Università degli Studi di Milano

Data Science and Economics

Algorithms for Massive Data

Market-Basket Analysis

A-priori and SON algorithms on IMDb database

Authors:

Emanuele Morales - 941935

Federico Pessina - 944886

A.Y. 2020/2021

Contents

1 Introduction

The aim of this project is to perform a Market-Basket analysis on the IMDb dataset, an Amazon website that manages information about movies, directors, staff, TV series and video games.

In order to perform a frequent item set analysis, finding the sets of items (actors) that appear in many of the same baskets (movies), we implemented in Spark our version of A-Priori and SON algorithm.

The first algorithm works by eliminating most large subsets as candidates by looking at smaller sets and recognizing that a large set cannot me frequent unless all its subset are (*monotonicity property*).

Starting from this algorithm we extended our analysis by applying it in the SON algorithm context, an algorithm that instead of processing all the baskets together, analyzes chunks of baskets together.

This kind of analysis returns the actors that are more likely to act together in the same movies.

Finally we also calculated the confidence of the frequent actors that says how likely actor Y appears when actor X also appears.

The use of spark is due to the huge amount of movies and actors that are involved in the analysis. The use of Spark moreover allows to create a solution that scales up with the size of data.

The link to the github can be found here: GITHUB

The link to the colab notebook can be found here: COLAB

2 Data

The dataset is available HERE. We considered a subset of the available tables. In particular we used:

- **title.principals** – Contains the principal cast/crew for titles:

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

  We are interested in:

  - tconst (string) - alphanumeric unique identifier of the title
  - nconst (string) - alphanumeric unique identifier of the name/person
  - category (string) - identify the category of the crew (actor, directors, ...). We are interested in actors/actress.

- **title.basic** - Other information about the titles:

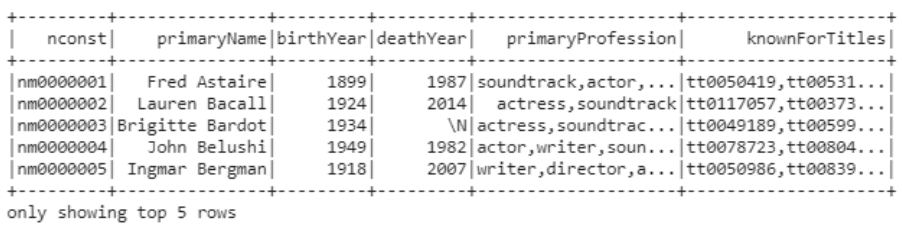
tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	genres
tt0000001	short	Carmencita	Carmencita	0	1894	\N	1	Documentary,Short
tt0000002	short	Le clown et ses c...	Le clown et ses c...	0	1892	\N	5	Animation,Short
tt0000003	short	Pauvre Pierrot	Pauvre Pierrot	0	1892	\N	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

only showing top 5 rows

  We are interested in:

  - tconst (string) - alphanumeric unique identifier of the title.
  - titleType (string) - identify the type of title (movies, short, tvseries...). We are interested in movies.

- **name.basics** – Contains information about the actors:



nconst	primaryName	birthYear	deathYear	primaryProfession	knownForTitles
nm0000001	Fred Astaire	1899	1987	soundtrack,actor,...	tt0050419,tt00531...
nm0000002	Lauren Bacall	1924	2014	actress,soundtrack	tt0117057,tt00373...
nm0000003	Brigitte Bardot	1934	\N	actress,soundtrac...	tt0049189,tt00599...
nm0000004	John Belushi	1949	1982	actor,writer,soun...	tt0078723,tt00804...
nm0000005	Ingmar Bergman	1918	2007	writer,director,a...	tt0050986,tt00839...

only showing top 5 rows

We are interested in:

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

From this table we will retrieve the names of the actors in output from the application of the algorithms.

3 Preprocessing

From title.principals table, which contains the keys for each productions (movies, tv series, video game,etc.) and the cast and the crew that participated to the production, we filtered the tuples categorised as 'actor' and 'actress', excluding 'producer', 'directors' etc., and we considered the two columns of interest (the code of the production and the related actor/actress).

To filter out from the title.principals table just the movies that will represent the baskets in the analysis, we imported the title.basics table, which contains the code of the productions and the categories to which they belong. From this table we considered only the tuples categorised as 'movie', excluding other categories such as 'short', 'video games' etc, obtaining a list of 536248 movies.

From the title.principals table we considered just the movies obtained from the title.basic table, getting a table containing the code of the movies and the related actors. Each movie can appear in more than one row, depending on the number of actors that performed in.

All these operations have been performed by using the module of Spark "SparkSQL", that is used for structured data processing. The output of this pre-preocessing operation is the following one, where "tconst" represents the movies and "nconst" the related actors:

```
+---------+---------+
|   tconst|   nconst|
+---------+---------+
|tt0002591|nm0029806|
|tt0002591|nm0509573|
|tt0003689|nm0694718|
|tt0003689|nm0101071|
|tt0003689|nm0910564|
+---------+---------+
only showing top 5 rows
```

Finally, all the actors performing in the same movies are grouped in the same list, obtaining the basket-by-basket configuration with data in a distributed file system. It follows a print of two movies with the related actors.

```
[['nm0029806', 'nm0509573'], ['nm0910564', 'nm0527801', 'nm0399988', 'nm0101071', 'nm0694718', 'nm0728289', 'nm0585503']]
```

Each lists represents a movie that contains actors, represented with their code. Here the character "[" begins a basket and the character "]" ends it.

4 Algorithms implementation

4.1 A-Priori

We are interested to find the pairs of actors that appear together frequently in movies. However, generating all the combinations of two actors that are in the dataset could be too much demanding. A-priori algorithm reduces the number of pairs that must be counted by exploiting the monotonicity property: *"if a set I of items is frequent, then so is every subset of I"*.

Therefore, in order to discover all the frequent pairs of actor, it is not necessary to analyze all the pairs of actors in the dataset, but it is sufficient to consider the pairs of actors taking into account only the performers that also singularly appear frequently. The notion of frequent in the algorithm is defined by the *support threshold*. An item, or a set of items, is frequent if it appears in baskets a number of times larger than the support threshold threshold.

We then created a function called *a-priori* that performs the algorithm, taking as input a RDD object (in our case the set of movies containing the actors) and a minimum threshold to identify the actors that are frequent.

The algorithm is performed by combining the main Spark's functions: *map*, *reduce* and *filter*.

The first step consists in finding the actors that appear frequently singularly. In order to do this, we applied:

- *map*: a map function that for each actor appearing in the RDD returns a tuple *(actor_code, 1)*.
- *reduceByKey*: a reduce function that sums all the tuples that have as key the same actor code.
- *filter*: a filter function to filter out the actors that performs a number of time less than the threshold value.

This procedure returns a list of actors that are frequent singularly.

In the second step of A-priori, we counted all the pairs that consist of two frequent items, by creating a list containing all the possible combinations between the frequent actors in output from the previous step.

Starting from this list of pairs of actors a new *map* - *reduce* - *filter* pipeline is implemented.

- *map*: a map function that scans the RDD and for each pair of actor returns a tuple *((actor1_code, actor2_code), 1)*.
- *reduceByKey*: a reduce function that sums all the the tuples that have as key the same pairs of actor codes.
- *filter*: a filter function to filter out the pair of actors that performed together a number of time less than the threshold value.

We used the *apriori* function implemented above in the *SON algorithm*.

4.2 SON algorithm

SON algorithm is an exact algorithm that removes *false negative* and *false positive* and that applies properly in a context in which data are distributed in chunks. Since we implemented this analysis in Spark, that split data in RDD, this algorithms is appropriate.

Starting from the basket-by-basket configuration, we split data in five chunks. We are interested in the pairs of actor that performed together in more than 140 movies, so the support threshold is set to 140. Since we are working separately on each chunk, this threshold has to be adjusted, by dividing it by the number of chunks. Therefore the adjusted threshold in our case takes the value of $\frac{140}{5} = 28$.

The first step consists in read each sample and, applying the *apriori* function previously create, and find the frequent pairs of item.

The results of each chunk are then merged, avoiding duplicates, creating a set of candidates. :

However this set of candidate could contain *false positive*, that means some itemsets that are frequent in the chunks but not frequent considering the whole dataset together.

In order to remove false positive, a scan is performed on the baskets-by-baskets file, counting the occurrences of the itemsets in the baskets.

The itemsets that appear a number of time less than the support threshold are filtered out. In our case, no pairs revealed to be false positive. In the following table are reported the number pairs of actors that performed together more than 140 times and the number of their appearance together.

actor_pair	movies
{nm0623427, nm0006982}	236
{nm2082516, nm0648803}	147

5 Results

Combining these results with the table containing the information about actors, it is possible to obtain the actual name of the actors.

From the first step of a-priori algorithm we also retrieve the number of movies performed singularly by the actors, in order to compute confidence.

primaryName	actors	movies
Kijaku Ôtani	nm2082516	161

primaryName	actors	movies
Matsunosuke Onoe	nm0648803	565

primaryName	actors	movies
Prem Nazir	nm0623427	436

primaryName	actors	movies
Adoor Bhasi	nm0006982	585

It is possible to calculate the confidence that says how likely actor Y appears when actor X also appears:

$$Confidence\{X \rightarrow Y\} = \frac{Support\{X, Y\}}{Support\{X\}}$$

$$Confidence\{Otani \rightarrow Onoe\} = \frac{147}{161} = 91\%$$

$$Confidence\{Onoe \rightarrow Otani\} = \frac{147}{565} = 26\%$$

$$Confidence\{Bhasi \rightarrow Nazir\} = \frac{236}{436} = 40\%$$

$$Confidence\{Nazir \rightarrow Bhasi\} = \frac{236}{585} = 54\%$$

6 Disclaimer

We declare that this material, which I/We now submit for assessment, is entirely my/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 my/our work. I/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 me/us or any other person for assessment on this or any other course of study.