Recommendation System for Implicit Feedback Dataset

1003 Course Project

Yueqiu Sun(ys3202), Nan Su(ns3783), Hanlu Zhang(hz1625)

May 2018

1 Introduction

1.1 Objective

Recommendation systems have become increasingly popular in recent years and can be used in various areas such as movies, news, music, products and research articles. These systems can provide users with personalized recommendations for products or services, which hopefully suit their unique taste and needs. Content-based filtering and collaborative filtering are classic approaches to produce recommendations, where content based filtering creates a profile for each user or product to characterize its nature, while collaborative filtering, which analyzes the relationships between users and interdependencies among products to identify new user-item associations, does not require any external information to be collected [1].

Our objective is to build a model based on implicit feedback from a user's past behaviour as well as similar decisions made by other users to predict songs that the user may have interest in, which is a collaborative filtering technique. To be more specific, we want to predict the preference of each user to each song, and thus to make recommendations to each user with a ranked list of songs.

1.2 Data set

We use the Million Song Dataset [2], which contains users, songs, and play counts. There are 1019318 unique users, 384546 unique MSD songs, and 48373586 user-song play count triplets, where users and songs are represented by ID sequences.

2 Baseline Model

We use the popularity ranking model [1] as our baseline model. In general, this is a simple approach which recommends items liked by most number of users [3]. For our implicit feedback scenario, this model is easy to implement, as it generates the recommendation list by only computing the number of total user who has listened to each song in the dataset. The model will recommend the top K songs in the ranking list for each user. Since popularity is defined on the entire user pool, the most popular items would be the same for each user and thus brings the major drawback to our attention of this approach, which is that there is no personalization involved in it.

3 Neighborhood Model

3.1 Methodology

Neighborhood methods predict users' preference based on other like-minded users. In Neighborhood methods, we calculate weights for each pair of the users using similairty metrics, which indicates the influence on the user' preference we are trying to predict. The weights can be computed using the similarity metric between users. The more similar the users are, the more weight we assign to the user, the more likely they share the same preference.

The similarity metrics we use is vector similarity. Let s_{ij} denote the similarity between user i and user j. The expression for vector similarity is as follows,

$$s_{iw} = \frac{\sum_{u \in U_i \cap U_j} r_{iu} * r_{wu}}{\sqrt{\bar{v_i} * \bar{v_w}}}$$

in which r_{ik} denote the normalized implicit votes i has on song k, U_i be the subset of the songs user i ha listened to, and

$$\bar{v_i} = \sum_{u \in U_i} r_{iu}$$

It is worth noting that if user i and user j have no song identical, then the similarity between user i and user j is 0. The squared term $\bar{v_i} * \bar{v_j}$ in the denominator serve to normalize votes so that users that have listened to a lot of songs will not *a priori* be more similar to other users.

The similarity metrics between users can be used as weights [4] to compute the predicted preference to each song. Denote user i 's predicted preference torwards song j as p_{ij}. The predicted preference is the weighted sum of other user's implicit votes on song j:

$$p_{ij} = \frac{1}{|W_i|} \sum_{w \in W_i} s_{iw} r_{wj}$$

in which W_i is the set of all the users except user i.

We can compute similarity matrix S between each pair of user:

$$S = RR^T(\bar{V}\bar{V}^T)^{-\frac{1}{2}}$$

in which R is the implicit votes matrix and V is a array of the total votes of each user. Assume there are n users, then the similarity matrix should be size n x n.

We can compute the preference of each user for every song:

$$P = SR$$

Assume there are n users and m songs, then the similarity matrix should be size n x m, same as the R matrix.

For each user, we recommend k songs with the highest preference to the user in which k is arbitrary.

3.2 Extension to Neighborhood Model

Inverse User Frequency

The use of Inverse User Frequency is inspired by the use of TFIDF in the text processing model. The idea is to reduce the weight of commonly loved item to better capture the similarity between users. Universally loved item is not as effective as its counterparts with regard to reflect the pattern of user's preference.

To implement Inverse User Frequency in Neighborhood Model, we define f_j as as $\frac{n}{n_j}$, where n_j is the number of unique user has listened to song j and n is the total number of users. We simply multiply f_i to the original implicit vote for song j to obtain the Inverse User Frequency.

Case Amplification

Case Amplification refers to applying power transformation (power larger than 1) to the weights to augment large weights and penalize small weights.

$$w_{ij} = \begin{cases} w_{ij}^{\rho} & w_{ij} \geq 0 \\ -|w_{ij}^{\rho}| & w_{ij} < 0 \end{cases}$$

3.3 Neighborhood Model time Complexity analysis

There is no training time for Neighborhood Model as it don't need to be trained. Assume there is n users and m different songs. To compute the recommendation list for a single user, the algorithm takes $O(nm + m \log m)$ time in which nm is the time to compute similarity and $m \log m$ is the time to sort the scores for each song. To compute the recommendation list for n users, the algorithm takes $O(n^2 m + nm \log m)$ time. Since we can convert the data to sparse matrix, the above expression can be greatly reduced. However, the upper bonnd for the algorithm remains the same.

4 Weighted Matrix Factorization

The article [1] presents a latent factor method for collaborative filtering of data sets where the only information available on users's preferences are implicit. The method proposed in this article attempts to assign a confidence rating to the observation of user preference of each user-item pair. A user not consuming an item is considered as negative feedback, but with very low confidence.

4.1 Latent Factor Models

Latent factor models attempt to project both users and items into a factor space, where factors represent some underlying property of the data. For example, music might be projected onto loudness, rhythm and styles. In most latent factor methods, factors are generated algorithmically from the data and do not have ready interpretations.

In recommender systems, the map to factor representation takes the form of a transformation matrix, projecting items into factor space. A proper choice of factors is supposed to result in the most significant and meaningful factors which can explain the majority of the features in the data set. These factors can then be used to give a lower-dimensional approximation of the data. This is very useful when dealing with large data sets such as those arising in recommendation, since processing in lower-dimensional factor space will be much quicker.

Recommendation in such latent factor models is completed by taking an inner product between users and items in this reduced dimensionality factor space.

4.2 Methodology

In the article, the latent factors are determined by minimizing the squared error between the actual user preference p_{ui} for each user u - item i pair and that generated from the inner product of the user and item in factor space, x_u and y_i, respectively. The objective function is weighted for observation confidence for each pair c_{ui} and regularized to avoid over-fitting. The resulting objective function is shown below:

$$\min_{x_*, y_*} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u ||x_u||^2 + \sum_i ||y_i||^2 \right)$$

Two confidence functions are proposed in the paper:

$$c_{ui} = 1 + \alpha r_{ui}$$

and

$$c_{ui} = 1 + \alpha \log(1 + r_{ui}/\epsilon),$$

where r_{ui} is the number of times user u consumed item i and α is a constant that controls the rate of increase in confidence.

User preference p_{ui} are generated by simply noting whether a user has consumed an item:

$$p_{ui} = \mathbf{1}_{r_{ui}>0}.$$

We use Alternating Least Squares(ALS) algorithm [5] to minimize the objective function. This algorithm first optimizes the user vectors x_u by holding the item vectors y_i constant and taking the derivative of the loss function with respect to the user vectors. Then we can find an analytic expression for x_u:

$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$$

where C^u is a diagonal matrix where $C^u_{ii} = c_{ui}$ and $p(u)$ contains all the preferences by u(the p_{ui} values). A significant speedup is achieved by using the fact that $Y^T C^u Y = Y^T Y + Y^T (C^u - I) Y$ since $Y^T Y$ does not depend on u and was precomputed, $C^u - I$ has the number of non-zero elements which equals to the number of items for which $r_{ui} > 0$.

After computing the user vectors, the item vectors are recomputed:

$$y_i = (X^T C^i X + \lambda I)^{-1} X^T C^i p(i)$$

where C^i is a diagonal matrix where $C^i_{uu} = c_{ui}$ and $p(i)$ contains all the preferences for i(the p_{ui} values). This step is done by using the same technique as with the user-factors. We employ a few sweeps of paired re-computation of user and item vectors till they stabilize.

Finally, after we get the user and item factors, we can recommend to user u the K available items with the largest value of $\hat{p}_{ui}$ where $\hat{p}_{ui}$ symbolizes the predicted preference of user u for item i.

5 Evaluation and Results

Evaluation Methodology We use the evaluation method proposed in the article [1]. The method evaluates the case where a ranked list is generated to each user. The list is sorted from the most preferred song to the least preferred song. Since we do not have an explicit feedback on whether a user likes a song

or not, precision based evaluation metrics are not very appropriate. A recall based measure seems more appropriate here, as a higher count of listening the same song indicates liking it.

The expected percentile ranking of songs in the test set is used to evaluate the model performance, which is given by:

$$\overline{rank} = \frac{\sum_{u,i} r^t_{ui} rank_{ui}}{\sum_{u,i} r^t_{ui}}$$

where $rank_{ui}$ is the percentile-ranking of song i for user u in the ranked list and r^t_{ui} is the unobserved value by user u for song i in the test set. In detail, $rank_{ui} = 0\%$ indicates that song i is predicted to be the most desirable one for user u, and thus is placed at the top of the ranked list. On the other hand, $rank_{ui} = 100\%$ indicates that song i is predicted to be the least desirable one for user u, and thus is placed at the end of the ranked list. A lower value of $\overline{rank}$ represents a better model performance, as it indicates that ranking actually listened songs closer to the top of the recommendation list.

For random predictions, which just placing item i in the middle of the ranked list, the expected value of $rank_{ui}$ is 50%. As a result, an algorithm which has $\overline{rank} \geq 50\%$ is no better than random.

Evaluation Results We sampled a small dataset from the whole dataset and split it into train and test set to evaluate our model performance. The popularity ranking model, which we use as our baseline model, is powerful as it can achieve $\overline{rank} = 22.67\%$. However, this model is not personalized as we have discussed before.

Neighborhood Model outperforms baseline model by a very large margin. After optimizing over hyper-parameter including if_ivf(whether to apply ivf transformation to the data) and ρ(case amplication factor). Neighborhood Model achieved $\overline{rank} = 12.06\%$.

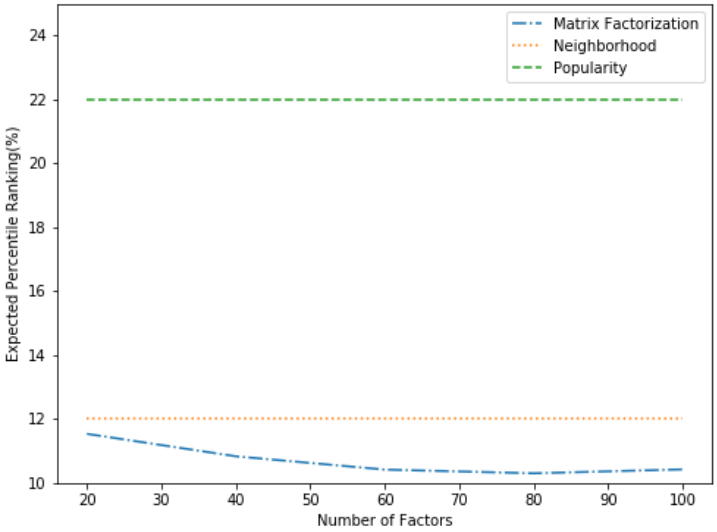

Figure 1: Comparing factor model with popularity ranking and neighborhood model.

Even better results are achieved by our factor model. We implemented the weighted matrix factorization model with different number of factors, ranging from 20 to 100. In addition, we tried different set of α, number of iterations and regularizations λ. In general, α around 1, iteration times around 10 and regularization around 0.01 give relatively better results. Results keep improving as number of factors increases till 60, after which the expected percentile ranking reaches 10.29% and then changes very slightly. Thus, we recommend working with the 60 factors, which gives a acceptable score and stays feasible within computational limitations.

Recommendation System Showcase

We present three examples to show the results of our recommendation engine. Please note that the user in the table is chosen randomly. We recommend two songs for each user using Neighborhood Models. The table shows both the listening history of the user and the according recommendation.

Table 1: Recommendation System Showcase

user_index	listening history	Recommendation
854	The Wild Boys(Duran Duran), Things I Don't Understand(Coldplay), Secret Hell(dEUS)	Jerry Was A Race Car Driver(Primus), More Than Everything(Gareth Emery)
1722	Ego(Beyoncé), Diamonds From Sierra Leone(Kenya West), I'm Done(The Pussycat Dolls)	Welcome To Hollywood(Beyoncé), Can't Help But Wait(Trey Songz)
2937	Sincerité Et Jalousie(Alliance Ethnik), Clint Eastwood(Gorillaz), Ragoo(Kings Of Leon)	Still Don't Give A Fuck(Eminem), Rebirth of Slick (Cool Like Dat)(Digable Planets)

6 Improvements

Weighted Matrix Factorization

The need to tune several parameters(λ, α) and several settings of the model(number of iterations, number of factors) makes this method very time-consuming. The article mentioned another confidence function

$$c_{ui} = 1 + \alpha \log(1 + r_{ui}/\epsilon),$$

which we could not try as it includes another parameter ϵ. Tuning all of these parameters and settings simultaneously is not really possible so it is difficult to find the best model.

Solution:

- The ALS optimization method might be replaced of Stochastic Gradient Descent(SGD) to speed up our calculation.
- Multi-threaded training routines, such as using Cython and OpenMP to fit the models in parallel among all available CPU cores. Even with CUDA kernels - fitting on compatible GPU's.

The Matrix Factorization method for explicit dataset is more mature than implicit dataset. Inspired by techniques used by traditional factor models, we can try:

Solution:

- Incorporating temporal information about viewing habits, so that changing user preferences over time could be accounted for.
- Adding bias to avoid large systematic tendencies for some users to give higher ratings than others, and for some items to receive higher ratings than others.

Neighborhood Models

The major drawback of Neighborhood Models is its huge computation cost at test time. For real time recommendation system, it is unacceptable to have such high computation time to obtain the results.

Solution:

- Instead of use all the users to predict one user's preference, we could use k nearest users to approximate the prediction. We could use K-D tree for indexing to reduce the computation cost.

References

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

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

[3] Eric Le. How to build a simple song recommender system, 2017.

[4] Empirical Analysis of Predictive Algorithms for Collaborative Filtering. pages 1–10, July 2011.

[5] Insight Data. Explicit matrix factorization: Als, sgd, and all that jazz, 2016.