Old Newspapers Frequent Itemset Mining

Algorithms for Massive Datasets

Experimental Project Report

By:

Filippo Guardassoni
972454

Srishti Gupta
964079

University of Milan

Department of Economics, Management and Quantitative Methods

Master's Degree in Data Science and Economics

Academic year 2022/2023

Abstract

Frequent Itemset Mining (FIM) is one of popular data mining technique with frequent pattern or itemset as representation of data. In other words, it is used to extract knowledge from a variety of datasets and applications. The main downsize of the process is that it is data and computing intensive. In fact, big data are the center of many realities now and growing exponentially: every day we create around 2.5 quintillion bytes. On the other hand, the chose algorithm must scan the dataset iteratively for many times to produce meaningful information. During the recent years, several efficient and scalable frequent itemset mining algorithms for big data analytics have been proposed by many researchers along with the rise of parallel and distributed computing to achieve higher efficiency. In this study, the Apriori and FP-Growth algorithms are chosen to analyze the Old Newspaper dataset composed of 16,806,041 records. After filtering and pre-processing the dataset, the algorithms are implemented in the Apache Spark framework. In particular, FP-Growth is implemented using MLlib, the Spark scalable machine learning library with tools that make practical ML scalable and easy. Apriori is not supported directly by Spark, and thus computed "manually" through various steps. In conclusion, FP-Growth reached a solution within reasonable time. On the other hand, Apriori revealed to be not efficient enough for large scale computation as a small subset was selected to be analyzed in order to prevent crashes.

Table of Contents

1. Introduction

This section presents a general and brief overview of the main concepts treated in this paper. The outline of the paper is then laid down.

1.1 Frequent Itemset Mining (FIM)

Frequent Itemset Mining (FIM) is nowadays a very important part when working with big data. It aims to extract important information through frequently occurring events such as patterns. This kind of analysis is also referred to Association Rule Mining, hinting to the research for relationships and correlations between the item sets. These items are usually stored in a transactional or relational database.

1.1.1 Market Basket Analysis

A famous application of FIM is Market Basket analysis which is often used to learn patterns for example of customer's behaviors in purchasing items at the supermarket, hence the name market basket. It gravitates around the concept of baskets (also called transactions). Each basket consists of a set of items, called itemset. It works by looking for combinations of items that occur together frequently in transactions. In the supermarket example, association rules are used to predict the likelihood of products being purchased together.

1.1.2 Definition of Support, Confidence and Lift

If an item appears in many baskets is said to be frequent. In addition, the count of the baskets in which an item is present is called *support*. With the aim of the FIM in mind, it is possible to filter the itemsets given a *support threshold* (or *minimum support)* in order to retain the most relevant information. Additionally, once the frequent itemsets are found, association rules can be derived and again filtered accordingly to parameters like confidence (likelihood) and lift (causal effect). These parameters define the interest of an association rule. All the criteria don't take an absolute minimum value, but it must be chosen accordingly to the specific dataset that is being analyzed.

1.2 Paper Structure

The project is structured as follows:

II. *Dataset*
III. *Data Pre-processing*
IV. *Model Definition*
V. *Conclusions and limitations*

VI. References

In chapter II, the dataset composition is outlined, and its characteristics are summarized. In chapter III, the fundamental steps to prepare the data for the analysis are run through. In chapter IV, the chosen algorithms are presented along with the theory behind them and their scalability. In chapter V, the conclusions of the study are exposed with the results, the inherent limitations, and the recommendations for future work. A list of the references is then presented in chapter VI.

2. Dataset

In this section, the original structure of the dataset *Old_Newspaper* is explained in detail.

2.1 Dataset Overview

The *Old_Newspaper* contains 16,806,041 records. It contains natural language text from various newspapers, social media posts and blog pages in multiple languages.

The columns of each row in the .tsv file are:

- **Language:** 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.

	Language	Source	Date	Text
0	Afrikaans	republikein.com.na	2011/09/14	Die veranderinge aan die Britsgeboude Avensis ...
1	Afrikaans	republikein.com.na	2011/01/20	Duitsland se mans- en vrouespanne is die afgel...
2	Afrikaans	sake24.com	2009/11/28	Mnr. Estienne de Klerk, uitvoerende direkteur ...
3	Afrikaans	sake24.com	2009/11/12	Mustek is se finansiële-resultate-advertensie ...
4	Afrikaans	sake24.com	2011/02/04	nadat LMS se raad van trustees in Junie verled...
5	Afrikaans	praag.co.za	2011/06/09	Hierdie hersirkulering van kaders werk net so ...
6	Afrikaans	rapport.co.za	2011/07/15	Volgens ao. November Filander, polisiewoordvoe...
7	Afrikaans	republikein.com.na	2011/05/20	Die plant is besonder gehard en kan selfs uite...
8	Afrikaans	republikein.com.na	2011/10/19	Dit volg op twee vorige ekspos deur SMEs Compe...
9	Afrikaans	praag.co.za	2009/06/01	Daarom moet 'n Afrikanerafvaardiging so gou mo...

Figure 1 Snap of the dataset

2.1.1 Cleaning and Filtering

The dataset is a subset taken from *HC Corpora* which is a collection of corpora for various language (66). It is an already cleaned version of the raw data from newspaper subset of the *HC corpus*. This study is conducted on only *English* language. The other columns are disregarded. After filtering, the resulted dataset is composed by 1.010.242 rows

	Text
0	He wasn't home alone, apparently.
1	The St. Louis plant had to close. It would die...
2	WSU's plans quickly became a hot topic on loca...
3	The Alaimo Group of Mount Holly was up for a c...
4	And when it's often difficult to predict a law...
...	...
1010237	Serve a taste of spring: Chop fresh vegetables...
1010238	The complaint alleges that Kuvan Adil Piromari...
1010239	But I'm in the mood. After six or more months ...
1010240	That starts this Sunday at Chivas. The Goats a...
1010241	The only outwardly religious adornment was a b...

1010242 rows × 1 columns

Figure 2 Snap of the reduced dataset

3. Data Pre-processing

Text is one of the unstructured data which need special treatment prior to further processes. Structured representation of a text is generally divided into two types: single word (bag of words) and multiple words. Bag of words is a structured representation form which collect all the words in the document without seeing the relationships among the words, while multiple word representation collects words in the text document by selecting the relationships among the words so that the semantic meaning of the text is maintained. In order to feed the correct data structure to the algorithms, a list of lists is created where each list represent a basket and each word an item. In order to keep the reproducibility of the results, a random seed of 1234 is initiated.

3.1 Natural Language Pre-Processing

3.1.1 Replace special characters and expand contractions

The first procedure is replacing some special characters that are evident to the eyes such as "/" and "ø". Then, using the package *contractions*, contractions (shortened words) are expanded. For example, "*It wasn't*" becomes "*It was not*".

3.1.2 Split sentences

At this moment, the list of lists contained lists of phrases. Each phrase was unpacked and split into words (items). Now, the basket-item structure is achieved.

3.1.3 Remove punctuation, blank spaces and capitalization

Being the words part of a bigger phrase, punctuation is still left inside each word if present. Using the package *ntlk* and *translate*, punctuation is removed along with blank spaces (created by removing additional symbols like "–" and the possessive case. In addition, capitalization is removed to avoid two identical words not to be recognized as such.

3.1.4 Remove digits

One of the best practices to work with natural language is to remove digits. In addition, words containing numbers like "*word1234"* are removed. On the other hand, words representing numbers like "*two"* are kept.

3.1.5 Remove stop words

Stop words are insignificant words for the processing of natural language data. Although there is no universal list of stop words and neither rules to identify them, the package *nltk* has a predefined list of the most used ones such as subject pronouns and articles.

3.1.6 Lemmatization

Stemming is a process to reduce the word to its root stem for example run, running, runs, runed derived from the same word as run. Stemming removes the prefix or suffix from word like ing, s, es, etc. The problem with stemming is that it doesn't produce reliable results even if stemming is not required for a specific word. For example, words like "really" become "realli". Lemmatization is a technique similar to stemming, however the process behind it is different. It uses a systematic way to reduce the words into their lemma by matching them with a language dictionary and considering the context. Therefore, lemmatization is performed on the dataset.

3.1.7 Subsampling

In order to for the subsample to be representative of the original dataset correctly, it is drawn randomly using the *random* package from python. In fact, this ensures that each row in the baskets has the same probability to be included in the subsample.

3.2 Algorithm Specific Format

For market-basket analysis, each basket is required to contain only a unique value of an item. For example, the basket {apple, orange, apple} is invalid. Using an ordered dictionary (to maintain at least the order of the words in the original phrase), only the unique values of the items are taken.

3.2.1 Apriori algorithm specific format

For the Apriori algorithm, the data should be in a text or csv file in which each basket corresponds to a line. Each basket and each item in the basket are separated by comma.

```
['home,alone,apparently',
'st,louis,plant,close,would,die,old,age,worker,making,car,since,onset,mass,automotive,production',
'wsu,plan,quickly,became,hot,topic,local,online,site,though,people,applauded,plan,new,biomedical,center,many,
deplored,potential,loss,building']]
```

As it is implemented in Apache Spark environment, the list of lists is then transformed in *resilient distributed dataset* (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel.

```
[['home', 'alone', 'apparently'], ['st', 'louis', 'plant', 'close', 'would', 'die', 'old', 'age', 'worker',
'making', 'car', 'since', 'onset', 'mass', 'automotive', 'production'], ['wsu', 'plan', 'quickly', 'became',
'hot', 'topic', 'local', 'online', 'site', 'though', 'people', 'applauded', 'plan', 'new', 'biomedical',
'center', 'many', 'deplored', 'potential', 'loss', 'building']]
```

To compute the support values, all the items are considered altogether.

3.2.2 FP-Growth algorithm specific format

The FP-Growth algorithm is directly implemented in Spark, and it take itemsets in the format of a dataframe with two columns: id, items. Additionally, the baskets are transformed from string to an array of strings in order to access words individually.

```
+---+--------------------+
| id|               items|
+---+--------------------+
|  0|['home',  'alone'...|
|  1|['st',  'louis', ...|
|  2|['wsu',  'plan', ...|
|  3|['alaimo',  'grou...|
|  4|['often',  'diffi...|
|  5|['certain',  'amo...|
|  6|['charlevoix',  '...|
|  7|['another',  'lon...|
|  8|['time',  'report...|
|  9|['trying',  'hit'...|
| 10|['mhta',  'presid...|
| 11|['absurdity',  'a...|
| 12|['gm',  'labor', ...|
| 13|['wandry',  'matt...|
| 14|['cheap',  'said'...|
| 15|['andrade',  'chi...|
| 16|['let',  'hair', ...|
| 17|['born',  'april'...|
| 18|['house',  'minor...|
| 19|['first',  'love'...|
+---+--------------------+
only showing top 20 rows
```

Figure 3 View of the FP-Growth input format

4. Model Definition

This Section gives an overview of the algorithms that were chosen along with a brief description of the theory behind them. Then, a solution to address the scalability of the algorithms in presence of massive datasets is proposed.

4.1 Apriori Algorithm

This algorithm is given by R. Agrawal and R. Srikant in 1994 for finding frequent itemsets in a dataset for Boolean association rule. The name of the algorithm is Apriori because it uses prior knowledge of frequent itemset properties. It is based on monotonicity property which states that if U is frequent than each subset S of U will also be frequent; it is also called *Apriori property*. This algorithm is used for mining frequent item sets and devising association rules from a database. The algorithm accepts the minimum support, confidence and lift as hyperparameters.

The idea is that it first scans all baskets and retrieve which items are Singularly frequent. The mapping process tells for each word whether it is frequent or not and if it is frequent, it is then associated to a univocally integer number. At this point, a 2^{nd} scan of data is performed only on frequent items. This 2^{nd} scan aims to find the association between the singleton words, i.e. to identify sets of doubleton words which are frequent together. Similarly, another scan may be utilized to identify frequent sets of tripleton words, and so on. At each scan non-frequent words are discarded.

The Algorithm operates on a straightforward way. When the support value of an item set exceeds a certain threshold, it is considered a frequent item set. The algorithm steps are as follows:

1. Create a list of all words that appear in every basket and create a frequency table.
2. Set the minimum level of support and confidence. Only those words whose support and confidence exceeds or equals the threshold support and confidence are kept and rest other words are discarded.
3. All potential pairing of words must be made, keeping in mind that EF and FE are interchangeable.
4. Check the number of times each pair appears in baskets.
5. Only those sets of words that meet the minimum support and confidence criterion is kept.
6. To find tripletons which are frequent together a rule self-join is needed to build a three-item set. For example, the item pairings GH, HK, KN and NR state that two combinations with same initial letter are sought from these sets.

- GHK is the result of GH an HK
- KNR is the result of KN and NR

7. Threshold criterion is applied again, and resultant set will be the significant frequent word set.

There are some critical issues of Apriori algorithm:

1. At each step, candidate sets must be built, and size of candidate item sets could be extremely large.
2. To build the candidate sets, algorithm must repeatedly scan the database.

These two properties inevitably make the algorithm slower. In fact, for example if the 1-itemset comes out to be very large, for ex. 10^4, then the 2-itemset candidate sets would be more than 10^7. Moreover, for a dataset with a large number of frequent items or with a low support value, the candidate itemsets will always be very large. These large datasets require a lot of memory to be stored in. Further, Apriori algorithm also scans the database multiple times to calculate the frequency of the itemsets in k-itemset. So, Apriori algorithm turns out to be very slow and inefficient, especially when memory capacity is limited, and the number of transactions is large. To overcome these redundant steps, other algorithms were proposed which have as a base Apriori such as FP Growth Algorithm.

4.2 FP-Growth Algorithm

It can be seen as modern version of the above Apriori algorithm, as it is faster and more efficient while obtaining the same goal. The idea behind the FP Growth algorithm is to find the frequent item sets in a dataset while being faster than the Apriori algorithm. The Apriori algorithm basically goes back and forth to the dataset to check for the co-occurrence of products in the data set. To be faster, the FP growth algorithm changed the organization of the data into tree rather than sets. This tree data structure allows for faster scanning, and this is where the algorithm wins time.

There are two parameters used by FP Growth algorithm to find the association rules in the dataset.

1. Support: It tells the number of times or percentage that words co-occur.
2. Confidence: It tells us the number of times that rule occurs. This can be stated differently as conditional probability of the right-hand side given the left-hand side.

The algorithm steps are as follows:

1. Counting the occurrences of individual words: the occurrence of individual words is counted and stored in table.

2. Filter out non-frequent words using minimum support criterion – Decide a value for the minimum support every word or word set with fewer occurrence than the minimum support will be rejected.

3. Order the item sets based on individual occurrences- For the remaining words, we will create an ordered table. This table contains the words that have not been rejected yet, and the items inside a basket will be ordered based on individual word occurrence.

4. Create the tree and add the baskets one by one- Now, we can create the tree staring with 1st basket. Each word is a node in tree, and we also add a count to each word, which we will use for counting later. Similarly, we will move on to 2nd, 3rd …n baskets and build a tree, once this FP tree is constructed it is much faster to traverse it and find information on the most frequent item sets.

4.3 Environment and Scalability

In this study, the old newspaper dataset is above the capacity of a single machine. Therefore, there is a need for distributed framework.

The usual choice is MapReduce, a programming model within the Hadoop framework for distributed computing based on Java. It is a way of structuring your computation that allows it to easily be run on lots of machines. It enables massive scalability across hundreds or thousands of servers in a Hadoop cluster. It allows writing distributed, scalable jobs with little effort. It serves two essential functions: it filters and distributes work to various nodes within the cluster or map. It is used for large scale data analysis using multiple machines in the cluster. A MapReduce framework is typically a three-step process: Map, Shuffle and Reduce.

Spark offers a fast and easy way to analyze that data across an entire cluster of computers. Spark is an open-source, super-fast big data framework widely considered the successor to the MapReduce framework for processing big data. It includes an interface for programming a full suite of clusters with comprehensive fault tolerance and support for data parallelism. Spark is considered a Hadoop enhancement to MapReduce used for big data workloads.

Spark is good for:

- Data heavy tasks: as it uses Hadoop File System (HDFS).
- Compute Heavy Tasks: as it uses RAM instead of disk, to store immediate outputs. e.g iterative solutions.

Hadoop Distributed File System (HDFS) is a fault tolerant distributed file system, used by Spark. HDFS enables splitting a big file into *n* chunks and keep in *n* nodes. When the file is accessed, then

different chunks of data must be accessed across the nodes. On the other hand, Resilient Distributed Dataset (RDD) is a fault tolerant immutable collection of distributed datasets stored in main memory. On top of RDD, Data Frame API is designed to abstract away its complexity and ease doing Machine Learning on Spark.

The main difference between the two frameworks is that MapReduce processes data on disk whereas Spark processes and retains data in memory for subsequent steps. As a result, Spark is 100 times faster in-memory and 10 times faster on disk than MapReduce.

As a result of these considerations, Apriori and FP Growth algorithms were implemented on Spark framework.

5. Conclusions

This last section presents the results obtained by the FP-Growth algorithm. Then are outlined the main limitations of this paper as well as the recommendations for future works.

5.1 Results

5.1.1 Results of Apriori algorithm

After a great number of trials, the Apriori algorithm was considered too inefficient to run with the whole dataset. In fact, these actions were taken:

1. Several values of minimum support and confidence were considered such as 0.01, 0.05, 0.001..
2. The code was run for as long as 7 hours without a final output.
3. Different best practices to optimize the code and the Spark environment were followed.

Then, a subsample of the dataset was randomly created using n = 500. Min. Support is equal to 0.01.

5.1.1.1 Phase-1 Output

The first step produces a tuple of unique frequent itemset along with their support values obtained by "*reduceByKey*" operator.

```
[('catcher', 2), ('mike', 4), ('napoli', 1), ('dropped', 2), ('lineup', 1), ('offense', 2), ('three', 19), ('run', 11), ('home', 18), ('give', 4)]
```

The first table is then created filtering for minimum support which contains 367 items.

5.1.1.2 Phase-2 Output

After defining a function to remove the duplicate items from the tables, an algorithm is created to generate combinations of items in a while loop. The loop will end when there isn't any combination whose support value is more than min support value. In this case, two more tables created: one

containing pairs and one with triples. The "*reduceByKey*" operator is again used to count the support values for these items.

```
[((['three'], 19), (['three'], 19)),
 ((['three'], 19), (['run'], 11)),
 ((['three'], 19), (['home'], 18)),
 ((['three'], 19), (['question'], 6)),
 ((['three'], 19), (['love'], 7)),
 ((['three'], 19), (['whose'], 5)),
 ((['three'], 19), (['work'], 12)),
 ((['three'], 19), (['cannot'], 6)),
 ((['three'], 19), (['let'], 6)),
 ((['three'], 19), (['go'], 15))]
```

Figure 4 Snap of the pair itemsets found by Apriori

```
[((('said', 'work'), 5), (['work'], 12)),
 ((('said', 'work'), 5), (['said'], 113)),
 ((('said', 'would'), 16), (['would'], 41)),
 ((('said', 'would'), 16), (['said'], 113)),
 ((('last', 'year'), 13), (['year'], 56)),
 ((('last', 'year'), 13), (['last'], 26)),
 ((('last', 'said'), 6), (['said'], 113)),
 ((('said', 'school'), 8), (['said'], 113)),
 ((('got', 'said'), 5), (['said'], 113)),
 ((('say', 'year'), 5), (['year'], 56))]
```

Figure 5 Snap of the triple itemsets found by Apriori

5.1.1.3 Confidence Table

Finally, the confidence values are calculated and printed in a dataframe in descending order.

	Antecedent	Consequent	Confidence
101	[louis]	[st]	100.000000
61	[going]	[said]	91.666667
45	[authority]	[said]	83.333333
78	[feel]	[said]	83.333333
33	[need]	[said]	71.428571
..	...	...	...
74	[said]	[feel]	4.424779
72	[said]	[getting]	4.424779
116	[said]	[county]	4.424779
42	[said]	[change]	4.424779
87	[said]	[little]	4.424779

[130 rows x 3 columns]

Figure 6 Snap of the Confidence Table

It is interesting to notice from the table that, for example, the pairs *["feel"] / ["said"]* and *["said"] / ["feel"]* have very different values because of the formula to calculate the confidence value.

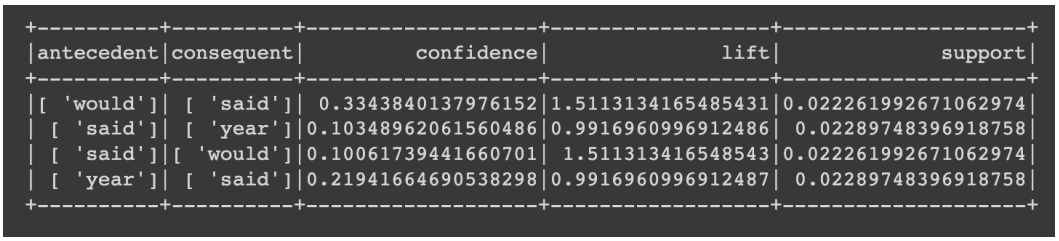
5.1.2 Results of FP-Growth algorithm

Again, the same kind of steps highlighted above were followed in the implementation of the FP-Growth algorithm. Unlike Apriori, frequent itemsets and association rules were found.

items	freq
['give']	13403
['among']	10668
['night']	17597
['event']	11121
['york']	11943
['health']	12779
['office']	15790
['better']	14466
['world']	16258
['never']	14725
['room']	11430
['early']	12198
['service']	18020
['change']	12885
['point']	22520
['including']	19745
['season']	27323
['national']	13620
['run']	20653

Figure 7 Snap of the Support values of FP-Growth

In this output table, the frequency of the itemsets is shown. Even though the items are not ordered, it can be seen that for example "*season*" was the most used word in this batch along with "*point*" and "*run*". "A*mong*" was the least used. However, no pattern can be inferred from this table: it cannot be said that there is a high probability that if the word "*point*" is used in a sentence, then "*season*" is also used.



antecedent	consequent	confidence	lift	support
['would']	['said']	0.3343840137976152	1.5113134165485431	0.022261992671062974
['said']	['year']	0.10348962061560486	0.9916960996912486	0.02289748396918758
['said']	['would']	0.10061739441660701	1.511313416548543	0.022261992671062974
['year']	['said']	0.21941664690538298	0.9916960996912487	0.02289748396918758

Figure 8 Confidence values and lift found by FP-Growth

The association rule table above represent the patterns that can be discerned from the dataset. For example, it can be said that when the word "*would*" is used, then "said" is also used with 33% probability. It has to be noticed that the pair ["*would*", "*said*"] has a different confidence than the pair ["*said*", "*would*"].

In conclusion, FP-Growth algorithm has revealed to the more efficient to implement when limited resources are involved (even time) to analyze a very big dataset.

5.2 Limitations and Recommendations for future work

This study was expecting to find out more and more relevant association rules. Nevertheless, predict patterns of words used in a sentence or in a speech (if there are any) is a very difficult task since a lot of variables are involved. The limitations of this paper are stated in the following paragraphs.

5.2.1 Limitations inherent to the analysis

Predicting if and when a word will be used in a sentence is a very challenging task to carry out. In fact, it is not said that association rules can be found within a text mining analysis. A variety of factors contribute to the use of one word instead of another like context. For example, consider a formal context in comparison with an informal one. In addition, the meaning of the sentence could remain unchanged with the use of synonyms and, sometimes, completely different words. Probabilistically speaking, as the search for frequent itemset becomes stricter (higher minimum support) as well as for more insightful rules (higher confidence and lift), the complexity rises exponentially. Furthermore, mostly only pairs of words are considered for such rules, as triplets (and so on…) are very difficult to encounter in random phrases.

5.2.2 Possible workarounds for the Apriori algorithm

The Apriori algorithm is very inefficient, but for research purposes it might still be appealing to implement it. A solution to the speed could be running the Apriori using different underlying data structures like Hash-tree, Trie or Hash-table and compare performances based on time. Additionally, Apriori should be implemented using more powerful resources such as Spark in *yarn* mode, or trying different scalable/Apriori-specific environments like Rapid Miner. Furthermore, the Apriori algorithm is subject to False Positive/False Negative problem. In this paper, the baskets created by Apriori are filtered for the ones originally contained in the dataset; still some might be present.

5.2.3 Recommendations for future work

The first step to improve this analysis would be to gather additional variables to create context for the analysis, and create a basket for each context. In this case, for example, the genre of the newspaper or even of the news itself. In this way, the patterns found could have more potential meaning. Apriori and FP-Growth were implemented in this analysis, however new ones are constantly being proposed and tested such as Eclat, PCY and YAFIM.

6. References

[1] Frequent Pattern Mining. *https://en.wikipedia.org/wiki/Frequent_pattern_discovery*

[2] Apache Spark. *https://spark.apache.org/*

[3] What is Spark. *https://databricks.com/spark/about*

[4] Apriori Algorithm. *https://en.wikipedia.org/wiki/Apriori_algorithm*

[5] Pankaj Singh, Sudhakar Singh, P. K. Mishra, and Rakhi Garg. *A Data Structure Perspective to the RDD-based Apriori Algorithm on Spark*

[6] Hongjian Qiu, Rong Gu, Chunfeng Yuan, Yihua Huang (2014). *YAFIM: A Parallel Frequent Itemset Mining Algorithm with Spark*

[7] Anand Rajaraman, Jeff Ullman, Jure Leskovec. *Mining of Massive Dataset*

[8] D S Maylawati 2018 IOP Conf. Ser.: Mater. Sci. Eng. 434 012043. *The concept of frequent itemset mining for text*

[9] Apriori Algorithm. *https://github.com/sergencansiz/apriori-pyspark/blob/master/AprioriPySpark.py*

[10] Numpy. *https://numpy.org.*

[11] Pandas. *https://pandas.pydata.org.*

[12] NTLK. *https://www.nltk.org/*

[13] Contractions. *https://github.com/kootenpv/contractions*

[14] Kaggle API. *https://github.com/Kaggle/kaggle-api*

[15] HC Corpora. *http://corpora.heliohost.org/*

[16] Random Sampling. *https://docs.python.org/3/library/random.html*