Algorithms for Massive Data

Experimental Project

Riccardo Sturla
ID: 13457A

Academic Year 2023-2024

Abstract

The project aims to implement a system finding frequent itemsets of skills in the LinkedIn Jobs & Skills dataset. The detector, performing the so called *Market-Basket analysis*, considers as baskets the sets of skills required for each job announcement and uses the A-Priori algorithm to find all the k-size frequent itemsets. The skills obtained can be then used for further analysis.

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 or any other person for assessment on this or any other course of study.

Contents

1 Introduction **3**

2 The Data **3**
2.1 Preprocessing 3
2.2 Insights 4

3 The A-Priori Algorithm **4**

4 Results **5**

5 Scalability **7**

6 Conclusion **7**

7 References **7**

1 Introduction

The aim of this project is to find all the frequent sets of required skills in a dataset of job announcements. To do so, we exploit the market-basket model.
The market-basket model of data is a technique describing a many-many relationship between two kinds of elements, called "items" and "baskets". Such model was originally developed in the analysis of true market baskets with the goal of finding which products were commonly bought together. Its applications are clearly not limited to stores or supermarkets, but range to many contexts, such as textual data problems.

Approaching the data in a scalable way, through the use of the Spark API for Python, PySpark, I implemented the A-Priori algorithm and built a detector of frequent skills.

This report is organized as follow: Section 2 deals with the dataset composition and preprocessing; Section 3 tackles the algorithm implementation phases; Section 4 shows and discuss the results; Section 5 discusses about the scalability of the detector.

2 The Data

The data considered in the analaysis is part of the Linkedin Jobs & Skills (2024) released under the ODC-By license, Version 1.0, and available on Kaggle, specifically the "job skills" file is utilized. Such dataset is composed of almost 1.300.000 rows[1] and two columns, one containing the link for the job insertion and one containing the list of requested skills for that job. In our specific case, each row represents a basket to which a set of items is associated, i.e. the skills.

2.1 Preprocessing

In order to make the data suitable for the market-basket model and for the iteration of an algorithm, a preprocessing phase must be performed. Null values are checked and removed, the text is converted to lowercase and each skill, which can be composed by one or more words, is "detected" and saved, splitting each list of items at any comma occurence. Finally, eventual punctuation or unnecessary characters are removed from the data.

Before implementing any algorithm, I decided to represent each item as an integer. In fact, frequent-itemset algorithms require to maintain many different counts as they pass through the data and it is possible that there is not enough main memory to store each of the counts. In that case, the algorithm will thrash and run many orders of magnitude slower. Thus, each algorithm has a limit on how many items it can deal with. For such reason, expecially when dealing with textual items, it is more space-efficient to represent items by consecutive integers from 1 to n, where n is the number of *distinct* items. So, I built an hash table that translates distinct skills to integers; that is done through the use of a python vocabulary, where each key is a skill and its corresponding value is an integer. By doing that, I was able to convert items to numbers and viceversa, everytime it was needed, for instance in the visualization of the results.

[1]Note: to make the computation and the experiments faster, a subset of 1% is considered, (see Scalability).

2.2 Insights

As said before, skills in the data can be composed of one or more words. The number of skills in each job announcement is various and some metrics are summarized in Table 1.

N. of Skills for Job	
Minimum:	1
Maximum:	463
Average:	21

Table 1: Number of skills per job in the entire dataset

3 The A-Priori Algorithm

The A-Priori algorithm is a common choice when dealing with frequent itemsets because it is designed to reduce the number of candidates that must be counted, so the weight on the memory, at the expense of performing more passes over the data.

The A-Priori algorithm utilizes "passes" to find frequent itemsets: one pass is taken for each set-size k. I first implemented each step of the algorithm separately and then I created an automatic function which mimics the steps of the A-Priori. First, given the support threshold, it examines the occurrences of the items to determine which of them are frequent as singletons. For the property of monotonicity for itemsets: if a set I of items is frequent, then so is every subset of I. Therefore, a pair cannot be frequent unless both its members are frequent. That being said, after the first pass, the function creates all the distinct pairs that consist of two frequent items. Such pairs form the candidates for the next iteration in which their occurrences are examined to determine the frequent ones. The process continues through a loop for each k-size itemset until no frequent itemsets of a certain size are found. In that case, monotonicity tells us there can be no larger frequent itemsets, so the algorithm stops. Calling C the set of candidates and L the set of truly frequent itemsets, the pattern of moving from one size k to the next size $k+1$ is summarized in Fig. 1.

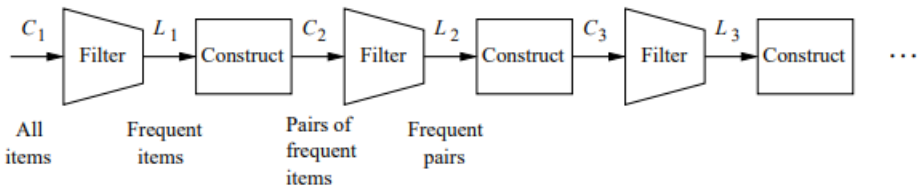

Figure 1: The A-Priori algorithm process

For the specific case of the function I implemented, the user can freely choose a percentage to compute the support threshold, but also a maximum itemsize to consider. For example, setting that parameter to 2 means that only frequent singletons and pairs will be computed (if possible). For the purpose of this report, as mentioned before, the percentage of the support threshold was set to 2% and a small sample of the entire dataset was considered. With such assumptions, the algorithm takes about 8 minutes to run. Note that, by definition, choosing a lower threshold would increase the time taken to run the algorithm and it would return a higher number of frequent skills, possibly making the results less relevant.

4 Results

The algorithm returns the list of frequent itemsets. Recall again that the support threshold was set to be 2% of the baskets, so, to be frequent, an itemset must appear in at least *0.02 x Number of baskets* jobs. The most frequent itemset in the sample considered is a singleton composed by the skill "communication". The first six positions of the ranking are occupied by singletons: that's something we expected since, due to the property of monotonicity, it is more likely to find frequent singletons than bigger itemsets. In particular, the A-Priori finds 83 frequent singletons, 69 pairs, 22 triples; no 4-size itemset is found, so the algorithm stops.

We can have a look at some details:

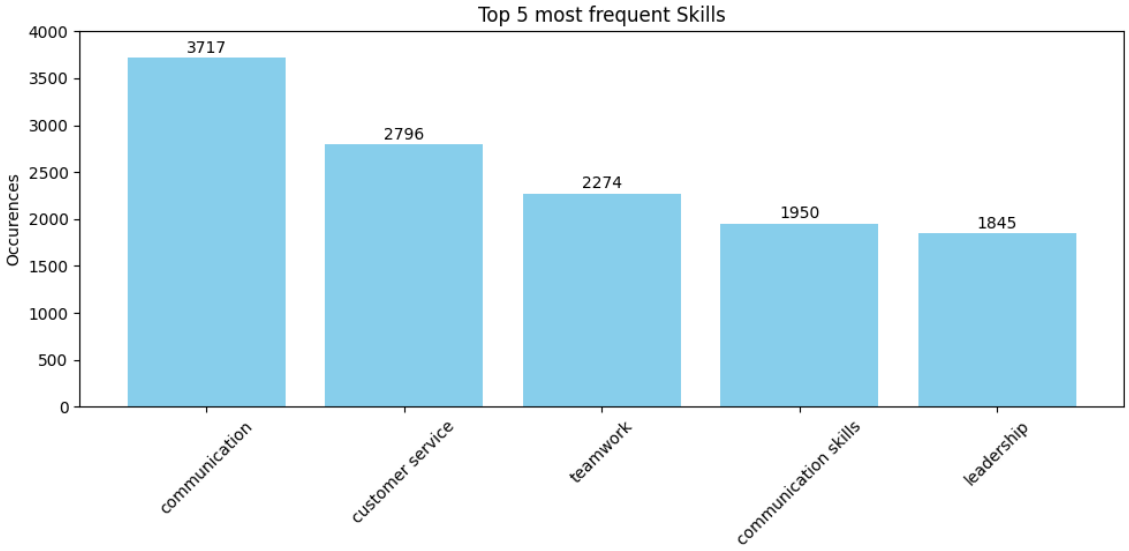

Figure 2: Most frequent singletons

The most frequent singletons are *communication, customer service, teamwork, communication skills, leadership*. Those are very general skills which don't give us any particular information. The large number of *customer service* could suggest the presence of many jobs that need the employer to directly interact with the clients.

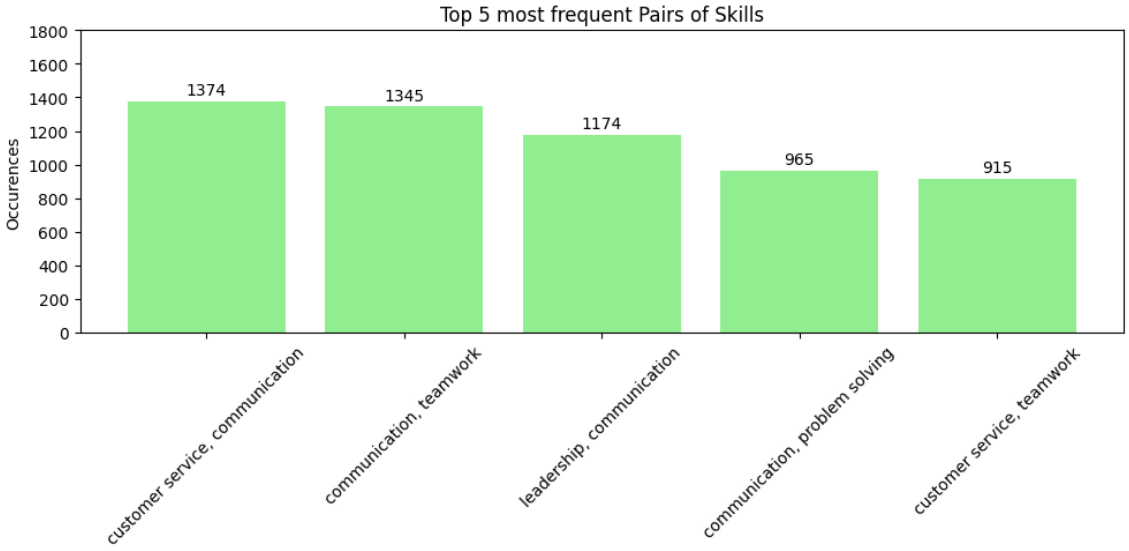

Figure 3: Most frequent pairs

Looking at the frequent pairs, we find some more interesting occurrences, with 4 out of the 5 couples including *communication*: the most frequent pairs are mainly combination of the most frequent singleton. That is obviously a consequence of how the A-Priori works.

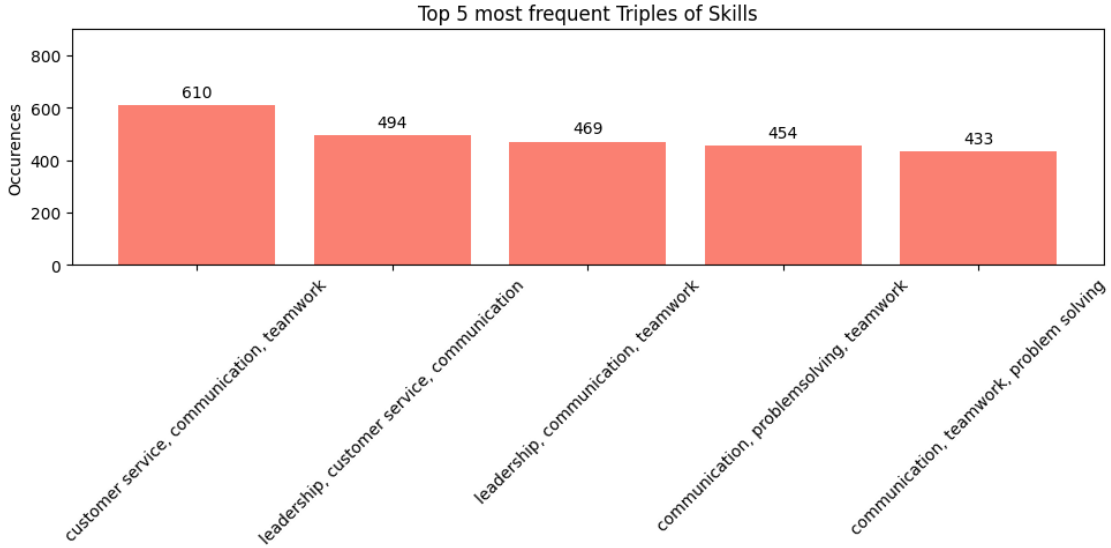

Figure 4: Most frequent triples

The same thing can be observed in the frequent triples. The most frequent 3-size itemset is *customer service, communication, teamwork*.

The frequent itemsets identified by the A-Priori could be then used as input for further analysis with other portions of the data.

5 Scalability

As previously said, for the purpose of the experiment just a sample of the entire dataset was considered in performing the algorithm. However, all the code was implemented to promote scalability, in a Spark-Hadoop enviroment, with the API for Python, PySpark.

The data are loaded in a *Resilient Distributed Dataset* and the algorithm uses PySpark RDD functions such as *Map*, *Reduce* and *GroupByKey* to operate in a parallelize way, promoting scalability. The algorithm scales-up well with data size, even if, depending on the computational power at disposal, it would require a significantly higher amount of time to run.

6 Conclusion

In the reported experimental project, I implemented a detector of frequent skills in job announcements using the A-Priori algorithm. The detector was built in Python using the PySpark enviroment, with attention on the scalability of the results.

The main difficulties were met working on the Spark syntax and avoiding operations which could lead to computational overloads. The algorithm is efficient even if the task is computational intense and requires some time to complete, based on the support threshold chosen and the amount of data considered.

7 References

1. Jure Leskovec, Anand Rajaraman, Jeffrey D. Ullman, *"Mining of Massive Datasets"*, 2011