Algorithms for Massive Datasets

Market Basket Analysis

March 2022

Abstract. The project aims at enforcing a Market Basket Analysis (MBA) of old newspapers using PySpark by finding frequent itemsets. Two different algorithms were implemented to perform this task: A-Priori Algorithm and the first complete pass of Toivonen algorithm. In the first section of the report we describe the dataset, then we explain the pre-processing phase we performed to clean the records and tokenize texts. In the last part we explain the implemented algorithms and show the results.
Keywords: Newspapers, Natural Language Processing, documents, A-Priori algorithm, Toivonen algorithm.

Paola Serra ID 960547 Giulia Hadjiandrea ID 941780
Università degli Studi di Milano
Data Science and Economics

Contents

1 Research Definition **2**
1.1 Dataset 2
1.2 Data cleaning and Preprocessing 3

2 Algorithms and Implementation **5**
2.1 A-Priori Algorithm 5
2.2 Toivonen Algorithm 8

3 Concluding remarks **8**

List of Figures

1 Code piece for Kaggle API 2
2 Dataset used for the analysis 3
3 Histogram of bag-of-words lenghts 4
4 Barplot representing the number of different itemsets by threshold . . . 6
5 Wordcloud of the most frequent singletons for a threshold of 0.0075 . . . 7
6 Algorithm run time plot 7

1 Research Definition

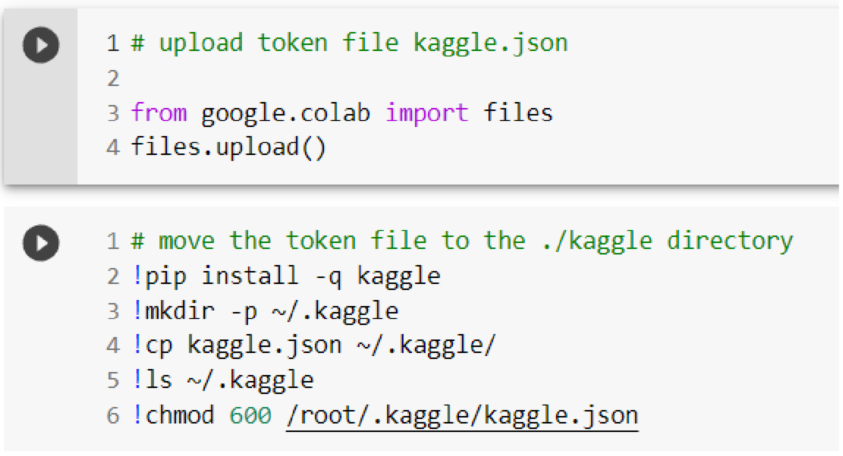
The aim of the project is to implement MBA-Market Basket Analysis on old newspapers by implementing scalable solutions for finding frequent itemsets. In a market basket analysis, one looks to see if there are combinations of products that frequently co-occur in baskets. The Old Newspaper Dataset has been analyzed in order to find words, considered as 'items' that occur more frequently in the 'baskets' of documents. To implement the task A-Priori algorithm and Toivonen algorithm were implemented after pre-processing and tokenizing the text.

1.1 Dataset

Data were taken from a Kaggle dataset called Old Newspaper, available for free as zip file [1]. The dataset is divided into four columns:

- Langauge: Language of the text.
- Source: Newspaper from which the text is from.
- Date: Date of the article that contains the text.
- Text: Sentence/paragraph from the newspaper

The file of size 5.61 GB was downloaded and unzipped thanks to the Kaggle's API.



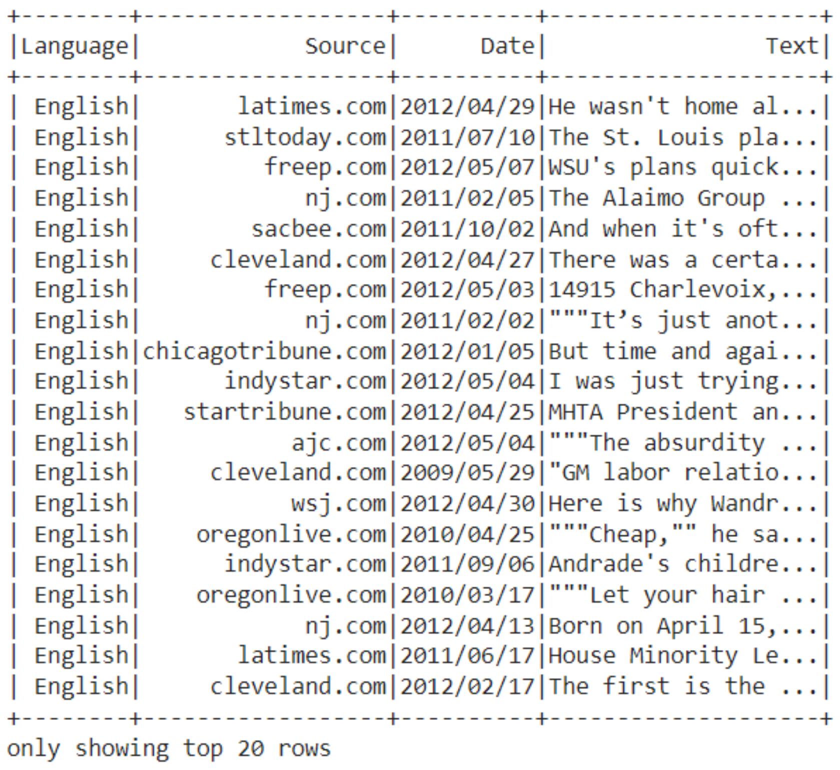
```
# upload token file kaggle.json

from google.colab import files
files.upload()
```

```
# move the token file to the ./kaggle directory
!pip install -q kaggle
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!ls ~/.kaggle
!chmod 600 /root/.kaggle/kaggle.json
```

Figure 1: Code piece for Kaggle API

For the implementation, a Jupyter notebook executable on Google Colab was used. Moreover, since the corpus contains 16,806,041 sentences/paragraphs in 67 languages, we only used the ones that contain English as language.



```
+--------+------------------+----------+--------------------+
|Language|            Source|      Date|                Text|
+--------+------------------+----------+--------------------+
| English|       latimes.com|2012/04/29|He wasn't home al...|
| English|      stltoday.com|2011/07/10|The St. Louis pla...|
| English|         freep.com|2012/05/07|WSU's plans quick...|
| English|            nj.com|2011/02/05|The Alaimo Group ...|
| English|        sacbee.com|2011/10/02|And when it's oft...|
| English|      cleveland.com|2012/04/27|There was a certa...|
| English|         freep.com|2012/05/03|14915 Charlevoix,...|
| English|            nj.com|2011/02/02|"""It’s just anot...|
| English|chicagotribune.com|2012/01/05|But time and agai...|
| English|       indystar.com|2012/05/04|I was just trying...|
| English|    startribune.com|2012/04/25|MHTA President an...|
| English|            ajc.com|2012/05/04|"""The absurdity ...|
| English|      cleveland.com|2009/05/29|"GM labor relatio...|
| English|            wsj.com|2012/04/30|Here is why Wandr...|
| English|     oregonlive.com|2010/04/25|"""Cheap,"" he sa...|
| English|       indystar.com|2011/09/06|Andrade's childre...|
| English|     oregonlive.com|2010/03/17|"""Let your hair ...|
| English|            nj.com|2012/04/13|Born on April 15,...|
| English|       latimes.com|2011/06/17|House Minority Le...|
| English|      cleveland.com|2012/02/17|The first is the ...|
+--------+------------------+----------+--------------------+
only showing top 20 rows
```

Figure 2: Dataset used for the analysis

1.2 Data cleaning and Preprocessing

Once the file containing data has been downloaded and unzipped, we used PySpark to represent the dataset as a RDD(Resilient Distributed Database), a collection of elements that can be divided across multiple nodes in a cluster to run parallel processing. The RDD was partitioned into two chunks, since typically 2-4 partitions for each CPU in the cluster are used; spark will run one task for each partition of the cluster[2]. Furthermore, data were divided into rows to get a list containing the language of the article, the year and the content. By counting the number of documents for each language we found out that documents with English as language are 1010242 and decided to keep only 10 percent of it due to the enormous computational cost in the generation of frequent itemsets. Moreover, we counted the number of documents for each year finding out that 309918 documents have an unknown date and that 6 documents have 1970 as date. Therefore, by investigating more and returning only documents with unknown date and date equal to 1970 we found out that the first are all written in Spanish while the second are all written in French. Also, some records were empty and we decided to filter them out. The kind of data one gets from the web is usually unstructured. It contains, for example, unusual text and symbols that need to be cleaned up so that a machine learning model can grasp their meaning, but also typos and abbreviations. The reliability of the model is highly dependent upon the quality of the data. Also, as Figure 3 shows, there is a different length for each bag of word produced by tokenization meaning that the number of frequent itemsets could be different for each document.

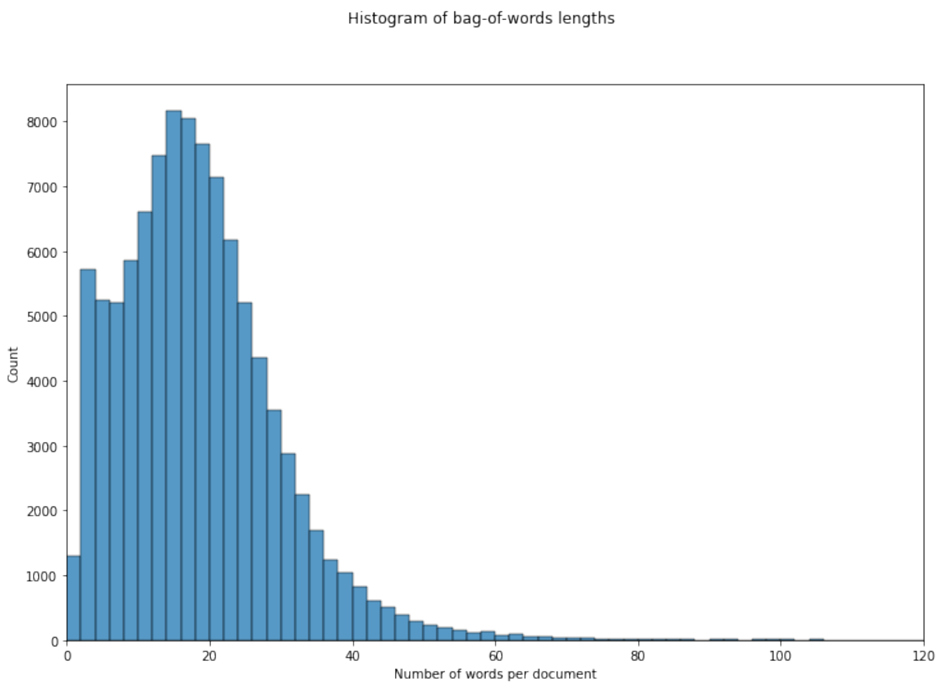


Figure 3: Histogram of bag-of-words lenghts

It may be helpful to get rid of unhelpful parts of the data, or noise, by converting all characters to lowercase, removing stop words and punctuation marks. In particular, stopwords were removed allowing the model to consider only key features because these words typically don't contain much information: there are many variations of those words that do not bring any new information and create redundancy, ultimately bringing ambiguity when training machine learning models for predictions. Finally, text was lemmatized and tokenized, separating it into smaller units called tokens. After the pre-processing phase, data have the following format:

Text = "Gattone was charged with possession of a controlled dangerous substance and possession of paraphernalia"

List = ['charged','controlled','dangerous','gattone','paraphernalia', 'possession','substance']

2 Algorithms and Implementation

Market Basket Analysis has been implemented by applying two algorithms: A-Priori algorithm and Toivonen algorithm.

2.1 A-Priori Algorithm

A-Priori was first implemented by R.Agarwal and R.Srinkat. The algorithm assumes that all subsets of a frequent itemset must be frequent(A-Priori property). If an itemset is infrequent, all its supersets will be infrequent. The algorithm is called A-priori because it uses prior knowledge of frequent itemset properties. The steps followed in the A-Priori Algorithm of data mining are:

- Join Step: This step generates (K+1) itemset from K-itemsets by joining each item with itself. More precisely, the algorithm finds frequencies by considering how many times the items occur in the dataset; frequencies are obtained for every single item.
- Prune Step: This step scans the count of each item in the database. Candidates consist of frequent items; to find occurrences of candidate itemsets in each basket we produced a list of the candidates that appear in the basket. If the candidate item does not meet minimum support, then it is regarded as infrequent and thus it is removed. This step is performed to reduce the size of the candidate itemsets.

The first threshold we considered is 0.006. This means that the algorithm had to look for singletons that appear in the basket at least 606 times. The algorithm found out that there are frequent singletons and, after computing the same for pairs, there are frequent itemsets. Moreover, the same process was applied to triples without any significant result. The same was done increasing to a threshold of 0.0075 with the algorithm finding 381 frequent singletons displayed in Figure 4 and 55 frequent itemsets. The last threshold to be set is a threshold of 0.01 with the algorithm finding 244 frequent singletons, displayed in Figure 3, and 22 frequent itemsets.

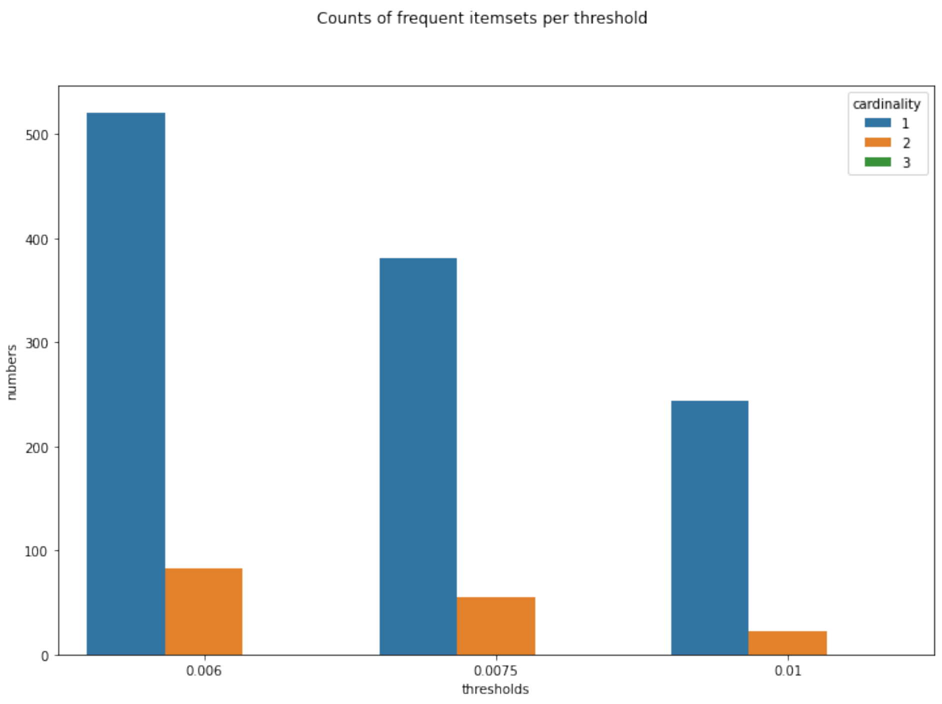


Figure 4: Barplot representing the number of different itemsets by threshold

Results show, as we expected, that the more the increase of the threshold, the smaller the size of frequent itemsets.In fact if a threshold is increased the candidates that are excluded and considered not frequent. Figure 4 summarizes results for singletons and pairs for each threshold. It is visible how, following an increase of the threshold, there is a decrease in the blue column representing singletons and orange column representing pairs. Figure 5 shows an example of a Wordcloud of the most used singletons.

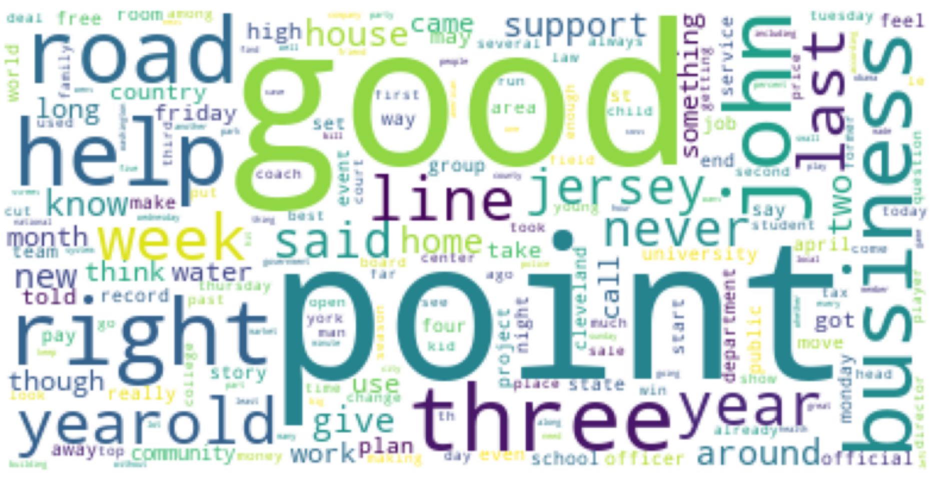


Figure 5: Wordcloud of the most frequent singletons for a threshold of 0.0075

The second part of our application of the A-Priori algorithm concerned the scalability of the algorithm. Figure 6 below represents the time the algorithm needed to perform the task on the dataset for a size of the RDD of 5000, 10000, 25000, 50000, 100000. By looking at the graph we can conclude that the algorithm does not scale well since we do not have decreasing return to scale.

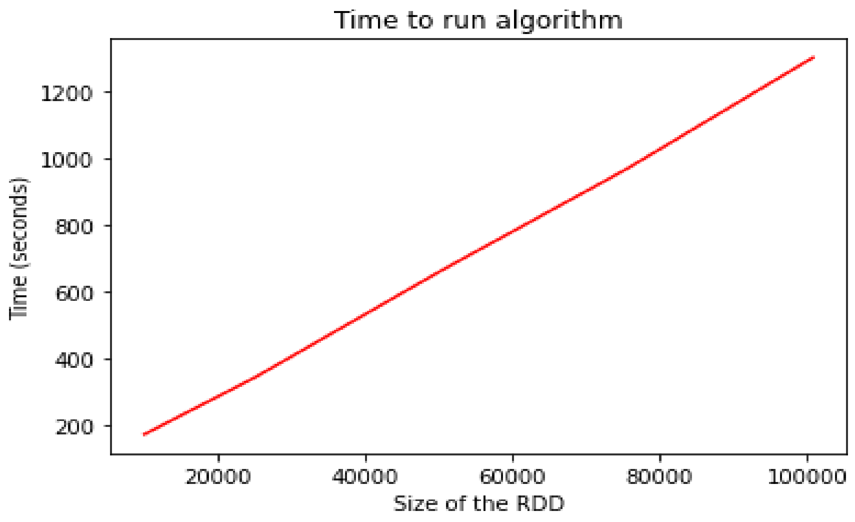


Figure 6: Algorithm run time plot

2.2 Toivonen Algorithm

The second algorithm we applied is the Toivonen algorithm. The Toivonen algorithm is one of the algorithms that find all frequent itemsets using at most 2 passes over data. In fact, in the first pass the algorithm performs on a small sampled data and then on all the data in the second pass. In our analysis we counted the negative border for the second pass; also, we stopped our analysis by doing only one complete pass on the negative border for each threshold percentage due to the time cost of loops. After finding candidates for frequent itemsets for the sample, we constructed the negative border; a negative border is a collection of itemsets that are not frequent in the sample but all of their immediate subsets are frequent. We started by setting the threshold to 0.006 like for the Apriori algorithm and the percentage of the sample to 0.1. We noticed that by changing the sample percentage to 5 percent and 15 percent the result does not change; we would have expected an increase in the frequent itemsets and a decrease of the list of elements in the negative border that are actually frequent. After counting the occurrences of the negative border in the whole dataset results show that all elements in the negative border are actually not frequent. The same happens by setting the threshold to 0.0075 and 0.03. Instead, results for a threshold equal to 0.01, show that there are two false negative; this means that the pairs are frequent even if they appear in the negative border. At this point the algorithm should be re-runned by changing the sample.

3 Concluding remarks

All the codes of the project are available on Github where every notebook reports the results of the methods we have used. What we wanted to identify in this project were the frequent itemsets to enforce Market Basket Analysis by using two different algorithms. The first one, A-Priori, found singletos and pairs but could not find frequent triples. Moreover, as we would expect, an increase in the threshold caused a decrease in the size of frequent itemsets. It is worth pointing that we did not further decrease the size of the threshold since it would have been unfeasible with Google Colab and since the frequent itemsets would end up being meaningful for the analysis. The second one, Toivonen algorithm, seemed to work well with threshold of 0.006, 0.0075 and 0.03. Instead, for a threshold on 0.01, it only found two frequent pairs that were in the negative border. We would have expected that with increase in the threshold there would be less false negative; this did not happen when we increased the threshold from 0.0075 to 0.01. To conclude, we are overall satisfied on the results since it seems that the A-Priori performance worked.

References

[1] "https://www.kaggle.com/datasets/alvations/old-newspapers"

[2] "RDD Programming Guide - Spark 2.2.1 Documentation"