Music Recommendation System

END TERM REPORT

by

Abhishek Srivastava
Rahul Kumar Singh
Ayush Sharma
Shivansh Kher

(Section: K18KY)
(Roll Number: 4,5,30,39)

Github Link-https://github.com/Rahulraj2509/Music-Reccomendation

Department of Intelligent Systems

School of Computer Science Engineering

Lovely Professional University, Jalandhar

Apr-2020

Introduction/Overview

A recommendation system is a program/system that tries to make a prediction based on users' past behavior and preferences. Recommendation systems are typically seen in applications such as music listening, watching movies and e-commerce applications where users' behavior can be modeled based on the history of purchases or consumption. We see them all around and it benefits a user in many ways because of its nature of prediction, value-add, and ease of consumption. A user wouldn't have to spend hours to decide about a particular movie or music or a product. The relevant and useful content gets delivered to the user at appropriate times. Our aim is to build such a recommendation system using the existing tools and technologies, in addition to adding our own flavor to the data parallelization aspect by using Spark and Deep Learning libraries such as Google's TensorFlow, Keras or Scikit-learn [2]. We hope to achieve similar performance, if not better than the researchers that have been working on such technologies.

Context

We see an explosion of Music streaming apps these days and sometimes wonder or wrack our brains as to which one serves our purpose and how do we get the relevant set of songs when we open the application. We have many songs recommendation systems out there and those are being used by popular companies like Spotify, SoundCloud, Pandora, Amazon Music, etc. And most of them do predict music or movies based on our previous watching/listening history and feedback. Most of them work based on Collaborative and Content-based filtering which they call the "Hybrid" model [3]. The companies use advanced machine learning algorithms and data processing engines like Spark and Hadoop to produce the best results possible. While all these technologies exist, this is our take on the song recommendation systems and how we can contribute even though minuscule, to the already popular technology. We are aware that Machine learning algorithms like Neural Networks and Deep Learning are used in such complex systems. Along with these algorithms, we will leverage Stochastic Gradient Descent in Spark [4] and execute the design on a distributed ecosystem like Hadoop, which is interesting.

Objective

The goal is to implement the Content-based filtering whose underlying concept is TF-IDF vectorization and Cosine Similarity and Collaborative Filtering using Stochastic Gradient Descent to build a comprehensive recommender system for songs.

Tasks involved

1. Literature Survey
2. Dataset collection
3. Data cleaning and pre-processing
4. Data visualization
5. Algorithm selection
6. Designing Recommender Architecture
7. Implementation of the algorithms

8. Quantitative analysis of the results

Approach

Our approach was a unified content-based and collaborative filtering model that could improve the song recommendation system in general. We considered a Stochastic Gradient Descent (SGD) approach due to a couple of reasons, one of them being the case that Mean Squared Error (MSE) for SGD as compared to ALS is a huge difference. And SGD tends to beat ALS while sometimes ALS recommends better songs. Our hunch was that this could be due to SGD trying to overfit more than ALS and is more susceptible to popularity bias. Although we can't back that up with empirical evidence or math yet, even Netflix's paper about movie recommendation vouches for the same. While there is an option to combine both the model into one like an ensemble approach, we didn't take that. Another reason for choosing SGD being its ability to perform better in A/B tests. A/B tests are randomized experiments with two variants A and B. It is an application of statistical hypothesis testing used more in the field of statistics.

Steps implemented

Collaborative Filtering:

- For this approach we have read user_artists(userID, artistID, weight) and artists(id, name) files as data frame and merged them.
- We scaled the weight column imported from the user_artists file.
- We formed the compressed sparse row matrix with userId as our rows and artistId as our columns and each entry in matrix contain weight(renamed to playcount) which is the number of times each user plays songs corresponding to artistId. Further we replaced each NaN value with 0.
- Splitting the data in test and train
- Defining Loss RMSE as our loss function.
- Calling Recommender class, which contains our main Stochastic Gradient Descent model that takes epochs, latent_feature, lambda, learning_rate as input parameters.
- Finally predicting the ratings in our test set.

Content Based Filtering:

The concepts of Term Frequency (TF) and Inverse Document Frequency (IDF) are used in information retrieval systems and also content based filtering mechanisms (such as a content based recommender). They are used to determine the relative importance of a document / article / song/ news item / movie etc. TF is simply the frequency of a word in a document. IDF is the inverse of the document frequency among the whole corpus of documents. In our system, a "Song" is a document and any feature that has more parity which could be single or multiple is the word. Here we are trying to find the importance of a document (a.k.a song) by utilizing the "genre" attribute. "genre" in this context is not a small set of standard genres we see in the world of music, instead it is a combination of tags associated, user created playlists and a combination of them. Hence we name it as a "feature" for our processing and convenience. The concept we are referring to can be thought of as analogous to the urban dictionary. Over a longer period of use, and evolution, the tags and playlists within the context of an app and then later among a wider population can become a permanent fixture and defines a genre.

When we translate the same concept to a recommender system, with the song data, and the genre(s) they belong to the consensus is that the more number of genres a song is associated with, more the popularity and acceptance.

Hence, we apply a TF-IDF vectorization, but while calculating TF-IDF, a log function is usually used to dampen the effect of high frequency words. For example: Pop = 3 vs Pop = 4 is vastly different from Pop = 10 vs Pop = 1000. In other words the relevance of a word in a document cannot be measured as a simple raw count.

After calculating TF-IDF scores, how do we determine which songs in a genre are closer to each other, rather matching the exact TF-IDF scores? This is accomplished by the Cosine Similarity which computes the proximity based on the angle between the vectors.

In this model, each item is stored as a vector of its attributes (which are also vectors) in an n-dimensional space and the angles between the vectors are calculated to determine the similarity between the vectors. Next, a song is characterized by its presence in multiple genres, associated with different tags and user generated playlists. The similarity between an item (song) with another item is characterized by the TF-IDF score obtained for a genre. A 2-D representation using Compressed Sparse Row (CSR) matrix with songs against genres is mapped. And when we find a matching TF-IDF score for two songs, then that genre will be considered and given a higher weightage. In this way, each song (in our dataset 17,000+) was compared against each other and a CSR matrix was generated.

From each row (each song, treated as a document) of the CSR matrix, highest value is selected and corresponding X and Y coordinates of the location in matrix is fetched. These numbers added into an array are sorted to get the top n popular songs. Then corresponding song "titles" along with their rank is returned in a sorted order.

- Steps

- We read ratings(userId, songId, rating, timestamp) and songs(songId, title, genre) "csv" files.
- we define 'featurize' method which is the main intuition behind the content based implementation, and takes songs file as input and added column for features named 'features'. Each row of our songs data frame contain a csr_matrix of shape (1, num_features).
- Each entry in csr matrix contain tf-idf value of the term. The 'featurize' method will return 2 items i.e songs dataframe and vocab which is a dictionary from term to int in sorted order.
- Randomly sampled our data for training
- Another method we define is 'cosine_similarity', which takes 2 input vectors of shape (1, number_features). This method is used to find the proximity between two csr_matrices.
- We define Mean Absolute Error as our loss function.
- And finally, our 'make_predictions' method returning one predicted rating for each element of ratings_test.

Algorithms implemented

Stochastic Gradient Descent for collaborative filtering, TF-IDF vectorization and Cosine similarity for content-based filtering were implemented in Python. Since, our idea is to make use of large datasets as and when the data grows, and train the model to predict better, we had to look for a high efficiency parallel processing system. Apache Spark achieves high performance for both

batch and streaming data, using a state-of-the-art DAG scheduler, a query optimizer, and a physical execution engine. This satisfied all our requirements and the concept of Resilient Distributed Datasets (RDD) introduced in this framework would break down the data, analyze and churn out the results that we visualized. A bottleneck in this regard was the learning curve for Scala or Pyspark. And naturally we selected Pyspark, but this was a major unknown and pushed us back few steps from our goal. The APIs used in Python 3.6 were largely different from the ones used in Pyspark and the way the dataframes are handled was nothing similar to Pandas dataframes. This stepback was one of the reasons why we weren't able to implement a Deep Learning model that took the outputs from Content and Collaborative filters to deliver accurate results. Although independently the content based and Collaborative filtering gives a very good precision our consensus is that probably Deep learning that uses Neural Networks would've improved results significantly.

Datasets

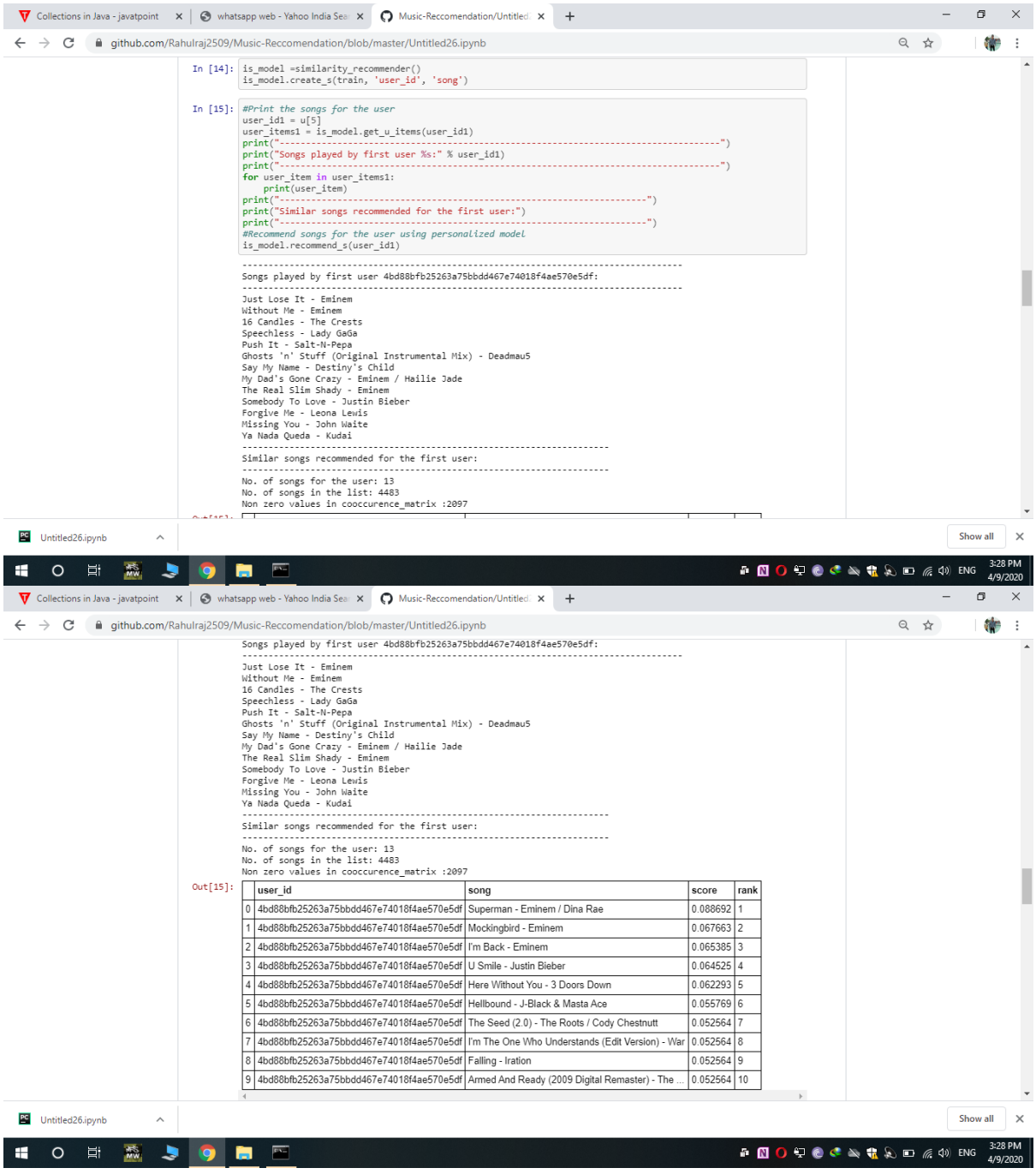
GroupLens[14] - HetRec 2011 - https://grouplens.org/datasets/hetrec-2011/ This dataset contains social networking, tagging, and music artist listening information from a set of 2K users from Last.fm online music system. http://www.last.fm

Dataset files used are:

- artists.dat - This file contains information about music artists listened and tagged by the users.
- tags.dat - This file contains the set of tags available in the dataset.
- user_artists.dat - This file contains the artists listened by each user. It also provides a listening count for each [user, artist] pair.

This dataset was built by Ignacio Fernandez-Tobas with the collaboration of Ivan Cantador and Alejandro Bellogn, members of the Information Retrieval group at Universidad Autonoma de Madrid (http://ir.ii.uam.es)

Million Song Dataset [1] - http://static.echonest.com/millionsongsubset_full.tar.gz Upon analysis of our dataset, we realized the magnitude was quite high to the ranks of millions. Since we cannot afford to run such a huge data and constrain the DSBA cluster or AWS, we are picking the smaller subset of 10,000 songs (1% of the entire dataset available from the source, 1.9GB). It contains "additional files" (SQLite databases) in the same format as those for the full set but referring only to the 10K song subset. Therefore, we can develop code on the subset, then port it to the full dataset.



	user_id	song	score	rank
0	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Superman - Eminem / Dina Rae	0.088692	1
1	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Mockingbird - Eminem	0.067663	2
2	4bd88bfb25263a75bbdd467e74018f4ae570e5df	I'm Back - Eminem	0.065385	3
3	4bd88bfb25263a75bbdd467e74018f4ae570e5df	U Smile - Justin Bieber	0.064525	4
4	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Here Without You - 3 Doors Down	0.062293	5
5	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Hellbound - J-Black & Masta Ace	0.055769	6
6	4bd88bfb25263a75bbdd467e74018f4ae570e5df	The Seed (2.0) - The Roots / Cody Chestnutt	0.052564	7
7	4bd88bfb25263a75bbdd467e74018f4ae570e5df	I'm The One Who Understands (Edit Version) - War	0.052564	8
8	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Falling - Iration	0.052564	9
9	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Armed And Ready (2009 Digital Remaster) - The ...	0.052564	10

Collections in Java - javatpoint | whatsapp web - Yahoo India Sea | Music-Reccomendation/Untitled

github.com/Rahulraj2509/Music-Reccomendation/blob/master/Untitled26.ipynb

In [16]:
```
user_id2 = u[7]
#Fill in the code here
user_items2 = is_model.get_u_items(user_id2)
print("----------------------------------------------------------------------")
print("Songs played by second user %s:" % user_id2)
print("----------------------------------------------------------------------")
for user_item in user_items2:
    print(user_item)
print("----------------------------------------------------------------------")
print("Similar songs recommended for the second user:")
print("----------------------------------------------------------------------")
#Recommend songs for the user using personalized model
is_model.recommend_s(user_id2)
```

```
----------------------------------------------------------------------
Songs played by second user 9d6f0ead607ac2a6c2460e4d14fb439a146b7dec:
----------------------------------------------------------------------
Swallowed In The Sea - Coldplay
Life In Technicolor ii - Coldplay
Life In Technicolor - Coldplay
The Scientist - Coldplay
Trouble - Coldplay
Strawberry Swing - Coldplay
Lost! - Coldplay
Clocks - Coldplay
----------------------------------------------------------------------
Similar songs recommended for the second user:
----------------------------------------------------------------------
No. of songs for the user: 8
No. of songs in the list: 4483
Non zero values in cooccurence_matrix :3429
```

Out[16]:

	user_id	song	score	rank
0	9d6f0ead607ac2a6c2460e4d14fb439a146b7dec	She Just Likes To Fight - Four Tet	0.281579	1
1	9d6f0ead607ac2a6c2460e4d14fb439a146b7dec	Warning Sign - Coldplay	0.281579	2
2	9d6f0ead607ac2a6c2460e4d14fb439a146b7dec	We Never Change - Coldplay	0.281579	3
3	9d6f0ead607ac2a6c2460e4d14fb439a146b7dec	Puppetmad - Puppetmastaz	0.281579	4

Untitled26.ipynb | Show all

ENG 3:28 PM 4/9/2020

Collections in Java - javatpoint | whatsapp web - Yahoo India Sea | Music-Reccomendation/Untitled

github.com/Rahulraj2509/Music-Reccomendation/blob/master/Untitled26.ipynb

```
Using TensorFlow backend.
```

In [1]:
```
import pandas
from sklearn.model_selection import train_test_split
import numpy as np
import time
```

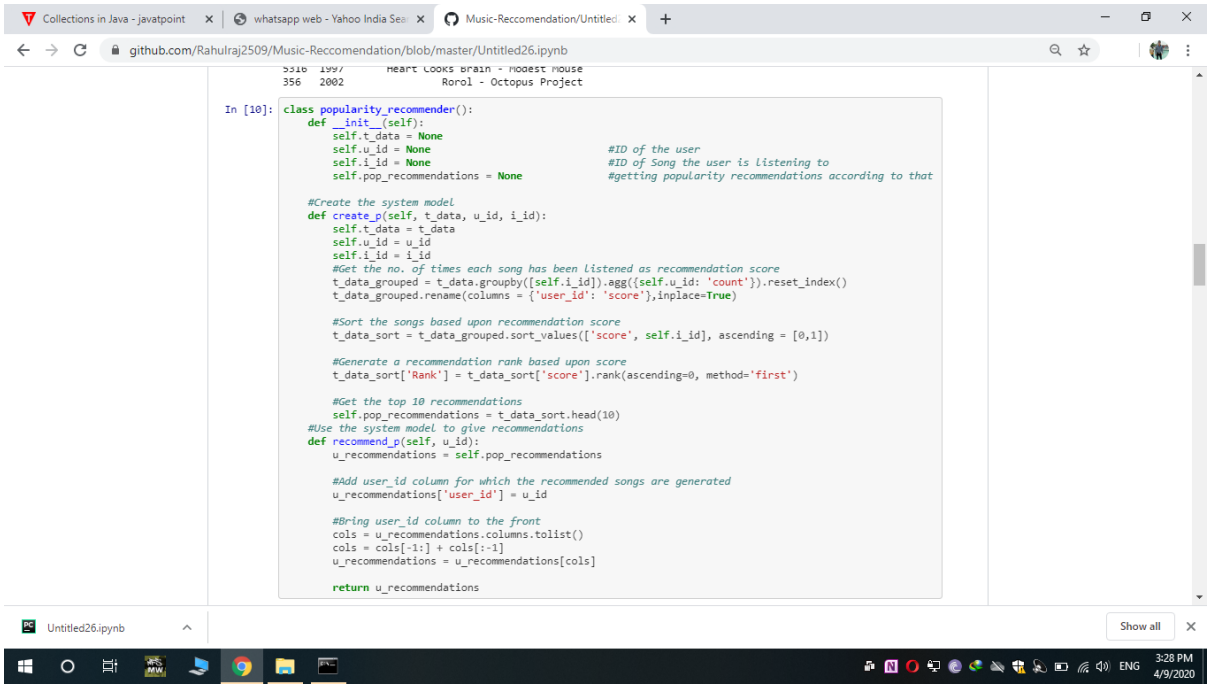
In [5]:
```
#Read user_id, song_id, listen_count
#This step might take time to download data from external sources
triplets = 'https://static.turi.com/datasets/millionsong/10000.txt'
songs_metadata = 'https://static.turi.com/datasets/millionsong/song_data.csv'
song_df_a = pandas.read_table(triplets,header=None)
song_df_a.columns = ['user_id', 'song_id', 'listen_count']
#Read song  metadata
song_df_b =  pandas.read_csv(songs_metadata)
#Merge the two dataframes above to create input dataframe for recommender systems
song_df1 = pandas.merge(song_df_a, song_df_b.drop_duplicates(['song_id']), on="song_id", how="left")
song_df1.head()
```

Out[5]:

	user_id	song_id	listen_count	title	release	artist_na
0	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOAKIMP12A8C130995	1	The Cove	Thicker Than Water	Jack Johnson
1	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBBMDR12A8C13253B	2	Entre Dos Aguas	Flamenco Para Niños	Paco De Lucia
2	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBXHDL12A81C204C0	1	Stronger	Graduation	Kanye W
3	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBYHAJ12A6701BF1D	1	Constellations	In Between Dreams	Jack Johnson
4	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SODACBL12A8C13C273	1	Learn To Fly	There Is Nothing Left To Lose	Foo Figh

Untitled26.ipynb | Show all

ENG 3:29 PM 4/9/2020



Performance Evaluation (quantitative)

Interpretation of results:

- The Root Mean Squared Error and Mean Absolute Error for Content Based Filtering is really good for a recommender system

Metric	Value
Root Mean Squared Error	0.367965
Mean Absolute Error	4.36796

- Collaborative Filtering metrics are in the process of evaluation. We will be updating when we receive the metrics. The delay is due to the enormous dataset and the challenge with the computational resources.

Task Division

S.No.	TASK	PERSON
1	Literature Survey	All
2	Problem Identification and Dataset Collection	All

S.No.	TASK	PERSON
3	Data Cleaning and Preprocessing	Abhishek and Rahul
4	Content and collaborative filtering	Ayush and Shivansh
5	Data Parallelization using Spark	Abhishek and Rahul
6	Prediction using Deep Learning(Future Work)	-
7	Experiments, Result Analysis, and Final Report	All

Tools/Technologies/Frameworks used

- Apache Spark 2.4.4
- Python 3.7
- Java 1.8
- Jupyter Notebook 6.0.0
- UNCC-DSBA cluster/ AWS EMR cluster
- Git for hosting the website

Packages used

- SciPy
- scikit-learn

Challenges

- It was challenging to understand the mathematical formulae, derivations and concepts behind content based and collaborative filtering.
- We had some confusion at one point regarding the latent features used in ALS algorithm.
- The learning curve to translate our understanding of TF-IDF vectorization and cosine similarity that are applied on bag of words to work for songs playlist was overwhelming, but we overcame it eventually.
- Enormous data sets that we use for song recommendation led us into a lot of performance related issues.
- Even the initial dataset we decided to implement this recommendation system had to be modified and additional datasets had to be merged to get all the required parameters.
- The understanding of the dataset and the relevant features were more time consuming than the actual implementation. Because lot of datasets had many features to offer, and we plotted various graphs such as correlation heat map, box plots (whisker plots) and histograms to reduce the dimensionality.

Things learnt

- Implementation of collaborative filtering using Stochastic Gradient Descent algorithm.
- Implementation Content based filtering using cosine similarity and TF-IDF vector.
- The complexity and challenges of pre-processing huge datasets and modeling the data suitably.
- During our literature survey, we learned different approaches to implement recommender systems.
- This project gave us a clear understanding of how to select datasets, what to look for in them and manipulate them to derive usable insights.

Conclusion

Recommendation systems are very prevalent nowadays and can be seen playing in the background of most websites and apps we visit. Whether we are visiting an e-commerce website like Amazon, a travel website such as Expedia, entertainment apps like Netflix, YouTube and Spotify, recommendation systems are an inevitable aspect. The inevitability arises due to the need to stay more relevant in business, acquire more customers and deliver an absolutely fabulous customer experience. In our project, we describe and attempt at developing one such recommendation system. We took into account, the Collaborative filtering and Content-based filtering to better predict the user's behavior. In the development of this project, we sought to overcome the widely known problems and shortcomings of such a system. In addition, we achieved data parallelization using Spark where we could make use of the RDD data structure and efficiently model the output. Although, we haven't implemented the Deep learning module that takes in the output from the filters, we hope to pursue our goal and make it more extensible to be used by an e-commerce or entertainment business.