Fundamentals of Big Data

(CMIS - 550)

Winter 2020

Date of Submission: 15th April 2020

Recreating Spotify's Discovery Weekly Recommendations Using Spark and Machine Learning

Submitted By:

Eduardo Cassinelli 260876480
Andres Polania 260700153

Introduction

Recommender systems were introduced in the mid-1990s to help consumers simplify their buying decisions in a world of overwhelmingly growing options. Simply put, these systems are designed to predict users' preferences and evaluations on different items (product or services) and establish a list of logically ranked suggestions thereof (Sharma & Singh, 2016). Recommender filters have become essential for many industries and, if well implemented, can afford companies with a clear advantage over competitors. Some clear examples of applied cases are travel (Booking.com), E-commerce (Amazon), movie streaming (Netflix), music streaming (Spotify), whose recommendation algorithms are driving innovation and defining business models. The aim of this paper is to examine the impact of recommender systems in the music industry with a specific focus on the Spotify algorithm.

1- The Technology Behind Recommender Systems

Recommender systems use different algorithms depending on the data structure and type of product. The most commonly used systems are collaborative filtering, content-based systems and hybrid systems (combining the former and the latter) (Serrano, 2018). While there are other types coming to prominence they are beyond the scope of this paper.

1.1- Collaborative Filtering-based Systems

Collaborative filtering is an algorithm that makes predictions based on people's engagement history with a specific product or service. It relies on the assumption that individuals with similar interests will also make comparable choices in the future (El Jamiy et al, 2015). The system analyzes a large group of users from which it filters a smaller set of users whose past behavior is similar. Once peer users/items with similar enough behavior are identified, the system recommends new items to different users with taste proximity (Sharma & Singh, 2016). As shown below, the basic principle of collaborative filters relies on the User and Item matrix (El Jamiy et al, 2015). In this specific example rows contain user purchase history and columns contain item categories. By cross comparing this data the algorithm can not only determine similarity between users' buying behavior but also the items to recommend in the future. The suggestions are items found on some profiles but not on other similar profiles and vice versa.

Users	***Item1***	***Item2***	***Item3***
U1	Yes	Yes	No
U2	No	Yes	No
U3	No	Yes	Yes

1.2- Content-Based Systems

Content-based systems are designed to recommend items with similar characteristics to those chosen by the user in the past (Serrano, 2018). They do so by matching up customers' preferences and interests with item attributes (movies, songs), and as a result recommend new items that fit the correlation. The result is a suggestion based on relevance that represents the user's level of interest in a particular object (El Jamiy et al, 2015). One example can be movie recommendations based on movie directors and genre. By analyzing

these two item attributes the algorithm can determine what future recommendations to make to a user that has watched a specific film. The output could be either movies in the same genre, movies made by the same director or both.

1.3- Hybrid Systems

Hybrid systems attempt to combine both content-based and collaborative algorithms in order to attenuate the shortcomings of each (Serrano, 2018). While content-based systems are very efficient in providing recommendations without considering other users' behavior or item rating, they are limited to item categories the user has expressed interest in. If the users interests are limited to, say, action movies then the algorithm would only suggest movies falling under that category. Collaborative systems, on the other hand, do not need to take item descriptions or related content into consideration when analyzing the data (Serrano, 2018). This allows them to accurately make recommendations of complex items regardless of their specific attributes which in turn enhances the variety of choices. However, the efficacy of collaborative systems depends on the number of users and/or item ratings in order to provide significant recommendations (El Jamiy et al, 2015). This issue can be mitigated to some extent by content-based recommender systems, which can predict item relevance even in the absence of prior ratings.

2- The Role of Big Data in Recommender Systems

Recommender systems depend highly on their ability to process large amounts of user data in order to establish statistically significant correlations and assumptions about users tastes and habits (Amir & Haider, 2014). Big Data technology has emerged to address the limitations of traditional technologies in processing such large volumes of data efficiently. At the core Big Data addresses three different challenges associated with any data intensive application (Serrano, 2018).

Volume: The amounts of data a recommender system has to take are not only massive but ever growing (Serrano, 2018). Successful applications such as Spotify can process hundreds of millions of users' data at a given time. Processing this much information is a clear technical challenge that can only be solved using Big Data .

Variety: As aforementioned, recommender systems may use different types of user data sources to make suggestions. Much of this data is unstructured (streaming history, log history) which cannot be processed by tradicional relational models (Serrano, 2018).

The enabling technologies of big data are parallel computing, cloud computing which allow multiple servers to work simultaneously on the same task. Techniques such as hadoop, HDFS and Spark (among others) allow us to effectively run and analyze massive amounts of unstructured data relatively inexpensively.

2.1- Recommender Systems in the music Industry

In the era of digitalisation, music has become easier to create, distribute, and access than ever. Music recommender systems (MRS) were developed to assist listeners in navigating through the myriad of available musical works and provide them with audio file suggestions to listen to (Shedl et al, 2018). These systems have gradually evolved from simple online streaming platforms to highly interactive automated spaces that engage

audiences with fresh new content. We will consider three examples to illustrate the evolution of the technology (Ciocca, 2017).

- One of the earliest music streaming platforms was developed by Songza in the early 2000's. The company focused on a manually selected list of songs which were hand-picked by experts and organized in genre categories for other people to listen to. Even though this system did not employ any type of automation, one could argue it was an early content-based attempt at recommending audio to users (Ciocca, 2017).

- Pandora released a music curation platform based on a content filtering algorithm. The platform added descriptive attributes to each audio file (genre, artist) and then employed a filter which made suggestions to users who had listened to songs with similar enough attributes (Ciocca, 2017).

- Finally, FM developed an algorithm with another approach using collaborative filtering to make suggestions based on people with similar enough streaming history (Ciocca, 2017).

As the technology evolves so does the scope of each platform. The most competitive music recommender systems today seek to expose music lovers to relevant content, new music genres and possibilities, and also allow new recording artists penetrate these audiences even in the absence of any historical data (Shedl et al, 2018).

2.2- Spotify

With close to 271 million users and featuring 50 million tracks, 450 million podcasts and 3 billion playlists, Spotify is arguably one of the most successful music streaming platforms in the world today (Irvine, 2018). Users can browse by parameters such as artist, album or genre, and can also edit, create and share playlists across the network and through social media. In addition, the platform has been revolutionary in creating a new business model which pays artists royalties based on number of streams as a proportion of total songs streamed.

Spotify's strength lies in its impressively accurate recommender technology allowing users to constantly discover new songs from different artists and in different genres. Out of the multiple services on the platform, Discover Weekly is one of the most popular (Ciocca, 2017). Every monday the algorithm automatically creates a weekly list of 30 songs that the listener would probably like, but has not yet listened to on the platform. The system is praised not only for being extremely accurate in predicting people's taste for music but also for eliminating barriers of entry for new artists looking to get exposure to massive audiences. In order to achieve this Spotify employs a combination of three different algorithms (Irvine, 2018).

Collaborative Filtering: Spotify employs both implicit and explicit collaborative filters. In a nutshell Implicit filters process streaming history while explicit filters process rating history (Irvine, 2018).

Natural Language Processing (Content-based): Analyzes text and blogs about description and opinions of users about the item (songs) (Ciocca, 2017).

Audio Models (Content-based): Analyzes the audio content itself (rhythm, beat, length, melodies)to classify it in a genre. This is important for new songs and artists coming into the system. Given new songs lack history, being classified by content allows the algorithm to be able to recommend these items to existing users (Ciocca, 2017).

3- Sample application: Recreating Spotify's Discover Weekly Playlist

For the development of the sample application, we decided to use the collaborative filtering algorithm, which will take the data about a user's previous behavior for music listening and compare it to similar users, in order to output recommendations for artists this user hasn't played before.

The application will be developed in a Google Collaborative notebook using Python and Spark, with the corresponding libraries needed as seen in the code below.

```
from pyspark import SparkContext
from pyspark.sql import SQLContext
from pyspark import SparkConf
from pyspark.mllib import recommendation
from pyspark.mllib.recommendation import *

sc = SparkContext()
sqlContext = SQLContext(sc)
```

For this purpose, a Audioscrobbler.com dataset obtained through a github project, containing three files will be used:

- ***artist_dataset.txt*:** contains a column with the artists' names and alias, and a second one with the corresponding IDs for each name or alias.
- ***artist_alias_dataset.txt:*** contains a column with the correct IDs of the artists, and a second one with the IDs corresponding to the alias.
- ***user_artist_dataset.txt:*** contains three columns: the users' IDs, the artists' IDs they have played, and the play count for each of these artists.

Each of these files will be stored in a variable.

```
artist_data = sc.textFile('/content/drive/My
Drive/CMIS550-TermProject/artist_dataset.txt')

artist_alias = sc.textFile('/content/drive/My
Drive/CMIS550-TermProject/artist_alias_dataset.txt')

user_artist_data = sc.textFile('/content/drive/My
Drive/CMIS550-TermProject/user_artist_dataset.txt')
```

Next, the data needs to be processed. For this, three functions are defined for each file that will convert the IDs into Integer types and create tuples for each row.

```
# This function will return a tuple with the ID of the artist converted
# into integer and the name of the artist.
def artist_info(data_set):
  artist_pair = data_set.split('\t')
  for i in range(len(artist_pair)):
    artist_pair[0] = int(artist_pair[0])
  return tuple(artist_pair)
```

```
# This function will return a tuple with both IDs of the artist
# converted into integers.
def alias_info(data_set):
  alias_pair = data_set.split('\t')
  for i in range(len(alias_pair)):
    alias_pair[0] = int(alias_pair[0])
    alias_pair[1] = int(alias_pair[1])
  return tuple(alias_pair)

# This function will return key and value pairs for the ID of the artist
# and the name of the artist and will convert the ID into an integer.
def user_info(data_set):
  user_data = data_set.split(' ')
  for i in range(len(user_data)):
    user_data[0] = int(user_data[0])
    user_data[1] = int(user_data[1])
    user_data[2] = int(user_data[2])
  return tuple(user_data)
```

Next, each function will be parsed to each RDD with the map function. The ***artist_alias*** RDD, already parsed, will be ***collectedAsMap***, in order to get the RDD with unique keys and values; which will be used in the next line of code to map the artists' alias IDs inside the ***user_artist_data*** RDD to the artists' correct IDs.

```
# Applies the artist_info function to the artist_data rdd
artist_data = artist_data.map(lambda x: artist_info(x))

# Applies the alias_info function to the artist_data rdd
artist_alias = artist_alias.map(lambda x: alias_info(x))

# Applies the user_info function to the user_artist_data rdd
user_artist_data = user_artist_data.map(lambda x: user_info(x))

# Collects only unique key and values from the artist_alias rdd
artist_alias_map = artist_alias.collectAsMap()

# Replaces the values of the user_artist_data rdd. With artist_alias_map,
# the alias IDs get replaced for the correct IDs.
user_artist_data = user_artist_data.map(lambda x: (x[0],
artist_alias_map.get(x[1], x[1]), x[2]))
```

After executing this code, the variable ***user_artist_data*** will contain all the necessary data to build the recommendation model using the spark machine learning library for collaborative filtering.

According to the Spark documentation, it is first needed to use the Rating function for each row in the ***user_artist_data*** RDD, as seen in the code below. After that, with the new ***plays*** RDD, we build our recommendations model specifying that our data has implicit feedback from the users (the number of plays for each artist), and specify the required arguments for the function.

```
# Building model
plays = user_artist_data.map(lambda p: Rating(int(p[0]), int(p[1]),
int(p[2])))

recommendation_model = ALS.trainImplicit(plays, rank=10, seed=200)
```

Before retrieving the recommendations, it is necessary to define the function below to extract the most played artists from the user for later comparison with the recommendations, in order to see if they make any sense.

```
def played_artists(user_id):
  # Filters the artists played by the specified users with the number of
      plays
  artist_data_testmap = artist_data.collectAsMap()

  user_played_artists_test = plays.filter(lambda x: x.user ==
      user_id).map(lambda x: (artist_data_testmap.get(x.product),
      x.rating))

  # return the top 10 played artists by the user
  return user_played_artists_test.sortBy(lambda x: x[1], False).take(10)

played_artists(1055449)
```

Next, the function for returning the recommendations by artists is defined as ***discovery_weekly.*** The first part of this function creates an RDD for all the played artists by a user, which will be used later in the function.

The recommendations are retrieved using the map function and the ***recommendProducts*** function. We need to output 100 recommendations since the ***recommendProducts*** function will not recommend only unknown artists to the user when using a collaborative filtering algorithm such as ***ALS***, which attempts to find a rating that relates each user to each item based on the items the user has already rated, so when finding recommendations the model will most times recommend these already rated products.

A for loop is used to append the names of the artists in the ***recommendations*** RDD to the empty list named ***recomendations_list***, using the ***lookup*** function. Another empty list named ***discovery_playlist*** is then created for later appending of the 10 new artists we will recommend to the user. The for loop that follows will allow filtering 10 artists from the recommendations_list that are not present in the ***user_played_artists*** RDD.

```
def discovery_weekly(user_id):
  # Filters the artists played by the specified users
  artist_data_map = artist_data.collectAsMap()
  user_played_artists = plays.filter(lambda x: x.user ==
      user_id).map(lambda x: artist_data_map.get(x.product)).collect()

  # Creates the recommendations using the recommendProducts function and
      appends it to a list
  recommendations = map(lambda x: x.product,
      recommendation_model.recommendProducts(user_id, 100))
  recommendations_list = []
  for artist in recommendations:
    recommendations_list.append(artist_data.lookup(artist)[0])

  # Creates a list with all the recommended artists the user has not played
      yet
  discovery_playlist = []
  for i in range(len(recommendations_list)):
    if recommendations_list[i] in user_played_artists:
      continue
    else:
```

```
      if len(discovery_playlist) < 10:
        discovery_playlist.append(recommendations_list[i])
      else:
        break

  return discovery_playlist
```

Finally, the ***discovery_weekly*** function is called with the same user's ID as the ***played_artist*** function, returning the ***discovery_playlist*** list with the recommendations for artists that the user has never played.

```
# Calls the function and returns the list with the recommendations
discovery_weekly(1055449)
```

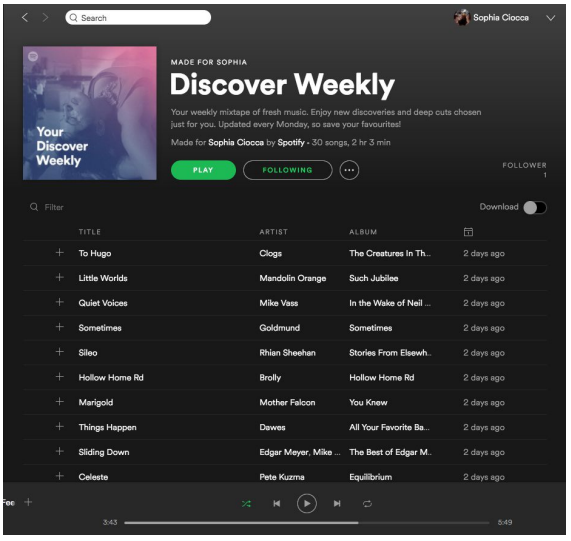
We can now compare the recommendations returned by the discovery_weekly function to the top 10 played artists, by the user to verify if the user can relate to these recommendations. Indeed, most of the recommendations go in-line with the most played artists by the user with ID ***1055449***, as seen in the table below.

Top 10 Artists for user ID 1055449		Discovery Weekly Recommendations for user ID 1055449	
Artist	**Genre**	**Artist**	**Genre**
Pearl Jam	Alternative Rock - Grunge - Hard Rock	The Cardigans	Alternative Rock - Indie Rock
Queens of the Stone Age	Alternative Rock - Stoner Rock - Alternative Metal	Counting Crows	Alternative Rock - Roots Rock - Indie Rock
Foo Fighters	Alternative Rock - Post-grunge - Hard Rock	Oasis	Alternative Rock - BritRock
A Perfect Circle	Alternative Rock - Hard Rock - Art Rock	The Beach Boys	Rock - Surf Rock - Psycodelia
Tool	Progressive Rock - Alternative Metal - Art Rock	Moby	Rock - Punk Rock - Ambient Rock
Radiohead	Art Rock - Alternative Rock - Experimental Rock	Portishead	Alternative Rock - Experimental Rock - Electronica
Jethro Tull	Progressive Rock - Hard Rock - Jazz Fussion	Michael Jackson	Pop - Pop Rock - Soul - Disco
The Smashing Pumpkins	Alternative Rock	Stevie Wonder	Jazz - Funk - R&B - Soul- Pop
Desert Sessions	Desert Rock - Stoner Rock - Alternative Rock	Beastie Boys	Alternative HipHop - Alternative Rock - Jazz Funk
The Black Crows	Blues Rock - Jam Rock	Björk	Experimental - Art Pop - Electronica

Most of the artists recommended by the application to the user are Alternative Rock bands. We can see a couple of recommendations for Jazz/Funk/Pop artists, most likely because two of the most played artists by the user are close to these genres.

4- Comparison of sample application results to real-world recommendations.

As stated before, Spotify uses a combination of algorithms in order to output the recommendations as new songs, not only based on the information of artists played by similar users, but also based on external data, and the inherent data (audio content) of song of the artist.



We are able to conclude that the results for the sample application are in the right track, but for it to output the same recommendations, not only at an artist level but also at a song level, it would take another data set with songs data and metadata to implement a content-based analysis, and query the songs based on the top artists for the user retrieved by the application.

Bibliography

- Ciocca, Sophia (2017). How Does Spotify Know You So Well. *Medium-Artificial Intelligence* *https://medium.com/s/story/spotifys-discover-weekly-how-machine-learning-finds-your-new-music-19a41ab76efe*

- Huq, Proma (2019). CCTP-607: “Big Ideas”: AI to the Cloud. *Spring 2019, https://blogs.commons.georgetown.edu/cctp-607-spring2019/author/ph625/*

- Schedl, Markus et al, (2018). Current challenges and visions in music recommender systems research. *Int J Multimed info Retr 7, 95-116* *https://doi.org/10.1007/s13735-018-0154-2*

- Serrano, Will (2010). Intelligent Recommender System for Big Data Applications Based on the Random Neural Network. *Big Data and Cognitive Computing*

- Gandomi Gandomi et al (2014). Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management 35 137–144*

- Sharma, Richa et al (2016). Evolution of Recommender Systems from Ancient Times to Modern Era: A Survey. *Indian Journal of Science and Technology, Vol 9(20), DOI: 10.17485/ijst/2016/v9i20/88005,*

- EL Jamiyl, Fatima (2015). The potential and challenges of Big data - Recommendation systems next level application. https://arxiv.org/pdf/1501.03424.pdf

- Liao, Kevin (2018). Prototyping a Recommender System Step by Step Part 2: Alternating Least Square (ALS) Matrix Factorization in Collaborative Filtering. *Towards Data Science.* *https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square-als-matrix-4a76c58714a1*

- Spark Documentation (2020). Collaborative Filtering - RDD-based API. *Apache Spark.* *https://spark.apache.org/docs/latest/mllib-collaborative-filtering.html*

- Boam, Eric (2019). I Decoded the Spotify Recommendation Algorithm. Here’s What I Found. *Medium.* *https://medium.com/@ericboam/i-decoded-the-spotify-recommendation-algorithm-heres-what-i-found-4b0f3654035b*

- ABDI-Project (2020). Music Recommender System using Apache Spark and Python. *Github.* *https://github.com/ADBI-Project/Project1*