UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE POLITICHE, ECONOMICHE E SOCIALI

Data Science and Economics

Market Basket Analysis

Implementation of A-Priori Algorithm

Luca Bertoletti

luca.bertoletti1@studenti.unimi.it

Contents

1 Introduction

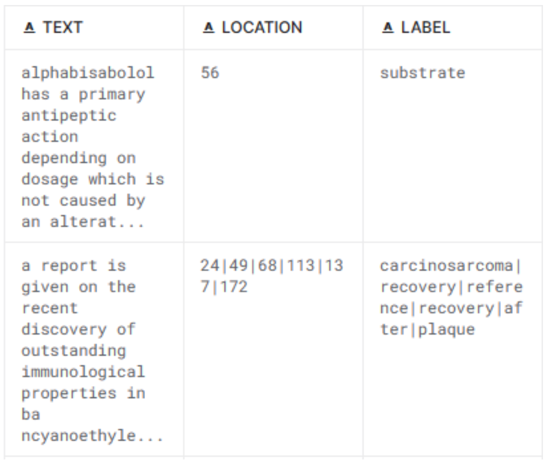
This project aims at performing a data mining technique, known as Market Basket Analysis, to find frequent itemsets. A frequent itemset is a set of items that occur together frequently in a dataset. The frequency of an itemset is measured by the support count, which is the number of transactions, or records, in the dataset that contain the itemset. Frequent itemset mining is a technique used to discover relationships between items in a dataset. It can be used to identify products that are often bought together, customer segments and other patterns in data.

Market Basket Analysis can also be applied to types of data different from customer transactions. In this project, indeed, the technique is applied to text data: in this case, items are words and baskets are sentences and the goal is to find frequent combinations of words in sentences.

In order to perform the analysis I implemented an A-Priori algorithm in Spark.

2 Dataset

The dataset used in this analysis is the MeDAL dataset [1](Medical Dataset for Abbreviation Disambiguation for Natural Language Understanding), a large dataset of medical texts curated for the task of medical abbreviation disambiguation, which can be used for pre-training natural language understanding models. The MeDAL dataset consists of 14,393,619 articles and reaches a size of almost 15GB. It consists of 3 columns: 'text', that contains the normalized content of an abstract, 'location', the location (index) of each abbreviation that was substituted and 'label', the word at that was substituted at the given location.



TEXT	LOCATION	LABEL
alphabisabolol has a primary antipeptic action depending on dosage which is not caused by an alterat...	56	substrate
a report is given on the recent discovery of outstanding immunological properties in ba ncyanoethyle...	24|49|68|113|137|172	carcinosarcoma|recovery|reference|recovery|after|plaque

Figure 1: Dataset preview

2.1 Data Pre-Processing

For the goal of this project, only the column 'text' has been considered. Each row of the dataset can be considered as a basket and words (and their combinations) can be considered as items. Therefore, the goal is to find words, or combination of them, that appear frequently in paper's abstracts.

Given the high dimensions of data, only a subset of rows (14357) has been considered. The dataset has then been converted into RDD (Resilient Distributed Dataset).

Since we are dealing with text, a function for pre-processing sentences have been applied for an easier algorithm execution. In particular, the function, in addition to remove punctuation and stopwords, it also lower-case, lemmatize and tokenize words.

An example of the processed text is shown below.

row = ['clostridium perfringens sialidase was purified by affinity chromatography kinetic properties of the enzyme were examined with sialyllactose and with mixed sialoglycolipids gangliosides as substrates with the latter ATP ...']

tokens = ['clostridium', 'perfringens', 'sialidase', 'purified', 'affinity', 'chromatography', 'kinetic', 'property', 'enzyme', 'examined', 'sialyllactose', 'mixed', 'sialoglycolipids', 'gangliosides', 'substrate', 'latter', 'atp', ...]

As last step of data pre-processing, I assigned to each word an integer because integers typically take up less space than strings.

3 A-Priori Algorithm

The A-priori algorithm works by exploiting the monotonicity property, which tells us that if an itemset I is frequent, then all its subsets are frequent as well. If no frequent itemsets of a certain size are found, then monotonicity tells us there can be no larger frequent itemsets, so we can stop. Thanks to this property, we can reduce the number of possible candidates to evaluate.

The algorithm has been implemented in the code through the 'apriori' function, that takes an RDD and a minimum support threshold as parameters. It then returns an RDD containing all the frequent items and their occurrences. The threshold (minimum support) has been calculated as

$$\text{threshold} = \text{number of baskets} * \text{fraction of number of baskets}$$

To be considered frequent, itemsets must exceed the threshold.

The algorithm starts by computing frequent singletons and then combines them to get frequent pairs. A loop starts to find frequent itemsets of size (indicated by letter 'i' in the code) greater than two :

- The output of the previous step is saved into an RDD, then new singletons are generated from this output and saved into another RDD.

- The new candidates are generated through the Cartesian product between the new singletons and the output

- Candidates are sorted to avoid permutations to be counted as different item and to remove the repetitions of words

- Frequency of candidates is checked and, if larger than threshold, they are added to the output

- The loop continues with larger itemsets until there are no more possible combinations.

3.1 Results

After having tried different values, I chose 0.03 as the threshold percentage. This value allowed to obtain results in a reasonable computational time. The plot in the Figure below shows the number of occurences for each itemset size.

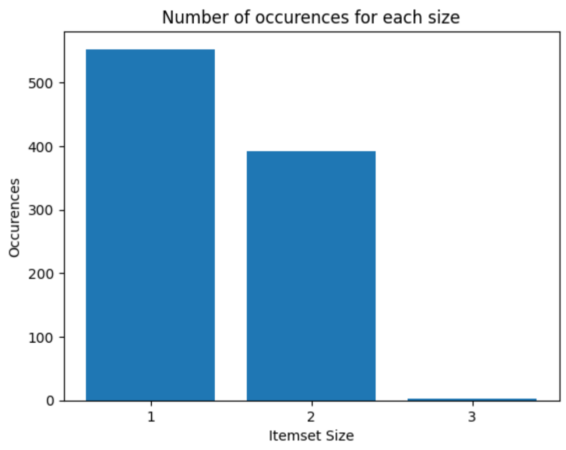


Figure 2: Number of occurences for each itemset size

The plot shows that the number of frequent singletons is way higher than the number of occurences in the other 2 sizes. As we expected, the number of frequent triples is really small and no itemsets of higher cardinality were found. In particular, only 2 frequent triples were found.

I also tried to lower the the threshold but the required time to run the algorithm were too much.

4 Computational time with increasing minimum support

To check how the algorithm behaves with different thresholds, I used a sample (25%) of the RDD. The sample is needed to allow the algorithm to complete in reasonable time.

The plot below shows the results of the experiment of increasing the threshold percentage from 0.03 to 0.08.

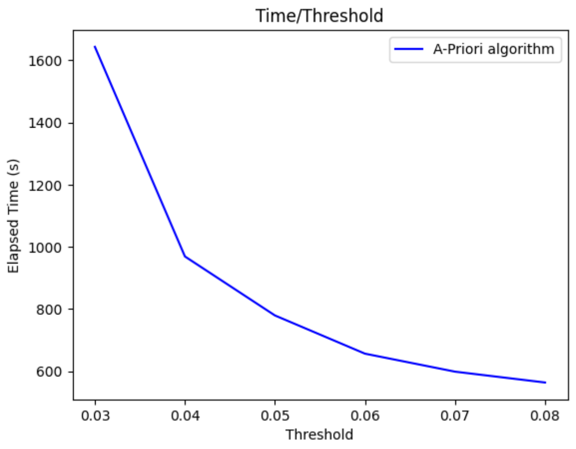


Figure 3: Time / Threshold

As expected, the time needed decreases with the increasing threshold showing an exponential relation. We pass from 1600 seconds (almost 27 minutes) for the smallest threshold to 10 minutes for the largest one.

5 Scalability

The use of Spark's RDD and Spark's functions such as 'map', 'FlatMap','reduceByKey' and 'filter' allows the scalability of the solution.

References

[1] Zhi Wen, Xing Han Lu, and Siva Reddy. Medal: medical abbreviation disambiguation dataset for natural language understanding pretraining. *arXiv preprint arXiv:2012.13978*, 2020.