Market Basket Analysis in a Big-Data Context

Federico Bassi, ID:993443

Algorithms for Massive Datasets Project, March 2022

1 Introduction

Market-basket analysis is a popular data mining technique which aims at discovering meaningful associations between entities by exploiting their co-occurrences. Given a set of transactions, or baskets, we would like to determine which items are frequently together in a transaction.

Market-basket analysis can be extended to types of data different from the retailing context, such as texts: in this case, items are words and baskets are sentences and we would like to discover meaningful co-occurrences between words in the sentences.

The following project aims at performing a market basket analysis on textual data using big-data approaches. In particular, in order to find frequent itemsets, i.e. combinations of words frequently associated in a sentence, I develop the Apriori algorithm using Pyspark, allowing parallelization of the computation.

The following report is organized as follows: Section 2 describes the dataset used to perform the analysis and the way in which data have been organized and pre-processed. Section 3 describes the implementation of the Apriori algorithm, while Section 4 explains why this algorithm could be scaled to handle big-data. Finally, Section 4 and 5 describe respectively the experiments and a discussion of the results obtained.

2 Dataset, data organization, data pre-processing

The Dataset used to perform the analysis is the MeDAL dataset (Medical Dataset for Abbreviation Disambiguation for Natural Language Understanding), published on Kaggle and released under the Apache 2.0 license. The dataset contains medical texts and has been curated for abbreviation disambiguation in the medical domain (Wen, Lu, and Reddy 2020). In particular, the dataset contains three columns: ”text”, which contains the abstract of a paper in the medical science where a term has been substituted by its abbreviation, ”location”, which describes the index of the substituted word and ”label”, which contains the word that was substituted at the given location.

In the context of this analysis, only the column ”text” has been retained. Each row of the dataset has been considered as a basket and words and combination of words have been considered as itemsets. Therefore, the goal of our analysis will be the one of finding frequent words and combination of words inside papers' abstracts.

Data have been downloaded from Kaggle and unzipped. Since the size of the file was high (almost 15 GB and 14,393,619 articles), only a subset of 1000 rows was retained and used for the analysis. The distribution of the lengths of the (tokenized) documents is displayed in Fig. 1.
The dataset has then been converted to a PySpark RDD (Resilient Distributed System), a data structure that can be divided into multiple nodes of a cluster to run parallel processing.

As is commonly done in text-related tasks, sentences have been pre-processed in order to disambiguate words and facilitate the algorithm. In particular, words have been put in lowercase, lemmatized and tokenized. Moreover, stopwords and punctuation have been removed.

For example, this is the result of the preprocessing process on the first row of our dataset:

row = ['alphabisabolol has a primary antipeptic action depending on dosage which is not caused by an alteration of the phvalue the proteolytic activity of pepsin is reduced by percent through addition of bisabolol in the ratio of the antipeptic action of bisabolol only occurs in case of direct contact in case of a previous contact with the ATP the inhibiting effect is lost']

tokenized_text = ['case' ,'alteration', 'direct', 'alphabisabolol', 'effect', 'contact', 'bisabolol', 'depending', 'primary', 'previous', 'action', 'ratio', 'atp', 'caused', 'pepsin', 'antipeptic', 'dosage', 'phvalue', 'proteolytic', 'addition', 'percent', 'reduced', 'inhibiting', 'occurs', 'lost', 'activity']

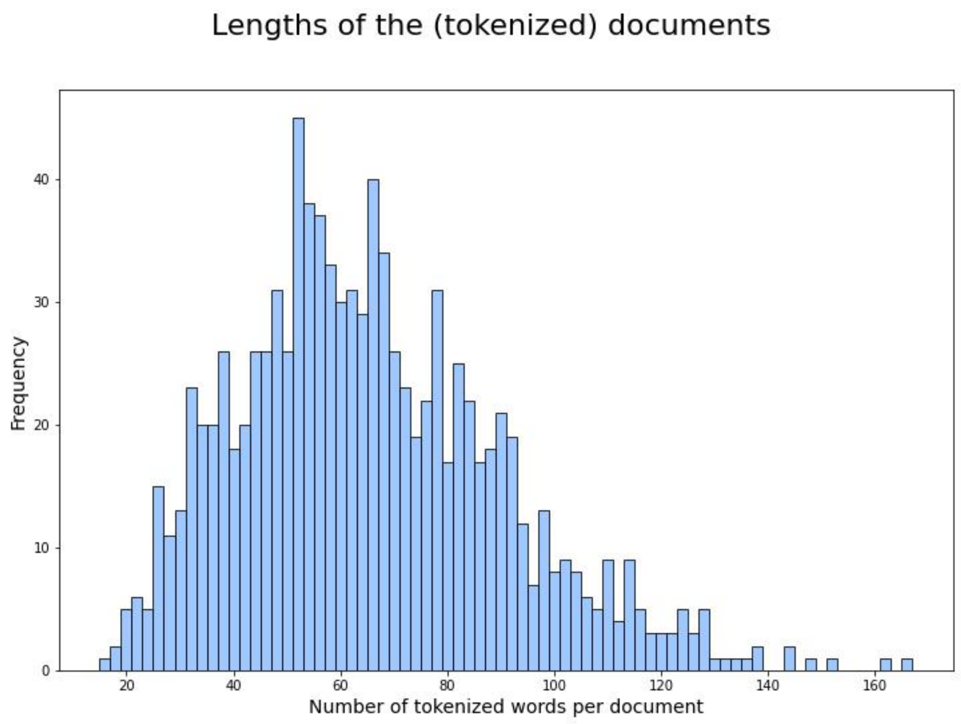

Figure 1: Lengths distribution

Finally, in order to optimize memory space, each string has been converted to an integer index by the function "create_indexing".

3 Algorithm

One of the most popular technique to mine frequent itemsets in a big-data context is the "Apriori" algorithm. Apriori works by exploting the "monotonicity property": if an itemset S is frequent, then all its subsets are frequent as well. Using this property, Apriori eliminates larger candidates sets by computing first the frequency of lower sets.

Apriori has been implemented in the code by the "apriori" function, which takes as input a Pandas dataframe, the percentage threshold (set by default as 1% of the number of baskets) and the maximum itemset size (set by default at none). The function outputs a list of frequent itemsets with the corresponding support. "Frequent" is defined according to the percentage threshold specified as argument. If the maximum itemset is not specified, the algorithm computes the frequences of sets of progressively higher cardinality, until no candidate is found.

The algorithm implementation is based -intuitively- on two phases. For each itemset size k:

- Scan the data and count the occurrences of each candidate (at the first iteration, all the words will be considered candidates). Filter the itemsets that exceed a given threshold and add those to the list of result;
- Compute the candidates for the next iteration by computing all the possible combination of size k+1 from the frequent itemsets found at the current iteration;

The algorithm stops when no candidate for the next iteration is found or when the maximum itemset size specified as an argument has been reached.

4 Scalability

Even if the current analysis has been carried out using a reasonable portion of the original dataset (1000 abstracts), the code has been implemented in order to be easily scalable, i.e. executed in a distributed and parallelized way. This was made possible by the use of Spark's RDD data structures and Spark's functions such as "map", "flatMap", "reduceByKey".

Spark has not been used when the data to be stored has reasonable size, i.e. it can be stored in RAM. This is the case, for example, of the list containing the frequent itemsets.

5 Experiments

Once the algorithm has been implemented and tested, I performed different kinds of experiments on the dataset in order to study:

- The evolution of the computational time as a function of the maximum itemset size and of the threshold;
- The number of itemsets found for each of the maximum itemset size;
- Confidence and lift for association rules.

5.1 Computational time as a function of max itemset size and threshold

First, I studied the evolution of the computational time of the algorithm as a function of the maximum itemset size and threshold. As Figure 2 shows, the time needed to perform the computation rises when the maximum itemset size augments. Moreover, the more the threshold is low, the more the computational time goes up.

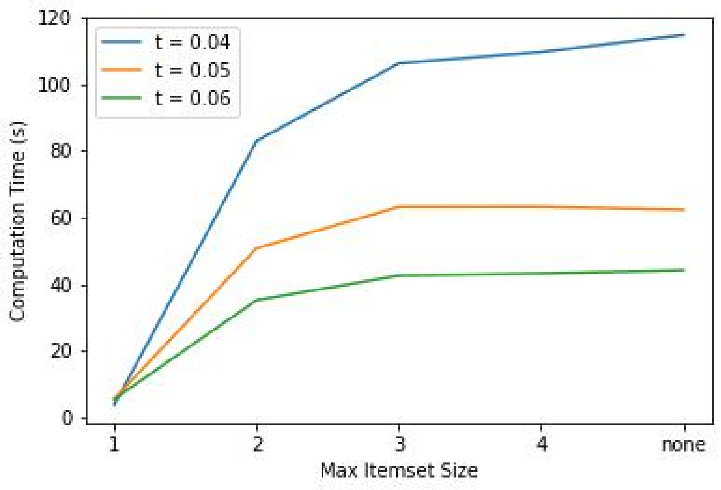

Figure 2: Computation Time

The results are in line with expectations: when the the maximum itemset size rises, the algorithms needs to add a pass through the data, while when the threshold is lowered, the number of itemsets that the algorithm consider rises. In both cases, we expect higher computational time.

5.2 Number of frequent itemsets for each maximum itemset size

Second, for a fixed threshold (20% in this case), I computed the number of frequent itemsets found by the algorithms as a function of the maximum itemset size.

The results of this experiment are displayed in Fig. 3. From this plot we can see that the number of frequent pairs is considerably higher than the number of frequent singletons. As expected, the number of frequent triples (and of itemsets of higher cardinality) is less than the number of frequent pairs.

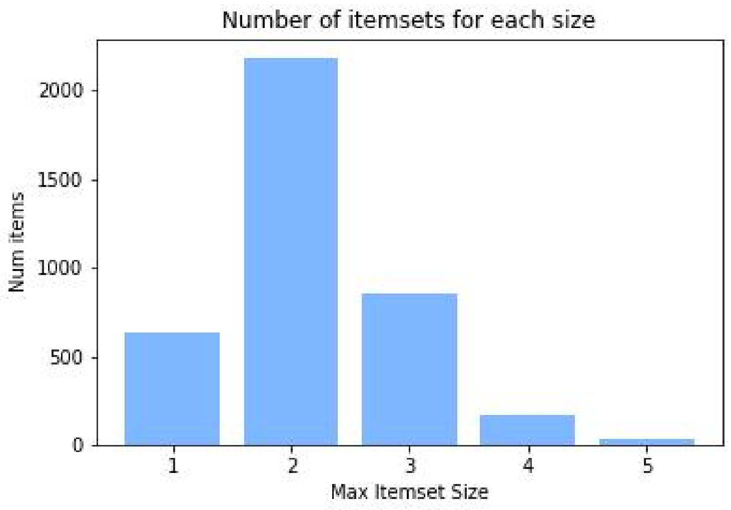

Figure 3: Number of frequent itemsets for each maximum itemset size

5.3 Confidence and lift for association rule mining

The final experiment aimed at computing confidence and lift for the possible association rules between frequent itemsets.

In particular, given an association rule A → B, confidence is defined as:

$$confidence(X \to Y) = \frac{supp(X \cup Y)}{supp(X)} \tag{1}$$

while lift is defined as:

$$lift(X \to Y) = \frac{supp(X \cup Y)}{supp(X) * supp(Y)} \tag{2}$$

It turns out that, the association rule with the highest confidence in our dataset is:

{'gel', 'weight'} → {'molecular'}

while the rule with highest lift is:

{'electrophoresis'} → {'gel'}

References

Wen, Zhi, Xing Han Lu, and Siva Reddy (Nov. 2020). "MeDAL: Medical Abbreviation Disambiguation Dataset for Natural Language Understanding Pretraining". In: *Proceedings of the 3rd Clinical Natural Language Processing Workshop*. Online: Association for Computational Linguistics, pp. 130–135. DOI: `10.18653/v1/2020.clinicalnlp-1.15`. URL: `https://aclanthology.org/2020.clinicalnlp-1.15`.