UNIVERSITÀ DELLA CALABRIA
DIPARTIMENTO DI INGEGNERIA INFORMATICA, MODELLISTICA, ELETTRONICA E SISTEMISTICA (DIMES)

Dipartimento di Eccellenza 2018-2022

Big Data Analytics Project - Computer Engineering for IoT 2020 / 2021

Spotify recommendation system: a proposal for a Spotify recommendation system based on the usage of the Frequent Pattern Mining algorithm *FP-Growth*.

Luigi Rachiele[†]

The rise of the web enabled new distribution channels like online stores and streaming platforms, offering a vast amount of different products. For helping customers finding products according to their taste on those platforms, recommender systems play an important role. In this project is shown a Spotify recommendation system based on a dataset updated by user via Twitter. The recommendation system is built on a Frequent Pattern Mining algorithm, the FP-Growth. In the following it will be explained the code, all written in Python the algorithm and all the platform used.

1 Introduction

Music recommendation has more and more become a worldwide recognition, especially with the technical approaches nowadays. Therefore, an ideal music recommender has to maximize the user's satisfaction but simultaneously "minimize the user's effort to start the recommender and providing feedback."

The music recommender must recommend user's songs of his playlist. In this paper **Spotify** will be the main example to show, because there are 100 million monthly active users with millions of songs and playlists. Also, Spotify gives to the developer an useful library called Spotipy.

2 The Project

For the recommendation system will be used a **Frequent Pattern Mining** algorithm, based on a dataset of playlist shared on twitter by several clients. All the project is written in Python. The user will provide is own playlist and the association rules given by the *Frequent Pattern Mining* algorithm will show, as prediction, a list of recommended songs that the user could add at its own playlist. These will be shown in the following.

2.1 Platform used

For the development of this project are used different platforms:

- Spotify: the musical service where the recommendation system works;
- Kaggle.com: for recovering the dataset;
- PyCharm: to develop the script that recover the user playlist from *Spotify*;
- Google Colab: to develop the remaining part of the script;
- Overleaf: to develop this LateX paper;
- Spark: is the distributed framework used as analytics engine for the data processing.

2.2 Libraries used

To develop this project are used several libraries but the most important ones are:

- **Spotipy**: library that gives important features for retrieving data from Spotify. In this part of the project is used to retrieve an user's playlist;
- **Pandas**: used to create and modify some dataframe in optimized way;
- **PySpark**: used to implment in Python the framework *Spark*;
- **...other**: more other library to support the most important ones.

2.3 Frequent Itemset and Association Rules Mining

Frequent pattern discovery (or FP discovery, FP mining, or Frequent itemset mining) is part of knowledge discovery in databases and data mining; it describes the task of finding the most frequent and relevant patterns in datasets. The concept was first introduced for mining transaction databases. Frequent patterns are defined as subsets (itemsets, sub sequences, or substructures) that appear in a data set with frequency no less than a user-specified or auto-determined threshold called *minSupport*.

The two principle algorithm used for the Frequent Itemset and Association Rules Mining are the **apriori algorithm** and the **FP-growth**. They are different in method, complexity and compu-

[†] University of Calabria

tation efficiency. The FP-growth scales better with the support threshold.

2.3.1 Apriori algorithm

The Apriori algorithm is based on the Apriori principle[1]:

- If an itemset is *frequent*, then all of its subsets must also be frequent;
- If an itemset is **not frequent**, then all of its supersets cannot be frequent

$$\forall X, Y : (X \subseteq Y) \rightarrow s(X) \geq s(Y)$$

- The *support* (number of transaction in which the item is present) of an itemset never exceeds the support of its subsets;

So the algorithm works in different step resumed here:

1. Candidate generation;
2. Candidate pruning;
3. Support counting;
4. Candidate elimination;

But this algorithm in the real application has a problem of bottleneck in two different point *candidate generation* and *multiple scans of database*. 10^4 frequent 1-itemset will generate 10^7 2-itemset candidates and needs *n+1 scans* with *n* as the length of the longest pattern.

2.3.2 FP-Growth

To compensate those bottleneck, for this project is chosen another type of algorithm, the **FP-growth**.

The algorithm is based on the principle of *compressing* a large dataset into a compact **Frequent Pattern tree** structure.
In this way the cost of the multiple scans will be avoided. Developing an efficient **FP-tree-based** frequent pattern mining method permits the use of a **divide-and-conquer** methodology and permits to avoid the **candidate generation** that create bottleneck in the Apriori algorithm.

The algorithm follows different step:

1. Scan DB once, find frequent 1-itemset;
2. Order frequent items in frequency descending order;
3. Scan DB again to construct the FP-tree as the figure 1 shows.

For each item construct its *conditional pattern-base*. Then construct its *conditional FP-tree* and repeat the operation on each newly created conditional FP-tree. This process goes until the resulting FP-tree is empty or it contains only one path that will generate all the combinations of its sub-paths, each of which is a frequent pattern.

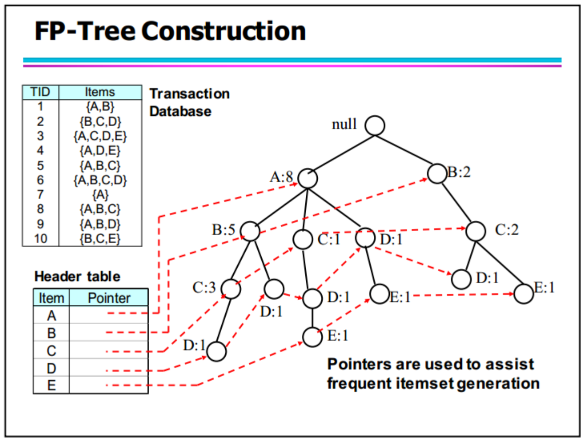


Fig. 1 FP-growth schema

2.3.3 Apriori vs FP-Growth

FP-growth is an order of magnitude faster than Apriori and the reason is because there is no *candidate generation* and no *candidate test*. Also there is an use of compact data structure and the elimination of repeated DB scan.

For this reason the FP-Growth scales much better with the decreasing of the support threshold as the figure 2 shows.

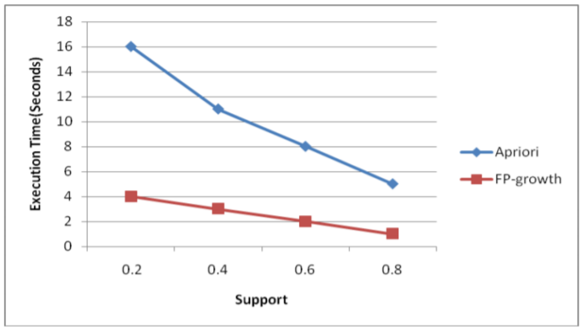


Fig. 2 FP-Growth vs Apriori algorithm

3 Dataset

The dataset chosen for this project is realized collecting the users who publish their nowplaying tweets via Spotify. "nowplaying" tweets are tweets, where the users explicitly state to which track they are listening to at the moment on the micro blogging platform Twitter.

3.1 Content

The CSV-file holding the dataset contains the following columns:"*user_id*", "*artistname*", "*trackname*", "*playlistname*", where:

- *user_id* is a hash of the user's Spotify user name;
- *artistname* is the name of the artist;
- *trackname* is the title of the track;

- *playlistname* is the name of the playlist that contains this track.

The separator used is "," and each entry is enclosed by *double quotes* and the escape character used is ".".

3.2 Dataset Analysis

The number of rows in the datasets are: *12774191*. The unique value are

- *15897* unique **user_id**;
- *303027* unique **artistname**;
- *2057995* unique **trackname**;
- *155522* unique **playlistname**.

Multiple analysis could be done in this dataset. The starting point was to plot some data.

First of all, in the picture 3 there are plotted the playlist for each different user.

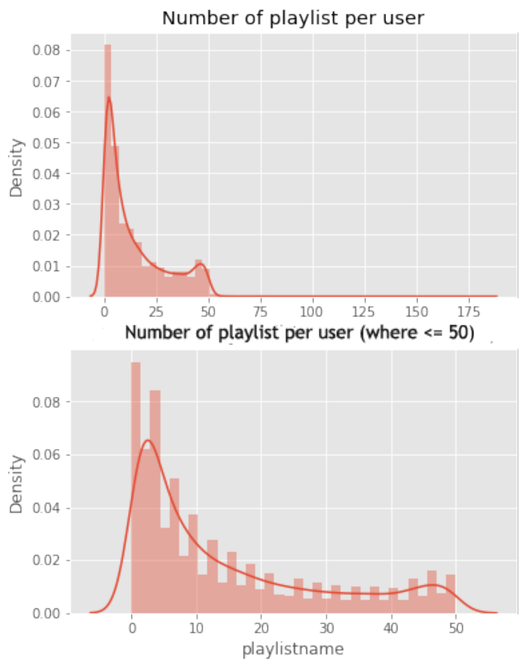


Fig. 3 Number of playlist for each different user

In the picture 4 are plotted the number of artist for each different user.

In the picture 5 are plotted the most common playlist names. It can be great if this data can be used in some way, maybe to recognize the genre of the playlist.

In the picture 6 is plotted the top 30 of the most popular artist in the dataset.

In the last picture 7 is plotted the top 30 of the most popular songs in the dataset.

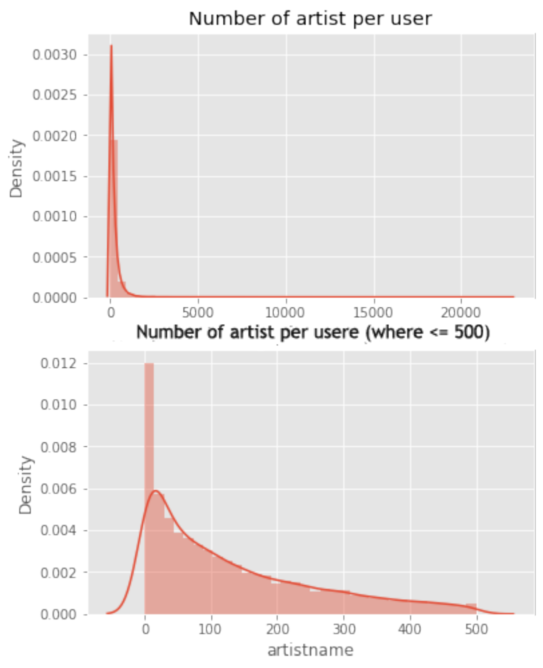


Fig. 4 Number of artist for each different user

3.3 Data Preprocessing

In this section the main focus is the data preprocessing. First of all, in the dataset are dropped the rows that are useless in this study:

- the rows containing null value;
- the rows that contain the songs with "intro, home, starred, closer", that's because it will affect negatively the algorithm of FP-growth.

Then, considering that all the value in the dataset are *Strings* it's better, for computation purpose, to map every value in Integer. That will help in term of computation when the algorithm will be processed.

```
spotify_data['trackname'] =
spotify_data['trackname'].str.lower().map(track_dict)
```

At this point, the dataset need to be in the format of transactions so apply a zip function as following:

```
spotify_summary = spotify_data.groupby
(['user_id','playlistname'])['trackname']
.apply(zip_list).reset_index()
```

The dataset now has the form of

```
[playlistname, list[track]]
```

Each time every big function is applied to data (also the dictionary are created) the data structure are stored in pickle file, to better load the program without performing again the operations.

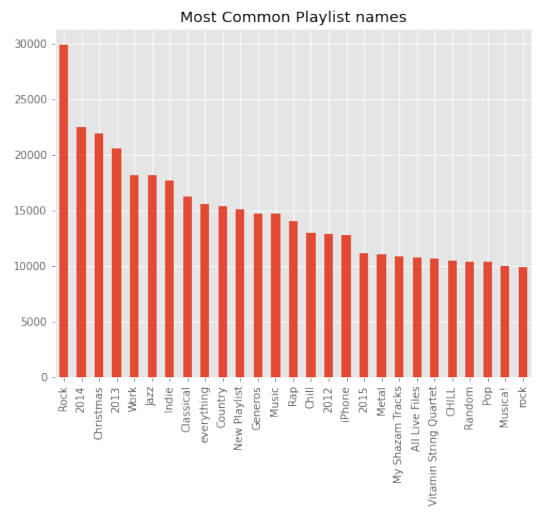


Fig. 5 Most common playlist name

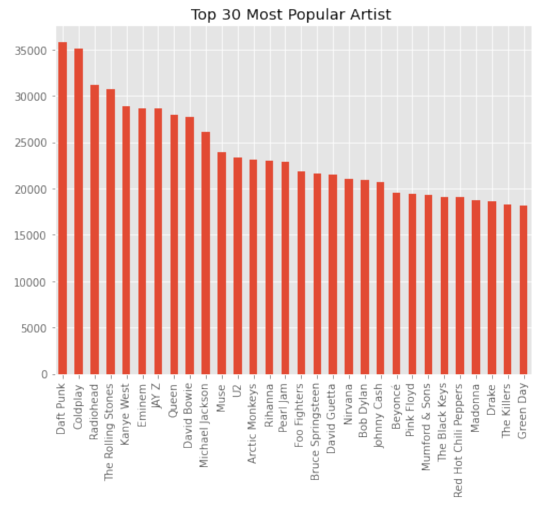


Fig. 6 Top 30 popular artist

3.4 Not all the dataset

The machine that is used for running is a single one (12 gb ram), not a cluster. This implies that a big sacrifice on the dataset has to be performed, because the algorithm FP-Growth, applied on the complete dataset, use a lot of memory and cause crash on the Colab virtual machine.

So, the idea was to use a part of the dataset, but how chose which one? The idea is to start by the user playlist in which the algorithm has to be applied. Starting from this, a new ad hoc dataset is created, containing only the playlist that contain at least one track that the user playlist contains. That permits to perform the algorithm with this machine and have a good model, but for sure not good as a model trained on the entire dataset.

4 Spark and FP-Growth

Once the dataset is ready, it's time to train the model.

First of all the *pandas dataframe* is converted in a *Spark dataframe*.

Then it needed to set the value of *minSupport* and *minConfi-*

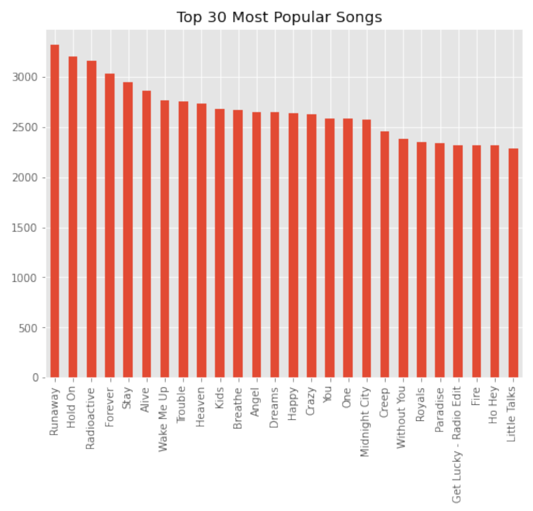


Fig. 7 Top 30 most popular **songs** in dataset.

dence. The *Support* is the number of transaction that contain the item X and Y, or also the probability that the item X and the item Y appears together in the same transaction and it is calculated in the following way:

$$\frac{\sigma(X,Y)}{|T|}$$

The Confidence instead measures how often items in Y appear in transactions that contain X, or also the probability that the item X appears in the transaction given that the Y is in the transaction.

$$\frac{\sigma(X,Y)}{Y}$$

The *minSupport* represents the threshold behind which an itemset can be considered frequent. After several tests the *minConfidence* chose is *0.6 minSupport* chosen is *0.05*. Value below that gives crash on Java and instead value above that gives useless result.

The playlist used as example in the test is the following:

```
["Till I Collapse", "Lose Yourself",
"Love is a Laserquest", "Madness", "Man in Black",
"Notion", "The Song You Sing", "Complicated","Get lucky"]
```

5 Output

After executing the algorithm the *Association Rules* are given in the form as show in the figure 8.

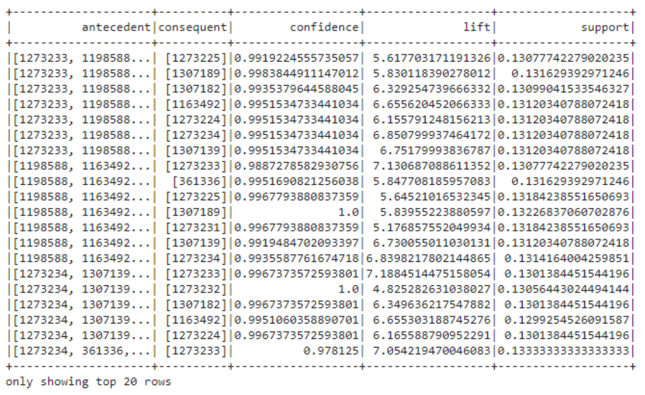
The last part consist to apply the model trained at the playlist given by user:

```
model.transform(playlist_dataset).show(5, False)
```

The output is a table with the prediction and it's all shown in the figure 9.

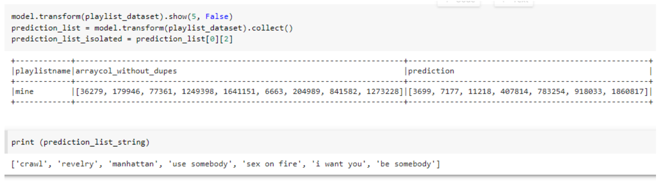
The start was with:

```
["Till I Collapse", "Lose Yourself",
"Love is a Laserquest", "Madness",
"Man in Black", "Notion",
"The Song You Sing", "Complicated","Get lucky"]
```



```
+--------------------+----------+------------------+------------------+-------------------+
|          antecedent|consequent|        confidence|              lift|            support|
+--------------------+----------+------------------+------------------+-------------------+
|[1273233, 1198588...| [1273225]|0.9919224555735057| 5.617703171191326|0.13077742279020235|
|[1273233, 1198588...| [1307189]|0.9983844911147012| 5.830118390278012|   0.13162939297124 6|
|[1273233, 1198588...| [1307182]|0.9935379644588045| 6.329254739666332|0.13099041533546327|
|[1273233, 1198588...| [1163492]|0.9951534733441034| 6.655620452066333|0.13120340788072418|
|[1273233, 1198588...| [1273224]|0.9951534733441034| 6.155791248156213|0.13120340788072418|
|[1273233, 1198588...| [1273234]|0.9951534733441034| 6.850799937464172|0.13120340788072418|
|[1273233, 1198588...| [1307139]|0.9951534733441034|  6.75179993836787|0.13120340788072418|
|[1198588, 1163492...| [1273233]|0.9887278582930756| 7.130687088611352|0.13077742279020235|
|[1198588, 1163492...|  [361336]|0.9951690821256038| 5.847708185957083|  0.131629392971246|
|[1198588, 1163492...| [1273225]|0.9967793880837359|  5.64521016532345|0.13184238551650693|
|[1198588, 1163492...| [1307189]|               1.0|  5.83955223880597|0.13226837060702876|
|[1198588, 1163492...| [1273231]|0.9967793880837359| 5.176857552049934|0.13184238551650693|
|[1198588, 1163492...| [1307139]|0.9919484702093397| 6.730055011030131|0.13120340788072418|
|[1198588, 1163492...| [1273234]|0.9935587761674718|6.8398217802144865| 0.1314164004259851|
|[1273234, 1307139...| [1273233]|0.9967373572593801|7.1884514475158054| 0.1301384451544196|
|[1273234, 1307139...| [1273232]|               1.0| 4.825282631038027|0.13056443024494144|
|[1273234, 1307139...| [1307182]|0.9967373572593801| 6.349636217547882| 0.1301384451544196|
|[1273234, 1307139...| [1163492]|0.9951060358890701| 6.655303188745276| 0.1299254526091587|
|[1273234, 1307139...| [1273224]|0.9967373572593801| 6.165588790952291| 0.1301384451544196|
|[1273234, 361336,...| [1273233]|          0.978125| 7.054219470046083|0.13333333333333333|
+--------------------+----------+------------------+------------------+-------------------+
only showing top 20 rows
```

Fig. 8 Association rules given as output by FP-Growth algorithm.



```
model.transform(playlist_dataset).show(5, False)
prediction_list = model.transform(playlist_dataset).collect()
prediction_list_isolated = prediction_list[0][2]
```

```
+------------+-----------------------------------------------------------------------+-----------------------------------------------------+
|playlistname|arraycol_without_dupes                                                 |prediction                                           |
+------------+-----------------------------------------------------------------------+-----------------------------------------------------+
|mine        |[36279, 179946, 77361, 1249398, 1641151, 6663, 204989, 841582, 1273228]|[3699, 7177, 11218, 407814, 783254, 918033, 1860817]|
+------------+-----------------------------------------------------------------------+-----------------------------------------------------+
```

```
print (prediction_list_string)
```

```
['crawl', 'revelry', 'manhattan', 'use somebody', 'sex on fire', 'i want you', 'be somebody']
```

Fig. 9 Output given by FP-Growth algorithm.

and the output is:

```
['crawl', 'revelry', 'manhattan',
'use somebody','sex on fire',
'i want you', 'be somebody']
```

6 Next idea for the project

The project has a lot of possible development in the future.

One of this is the usage of a cluster (or a better machine) and compute the algorithm in all the dataset, without creating the one ad-hoc. It will for sure give better result on the precision of the prediction.

Another possible focus for next project is not by changing hardware but changing semantic. It could be possible to apply some text based analysis on the name of the playlist. It should be cool to figure out the connection between all the different name, many of which has in common words and tracks.

Notes and references

1 **Tagarelli, Andrea**. "*Frequent Itemset and Association Rules*". Big Data Analytics, 2020, UNICAL. Class Lecture.