Market Basket Analysis

Algorithms for Massive Data project

Alessia Poli (989824)   Eleva Valentina Serbu (988351)

May 2023

Abstract

This report inspects how *Market Basket Analysis* performs on the MeDAL dataset, a medical abbreviation disambiguation dataset for natural language understanding. We implemented the well known *A priori algorithm* to discover frequent itemsets and association rules from transactional data. The analysis requires some preliminary processing of data, so they will also be object of discussion. And, finally, we understand the reults of the analysis by focusing on three metrics of Market Basket Analysis: support, confidence, and interest.

1 Introduction

Market Basket Analysis is a data mining technique, aiming at discovering associations between items that are frequently observed together. By identifying co-occurrences, MBA defines association rules, as the representation of relationships among set of items.

We implemented MBA on the MeDAL dataset (*Medical Dataset for Abbreviation Disambiguation for Natural Language Understanding*), a large medical dataset designed for natural language understanding. MBA performed on text data analyses the co-occurrences of words in sentences, to identify some patterns among text structures. So, in this particular analysis words are items, and sets of words will form the baskets. To do so, we developed MBA using the *A Priori algorithm*, with Pyspark, allowing parallelization of computation.

MBA requires some processing and preparation of data. Once we have performed the preliminary operations which will be described in Section 2, we can actually run the algorithm. To get a complete understanding of the steps and procedures implemented, Section 3 aims at describing the theoretical framework behind MBA and the A Priori algorithm. Section 4 describes the phases of the Market Basket Analysis, and the results of our analysis performed on the MeDAL dataset. Finally, in Section 5 we draw the conclusions of such study.

2 The dataset and data preparation

To implement MBA, we imported from Kaggle the MeDAL dataset (*Medical Dataset for Abbreviation Disambiguation for Natural Language Understanding*). It consists in 15 GB of medical texts, which had been curated by Wen, Lu, and Reddy (2020) for abbreviation disambiguation, and for natural language understanding pre-training in the medical domain. The sentences to feed to the algorithm are in the *text* column of the *full-data.csv* file, using words as items.

Each row corresponds to each of the abstracts of the 14,393,619 medical articles collected by the authors. We aim at finding recurrent combinations of words inside the sentences in the abstracts, in order to define association rules, so that the presence of a set of words implicate another word, or combination of words.

The file is extremely heavy because of its dimension (15 GB). So, due to computational reasons, and for the purposes of our own analysis, we only consider a random sample of the texts 0.001% of the original dataset).

Besides retaining only a portion of the whole dataset, we also have performed some preprocessing operations to prepare data for the main analysis. Data preparation has two goals: on one hand to reduce the time required by the algorithm to read data and to produce candidates for finding frequent itemsets, on the other hand to eliminate those elements from the sentences that would not have any meaning and usefulness when inspecting the output of the analysis.

The dataset has been transformed in an RDD (Resilient Distributed Dataset), an Apache Spark data structure distributed collection of elements of data, partitioned across nodes in a cluster, that can be processed in parallel. More specifically, installing PySpark in a Google Colab setting allows to exploit the power of distributed computing, when working with large datasets, ando more importantly to perform a scalable data analysis.

By converting the dataset into an RDD, we only extract the columns with texts. Once we have retrieved them, we perform some operations directly related with texts:

- remove punctuation,
- remove stopwords,
- perform tokenization, by breaking up the text into tokens, that correspond to words,
- perform lemmatization, grouping inflected forms of a word, based on its lemma, so they can be considered as a single item.

E.g., a row from the original dataset is presented in following form:

TEXT = [the effect of hip joint surgery on routine SS enzyme values was studied the increase in creatine kinase was most marked and normal levels were restored again T3 approximately week asparatate aminotransferase was slightly elevated throughout the first two postoperative weeks alanine aminotransferase was essentially unchanged during the first three PODs in contrast to lactate dehydrogenase which reached preoperative values again T3 only days alkaline phosphatase showed an increase after week whereas samylase was essentially normal throughout the weeks studied']

and after preprocessing it appears as follows:

TEXT = ['level', 'lactate', 'phosphatase', 'kinase', 'surgery', 'postoperative', 'slightly', 'effect', 'first', 'pod', 'dehydrogenase', 'enzyme', 'aminotransferase', 'studied', 'approximately', 'normal', 'alanine', 'whereas', 'showed', 'elevated', 't3', 'three', 'week', 'contrast', 'asparatate', 'unchanged', 'day','marked', 'alkaline', 'value', 'throughout', 'two', 'hip', 'creatine', 'essentially', 's', 'reached', 'routine', 'preoperative', 'increase', 'samylase', 'restored', 'joint'].

Then, we encoded our dataset so that we have a vocabulary of words that are actually contained in the text. This operation allows to not waste memory, as it is more space-efficient to represent items by consecutive integers from 1 to n. When encoding words in a vocabulary, words are taken only once, even when we incur in duplicates. In fact, words are mapped in a key-value pair, in which the word is the key and the value corresponds to its index: repeated words will have the same value, and so collected only once inside the vocabulary. To save even more memory, the vocabulary is also built with the value indexes instead of the word themselves. So, the sentence we used as an example above will also look like this:

[66791, 121031, 60696, 0, 142078, 217510, 295926, 112051, 301929, 18249, 250597, 133022, 112052, 84868, 66792, 166236, 181306, 121032, 232582, 15205, 99984, 323153, 193449, 244571, 51498, 139119, 253637, 184384, 136046, 48517, 12152, 21287, 298949, 229637, 265759, 290017, 175330, 235552, 244572, 78858, 33309, 99985, 90898, 136047, 124111, 323154, 115084, 223572, 93968, 33310, 187354, 298950, 103002, 320146, 130029, 81824, 1]

After a final step of caching, the data preprocessing is completed and the data is ready for the algorithm implementation.

3 MBA: theoretical framework

3.1 Market Basket Analysis

Market Basket Analysis is a data mining technique, that was first used by retailers to unveil patterns in customers' behaviour. MBA finds its applications in other context too, e.g., finding combinations of words in documents that are recurrently observed. So, this analytical approach has been generalized to the concept of finding frequent itemsets, in order to define association rules.

Association rules count the frequency of items that occur together, seeking to find associations that occur far more often than expected. They produce a pattern consisting of two parts:

- the *antecedent* which is declared as *if*,
- and *consequent* which is declared by *then.*

Association rules are represented in the following form: $I \rightarrow \{j\}$. Where I is the antecedent, which consists in an itemset of one or more word, and $\{j\}$ is the consequent, which can be either set of words or a signle word.

Market basket analysis allows to find associative rules between a combination of items, in this specific case of words. Each basket consists in a set of items. The number of items in a basket, generally, is much smaller than the total number of items. On the contrary, the number of baskets is assumed to be very large.

To identify the most frequent baskets we use the notion of support. *Support* of an itemset I is the number of itemsets in a dataset containing all items in I; most of the times, it is computed as a percentage. Given a support threshold s, a set of items is said to be frequent if and only if it appear at least s times.

A further metric able to define frequent itemset is *Confidence*. *Confidence* is defined as the ratio of the number of itemsets that include all items in I and j to the number of itemsets that include all items in I. It is the percentage of baskets, that among all elements contain I, also contain j.

$$Conf = \frac{\text{supp}(I \cap \{j\})}{\text{supp}(I)} \tag{1}$$

Confidence ranges from 0 to 1: the closer the value to 1, the more likely item j will be part of the frequent itemset I.

Moreover, not all high-confidence rules are to be considered interesting. A rule may have high *confidence* just because of an elevated number of occurrences of a certain item. Preoprocessing should help in this sense, by eliminating words recurring many times, such as stopwords. However, *interest* of the association rule is formalised as follows:

$$\text{Interest} = \text{conf}(I \to \{j\}) - \text{pr}[I] \tag{2}$$

And should be interpreted as described:

- If $= 0$: I has no influence on j.
- If $>> 0$: the presence of I in a basket causes the presence of j.
- If $<< 0$: the presence of I in a basket discourages the rpesence of j.

It must be assumed that there is not a too large number of frequent itemsets. The *support* threshold should be adjusted to find a reasonable number of frequent itemsets.

3.2 The A Priori algorithm

The *A-Priori algorithm* is one of the most well-known algorithms used in association rule mining. It identifies the items in a dataset and further extends them to larger and larger item sets. This data mining technique works by exploiting the *monotonicity property*: if a set I of items is frequent, then so is every subset of I.

The *A-Priori Algorithm* is designed to reduce the number of pairs that must be counted, at the expense of performing two passes over data, rather than one pass. It is structured in two main passes:

1. Candidates generation: the algorithm generates a set of candidates based on a minimum support threshold provided as input. The first pass produces two tables. The first table, if necessary, translates item names into integers from 1 to n, n being the dimension of the dataset. Such step, in our case, is not performed, as we created our own vocabulary of words contained in the dataset. The latter table is an array, in which each element counts the occurrences of the corresponding item.
2. Support calculation for each k-itemset candidate, with k being the dimension of the itemset. The algorithm produces an array indexed from 1 to m, m being the number of items, whose entries is either 0 or 1, whether the item is frequent or not. Items are considered frequent based on their level of *support*, given that it is higher than the fixed threshold s. The output of the second pass is an array of the most frequent itemset.

The process continues until no more frequent itemsets can be generated and association rules are generated by iterating over each frequent itemset and generating all possible rules based on the itemset. By going through these phases, the a priori algorithm effectively identifies frequent itemsets and association rules, providing insights into item and itemsets occurrences, and into word patterns in sentences in the medical dataset.

4 Algorithm implementation and results

We implemented the *Market Basket Analysis* and the *A priori algorithm* on a sample of the MeDAL dataset. We imposed a random draw of a sample of 0.001%[1], as it allowed us to have an amount of data that lead to significant results, and to run the algorithm for a reasonably long computational time, as we used Pyspark in Google Colab. The same analysis could be performed on bigger size, as it is executed in a distributed and parallelized way. Such approach is helpful for us, as we want to inspect the functionality of the algorithm rather than its scalability.

The whole process of the a priori algorithm is performed iteratively. Firstly, we computed the process to identify the most frequent items (*singletons*). They are found following these steps:

- Compute the minimum support threshold for items to be considered as frequent. We set this value as the 10% of the number of elements in the RDD: RDD count $* 0.1 = 1441.76$.
- Singletons are found as the items that recur at least s times in the dataset. They are used to compute the candidates to become frequent pairs.
- A MapReduce task is implemented to pair words with the corresponding number of occurrences: Map: after flattening the sentences in the RDD, the Map task receives as inputs the words from each text and produces a key-value pair associating to words (key) the value 1 for each token.

  $$\text{Map: word} \rightarrow (\text{word}, 1)$$

  In an intermediate step, the outputs of the map sharing the same key are grouped in a list of the form:

  (word, [1, 1, ...])

  the list of 1 has length equal to the number of times the word is encountered. Reduce: then, in the reduction operation the values of a key are summed up. So, given that the values associated to a word are all 1, the summation corresponds to the length of the list of 1.

  $$\text{Reduce: } (\text{word}, [1, 1, ...]) \rightarrow (\text{word}, m)$$

  where m is equal to the number of 1. Moreover, key-value pairs with value below the minimum support threshold are filtered out.

[1]A sample of the 0.001% of the whole dataset requires many hours for the implementation of the whole algorithm. For this reason we suggest to decrease such sample to rerun the whole algorithm, for the sake of verifying how it works

- Finally, the output of the Reduce step are filtered, so that only key-value pair having value at least s will be considered. The filtering phase produces the most frequent singleton.

Reminder: in each of the steps of the algorithm, the tasks are performed with the vocabulary containing the index values corresponding to each word, for computational reasons, and, then, once the most frequent items are obtained, it is converted to words. So, here is an example of output produced by the MapReduce:

('treatment', 2349) $\rightarrow$ the word "treatment" recurs 2349 times
('patient', 3569) $\rightarrow$ the word "patient" recurs 3569 times

In the next step, we used the most frequent singletons to produce the most frequent pairs of words in the dataset. Frequent singletons are combined to create pairs, that will be the the candidates fed into the MapReduce, which will end up in giving the most frequent pairs as result.

- The Map step receives the combinations obtained, and return a key-value pair, where key is the pair of items and the value will be equal to 1. Such result is produced only for pairs that are actually found in the text data.

$$\text{Map: Pair candidate} \rightarrow ((\text{word}_1, \text{word})_2), 1)$$

  Pairs could be observed multiple times in the a sample of MeDal dataset, so we may have multiple identical key-value pairs, as output of the Map step. The values associated to repeating keys should be grouped in a list as follows:

$$((\text{word}_1, \text{word}_2), [1, 1, ...])$$

  where 1 is repeated as many times as the pair is observed.

- In the Reduce phase values in the key-value pairs are summed.

$$\text{Reduce: } ((\text{word}_1, \text{word}_2), [1, 1, ...] \rightarrow ((\text{word}_1, \text{word}_2), m),$$

  m being the number of times the pair is observed (*support*), which is equal to the number of times 1 repeats in the list of values.

- Finally, these key-value pairs are filtered, so that the MapReduce only returns pairs of words that have support higher than the fixed threshold.

Two of the most frequent pairs are:

(('result', 'increased'), 691) $\rightarrow$
the pair of words indexed *result* and *observed* recurs 691 times

(('treatment', 'study'), 830) $\rightarrow$
the pair of words indexed *treatment* and *study* recurs 830 times

Pairs produced by the MapReduce function are combined with a third item from the vocabulary, in order to find triplet candidates to find frequent itemsets of three words. The operations performed on triplets are exctly the same as the once applied to find frequent items and frequent pairs:

$$\begin{gathered}\text{Map: Triplet candidate} \rightarrow ((\text{word}_1, \text{word}_2, \text{word}_3), 1)\\ \text{Intermediate phase: } ((\text{word}_1, \text{word}_2, \text{word}_3), [1, 1, ...])\\ \text{Reduce: } ((\text{word}_1, \text{word}_2, \text{word}_3), [1, 1, ...] \rightarrow ((\text{word}_1, \text{word}_2, \text{word}_3), m)\end{gathered}$$

The resulting most frequent triplets of the dataset are:

(('analysis', 'clinical', 'study'), 102) $\rightarrow$
the itemset of words indexed *analysis*, *clinical* and *study* recurs 102 times

(('patient', 'cell', 'disease'), 178) $\rightarrow$
the itemset of words indexed *patient*, *cell* and *disease* recurs 102 times

Such operations can be generalized in the *a priori algorithm*: after finding the item candidates, it iteratively performs the MapReduce operations, that each time are fed with combinations of the results of the previous step and an additional item $\{j\}$. The algorithm can be generalized as follows:

$$\begin{gathered}\text{Map: Combination candidate} \rightarrow ((I \cup \{j\}), 1)\\ \text{Intermediate phase: } ((I \cup \{j\}), [1, 1, ...])\\ \text{Reduce: } ((I \cup \{j\}), [1, 1, ...] \rightarrow ((I \cup \{j\}), m)\end{gathered}$$

I is an itemset of size k, which has resulted as frequent at the k-th iteration. In the following iteration, such baskets are combinated with an additional item $\{j\}$, to form candidates of size *(k+1)*. Frequent baskets of size (k+1), that will be fed as candidates in the following step. This process stops as soon as no more frequent itemset is identified, based on the given support threshold (s).

At each iteration of the algorithm, combinations are filtered, so that only those having support higher than the minimum support. The iterative process stops when no the algorithms cannot find any more candidates. The final output is a rdd with frequent itemset with at most k items to 4.

Finally, frequent itemsets, as found in the a priori algirthm, are computed for generating association rules. After creating a dictionary, containing the most frequent itemset and their counts, we iteratively combined those itemset for creating baskets of different sizes. Such combinations are formed by two components: the antecedent, the original itemset, and the consequent, the item or items added. Association rules are considered relevant if they have confidence higher than 50%.

Association rules are evaluated based on their confidence and their interest. Here are a few example of the association rules associated to the highest level of confidence in the sample considered.

Antecedent	Consequent	Support	Confidence	Interest
(human, protein, response)	(cell)	0.003920	0.838235	0.608096
(increased, present, using)	(study)	0.003301	0.738462	0.431288
(present, significantly, treatment)	(study)	0.002820	0.694915	0.387741

Table 1: Support, Confidence, and Interest for Association Rules

For example, the *(human, protein, response, cell)* itemset occurs in the 0.39% of the whole texts considered. The same itemset has confidence 83.82%, which means that in the 83.82% of the texts where the antecedent *(human, protein, response* is present, there is also the consequent *(cell).* Such a high confidence may be due to the fact that the word *cell* occurs many times in the whole dataset, for this reason we examine the interest. The interest is equal to 66.81%, which is quite high, which lets us understand that the antecedent actually encourages the presence of the consequent.

From what we can see from our results, we understand that itemset having as consequent the word *(study)* have lower itnerests, with respect to higher level of confidence. So we can conclude that association rules containing the word *(study)* as the right side of the former, are considered as so mainly due to the high number of occurrences of the word *study.*

Such analysis can be performed looking at the final results we obtained, that are stored in the .csv file named ”rules.csv”.

5 Conclusions

We have been able to condut our analysis on the MeDal dataset, thanks to the implementation of the PySpark setup.

Once all the preprocessing operations have been completed, we have implemented the Market Basket Analysis in two ways. The first allow us to analyse in detail how singletons, pairs and triplets have been computing. In the latter, we implemented the A-Priori algorithm to get as the final output the association rules, pushing the maximum size of itemset to 4, in order to see also how increasing the size of the basket affects the results of association rules.

Of course, support and confidence displayed are above the threshold we set and decrease as the number of items considered in a basket, due to the monotonicity property. Interest is actually the measure we are interested to look at the quality of the association rules we found, and thanks to this metric we are able to discriminate between the rules that have high confidence only because a particuar word is repeated many times among the whole dataset, and the rules that are actually interesting, where the antecedent actually increases the probability of seeing the cosneqeunt in concurrence.

The aim of the projct has been to develop the A Priori alorithm for *Market BasketAnalysis*, and to make the algorithm scalable for larger dataset, such as the MeDal dataset in its entirety.