Music Recommender System
Yuhang Lin (Graduate), Fayha Almutairy (Graduate), Nisha Chaube (Graduate)
Participated in CS Fair (Booth: 9 - north, Advanced Projects (CS 2xx/3xx level))

Abstract

According to Nielsen’s Music 360 2014 study, 93% of the U.S. population listens to music, spending more than 25 hours each week jamming out to their favorite tunes [1]. Recommender systems have taken the entertainment and e-commerce industries by storm. Amazon, Netflix, and Spotify are great examples.

In this project, we have designed, implemented and analyzed song recommendation systems using various algorithms. Music recommendation is a very difficult problem as we have to structure music in a way that we recommend the favorite songs to users which is never a definite prediction. It is dynamic and sometimes influenced by factors other than users' or songs' listening history. We have used Million Song Dataset [2], a freely-available collection of audio features and metadata for a million contemporary popular music tracks provided by Echo Nest, to find the correlations between users and songs and to learn from the previous listening history of users to provide recommendations for songs which users would prefer to listen most. We will discuss the problems we faced, methods we have implemented, results and analysis. We have achieved best results using neural network collaborative filtering algorithm.

1. Introduction

Music recommender system is a system which learns from the user's past listening history and recommends them songs which they would probably like to hear in future. We have implemented various algorithms to try to build an effective recommender system. We firstly implemented popularity based model which was quite simple and intuitive. Collaborative filtering algorithms which predict (filtering) taste of a user by collecting preferences and tastes from many other users (collaborating) are also implemented. We have also done experiments on neural collaborative filtering to find the best parameters for the model.
Our models will try to analyze and minimize the error between predicted ratings and actual ratings.

2. Literature review

The existing recommender systems using collaborative filtering algorithms have gained a great success. Netflix opened a challenge for the best collaborative filtering algorithm [3], and the winning algorithm using latent factor models could make 10.09% improvements over the algorithm used by Netflix at that time. Amazon uses user-user based and item-item based collaborative filtering [4], which greatly contributes to the
success of the business. Recently a newer algorithm using neural network, neural collaborative filtering (He 2017) [5], was proposed.

For content based algorithm, a lot of researchers have proposed different methods using Machine Learning technique, such as Decision Tree based [6], Support Vector Machine based [7], and even logistic regression [8]. We can fully utilize the knowledge we learnt from the class to implement these algorithms.

Music recommendation system shares some similarities with other commercial recommendation systems, but it focuses more on providing good and personalized advice on music, rather than goods for users to buy. The ideal music recommender system should be able to automatically recommend personalized music to human listeners. Different from books or movies, the length of a piece of music is much shorter, and the times that listening their favorite songs are normally more than once, which are the main difficulties we are going to face in this project.

3. Implementation Details:

3.1 Metrics

Root Mean Square Error

Since we aim to predict the ratings that each user will give to each item, we want to minimize the difference between the predicted value and the actual value. A common metric for such problems is the root mean square error (RMSE).

The RMSE formula is defined as:

\[
RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}}
\]

The regression line predicts the average y value associated with a given x value. Note that is also necessary to get a measure of the spread of the y values around that average. To do this, we use the root-mean-square error (r.m.s. error).

To construct the r.m.s. error, we first need to determine the residuals. Residuals are the difference between the actual values and the predicted values:

\[
y_i - \hat{y}_i
\]

\(y_i\) is the observed value for the ith observation and \(\hat{y}_i\) is the predicted value.
Sparsity
Another important aspect to look at when dealing with a collaborative filtering problem is the sparsity of the user-item matrix. The sparsity is defined as the ratio of the number of known ratings over the total number of possible ratings.

\[ \text{sparsity} = \frac{\text{total number of ratings}}{\text{number of unique users} \times \text{number of unique items}} \]

3.2 Dataset
We used dataset provided by Million Songs Dataset, which is also posted on Columbia University.
Source: https://labrosa.ee.columbia.edu/millionsong/

- The core data is the Taste Profile Subset released by The Echo Nest as part of the Million Song Dataset.
- It contains around 48 million (userid, songid, play count) triplets/features collected from histories of over one million users and metadata of millions of songs (280 GB).
- The users are anonymous with no availability of their demographic data or song listening timestamp.
- The feedback is implicit as play-count is given instead of explicit ratings.

We worked on the subset of 10K triplets which is present on turi.com because working on entire dataset is memory and CPU intensive, but the code should transpose seamlessly on the entire dataset.

Files:
10000.txt downloaded from https://static.turi.com/datasets/millionsong/10000.txt
song_data.csv downloaded from https://static.turi.com/datasets/millionsong/song_data.csv

Basic Dataset Statistics
For the first 10k triplets (user_id, song_id, listen_count), it contains 365 unique users and 5175 unique songs

```
N_users = song_df.user_id.unique().shape[0]
N_items = song_df.song_id.unique().shape[0]
print('Number of users = ' + str(N_users) + ' | Number of songs = ' + str(N_items))
```

Number of users = 365 | Number of songs = 5175

The top 10 triplets are listed below:
There are 5 columns we can get from “song_data.csv”: “song_id”, “title”, “release”, “artist_name” and “year”.

After merging the song metadata and user listening history using “song_id”, we can get:

<table>
<thead>
<tr>
<th>user_id</th>
<th>song_id</th>
<th>listen_count</th>
<th>title</th>
<th>release</th>
<th>artist_name</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b8034d063b5cc32127f653f3d9e43d87dca9e</td>
<td>SOAKIMP12A8C130995</td>
<td>1</td>
<td>Thicker Than Water</td>
<td>Jack Johnson</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>b8034d063b5cc32127f653f3d9e43d87dca9e</td>
<td>SOBBMDR12A8C13253B</td>
<td>2</td>
<td>Ente Dois Aqaus</td>
<td>Paco De Lucia</td>
<td>1976</td>
</tr>
<tr>
<td>2</td>
<td>b8034d063b5cc32127f653f3d9e43d87dca9e</td>
<td>SOBKHDL12A81C204C0</td>
<td>3</td>
<td>Stronger</td>
<td>Kanye West</td>
<td>2007</td>
</tr>
<tr>
<td>3</td>
<td>b8034d063b5cc32127f653f3d9e43d87dca9e</td>
<td>SOBOYAJ12A6701BF1D</td>
<td>1</td>
<td>Constellations</td>
<td>Jack Johnson</td>
<td>2005</td>
</tr>
<tr>
<td>4</td>
<td>b8034d063b5cc32127f653f3d9e43d87dca9e</td>
<td>SODCBBL12A8C13273</td>
<td>1</td>
<td>Learn To Fly</td>
<td>Foo Fighters</td>
<td>1999</td>
</tr>
<tr>
<td>5</td>
<td>b8034d063b5cc32127f653f3d9e43d87dca9e</td>
<td>SODDNQT12A6D4F5F7E</td>
<td>5</td>
<td>Alegria Por El Rock N Roll</td>
<td>Heróes del Silencio</td>
<td>2007</td>
</tr>
<tr>
<td>6</td>
<td>b8034d063b5cc32127f653f3d9e43d87dca9e</td>
<td>SODXTY12A8016DF3B</td>
<td>1</td>
<td>Paper Gangsta</td>
<td>Lady GaGa</td>
<td>2008</td>
</tr>
<tr>
<td>7</td>
<td>b8034d063b5cc32127f653f3d9e43d87dca9e</td>
<td>SOFUGAY12A8017B0A8</td>
<td>1</td>
<td>Stacked Actors</td>
<td>Foo Fighters</td>
<td>1999</td>
</tr>
<tr>
<td>8</td>
<td>b8034d063b5cc32127f653f3d9e43d87dca9e</td>
<td>SOFRQTD12A81C233C0</td>
<td>1</td>
<td>Sehr kosmisch</td>
<td>Harmonia</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>b8034d063b5cc32127f653f3d9e43d87dca9e</td>
<td>SOHQWYZ12A6D4FA701</td>
<td>1</td>
<td>Heaven's gonna burn your eyes</td>
<td>Thievery Corporation feat. Emiliánna Torrini</td>
<td>2002</td>
</tr>
</tbody>
</table>

Some songs’ release year are missing (listed as 0), as we can see from above. If we use these data to build a matrix, the sparsity level is 99.5%.

### 3.3 Libraries used in this project:
Surprise, TensorFlow, Keras, scikit-learn, graphLab, Pandas, Matplotlib, and Numpy

### 3.4 Algorithms
We have implemented various techniques and algorithms to build an efficient recommendation system:
Popularity based model

It is the most basic and simple algorithm. We find the popularity of each song by looking into the training set and calculating the number of users who had listened to this song. Songs are then sorted in the descending order of their popularity. For each user, we recommend top most popular songs except those already in his profile. This method involves no personalization and some songs may never be listened in future.

Nearest neighborhood model

Nearest Neighbourhood model involves collecting information from many users and then making predictions based on some similarity measures between users and between items. This can be classified into user-based and item based models.

In item-based model, it is assumed that songs that are often listened together by some users tend to be similar and are more likely to be listened together in future also by some other user. According to user based similarity model, users who have similar listening histories, i.e., have listened to the same songs in the past tend to have similar interests and will probably listen to the same songs in future too.

We need some similarity measure to compare between two songs or between two users. Cosine similarity weighs each of the users equally which is usually not the case. User should be weighed less if he has shown interests to many variety of items (it shows that either she does not discern between songs based on their quality, or just likes to explore). Likewise, user is weighted more if listens to very limited set of songs. The similarity measure, \( w_{ij} = P(i/j) \), also has drawbacks that some songs which are listened more by users have higher similarity values not because they are similar and listened together but because they are more popular.

SVD

SVD is a matrix factorization technique that is usually used to reduce the number of feature of a dataset by reducing the matrix from \( N \) space to \( K \) space where \( K < N \). For the purpose of the recommendation system however, we are only interested in the matrix factorization part keeping same dimensionality. The matrix factorization is done on the user-item ratings matrix built. The below is explained based on the paper by (Yehuda Koren, 2009)
Each item can be represented with a q vector. Similarly, each user can be represented by a p vector such that the dot product of these 2 vectors is the expected rating.

\[ \text{expected rating} = \hat{r}_{ui} = q_i^T p_u \]

Find p and q such that it minimizes the following:

\[ \text{minimum } (p, q) \sum_{(u,i) \in K} (r_{ui} - q_i^T p_u)^2 \]

For our model to be able to generalize well and not overfitting the training set, we introduce a penalty term our minimization equation. This is represented by a regularization factor multiplied by the square sum of the magnitudes of user and item vectors.

\[ \text{minimum } (p, q) \sum_{(u,i) \in K} (r_{ui} - q_i^T p_u) + \lambda (\|q_i\|^2 + \|p_u\|^2) \]

**Neural Collaborative Filtering**

This algorithm is similar to SVD. However, instead of applying dot project to user vector and item vector, neural network is used. User latent vector and item user vector are first fed into embedding layer and then the result is fed into Neural Network layers. We then compare the output of neural network with the actual rating to train the parameters. The graph below shows the network architecture.
3.5 Data Transformation
Our dataset doesn’t contain explicit rating so we decided to try with two different ways to get a transformed rating. One way is to use listen count as rating directly and the other way is to scale the listen count to 0 to 5 by the rule below:

The rating of user \( u \) on item \( i \) is based on the listen count \( C(u, i) \)

\[
R(u, i) = \begin{cases} 
5, & \text{if } C(u, i) > 50 \\
4, & \text{if } C(u, i) \leq 50 \\
3, & \text{if } C(u, i) \leq 40 \\
2, & \text{if } C(u, i) \leq 30 \\
1, & \text{if } C(u, i) \leq 20 \\
0, & \text{if } C(u, i) \leq 10 
\end{cases}
\]

3.6 Tuning parameters
We did a lot of experiments to find the best parameters for different models. First, we compared the result with and without L2 regularizer. With L2 regularizer (shown on the left) we can see it’s good at preventing overfitting. As the number of epochs increases, the validation error doesn’t increase so quickly like it does without L2 (shown on the right). However, after restoring from the session with the best validation error, the final test error without L2 (0.216) is actually better than with L2 (0.233).
Then we compared the result with and without dropout. Dropout can prevent the model from overfitting but again, we found that it didn’t help in the final test error (test error of 0.219 with dropout).

We were curious about the best optimizer. According to our experiment, Adam is the best choice.

Having decided to use Adam, we searched for the best learning rate and it turned out that 0.003 is the best choice.
4. Conclusion

We tuned the parameters and the table below shows the best result using item based, user based and SVD after using listen count and transformed rating. We can see that after doing transformation from listen count to 0-5 rating, all three models got improvement. And SVD always performs the best among them.

<table>
<thead>
<tr>
<th></th>
<th>Item Based</th>
<th>User Based</th>
<th>SVD</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Using Listen count (RMSE)</strong></td>
<td>6.594</td>
<td>6.621</td>
<td>5.566</td>
</tr>
<tr>
<td><strong>Using Rating (RMSE)</strong></td>
<td>3.929</td>
<td>3.970</td>
<td>1.1212</td>
</tr>
</tbody>
</table>
The graph and table below shows the result of experiments using neural collaborative filtering algorithm.
We can see that least test error of about 0.216 is obtained with learning rate of 0.003, Adam optimizer, truncated normal initializer, and 128, 64, 32, 16, 8 neurons in consecutive hidden layers.

Finally, we compare the best RMSE using all different algorithms we used. Obviously, neural collaborative filtering can give us the lowest test error.
5. Role of members

We did literature review together and studied about various possible approaches—pros/cons actively. Each one of us implemented all algorithms on our own to compare our results and better understand the problem. We also involved ourselves in discussion of our results and how we could improve our model performance. Each one of us worked to prepare the presentation and report together.

Yuhang Lin:
- Performed experiments with different parameters for neural collaborative filtering algorithm;
- Visualized the results;
- Presented at CS fair

Fayha Almutairy:
- Did experiments with Collaborative filtering algorithms (Item-based, User-based, Matrix factorization (SVD), and Neural networks).

Nisha Chaube:
- Performed experiments with Popularity based algorithms
- Presented at CS fair

6. Contributions

We did exhaustive experimentation using different models, tried to study the impact of using old and new models. Implemented neural network with different layers to minimize test error. Our aim was to not only apply techniques that we learned in class but to also explore other aspects and ideas to solve our task.

We are the first to implement Neural Collaborative Filtering on Million Song Dataset. We explored the best learning rate, best optimizer, best initializer, with and without L2 and dropout regularizers for this dataset.

7. Future work

In the future, we would like to try the following things:
1. Using audio signal (e.g. audio frequency) to recommend songs
2. Trying content based algorithm
3. Trying Convolutional Neural Network
4. Making the recommender system a real-time system
5. Trying clustering techniques to recommend music
8. References


