Università degli Studi di Milano

Data Science for Economics

Algorithm for Massive Data:

Market-basket analysis on skills for job positions

Student:
Beatrice Cagnin

ACADEMIC YEAR 2023-2024

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 offenses 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

The aim of this project is the implementation of market-basket analysis through A-Priori algorithm, in order to find which skills or set of skills appear more frequently on job positions proposed by LinkedIn.

The main feature used to process the involved large amount of data is PySpark,the Python API for Apache Spark, that enables to work with massive datasets. The advantage of using it is that it allows to distribute the data across multiple nodes and so to work efficiently with large amounts of information.

After having analyzed the dataset and performed some simple data cleaning, it is taken a sample of the whole dataset on which the A-Priori algorithm is applied. First, frequent itemsets have been found separately (singletons, pairs and triples), then a function is defined to apply the algorithm on the loop for different k-itemsets.

2 Dataset

The dataset is downloaded using personal Kaggle API username and key. It contains three .csv files: job_skills.csv, job_summary.csv and linkedin_job_posting.csv. For this project, only the first one has been considered.

The created dataset is composed of two columns: 'job_link' and 'job_skills', so each row contains all the skills required for the corresponding job position. Indeed, each cell contains a string which is the list of the desirable skills attached to the job offer.

job_link	job_skills
https://www.linke...	Building Custodia...
https://www.linke...	Customer service,...
https://www.linke...	Applied Behavior ...
https://www.linke...	Electrical Engine...
https://www.linke...	Electrical Assemb...
https://www.linke...	Access Control, V...
https://www.linke...	Consultation, Sup...
https://www.linke...	Veterinary Recept...
https://www.linke...	Optical Inspectio...
https://www.linke...	HVAC, troubleshoo...

Figure 1: Project dataset

In particular, the project is based on the 'job_skills' column, since the final aim is to find which skills or set of skills are more frequent in job positions. Given that working with all the 1.294.374 rows takes a lot of time, a random sample of 12.851 rows (corresponding to 1% of them) is considered.

2.1 Data preparation

As mentioned before, only 'job_skills' column is considered. After having deleted null values and transformed that column from row type to list, it is assigned to a RDD (Resilient Distributed Dataset), a tool of PySpark that allows to break data into partitions, two in this case, and to split across various clusters.

Assuming that there is no job offer with a duplicate skill, baskets and items are retrieved applying map() and split() functions. In particular, to obtain baskets is used standard map() function, that returns a list of lists of strings, while for items is employed flatMap() function, that outputs a list of strings.

For space efficiency, baskets are converted into sets of indexes representing items. To do that, items are transformed into dictionaries having the item name as key and a positional index as value.

3 Theory related to market-basket analysis and a-priori algorithm

3.1 Market-basket analysis

The market-basket model of data is a many-many relationship between two kinds of objects, *items* and *baskets*, where each basket consists of a set of items, indeed it's also called itemset. Some assumptions are generally made on the shape of the data: the number of items in a basket is small, the number of baskets is very large and the data are represented in a file consisting of a sequence of baskets.

A set of items is frequent if it appears in many baskets. Considering s as the threshold support, I as a set of items, *support for* I as the number of baskets for which I is a subset, it can be said that 'I is frequent if its support is equal or greater than s'.

3.2 A-Priori algorithm

The A-Priori algorithm is the most used in market-basket analysis to find frequent itemsets within large datasets. Essentially, it works by eliminating most of the candidate large sets by looking first at smaller sets and recognizing that a large set cannot be frequent unless all its subsets are; the latter rule is called *monotonicity* of itemsets.

This algorithm uses a minimum support threshold, that defines how frequent an itemset needs to be in order to be considered "interesting" for the analysis. Items or itemsets that appear less frequently than the threshold are discarded. Therefore, it is crucial to select a support that filters most of the data (to maintain the algorithm light) while not discarding interesting connections.

4 Algorithm application

The application of the A-priori algorithm starts with the definition of the threshold. It is set to 257, calculated as the 2% of the number of baskets, which means that an itemset is frequent if it appears more than 257 times. As before it's not used the common percentage, 1% for time-running efficiency.

As stated before, the algorithm is applied first separately for singletons, pairs and triples, and then a function is created to generates frequent increasing k-itemsets, as long as every subset of size k-1 is frequent (monotonicity constraint).

The A-priori algorithm is composed of various passes.
The first pass begins with a *.flatMap()* transformation: it examines all the items in all the baskets and for every item in a basket creates a key-value pair with the item as the key and 1 as the value. Next it performs *.reduceByKey()* that sums all the 1s for a given key, meaning

that outputs key-value pairs with the integer of the skill and its count in the .baskets. Finally with *.filter()* , the algorithm keeps only the items that has a value greater than the support. After counting the number of frequent singletons, a new variable is created, named 'frequent_singletons', to keep track of them. Then the most five frequent singletons have been retrieved with their frequencies, thanks to the hash table, 'reverse_it_index' in the project.
In the second pass, as well as in the third one, the method is almost the same with a difference in the *.flatMap()*: for frequent pairs the function examines all unordered candidate pairs whose singletons are both in basket and are both in 'frequent_singletons', for frequent triples the function examines all unordered candidate triples whose singletons are all in basket and underlying pairs are frequent, so are in 'frequent_pairs'.

In order to find frequent itemsets of all possible sizes, a generalized A-Priori algorithm is applied. The structure is similar to the previous explained but it's inside a loop.
The reason why the first pass is taken distinct from the loop is that the latter generates k-itemsets using combinations that are filtered in *.flatMap()* to ensure that every subset of size k-1 must also be in the set of frequents (monotonicity assumption), in order to avoid counting for all possible tuples candidates to be the frequent itemsets. This is not done in the first pass because all the candidate singletons are frequent.

This is also the cause of the possible differences between the previous separated approach and this general approach: in the case of k=2 for example, the first algorithm only checks that the singletons are individually frequent, without explicitly checking the relationship between the pairs; in the loop approach each item in the pair must be part of a valid subset of k-1 items, thus requiring that both singletons must be in frequents, but also that any k-1 subsets of each pair must be frequent.

All of this is done because the space required for the operation decreases drastically while looping.

Going back to the generalized A-Priori algorithm loop structure, once candidate pairs are generated, they are filtered for the support, thus obtaining frequent pairs. 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 until there are no frequent itemsets of a certain size, so no itemset appears in a number of baskets greater than the support.

5 Frequent skills on LinkedIn job positions

Applying the **A-priori algorithm** on job_skills RDD **outputs**:

- 66 frequent singletons for both the algorithm applied singularly and the generalized one
- 134 frequent pairs for the singular approach and 43 for the generalized one
- 5 frequent triples for both algorithms
- no frequent itemsets of size 4 (stated by the generalized approach)

The first five most **frequent singletons** are:

1. 'Communication', with frequency = 3605
2. 'Teamwork', with frequency = 2220
3. 'Leadership', with frequency = 1826
4. 'Customer service', with frequency = 1588
5. 'Communication skills', with frequency = 1118

The first five most **frequent pairs** are:

1. ('Teamwork', 'Communication'), with frequency = 2728
2. ('Communication', 'Leadership'), with frequency = 2322
3. ('Customer service', 'Communication'), with frequency = 1552
4. ('Problemsolving', 'Communication'), with frequency = 1344
5. ('Teamwork', 'Leadership'), with frequency = 1316

The first five most **frequent triples** are:

1. ('Teamwork', 'Communication', 'Leadership'), with frequency = 510
2. ('Problemsolving', 'Teamwork', 'Communication'), with frequency = 402
3. ('Teamwork', 'Customer service', 'Communication'), with frequency = 380
4. ('Problemsolving', 'Customer service', 'Communication'), with frequency = 321
5. ('Teamwork', 'Communication', 'Problem Solving'), with frequency = 295

6 References

Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman.
Mining of Massive Datasets, Chapter 6.