Market - Basket Analysis

Algorithms for massive datasets project - Irene Aggiudicato (960261) & Davide Riva (944176)

Abstract

The aim of this short paper is to find the frequent itemsets in a dataset containing old newspaper pieces using PySpark. The dataset contains records in several languages, we have decided to take into consideration the English and Italian ones. The scope of the project is to perform a market basket analysis (MBA) using lemmatized words as items and documents as baskets.

To perform this task we apply A-Priori algorithm and subsequently measure its performance on sampled data (first 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.

Data

The data we have taken into consideration are contained in a dataset that is freely available on Kaggle. (Old Newspapers | Kaggle)

The dataset is organized in four columns:

- Language: in which language the document is written. In particular, we can have 67 different languages.
- Source: from which newspaper the document is taken.
- Date: date in which the article was published.
- Text: the document itself.

The dataset is quite large, in fact it occupies 5.61 GB of memory. For this reason, we decided not to download it locally, instead we used Kaggle API to download it during code execution. As for the implementation itself, we used a Jupyter notebook executable on Google Colab.

Because of the amount of data, we decided to focus just on some records. In particular we have analysed just those that contains English or Italian articles. Of course, we can use the same algorithms and techniques for other kinds of records, since the approaches will hold for whatever language.

Analysis

In order to ensure applicability to large datasets, we used PySpark to manipulate and analyze our records. First, we initialized a SparkContext object, so to treat the data as a PySpark Resilient Distributed Dataset (RDD). This lets us have a gateway to Apache Spark functionalities through the usage of a Jupyter notebook in a Python environment.

Once we have downloaded and unzipped the tsv file containing the dataset through Kaggle API, we transformed it in an RDD partitioned in 2 chunks. In principle, there is no correct number of partitions to be set, however we can say that it usually depends on how many cores the machine on which we are performing our analysis has and it can't be either too low with respect to the number of cores (which would cause inefficiency in the allocation of work especially if partitions are skewed) nor too high (since loading chunks is time-expensive). For this reasons, having acknowledged that Google Colab provides just 2 cores, we decided, on a purely empirical basis, to use this value as number of partitions.

Originally, the records were stored as strings that contained all the features: language, source, date and document. This is why the first action we did was to split each row so to obtain lists containing the language of the articles, the year and the content. Once having done this, we selected documents in English and Italian language.
For sake of knowledge, we decided to filter the documents according to the years in which they had been published. We have seen that the major part of them had been written between 2006 and 2013. However, a discrete amount of them, more than 300.000 articles, had no date, whereas 6 dated back to 1970. Moreover, the heading row had year equal to "date".

Looking directly on those written in 1970, we found out that they are all written in French, a language that we have decided not to take into consideration. As for the ones having unknown date, they were all in Spanish.

We also found out that some records were empty so we filtered them out, reducing the overall number of records by 46 units.

English articles

Let's begin with English documents. These ones amounted to more than 1 million, which would take a very long time to process, so we decided to keep only a 10% sample of the original data in order to test our code. We will see later how it scales with respect to the size of the dataset.

Before entering the details of market-basket analysis, we have to talk about all the pre-processing that enables us to clean the data.

Pre-processing phase

The pre-processing phase was about cleaning the documents from all the stopwords and the punctuation, plus some other noisy items that we considered useless for the aim of the project. This can be done through the use of the libraries *nltk*, *re* and *string*.

We also tokenized the texts and decided to lemmatize words, since our analysis does not care about different declinations of them.

At this point of the analysis our baskets were more than 100000. Below we report an histogram representing how long are the bag-of-words that had been produced by the tokenization. We thought that this could be interesting for market basket analysis, as longer documents may give rise to an higher number of frequent itemsets.

As we can see, the mode of this graph appears to be equals to 25 more or less and almost all articles are shorter than 50 words.

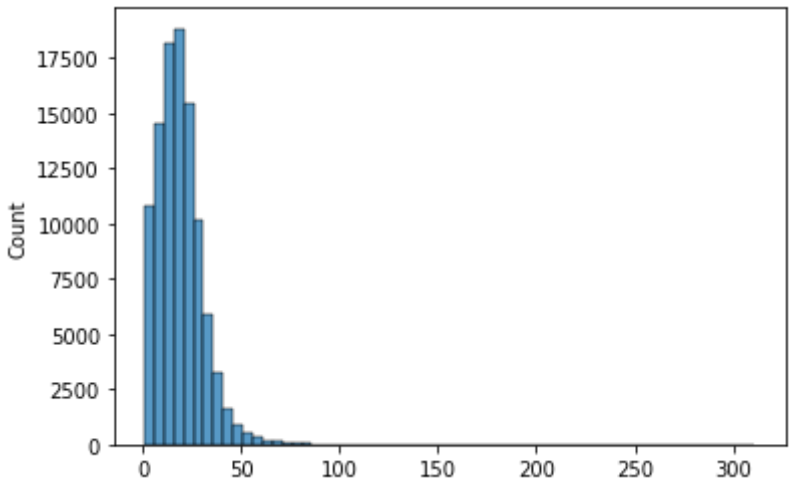

Figure 1: histogram of the length of the articles

In the end, data have the following form:

```
Sentence = "It isn't a question of whether the school wants the former Florida
and Utah coach. It's a question of whether he wants the school. Why would he step
into this cesspool when he can have his pick of jobs?"`

`List = ['cesspool',   'coach',   'florida',   'former',   'isnt',   'job',
'pick',   'question',   'school',   'step',   'utah',   'want',   'whether',
'would']
```

Starting from these lists of words, namely *bag-of-words*, we want to discover which are the singletons and the combinations that appear at least in a certain number of baskets (documents). Clearly, we have to define a threshold, which is a percentage of the overall number of baskets which puts the boundary between frequent and infrequent itemsets. If a word or a combination of words appears in the text a number of times greater than or equal to our threshold multiplied by the dataset size, it will be counted as frequent.

A-Priori Algorithm

This type of algorithm is designed to reduce the number of candidates that must be counted, at the expense of performing two or more passes over data, rather than one pass. At the first pass, each word (singleton) is a candidate, whereas for pass K > 1 candidate itemsets are sets of K items whose immediate subsets, of cardinality K-1, are all frequent.

The mechanism, for pass K, is the following:

- take the list of frequent itemsets of cardinality K-1;
- generate all candidate itemsets of cardinality K whose immediate subsets are in the list;
- for each basket, add one to the counter of candidate *c* if each element of *c* is in the basket;
- finally, examine the counts to determine which itemsets are frequent.

If no frequent itemsets of a certain size are found, then we can conclude that there can't be larger frequent itemsets, and the algorithm ends.

To find occurrences of candidate itemsets in each basket, we measured the performances of three different methodologies:

- Plan A: using sets and built-in function `issubset` to produce a list of the candidates that appear in the basket;
- Plan B: using function `searchsorted` of library `numpy` to verify the position in which items would be put in alphabetically-ordered basket, so that if an item is not present in its expected position we are sure the whole itemset is not present in the basket;
- Plan C: collecting the combinations of words in each basket using function `combinations` from library `itertools`.

Results show plan A is much faster than the other two. To be more precise, we have found out that Plan A generally takes 2 seconds to process 100 baskets, Plan B appears to be the slowest, taking on average 48 seconds for running, whereas Plan C takes more or less the half. This is the reason why, from now on, we have decided to use the first methodology.

We started with a threshold equal to 0.01. This means that we are asking to the algorithm to find all the singletons which are present in the documents at least 1010 times. We have found out that we have 251 frequent singletons. We performed the same computation for pairs and found 20 of them. As triples were absent from the results even with smaller values of the threshold (up to 0.0075) we have decided to focus just on singletons and pairs.

Changing the value of the threshold to 0.02, we found 68 frequent singletons, displayed in figure 2, whereas just two are the pairs that can be considered as frequent.

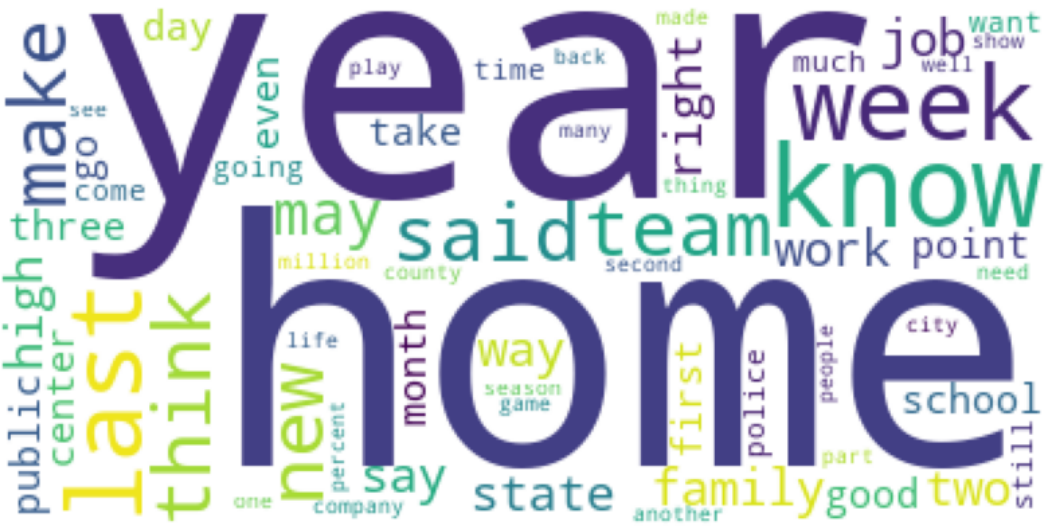

Figure 2: Wordcloud of the most used singletons

The last threshold to be set was 0.03. In this case the pairs completely disappear and the singletons are just 27.
This was coherent with what we expected: as the threshold increases, frequent itemsets become fewer. This is because, having an higher threshold means having less candidates as the words should practically appeared a lot in the baskets. The barplot below summarizes our results, displaying the number of different frequent itemsets (singletons in blue and pairs in orange) according to different thresholds.

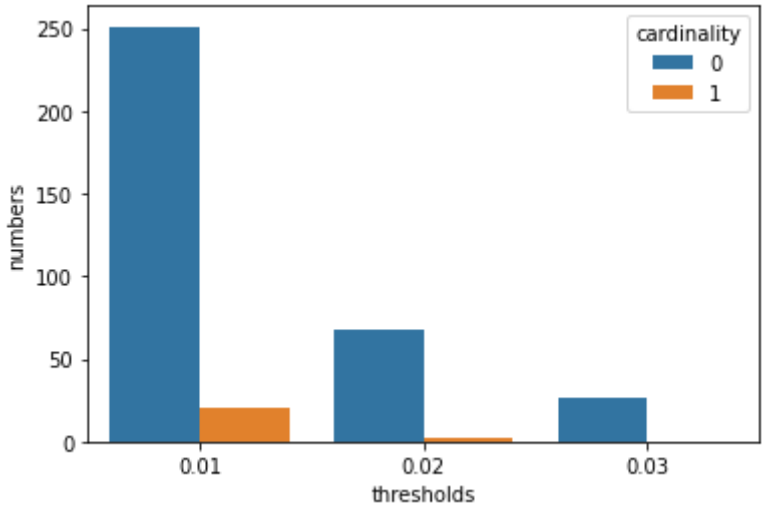

Figure 3: barplot representing the number of different itemsets found out according to the different values of the thresholds

Scalability of Apriori Algorithm

Once having done this, we wanted to prove the scalability of the algorithm. For this reason, we decided to set the threshold at 0.01 while changing the size of the dataset by sampling it.

We tried five different fractions (0.01, 0.05, 0.1, 0.5, 1) and we present below the plot representing the time spent to perform the task on the sampled datasets. Needless to say, having a larger dataset always results in a slower work. In fact, in our case, with a size close to 1000 records we get 150 seconds, whereas with 100000 records we get more than 2000 seconds.

We wished a logarithmic, or at least concave, function, so a situation in which we have decreasing "returns to scale". However, the algorithm does not scale as good as we would expect.

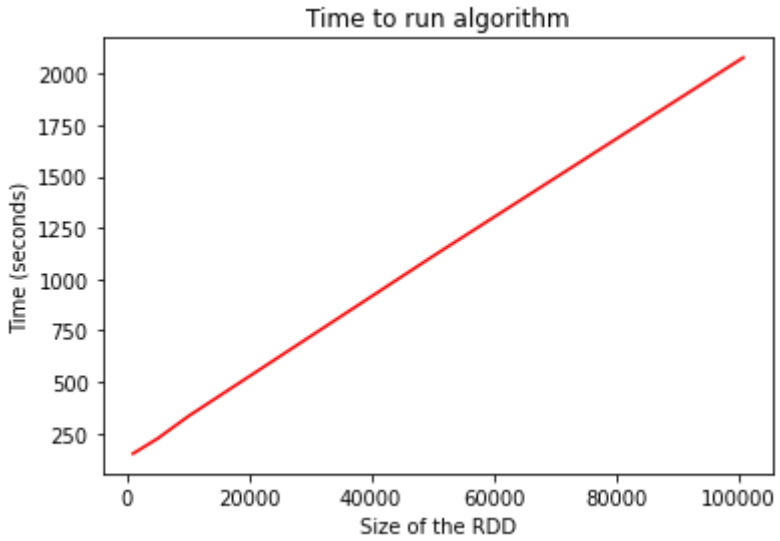

Figure 4: plot of the time to run the algorithm according to the size of the dataset.

Toivonen's algorithm

The next step was the implementation of Toivonen algorithm. The approach used by this algorithm is a little bit different from the previous one. In fact, it performs one pass over a small sample and one pass over all data.

In principle, the algorithm is designed to return neither false positives nor false negatives by looping on the dataset, each time resampling from it, but in our case, in order to avoid long loops, we only measured the performance of the A-Priori algorithm on a sample of data, i.e. the first pass of Toivonen algorithm.
In particular, we built and measured the negative border: the collection of itemsets that are not frequent in the sample, but whose immediate subsets are all frequent in the sample.
In general, when no member of the negative border is frequent in the whole, then there can be no itemset whatsoever that is frequent in the whole but neither in the negative border nor the collection of frequent itemsets for the sample. So we can take the cardinality of the negative border as a measure of the inaccuracy of running the underlying algorithm on a sample rather than the whole dataset.

We expect that the larger the sample is, the shorter the list of negative border elements that are frequent in the overall dataset will be. In what follows, the underlying A-Priori algorithm is performed with the three different values of threshold that we have already used before: 0.01, 0.02 and 0.03. Since sampling produces small, downloadable datasets, we neglected RDDs for this part, considering the dataset as a list of lists and applying `apriori` function from library `apyori`.
Unexpectedly, the false negative found by the negative border is always the same even with different percentage of sampling of the dataset. This means that having a threshold of 0.01 will cause us to find always two elements as false negative indistinctively as we are sampling the dataset as 5%, 10% nor 15%. The same happens when we are dealing with a threshold of 0.02. In this case the false negative that we've got when we are taking into consideration three different portions of dataset is always 1. Whereas, when we use 0.03 as threshold none itemsets are found as frequent in the dataset but not founded by the apriori algorithm.
In general, we can say that the apriori algorithm works well with the dataset, leaving aside just few, if none, frequent itemsets. Moreover, even if we are not able to completely justify the assumptions for which having larger percentage of portion of dataset would have caused a

shorter list of the itemsets in the negative border, we can't neither reject it, as the outcomes simply do not change.

Let's now focus on Italian articles and see how the approaches that we have already used for English articles are effectively and useful.

Italian articles

Preprocessing phase

Having analysed all the articles in English, we then took into consideration also the Italian records, applying the same techniques.
The 10% sample of the Italian dataset contains 16782 baskets. Once again, we decided to check lengths of single records, in order to get an idea of forthcoming results. As we can see in figure 5, we have a great amount of documents containing a single word. Then we have more or less 3000 articles with a length around 50 words and a few longer than 100. With this statistics, we can already draw a comparison between English and Italian articles: even if the former are more numerous, it appears that fewer of them are long, which may result in a larger number of frequent itemsets for Italian articles holding the same threshold values.

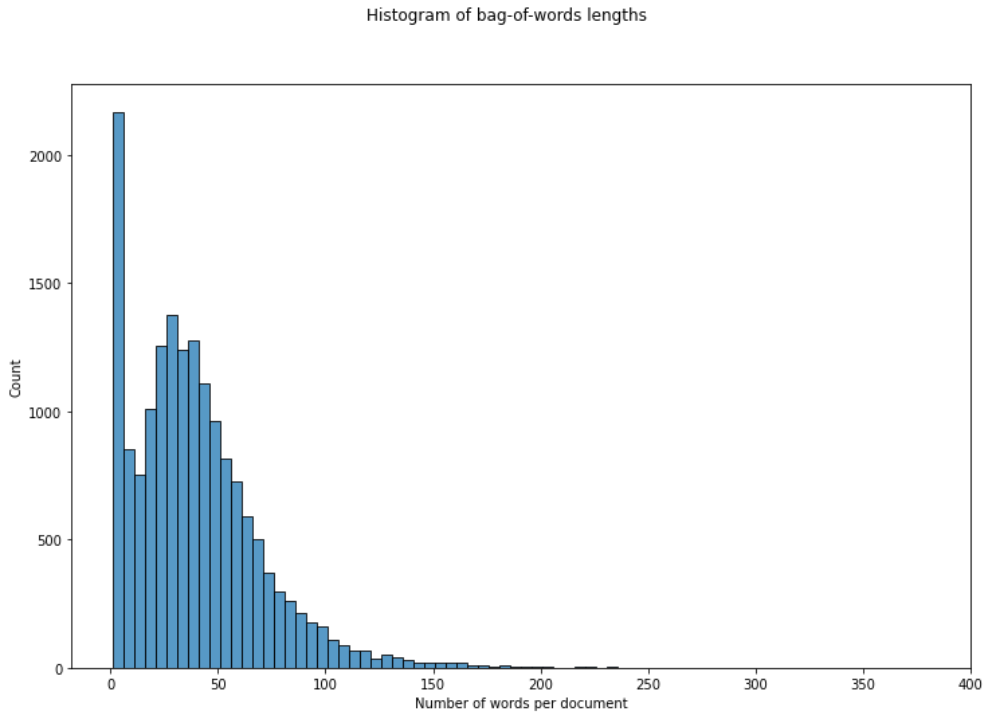

Figure 5: Length of Italian articles

The result that comes out from this first phase of process is the following:

```
Sentence = "Si avvicinano il Kangourou e le Olimpiadi della matematica,
competizioni individuali e a squadre che hanno visto coinvolti con grande
entusiasmo, conquistandosi anche posizioni di tutto rilievo all'interno della
classifica, un buon numero di ragazzi bresciani."

List =["all'interno",  'avvicinano',  'bresciani',  'buon',  'classifica',
 'coinvolti',  'competizioni',  'conquistandosi',  'entusiasmo',  'grande',
 'individuali',  'kangourou',  'matematica',  'numero',  'olimpiadi',
 'posizioni',  'ragazzi',  'rilievo',  'squadre',  'visto']
```

Apriori Algorithm

After cleaning the dataset, we performed the analysis. Once again, we started with the A-Priori algorithm with a threshold equal to 0.01, which in this case means 168 occurrences.
The outcome provides us 790 frequent singletons and 611 frequent pairs. Also in this case, slightly lower values for the threshold weren't enough to find frequent triplets, and much smaller values resulted in very slow running. For this reason we have taken into consideration just singletons and pairs.

For a threshold equal to 0.02 we found 169 frequent singletons, displayed in the following word cloud (figure 6) and 33 frequent pairs.

Figure 6: WordCloud representing the most frequent singletons in Italian language

With a threshold value of 0.03 the amount of frequent singletons and pairs decreases further: respectively 88 and 10.

As said at the beginning, even if we have fewer articles for Italian language, we have more frequent itemsets with respect to English records. Probably the reason is that, as the histogram of figure 1 and figure 5 show, Italian baskets are longer and thus the bag of words are wider and the probability of containing frequent itemset increases.
Below we report the barplot that we have created with the exact number of singletons and pairs found out by the Apriori algorithm as the threshold value changes.

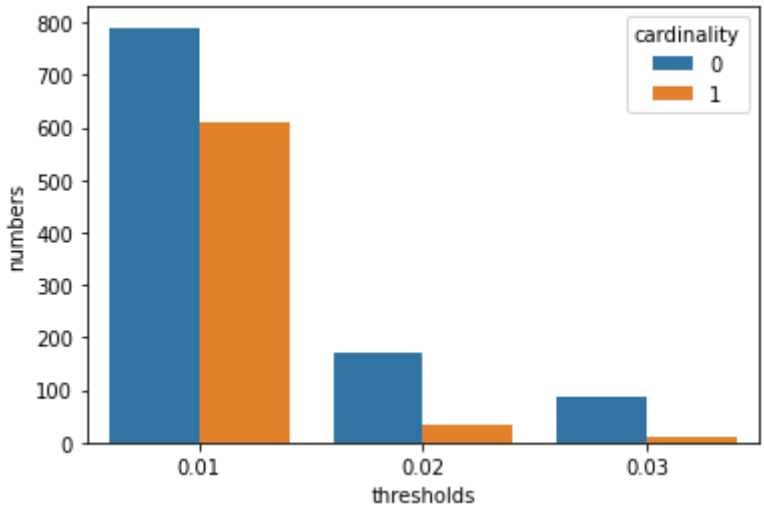

Figure 6: barplot representing the number of itemsets (singletons in blue and pairs in orange) according to the value of the threshold

Toivonen's algorithm

For what concerns Toivonen algorithm the approach we used is the same as for the English articles.

We run A-Priori algorithm with a threshold of 0.01, changing just the percentage of sampling of the dataset. Sampling 5% of the data, we found 117 false negatives. Sampling 10% we found 71, finally with a sampling equals to 15% we found 57 false negative.
We have tried to change the threshold, checking for itemsets in the negative border with a threshold equals to 0.02. In this case we had 18 false negative with a sampling of 5%, 11 if we take the 10% of the dataset and 7 with a sampling of 15%.
In the end we have changed again the threshold for algorithm and we have decided to analyse what happens with 0.03. In this case we had that with a sampling of 5% we had 10 false negative and 7 false negative when we were considering a sampling of both the 10% and 15%.
We want to recall what happened with this type of analyses when we were dealing with English articles. In that case number of false negative that we could find according to the different values of the thresholds used, were always the same even when we took different percentage of sampling, a conclusion that was not particularly in accordance to what we would expect.
With Italian articles, however, we can confirm our thought about the behaviour of the negative border. In fact in this case, for whatever threshold we take into consideration, we have a decreasing number of false negative as the sampling that we take increases. Moreover, as the threshold is smaller, we have a larger number of itemsets considered as false negative. This is because, of course, when we are dealing with a small threshold, let's take for example 0.01, we have more candidates as frequent itemsets with respect to those that we find when we are using an higher value of the threshold and so the probability of having false negative increases.

Conclusions

The aim of this report was to perform a market-basket analysis using Apriori algorithm and the first passage of Toivonen's algorithm.
We took into consideraiton a dataset composed by old newspaper taken from Kaggle and executed through Kaggle APIs in a jupyter notebook on Google Colab. The records were composed by the language in which the articles were written, the sources on which the articles have been taken, date of publication and the content. We have decided to take into consideration those articles written in English and in Italian. Even if the number of documents of the former was larger, the latter appears to have longer articles. In fact, the major part of them had a length of 50 words in contrast with the average length of 25 for English ones.

After a work of pre-processing in which we have clean our dataset we have performed the two algorithms.

For the apriori algorithm we have used three different value of threshold for seeing the different behaviour of the algorithm. We have noticed that once we have increased the threshold the candidates words of frequent singletons and pairs were fewer. From previous empirical analyses, we have seen that even using a very small value of the threshold we couldn't find frequent triplets. This is why we have decided just to take into consideration singletons and pairs. This outcome appears to be true for both English and Italian articles, even if the latter, due to the length of the documents themselves, had an overall higher values of frequent itemsets.

We have then worked on the scalability of the algorithm and we have found out that the time the algorithms takes for performing the analysis is smaller when we are working with a smaller size of the dataset, maintaining a linear increasing of the time for running. In this case, we would expect a logarithm or at least a concave function representing the scalability, however this was not the case.

We have then worked on the Toivonen's algorithm. We have built a negative border, as to say, a list of frequent itemsets which are effectively frequent in the original dataset but are not found in a sample of the dataset through the Apriori algorithm. This type of approach can be used to see the accuracy of the apriori algorithm itself.
For what concerns the English articles, we have seen that even if we used different values of sampling the dataset, we have found the same values of the negative border's length according to the different threshold. However, we were satisfied about the results, in fact just few (or none) itemsets were left behind by the apriori algorithm.
Whereas, for Italian articles we had different results. In this case, the length of the negative border actually changed as the percentage of the portion of dataset taken into consideration change as well. We had an higher number of frequent itemsets as false negative as the percentage of dataset that we take into consideration decreases.

References

- Lecture notes
- A. Rajaraman, J. Ullman "Mining of massive datasets"

Declaration

“We declare that this material, which We now submit for assessment, is entirely our 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 our work. We understand that plagiarism, collusion, and copying are grave and serious offences 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/us or any other person for assessment on this or any other course of study.“