Market Basket Analysis with A-priori Algorithm

Luca Rigamonti 08920A

2nd april 2025

1 Introduction

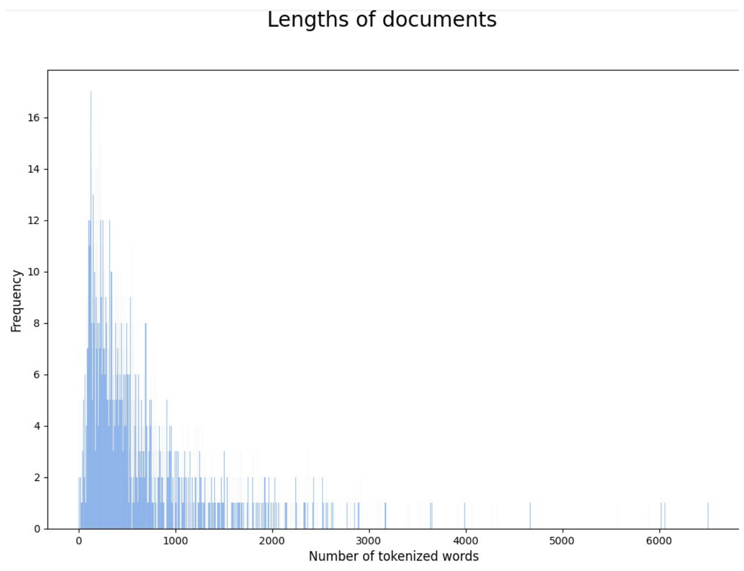
This project applies a data mining technique known as Market Basket Analysis to identify frequent itemsets within a dataset. A set of items that appears in many baskets is said to be 'frequent.' The frequency of an itemset is measured using the support count, which is the number of transactions or records in the data set containing that specific itemset.
Frequent itemset mining is a valuable resource to discover relationships among items in a dataset. It can be used to discover what items typically go together, discover customer segments, and discover other hidden patterns in the data.
The original application of the market-basket model was in the analysis of true market baskets.In this project, the technique is applied on text data (items are words and basket are sentences) in order to find frequent combinations of words in books reviews.

2 Data

In this project it has been used the Amazon Books Reviews dataset that is a large dataset (3.04 GB) that contains goodreads-books reviews and descriptions of each book.
It is composed by two folders, Books-rating and books-data but,for this project, just the first one is used. In particular, the dataset contains 10 columns; in the context of this analysis, only the column "review/text" has been considered. Each row of the dataset has been considered as a baskets and word and combinations of words as items. Therefore, the goal of the analysis is to find words and combinations of words that appear frequently in the reviews.

Since the size of data is high, only a subset of the data was retained and used for the analysis and it has been converted to RDD with the PySpark package (Resilient Distributed Dataset). Considering the fact that I am dealing witha text,I apllied a function of preprocessing sentences to reduce the complexity (in particular, words have been put in lowercase, lemmatized and tokenized and also stopwords have been removed.)
For example,

row=['A great resource for paper type crafts. From paper mache to origami this book gives you projects you can follow right from the book or inspirations for your own creations. Clear instructions and excellent step by step photos. Tried some of the projects myself with success and had fun with the kids doing it. Great for a sunny day when you want to do something different or are stuck inside on a rainy day.']
tokenized= ['rainy','success','tried', 'inside', 'creation', 'step', 'fun', 'want', 'resource', 'excellent', 'day', 'paper', 'clear', 'craft', 'great', 'follow', 'book', 'give', 'right', 'type', 'inspiration', 'something', 'origami', 'kid', 'sunny', 'stuck', 'instruction', 'project', 'different', 'mache', 'photo']



At the end, to optimize memory space, each string has been converted to an integer index.

3 Algorithm

In this project I applied the A-Priori Algorithm, that it is one of the most popular technique to mine frequent itemsets, designed to reduce the number of pairs that must be counted, at the expense of performing two passes over data, rather than one pass. A-priori works using the monotonicity property that states that if an itemset is frequent, then all of its subsets must also be frequent. So, iff no frequent itemsets of a given size are found, the monotonicity property ensures that larger itemsets cannot be frequent either, allowing us to stop the process.

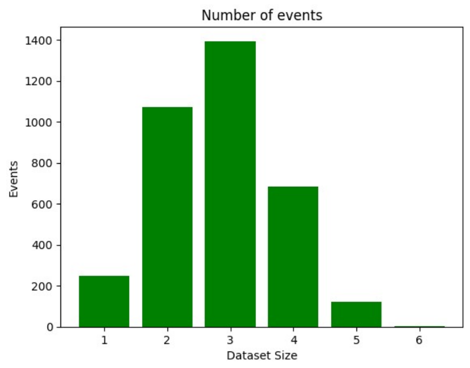
The algorithm is used in the code by calling the apriori function with an RDD and a minimum support as parameters. It returns an RDD of all frequent itemsets along with their frequency of occurrence. The minimum support value is calculated as:

$$threshold = numberofbaskets \times fractionofbaskets$$

In this project, I used 0.03 as fraction of the number of baskets. So, to be considered frequent, itemsets must exceed the threshold (89.91 in this case).

The algorithm then finds frequent singletons and combines them to form frequent pairs. It iterates to find more frequent itemsets of greater size: The previous output is stored in an RDD, and new singletons are obtained. New candidates are generated by taking the Cartesian product and sorted to remove duplicates. Candidates above the threshold are added to the output. The process continues until there are no more frequent itemsets.

The plot in the figure below illustrates the frequency of occurrences for each itemset size. From this plot, we can see that the number of frequent triples is the higher, followed by frequent pairs; then the number of frequent quadruples (and of itemsets of higher cardinality, as well as itemset of singletons) is less than the number of frequent triples.

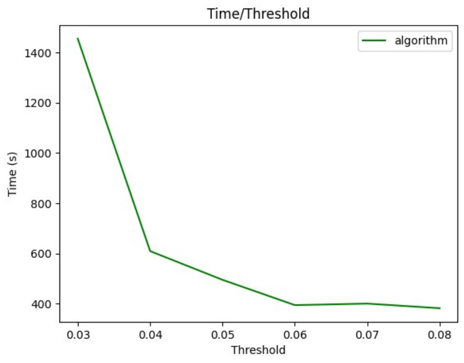


4 Scalability

Even if the analysis has been done using a portion of the dataset, the code has been written in order to be scalable. In particular, this is possible thanks to Spark's RDD data structure.
Moreover, Spark has functions for data processing that allows operations to be executed in parallel, improving performance and reducing computation time. So, Spark makes it possible to scale up the analysis for a large dataset keeping good efficiency and reliability.

To analyze the algorithm's behavior with different thresholds, I used a 15 percent sample of the RDD, in order to see the time needed to compute a-priori algorithm with threshold between 0.03 and 0.08.

As we can see, the time needed decreases with higher threshold.


Time/Threshold
algorithm
Time (s)
Threshold