Algorithms for massive datasets

Finding Frequent Itemsets

Report

by Maria Giuseppina Brunelli

Student number: 960160

Email: mariagiuseppinabrunelli@studenti.unimi.it

Contents

1 Introduction

The purpose of this project is to implement "Market Basket Analysis" for the IMDB dataset with the goal of finding frequent itemsets, in particular, by considering movies as baskets and actors as items. In this context therefore, an itemset, represented by a single actor or a set of them, is considered frequent if it appears at least in a number (called support) of movies, higher than the minimum support value, whereas in a traditional "Market Basket Analysis", food itemsets are considered frequent if they appear in number of baskets at least equal to the mimimum support value.

2 Dataset

The dataset used in this project, made available on Kaggle.com, is the IMDB dataset, which contains information about movies retrieved from the IMDB website, in particular, the dataset contains the following files:

• **title.akas.tsv.gz**: contains information about movies' titles such as title, region, language

• **title.basics.tsv.gz**: contains other information about movies' titles such as the identifier, type of title, primary title, original title, generes

• **title.principals.tsv.gz**: contains the principal cast/crew for titles

• **title.ratings.tsv.gz**: Contains the IMDB rating and votes information for titles

• **name.basics.tsv.gz**:contains information about names, such as identifier, primary name, profession, titles the person is known for

For the purpose of this project two files of the dataset have been used, in particular:

• **name.basics.tsv.gz**

• **title.basics.tsv.gz**

3 Data Organization

As stated above the first file "name.basics.tsv.gz" contains information about all the people who took part in specific movies, in particular the dataset appears as shown in Table 1. The second file, "title.basics.tsv.gz", instead, contains information about movies' titles as shown in Table 2.

nconst	primaryName	birthYear	deathYear	primaryProfession	knownForTitles
nm0000001	Fred Astaire	1899	1987	soundtrack,actor,..	tt0050419,tt00531,..

Table 1: Name basics

tconst	titleType	primaryTitle	originalTitle	startyear	endYear	genres
tt0000001	short	Carmencita	Carmencita	1894		Documentary, Short

Table 2: Movies basics

Since the goal of this project is to find frequent itemsets, by considering movies as "baskets" and actors as "items", a restructuring of the two tables was necessary, in such a way that we could obtain a resulting table showing for each movie, the list of actors that took part in it. This operation was done by exploiting PySpark SQL functions, which allowed to "explode" the list of movies of Table 1 and then to "groupby" the title of each movie. The result of this operation is shown in Table 3

tconst	PrimaryNames	nconst
tt0005605	Minnie Berlin,...	nn0075601, ...

Table 3: Baskets: movies - Items: actors

4 Preprocessing techniques

Before using the data to derive the final datasets used in the project (shown in Table 3), simple pre-processing techniques have been applied to the original data. Specifically, any null value was removed, as well as all the columns available in the original files which were irrelevant for the purpose of the project.

Moreover, after a careful analysis of the data, it appeared that some records in the **primaryProfession**

field of Table 1, were populated with different types of professions, as for example "soundtack", "writer", "camera department", "visual effects" and of course "actor or actress". Here, as said, the primary focus is on actors, however, as it is known, there are cases in which actors take also other roles in the same movie (for example some actors are also directors of the same movie), while other actors are strictly actors. Therefore in order to take this into account, two cases were finally considered:
- **actors**: from the original data, the records containing only the string "actor" or "actress" in the primaryProfession field were kept
- **actors with other roles**: from the original data, the records containing at least the string "actor" or "actress" in the primaryProfession field were kept.

As a result two different datasets of the form shown in Table 3 were created, one containg, for each movie title, the list of people with only actor/actress role (which will be referred to, in the rest of this report as "Actors only"); the second one containing, for each movie title, the list of people with more than actor/actress role (which will be referred to, in the rest of this report as "Actors with other roles").

5 Algorithms

In order to find frequent itemsets, two algorithms have been considered, in particular the Apriori algorithm and the FP-growth algorithm. The first one has been implemented from scratch, while the second one was already available in the "MLlib" Spark library.

5.1 Apriori Implementation

The Apriori aglorithm takes in input the file of transactions/baskets and a minimum support value, and returns a list of frequent itemsets, with the respective frequency. The algorithm implementation could be described in two main steps: for the sake of the description these will be referred to as **Step-One** and **Step-Two**.

Step-One The first step includes three sub-steps:

1. Each item is associated with a counter initialized to one (one occurence). As a result we obtain a pair (item, 1)

2. The counters initialized to 1, associated to each item in the previous step, are summed for each item. As a result, we obtain a pair (item, occurrencesCounter)

3. Retain only the items that have occurrencesCounter equal or higher than the minimum support value given in input. As a result, we obtain the list of frequent items with their respective frequency.

Step-Two In this step:

1. Frequent items retrived in Step-One, are combined in pairs, to form the list of candidate pairs.

2. For each candidate pair, the algorithm checks if the pair is a subset in each basket and if so it increases the respective frequency counter.

3. If the calculated frequency for each candidate pair is equal or higher than the choosen support value, the candidate pair becomes a frequent pair.

The last three substeps are iterated in the same way to generate candidate triples and check their frequency, and so on with quadruples, quintuples, until there are no frequent itemsets left to be combined together in order to form other candidates.

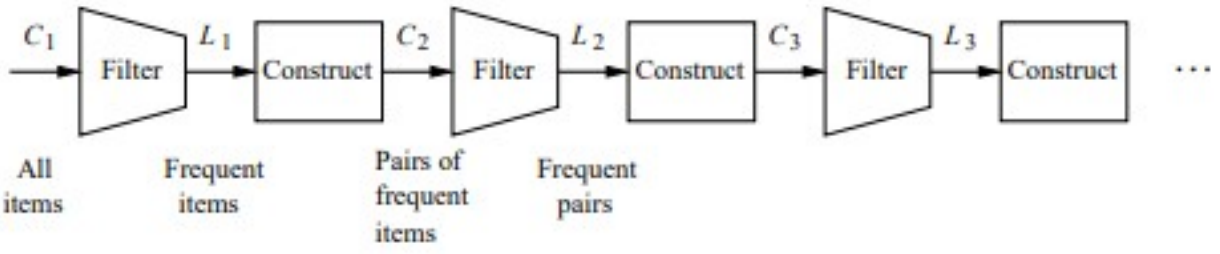

Figure 1: A-priori steps

5.2 FP-growth

Fp-growth algorithm takes as input a dataset of transactions and a minimum support value. In the first step, it calculates item frequencies and checks theses frequencies against the miminum support value (passed in input). In the second step of FP-growth uses a suffix tree (FP-tree) structure

to encode transactions without generating candidate sets explicitly. After the second step, the frequent itemsets can be extracted from the FP-tree.

6 Solution scaling

The algorithms used in this project can be applied also to bigger datasets, in other words the proposed solutions scale up with data size.

In particular, the A-priori algorithm, per its nature scales up by filtering the data according to the definition of "Monotonicity of Itemsets". This definition, states that: "If a set I of itemset is frequent, then so is every subset of I". This means that if we are given a minimu support value s, then we say that an itemset is maximal if no superset is frequent. If we obtain the list of maximal itemsets, then we know that all subsets of a maximal itemset are frequent, and on the other hand subsets of non maximal itemsets cannot be frequent.

Therefore at each step of the algorithm, it is possible to discard all the subsets that have a support lower than the minimum one, since we are sure that when those itemsets will increase, the respective support will be even lower. Moreover, this particular implementation of the Apriori algorithm, scales up with data size because of the use of a parallelized computation. Thanks to the use of Spark, indeed, the computation is split in tasks distirbuted across different worker nodes, whereas executors run on these nodes and execute the task assigned to them.

The second algorithm, considered in this projects, also scales up with data size thanks to the implementation of a parallel version of FP-growth that distributes the work of growing FP-trees based on the suffixes of transactions, and therefore is more scalable than a single-machine implementation.

7 Experiments

As described above, after the distinction between people that play only the actor role and those who play also other roles in movies, two datasets have been defined: "Actors only" and "Actors with other roles".

In order to decide the minimum support value to use in each of the two cases, a preliminary analysis was run to determine the number of movies appearances of each actor/actress. In particular, for the first case, most actors, 70% circa, appear in at least one movie, while many others appear in at least 4 movies, as shown in figure 2. Since the goal of the project is to find frequent itemsets, if most actors appear in 4 movies or less they cannot be regarded as frequent, therefore it seems reasonable that, to find frequent itemsets, the minimum support value should be a value at least higher than 4.

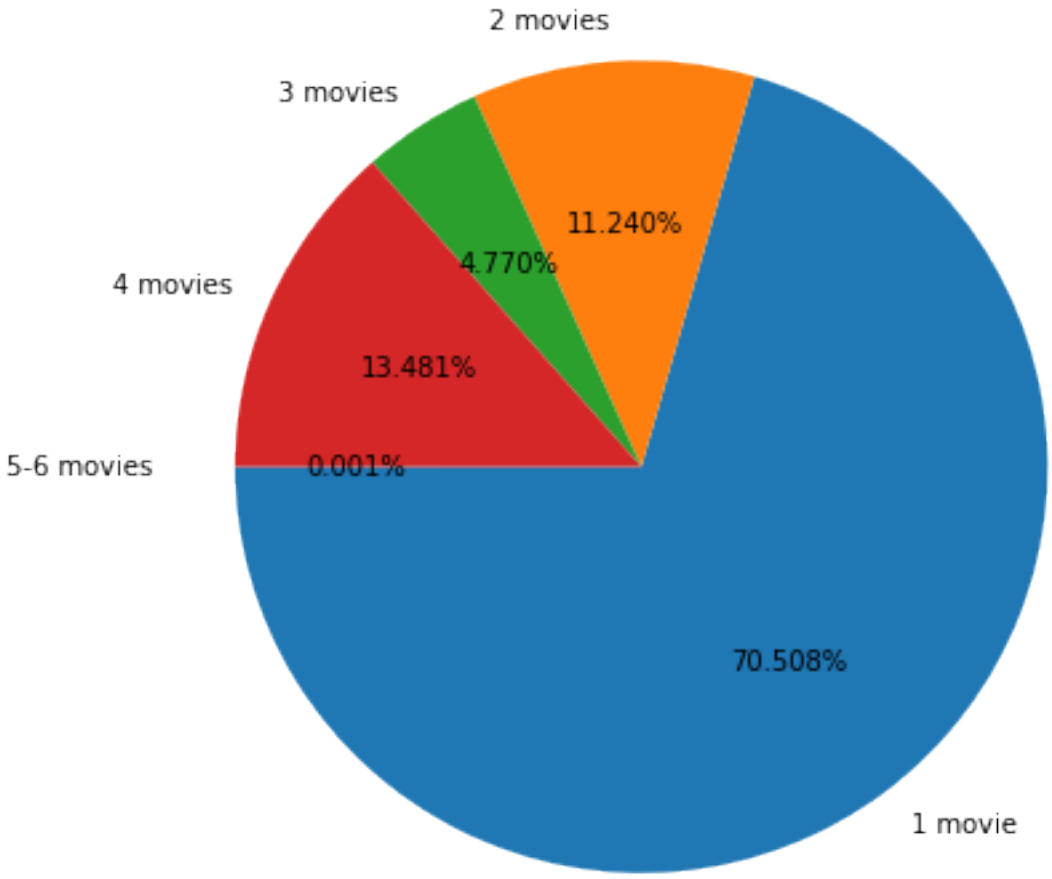

Figure 2: Number of movies appearances - "Actors only"

The same reasoning applies to the second case, where the number of appearances follows more or less the same proportions of the first case, but with even more actors that appear in 4 movies, as

shown in figure 3. Also in this case therefore, the minimum support value should be higher then 4.

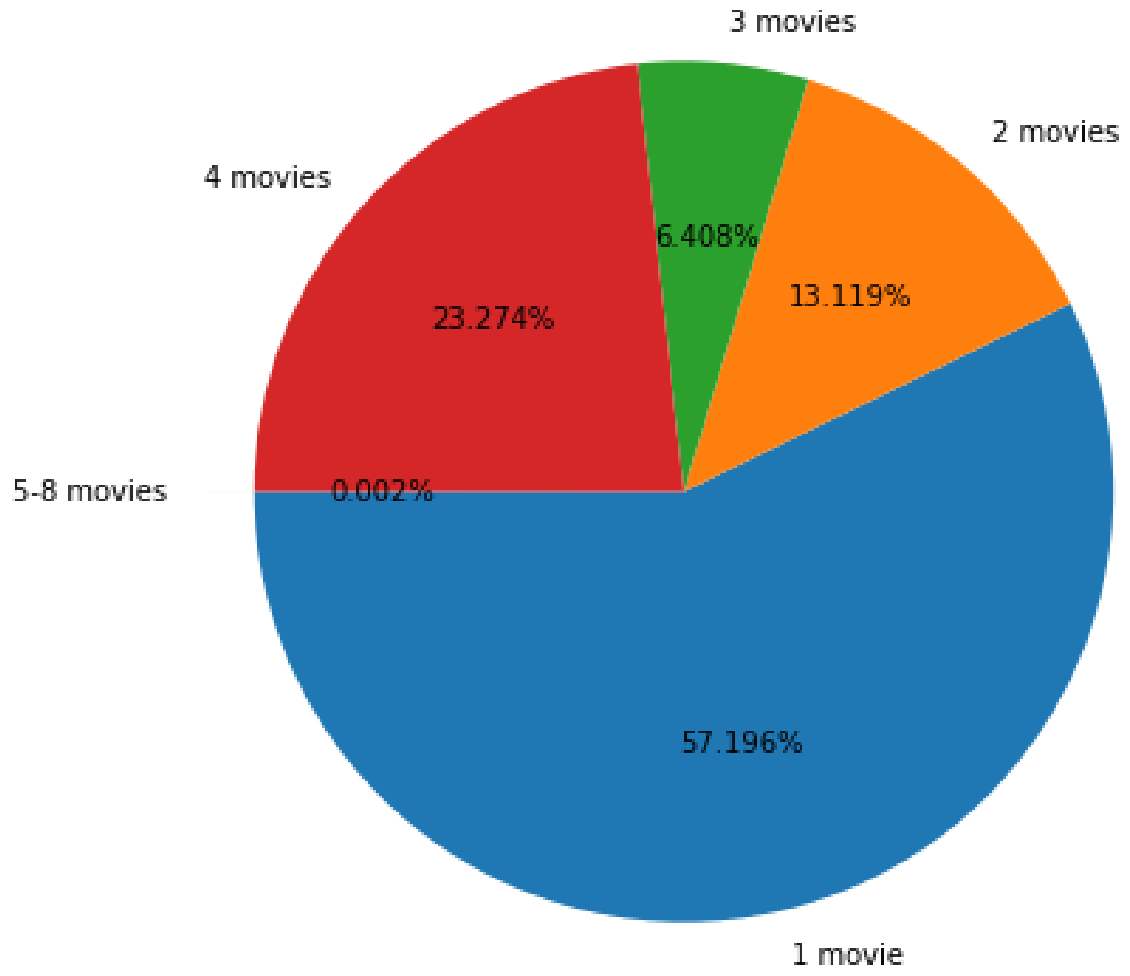

Figure 3: Number of movies appearances - "Actors with other roles"

Summarizing, four experiments have been executed:

- Apriori algorithm with "Actors only" dataset
- Apriori algorithm with "Actors with other roles" dataset
- FP-growth algorithm with "Aactors only" dataset
- FP-growth algorithm with "Actors with other roles" dataset

8 Results discussion

Apriori algorithm with "Actors only" dataset

This implementation took 194.65 seconds (3 minutes circa) to find frequent itemsets and returned 24 items.

Apriori algorithm with "Actors with other roles" dataset

This implementation took 1012.08 seconds (16 minutes circa) to complete and found 61 frequent items.

FP-growth algorithm with "Aactors only" dataset

This algorithm took 87.39 seconds(2 minutes circa) to return 24 items.

FP-growth algorithm with "Actors with other roles" dataset

This algorithm took 84.89 seconds (less than 2 minutes) to return 61 items.

As expected the two algorithms, Apriori and FP-growth returned the same result, in both cases, since the operation can be considered deterministic. The difference between the two algorithms however, is in the excution time. However, also this difference could be anticipated since the algorithm implemented in the MLlib library does not generate candidates for each pass of the algorithm, and this results into more efficiency in terms of execution time with respect to Apriori.

The Apriori algorithm, instead took almost 3 minutes for finding frequent itemsets in the "Actors only" dataset as opposed to almost 16 minutes for the second case; also this result could be expected, because in the second case there is more data to be processed, thus more candidate itemsets to generate, which results in an increase in the execution time.

Finally, both algorithms returned only singletons, meaning frequent itemsets composed of single items. This can be expected given the domain to which this type of analysis has been applied and given the type of data considered: the "Actors only" dataset, as well as the "Actors with other roles" dataset, are not strictly based on movies, but also on documentaries, shorts, TV shows and more.

Typically actors that play a role in cinema movie do not play roles also in shorts or in documentaries, this therefore reduces the chance to find one actor in many baskets (movies) (indeed the highest number of actors appearances is 8 as shown in the figure above), thus it is even harder to find pairs or triplets of actors in more than one movie, since indeed the dataset is very variegated. On the

other hand lowering the minimum support value would not result into a meaningful analysis, since here the number of baskets (movies) is very high and the support is very low as it is, thus we would obtain results that are not truly frequent.

The results obtained with "Actors only" dataset are shown in Figure 4, while the results obtained with "Actors with other roles" are shown in Figure 5

nconst	freq	primaryName	primaryProfession
nm4517234	6	Claydee Lupa	actor
nm0472407	6	Dora Krsková	actress
nm4079966	5	Jordan Bernarde	actor
nm0017292	5	Bill Alcorn	actor
nm6038165	5	Kuan-Ting Liu	actor
nm0662565	5	Reed Parker	actor
nm0186937	5	Hal Le Sueur	actor
nm1708736	5	Sloan DeForest	actress
nm5756622	5	Aaliyah Rose	actress
nm3294984	5	Carl Paul Ezold Jr.	actor
nm3304752	5	Giselle Davila	actress
nm5457566	5	Katie Reese	actress
nm1948317	5	Chelsea Johnson	actress
nm2838205	5	Evangeline Gabrie...	actress
nm5613952	5	Audrey Luo	actress
nm0686428	5	Demetra Plakas	actress
nm0361727	5	Reiko Oimori	actress
nm1218414	5	Yannis Zoubantis	actor
nm5092676	5	Joey Biohazard	actor
nm2591423	5	Logan William McPeak	actor
nm8805467	5	Marco Joseph	actor
nm9828634	5	Simon James Kippen	actor
nm0891937	5	Candy Vegas	actress
nm1163810	5	Yanhua Chen	actress

Figure 4: "Actors only" result

nconst	freq	primaryName	primaryProfession
nm1512084	8	Wolfgang Hofer	writer,soundtrack...
nm0809850	7	Susan Duncan Smith	soundtrack,music_...
nm6232022	6	Bill Mackenzie	actor,director,vi...
nm0472407	6	Dora Krsková	actress
nm4517234	6	Claydee Lupa	actor
nm4079966	5	Jordan Bernarde	actor
nm9294565	5	Greg Zatzkis	miscellaneous,act...
nm6580550	5	Remy Cashman	actress,producer,...
nm5809770	5	The Ken Darby Sin...	music_department,...
nm2281222	5	Ted Newsome	producer,director...
nm0017292	5	Bill Alcorn	actor
nm2013626	5	CocoRosie	actress,music_dep...
nm9767585	5	Freidric Macapaga...	camera_department...
nm4398357	5	Leon Fontaine	producer,writer,a...
nm7172326	5	Joe Nicolosi	actor,transportat...
nm6038165	5	Kuan-Ting Liu	actor
nm6083431	5	Tawan Sanders	director,actress,...
nm0662565	5	Reed Parker	actor
nm0186937	5	Hal Le Sueur	actor
nm3727115	5	Jenny Lee	miscellaneous,act...
nm1708736	5	Sloan DeForest	actress
nm5923758	5	Rakesh Adiga	actor,director,wr...
nm6232025	5	Sara Higgins Mack...	actress,costume_d...
nm5751952	5	Cameron Cook	actor,producer,di...
nm1858039	5	Meelis Muhu	actor,producer,di...
nm3294984	5	Carl Paul Ezold Jr.	actor
nm6168142	5	Mandy Faura	writer,director,a...
nm10261459	5	Emmett van Halm	actor,camera_depa...
nm2968946	5	Sabrina Chen-Louie	producer,actor,mi...
nm1780432	5	Doug Pinnick	actor,composer,mu...
nm3304752	5	Giselle Davila	actress
nm5457566	5	Katie Reese	actress
nm1835848	5	John W. Sloan	actor,director,wr...
nm1948317	5	Chelsea Johnson	actress
nm1163810	5	Yanhua Chen	actress
nm0300543	5	Per Gade	music_department,...
nm0332297	5	Rui Goulart	director,writer,a...
nm2838205	5	Evangeline Gabrie...	actress
nm2206015	5	Mark Morton	soundtrack,actor,...
nm8745061	5	Ben Kawaller	actor,writer
nm1827550	5	Gina Novotchin	make_up_departmen...
nm5613952	5	Audrey Luo	actress
nm6236508	5	Alan Mallyon	producer,actor,wr...
nm5973476	5	J. Ryan Hickey	special_effects,m...
nm9117654	5	Emma Foley	actress,director,...
nm5756622	5	Aaliyah Rose	actress
nm0686428	5	Demetra Plakas	actress
nm0112122	5	Lucius Brooks	actor,soundtrack
nm0361727	5	Reiko Oimori	actress
nm1218414	5	Yannis Zoubantis	actor
nm5230491	5	Lan Lanh	soundtrack,actres...
nm5092676	5	Joey Biohazard	actor
nm2591423	5	Logan William McPeak	actor
nm2611302	5	Miguel Rodrigáñez	music_department,...
nm8805467	5	Marco Joseph	actor
nm6312662	5	Lucas Klauss	writer,producer,a...
nm9828634	5	Simon James Kippen	actor
nm4706609	5	Jonathan Thompson	director,actor,ed...
nm1795875	5	Ismo Heikkilä	actor,sound_depar...
nm0891937	5	Candy Vegas	actress
nm8026299	5	Mike Torres	actor,casting_dir...

Figure 5: "Actors with other roles" result

9 Conclusion

In this project two different algorithms have been used to retrive frequent itemsets. The firts one, the Apriori algorithm implemented with Spark, requires a higher execution time than the second one, FP-growth. However, both algorithms find the same result, as expected.

10 Tools

For this project the following tools have been used:

- Google Colab
- Kaggle
- Spark/PySpark v. 3.0.2
- Python v. 3.6

"I/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.".