FREQUENT ITEMSETS IN IMDB DATASET

Market Basket Analysis using actors as items

Martina Viggiano (954603)

MSc Data Science and Economics - Università degli Studi di Milano

Abstract. This project presents an analysis on kaggle *IMDB dataset* provided by *ashirwadsangwan*[1]. The object was to implement a scalable market basket analysis considering movies as baskets and actors as items, in order to retrieve association rules between the items.

Keywords: Market basket analysis; Frequent itemsets; Association rules; Apriori algorithm; FP-Growth algorithm.

1 Introduction

1.1 IMDB Dataset

The dataset we used was composed of five sets, each providing a different piece of information. In our case we only used the following three sets:

- *title.basics*: set presenting details of video shooting from which we extracted the type/format of the title - i.e. movie, short, tv series, tv episode, video, etc - and we filtered for movies;
- *title.principals*: set providing information about the cast and the crew working for each video shooting present in the dataset. In this case, we only extracted the alphanumeric unique identifier of actors and actress;
- *name.basics*: set displaying personal details about the cast and the crew, and we used it to associate each alphanumeric unique identifier to the corresponding name and surname of the actor/actress.

Since each set provided only one aspect of the information we need to develop the project, we joined the pre-processed sets - i.e. filtering *title.basics* for movies and *title.principals* for actors/actresses - such that for each row of the dataframe we built we had a movie and one actor/actress playing in it.
Then we processed this dataframe again in order to obtain a unique row for each movie, in which we paired one movie to the entire cast, which corresponded to list of actors/actresses on the set, namely retrieving the basket of items. What we obtained was a dataframe with two columns - IDs of movies *tconst* and sets of items (actors) - and 1694722 rows - i.e. the number of movies in our original data.

```
+---------+--------------------+
|   tconst|              actors|
+---------+--------------------+
|tt0000335|[nm1012612, nm067...|
|tt0000630|         [nm0624446]|
|tt0000676|[nm0140054, nm009...|
|tt0000793|         [nm0691995]|
|tt0000862|[nm0264569, nm528...|
+---------+--------------------+
only showing top 5 rows
```

Fig. 1: Dataset of baskets.

Moreover, we built also a smaller dataset, composed of a fraction[1] of the initial set of movies, and we processed it following the steps above. This smaller set was created to run some tests and comparisons in a quicker way with respect to the whole dataset. What we obtained was a dataframe with two columns - IDs of movies *tconst* and sets of items (actors) - and 169199 rows - i.e. the number of movies in our original data.
For the sake of simplicity, in this project we decided to work with unique identifiers instead of extended titles of movies and name and surnames of the cast. However we also provided few lines of code that can be used to extract and use the proper nouns instead of unique IDs.

2 Market Basket Analysis

In the following sections we are going to display what was done to perform a Market Basket analysis on IMDB dataset, considering sets of actors/actresses as itemsets and movies as baskets.
We operated two algorithms: Apriori algorithm and Frequent Pattern Growth algorithm.

2.1 Apriori Algorithm

Apriori is an algorithm designed for mining frequent itemsets, which is based on computing frequency of only a reduced number of sets, and which is developed in two passes. After setting a given threshold for support, it starts by selecting the frequent singletons - i.e. the single items which in our case are the actors/actresses - and then extends the sets by getting larger itemsets as long as those itemsets are sufficiently often in the database, thus the frequency of the itemsets lie above the threshold.
Moreover, the algorithm is based on the property of monotonicity: if a set is frequent, then its subsets are frequent. For example, this implies that, if a pair of items is frequent, then the two items in it must be singularly frequent.

[1] We selected a random sample of 10% of the original set *title.basics*.

Starting from these definitions, we built from scratch our Apriori algorithm.
First we created a list of lists composed of each itemset and we flattened it. Then we created tuples associating to each item its frequency count: we started by setting 1 for each item and then summing every time we found the same item in the RDD.
After that we filtered the singletons based on the threshold chosen and we kept only items with frequency above the level provided.
Next, we built all the possible combinations of pairs of frequent singletons retrieved in the previous step: we checked if each pair we retrieved were present in out set of baskets and then we filtered again for the threshold.
We repeated the steps above to search for frequent triplets, starting from the results we obtained previously. We did not iterate further than triplets since, as we will show later, even triplets appeared so unusual to find, that it would be even more rare to retrieve larger itemsets.
At the end of the process we obtained a list of tuples of frequent itemsets and the corresponding frequency.
We run the algorithm first on the entire dataset composed of 1694722 rows and on the fraction with 169199 rows. For both sets we repeated the computation considering 3 different levels of support: 0.0003, 0.0004 and 0.0005.
As we expected, we experienced a trade-off between the time spent by the al-

	Actors	Frequency
0	(nm0006982, nm0623427)	236
1	(nm0006982, nm0419653)	162
2	(nm0006982, nm0046850)	169

Fig. 2: Model on the whole dataset setting support at 0.0004.

gorithm to complete the steps and the number of frequent itemsets retrieved. By setting a lower value of support, the amount of data the algorithm had to compute increased, thus it took more time to process it. On the other hand, by lowering the threshold, we also got to retrieve a higher number of frequent itemsets, because a lower frequency was sufficient for a set (and item) to be considered frequent.

	Actors	Frequency
0	(nm0619779, nm0006982)	16
1	(nm0619779, nm0623427)	15
2	(nm0006982, nm0623427)	30
3	(nm0006982, nm0046850)	21
4	(nm0006982, nm0419653)	19
5	(nm0046850, nm0419653)	16
6	(nm0623427, nm0419653)	14
7	(nm0616102, nm0006982)	13
8	(nm2366585, nm2384746)	12
9	(nm2082516, nm0648803)	15

Fig. 3: Model on the fraction of the dataset setting support at 0.0003.

2.2 Frequent Pattern Growth Algorithm

The Frequent Pattern Growth algorithm - we will refer to it as FP-Growth later in this paper - differently from Apriori algorithm, instead of generating candidates, represents the database in a tree form, called FP tree. This tree structure consists of a compressed representation of the itemset database which keeps also track of the association rules between the itemsets. It maps each itemset to a path in the tree one at a time, with the idea that more frequently occurring items will have better chances of sharing items. [3] [4]
One of the main advantages of this algorithm is that it is faster than Apriori algorithm. Thus, in this project we deployed the Spark FP-Growth algorithm to explore data, to test and compare models and to display some results in a faster way with respect to the Apriori algorithm we implemented from scratch.
As we have done with Apriori algorithm, we computed several models, each for a different value of support: 0.00001, 0.00002, 0.00003, 0.00004 and 0.0001. In this case, the levels are smaller that before, since, as we will show later, the algorithm quickly reduces the number of frequent itemsets retrieved by increasing the threshold.
On the other hand the speed of FP-Growth is way faster than Apriori. As a matter of fact, while for Apriori algorithm with support level set equal to 0.0003 the process took more than 2 hours, FP-Growth completed the computation using a lower threshold - 0.0001 - in only 105 seconds.
By computing those five models, we were able to compare some aspects of this market basket process.
First we displayed the change in time spent in completing the computation with respect to level of threshold provided. As we can see from Figure 4By increasing value of support, the time spent in computation slightly decreases.

On the same grounds, by studying the relation between the support and the

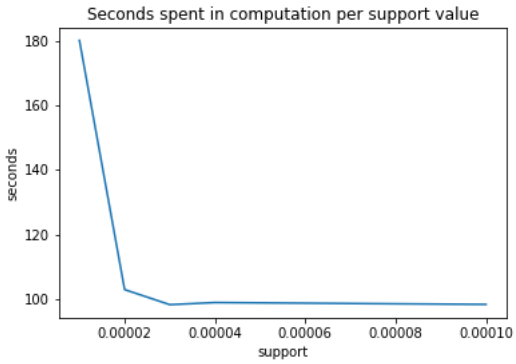

Fig. 4: Time spent with respect to level of support.

number of association rules retrieved, we see that increasing the threshold, the number of items extracted decreases significantly.

Then we studied whether we had any connection between the value of sup-

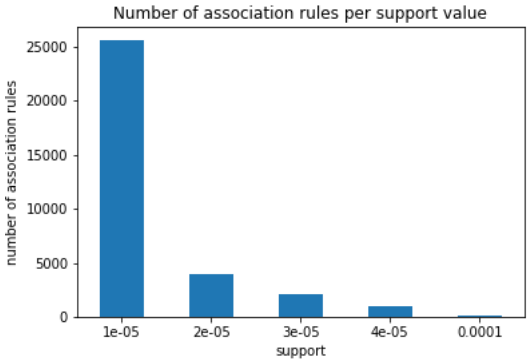

Fig. 5: Number of association rules with respect to level of support.

port and the average confidence value of each model. To do so, we computed the arithmetic mean of the column *confidence* retrieved by the *associationRules* function of FP-Growth. What we obtained is not so clear: what we can state seeing Figure 6 is that for all five models, the average confidence is around 0.9, which is a quite high level of confidence.

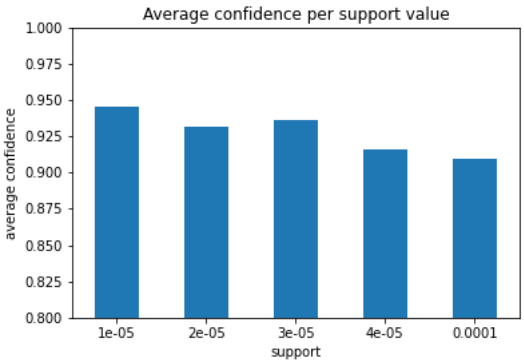

Fig. 6: Average value of confidence with respect to level of support.

3 Conclusions

?.

4 Declaration

I declare that this material, which I now submit for assessment, is entirely my 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 work. I 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.

References

1. IMDb Dataset - Dataset from IMDB to make a recommendation system. Available at: https://www.kaggle.com/ashirwadsangwan/imdb-dataset.
2. Jure Leskovec, Anand Rajaraman, Jeffrey D. Ullman: Mining of Massive Datasets. Cambridge University Press. 2014.
3. Software Testing Help website: Frequent Pattern (FP) Growth Algorithm In Data Mining. 2021. https://www.softwaretestinghelp.com/fp-growth-algorithm-data-mining/.
4. Chonyy - Towards Data Science website: FP Growth: Frequent Pattern Generation in Data Mining with Python Implementation. 2020. https://towardsdatascience.com/fp-growth-frequent-pattern-generation-in-data-mining-with-python-implementation-244e561ab1c3.