recommendations for users. The ALS model outperformed KNN and the baseline model in regards to recall and has a decent catalog coverage rate. Furthermore, the ALS model is productionalizable, since it is scalable to input size in regards to its recall and training time.

Moving forward, we can further improve recall and coverage by exploring more advanced algorithms like Factorization Machines, which are shown to be highly effective on datasets with implicit feedback. Furthermore, we can explore how incorporating artist metadata (genre, birth date) and user demographics (gender, age) in our models can improve recall and coverage.

Appendix

[1] Distance Metric and Parameters

BM25, like TF-IDF, is an information retrieval model that was initially developed for search engines and has proven to be extremely effective in calculating similarity between text documents. Conceptually, these IR methods can be adapted for non-text purposes. In our case, we will treat each artist as an individual text document and each user as a specific term in all documents.

The idea behind BM25 is to utilize a weighting function that provides a sort of geometric middle ground between Jaccard and Cosine distances. It is often more accurate than TF-IDF, which uses a fixed function (typically logarithmic, sometimes square roots) for term frequency (TF). In the implicit package, this weighting function is tuned using a parameter K1, where Jaccard distance is essentially computed when K1 = 0 and Cosine distance is used when K1 = positive infinity.

BM25 can also be tuned by a parameter B, which decides how length normalization is handled in its computation. Play counts are scaled by the ratio of an artist’s popularity to the average popularity of all artists. The parameter B allows us to control how much influence length normalization has on our recommendations. A lower B would be biased towards more popular artists, which makes sense in our case given the sparsity of our data.