Market Basket Analysis:

LinkedIn Jobs & Skills dataset

Report for the Project of Algorithm for Massive Data

Mina Beric

November 2024

UNIVERSITÀ
DEGLI STUDI
DI MILANO

Contents

1 Introduction

The scope of this project is to implement a system for identifying frequent itemsets of varying sizes, commonly referred to as Market Basket Analysis. The market-basket model represents a many-many relationship between two kinds of elements, called *"items"* and *"baskets"*. The goal is to develop a system that uses the A-Priori algorithm to identify the most frequently occurring job skills across various LinkedIn job offers.

The second section provides an overview of the essential steps needed to set up the environment and connect to the Kaggle API for importing the necessary tables. It also covers a description of the dataset used for the analysis.

The third section covers a brief exploratory data analysis and the preprocessing steps to structure the baskets, which will serve as input for the algorithm.

In the fourth section the theory behind the A-Priori algorithm is explained, alongside the details of its implementation.

The last section is dedicated to present the obtained results.

It is worthy to mention that the development and testing environment for this project was provided by Google Colab, which offers a free hosted runtime. This setup utilized Google Drive personal accounts to store the Job&Skill dataset, which was downloaded after establishing a connection with the Kaggle API using my personal tokens. The dataset was then uploaded to the notebook.

Considering the requirements for the task, the algorithm implementation was developed with attention to the available memory of the virtual machine and the execution time of each code chunk. The primary tool used to handle the large volume of data in this project is *PySpark*, a Python library designed for working with massive datasets. PySpark's key advantage is its ability to distribute data across multiple nodes, enabling efficient processing. The entire process is automated to run from start to finish in approximately 10 minutes.

2 The Dataset

The dataset used in this analysis is sourced from Kaggle and imported into Google Colab using the Kaggle API. The API connection is established via

an authentication token, which is downloaded as a JSON file containing the personal API credentials (Kaggle username and API key). This token allows secure, remote access to the dataset.

Since the Google Colab notebook is authorized under the Google account, mounting Google Drive storage requires an initial authorization code when accessed from a different account. Upon executing the code in a new environment, users must log in with Google credentials and paste an authorization code into the notebook to grant access to Google Drive storage.

After uploading the JSON token and installing necessary dependencies, such as Java and Spark, we initialize a Spark context, allowing for efficient data processing. The dataset's relevant CSV files are then imported as PySpark DataFrames.

Regarding the dataset, it contains 1.3 million Job listings scraped from LinkedIn in 2024. From the three .csv files (*job_skill, job_summary and linkedin_job_posting*) only *job_skill* will be considered for the purpose of the project. The file is composed by two columns, the *job_skill* column has 1294374 rows. For every job offer, the list of skills is included per cell.

3 Data Pre-processing

The first step in data pre-processing is cleaning the dataset to remove any missing values to ensure data integrity. In the explanatory data analysis a first inspection identified *Communication, Teamwork* and *Leadership* as the most common job skills requested across all job listings.

Due to the large size of the dataset (with over 1.29 million rows), working with the entire dataset is not feasible due to memory and performance constraints. Therefore, a random sample of 1% of the data is selected for further analysis. This reduces the dataset to 12,977 records and it is important to note that all subsequent analyses and considerations are based on this sample.

With the dataset now cleaned and sampled, the next step is to convert it into a form that can be used by the A-Priori algorithm. The job skills data is first converted into a *Resilient Distributed Dataset (RDD)*, which is a distributed collection of data that is splitted (in this case in 6 partition) and processed in parallel across multiple nodes.

First, the row of the job_skills column is flattened into individual elements, with each job skill string represented as a distinct element in an RDD. Af-

terward, the skills for each job listing are split (using the *()split* function) into a list of individual skills, creating a new RDD where each job's skills are represented as a list.

To optimize the data processing for the A-Priori algorithm, it is more space efficient to hash the items by assigning each unique job skill to a unique index using a hash table. To accomplish this, The skills are first flattened into a single list, and duplicates are removed. Then, each skill is mapped to an index, creating a dictionary *(skill_to_index)* that maps skills to unique integers. This step ensures that each skill has a unique identifier, which is essential for the efficient implementation of the A-Priori algorithm.

Finally, for the Basket creation, the job skills are transformed into baskets by mapping the job skills to their respective indices from the hash table. This final transformation results in a set of baskets where each basket is a collection of job skill indices. Each job posting is now represented as a set of skills, which is a suitable representation of a basket for the Market Basket Analysis.

4 The Algorithm

The support threshold is first defined to determine the minimum frequency required for an itemset to be considered *"frequent"*. Mathematically, it can be expressed as follows: given a support threshold s, a set of items I, and the support of I, denoted as $supp(I)$ and representing the number of baskets containing I as subset, we define an itemset to be frequent if $supp(I) > s$.

To optimize computation time, the threshold s is set at 1.5% of the total number of baskets, ensuring that frequent itemsets are those that appear with at least this level of frequency.[1]

The core principle of the A-Priori algorithm relies on the ***Monotonicity Property*** of itemsets, that asserts that if an itemset I is frequent, then every subset (let's call it J) of I must also be frequent. In mathematical terms, it is described as follow:

[1] A 1.5% support threshold was chosen over the typical 1% primarily to optimize computational efficiency. Through multiple trials, this threshold was found to provide the best balance between sample accuracy, meaningful results, and runtime, ensuring that the code executed within a reasonable time frame (10 minutes more or less) without compromising the quality of the findings.

$$J \subseteq I \Rightarrow supp(J) \geq supp(I)$$

A subset is considered *maximal* when it has no frequent supersets. The implementation of the A-Priori algorithm follows two steps:

- Finding frequent Singletons (Itemsets of Size 1)
- Finding frequent largest sets (Itemsets of Size greater than 1)

The goal is to iteratively expand the size of the itemsets until no larger frequent itemsets are found. The algorithm is implemented through *PySpark* operations to efficiently handle the large number of baskets and items within them.

The first pass of the algorithm identifies frequent individual skills (or singletons). Initially, a map transformation is performed by creating key-value pairs in the form *(skill, 1)* for each skill in a basket. Then a reduce task is performed (`reduceByKey()`) to sum the counts (of the *1ones*) for each key across all baskets. Skills that meet the threshold are retained through the `filter()` transformation. The number of frequent singletons is recorded (with the variable *frequents*) and the most frequent singleton is identified through the hash table.

The algorithm then proceeds to identify frequent itemsets of increasing sizes ($k = 2, 3, ...$) by generating candidate itemsets of size k and checking their frequency. To ensure efficiency, only those candidate itemsets whose subsets (i.e., itemsets of size $k - 1$) were frequent in the previous step are considered, taking advantage of the monotonicity property. By applying this property, the algorithm avoids generating and counting all possible combinations of size $\boldsymbol{k}$, significantly reducing the computational complexity. This means that when the skill tuples are mapped, a filter is applied to retain only those where all the subsets are contained within the *frequents* variable, thereby avoiding the need to count all possible tuples[2]

Each candidate itemset is counted, and only those that meet the support threshold are retained as frequent itemsets. This process repeats for increasing values of k, generating larger itemsets, counting their occurrences,

[2]Also an additional detail is that in the map operation the baskets are sorted. This is necessary because the generated itemsets are tuples and tuples with same items but in different order would be counted separately which is not desirable. Since the baskets are sets, by sorting them this issue is avoided.

and filtering out non-frequent ones. The search continues until no frequent itemsets of a particular size k are found, indicating that no itemset of that size appears in a number of baskets greater than the support threshold. At this point, the algorithm terminates, having identified all frequent itemsets.

The iterative structure of the two main passes of the Apriori algorithm, as previously described, can be extended to identify larger frequent itemsets for sizes $k > k_{\text{initial}}$, typically within a small range (up to 3). This range aligns with the usual cardinality of elements in frequent sets that are most relevant for analysis.

A crucial factor in this process is the pre-defined support threshold s, which must be carefully selected based on the data domain and the specific insights we seek. For effective market-basket analysis, frequent itemsets should constitute a "meaningfully small" fraction of all possible unique combinations—highlighting those occur most often.

Given the likely large size of the dataset, which may exceed main memory capacity, it's essential to retain only a manageable number of frequent itemsets after each pass of the Apriori algorithm. To achieve this, instead of setting the support threshold with the usual 1%, we set it even more higher (1.5% in this case) to ensure the "rarity" of frequent itemsets, making it feasible to store only the necessary frequent itemsets of each size k in the data structure.

This approach significantly reduces the space and time required, as it minimizes the number of candidate itemsets by leveraging the algorithm's monotonicity assumption. Rather than brute-force counting all possible combinations, only valid combinations of frequent itemsets are considered, which is especially crucial when working with massive datasets.

Using the same reasoning and assuming that frequent itemsets are relatively sparse, the implementation in this project is designed to **scale** effectively to larger datasets. By setting the support threshold s appropriately, each step yields a manageable number of candidate itemsets, ensuring the algorithm remains both efficient and scalable.

Lastly, it is important to note that the dictionary *itemset_counts* is used to track the number of frequent itemsets at each size. This data is then used for plotting purposes to visualize the distribution of frequent itemsets by their size.

5 Results

To analyze the results, we rely on the constructed `apriori()` function implemented.[3] The algorithm identified a total of 96 frequent singletons, with ***Communication*** being the most frequent itemset of Size 1. As the size increased, fewer frequent itemsets are identified. Specifically, 64 frequent pairs, with ***Communication*** combined with ***Teamwork*** occuring most frequently. For itemsets of size 3, the algorithm detected 9 frequent itemsets with the combination of ***Communication, Teamwork*** and ***Leadership*** emerging as the most frequent ones.

These combinations are unsurprising, as they reflect general skills commonly sought after in the job market for employability, particularly in terms of personality traits and soft skills. In contrast, more specialized skills like "biochemical engineer" are less frequently listed and cater to more niche roles.

The algorithm terminates the search for larger itemsets once no frequent itemsets of size 4 are found. The visualization provides an overview of the results:

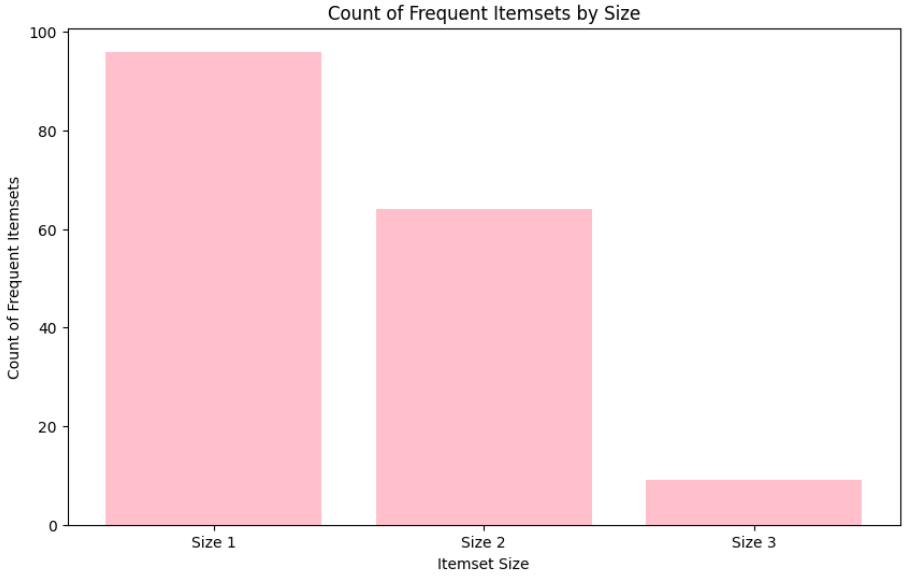

Figure 1: Results of the MBA on LinkedIn Job & Skill dataset

[3]Of course we need to keep in mind that all considerations made are based on the sample taken from the original dataset.

6 References

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