Report of the project
Algorithms for massive data
Market-basket analysis

Nicolò Pignatelli, 13831A

Academic year 2023-2024

Contents

1 Abstract

In this report I present my project about market-basket analysis. Market-basket model of data is a form of representation of a many-to-many relationship between baskets and items. Each basket contains a certain number of items. It is assumed that the number of baskets is so large that it cannot fit in main memory. The analysis consists of finding the frequent itemsets, that is the tuples (singletons, pairs and so on) of items that appear in more than a certain number of baskets.
The goal of this project is to implement a detector in order to retrieve the sets of job skills that are more frequent in some *LinkedIn* pages. To do so, I implement the A-Priori algorithm. The main feature that I use in order to process the involved large amount of data is *PySpark*, a *Python* library. It enables to work with massive datasets. Except for *itertools*, no other library or similar is imported. The advantage of using *PySpark* is that it allows to distribute the data across multiple nodes and so to work efficiently with large amounts of information. Thanks to this, it is also very fast to process operations.
After briefly illustrating the dataset and the data pre-processing, I explain how the A-Priori algorithm works. In the end, I present the obtained results.
It is noteworthy to mention that the code is specifically designed to be downloaded and run automatically from beginning to end in a feasible amount of time, that I set to around 10/15 minutes. The only thing to do is to insert your credentials of the *Kaggle* API.

2 The dataset

Data are retrieved from this data set, released under the ODC-By license, Version 1.0. According to the *About Dataset* section, it contains 1.3 million job listings scraped from *LinkedIn* in the year 2024. I download it via the *Kaggle* API. It is composed of three *.csv* files: *job_skills*, *job_summary* and *linkedin_job_posting*. For this project, only the first one is considered. It is composed of two columns: *job_links* and *job_skills*. I consider only the second, as the first is made only by links that refer to the *LinkedIn* page of the job offer. The *job_skills* column has 1294374 rows. Every cell contains a string which is the list of the desirable skills attached to the job offer. The only operation to do is to drop *NAs*, if any.
Even if the code is written in order to deal with large datasets, still I have technological constraints. If I run the code with all the data, after half an hour the algorithm would not retrieve the number of frequent itemsets of size 3. So, I decide to consider a random sample of 1% of the rows, so more or less 13 thousands strings. From now on, consider that all work is carried out on the sample.

3 Data pre-processing

As already mentioned, I consider only the *job_skills* column. I assign it to a resilient distributed dataset, shorlty rdd. The rdd is the main tool that *PySpark* offers in order to deal with large amounts of data. Once the column is passed to it, the data are broken into partitions and splitted across various clusters. In this case, the dataset is divided across six partitions. Then I transform the rdd so that each element is no more a Row type but the string of skills. In the end, I create the rdd named *baskets* with the function *.split()* that convert a string into a list of strings. Now I am also able to analyse some statistics about the skills in the dataset. The highest number of skills required for a job is 195, whereas the lowest is just 1. Averagely, around 21 skills are desired. I assume that there is no job offer with a duplicate skill.
Now I consider that my data are strings and that a consecutive integer representation from 1 to n, where n is the total number of unique skills, would be more space-efficient. This means that I need to hash my items. I start from my rdd, baskets, that contains lists of skills as strings. From it, I create a rdd that consider every skill as a single element. Thanks to this rdd, I count all the skills removing duplicates and I obtain that there are around 100 thousands single skills in my dataset. Now that all duplicates are removed, I assign to every skill its index in the rdd, so that I create the hash table that I need. After transforming my hash table into a proper dictionary, I pass my baskets rdd and hash every skill. Lastly, during this pass I also transform the lists of my rdd into sets, which is the correct representation of a basket.

4 The algorithm

Once that the rdd baskets is correctly processed and that I have the hash table that maps from the skill to the integer and viceversa, I only miss the support parameter, that discriminates between what is frequent and what is not. I set it to 2%, so I consider frequent an itemset if it appears in more than this percentage of baskets. Again, I do not choose the usual value of this parameter, 1%, for complying with the schedule of 10/15 minutes of total running of the whole file.
The algorithm is composed of various passes. To implement them, I use some *PySpark* operations, in order to be able to be efficient with the big number of baskets and items in them. In the first pass, I start by performing a map transformation: I examine all the items in all the baskets and for every item in a basket create a key-value pair with the item as the key and 1 as the value. Then I perform a reduce task where I sum all the 1s for a given key. At this point I have key-value pairs with the integer of the skill and its count in the baskets. Lastly, I filter by keeping only the items that has a value greater than the support. After counting the number of frequent singletons and retrieving the most frequent one thanks to the hash table, I keep track of them with the variable named *freqs*.
The reason why the first pass is separated from the subsequent passes, during which the algorithm retrieves the frequent itemsets with more than one item, is that in the following steps when I map the tuple of skills I filter only for those for which all the subsets belong to *freqs*, in order to avoid counting for all possible tuples. These tuples are the candidates to be the frequent itemsets. This is not done in the first pass, or better all the singletons are candidates. To be clear, for example in the first pass I count for all the items and then keep only the ones that are more frequent, so appear in more baskets, than the support. Then I count only the pairs for which both the singletons are frequent. I am able to do so thanks to the monotonicity assumption, for which if an item is frequent then all its subsets are frequent. All of this is performed because the space required for the operation decreases dramatically, as we do not count for all possible pairs. This is very important when the involved amount of data is massive. Once I have the candidate pairs, I keep only the frequent pairs by filtering for the support. The algorithm goes on, every time recording the frequent sets of the corresponding size, generating the new bigger ones and creating the candidates only if all the subsets of the latter with the size lowered by one are frequent. Once the frequent itemsets are collected, the search continues. This ends when there are no frequent itemsets of a certain size, so no itemset appears in a number of baskets greater than the support.
An additional thing to notice is that in the map operation of the second or higher passes I need to sort the baskets. This is done because the generated itemsets are tuples for the machine. Tuples with the same items but in different orders, as for example (1,2) and (2,1), would be counted separately and this is not desired. If I firstly sort the baskets, which are sets, this cannot happen.

5 Discussion

The results of the analysis are summarized in *Figure 1.* Of course, they are subject to the sampling process.

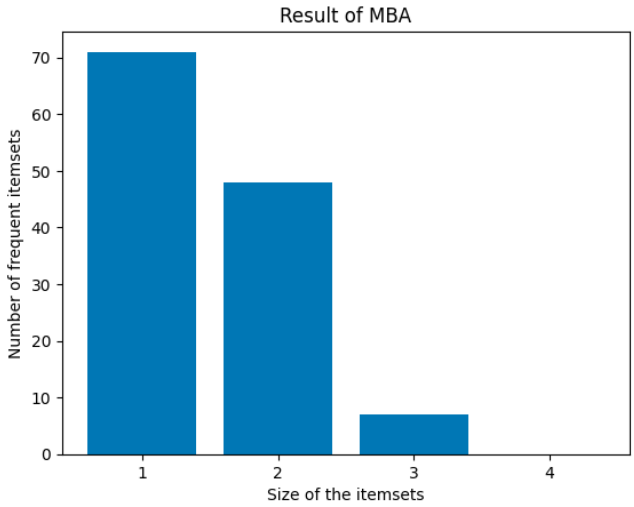

Figure 1: Plot of the results of the analysis

From size 1 to size 3 the number of frequent itemsets decreases until there are no more frequent itemsets of size 4.
Another interesting insight is about the most frequent itemset. There are always the same three skills that compose it. *Communication* is the most desired skill and it appears in both the most frequent pair, with *Teamwork*, and triple, with again *Teamwork* and *Leadership*. Probably this result is not surprising as these are soft skills that are always welcomed as key ingredients for a good employee, whatever job they are applying for.

6 References

Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2nd edition, 2014