Università degli Studi di Milano

Data Science and Economics

Algorithms for Massive Data

Market Basket Analysis

Implementation of APriori and FP-Growth Algorithms on IMDB dataset

Antonella D'Amico 961150
Luca Romano 980068

Academic year 2020/2021

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.

Contents

1 Introduction **3**

2 Dataset description **3**

3 Pre-processing phase **5**

4 Algorithms implementation **6**

4.1 A-Priori Algorithm 6

4.2 FP-Growth Algotithm 7

5 Results **7**

5.1 A-Priori Algorithm 7

5.2 FP-Growth Algorithm 8

5.3 Comparison 9

List of Figures

2.1 title.principals table 3
2.2 title.basics table 4
3.1 Title.principals filtered for actors/actresses 5
3.2 Title.basics filtered for movie 5
3.3 Baskets 5
4.1 A-Priori process 6
5.1 Results A-Priori with threshold 0.0003 7
5.2 Apriori for threshold between 0.0003 and 0.0008 on 25% size sample 8
5.3 Time over number of baskets with fixed threshold 8
5.4 FP-Growth frequent singletons reults (sample of) 9
5.5 FP-Growth frequent pairs result 9
5.6 Apriori and FP-Growth by varying thresholds on 25% sample size baskets 10
5.7 Time-sample size to compare scalability between the two algorithms 10

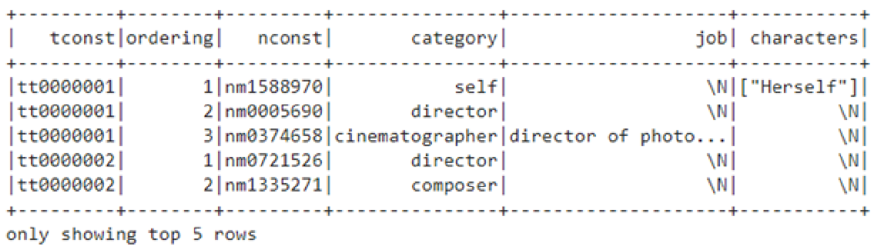
Introduction

The aim of the project is to perform a Market-Basket-Analysis on the IMDb dataset stored in Kaggle repository[1]. The dataset contains information related to Titles and Crew/cast.
The goal of market basket analysis is to find frequent itemsets, those items that are more likely to appear together in the same baskets for a given number of times. In this project we consider movie as baskets and actors as items.
In order to perform our analysis we implemented an Apriori algorithm in Spark.

Dataset description

The dataset used is IMDb stored in Kaggle repository. It contains information about movies, tv series, videos and information related to cast and crew.
For our purpose we only consider a subset of tables:

- *title.principals* which contains information related to cast/crew about titles. The table is composed by the following variables:
  - *tconst*: titles unique ID;
  - *ordering*: number to uniquely identify rows for a given title ID;
  - *nconst*: name/person unique ID;
  - *category*: job role that person in the cast/craw was in;
  - *job*: the specific job title;
  - *characters*: the name of the character played by the actor/actress

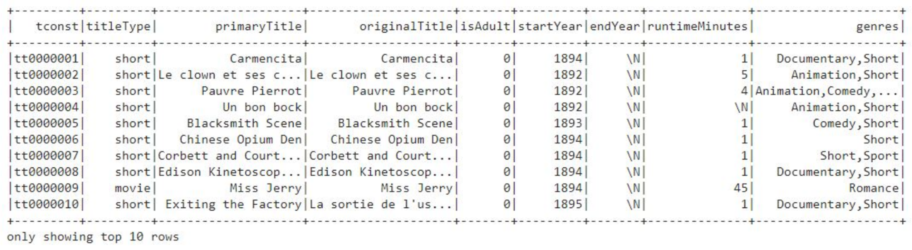


```
+---------+--------+---------+---------------+--------------------+-----------+
|   tconst|ordering|   nconst|       category|                 job| characters|
+---------+--------+---------+---------------+--------------------+-----------+
|tt0000001|       1|nm1588970|           self|                  \N|["Herself"]|
|tt0000001|       2|nm0005690|       director|                  \N|         \N|
|tt0000001|       3|nm0374658|cinematographer|director of photo...|         \N|
|tt0000002|       1|nm0721526|       director|                  \N|         \N|
|tt0000002|       2|nm1335271|       composer|                  \N|         \N|
+---------+--------+---------+---------------+--------------------+-----------+
only showing top 5 rows
```

Figure 2.1: title.principals table

- *title.basics* contains the following information about titles:
  - *tconst*: titles unique ID;
  - *titleType*: format of the title;

- *primaryTitle*: popular title;
- *originalTitle*: original title in original language;
- *isAdult*: if the title is for adult public;
- *startYear*: release year of the title;
- *endYear*: end years just for TV series;
- *runtimeMinutes*: duration of the title;
- *genres*: genres associated with the title

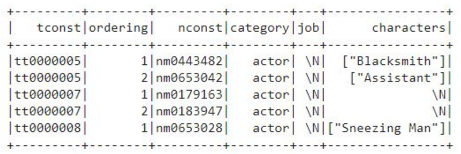


```
+---------+---------+--------------------+--------------------+-------+---------+-------+--------------+--------------------+
|   tconst|titleType|        primaryTitle|       originalTitle|isAdult|startYear|endYear|runtimeMinutes|              genres|
+---------+---------+--------------------+--------------------+-------+---------+-------+--------------+--------------------+
|tt0000001|    short|          Carmencita|          Carmencita|      0|     1894|     \N|             1|   Documentary,Short|
|tt0000002|    short|Le clown et ses c...|Le clown et ses c...|      0|     1892|     \N|             5|     Animation,Short|
|tt0000003|    short|      Pauvre Pierrot|      Pauvre Pierrot|      0|     1892|     \N|             4|Animation,Comedy,...|
|tt0000004|    short|         Un bon bock|         Un bon bock|      0|     1892|     \N|            \N|     Animation,Short|
|tt0000005|    short|    Blacksmith Scene|    Blacksmith Scene|      0|     1893|     \N|             1|        Comedy,Short|
|tt0000006|    short|   Chinese Opium Den|   Chinese Opium Den|      0|     1894|     \N|             1|               Short|
|tt0000007|    short|Corbett and Court...|Corbett and Court...|      0|     1894|     \N|             1|         Short,Sport|
|tt0000008|    short|Edison Kinetoscop...|Edison Kinetoscop...|      0|     1894|     \N|             1|   Documentary,Short|
|tt0000009|    movie|          Miss Jerry|          Miss Jerry|      0|     1894|     \N|            45|             Romance|
|tt0000010|    short| Exiting the Factory|La sortie de l'us...|      0|     1895|     \N|             1|   Documentary,Short|
+---------+---------+--------------------+--------------------+-------+---------+-------+--------------+--------------------+
only showing top 10 rows
```

Figure 2.2: title.basics table

Pre-processing phase

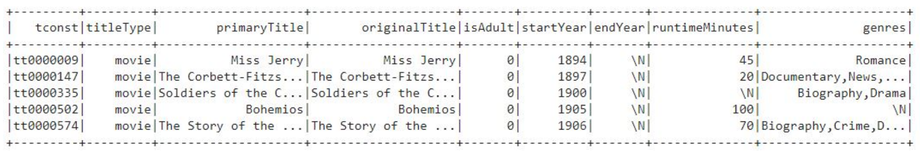
From the subset of tables described in the previous section, we have considered only some variables in order to implement our algorithm.
In title.principals table we filtered over the column *category* for **actor** and **actress** values, since they will represent our items.



```
+---------+--------+---------+--------+---+----------------+
|   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"]|
+---------+--------+---------+--------+---+----------------+
```

Figure 3.1: Title.principals filtered for actors/actresses

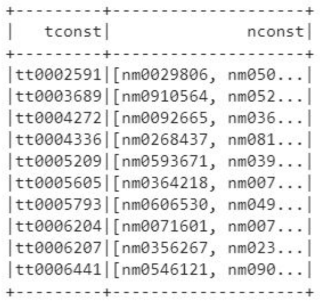
From title.basics table we filtered over the column *titleType* just for **movie** values over which we will build our baskets.



```
+---------+---------+--------------------+--------------------+-------+---------+-------+--------------+--------------------+
|   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...|
+---------+---------+--------------------+--------------------+-------+---------+-------+--------------+--------------------+
```

Figure 3.2: Title.basics filtered for movie

Once we performed this cleaning phase, we created our baskets by grouping actors/actresses for each movie.
We consider only two variables: *nconst* and *tconst* which, as said before, represent the unique ID for actors/actresses and movies.



```
+---------+--------------------+
|   tconst|              nconst|
+---------+--------------------+
|tt0002591|[nm0029806, nm050...|
|tt0003689|[nm0910564, nm052...|
|tt0004272|[nm0092665, nm036...|
|tt0004336|[nm0268437, nm081...|
|tt0005209|[nm0593671, nm039...|
|tt0005605|[nm0364218, nm007...|
|tt0005793|[nm0606530, nm049...|
|tt0006204|[nm0071601, nm007...|
|tt0006207|[nm0356267, nm023...|
|tt0006441|[nm0546121, nm090...|
+---------+--------------------+
```

Figure 3.3: Baskets

Algorithms implementation

4.1 A-Priori Algorithm

The aim of the **A-Priori Algorithm** is to reduce the numbers of possible candidates to be evaluated.
This action is performed by exploiting the *monotonicity property* which states that *"If a set I is frequent, each subset of I ($J \subseteq I$) needs to be frequent"*.
In this way we are able to filter "a priori" the candidates and reduce the number of possible elements to be evaluated.

The process starts by computing the singletons whose frequency is greater than the *minimum support* and then the output of the first pass is used to compute the possible candidates of the second pass and so on.
More in depth, the pairs are produced in the second pass by combining frequent singletons. From the 3^{rd} pass ($i = 3$):

1. The output generated from the previous pass is saved into an RDD;
2. New singletons are produced from this output and saved into another RDD;
3. The candidates are created by performing the cartesian product between the new singletons and the output;
4. Possible candidates are created by checking whether the size of the set of the new candidates is equal to i. Duplicates and permutations are removed;
5. Frequency is incremented each time the possible candidate is a subset of a basket;
6. Filter the possible candidates whose frequency is greater than minimum threshold, overwrite the output RDD and add them also to the output of the previous passes.

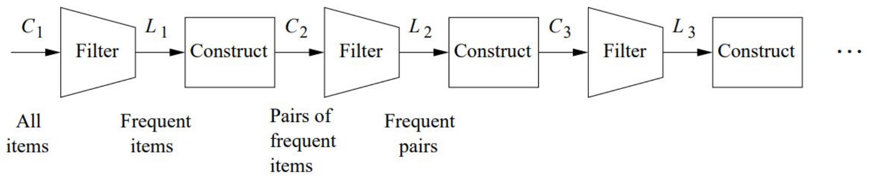


Figure 4.1: A-Priori process

(a) source: Mining of Massive Datasets

If the last pass did not produce any frequent item, the algorithm stops printing the frequent items found in previous passes, otherwise repeats the cycle incrementing *i*. The minimum support is calculated as

$$minimum\ support = threshold * number\ of\ baskets$$

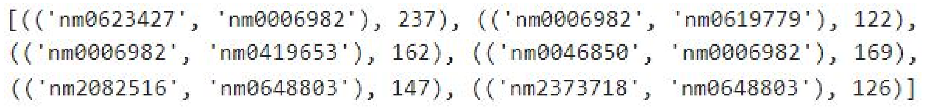
4.2 FP-Growth Algotithm

The FP-Growth is an algorithm which is available in Spark library and it is considered as a development of A-Priori algorithm. The main difference between the two algorithms is that the FP-Growth is able to search for frequent itemsets through the concept of tree development without generating candidate itemsets. This is the strength of FP-Growth with respect to A-Priori and this is why the first one is faster.

Results

5.1 A-Priori Algorithm

Initially we performed some tests with threshold 0.01 but 3936 as minimum support was too high and did not provide any frequent item.
After some more tests we lowered the *threshold* down to 0.0003, which corresponds to 118 as minimum support. This value gave us more relevant results as shown in 5.1:



```
[(('nm0623427', 'nm0006982'), 237), (('nm0006982', 'nm0619779'), 122),
(('nm0006982', 'nm0419653'), 162), (('nm0046850', 'nm0006982'), 169),
(('nm2082516', 'nm0648803'), 147), (('nm2373718', 'nm0648803'), 126)]
```

Figure 5.1: Results A-Priori with threshold 0.0003

We also tried to lower it down even more, but the amount of time required with lower supports was too much for the operational times allowed by Google Colab so to check the behavior of the algorithm with different threshold we used a 25% size sample of the total baskets using the `sample` function by Pyspark.

The result obtained by varying the *threshold* between 0.0003 and 0.0008 on a 25% size sample of the baskets shows an exponential relation between time and the different support.

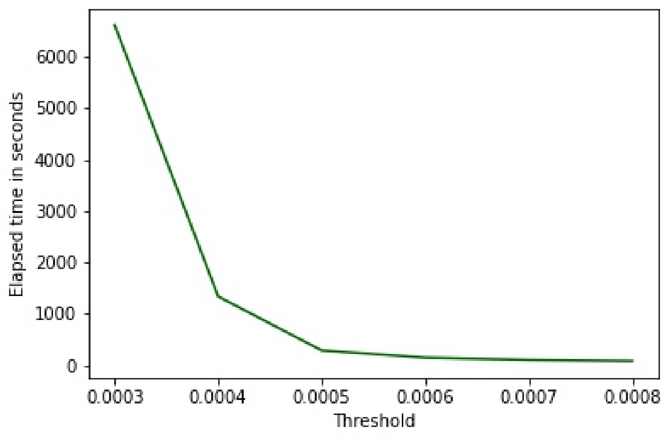


Figure 5.2: Apriori for threshold between 0.0003 and 0.0008 on 25% size sample

Considering the evolution of a fixed threshold and the increasing size of the sample we can check the behavior of the algorithm as we scale the dataset to greater sizes and, as can be seen in 5.3, it follows a linear trend.

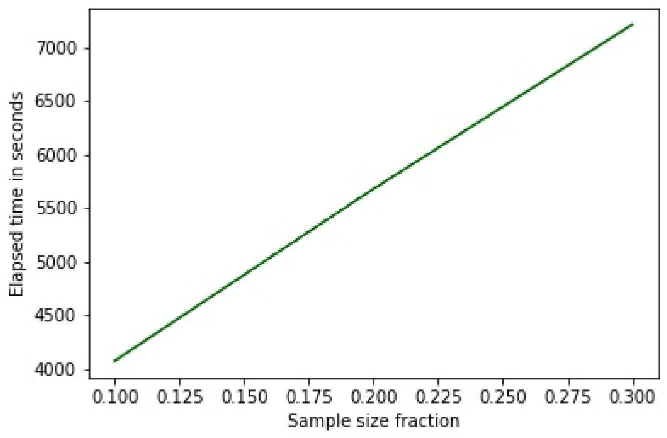
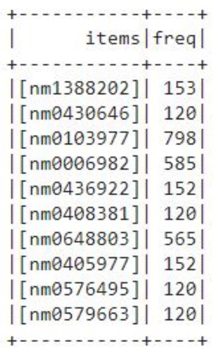


Figure 5.3: Time over number of baskets with fixed threshold

5.2 FP-Growth Algorithm

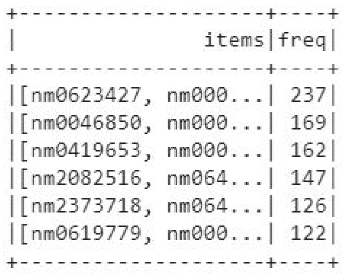
Regarding the analysis performed using FP-Growth algorithm we decided to run it over the whole dataset by setting the minimum support equal to 0.0003. In 5.4 are displayed a sample of 10 frequent obtained singletons.



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

Figure 5.4: FP-Growth frequent singletons reults (sample of)

Then, we decided to query the obtained results in order to retrieve the more frequent pairs. As expected the result is the same of our A-Priori algorithm.



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

Figure 5.5: FP-Growth frequent pairs result

5.3 Comparison

As reference we compared the results obtained with our implemented A-Priori algorithm with respect to the FP-Growth from Pyspark.
As shown in 5.6 the two performances are very different; the one provided by the Pyspark library is much faster and optimized considering that it requires almost the same amount of time regardless of the threshold used.

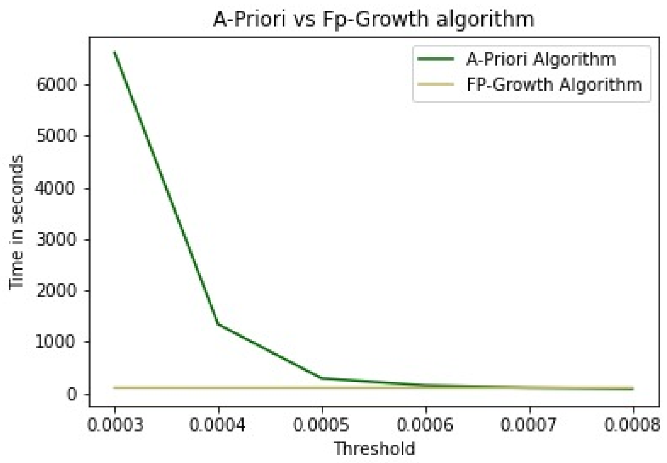


Figure 5.6: Apriori and FP-Growth by varying thresholds on 25% sample size baskets

This result is consistent also in the scaling of the sample size: while in our algorithm the sample size is linearly dependent with time, the FP-Growth ensures a constant time execution time no matter the size of the sample.

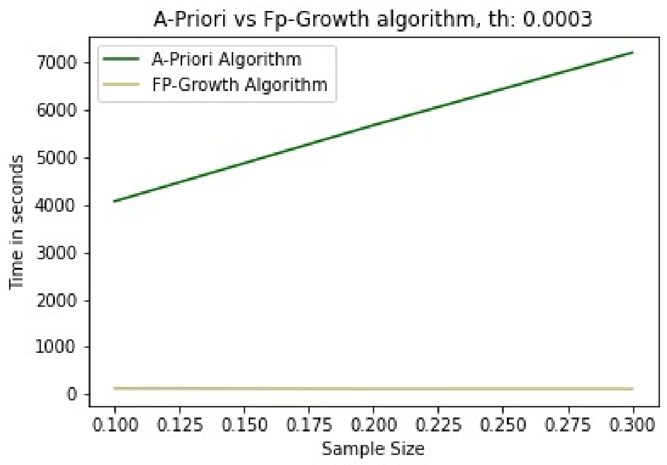


Figure 5.7: Time-sample size to compare scalability between the two algorithms

References

[1] `https://www.kaggle.com/ashirwadsangwan/imdb-dataset`

[2] J.Leskovec, A.Rajaraman, J.Ullman, *Mining of Massive Datasets*, (2011)

[3] Lecture Notes