Market Basket Analysis on LinkedIn Skills

Francesco Scateni

A.Y. 2023-2024

Contents

1 Introduction

Market Basket Analysis is a technique used to understand customer purchasing behavior by analyzing the occurrence of items in transactions.

Actually, it can be used to analyze frequent itemsets for any dataset: indeed this report focuses on analyzing a dataset of "Linkedin Jobs & Skills" to discover the frequent sets of skills items and the association rules between them.

The analysis is performed using PySpark via a Google Colab notebook written in python language. Finally, two algorithms, "Apriori" and "FP-Growth", are tested to do this type of analysis.

2 Dataset Description

The dataset used is taken from the Kaggle website and is composed of three files: "job_skills.csv", "job_summary.csv" and "linkedin_job_postings.csv". However, for the Market Basket Analysis, only the first file is considered: simply composed of two columns:

- "job_link", which includes the links for the jobs and each of them identifies a different basket.
- "job_skills", which includes a list of skills of different content and size for each link/basket, which are separated by commas.

In total there are almost 1.3 million rows and therefore baskets.

3 Preprocessing

Before proceeding with the algorithm, I performed some necessary data cleaning and preprocessing. First, I imported the dataset, showing the first few rows to understand how it is structured, then I calculated the exact number of rows.

After that, I counted the null values, and since there are only 2007 of them (out of over 1 million rows), I could eliminate them from the dataset without affecting the analyses.

In the next step I wanted to check the first two rows in detail and I noticed that they include several different skills; furthermore they are of

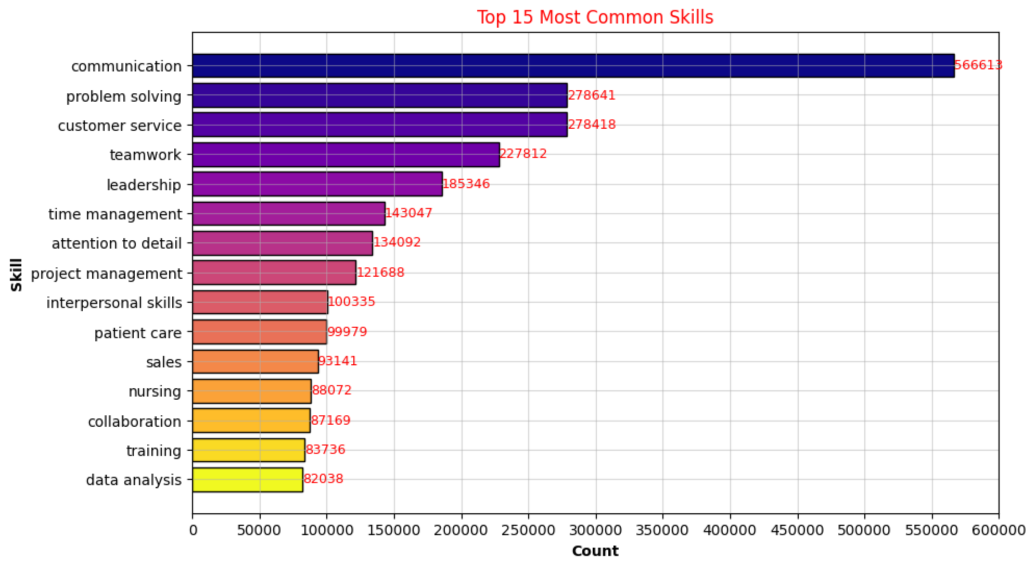

Figure 1: A plot showing the top 15 most common skills

different sizes (14 the first and 19 the second row) noting that the baskets are of variable size.

Then I wanted to calculate the number of unique skills to get an idea of how to proceed with the algorithm later: to do this I created a column containing the arrays of skills separated by commas and then I exploded the array creating a ”skill” column.

This way the dataset contains one row for each skill, which makes it easier to count the unique skills, although doing so increased the size to about 27 million rows.

After that, I removed the blank spaces at the beginning of each line and any special characters, then converted all the skills to lowercase, obtaining an accurate count of unique skills, which turns out to be approximately 2.7 million.

Then I grouped the skills and counted them showing the most and least common. In the top 15 I noticed that there were two redundant skills: in fact there was a skill ”problem solving” and a ”problemsolving”, which obviously referred to the same thing; same concept for the skills ”communication” and ”communication skills”.

For this reason, in the next step I merged the two items under a single skill: "problem solving" in the first case, "communication" in the second to have a more accurate count. Then I re-counted the most common skills and plotted them, as shown in the figure 1.

Finally, using an inner join on the "skill" column, I merged the dataset in which I had grouped the skills (grouped_skills) with the initial clean dataset (cleaned_skills); then I grouped the skills belonging to the same "job_link", creating a final dataset ready for analysis called "baskets", containing the same two columns of the initially imported dataset.

4 Apriori Algorithm

The Apriori algorithm starts by finding individual items that meet a minimum support threshold.

It then iteratively combines these items to find larger itemsets that also meet the support threshold. This process continues until no more frequent itemsets are found.

The Apriori method is based on the monotonicity property:

$$I \text{ freq} \to \text{each } J \subseteq I \text{ is freq}$$

$$\text{supp}(J) \geq \text{supp}(I)$$

So, if a set I is frequent, than a subset J needs to be frequent. The key idea is to use a "bottom-up" approach, where the algorithm first finds the smallest itemsets and uses them to build up to larger ones, pruning those that do not meet the support threshold.

4.1 Implementation and Results

First I created the RDD of the "baskets" dataset to take advantage of the partitioning property and parallel processing.

Then, I used pyspark transformations and actions by dividing the sequences into steps and stopping at the count of triplets, which I considered itemsets of sufficient size.

In the first step I used the following transformations:

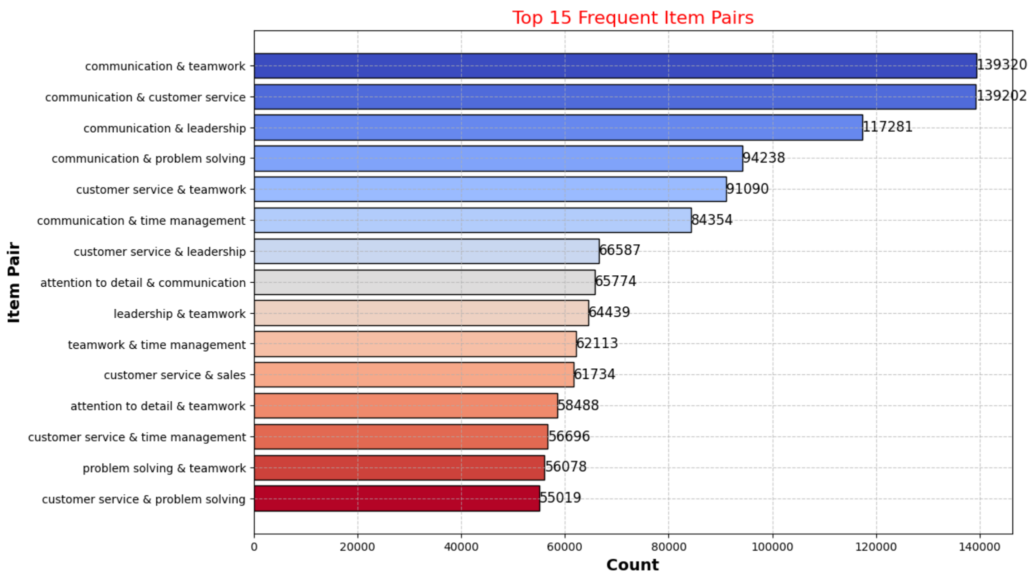

Figure 2: Enter Caption

- flatMap() selects each ”basket” and transforms it into a sequence of pairs. The result is an RDD of pairs (element, 1), where each element is associated with a starting count of 1.
- reduceByKey() groups (item, count) pairs by key (i.e., by item) and adds the counts. So, if an item appears multiple times in different baskets, its counts are added together to get the total.
- filter() selects only those pairs (element, count) where the count is greater than the support threshold, previously set and equal to 5000 baskets. This threshold is the minimum number of occurrences that an element must have to be considered ”frequent”.

Finally, I printed the total number of frequent singletons found as output, using spark's actions count(); then I selected 15 random singletons using take(). Some examples are: ”machine learning”, ”leadership”, ”consulting” etc.

In the second step I created a set containing the found singletons using the collect() action. Then, I used the same transformations as before, but adding the combinations() function, which creates all possible pairs of items and filters only the pairs composed of frequent singletons.

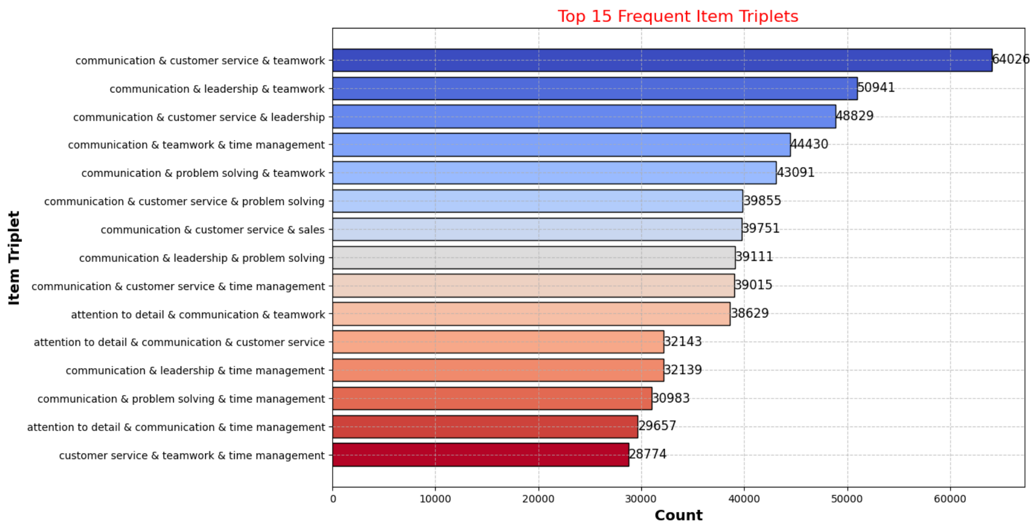

Figure 3: Enter Caption

Finally, in the third and final step, the same procedure is repeated starting from the "frequent_pairs" found in the second step.

Figure 2 shows the top 15 most frequent pairs, while figure 3 shows the top triplets.

5 FP-Growth Algorithm

This algorithm is an improvement to the Apriori method, indeed a frequent pattern is generated without the need for candidate generation. FP growth algorithm represents the dataset in the form of a tree called a frequent pattern tree or FP tree, that will maintain the association between the itemsets.

The dataset is fragmented using one frequent item and this fragmented part is called "pattern fragment". The itemsets of these fragmented patterns are analyzed, thus with this method, the search for frequent itemsets is reduced comparatively.

More in detail: Frequent Pattern tree is a tree-like structure that is made with the initial itemsets of the database. The purpose of the FP tree is to mine the most frequent pattern. Each node of the FP tree represents an item of the itemset, except for the root node, that represents null; the lower nodes represent the itemsets.

As Apriori, also FP-growth is divided in step: firstly it scans the dataset to find the occurrences of the itemsets (count the singletons), than it constructs the FP tree.

The next step is to scan the dataset again examining the baskets and find out the itemset with the max count, that is taken at the top; than the branch of the tree is constructed with itemsets in descending order of count.

The next basket is examined and, if any of its itemset is already present in another branch, then this basket branch would share a common prefix to the root. This means that the common itemset is linked to the new node of another itemset in this transaction.

Also, the count of the itemset is incremented as it occurs in the baskets. Both the common node and new node count is increased by 1 as they are created and linked according to baskets.

The further step is to mine the created FP tree. For this, the lowest node, which represents the frequency pattern length 1, is examined first along with its links. From this, it traverses the path, which is called "conditional pattern base", in the FP tree.

In the next step, the algorithm constructs a conditional FP tree, which is formed by a count of itemsets in the path. The itemsets meeting the threshold support are considered in the conditional FP tree. Finally, frequent patterns are generated from the conditional FP tree.

5.1 Implementation and Results

To implement the FP-Growth algorithm, I used the built-in function of the pyspark library. First, the model is fitted on the baskets dataset, then frequent itemsets, association rules and predictions are computed for each basket.

From the output shown we can see itemsets of different size, but the skills contained are very similar to those found by the Apriori algorithm. Some examples are: "critical thinking", "communication", "team leadership", "customer service" etc.

The main difference is that I didn't have to do three different steps to calculate singletons, pairs and triplets, because it's all done automatically by the fpGrowth.fit() function.

Finally, it's quite obvious that the skill "communication" is the one with the highest confidence in appearing in an itemset together with other skills; it's also often considered in predictions.

6 Conclusion

In this report, I applied both the Apriori and FP-Growth algorithms to a LinkedIn dataset of jobs and associated skills.

After preprocessing the data and preparing it for analysis, I used these algorithms to identify frequent itemsets and generate association rules.

The Apriori algorithm, though effective, required multiple steps to discover frequent itemsets, whereas the FP-Growth algorithm was more efficient, generating frequent patterns without the need for candidate generation.

Despite the differences in the approach, both algorithms yielded similar results, with skills like ”communication” and ”problem solving” emerging as highly frequent and commonly associated with other skills.

7 Final 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, and including any code produced using generative AI systems. 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.