Market-basket Analysis

Martina Casati 942413 Andrea Frattini 959213

Contents

List of Figures

Abstract

The goal of our project is to point out which actors and actresses or combination of both, are more frequently present in international movies. In order to meet the aim of the project, we performed Market-Basket analysis on IMDB dataset, where actors and actresses have been considered as 'items' while movies have been identified as 'baskets'. In the stream of the project we implemented different taylored algorithm: the Apriori algorithm and the FP Growth algorithm. In order to perform our analysis we implemented our algorithms on Spark, which is useful for managing large datasets. The project has been written on Python 3 using both Google Colaboratory, for a better reproducibility of the results, and Google Cloud Platform's VM for the scalability analysis.

1 Introduction

1.1 Dataset

The dataset used in this project is the IMDB dataset, downloaded from Kaggle under IMDb non-commercial licensing. It contains 5 different datasets that provide detailed informations about movies, more in detail:

1. title.akas.tsv.gz - Contains the following information for titles:
    - titleId (string): an alphanumeric unique identifier of the title.
    - ordering (integer): a number to uniquely identify rows for a given titleId.
    - title (string): the localized title.
    - region (string):the region for this version of the title.
    - types (array): Enumerated set of attributes for this alternative title. One or more of the following: "alternative", "dvd", "festival", "tv", "video", "working", "original", "imdbDisplay". New values may be added in the future without warning.
    - attributes (array): Additional terms to describe this alternative title, not enumerated.

- isOriginalTitle (boolean): 0: not original title; 1: original title

2. title.basics.tsv.gz - Contains the following information for titles:

   - tconst (string) - alphanumeric unique identifier of the title.
   - titleType (string) – the type/format of the title (e.g. movie, short, tvseries, tvepisode, video, etc).
   - primaryTitle (string) – the more popular title / the title used by the filmmakers on promotional materials at the point of release.
   - originalTitle (string) - original title, in the original language.
   - isAdult (boolean) - 0: non-adult title; 1: adult title.
   - startYear (YYYY) – represents the release year of a title. In the case of TV Series, it is the series start year.
   - endYear (YYYY) – TV Series end year. for all other title types.
   - runtimeMinutes – primary runtime of the title, in minutes.
   - genres (string array) – includes up to three genres associated with the title.

3. title.principals.tsv.gz – Contains the principal cast/crew for titles:

   - tconst (string) - alphanumeric unique identifier of the title.
   - ordering (integer) – a number to uniquely identify rows for a given titleId.
   - nconst (string) - alphanumeric unique identifier of the name/person.
   - category (string) - the category of job that person was in.
   - job (string) - the specific job title if applicable, else.
   - characters (string) - the name of the character played if applicable, else.

4. name.basics.tsv.gz – Contains the following information for names:

   - nconst (string) - alphanumeric unique identifier of the name/person.
   - primaryName (string)– name by which the person is most often credited.
   - birthYear – in YYYY format.

- deathYear – in YYYY format if applicable, else .
- primaryProfession (array of strings)– the top-3 professions of the person. knownForTitles (array of tconsts) – titles the person is known for.

5. title.ratings.tsv.gz – Contains the IMDb rating and votes information for titles:
   - tconst (string) - alphanumeric unique identifier of the title.
   - averageRating – weighted average of all the individual user ratings.
   - numVotes - number of votes the title has received.

1.2 Setting up the environments

First we have created the Spark environment for Google Colab and then, due to running time and RAM crashes reasons, we decided to use a virtual machine of Google Cloud Platform in order to get much more memory space and to speed up the computations.
The problem of this tool was that the Spark environment we have set for Colab didn't work for the virtual machine, since it works with a Jupyter Notebook. Therefore, we needed to create a new environment in order to run the notebook when the runtime is connected to the VM.

1.3 Setting up the virtual machine

We created a e2-highmem-16 virtual machine through Google Cloud Platform with 16 vCPU and 128 GB of memory, located in europe-west6-a. The persistent disk as a size of 100 GB and the operation system is Debian 10 (Linux). After the creation of this machine we connected our localhost with the jupyter notebook interface of the VM and, at the end, we also connected the Colab runtime with the jupyter notebook itself.

Figure 1.1: Characteristic of the GCP's VM

1.4 PreProcessing phase

For the task of the project and the creation of baskets, we have taken into account the following files only:

- title.basics.tsv
- title.principals.tsv
- name.basics.tsv

In order to make their identification easier we have assigned them new intuitive names, namely FilmTitle, Job and NameCast.
Since the task of the project is considering movies as baskets and actors/actresses as items, we extrapolated actors from the Job table, as shown in figure 1.2

```
actors =  """ SELECT *
              FROM Job
              WHERE (category = 'actor') OR (category = 'actress')
"""
actors = spark.sql(actors)
actors.show(10)
```

```
+---------+--------+---------+--------+---+--------------------+
|   tconst|ordering|   nconst|category|job|          characters|
+---------+--------+---------+--------+---+--------------------+
|tt0000005|       1|nm0443482|   actor| \N|      ["Blacksmith"]|
|tt0000005|       2|nm0653042|   actor| \N|       ["Assistant"]|
|tt0000007|       1|nm0179163|   actor| \N|                  \N|
|tt0000007|       2|nm0183947|   actor| \N|                  \N|
|tt0000008|       1|nm0653028|   actor| \N|    ["Sneezing Man"]|
|tt0000009|       1|nm0063086| actress| \N|["Miss Geraldine ...|
|tt0000009|       2|nm0183823|   actor| \N|    ["Mr. Hamilton"]|
|tt0000009|       3|nm1309758|   actor| \N|["Chauncey Depew ...|
|tt0000011|       1|nm3692297|   actor| \N|        ["Acrobats"]|
|tt0000014|       1|nm0166380|   actor| \N|    ["The Gardener"]|
+---------+--------+---------+--------+---+--------------------+
```

Figure 1.2: Table filtered for actors and actresses' job

and then we extrapolated movies from the table FilmTitle, as shown in figure 1.3

```
[ ] movies =  """ SELECT *
                  FROM FilmTitle
                  WHERE titleType = 'movie'
              """
    movies = spark.sql(movies)
    movies.show(10)
```

```
+---------+---------+--------------------+--------------------+-------+---------+-------+--------------+--------------------+
|   tconst|titleType|        primaryTitle|       originalTitle|isAdult|startYear|endYear|runtimeMinutes|              genres|
+---------+---------+--------------------+--------------------+-------+---------+-------+--------------+--------------------+
|tt0000009|    movie|          Miss Jerry|          Miss Jerry|      0|     1894|     \N|            45|             Romance|
|tt0000147|    movie|The Corbett-Fitzs...|The Corbett-Fitzs...|      0|     1897|     \N|            20|Documentary,News,...|
|tt0000335|    movie|Soldiers of the C...|Soldiers of the C...|      0|     1900|     \N|            \N|     Biography,Drama|
|tt0000502|    movie|            Bohemios|            Bohemios|      0|     1905|     \N|           100|                  \N|
|tt0000574|    movie|The Story of the ...|The Story of the ...|      0|     1906|     \N|            70|Biography,Crime,D...|
|tt0000615|    movie|  Robbery Under Arms|  Robbery Under Arms|      0|     1907|     \N|            \N|               Drama|
|tt0000630|    movie|              Hamlet|              Amleto|      0|     1908|     \N|            \N|               Drama|
|tt0000675|    movie|         Don Quijote|         Don Quijote|      0|     1908|     \N|            \N|               Drama|
|tt0000676|    movie|Don Álvaro o la f...|Don Álvaro o la f...|      0|     1908|     \N|            \N|               Drama|
|tt0000679|    movie|The Fairylogue an...|The Fairylogue an...|      0|     1908|     \N|           120|   Adventure,Fantasy|
+---------+---------+--------------------+--------------------+-------+---------+-------+--------------+--------------------+
```

Figure 1.3: Table filtered for movies titleType

As final step, we created the baskets joining the "Job" and "FilmTitle" tables, considering only the actors, actresses and movies. Then, we grouped by the ID of actors and actresses (tconst) and we listed the title of the films in which they played, under the 'ncosnt' column.

```
[ ] B =  """  SELECT Job.tconst, Job.nconst
              FROM Job LEFT JOIN FilmTitle ON Job.tconst = FilmTitle.tconst
              WHERE (Job.category == 'actor' OR Job.category == 'actress') AND FilmTitle.titleType == 'movie'
    """
    a = spark.sql(B)
    baskets = a.groupBy('tconst').agg(collect_set('nconst').alias('nconst'))
    baskets.createOrReplaceTempView('baskets')
    baskets.show(5)
```

```
+---------+--------------------+
|   tconst|              nconst|
+---------+--------------------+
|tt0002591|[nm0029806, nm050...|
|tt0003689|[nm0910564, nm052...|
|tt0004272|[nm0368875, nm009...|
|tt0004336|[nm0268437, nm081...|
|tt0005209|[nm0593671, nm039...|
+---------+--------------------+
```

Figure 1.4: Baskets

After having defined baskets, we create our Resilient Distributed Dataset.

```
act_bask = baskets.select('nconst').rdd.flatMap(list)
print(act_bask.collect()[:3])
```

```
[['nm0029806', 'nm0509573'], ['nm0910564', 'nm0527801', 'nm0399988', 'nm0101071', 'nm0694718', 'nm0728289', 'nm0585503'], ['nm
0368875', 'nm0092665', 'nm0492302', 'nm0445507', 'nm0776747', 'nm0383278', 'nm0192062', 'nm0285643', 'nm0793189']]
```

Figure 1.5: ResilientDistributedDataset

2 Algorithms and Implementation

In the following part, we implemented two different algorithms: the A-priori and the FP-Growth algorithm.

2.1 A-priori Algorithm

Apriori is a pretty straighforward algorithm for frequent itemset mining. The aim of the Apriori is to reduce the numbers of possible candidates to be evaluated and it is composed by different steps:

1. the process starts by computing the support for itemsets of size 1, e.i. singleton;

2. applying the minimun support , the process continues by filtering all the itemsets that do not meet the threshold;

3. moving on to itemsets of size 2, e.i. pairs, the process repeat steps 1 and 2;

4. the process continues until no additional itemsets satisfying the minimum threshold can be found, as shown in figure 2.1

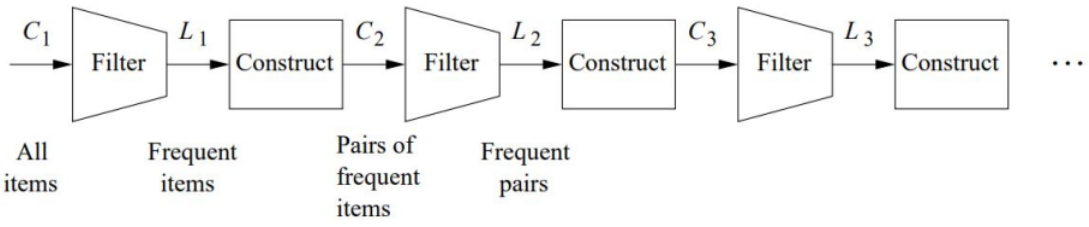

Figure 2.1: Apriori algorithm process

Focusing on our project, we firstly initialized a threshold value after which the pairs of actors and actresses are considered frequent. We started with a threshold equal to the 1% of the baskets because it is the rule of thumb. Nevertheless this value turned out to be too high and the output was an empty list, so we gradually lowered the threshold untill reaching a consistent result with 0.0003, which corresponds to 118 as minimum support.
Then, we have initialized the "apriori" function and we run it over the entire dataset using both Google Colab and Google Cloud Platform's VM, showing the results through a dataframe as follow.

```
[ ]  AprioriResults = apriori(act_bask, threshold)
     result = spark.createDataFrame(AprioriResults).toDF("Pairs of actors and actresses", "Number of movies")
     result.show()
```

```
+----------------------------+----------------+
|Pairs of actors and actresses|Number of movies|
+----------------------------+----------------+
|         {nm0623427, nm000...|             237|
|         {nm0006982, nm061...|             122|
|         {nm0006982, nm041...|             162|
|         {nm0046850, nm000...|             169|
|         {nm2082516, nm064...|             147|
|         {nm2373718, nm064...|             126|
+----------------------------+----------------+
```

Figure 2.2: Apriori Results

2.2 FP-Growth Algorithm

FP Growth is an algorithm which is available in Spark library and it's a good andvanced alternative with respect to the classic Apriori algorithm; the first step of FP-Growth is to calculate item frequencies and identify frequent items. Different from Apriori-like algorithms designed for the same purpose, the second step of FP-Growth uses a suffix tree (FP-tree) structure without generating candidate sets explicitly, which are usually expensive to generate with large datasets.
Concerning our implementation, we decided to run FP-Growth algorithm over the entire dataset considering 0.0003 as support threshold, as we did before for the Apriori algorithm.
In figure 2.3 are displayed the ten most frequent items.

items	freq
[nm1388202]	153
[nm0430646]	120
[nm0103977]	798
[nm0006982]	585
[nm0436922]	152
[nm0408381]	120
[nm0648803]	565
[nm0405977]	152
[nm0576495]	120
[nm0579663]	120

Figure 2.3: FP-Growth 10 most frequent singletons

Then, we tried to retrieve the most frequent pairs, and the results are shown below in figure 2.4. As we expected, the results are the same of our Apriori algorithm.

items	freq
[nm0623427, nm000...	237
[nm0046850, nm000...	169
[nm0419653, nm000...	162
[nm2082516, nm064...	147
[nm2373718, nm064...	126
[nm0619779, nm000...	122

Figure 2.4: FP-Growth most frequent pairs

3 Scalability and performance

First, we tried to run the A-Priori algorithm through Colab and it took 59 seconds to complete the computations and 1h and 21 minutes for showing the results (due to the .collect() action operation), and then we run the same code through the virtual machine and it took 34 seconds for computations and 37 minutes for displaying the dataframe.
After that, we run both A-Priori and FP-Growth using different threshold sizes first on Colab and then on the VM, putting all the results in a unique graph, as shown below.

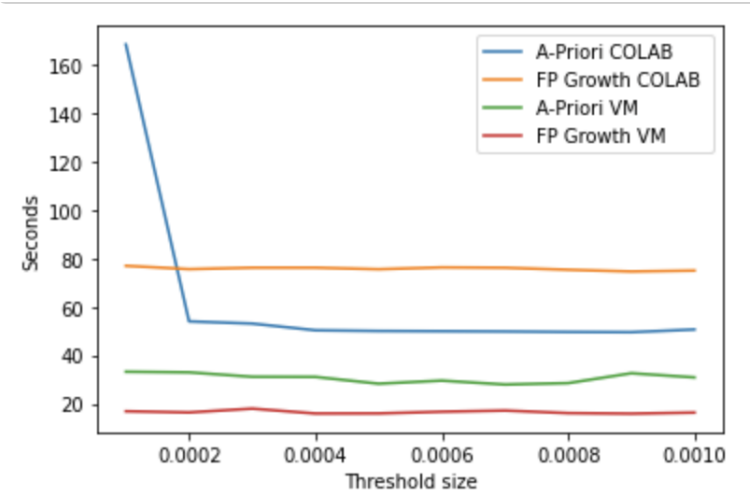

Figure 3.1: Apriori and FP-Growth run on Colab and GCP

As expected, using our virtual machine the computation took much less time with respect to the standard Google Colab.
Subsequently, we run only the A-Priori algorithm using, as usual, 0.0003 as threshold value but varying the size of the sample of the dataset, comparing the running time between Colab and GCP's VM. As we can see from the graph the time needed by the VM is more or less 50% lower than Colab.

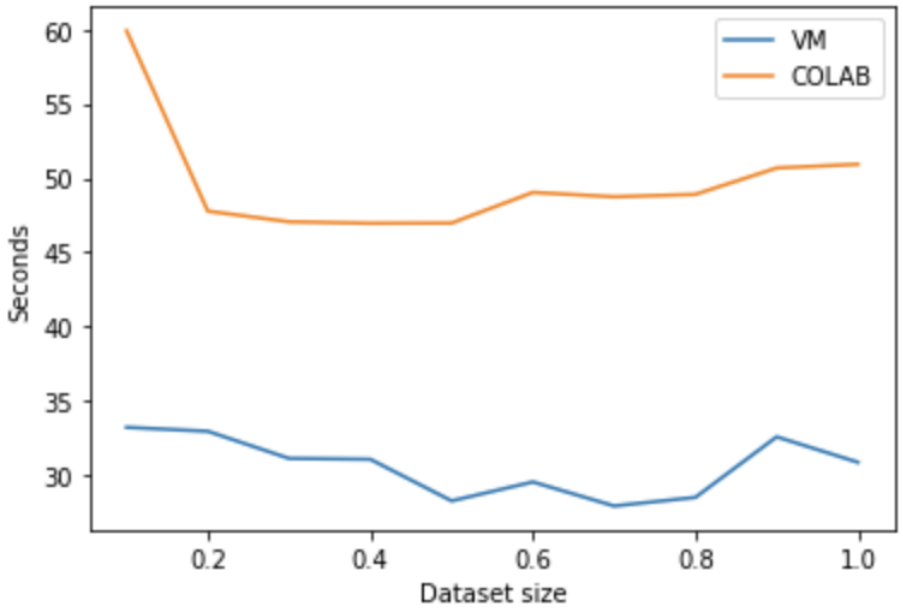

Figure 3.2: Apriori run on Colab and GCP

In the end, the last graph shows us that the FP-Growth algorithm, run through the virtual machine, is the 80% faster than the performance of Google Colab.

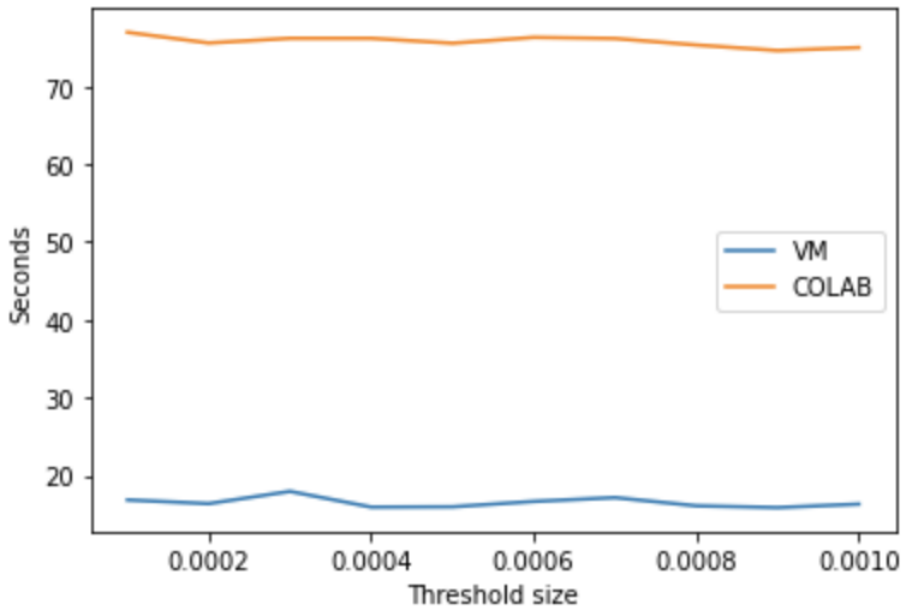

Figure 3.3: FP-Growth run on Colab and GCP

4 Conclusion

The aim of our project was to find frequent itemsets (actors and actresses) that appear together in more than a certain value of films (baskets). We can say that we have reached our goals since both the A-Priori and the FP-Growth algorithms retrieved same results, using a threshold value equal to 0.0003 and the whole dataset, as shown in figures 2.2 and 2.4. Due to the time needed by Colab for running our algorithms, we decided to use the computational power of a virtual machine in order to speed up the calculations. Throughout our analysis, the first thing to point out has been the choice of the threshold value, which is fundamental for getting a consistent result. Moreover, we also performed a scalability analysis using Colab and the GCP's VM in order to understand which was the better way for getting the same results with less running time: as shown before through figure 3.1 the virtual machine is the best choice due to its larger RAM memory space, as expected.

5 Declaration

We declare that this material, which We now submit for assessment, is entirely our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of our work. We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by us or any other person for assessment on this or any other course of study.