Artists Recommendation System Through Python

08.07.2019

Kris Seo
Mentor: Srdjan Santic

Overview

- Background

    In recent decades, music streaming service becomes ubiquitous as people with airpods can be easily seen and technologies on music storage change people's music consumption behavior. As one of the frequent streaming service users, I want to discover more and more artists or songs to extend my music preferences and to explore of trendy music. However, as online music services with high speed Internet enables enormous music collections of artists, it will be challenging streaming service users to figure out which artists or songs they will prefer from the collections.

- Problem

    Streaming service users usually listen what they found interesting; so, the artists and the songs are limited without a recommendation system. The limitation of their preferences might lead users to get frustrated or overwhelmed to the point that they interrupt the using of service.

- Clients

    Every music streaming company or Business to Customer company can be my client. They care about this problem because they want to keep their users and the provided recommendations are critical core to retain users.

- Data

    I acquired users data from ocelma.net which used Last FM API to collect 360,000 users and their top artists. While I acquired spotify data from Spotify Rec System which contains Track ID of Spotify, Artist_name, track_name and playlist id. To extract each track's audio features, I used Spotify API and applied get_audio_features function.

- Approach

    To start building a recommendation system, I will like to inform some basic methods for Recommendation Systems. Usually Recommendation Systems follow these methods, Content Based Filtering, Collaborative Based Filtering, and lastly Hybrid Based Filtering.

    Content Based Filtering extracts relevant features of items and predicts which items the user will like based on user's history. The prediction only depends on the contents of the items. Collaborative Filtering uses user-item intersection history and predicts user preference by user-item intersection. With Collaborative Filtering, music recommenders will bring similar users based on the user's history and present the songs that other similar users like. I want to combine these filtering methods for Hybrid recommendation system.

- Deliverables

  Codes are available on here.

Goals

1. Use Spotify API, Spotipy, to import track's features based on its uri.
2. Use Tf-Idf vectorizer/Countvectorizer for Content Based Filtering to build a recommender system.
3. Use Matrix Factorization for Collaborative Filtering to build a recommender system for users.
4. Use Last FM and Implicit Alternating Least Square to build Hybrid Content-Collaborative Filtering to build a rec. system for users.

Data Wrangling

I. Data Preprocessing

After downloaded csv files of Spotify data, I had to use spotipy to extract each track's features based on provided track's uri and also artists' genres from the names. All track features dataframe and genre_artists data frame are saved as pickle file to load it faster later. Also, Last FM data were merged on each user's id to get play counts.

II. Data Cleaning

After Spotipy functions were applied to figure out each track features, there were tracks with minimum loudness, -60 db, which people cannot hear and features of tracks were all equal to the minimum value. So these tracks were treated as missing values and dropped. Then, no more missing values were detected.

III. Data Integration

I merged Artists Genre dataframe collected from Spotify and Last FM users profile dataframe based on the artist's name. The hybrid recommender system will be based on users and artists matrix and the matrix of artists and genres.

Before I build a recommender system, the data that I collected has to be cleaned and modified. In order to perform recommendations, the metrics I used

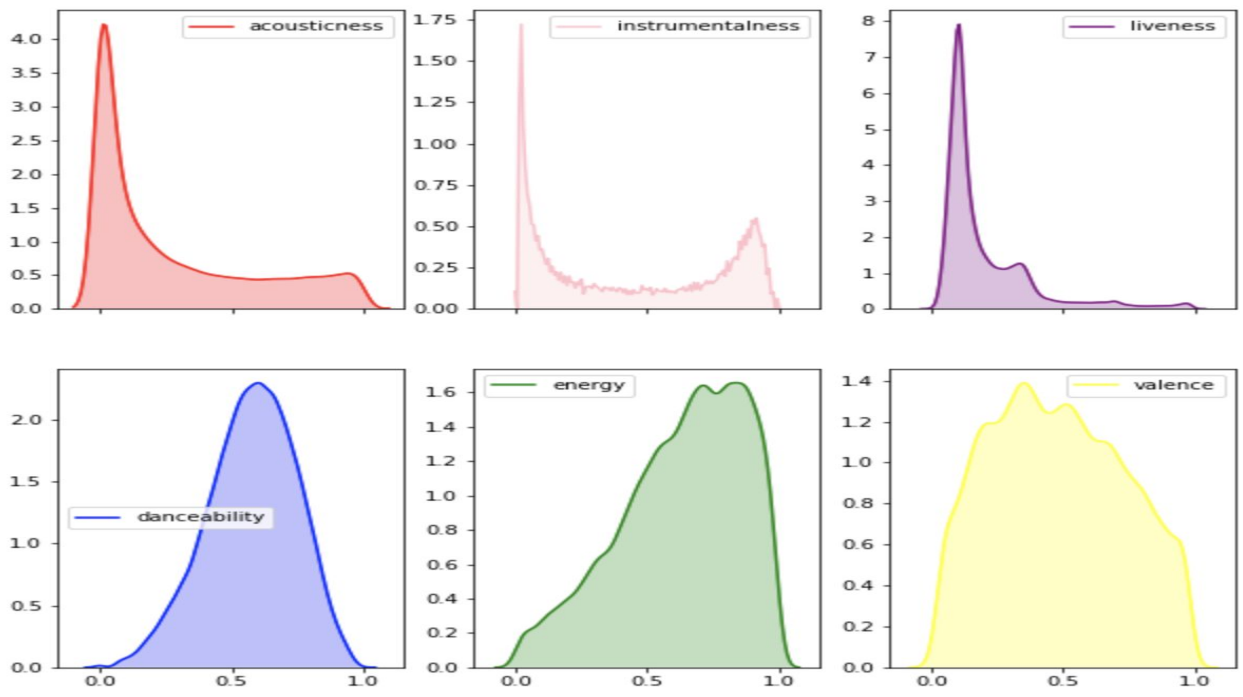
- Track Feature Matrix: Each row represents tracks with Spotify features
- Artists Genres Matrix: Each row represents artists with their genres.
- Users Artists Matrix : Each row represents users with their played artists.

For Users Artists Matrix, the values in the matrix is users play counts of their artists.

EDA

Spotify Track Features

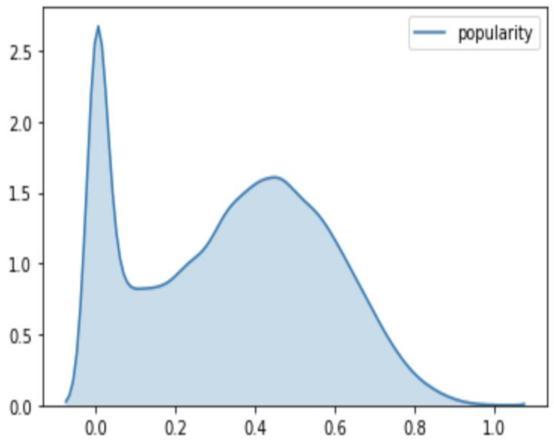
Figure1: Kernel Density Distributions of each features of tracks from track_features df.



Distributions of acousticness, instrumentalness and liveness are similar to each other. Each distribution contains lots of 0 values with low standard deviation; however, distributions of other except liveness, have a dip at each mean. This dip might represents that it has lots of mins and max values can be understood that classical and modern music. Density of Danceability and valence are normally distributed. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).

For recommendations system, long-tail part of distributions is used because items in the long tail are the items cater to people with unique niche interests. Therefore, to generate robust recommendation system, usage of data in long-tail should be considered instead of skewed part.

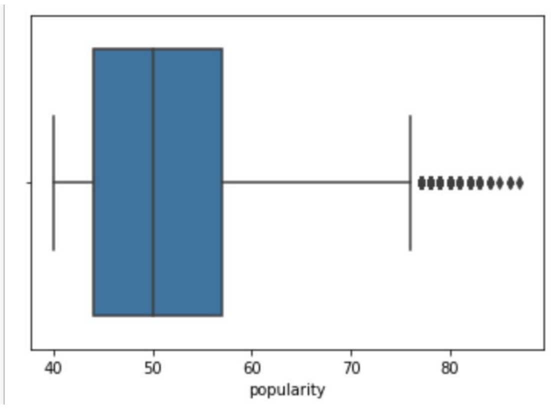
Figure2: Kernel Density Distribution of popularity of tracks from track_features df.



There were lots of tracks with 0 popularity. According to Spotipy documentation, 0 popularity means "Generally speaking, songs that are being played a lot now will have a higher popularity than songs that were played a lot in the past."

Users won't prefer listening today's non-popular song. So I set up some threshold based on quantiles to prevent noisy recommendations. 11845 songs are not listened to by regular users.

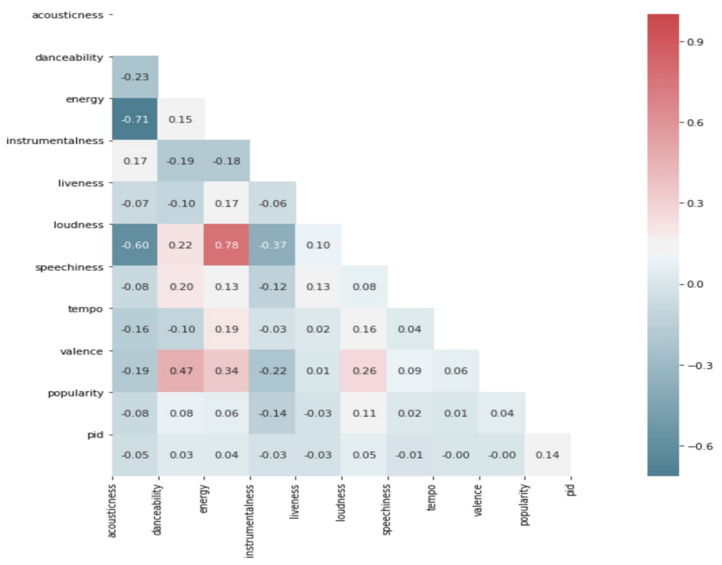
Figure3: Box plot of popularity of tracks after threshold applied.



Now, minimum popularity set to be 40 because the maximum popularity was not 100.

With popular songs, I could calculate the cosine similarity of each audio features after standardized all features to have mean 0 and standard deviation 1.

Figure4: Heat Map of each features to show the correlation

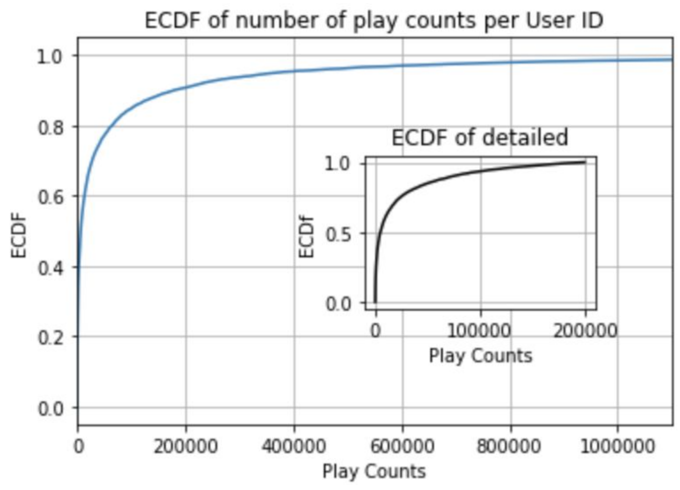


Energy and loudness are negatively correlated with acousticness, but strongly positively correlated to each other.

There is no significant factor that affects popularity; however, there is a slightly negative correlation between instrumentalness and popularity. It might be the fact that classical songs are practically instrumental and are not frequently on Top Charts.

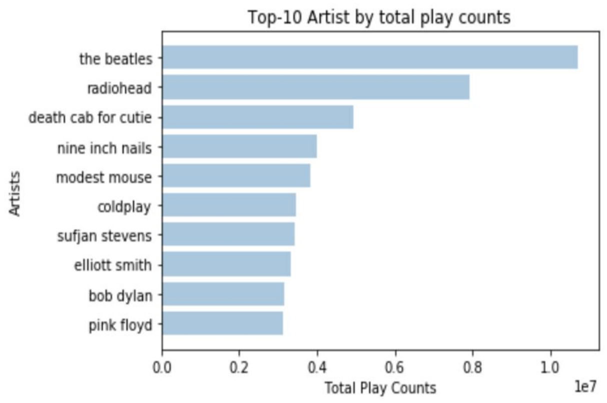
Last FM Artists Data

Figure5: ECDF of play counts per User



According to ECDF, the difference of minimum play counts and maximum play counts is huge for total plays and ECDF plot has a long tail. Median of play counts is 12,400. There are numerously many unique and underground artists, which can produce noisy and ambiguous recommendations, so reasonable threshold for counts is needed.

Figure 6: Top-10 Artists by total play counts and by individual play counts



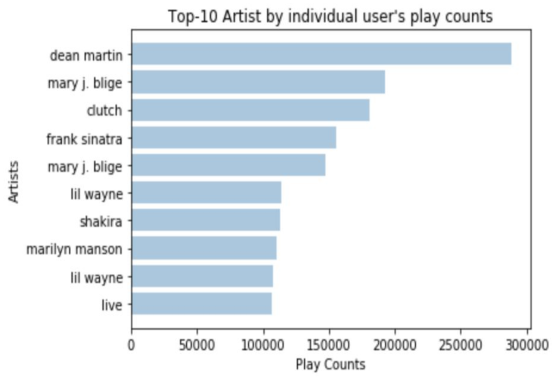
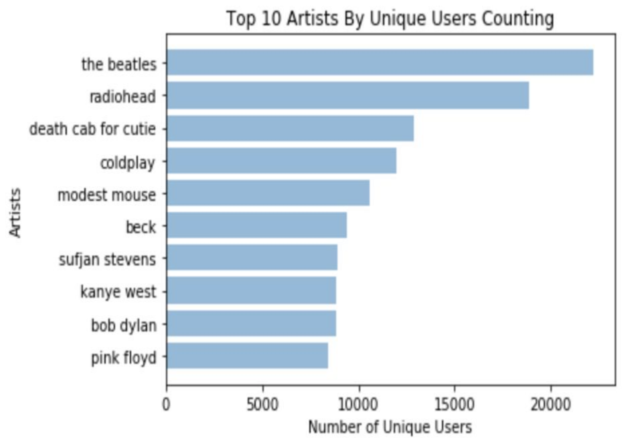


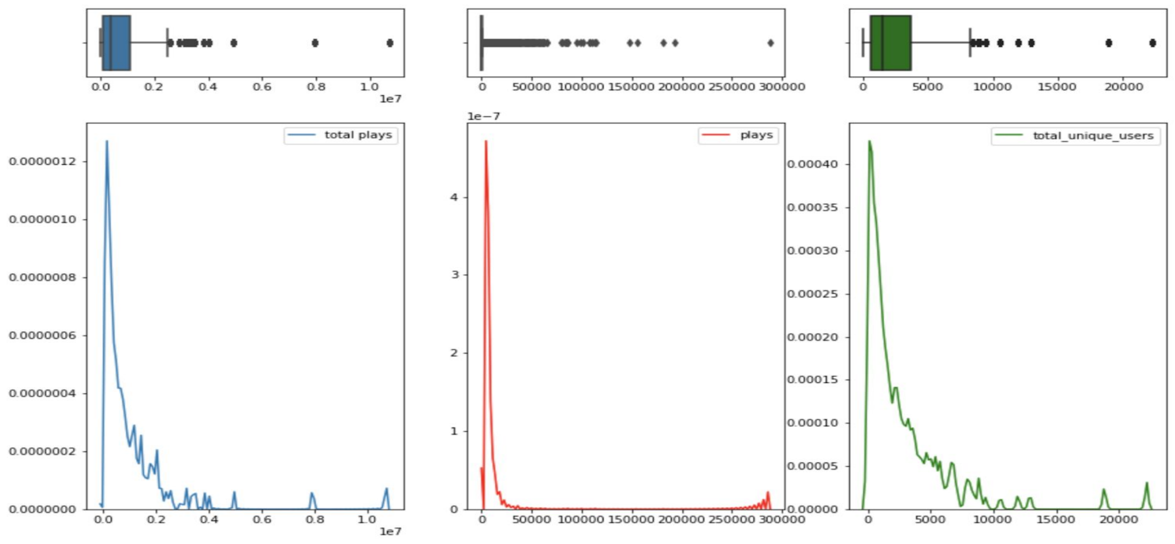
Figure7: Top-10 Artists by number of unique users



Top 10 artists in Figure 7 are pretty common in Top 10 artists by total play counts in Figure 6. From this bar graph, users will likely listen to popular and well-known artists.

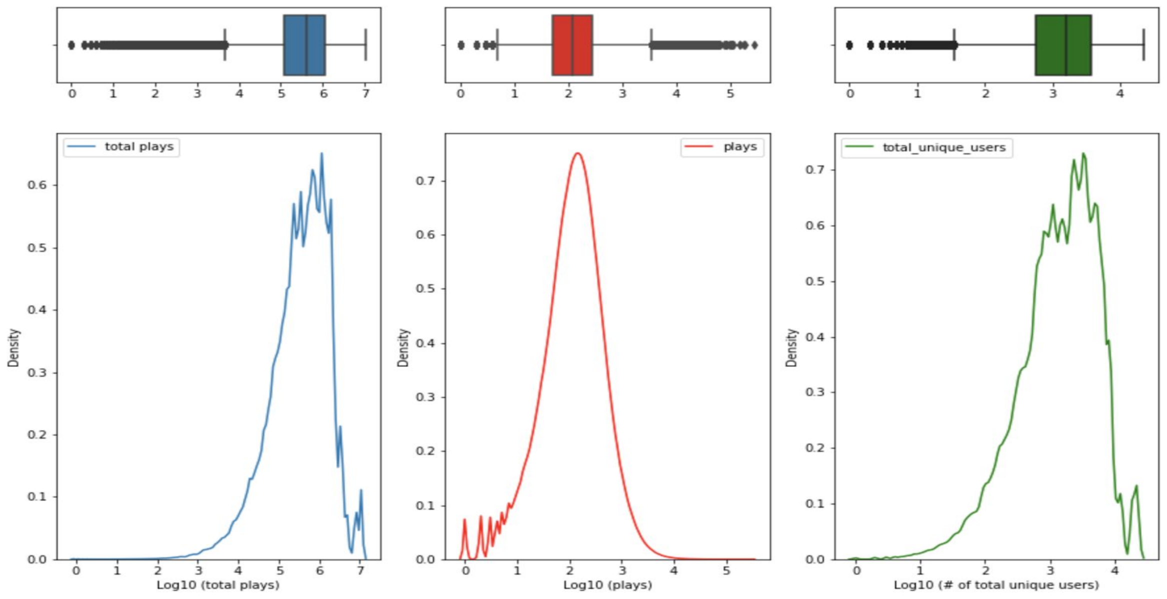
Let's see the distributions of each counting features to set up threshold.

Figure8: Kernel Density Distributions of each counting features with boxplots.



Again, these counting feature distributions are similar to the exponential distributions in Figure 1 and have the long tails. Long tails are actually driving a significant volume of effective users for recommendation systems. However, since the counting values are too large, distributions will transform log distributions to figure out threshold clearly.

Figure9: Log10 Transformed plots of Figure 9.



Now, the distributions became more ‘normal’ than previous. For total play counts and plays, the reasonable threshold, which is the beginning point of the long tail, are at 30 percent quantile of each metrics. However, for total unique users, 30 percent quantile exclude more than half of the artists. So, I set the threshold as 50 users to genuinely extract non-popular artists. After applying all thresholds, I could obtain 61,340 users and 3,647 artists which are enough for building recommendation systems.

Statistical Inference

To build a hybrid recommender system, I tried to predict artists' genres based on their audio features: acousticness, danceability, energy, and so on and conducted t-test to see how genres might have different audio features.

Assumption was "One artist's song features and the other's song features are different from each other." However, when I conducted the t-test for A$AP Rocky and Linkin Park's audio features, the p-value was around .9857 with t-statistic = 0.01812. Hence, the result fails to reject the null hypothesis even though A$AP Rocky and Linkin Park are two popular artists in different genres (Hip Hop and Rock). Ultimately, the most genres were not distinguishable based solely on their audio features.

A/B test will be a proper statistical inference method to improve the recommender systems and to evaluate which recommendation model performs the best since recommender systems are hard to evaluated by scores. It is ambiguous how users will respond to my recommendation systems.

Modeling

In this project, I implemented different approaches by using different data. Collaborative Filtering Recommendation System has a cold start problem, which occurs when a new user or item enters to the system, because the recommender has lack of user's preference or item information.

To overcome a cold-start problem, I used a Content Based Recommendation because it does not require any new information to figure out the user's preference and other similar items will be recommended. For instance, if a user listens rock or metal genre of music, then the recommender might recommend the user rock or metal genre because rock or metal music has similar audio features. The general idea behind this recommender system is if a user liked a particular item, then he or she will also like an item that is similar to it. However, a content based recommendation is usually non-personalized and a lack of novelty because the user will always receive the same items similar to user's favorite items.

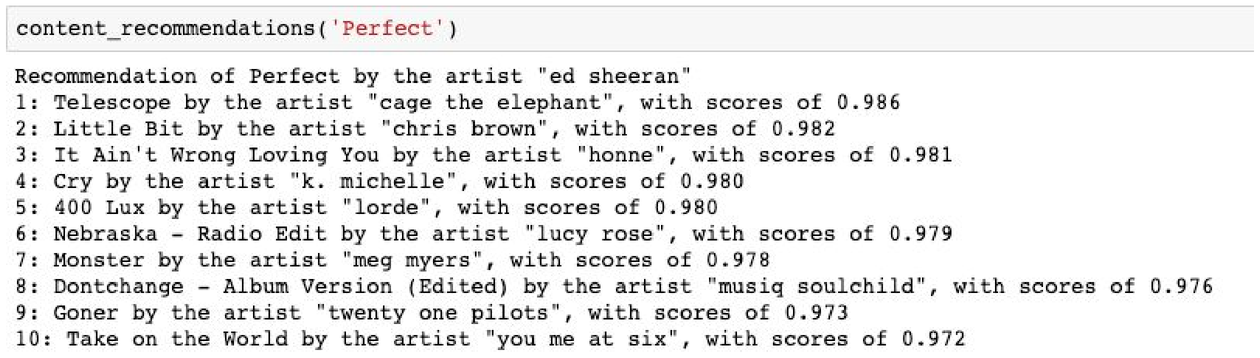
Therefore, to build a personalized recommendation system, a collaborative filtering recommendation is more robust than a content based recommendation.

Content Based Recommendation by Spotify Track Audio Features

I will use a standardized scale of Spotify's audio features such as acousticness, danceability, energy, instrumentalness, liveness, loudness, speechiness, tempo, and valence to calculate similarity of tracks with the function, $similarity = cos(\theta) = \frac{u \cdot v}{|u||v|}$, which is a dot product of normalized vector u and normalized vector v.

Then, based on cosine value, which is between (0,1), I sorted out the similarity scores in decreasing order to obtain most similar tracks of given track's features.

Figure10: Content Based Recommendation for the track, Perfect.



```
content_recommendations('Perfect')
```

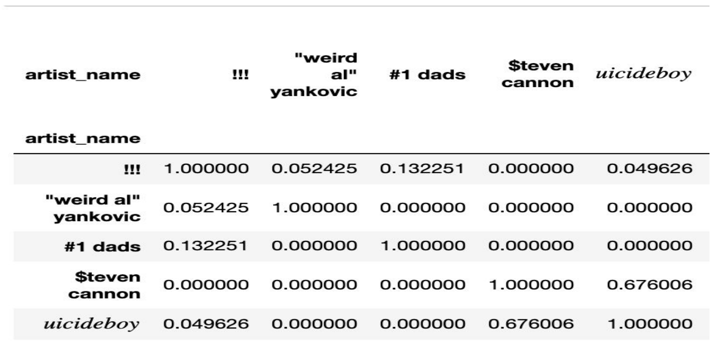
```
Recommendation of Perfect by the artist "ed sheeran"
1: Telescope by the artist "cage the elephant", with scores of 0.986
2: Little Bit by the artist "chris brown", with scores of 0.982
3: It Ain't Wrong Loving You by the artist "honne", with scores of 0.981
4: Cry by the artist "k. michelle", with scores of 0.980
5: 400 Lux by the artist "lorde", with scores of 0.980
6: Nebraska - Radio Edit by the artist "lucy rose", with scores of 0.979
7: Monster by the artist "meg myers", with scores of 0.978
8: Dontchange - Album Version (Edited) by the artist "musiq soulchild", with scores of 0.976
9: Goner by the artist "twenty one pilots", with scores of 0.973
10: Take on the World by the artist "you me at six", with scores of 0.972
```

Content Based Recommendation by Artists in LastFM with Genres from Spotify

Now, I apply Tf Idf-Vectorizer, i.e. Term Frequency- Inverse Document Frequency, and CountVectorizer from sklearn.feature_extraction on Artists' Genres Metadata to build other content recommendation system. In order to implement the system with genres metadata, I need to vectorize genres to detect which artists are correlated by genres and build a matrix where each column represents a genre and each row represents an artist. Then, cosine similarity or other metric of similarity can be calculated via the function above.

Difference between Tf Idf-Vectorizer and CountVectorizer is that Tf-Idf tends to reduce the importance of frequent words in the document. In the project, there are about 1843 unique subcategorized genres and Tf-Idf can weigh more to the adjacent words. For instance, Anita Wilson's genres are contemporary gospel, gospel and gospel r&b then Tf-Idf will put light weight to gospel and try to emphasize adjacent words. While CountVectorizer is a simple frequency counter for each genre and gives the same weight to all words in the document. Then, I again calculate cosine similarity to measure genre's similarity by Tf-Idf Vectorizer and CountVectorizer. Lastly, I used fuzz ratio since some artists' names are ambiguous or commonly misspelled.

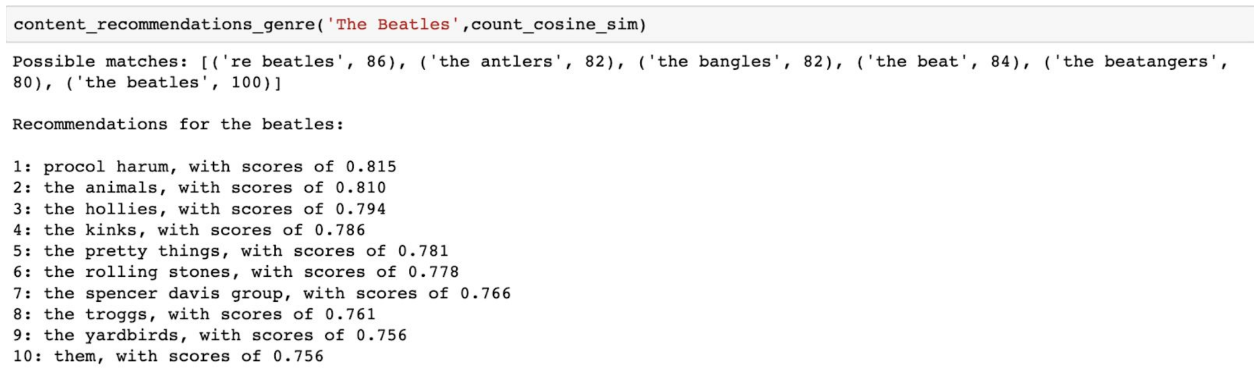
Figure11: Cosine Similarity Score of Tf-Idf Vectorizer



artist_name	!!!	"weird al" yankovic	#1 dads	$steven cannon	*uicideboy*
artist_name					
!!!	1.000000	0.052425	0.132251	0.000000	0.049626
"weird al" yankovic	0.052425	1.000000	0.000000	0.000000	0.000000
#1 dads	0.132251	0.000000	1.000000	0.000000	0.000000
$steven cannon	0.000000	0.000000	0.000000	1.000000	0.676006
uicideboy	0.049626	0.000000	0.000000	0.676006	1.000000

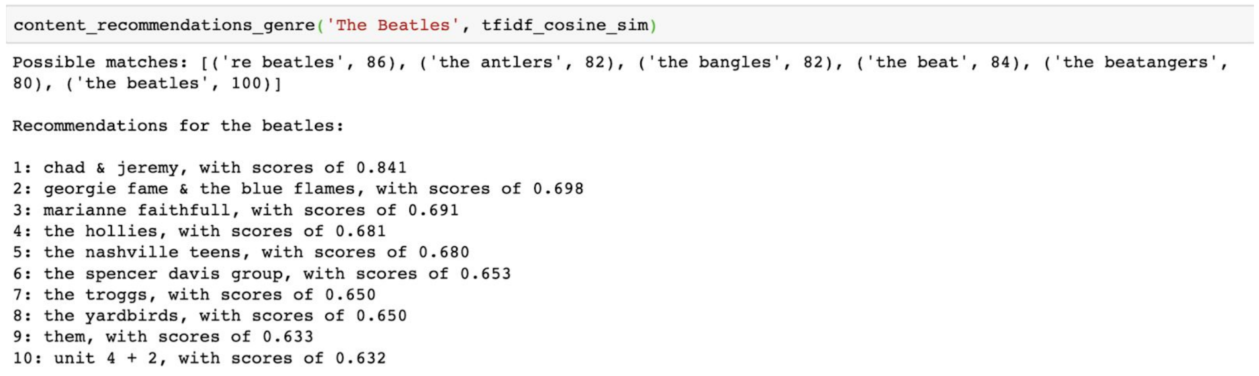
Figure12: Content Recommendation Based on Genres with FuzzRatio

Compared to other website's similar artists for The Beatles, these recommended artists are similar.



```
content_recommendations_genre('The Beatles',count_cosine_sim)
```

```
Possible matches: [('re beatles', 86), ('the antlers', 82), ('the bangles', 82), ('the beat', 84), ('the beatangers',
80), ('the beatles', 100)]

Recommendations for the beatles:

1: procol harum, with scores of 0.815
2: the animals, with scores of 0.810
3: the hollies, with scores of 0.794
4: the kinks, with scores of 0.786
5: the pretty things, with scores of 0.781
6: the rolling stones, with scores of 0.778
7: the spencer davis group, with scores of 0.766
8: the troggs, with scores of 0.761
9: the yardbirds, with scores of 0.756
10: them, with scores of 0.756
```



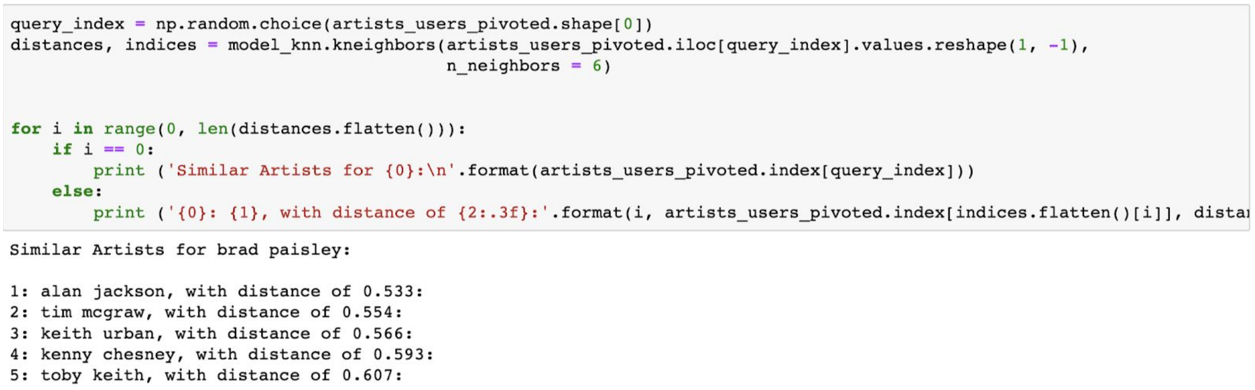
```
content_recommendations_genre('The Beatles', tfidf_cosine_sim)
```

```
Possible matches: [('re beatles', 86), ('the antlers', 82), ('the bangles', 82), ('the beat', 84), ('the beatangers',
80), ('the beatles', 100)]

Recommendations for the beatles:

1: chad & jeremy, with scores of 0.841
2: georgie fame & the blue flames, with scores of 0.698
3: marianne faithfull, with scores of 0.691
4: the hollies, with scores of 0.681
5: the nashville teens, with scores of 0.680
6: the spencer davis group, with scores of 0.653
7: the troggs, with scores of 0.650
8: the yardbirds, with scores of 0.650
9: them, with scores of 0.633
10: unit 4 + 2, with scores of 0.632
```

Collaborative Filtering Recommendation Artists Based with KNN

K-Nearest Neighbors is an algorithm in supervised learning to find the distance between a given instance and all the other artists in the data by selecting K number of examples closest to the query. Setting K = 5 will generate top 5 similar artists of given instance. To apply KNN to the data, the data has to be pivoted where artists represent as rows and users represent as columns with setting missing value as 0. Then fit the model with metric = cosine to measure similarity between artist vectors. However, because the recommender system is based on the artists with KNN on the actual play count data, outliers can influence the distance metrics.

Figure13: Collaborative Filtering Recommendation with KNN with Random Query



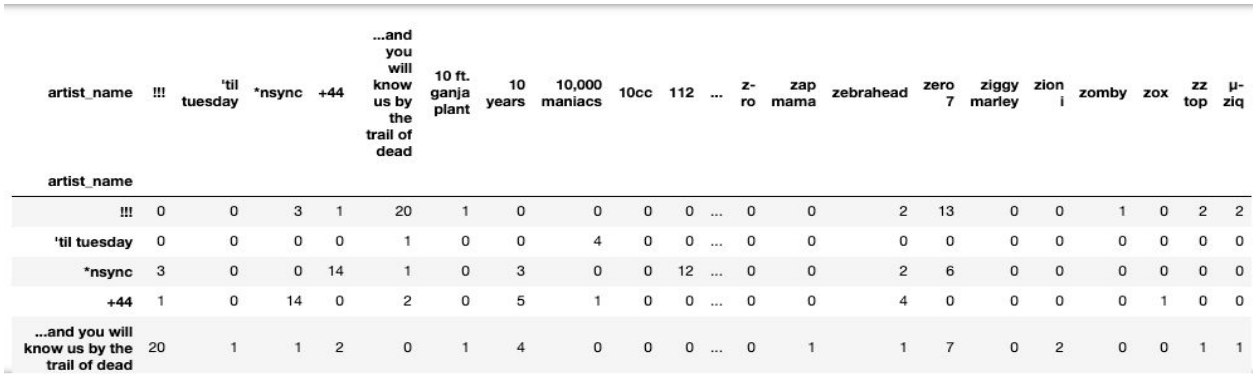
```
query_index = np.random.choice(artists_users_pivoted.shape[0])
distances, indices = model_knn.kneighbors(artists_users_pivoted.iloc[query_index].values.reshape(1, -1),
                                          n_neighbors = 6)


for i in range(0, len(distances.flatten())):
    if i == 0:
        print ('Similar Artists for {0}:\n'.format(artists_users_pivoted.index[query_index]))
    else:
        print ('{0}: {1}, with distance of {2:.3f}:'.format(i, artists_users_pivoted.index[indices.flatten()[i]], distan
```

```
Similar Artists for brad paisley:

1: alan jackson, with distance of 0.533:
2: tim mcgraw, with distance of 0.554:
3: keith urban, with distance of 0.566:
4: kenny chesney, with distance of 0.593:
5: toby keith, with distance of 0.607:
```

Collaborative Filtering Recommendation with Co-occurrence Matrix

Another way of building artist-based recommender is to define a co-occurrence matrix based on played artists. As above, KNN algorithm was based on the actual play count data, which can easily be influenced by outliers, so I binarized the play count data to indicate either a user listens an artist or not. To define the co-occurrence matrix, I need to transform users and artists intersection data into an artist by artist matrix and any particular element of this matrix will be the number of times its column's artist appeared together with the artists contained by its row's user. Then, I transposed the users and artists intersection data and calculate the dot product of transposed matrix and the original matrix and set all diagonal values 0 since I didn't care about how many times artists appear with themselves.

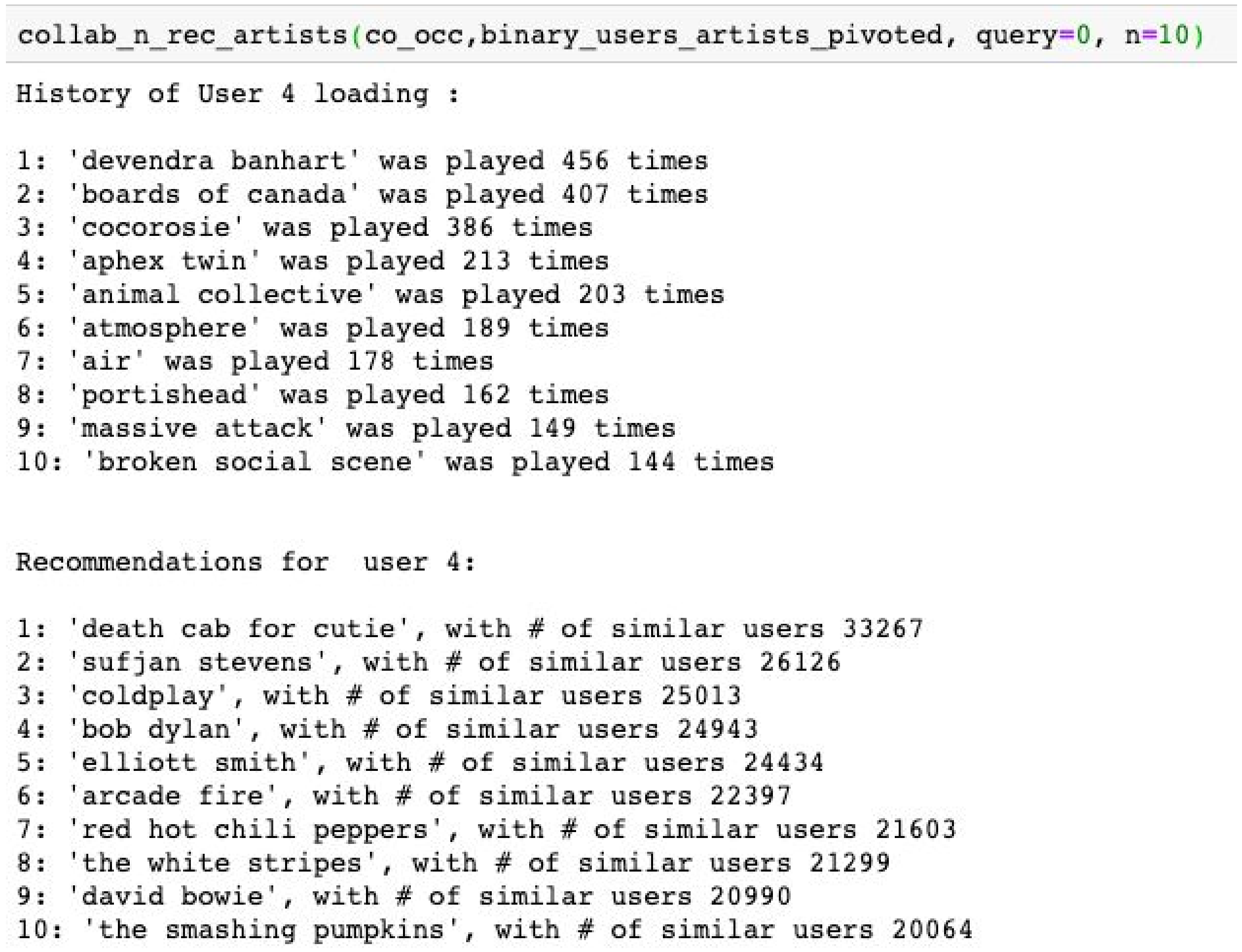
Figure14: Artists by Artists Co-Occurrence Matrix



artist_name	!!!	'til tuesday	*nsync	+44	...and you will know us by the trail of dead	10 ft. ganja plant	10 years	10,000 maniacs	10cc	112	...	z-ro	zap mama	zebrahead	zero 7	ziggy marley	zion i	zomby	zox	zz top	µ-ziq
artist_name																					
!!!	0	0	3	1	20	1	0	0	0	0	...	0	0	2	13	0	0	1	0	2	2
'til tuesday	0	0	0	0	1	0	0	4	0	0	...	0	0	0	0	0	0	0	0	0	0
***nsync**	3	0	0	14	1	0	3	0	0	12	...	0	0	2	6	0	0	0	0	0	0
+44	1	0	14	0	2	0	5	1	0	0	...	0	0	4	0	0	0	0	1	0	0
...and you will know us by the trail of dead	20	1	1	2	0	1	4	0	0	0	...	0	1	1	7	0	2	0	0	1	1

For example, '+44' appeared 1 time with '!!!' contained by first user's preference vector and 'zero 7' appeared 13 times with '!!!'. Then, I calculated dot product of the co-occurrence matrix and the user's artist preference vector to get a recommended vector and weighted numbers obtained in the vector would be how many times artists appeared together with the user's artists. Lastly, I set to zero all artists already played by the user; artists with high weighted values should be recommended.

Figure15: Collaborative Filtering Recommendation with co-occurrence matrix.



```
collab_n_rec_artists(co_occ,binary_users_artists_pivoted, query=0, n=10)
```

```
History of User 4 loading :

1: 'devendra banhart' was played 456 times
2: 'boards of canada' was played 407 times
3: 'cocorosie' was played 386 times
4: 'aphex twin' was played 213 times
5: 'animal collective' was played 203 times
6: 'atmosphere' was played 189 times
7: 'air' was played 178 times
8: 'portishead' was played 162 times
9: 'massive attack' was played 149 times
10: 'broken social scene' was played 144 times


Recommendations for  user 4:

1: 'death cab for cutie', with # of similar users 33267
2: 'sufjan stevens', with # of similar users 26126
3: 'coldplay', with # of similar users 25013
4: 'bob dylan', with # of similar users 24943
5: 'elliott smith', with # of similar users 24434
6: 'arcade fire', with # of similar users 22397
7: 'red hot chili peppers', with # of similar users 21603
8: 'the white stripes', with # of similar users 21299
9: 'david bowie', with # of similar users 20990
10: 'the smashing pumpkins', with # of similar users 20064
```

Train-Test Split

Typically, in usual machine learning applications, I need to test whether the model trained is good or bad on new data that has not been seen previously. Usually, to split train and test sets, I take random samples of the data and randomly picks a certain percentage of elements. However, this train-test split won't work for a recommendation system because I need all of the binarized users and items intersections to find the proper matrix factorization. Therefore, the better method is hiding 20% of the users and artists intersection chosen at random by setting 0 and set as a train set. During the test phase, I check how many recommend artists that actually were played in the end by the user during the test by calculating AUC score. The other better method will be using built-in train-test split function in Light FM data. Light FM train and test set do not have common intersections of users and artists, so the train and test sets are independent of each other.

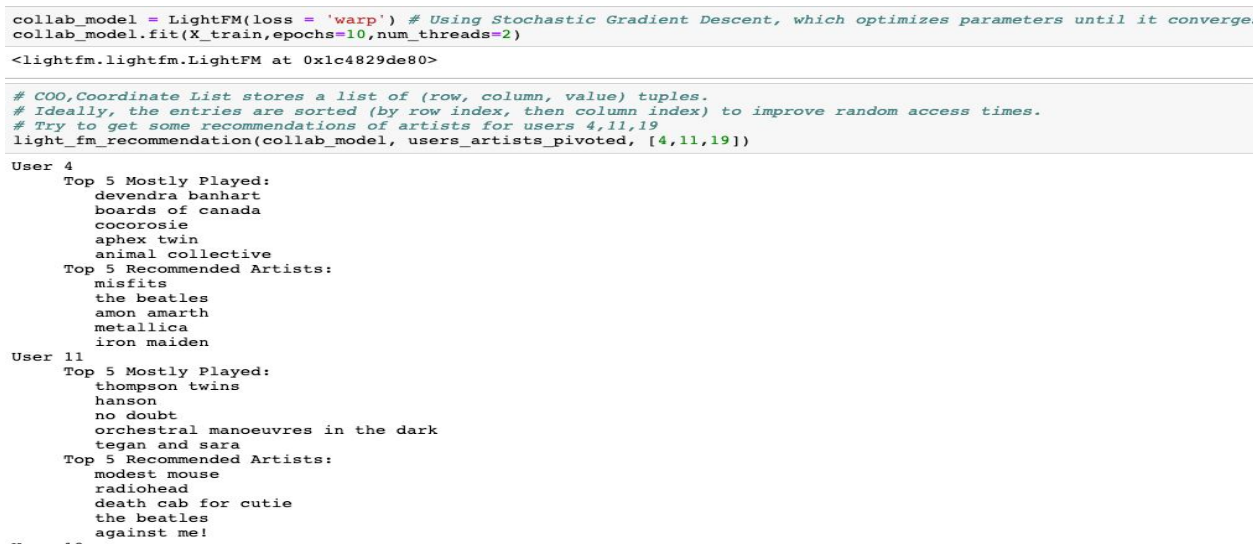
Collaborative Filtering Recommendation with Light FM

There are many libraries for recommendation systems in Python and Light FM is a well-known library to build a hybrid recommendation with implicit data. Light FM model learns embeddings, hidden representations in a high-dimensional space, for users and items in a way that encodes user preferences over items. Multiply estimated latent vectors for users and item metadata to produce scores for every item for a given user; items scored highly are more likely to be interesting to the user.

Moreover, since the data is implicit and consists of (user,positive artists, negative artists) triplets, I set the loss function as weighted approximate rank pairwise and WARP will weights the gradient updates using the estimated rank of artists for users. WARP will define loss as rank.The model's parameters will be optimized until they converge by forcing loss function to convex loss function. WARP considers each user and artist intersection in turn, choosing a negative artist at random from all other artists and compute predictions for both artists. If the negative artists exceeds that of artists with some margin, i.e. nonzero indicators, then SGD updates to rank positive artists higher and negative artists lower. Lastly, if no violation is found, then sampling negative artists is continued until it finds a violation. As a result, the model's prediction will produce recommendation scores for all artists.

AUC score for this light fm model measures the quality of the overall ranking. In the binary case, it can be interpreted as the probability that a randomly chosen positive item is ranked higher than a randomly chosen negative item. If auc score is close to 1, it will indicate the model is ordering positive artists better than the basic model, randomly ordering positive artists.

Figure16: LightFM Recommendation Only Using User-Artist Intersection Data



```
collab_model = LightFM(loss = 'warp') # Using Stochastic Gradient Descent, which optimizes parameters until it converge
collab_model.fit(X_train,epochs=10,num_threads=2)
```

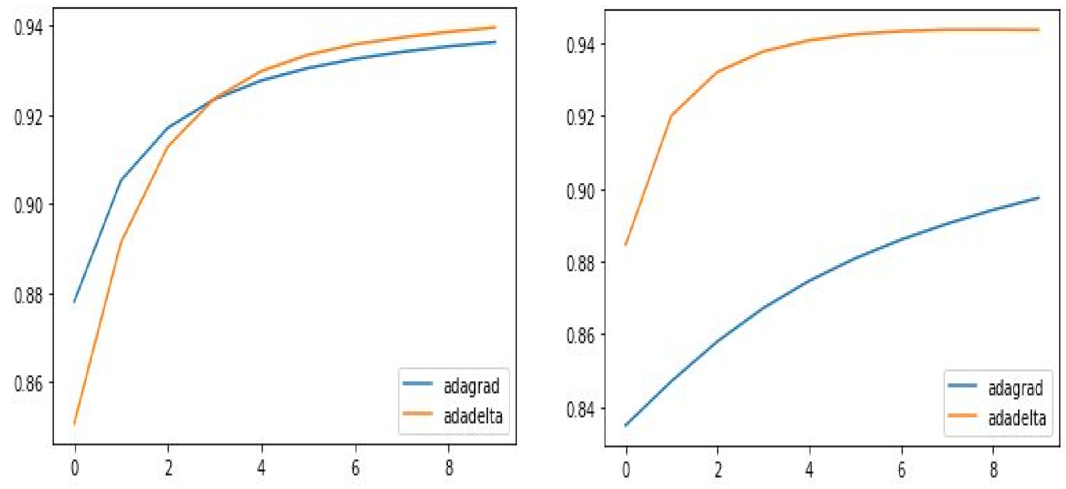
```
<lightfm.lightfm.LightFM at 0x1c4829de80>
```

```
# COO,Coordinate List stores a list of (row, column, value) tuples.
# Ideally, the entries are sorted (by row index, then column index) to improve random access times.
# Try to get some recommendations of artists for users 4,11,19
light_fm_recommendation(collab_model, users_artists_pivoted, [4,11,19])
```

```
User 4
     Top 5 Mostly Played:
        devendra banhart
        boards of canada
        cocorosie
        aphex twin
        animal collective
     Top 5 Recommended Artists:
        misfits
        the beatles
        amon amarth
        metallica
        iron maiden
User 11
     Top 5 Mostly Played:
        thompson twins
        hanson
        no doubt
        orchestral manoeuvres in the dark
        tegan and sara
     Top 5 Recommended Artists:
        modest mouse
        radiohead
        death cab for cutie
        the beatles
        against me!
```

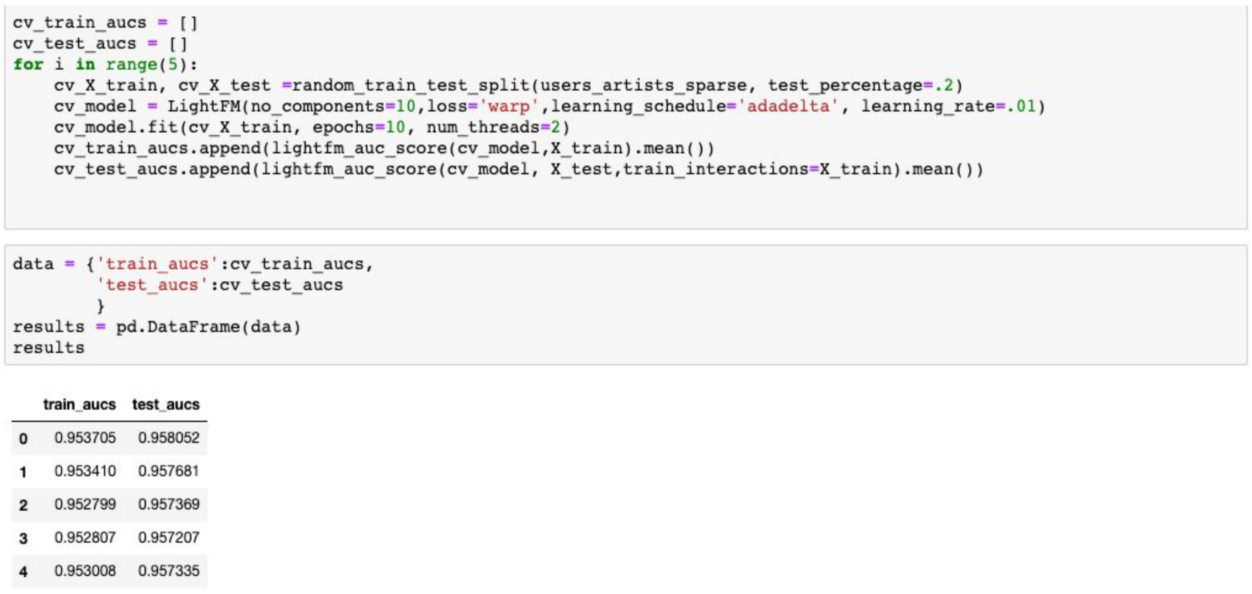
Figure17: Tuning Hyperparameters for LightFM models

Epoch = 10 Epoch = 30



Learning_schedule should be adadelta and number of epochs = 10 and Adadelta works well with high number of hidden factors. Fit_partial will update the coefficient and adadelta is the method dynamically adapts over time using only first order information and has minimal computation.

Figure18: Cross validation for AUC score



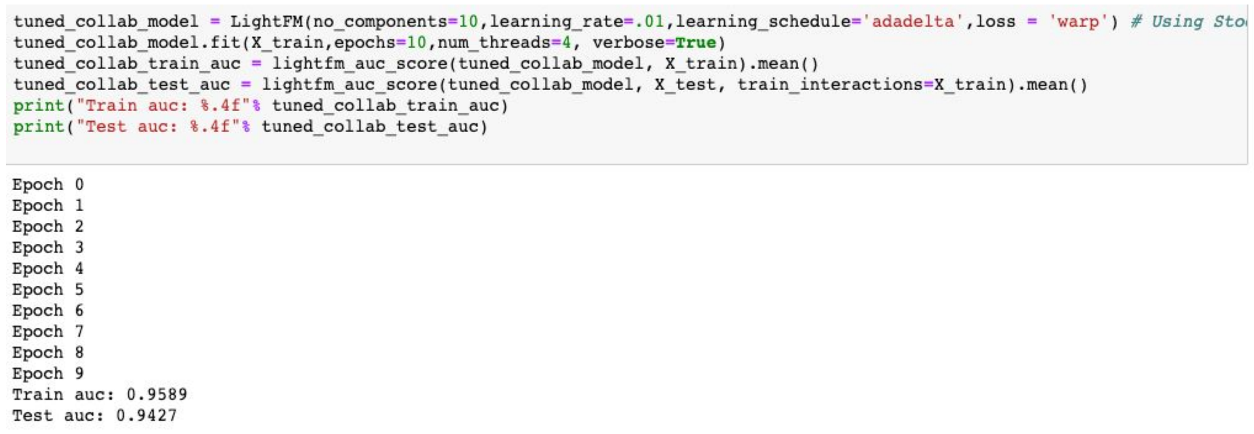
```
cv_train_aucs = []
cv_test_aucs = []
for i in range(5):
    cv_X_train, cv_X_test =random_train_test_split(users_artists_sparse, test_percentage=.2)
    cv_model = LightFM(no_components=10,loss='warp',learning_schedule='adadelta', learning_rate=.01)
    cv_model.fit(cv_X_train, epochs=10, num_threads=2)
    cv_train_aucs.append(lightfm_auc_score(cv_model,X_train).mean())
    cv_test_aucs.append(lightfm_auc_score(cv_model, X_test,train_interactions=X_train).mean())
```

```
data = {'train_aucs':cv_train_aucs,
        'test_aucs':cv_test_aucs
        }
results = pd.DataFrame(data)
results
```

	train_aucs	test_aucs
0	0.953705	0.958052
1	0.953410	0.957681
2	0.952799	0.957369
3	0.952807	0.957207
4	0.953008	0.957335

Since the train-test-split was not usual, I split the train and test sets for five times as a method of cross validation.

Figure18: Evaluation of Tuned Model



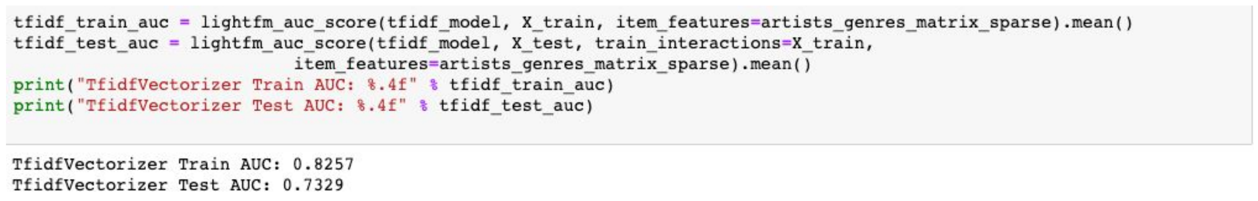
```
tuned_collab_model = LightFM(no_components=10,learning_rate=.01,learning_schedule='adadelta',loss = 'warp') # Using Sto
tuned_collab_model.fit(X_train,epochs=10,num_threads=4, verbose=True)
tuned_collab_train_auc = lightfm_auc_score(tuned_collab_model, X_train).mean()
tuned_collab_test_auc = lightfm_auc_score(tuned_collab_model, X_test, train_interactions=X_train).mean()
print("Train auc: %.4f"% tuned_collab_train_auc)
print("Test auc: %.4f"% tuned_collab_test_auc)
```

```
Epoch 0
Epoch 1
Epoch 2
Epoch 3
Epoch 4
Epoch 5
Epoch 6
Epoch 7
Epoch 8
Epoch 9
Train auc: 0.9589
Test auc: 0.9427
```

Hybrid Recommendation with Light FM

With vectorized artists and genres matrix by Tf-IDF and CountVectorizer, I implement hybrid recommendation via Light FM.

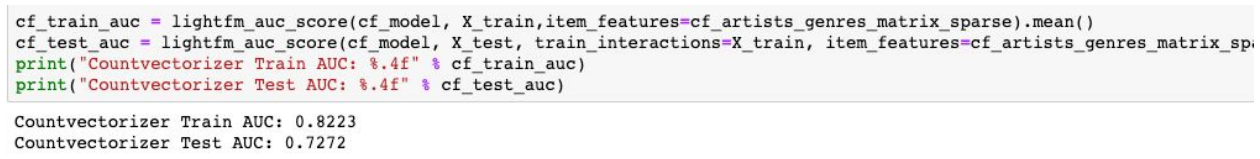
Figure19: AUC Tf-IDF Vectorized Artists and Genres Matrix and Users-Artists Matrix



```
tfidf_train_auc = lightfm_auc_score(tfidf_model, X_train, item_features=artists_genres_matrix_sparse).mean()
tfidf_test_auc = lightfm_auc_score(tfidf_model, X_test, train_interactions=X_train,
                           item_features=artists_genres_matrix_sparse).mean()
print("TfidfVectorizer Train AUC: %.4f" % tfidf_train_auc)
print("TfidfVectorizer Test AUC: %.4f" % tfidf_test_auc)
```

```
TfidfVectorizer Train AUC: 0.8257
TfidfVectorizer Test AUC: 0.7329
```

Figure20: AUC Count Vectorized Artists and Genres Matrix and Users-Artists Matrix

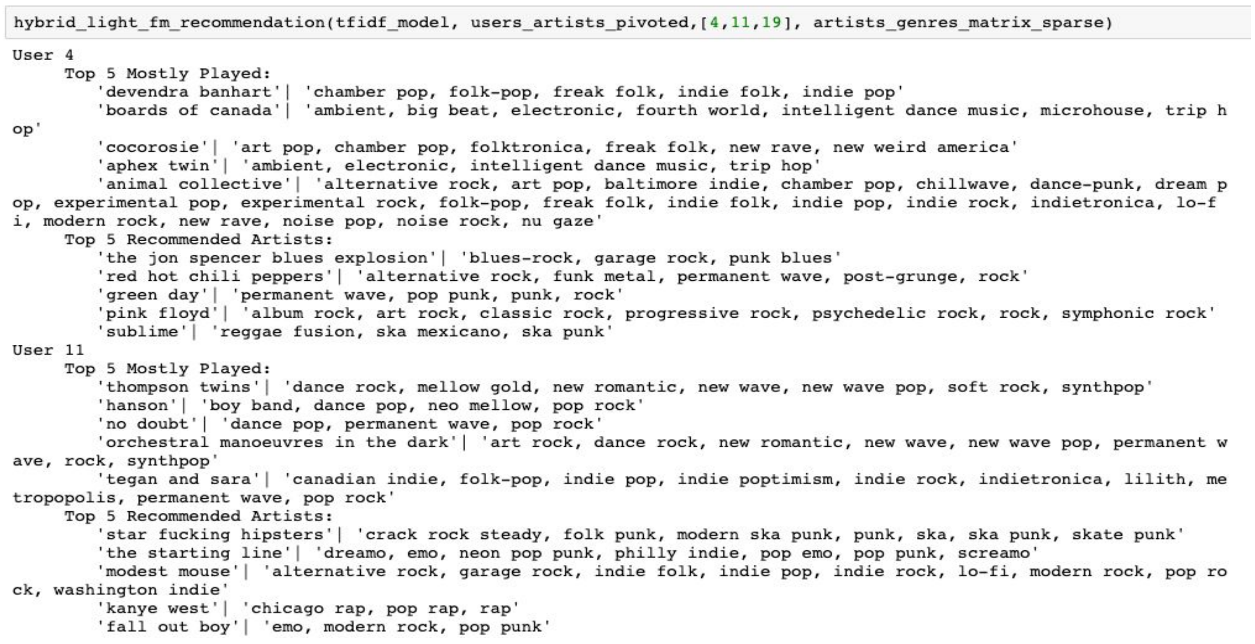


```
cf_train_auc = lightfm_auc_score(cf_model, X_train,item_features=cf_artists_genres_matrix_sparse).mean()
cf_test_auc = lightfm_auc_score(cf_model, X_test, train_interactions=X_train, item_features=cf_artists_genres_matrix_sp
print("Countvectorizer Train AUC: %.4f" % cf_train_auc)
print("Countvectorizer Test AUC: %.4f" % cf_test_auc)
```

```
Countvectorizer Train AUC: 0.8223
Countvectorizer Test AUC: 0.7272
```

The results are pretty good. These recommenders are doing better than randomly recommending artists!

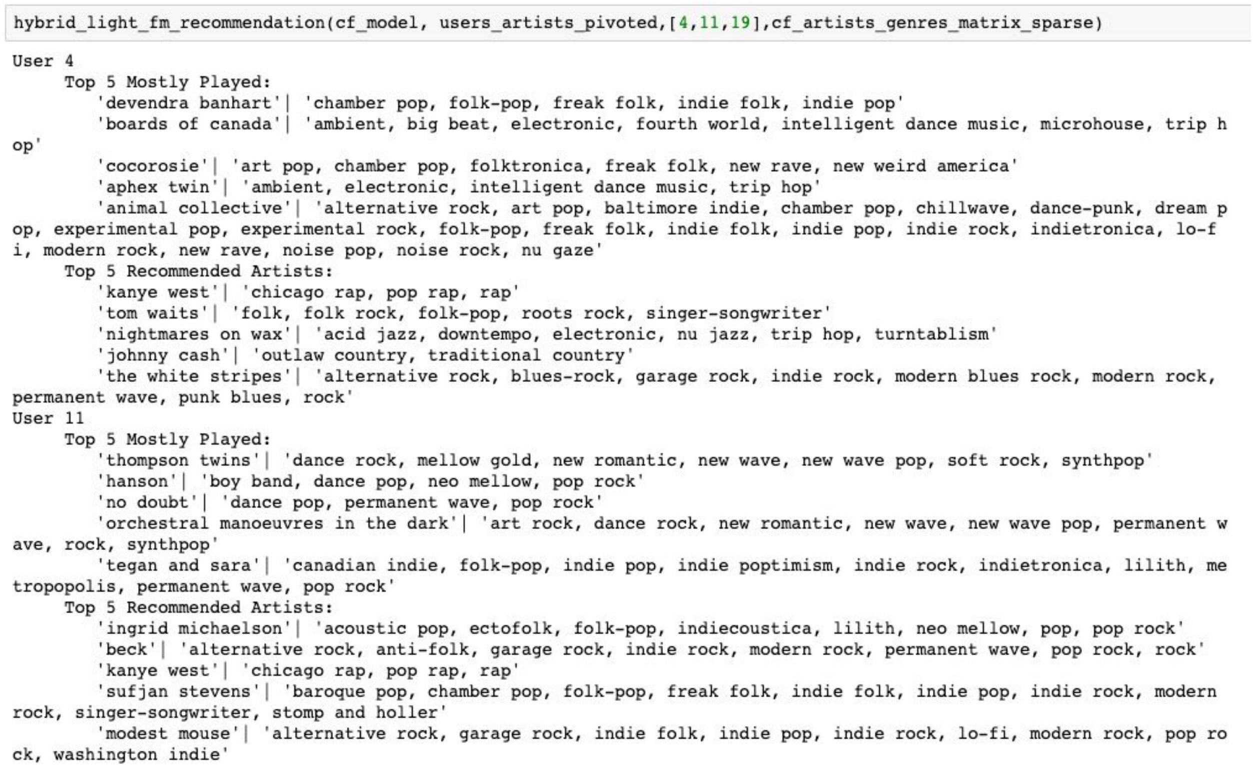
Figure21: Tf-Idf Vectorized and Light FM Recommendation



```
hybrid_light_fm_recommendation(tfidf_model, users_artists_pivoted,[4,11,19], artists_genres_matrix_sparse)
```

```
User 4
     Top 5 Mostly Played:
         'devendra banhart'| 'chamber pop, folk-pop, freak folk, indie folk, indie pop'
         'boards of canada'| 'ambient, big beat, electronic, fourth world, intelligent dance music, microhouse, trip h
op'
         'cocorosie'| 'art pop, chamber pop, folktronica, freak folk, new rave, new weird america'
         'aphex twin'| 'ambient, electronic, intelligent dance music, trip hop'
         'animal collective'| 'alternative rock, art pop, baltimore indie, chamber pop, chillwave, dance-punk, dream p
op, experimental pop, experimental rock, folk-pop, freak folk, indie folk, indie pop, indie rock, indietronica, lo-f
i, modern rock, new rave, noise pop, noise rock, nu gaze'
     Top 5 Recommended Artists:
         'the jon spencer blues explosion'| 'blues-rock, garage rock, punk blues'
         'red hot chili peppers'| 'alternative rock, funk metal, permanent wave, post-grunge, rock'
         'green day'| 'permanent wave, pop punk, punk, rock'
         'pink floyd'| 'album rock, art rock, classic rock, progressive rock, psychedelic rock, rock, symphonic rock'
         'sublime'| 'reggae fusion, ska mexicano, ska punk'
User 11
     Top 5 Mostly Played:
         'thompson twins'| 'dance rock, mellow gold, new romantic, new wave, new wave pop, soft rock, synthpop'
         'hanson'| 'boy band, dance pop, neo mellow, pop rock'
         'no doubt'| 'dance pop, permanent wave, pop rock'
         'orchestral manoeuvres in the dark'| 'art rock, dance rock, new romantic, new wave, new wave pop, permanent w
ave, rock, synthpop'
         'tegan and sara'| 'canadian indie, folk-pop, indie pop, indie poptimism, indie rock, indietronica, lilith, me
tropopolis, permanent wave, pop rock'
     Top 5 Recommended Artists:
         'star fucking hipsters'| 'crack rock steady, folk punk, modern ska punk, punk, ska, ska punk, skate punk'
         'the starting line'| 'dreamo, emo, neon pop punk, philly indie, pop emo, pop punk, screamo'
         'modest mouse'| 'alternative rock, garage rock, indie folk, indie pop, indie rock, lo-fi, modern rock, pop ro
ck, washington indie'
         'kanye west'| 'chicago rap, pop rap, rap'
         'fall out boy'| 'emo, modern rock, pop punk'
```

Figure22: CountVectorized and Light FM Recommendation



```
hybrid_light_fm_recommendation(cf_model, users_artists_pivoted,[4,11,19],cf_artists_genres_matrix_sparse)
```

```
User 4
     Top 5 Mostly Played:
         'devendra banhart'| 'chamber pop, folk-pop, freak folk, indie folk, indie pop'
         'boards of canada'| 'ambient, big beat, electronic, fourth world, intelligent dance music, microhouse, trip h
op'
         'cocorosie'| 'art pop, chamber pop, folktronica, freak folk, new rave, new weird america'
         'aphex twin'| 'ambient, electronic, intelligent dance music, trip hop'
         'animal collective'| 'alternative rock, art pop, baltimore indie, chamber pop, chillwave, dance-punk, dream p
op, experimental pop, experimental rock, folk-pop, freak folk, indie folk, indie pop, indie rock, indietronica, lo-f
i, modern rock, new rave, noise pop, noise rock, nu gaze'
     Top 5 Recommended Artists:
         'kanye west'| 'chicago rap, pop rap, rap'
         'tom waits'| 'folk, folk rock, folk-pop, roots rock, singer-songwriter'
         'nightmares on wax'| 'acid jazz, downtempo, electronic, nu jazz, trip hop, turntablism'
         'johnny cash'| 'outlaw country, traditional country'
         'the white stripes'| 'alternative rock, blues-rock, garage rock, indie rock, modern blues rock, modern rock,
permanent wave, punk blues, rock'
User 11
     Top 5 Mostly Played:
         'thompson twins'| 'dance rock, mellow gold, new romantic, new wave, new wave pop, soft rock, synthpop'
         'hanson'| 'boy band, dance pop, neo mellow, pop rock'
         'no doubt'| 'dance pop, permanent wave, pop rock'
         'orchestral manoeuvres in the dark'| 'art rock, dance rock, new romantic, new wave, new wave pop, permanent w
ave, rock, synthpop'
         'tegan and sara'| 'canadian indie, folk-pop, indie pop, indie poptimism, indie rock, indietronica, lilith, me
tropopolis, permanent wave, pop rock'
     Top 5 Recommended Artists:
         'ingrid michaelson'| 'acoustic pop, ectofolk, folk-pop, indiecoustica, lilith, neo mellow, pop, pop rock'
         'beck'| 'alternative rock, anti-folk, garage rock, indie rock, modern rock, permanent wave, pop rock, rock'
         'kanye west'| 'chicago rap, pop rap, rap'
         'sufjan stevens'| 'baroque pop, chamber pop, folk-pop, freak folk, indie folk, indie pop, indie rock, modern
rock, singer-songwriter, stomp and holler'
         'modest mouse'| 'alternative rock, garage rock, indie folk, indie pop, indie rock, lo-fi, modern rock, pop ro
ck, washington indie'
```

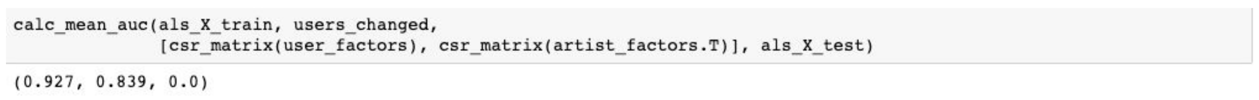
Hybrid Recommendation with Implicit Alternating Least Squares

Collaborative Based Filtering Recommender does not require any information about users or items. To figure out how users and items are related to each other, I can use Matrix Factorization, which is very similar to Singular Value Decomposition. Applying matrix factorization on user and item intersection matrix, R (m users and n items), will produce two separate matrices, U (m users and k hidden factors) and V (k hidden factors and n items). Multiplying these two feature matrices will approximate the original user and item intersection matrix.

Implicit Alternating Least Squares will randomly initialize U feature matrix and solve for V; then, the solution matrix V will solve for U. Keep iterating back and forth these procedures until U and V converge that approximate the original matrix as best as I can.

More information can be found here.

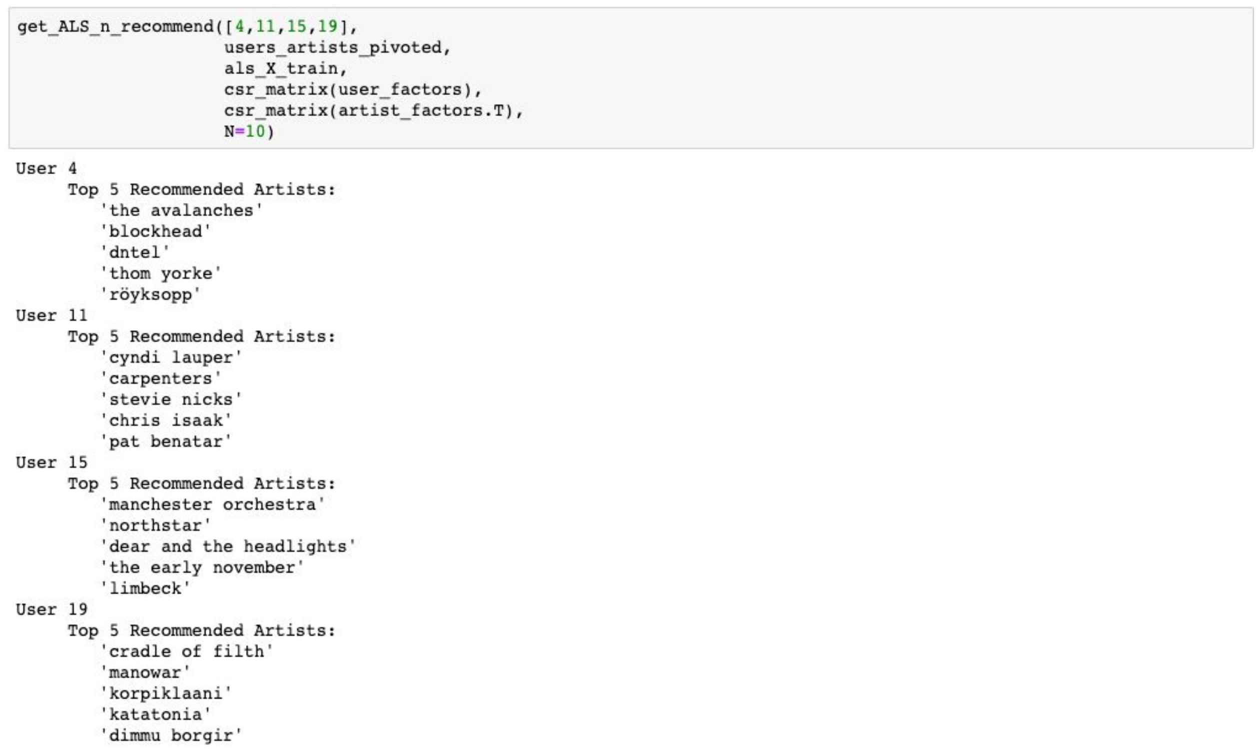
Figure23: AUC score for Implicit Alternating Least Squares and RMSE



```
calc_mean_auc(als_X_train, users_changed,
              [csr_matrix(user_factors), csr_matrix(artist_factors.T)], als_X_test)
(0.927, 0.839, 0.0)
```

AUC scores are pretty decent and RMSE is 0.

Figure24: Implicit Alternating Least Squares Recommendation



```
get_ALS_n_recommend([4,11,15,19],
                    users_artists_pivoted,
                    als_X_train,
                    csr_matrix(user_factors),
                    csr_matrix(artist_factors.T),
                    N=10)
User 4
     Top 5 Recommended Artists:
        'the avalanches'
        'blockhead'
        'dntel'
        'thom yorke'
        'röyksopp'
User 11
     Top 5 Recommended Artists:
        'cyndi lauper'
        'carpenters'
        'stevie nicks'
        'chris isaak'
        'pat benatar'
User 15
     Top 5 Recommended Artists:
        'manchester orchestra'
        'northstar'
        'dear and the headlights'
        'the early november'
        'limbeck'
User 19
     Top 5 Recommended Artists:
        'cradle of filth'
        'manowar'
        'korpiklaani'
        'katatonia'
        'dimmu borgir'
```

Future Work

- Implementing track recommendation via artist recommendation
- Building Recommendation with Convolutional Neural Network
- A/B Test for each recommender systems via online users to evaluate models.