Music Recommendation System

Aditya Maheshwari (am2971), Animesh Sharma (as3592), Yaniv Bronshtein (yb262), Fan Shen (fs470) , Raju Datla (vkd20), Toshitt Ahuja (ta498), Vipul Gharde (vig4), Wanying Mo (wm318)

Abstract

This study evaluates the application of a supervised Fully Connected Feedforward Deep Neural Network and unsupervised learning models like the Matrix Factorization based Singular Value Decomposition to the task of generating both artist and song recommendations to both new and existing users to the task of creating a robust Music Recommendation System. It was found that the neural network was not

Introduction

With the rise of streaming platforms, the amount of music available to consumers at their fingertips or using their favorite AI(Google, Alexa, Siri) has skyrocketed. At first glance, users seemingly benefit from this plethora of available music. The opposite is true or was valid without the help of robust, state-of-the-art recommender systems. Recommender systems thus face a powerful dilemma: on the one hand, they must observe user preferences and cater albums or individual songs to those preferences. On the other hand, they must be careful to introduce a degree of novelty so that struggling artists can have exposure.

Based on the article *Collaborative Filtering in Recommender Systems* by Schafer, et. al, the technique is used to help find new items a consumer might like, advise on a particular item(good/bad), and help a consumer find other consumer(s) such as with dating apps.

In our task of building a music recommendation system, we explored collaborative filtering techniques such as unsupervised Centered K Nearest Neighbors and Singular Value Decomposition using the Surprise Scikit library in Python. In addition, we attempted using a fully connected Feedforward network using Google's Tensorflow 2 framework to see if the task could be converted to a supervised task.

Data Collection

In our project, we used two Kaggle datasets. The first, *Spotify Features*, found at:

https://www.kaggle.com/iqbalbasyar/spotify-genre-classification/data?select=SpotifyFeatures.csv

contained around 232.7K observations and 18 features.[2] The key features were genre, artist_name, track_name, popularity, and various acoustic features. Originally, the dataset was used to predict the genre of a song. As a target variable, genre was too broad for a catered recommendation system. As such we sought out a larger dataset that would allow us to use genre, artist, track(song), and playlist as the features. The target variable would have to be engineered. Our second dataset found at

https://www.kaggle.com/andrewmvd/spotify-playlists

contained around 12.9M observations and only 4 features, namely user_id, the artist, the track(song), and the playlist. Our new dilemma was that the genre feature was missing.

To create the ultimate dataset, our first approach was to create the artist-track feature in both datasets by concatenating the artist and track features, lowercasing, removing all whitespaces, and then performing a fuzzy join using the Levenshtein distances between the strings. The algorithm procedure ran in polynomial time so we abandoned the approach. As such, we made the executive decision to perform a separate ML task on each dataset. We ultimately performed supervised Deep Learning on the former dataset and unsupervised collaborative filtering on the latter dataset.

Although we ran out of time, we had collected over 80 observations from random users using a survey hosted on Google Form. The first question asked a responder to select their top three genres from a list of the top 10 genres. In the survey, we included only 10 but the dataset contained 27 distinct genres. The second question was a follow up that asked the responder to select their top 5 artists based on the three genres selected. We recognize the limited capabilities of the survey and the flaws in the survey that randomly selected 10 artists from each of the 10 genres. We received a complaint from responders that the majority of the artists were unfamiliar to them, leading them to choose artists they recognized rather than ones based on their preferences. **Figure 1a** shows that as expected, Pop and Hip-Hop were tied as the two most popular genres and **Figure 1b** shows that Avril Lavigne and Drake were the most popular artists and also most popular from the respective genres.

Experiments

For the first Deep Learning task, we utilized Google's Tensorflow 2 framework to create the Feedforward network as shown in **Figure 3.** Meanwhile, **Figure 2a** and **Figure 2b** show the process undertaken to create the input matrix fed into the tensorflow model. Ultimately, we observed that the sparse matrix was causing major issues leading to our model seriously underperforming as illustrated in **Figure 4.** We attempted to alleviate the issue by experimenting with Dropout at various layers of the model, and Laplace Smoothing on the input matrix to deal with sparsity. Unfortunately, multiple executions of the algorithm often led to even worse results. The model in **Figure 4** lacks both dropout and Laplace Smoothing.

For the second collaborative filtering task, we performed data cleaning on the artist feature to minimize the number of classes. Particularly, we used the base python, re, nltk, and Unidecode libraries to remove featured artists, punctuation, bad characters and text after the ampersand, convert non-English alphabets to English alphabets, remove whitespace, and lowercase. We show some sample transformations in **Figure 5** that lead to the elimination of over 30000 classes.

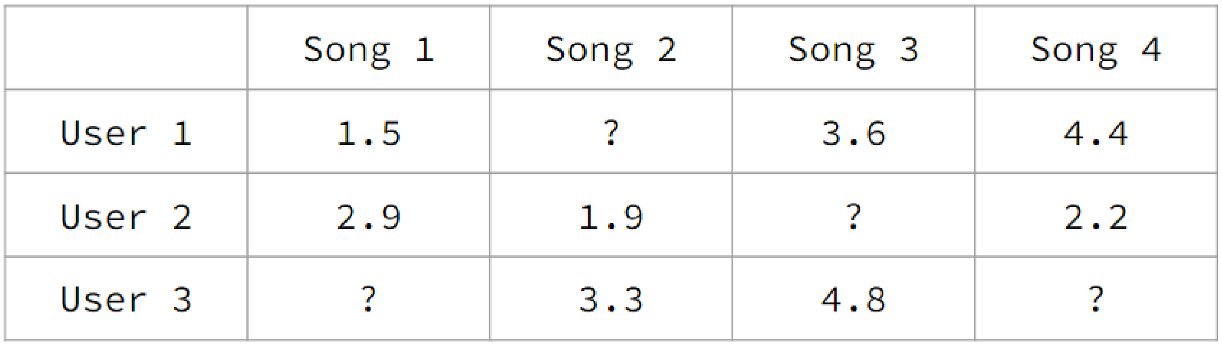
Since the results from the supervised method (Neural Nets) were less than ideal, we decided to pivot to unsupervised learning. After doing some research on unsupervised algorithms for recommendation systems we decided to settle on Matrix Factorization. Matrix factorization algorithms work by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices.

The Singular Value Decomposition algorithm

For understanding the concept of SVD algorithm, we first need to look at what composes matrix representations, and their factorization.

The matrix representation:

There are a lot of ways a person can think of recommending a songs to someone. One strategy that turned out to be very good is treating movie ratings as a Users x Songs matrix like this:



	Song 1	Song 2	Song 3	Song 4
User 1	1.5	?	3.6	4.4
User 2	2.9	1.9	?	2.2
User 3	?	3.3	4.8	?

Figure 6: User - Song interaction matrix

In that matrix, the question marks represent the songs a user has not rated. The strategy then is to predict those ratings somehow and recommend to users the songs they will probably like.

Matrix Factorization:

Users always follow a logic for rating or listening frequently to a particular song. It is usually assumed that the more hip-hop or pop the artist is, the more likely users are to listen to that song. That process can be represented by a linear formula of the following kind:

$$(\theta_u)^T \times x_m$$

where x   is a column vector with the values of the artists of the song m and θ_u is another column vector with the weights that user u gives to each feature. Each user has a different set of weights and each song has a different set of values for its features.

Singular Value Decomposition (SVD):

Enter Singular Value Decomposition (SVD). SVD is a fancy way to factorizing a matrix into three other matrices ($A = U\Sigma V^T$). The way SVD is done guarantees those 3 matrices carry some nice mathematical properties.

According to Luis Argerich:

*"The matrix factorization algorithms used for recommender systems try to find two matrices: P,Q such as P*Q matches the KNOWN values of the utility matrix"*

This principle appeared in the famous SVD++, *"Factorization meets the neighborhood" paper that unfortunately used the name "SVD++" for an algorithm that has absolutely no relationship with the SVD"*

Let's start by pointing out that the method usually referred to as 'SVD' that is used in the context of recommendations is not strictly speaking the mathematical Singular Value Decomposition of a matrix but rather an approximate way to compute the low-rank approximation of the matrix by minimizing the squared error loss. *A more accurate, albeit more generic, way to call this would be Matrix Factorization.* (Xavier Amatriain)

To summarize:

1. SVD is a somewhat complex mathematical technique that factorizes matrices into three new matrices and has many applications, including Principal Component Analysis.
2. Factorizing a matrix into two other ones and using gradient descent to find optimal values of features and weights.
3. The synonym for this is Matrix Factorization.
4. Because of the good results and the fame that followed, people still call that technique SVD.

Challenges

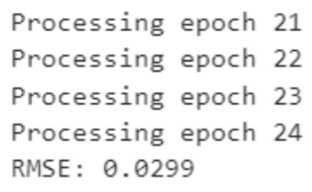
The dataset we used for unsupervised learning had 12.9 million rows and 4 columns. We had to further engineer some features (like frequency based ratings) that made the size of the dataset even larger. The problem with working on huge datasets are manifold, including but not limited to:

- Enormous computational complexity and huge memory requirements
- Difficulty in detecting and weeding out low quality samples
- Difficulty in understanding the quirks of the dataset to effectively preprocess it.

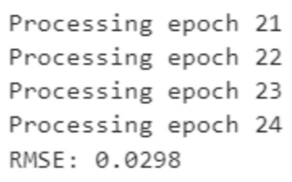
With the limited time we had , we could only focus on certain aspects of the problems described above. While we would’ve liked to try other unsupervised learning methods like centered KNN or to tune the hyperparameters on the algorithms we used to train our model, due to resource and time limitations, we had to pick our battles.

Results

- **Epochs:** 25
- **Metric:** Root Mean Square Error (RMSE)
- For 2-fold cross validation:



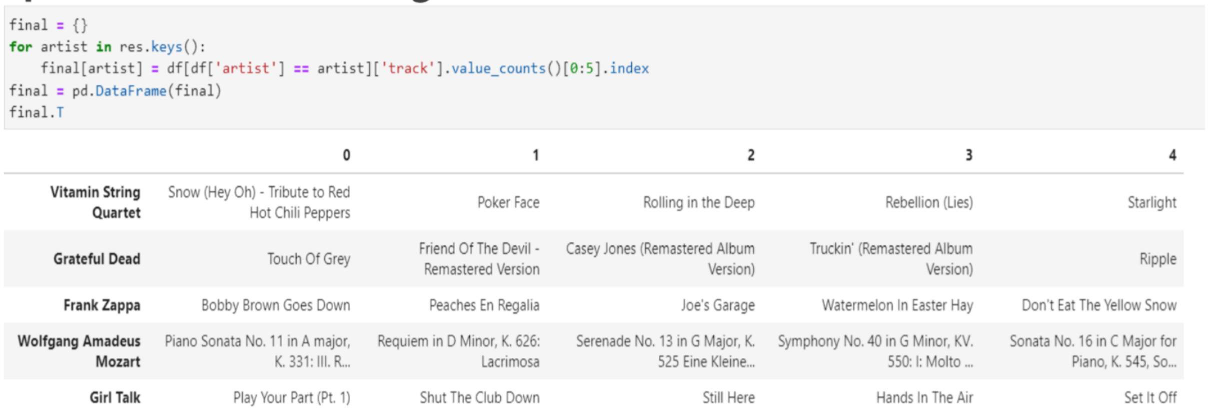
```
Processing epoch 21
Processing epoch 22
Processing epoch 23
Processing epoch 24
RMSE: 0.0299
```



```
Processing epoch 21
Processing epoch 22
Processing epoch 23
Processing epoch 24
RMSE: 0.0298
```

Input uid: 00055176fea2346e027cd3302289378b

Output artists and songs:



```python
final = {}
for artist in res.keys():
    final[artist] = df[df['artist'] == artist]['track'].value_counts()[0:5].index
final = pd.DataFrame(final)
final.T
```

	0	1	2	3	4
Vitamin String Quartet	Snow (Hey Oh) - Tribute to Red Hot Chili Peppers	Poker Face	Rolling in the Deep	Rebellion (Lies)	Starlight
Grateful Dead	Touch Of Grey	Friend Of The Devil - Remastered Version	Casey Jones (Remastered Album Version)	Truckin' (Remastered Album Version)	Ripple
Frank Zappa	Bobby Brown Goes Down	Peaches En Regalia	Joe's Garage	Watermelon In Easter Hay	Don't Eat The Yellow Snow
Wolfgang Amadeus Mozart	Piano Sonata No. 11 in A major, K. 331: III. R...	Requiem in D Minor, K. 626: Lacrimosa	Serenade No. 13 in G Major, K. 525 Eine Kleine...	Symphony No. 40 in G Minor, KV. 550: I: Molto ...	Sonata No. 16 in C Major for Piano, K. 545, So...
Girl Talk	Play Your Part (Pt. 1)	Shut The Club Down	Still Here	Hands In The Air	Set It Off

Conclusion

Overall, we learned a great deal from this project. In order to overcome inconsistencies in our two datasets, we attempted various techniques and tools including fuzzy matching, data cleaning, feature concatenation, user_id generation, and foregoing pandas altogether by using PySpark. In contrast to our studies where we have traditionally used “toy” datasets, dealing with large datasets like dataset 2 leads to traditional methods breaking down.

Appendix

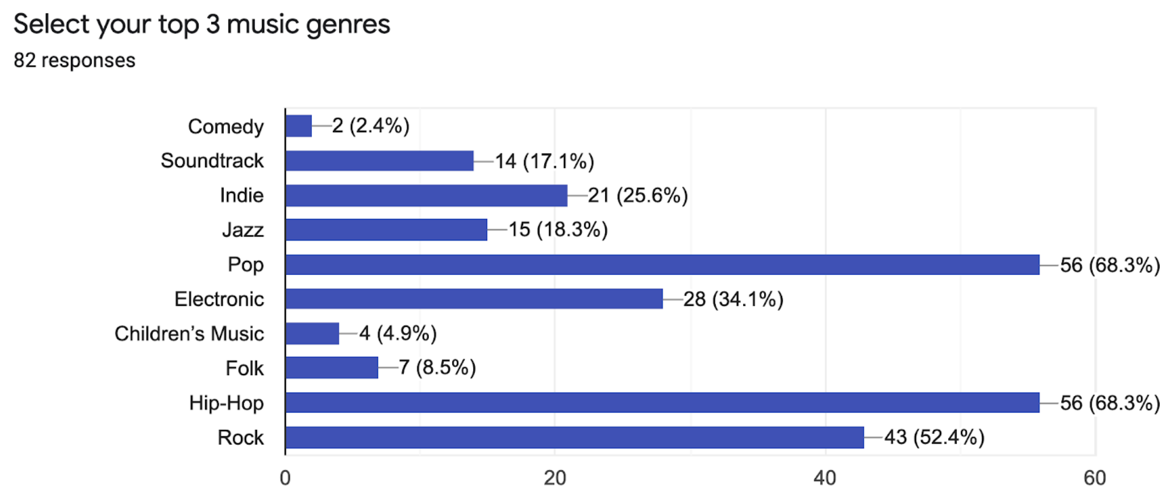


Figure 1a. Distribution of answers among 82 responds to Question 1 of the google survey showing that Pop and Hip-Hop were tied

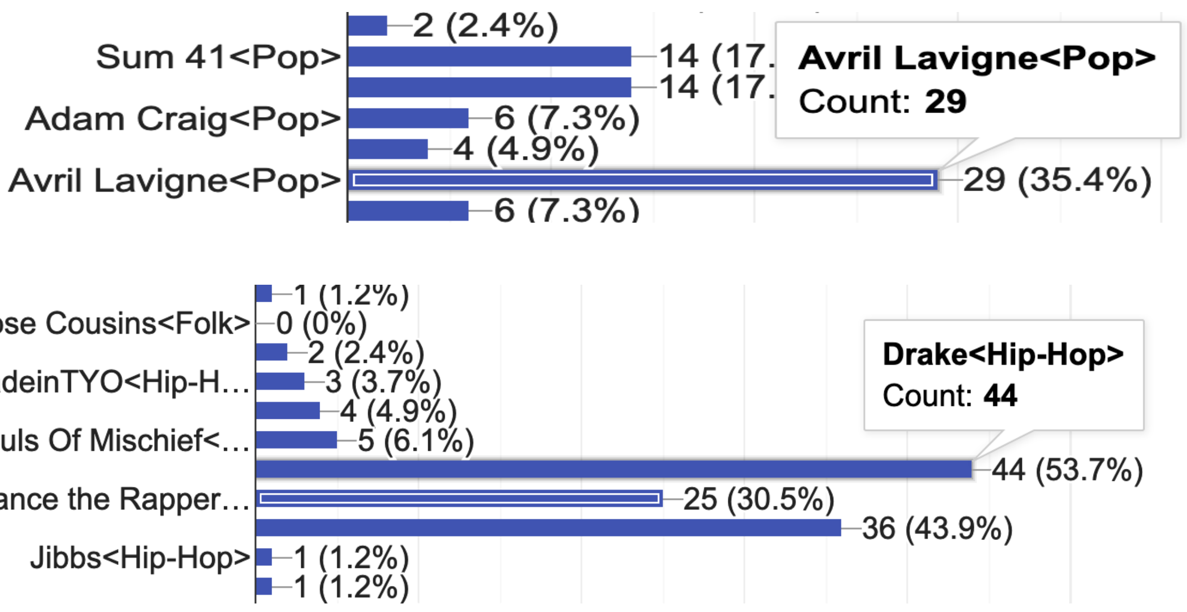
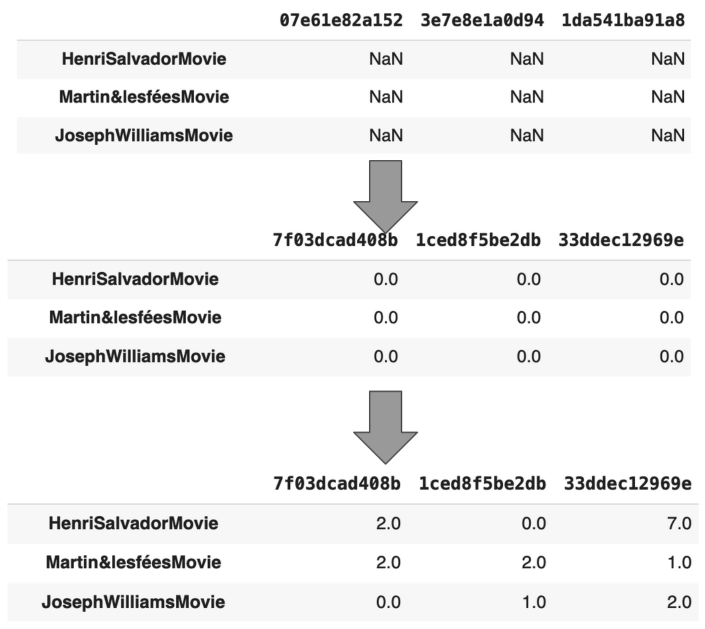


Figure 1b. Distribution of answers among 82 responds to Question 2 of the google survey showing that Avril Lavigne and Drake were the two most popular artists

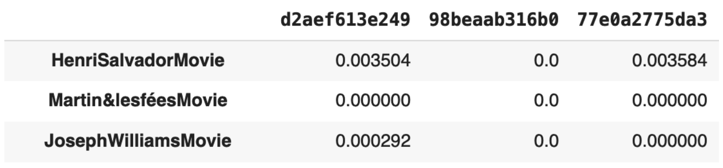


	07e61e82a152	3e7e8e1a0d94	1da541ba91a8
HenriSalvadorMovie	NaN	NaN	NaN
Martin&lesféesMovie	NaN	NaN	NaN
JosephWilliamsMovie	NaN	NaN	NaN

	7f03dcad408b	1ced8f5be2db	33ddec12969e
HenriSalvadorMovie	0.0	0.0	0.0
Martin&lesféesMovie	0.0	0.0	0.0
JosephWilliamsMovie	0.0	0.0	0.0

	7f03dcad408b	1ced8f5be2db	33ddec12969e
HenriSalvadorMovie	2.0	0.0	7.0
Martin&lesféesMovie	2.0	2.0	1.0
JosephWilliamsMovie	0.0	1.0	2.0

Figure 2a. Procedure to creating input matrix into tensorflow model



	d2aef613e249	98beaab316b0	77e0a2775da3
HenriSalvadorMovie	0.003504	0.0	0.003584
Martin&lesféesMovie	0.000000	0.0	0.000000
JosephWilliamsMovie	0.000292	0.0	0.000000

Figure 2b. Final input matrix showing normalized frequencies for artist-genre and user_id combinations

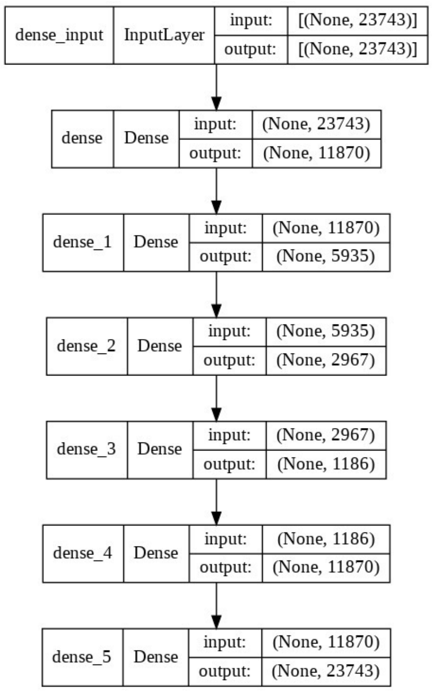
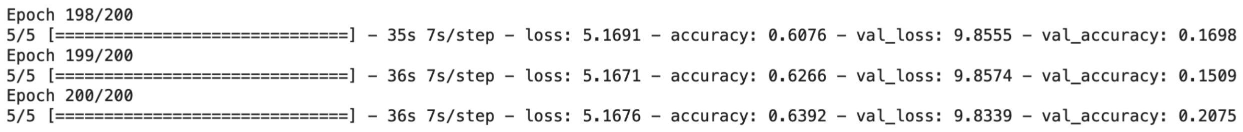
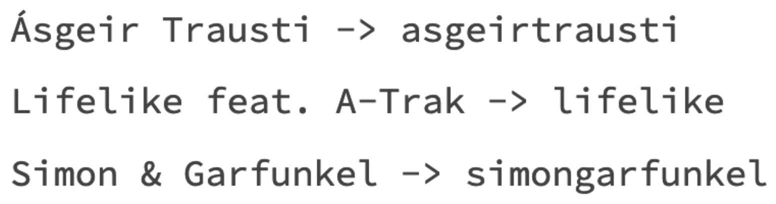

dense_input | InputLayer | input: [(None, 23743)] | output: [(None, 23743)]
dense | Dense | input: (None, 23743) | output: (None, 11870)
dense_1 | Dense | input: (None, 11870) | output: (None, 5935)
dense_2 | Dense | input: (None, 5935) | output: (None, 2967)
dense_3 | Dense | input: (None, 2967) | output: (None, 1186)
dense_4 | Dense | input: (None, 1186) | output: (None, 11870)
dense_5 | Dense | input: (None, 11870) | output: (None, 23743)

Figure 3. Neural Network for Task 1



```
Epoch 198/200
5/5 [==============================] - 35s 7s/step - loss: 5.1691 - accuracy: 0.6076 - val_loss: 9.8555 - val_accuracy: 0.1698
Epoch 199/200
5/5 [==============================] - 36s 7s/step - loss: 5.1671 - accuracy: 0.6266 - val_loss: 9.8574 - val_accuracy: 0.1509
Epoch 200/200
5/5 [==============================] - 36s 7s/step - loss: 5.1676 - accuracy: 0.6392 - val_loss: 9.8339 - val_accuracy: 0.2075
```

Figure 4. Tensorflow output for last 3 epochs showing loss, accuracy, validation loss, and validation accuracy



```
Ásgeir Trausti -> asgeirtrausti
Lifelike feat. A-Trak -> lifelike
Simon & Garfunkel -> simongarfunkel
```

Figure 5. Sample transformations leading to reduction of classes for the second dataset

PseudoCode

```
1) Install Requisite packages:
scikit-learn, scikit-surprise, pandas, numpy, random, nltk, pyspark, unidecode, re
2) Import all the requisite packages
3) Preprocess the dataset:
    a) Initialize a pandas DataFrame to read the spotify_dataset.csv file.
    b) Read the .csv file, with the condition to skip the rows with bad data and drop all the NA rows.
    c) Replace all the occurrences of “, name & blank spaces with null space
    d) Convert the data in the artist column to lower case
    e) Create a Regular Expression: [\s\S]+[\s]+(feat\.|ft\.|featuring|ft|feat)[\s]+[\s\S]+ to check for occurrences of ft., feat., featuring, ft, feat.
    f) Create a Regular Expression: [\s\S]*(&|and|\+)[\s\S] to check for occurrences of &, and, +
    g) Run the above regex on the data in artist column to filter out.
    h) Tokenize all the data in artist on the basis or words, and remove all punctuations and spaces
    i) Return this processed artist column as clean_artist
```

    j) Save it to a .csv file

4) Giving each artist an ID:
    a) Count the number of unique clean_artist values
    b) Create a range of the above length
    c) Generate a normal distribution to get the artist IDs and convert them to hexcode
    d) Assign these artist IDs to the respective artist from the clean_artist column using a pandas series
    e) Convert these artists IDs to string data type
    f) Store this new data frame as a .csv file

5) Counting frequency of user listening to each artist according to our dataset:
    a) Read the csv into a Spark DataFrame
    b) Run GroupBy User_ID and Clean_Artist
    c) Run the agg(count(*)) to count the number of entries of each artist for each user recursively
    d) Rename this new column name to 'freq'
    e) Scale these frequencies to range(1,5) using the MinMaxScaler

6) Inner Join the above 2 DataFrames (frequency and artist_ID) using the key as clean_artist and return the table containing

'user_ID', 'artist_ID', 'clean_artist', and 'freq'. This can be run as an SQL query.

7) Function for returning all predicted artists for a user, given user_ID:
    a) Fetch all the artist names for a given user_ID (can be done using SQL in Spark DataFrame)
    b) Then convert this DataFrame into a dictionary of items.
    c) Read the unique values of this dictionary to get the final output list containing the artists
    d) Return this list

8) Function for returning corresponding artist_name for a given artist_ID:
    a) Fetch all the unqiue artist names for given artist_IDs (can be done using SQL in Spark DataFrame)
    b) Fetch the top 5 artists one-by-one
    c) Return this list to step 9

9) Function for returning corresponding track_names for a given artist_name:
    a) Using the top 5 artist names, fetch the top 5 songs for each artist
    b) Store this as a dictionary, where keys are artists, values are track_name

c) Return this dictionary

10) Load the Reader, Dataset packages:

    a) Initialize Reader package with a rating scale (1,5), which will using frequency as a rating metric

    b) Initialize Dataset package with our DataFrame containing ‘user_ID’, ‘artist_ID’, and ‘freq’

11) Build a full training set

12) Run the SVD training algorithm for 25 epochs, and set verbose=True for printing the updates on progress of training

13) Fit this trained SVD model on the trainset

14) Generating predictions for a user, given their user_ID and artist data:

    a) Run SVD.predict with params as user_ID and iid as artist_ID. This will return a set of predictions.

    b) Run this set on the functions mentioned in steps 7, and 8 by passing appropriate parameters

    c) It returns a list of artist_IDs.

15) Run this list of artist_IDs on the function from Step 9 to return a dictionary of tracks.

16) Return.

Code

```
from surprise import Dataset, Reader, SVD, accuracy
from sklearn.preprocessing import MinMaxScaler
from surprise.model_selection import KFold
import pandas as pd
import numpy as np
import random
import json
df = pd.read_csv('./spotify_dataset.csv', on_bad_lines="skip")
df.head()
df.columns = df.columns.str.replace('"', '')
df.columns = df.columns.str.replace('name', '')
df.columns = df.columns.str.replace(' ', '')
df.columns
size = lambda x: len(x)
df_freq = df.groupby(['user_id', 'artist']).agg('size').reset_index().rename(columns={0:'freq'})[['user_id', 'artist', 'freq']]
df_freq.head()
artist = pd.unique(df['artist'].values.ravel())
artist = pd.Series(np.arange(len(artist)), artist)
df_freq["artist_id"] = df_freq[['artist']].applymap(artist.get)
```

```
artist_df = pd.DataFrame(df_freq["artist_id"].unique(), columns =
["artist_id"])

artist_df["artist"] = df_freq["artist"].unique()

df_freq.drop('artist', axis = 1, inplace = True)

df_freq.head()

with open('./artist_lookup.json') as json_file:

   artist_lookup_dict = json.load(json_file)

with open('./user_track_lookup_dict.json') as json_file:

   user_track_lookup_dict = json.load(json_file)

scaler = MinMaxScaler(feature_range=(1,5))

df_freq[["freq"]] = scaler.fit_transform(df_freq[["freq"]])

df_freq

reader = Reader(rating_scale=(1, 5))

data = Dataset.load_from_df(df_freq[['user_id', 'artist_id', 'freq']],
reader)

# First train an SVD algorithm on the movielens dataset.

trainset = data.build_full_trainset()

svd = SVD(n_epochs=25, verbose = True)

svd.fit(trainset)


# Then predict ratings for all pairs (u, i) that are NOT in the training
set.

# testset = trainset.build_anti_testset()
# predictions = algo.test(testset)

preds = []

for id in df_freq.artist_id.values:

        preds.append(svd.predict(uid="00055176fea2346e027cd330228
9378b", iid=id))

iid=[]

for pred in preds:

   iid.append(pred.iid)

iid=list(dict.fromkeys(iid))

def get_top_songs(artist, num_songs = 5):

   return list(df[df['artist'] ==
artist]['track'].value_counts()[0:num_songs].index)

def get_top_songs(artist, num_songs = 5):

   return list(df[df['artist'] ==
artist]['track'].value_counts()[0:num_songs].index)

res = {}

for i in iid[:5]:

   artist = artist_df.loc[artist_df.artist_id == i].artist.values[0]

   res[artist] = get_top_songs(artist)

for key, value in res.items():

   print(key, ' : ', value)
```

Reference

1. Schafer, J. B., Frankowski, D., Herlocker, J., & Sen, S. (2007). Collaborative filtering recommender systems. In *The adaptive web* (pp. 291-324). Springer, Berlin, Heidelberg.