Music Recommendation System Analysis Using Million Song Dataset

Dhwani Jain: dj384@cornell.edu | Srishti Gupta: sg2435@cornell.edu | Shaumik Ashraf: sa2243@cornell.edu

Abstract

By utilizing the Million Song Dataset (MSD), we tried different algorithms for a music recommendation system in order to evaluate them for performance. In this project, we compare the popularity model, item-based collaborative filtering model, item-based model with Singular Value Decomposition(SVD), and k-Nearest Neighbour(k-NN). For evaluating machine learning models we computed the F-1 score for popularity model and k-NN, while root mean squared error (RMSE) for popularity, item-based and SVD models. We concluded that the item-based model with SVD performed the best for music recommendation amongst the algorithms implemented.

Motivation

Music is something that almost everyone can relate to and music recommendation systems are now common on platforms such as Youtube or Spotify. By exercising different machine learning techniques on a skeleton music recommendation system, we gain a deeper understanding of the various techniques and their application to personalize music. We can draw our own conclusions on the best implementation for a music recommendation system on the basis of various performance metrics and compare it to the existing literature. Thus this type of project is method based.

Background

Generally algorithms can make non-personalized or personalized recommendations to users. For the non personalized system, we explored a popularity based model, which can also be regarded as the baseline model and naive implementation. For personalized recommendation systems there is content-based as well as collaborative-based filtering mechanisms, of which we pursued the latter**.** We initially focused on item-item based recommendation systems. One of the disadvantages of item similarity is a large, sparse co-occurrence matrix which could increase computation time and memory, which can be improved by an alternative SVD implementation. Another commonly used collaborative filtering algorithm is k-NN, which also has the drawbacks of large memory consumption. While memory consumption and runtime are of concern, this biggest factor for a recommendation system is its ability to successfully predict songs liked by a user.

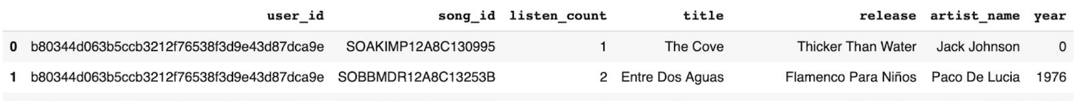
Method

Data Exploration

The data we used is from the Million Song Dataset (MSD). We have 2 files from where we extracted the data to train our models. A brief description of each of them is provided below:

1. User Data File: This dataset contains 3 columns namely 'user_id', 'song_id', 'listen_count' for 76353 unique users with a total of 2000000 rows of user data and their songs count.
2. Song Data File: This dataset contains 5 columns namely ‘song_id’, ‘title’, ‘release’, ‘artist_name’ and ‘year’. This contains the data for 999056 unique songs with a total of 1000000 songs.

We joined our 2 datasets by song_id to better understand and explore the data as shown below.



	user_id	song_id	listen_count	title	release	artist_name	year
0	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOAKIMP12A8C130995	1	The Cove	Thicker Than Water	Jack Johnson	0
1	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBBMDR12A8C13253B	2	Entre Dos Aguas	Flamenco Para Niños	Paco De Lucia	1976

There are no empty rows in the dataset. Many users only listened to 1 song, while one user listened to one song 2213 times (maximum listen count). The average listen count was 3, with a standard deviation of 6.5. The resulting listen count matrix is very sparse. See appendix for the top 20 songs and artists.

Models used for training

Popularity Based Recommendation System: This is the most naive form of recommendation system that does not make personalized song recommendations to the users. In this algorithm songs are sorted based on their popularity and each user is recommended the top most popular songs. This was the baseline model for comparing all other models.

Collaborative Filtering via Item Similarity: The intuition behind item-based models is to make recommendations by finding other songs to those in similar users' listening history. The model is trained on the data from existing users to create a cooccurrence matrix, which is normalised by cosine's similarity (given by the formula below).

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2}\sqrt{\sum\limits_{i=1}^{n} B_i^2}},$$

A score is assigned to each item-item pair in the cooccurrence matrix and based on user listening history, a weighted average of the items from the matrix is taken. The scores are sorted in descending order and the next song is recommended to the user based on the score. The main disadvantage of this algorithm is the sparsity of the cooccurrence matrix, reducing accuracy while also increasing memory consumption and runtime.

Singular Vector Decomposition (SVD) Optimization: SVD factorizes the rating matrix M into a latent feature space relating users to songs and songs to users. This is achieved by factoring M into user matrix and item matrix with minimum of error.

$$M = U^T V \quad \textit{where } U \in R^{k\times m},\ V \in R^{k\times n}$$

U is a low-rank representation of the M users represented by k-dimensional vectors, and V represents the n items (in this case songs) also represented by k-dimensional vectors/features. Each of the k vectors is a feature and the user matrix U, represents the user's degree of interest in that feature. These can be features like pop, rock etc (genre of music) in the context of song recommendations. Matrix V represents the relevance of each song to the k features. To get the U and V matrices, the following error is minimized:

$$Err(s) = argmin\ \Big(\sum_{u\epsilon U,\, s\epsilon S} (V(score_{u,s}) - V\ ) \qquad \textit{where } V(score_{u,s}) \textit{ is the predicted score}$$

Using the U and V matrices, personalized recommendations for a user can be calculated by ranking items in descending order of the predicted score. The score for a user u for songs s, is calculated as follows:

$$Score_s = U^T_u V_s$$

Collaborative Filtering via K-Nearest Neighbors (KNN): A KNN model is commonly used for classification problems by storing the entire training data set in memory, finding the distance from a new point to every point in the training data set, selecting the K nearest data points by some distance metric, and then determining a predictive label by simple or weighted voting. For the context of collaborative filtering, an unlabelled, unsupervised variant of KNN can be done by taking the K nearest points - where each point represents a user's listen counts - and using it to recommend new songs. There are three hyperparameters to consider for KNN: Count Threshold, K, and Distance Metric. Since KNN stores the entire dataset, it is very memory intensive, so outlier filtering is required to make KNN feasible. The Count Threshold value is the minimum number of times a user must have listened to any song to not be considered an outlier. The K value is the number of nearest neighbors to consider when pulling new song recommendations, and the Distance Metric is the formula for determining the nearest neighbors. The go-to metric for KNN is usually Euclidean distance (or Minkowski distance with p=2) whose formula is shown below. Cosine similarity is another common metric used specifically in the context of collaborative filtering because it provides a normalized distance value.

$$euclidean(A,B) = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2}$$

Metrics to rate and compare models

Precision, Recall and F1 score: While working with the subset of 10000 songs we used precision recall and F1 scores as a metric to compare different models. These metrics are computed as per the formulae below. Precision is a measure of accuracy, while recall is a measure of completeness. F1 score is a composite score of both precision and recall and is useful for comparison purposes.

$$Precision = \frac{True\ Positive}{True\ Positive\ + False\ Positive},\quad Recall = \frac{True\ Positive}{True\ Positive\ + False\ Negative},\quad F1 = \frac{2\times Precision}{Precision + Recall}$$

Root Mean Square Error (RMSE): We aim to predict the ratings that each user will give to the songs, and we want to tune the models to minimize the difference between the predicted value and the actual value. This metric is useful as it can be used to compare the accuracy of recommendations of different algorithms. The RMSE formula is defined as:

$$RMSErrors = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y_i} - y_i)^2}{n}}$$

where $\hat{y}_i$ is the predicted rating value and y_i is the actual rating value provided by a user to the song.

Experimentation and Results

Initial experimentation with a subset of the dataset

We are using a large dataset to build the recommendation system. To start off the project and understand the different algorithms, we used a subset of the data with the first 10000 songs. Using this subset we implemented the popularity based, item based, and KNN algorithms on the dataset. To compare these models we computed the precision, recall and F1 scores (Refer to appendix for the precision recall curve). Based on the initial analysis, item based method outperformed both the popularity based method and KNN with F1 scores of 39%, 19%, 7% respectively. (In hindsight, the best combination of KNN hyperparameters still yields only 10%).

Popularity Recommendation System - Further Analysis: We split the whole dataset into training and test data in a ratio of 70% and 30%. We used the turi library, with the target parameter as 'listen count' for preparing the training dataset. Using this library we obtained the 10 most popular songs and then recommended those to the users. Please refer to the appendix to view the screenshots for the following analysis. We noticed that it only recommends popular songs to all the users and the recommendation is not personalized for each user. Turi has an inbuilt class for computing the RMSE values and using that the RMSE value for the test dataset was 7.84.

Item Similarity: Turi library has an inbuilt class for item similarity recommendation system which uses cosine similarity. We started by preparing the training dataset using the turi library and then made recommendations on the test dataset. As an example, refer to appendix for screenshots for recommending songs to a particular user based on item similarity using this algorithm. Using the inbuilt function in Turi we computed the RMSE for the test dataset to be 5.90.

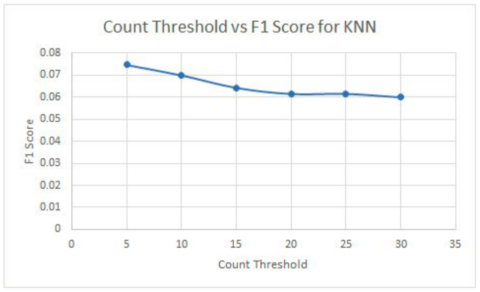
SVD Optimization: We used the surprise library to carry out matrix factorization. We used gridsearch to tune the number of factors in the range [0,2,4,6,8,10] and based on the RMSE values obtained we obtained the optimal number of features to be 8. Using k=8, we trained our model on the training dataset and made predictions on users from the test dataset. We performed a 5 fold cross validation for this method and the average RMSE value was computed to be 5.65.

In the experiments so far we used listen count as a representation of rating for training the model. However, this might not be the best way to account for user preferences as the data is heavily skewed with some users listening to the same song >2000 times. Hence we decided to experiment with the models by scaling the listen count data in a range of (0,5). After scaling the ratings column, we re-ran all the experiments and the RMSE values obtained are reported below:

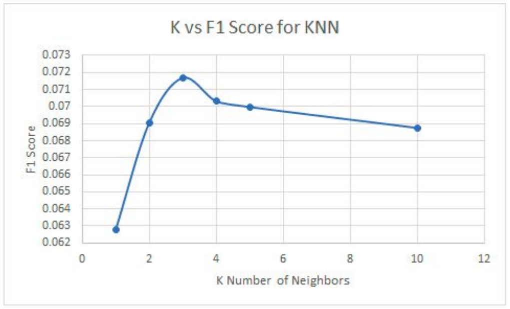
	Popularity	**Item based**	**SVD**
RMSE (user rating)	4.09	3.99	1.56

KNN Optimization: For all experiments of the KNN model, the 10000 songs 76353 user subset was used. 3000 users were held out from the data for testing regardless of the Count Threshold, and a second holdout was then done on 30% of the songs per user for the 3000 users. The default hyperparameter values were a Count Threshold of 20, K value of 1, and Cosine Distance.

Count Threshold: Since KNN models consume a lot of memory, good data preprocessing to narrow down the train dataset is necessary for a feasibly runnable model. The Count Threshold was a designated cutoff where any users who listened to less songs than the Count Threshold value was considered an outlier and removed from the dataset. Any songs which then had no user representation were also filtered out. See appendix for Count Threshold values and resulting dataset sizes. Any Count Threshold less than 5 would result in a Segmentation Fault (Memory Allocation Error). The lowest Count Threshold performed best, presumably because of the larger training dataset. It is worth noting that a larger dataset with better hardware may have yielded better results.



K variation: K values of 1, 2, 3, 4, 5, and 10 were tried over a dataset with Count Threshold 20 and Cosine Distance Metric. K=3 is the best performing value.

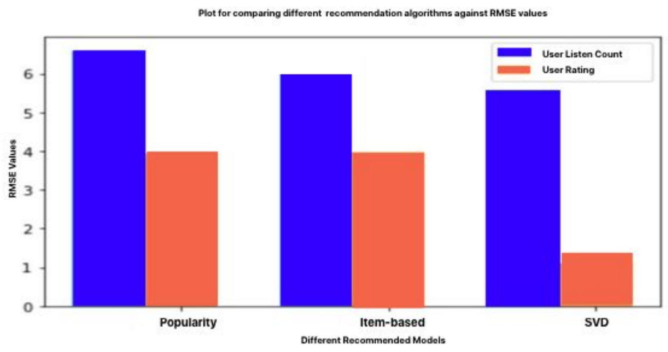


Distance Metric: Both Euclidean metric and Cosine Similarity metric were tested with the KNN model, and the Euclidean Metric was found the better. See appendix for results.

Conclusion from our experiments:
For KNN we computed the F1 score and realized that the prediction accuracy is very low. Using the most optimal KNN build (K = 3, Count Threshold = 5, and Euclidean Distance Metric), the F1 score for KNN with our dataset was still computed to be ~10.31% which is even lower than the baseline model of popularity based method of ~19%. The KNN model was also the least efficient in terms of runtime and memory consumption, and could not be run on the larger dataset unlike the other 3 models. In comparing the RMSE scores for the item similarity, popularity model, and item-based with SVD model, the latter outperformed the former two. We also noticed that these models gave less error when compared based on user rating in range(0, 5). The RMSE values computed for the different models are reported below.

	Popularity	Item based	Item based with SVD
RMSE (user listen count)	7.84	5.90	5.65
RMSE (user rating)	4.09	3.99	1.56

The graph below shows our results from the experiments when comparing different recommendation algorithms.



Conclusion and future work

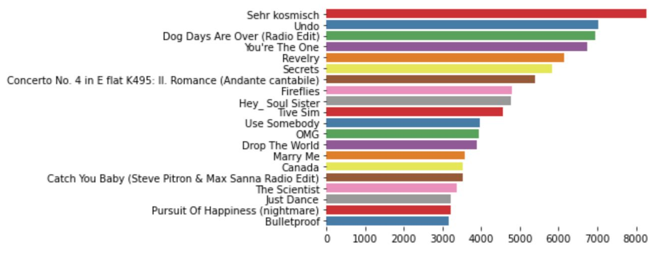
We did numerous experiments on different models as described above. We tried various implementations of the different algorithms, and to attain efficiency in the algorithm, we decided on using the libraries like turi, surprise and sklearn in our project. Our aim was to learn and understand which algorithm gives accurate and efficient results when it comes to recommending music to a particular user. Based on our experiments, SVD works best for the recommendation system. For future work we would like to explore datasets with more song features so that the accuracy can be improved and we would also like to try out clustering algorithms as well as content based recommendation systems. In addition, we also think using neural networks for collaborative filtering could be useful and we would like to explore that aspect as well.

References

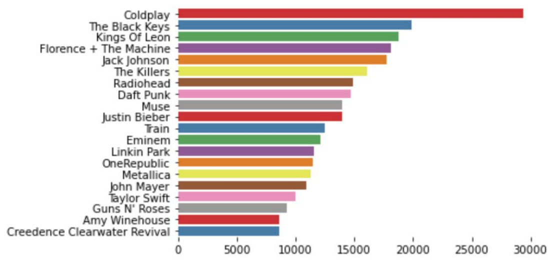
1. http://millionsongdataset.com/
2. https://static.turi.com/datasets/millionsong/10000.txt
3. https://static.turi.com/datasets/millionsong/song_data.csv
4. https://www.kaggle.com/c/msdchallenge
5. https://www.youtube.com/watch?v=18adykNGhHU
6. http://cs229.stanford.edu/proj2012/NiuYinZhang-MillionSongDatasetChallenge.pdf
7. https://github.com/ugis22/music_recommender
8. https://analyticsindiamag.com/singular-value-decomposition-svd-application-recommender-system/
   Everyone listens to music, but how we listen is changing. [online] Available at: http://www.nielsen.com/us/en/insights/news/2015/everyone-listens-to-music-but-howwe-listen-is-changing.html [Accessed 10 Oct. 2017]. [2] Labrosa.ee.columbia.edu. (2017). Million Song Dataset | scaling MIR research. [online] Available at: https://labrosa.ee.columbia.edu/millionsong/ [Accessed 10 Oct. 2017].

Appendix

1. **Top 20 songs and top 20 artists of dataset**

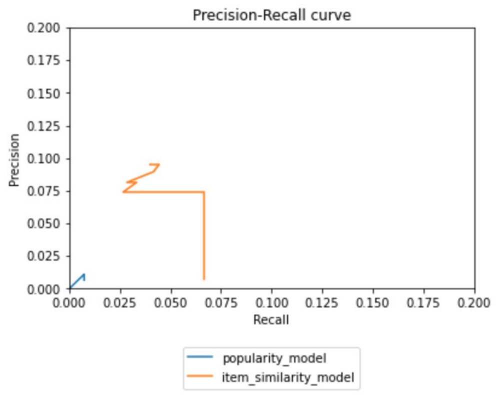


Top 20 songs in the dataset



Top 20 artists in the dataset

2. **Precision-Recall comparing popularity model and item-similarity model with subset of data**



3. Popularity recommendation System:

```
+------+--------------------+--------------------+------+
| user |        song        |        score       | rank |   Starshine
+------+--------------------+--------------------+------+   221
|  1   | SOFCGSE12AF72A674F | 25.45744680851064  |  1   |   The Quest
|  1   | SOACBLB12AB01871C7 | 18.196428571428573 |  2   |   Gestern_Morgen
|  1   | SOZPMJT12AAF3B40D1 | 15.595238095238095 |  3   |   Working With Homesick
|  1   | SOXQIUR12A8AE4654A | 14.718446601941748 |  4   |   Clara meets Slope - Hard To Say
|  1   | SOGSDHY12AB017BF39 | 14.692307692307692 |  5   |   Caroline
|  1   | SOJSXJY12A8C13E32E | 14.530054644808743 |  6   |   MIC (Speak Life Album Version)
|  1   | SOAFPSO12AF72A4521 | 13.928571428571429 |  7   |   Recado Falado (Metrô Da Saudade)
|  1   | SOQBUFQ12A6D4F7F4C | 13.446153846153846 |  8   |   Hounds of Love (new mix)
|  1   | SONDKOF12A6D4F7D70 | 12.58108108108108  |  9   |   Starshine
|  1   | SOPKTFQ12A67021600 | 11.492307692307692 |  10  |   221
|  2   | SOFCGSE12AF72A674F | 25.45744680851064  |  1   |   The Quest
|  2   | SOACBLB12AB01871C7 | 18.196428571428573 |  2   |   Gestern_Morgen
|  2   | SOZPMJT12AAF3B40D1 | 15.595238095238095 |  3   |   Working With Homesick
|  2   | SOXQIUR12A8AE4654A | 14.718446601941748 |  4   |   Clara meets Slope - Hard To Say
|  2   | SOGSDHY12AB017BF39 | 14.692307692307692 |  5   |   Caroline
|  2   | SOJSXJY12A8C13E32E | 14.530054644808743 |  6   |   MIC (Speak Life Album Version)
|  2   | SOAFPSO12AF72A4521 | 13.928571428571429 |  7   |   Recado Falado (Metrô Da Saudade)
|  2   | SOQBUFQ12A6D4F7F4C | 13.446153846153846 |  8   |   Hounds of Love (new mix)
|  2   | SONDKOF12A6D4F7D70 | 12.58108108108108  |  9   |
|  2   | SOPKTFQ12A67021600 | 11.492307692307692 |  10  |
+------+--------------------+--------------------+------+
```

4. Collaborative Filtering - Item Similarity:

```
+------+--------------------+----------------------+------+
| user |        song        |        score         | rank |
+------+--------------------+----------------------+------+
| 100  | SOFRQTD12A81C233C0 | 0.01054956078529358  |  1   |   Sehr kosmisch
| 100  | SONYKOW12AB01849C9 | 0.010125592947006226 |  2   |   Secrets
| 100  | SOUSMXX12AB0185C24 | 0.009257566928863526 |  3   |   OMG
| 100  | SOIZLKI12A6D4F7B61 |  0.0086257004737854  |  4   |   Supermassive Black Hole (Twilight Soundtrack Version)
| 100  | SOAXGDH12A8C13F8A1 | 0.00831252932548523  |  5   |   Dog Days Are Over (Radio Edit)
| 100  | SODJWHY12A8C142CCE | 0.008293513059616089 |  6   |   Hey_ Soul Sister
| 100  | SOUFPNI12A8C142D19 | 0.008132596015930176 |  7   |   Heartbreak Warfare
| 100  | SOTWSXL12A8C143349 | 0.007302753925323487 |  8   |   Love Story
| 100  | SOEGIYH12A6D4FC0E3 | 0.007162244319915772 |  9   |   Horn Concerto No. 4 in E flat K495: II. Romance (Andante cantabile)
| 100  | SOKLRPJ12A8C13C3FE | 0.007014883756637573 |  10  |   The Scientist
+------+--------------------+----------------------+------+
```

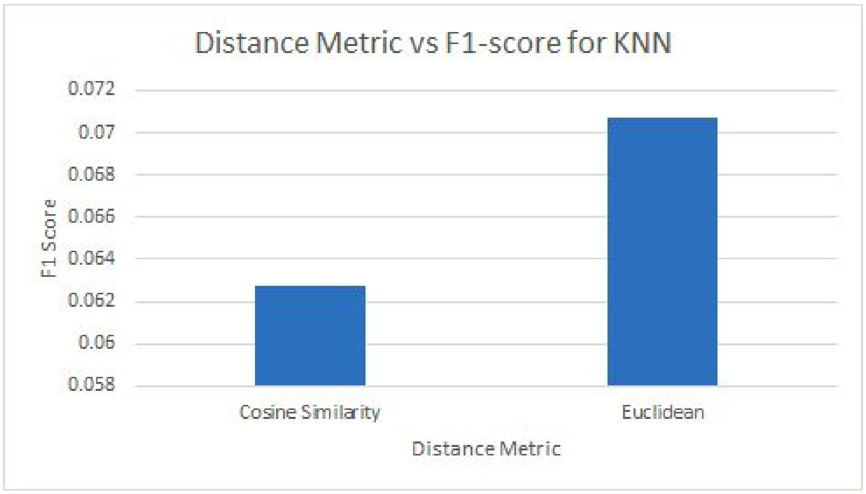
5. SVD on Item Similarity:

```
+------+--------------------+--------------------+------+
| user |        song        |        score       | rank |
+------+--------------------+--------------------+------+   Sehr kosmisch
| 100  | SOFRQTD12A81C233C0 | 3.0754465613086466 |  1   |   You're The One
| 100  | SOBONKR12A58A7A7E0 | 3.0670548233707193 |  2   |   Dog Days Are Over (Radio Edit)
| 100  | SOAXGDH12A8C13F8A1 | 3.066875581028009  |  3   |   Undo
| 100  | SOAUWYT12A81C206F1 | 3.065289378495956  |  4   |   Revelry
| 100  | SOSXLTC12AF72A7F54 | 3.0609028715212587 |  5   |   Supermassive Black Hole (Twilight Soundtrack Version)
| 100  | SOIZLKI12A6D4F7B61 | 3.0606270473022703 |  6   |   Horn Concerto No. 4 in E flat K495: II. Romance (Andante cantabile)
| 100  | SOEGIYH12A6D4FC0E3 | 3.056881646024132  |  7   |   Hey_ Soul Sister
| 100  | SODJWHY12A8C142CCE | 3.0544256124217752 |  8   |   Billionaire [feat. Bruno Mars]  (Explicit Album Version)
| 100  | SOPTLQL12AB018D56F | 3.0535880479533914 |  9   |   Canada
| 100  | SOPUCYA12A8C13A694 |  3.05313151548996  |  10  |
+------+--------------------+--------------------+------+
```

6. Count threshold for kNN for different training dataset size

Count Threshold	Training Dataset Size
5	65305
10	49018
15	35577
20	26980
25	21187
30	16540

7. **<u>Bar Graph for Distance Metric vs F-1 score for KNN</u>**



8. **<u>Hardware and Software</u>**

All code was done in Python 3 using the following libraries in addition to the standard python library: Turi, Surprise, Pandas, Numpy, sklearn, and scipy. The popularity based model and collaborative filtering model were done on Google Colaboratory, powered by Google Cloud and Jupyter Notebooks, while KNN was done locally on Spyder IDE, Anaconda environment, Windows 10 with 12 GB RAM and intel i5-8th generation CPU.