Mining Consistent Popular Topics in Tweets

Data Mining Course Project

Riccardo Scilla - 213723
University of Trento
2nd year, CS program
riccardo.scilla@studenti.unitn.it

ABSTRACT

This work is about the mining of consistent frequent topics in tweets considering the time of their publication. A modification of the FP-Growth algorithm for Frequent Itemsets mining is proposed with the aim to extract terms that become frequent together throughout time. A comparison with Apriori algorithm is described to show the scalability of the process.

1 INTRODUCTION & MOTIVATION

Data Mining is used to extract relevant information from large databases in an efficient way. Among the various applications in which Data Mining is used, a very important and common one is the Frequent Itemset Mining, which aim is to find set of items that appear "frequently" in baskets. A typical example is the Market-Basket Analysis, where baskets represent the sets of products that are bought together in a market and through the Frequent Itemset technique we want to identify the items that people buy together frequently.

The aim of this project is to extract Time Consistent Topics considering a dataset of tweets, where each tweet is represented by a short text and a specific time when it is sent by a user. In this context, Topics are identified by sets of words that appears together very frequently in time.
Some techniques for mining frequent itemsets consider the whole list of records in the database to determine whether an itemset is frequent or not, but in the case analyzed here the time when tweets are sent is fundamental because it allow us to extract information about the "consistency" of the topics. This means that we not interested so much about which set of words appears frequently in general over all the data, but what we want is find set of words that when they appears, they appears together. This is particularly important because we could find topics that are not that frequent if considered the entire time interval of a dataset, while they become interesting if concentrated in a smaller time duration.

For example, consider a dataset containing 100.000 tweets spread in one month; with a minimum support of 0.6%, a word is considered frequent if it appears in at least 600 tweets. If we consider a frequent word that has been used constantly for the entire month, it would appear, on average, 20 times per day. Consider now a new word that has been introduced in tweets just last week and it appears 60 times per day, thus in 420 tweets. So it is not frequent, even though it is 3 times more popular than the first word considered. However, if we consider only the last week as timespan and we compute the support, we will obtain a support of ~2.0%. Therefore this new word would appear interesting and useful to extract more information from the data.

A way to solve this problem is to consider a commonly used Frequent Itemset algorithm and extend it to take into consideration also time while mining the data. In this work I will discuss a modified version of the FP-growth algorithm which exploits temporal supports of tweets.

2 RELATED WORK

How previously anticipated, in this work Topic are identified by a set of words that appear frequently in data. This can be generalized in the problem of finding Frequent Itemsets in data.

Frequent Itemsets are a set of items that appear in the data very "frequently". To better define what "frequently" means in the general statement of the problem, we have to define the **Support Threshold** $\sigma \in [0, 1]$ and the **Support** of a Itemset I in the data D as the fraction of times in which I appears in D. The set I is considered "frequent" if its support is greater than the *Support Threshold.*

$$support(I, D) = \frac{\text{\# I in D}}{|D|}$$

In this work, the concept of support is modified in order to consider also time. Given that the tweets present both sets of words (items) and time, we can associate at each item a timespan $[t1, t2]$ in which it exists in the data. For example, we could find that the itemset "A,B,C" is found 2 times in the interval [25Jul20, 30Jul20]. At this point to understand whether an itemset I is "frequent" or not, we can consider as Support the faction of records that contains that item in its timespan and compare it with the *Support Threshold.* Calling **Temporal Support** of a itemset I (TS_I) the duration of the timespan $[t1, t2]$ of the itemset I, the formula to compute the Support becomes:

$$support(I, d_I) = \frac{\text{\# I in } d_I}{TS_I}$$

The *Temporal Support* is filtered using a **Temporal Support Threshold** $[\tau_1, \tau_2]$ that are respectively the minimum and maximum duration of time in which we can find a frequent itemset. This is compared with the *Temporal Support* to filter out itemsets that appears in a timespan which is too short or too large, depending on the user needs.
For example, if an itemset I appears only one time in a certain time t, it will have *Support* 1, but of course it cannot be considered as frequent: setting τ_1 of the *Temporal Support Threshold* to, e.g., 2, we can handle this situation.
On the other hand, if a word appear in every transaction in our data, it is frequent but not so interesting for our application: setting τ_2 of the *Temporal Support Threshold* to, e.g., 7, we can retrieve only those itemsets that appears at most one week.

The algorithm used to mine Frequent Itemsets in this context, is a modified version of the FP-Growth algorithm.
A common technique used for Frequent Itemset mining is the Apriori algorithm. This procedure scan multiple times the data with the aim of finding frequent itemsets: the first scan finds frequent items, the second frequent pairs and so on. Once frequent items are found, the algorithm generates pair candidates taking unions of frequent items. This procedure follow the idea of Monotonicity in a negative form, that is if an item is not frequent, then no pair including it can appear in a frequent pair. This allows pruning steps that for sure will not generate Frequent Itemsets.
Considering the problem we want to solve in this work, the concept of time can be introduced by modifying the meaning of what is a frequent by considering the modification on the *Support* with the temporal duration explained before.

The candidate generation procedure is the bottleneck the Apriori algorithm, so that the whole process leads to high costs in terms of time and brings even more problems if considered scalability with bigger datasets.
For this reason FP-growth is used: it is a method of finding frequent patterns without the need for candidate generation. The idea is to store the database in the form of tree, called FP-Tree, which structure maintains the associations between the itemsets. Using the FP-Tree there is no need to scan the database multiple times, but only two are required for processing.
Once the FP-Tree is generated, it can be explored with the aim to build conditional patterns based on each item. From these patters, the generation of FP-Tree is repeated by building a conditional FP-Trees for each item base.

The way time is introduced with respect to the common FP-Growth algorithm is in how the *Support* is computed: using the modified *Support* described before and the filter given by the *Temporal Support Threshold*, we can arrange the items give importance the ones that appear frequently also in a short time.

The previous step allow us to extract all frequent itemsets in time from data. These could be many depending on the type of the data and on the parameters used. Also, many frequent itemsets could be included in others that are frequent, too. For example, given "A,B" a frequent itemset in its timespan, we could find also "A,B,C" that is frequent in the same timespan.
To have a clearer view on which are the frequent topics, a post-process is applied to the frequent itemsets found with FP-Growth and only the maximal ones are extracted: these are the frequent itemsets for which none of its immediate supersets are frequent.

3 PROBLEM STATEMENT

The algorithm takes as input the Dataset and different parameters and returns as output a set of Frequent Itemsets that represent the topic. The detailed definition is described as follows:

Input:

- Dataset D in csv format containing the information of a tweet per line. Each tweet has:
  - integer representing the day when it is written
  - string of words separated by a space
- Support Threshold $\sigma \in [0, 1]$
- Temporal Support Threshold $[\tau_1, \tau_2]$, each value $\in \mathbb{N}$

Output:

- Time-related Frequent Itemsets, i.e. set of words that represent a topic.

4 SOLUTION

The solution I implemented for the problem of finding consistent popular topics in tweets is structured in different steps:

(1) *Preprocessing*: many processing steps to transform raw data into something useful for the other steps of the application.
(2) *Import*: import the preprocessed csv and arrange transaction information in a list.
(3) *FP-Growth*: core part of the solution; build a temporal FP-tree based on the preprocessed data and mine it recursively to obtain all the Frequent Itemsets.
(4) *Maximal Frequent Itemsets*: manipulate the Frequent Itemsets in order to obtain the Maximal Frequent Itemsets.

4.1 Preprocessing

Dataset preprocessing is required in order to remove corrupted data or information that we already knew are useless. The remaining part needs to be organized in an efficient way to be handled by the FP-growth algorithm. More information about the Dataset and its preparation is described in section *6. Dataset*. The pseudocode for the preprocessing is shown in *Algorithm 1*.

Algorithm 1: Preprocessing

```
Data: Raw Data in csv format
Result: Preprocessed Data in csv format
data ← read csv raw data;
Extract date and text fields in data;
Remove rows with nan fields and duplicates in data;
for text field in data do
    remove urls;
    extract characters, numbers hashtags and tags
      (remove punctuation and emoji);
    remove stand-alone numbers and characters;
    lowerize;
    tokenize;
    remove stopwords;
end
for date field in data do
    convert string to datetime type;
end
Sort data according to date field;
Create dates_dict where each date is associated to an
  integer with increasing value;
Replace date field in data according to the respective
  integer in dates_dict;
```

4.2 Import

Data in csv is manipulated to consider every preprocessed transaction as a list. For example:

$$1, "A, B, C" \longrightarrow [1, [A, B, C]]$$

The result of these operation on all the preprocessed csv is given as input of the temporal FP-Growth algorithm.

4.3 FP-Growth

Implementation of FP-Growth algorithm modified to consider time of itemsets. FP-Growth works in a divide and conquer way. It requires two scans on the database. FP-Growth first computes a list of frequent items sorted by *Support* in descending order during its first database scan. In its second scan, the database is compressed into a FP-tree, where each item is represented as Node of the tree.

The Node attributes are in the following table:

Class: FP-Node	
itemName	Name of the item
count	Number of times the item appears in that branch
children	Set of Nodes that are children of it in that branch
time	List of integers representing the time in which the item appears in that branch
next	Link to a Node with the same itemName in another branch

The presudocode for the FT-tree construction is shown in *Algorithm 2* below.

```
Algorithm 2: construct_FP-Tree
Data: Preprocessed data, Support Threshold, Temporal
      Support Threshold
Result: FP-tree, HeaderTable of Nodes, SupportTable and
        TimespanTable of Items
Initialize HeaderTable, SupportTable and TimespanTable;
Scan 1: for itemset in data do
    Store in HeaderTable the count of each item and in
      TimespanTable the time it appears;
    Delete items below support and timespan thresholds
      or save in SupportTable;
end
if no items left then
    return
end
Create root Node;
currentNode ← root;
Scan 2: for itemset in data do
    Take only frequent items and sort them by Support in
      decreasing order;
    for item in itemset do
        if item is children of currentNode then
            increment the frequency of the item
        else
            create newNode and append it as children of
              currentNode;
            if item is not in HeaderTable then
                add newNode to the HeaderTable[item]
            else
                link it at the of HeaderTable[item]
            end
            currentNode ← newNode
        end
    end
end
```

Then FP-Growth starts to mine the FP-tree for each item whose support is larger than the *Support Threshold* and temporal support is larger than *Temporal Support Threshold* by recursively building its conditional FP-tree. The algorithm performs mining recursively on FP-tree. The pseudocode for the FT-tree mining is shown in *Algorithm 3*.

```
Algorithm 3: mine_FP-Tree
Data: HeaderTable, SupportTable, Support Threshold,
      Temporal Support Threshold, prefix (init empty),
      frequentItemsets (init empty)
Result: List of Frequent Itemsets
Sort SupportTable in increasing order;
for item in sorted SupportTable do
    newFrequentItemset ← prefix + item;
    if newFrequentItemset contains more than 2 items then
        add newFrequentItemset in frequentItemsets
    end
    tree_node ← HeaderTable[item];
    Initialize conditional_path;
    while tree_node do
        prefix ← path from tree_node to the root;
        if prefix then
            add prefix in conditional_path;
        end
        tree_node ← tree_node.next;
    end
    newHeaderTable, newSupportTable ←
      construct_FP-Tree
     (conditional_path, Support Threshold, Temporal
      Support Threshold);
    if newHeaderTable contains items then
        mine_FP-Tree (newHeaderTable,
          newSupportTable, Support Threshold, Temporal
          Support Threshold, newFrequentItemset,
          frequentItemsets);
    end
end
```

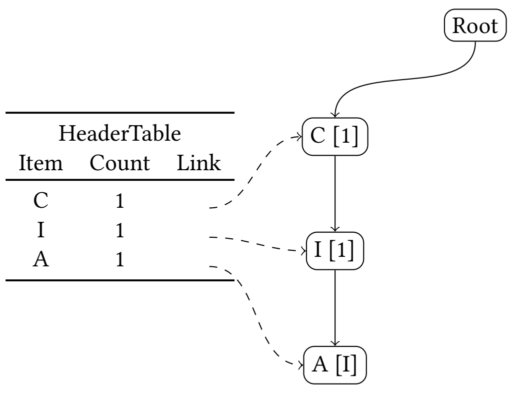
4.3.1 Example. To make more clear the idea on how we can exploit the the temporal information in FP-growth algorithm, I will show a simple example. Suppose we have the data shown in the left table, *Support Threshold*=0.5 and *Temporal Support Threshold*=2. In the first scan we compute the Timespan and Support: items F and H are removed because its Support is lower than 0.5; E is removed because it appears less than 2 times.

Time	Itemset
1	A, C, F, H, I
2	A, B, C, G
3	B, C, D, G, I
4	A, C, I
5	C, D, E, H, I
6	A, D, F, G

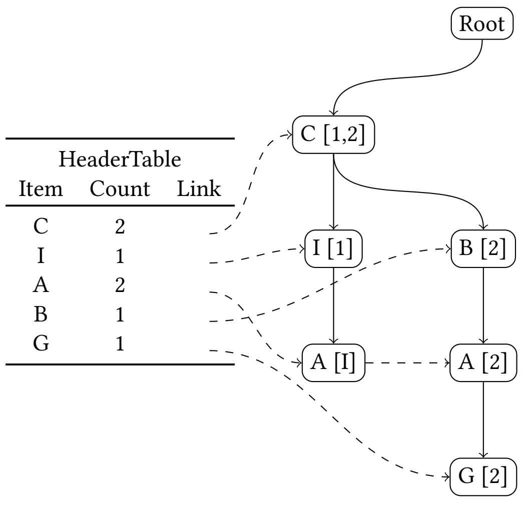
Item	Timespan	Support
A	[1,2,4,6]	0.67
B	[2,3]	1
C	[1,2,3,4,5]	1
D	[3,5,6]	0.75
E	[5]	1
F	[1,6]	0.3
G	[2,3,6]	0.6
H	[1,5]	0.4
I	[1,3,4,5]	0.8

In the second scan, each itemset is sorted by the Support of the item (eg. [1| C, I, A]) and appended as sequence of Nodes starting

from the root. The HeaderTable stores the information about the count and the first Node in the tree with a given ItemName.
The situation of the tree after the process of the first itemset is the following:



Adding the second transaction [2| C,B,A,G], Node C is updated and a new branch is added from it. Node B and G are added in the HeaderTable, while Node A is updated and A.next link is added from the Node A of the previous branch. The situation of the tree after the process of the second itemset is the following:



This process is repeated for every itemset in the our data. Once done, the second step is to mine the tree, generating conditional patterns. Starting from the items with lower Support in the SupportTable, we get the Link from the HeaderTable and compute the prefix of that path; then we move to the next node and repeat it until there are no more next nodes.
For example, consider item A form the tree shown above: the Node linked to A is (A [1]), which prefix is [1|I,C]; the next node is (A [2]) which prefix is [2|B,C]. The conditional pattern is the list of these two, that are used as input for a generation of a new FP-Tree. Of course, also for this Tree *Support Threshold* and *Temporal Support Threshold* are used to determine whether an Item is frequent or not. In this small example, {A} would not be considered as a frequentItemset, because it is a single item; but following the procedure for its two conditional patterns, we will get that C is also frequent, so {A,C} would be a frequentItemset.

4.4 Maximal Frequent Itemsets

The pseudocode for Maximal Frequent Itemsets extraction is explained in *Algorithm 4*.
First, user can specify what is the maximum length of the Itemset to obtain, to filter out many topics that could be meaningless or too generic. Then the Frequent Itemsets are aggregated according to their support: all the Itemsets that have a Support that is lower than a certain value are associated to that value. Last, scanning all the Frequent Itemsets, if one is not a subset of another Frequent Itemset, then it is Maximal.

Algorithm 4: Maximal Frequent Itemsets

```
Data: List of Frequent Itemsets, Max length
Result: List of Maximal Frequent Itemsets
Filter all the Frequent Itemsets below the Max length;
Group Frequent Itemsets according to their Support;
Aggregate each group of Frequent Itemsets with the ones
  that have a lower Support;
for Frequent Itemset i in List of Frequent Itemset do
    for Frequent Itemset j in group of Frequent Itemset do
        if i is not j and i is subset of j then
            i is not a Maximal Frequent Itemset;
            go to the next Frequent Itemset i;
        end
    end
    i is a Maximal Frequent Itemset;
end
```

5 IMPLEMENTATION

The code regarding the Preprocessing, FP-Growth and Maximal Frequent Itemsets is written in Python 3.7.9 64bit. Data is originally saved in a csv file and converted in preprocessing into another csv file, with the format explained in Dataset Section.
The specifications of the machine used are: i7-7700HQ with 16GB main memory, 1TB hard disk having Microsoft Windows.
The FP-Growth method is performed on a single thread, due to its efficiency in processing the data for the dataset used in this work.
Other operations are executed in paralyzed way using pySpark 3.0.1, which allow to handle portion of data using map-reduce model. The import of preprocessed data is done using pySpark to optimize the procedure parallelizing with 4 workers. Also for the Maximal Frequent Itemset extraction I used pySpark with 8 workers.

6 DATASET

The dataset used for the project contains 179k tweets and is taken from https://www.kaggle.com/gpreda/covid19-tweets. It presents 13 column values such as *user_name*, *user_location*, *user_followers*, *date*, *text*, *hashtags*.

A python script is used to perform Data Preparation so that we retrieve data that is useful and correctly formatted for the rest part of the project. First, most of the column values are not useful for our purpose, to they are filtered out and only *data* and *text* are maintained. All the records that contains at least one *nan* value are discarded, together with the duplicated records. The remaining columns *date* and *text* are preprocessed separately.

Text values are of type string and contains words, numbers, punctuation, URLs, emoji, hashtags and tags. First URLs are removed together with punctuation and emoji. Then all the standalone characters and numbers are removed. Secondly lowerize and tokenize is performed on each record and all the duplicated words are removed. Finally, the library NLTK is used to remove English stopwords, i.e. words which does not add much meaning to a sentence and so they can be safely ignored. In this way we obtain a list of good words, hashtags and tags that can be processed.
In many situations I found that words that are separated by a space, in reality refers to the same concept. For example, *new york* refers to the city, but, because of the space, the tokenization produces [*new,york*] as independent words. Because these two words are very likely to be found together, their combination is considered as frequent itemset. To avoid this situation, I replaced these kind of set of words as:

$$new\ york \longrightarrow new\_york$$

so that it is considered as a single word. This commonly happens also for name-surname pairs.

Date values are of type Date in the format YYYY-MM-DD H:M:S and are distributed over a period from 25Jul20 to 30Aug20. The script imports these dates as string, so the first thing done is to convert strings to datetime, following the format described before but discarding the information of H:M:S. After this, all the dates are sorted and an integer value is assigned in order to have a discrete sequence of dates. For example, to the date 2020-07-25 is assigned the value 1, because it's the first date in the data; to the date 2020-07-27 is assigned the value 3 and so on until 2020-08-30 which is assigned the value 38.

The integer obtained is concatenated to the preprocessed text related to that record, so we get a table of [integer|good words]. It is saved in csv file that can be imported as input of the actual program.

7 EXPERIMENTAL EVALUATION

The procedure explained is compared with a baseline model that is the Apriori algorithm modified to obtain temporal consistent Frequent Itemset. This algorithm is run on a single thread as the FP-Growth.
The experiments were performed on the preprocessed data that is sampled taking random transactions to see the behaviour in different situations. The comparison between the baseline Apriori and temporal FP-Growth is performed considering the same set of sampled data.
To the evaluate the performance I run multiple tests changing the parameters of the algorithms. These parameters are:

- Number of transactions in data
- Support Threshold
- Temporal Support Threshold

The performance evaluation consists the Speed and Quality of the results. To test the Speed I run multiple experiments changing one parameter at the time and plotting the results. Given that the preprocessing and Maximal Frequent Itemset extraction steps are the same for both models, only the time to compute Apriori and FP-Growth is taken into consideration. To test the quality I consider the similarity on the Maximal Frequent Itemset that I find using the two techniques and briefly their content.

Speed Evaluation
To test how the number of transaction in data affects the performance, I choose different values of number of transactions as [1000, 5000, 10000, 50000, 100000, 500000] while the Support Threshold is 0.5 and the Temporal Support Threshold is set with a minimum of 3 and a maximum of 21 days. These last two parameters remain the same for all the experiments.
The results on the time to execute the various run are shown in tables 1 and 2. A plot of these times is shown in figure 1.
As we observe, the time consuming in temporal FP-Growth in each group of transactions is less than it in the Apriori, and the difference increases more and more as the number of transactions increases. The time to execute the FP-Growth remains relatively flat even though the number of transactions increases.

	Apriori		
# Transactions	Time (s)	# FI	# MFI
1000	0.12	4	1
5000	0.80	37	12
10000	2.65	170	20
50000	40.87	206	93
100000	141.31	657	196
150000	270.18	954	276

Table 1: Apriori: # Transactions

	FP-Growth		
# Transactions	Time (s)	# FI	# MFI
1000	0.01	4	1
5000	0.06	37	12
10000	0.13	170	20
50000	0.53	206	93
100000	1.51	657	196
150000	1.59	954	276

Table 2: FP-Growth: # Transactions

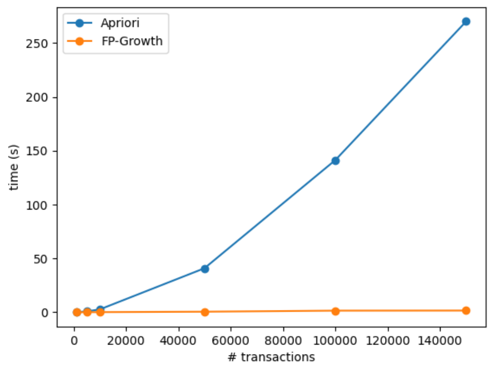


Figure 1: # Transactions-Time comparison

To test how the Support Threshold affects the performance, I choose different values as [0.1, 0.3, 0.5, 0.7] while the number of transactions are 20000 and the Temporal Support Threshold is set with a minimum of 3 and a maximum of 21 days. These last two parameters remain the same for all the experiments.

The results on the time to execute the various run are shown in tables 3 and 4. A plot of these times is shown in figure 2.
As we can see from the results, with lower support threshold, more words are accepted so the processing time increases. This effect does not impact too much in the case of FP-Growth, which maintains the time below the second.

Apriori			
Support Threshold	Time (s)	# FI	# MFI
0.1	364.38	720	238
0.3	43.86	223	91
0.5	7.93	114	42
0.7	1.07	17	11

Table 3: Apriori: Support Threshold

FP-Growth			
Support Threshold	Time (s)	# FI	# MFI
0.1	0.28	720	238
0.3	0.30	223	91
0.5	0.26	114	42
0.7	0.18	17	11

Table 4: FP-Growth: Support Threshold

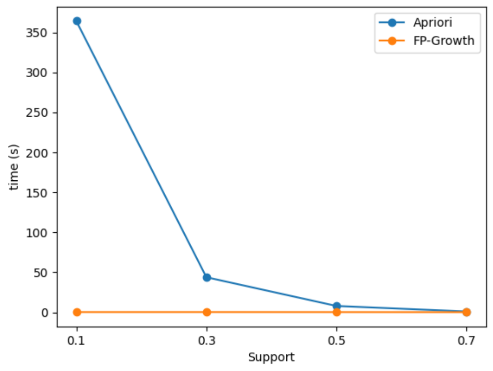


Figure 2: Support Threshold-Time comparison

To test how the Temporal Support Threshold affects the performance, I choose different values: the minimum number of days is set 3 for all the test while the maximum changes as [7, 14, 21, 28, 35]. The number of transactions are 20000 and the Support Threshold is set to 0.5. These last two parameters remain the same for all the experiments.
The results on the time to execute the various run are shown in tables 5 and 6. A plot of these times is shown in figure 3.
From the results we can see that increasing the time window tolerance for the items, more Frequent Itemsets are generated. This procedure has in impact on the Apriori method while FP-Growth it's almost insensible.

Apriori			
Temporal Support Threshold	Time (s)	# FI	# MFI
[3, 7]	4.11	94	28
[3, 14]	5.12	97	31
[3, 21]	7.33	114	42
[3, 28]	14.16	173	54
[3, 35]	26.97	243	83

Table 5: Apriori: Temporal Support Threshold

FP-Growth			
Temporal Support Threshold	Time (s)	# FI	# MFI
[3, 7]	0.19	94	28
[3, 14]	0.20	97	31
[3, 21]	0.20	114	42
[3, 28]	0.21	173	54
[3, 35]	0.22	243	83

Table 6: FP-Growth: Temporal Support Threshold

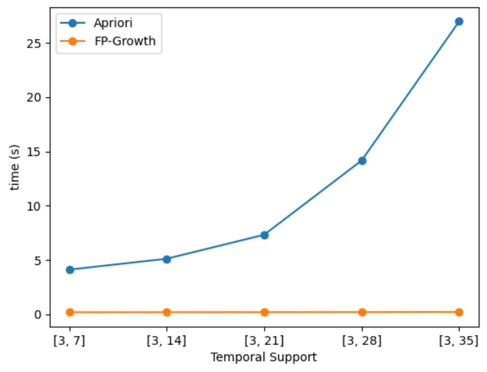


Figure 3: Temporal Support Threshold-Time comparison

Quality Evaluation

Considering the experiments shown before, we can see that the number of Frequent Itemsets and Maximal Frequent Itemsets are basically the same, so we can assume that they produce similar results. To better see what the algorithm produce, I ran different tests using the FP-Growth processing and the whole dataset. I take a sample of the results obtained using different parameters. In table 7 the topics using Support 0.6 and Temporal Support [7,21] are shown. We can see that the most topics are of two words and a many hashtags ad tags are present. This is because of they intrinsically must be the same, so if they become viral it's very likely to become a topic. Many are related to Indian or American people so we can guess the location of where are sent. Many others are related to investments like crudeoil and eth but also ebay.
The dataset was on covid19 tweets, but as we can see only few topics shows explicitly the "covid19" words. This is caused by the fact that the time window is restricted to 3 weeks.

To see the results on the whole timespan, I ran another test using Support 0.7 and Temporal Support [7,38]. The results are shown in table 8. This test produces 20000 Maximal Frequent

Topic	Timespan
{@infodivpune, @mahadgipr}	[16, 30]
{#getfit, #fitspiration}	[8, 14]
{169d, 148d, 109d}	[17, 25]
{reimburse, nits}	[10, 17]
{@chouhanshivraj, madhya_pradesh}	[2, 19]
{#thursdaymorning, #thursdaythoughts}	[7, 21]
{#bruh, #familyguy}	[2, 10]
{#crudeoil, #commodities}	[18, 25]
{southafrica, #saps}	[2, 12]
{#paidleaveforall, #savechildcare}	[1, 16]
{#ebayseller, #ebaydeals, #ebayvintage, #ebayfinds}	[30, 38]
{conduction, unsympathetic}	[17, 25]
{@cmofkarnataka, @bsybjp}	[10, 18]
{bhopal, madhya_pradesh}	[2, 21]
{@mdavidcartoons, #thedrum}	[19, 25]
{#bbtvi2020, #vergonhatvi, #bb2020tvi, #selectivitat2020}	[2, 10]
{lga, sensitization}	[10, 22]
{#amitabhbachchan, #abhishekbachchan}	[10, 17]
{#ncyt, #novacyt}	[3, 17]
{#r2p, #macron}	[24, 30]
{#neebank, #eth}	[12, 21]
{#kibarjalurgemilang, #merdekamoment, #malaysiakumerdeka, #malaysiaprihatin}	[17, 30]
{singh_chouhan, shivraj, madhya_pradesh}	[2, 19]
{#oc, #staylocal, #occovid19, #variance}	[2, 16]
{hmf, maldivian}	[10, 22]
{@juniorbachchan, #abhishekbachchan}	[10,17]

Table 7: Topic: 0.6 Support, [7,21] Temporal Support

Topic	Timespan
{#covid19, police, deaths}	[1, 38]
{get, treatment}	[2, 38]
{#covid19, #covid, good}	[1, 38]
{support, free}	[1, 38]
{lost, #covid19, job}	[1, 38]
{#covid19, #covidvic19}	[12, 38]
{july, spread}	[1, 38]
{#covid19, part, people}	[1, 38]
{cases, deaths, infection}	[2, 38]
{understanding, better}	[1, 38]
{cases, hospitals}	[2, 38]
{follow, retweet}	[2, 38]
{representation, lack}	[2, 37]
{@fema, @realdonaldtrump}	[1, 38]
{#covid19, also, positive}	[1, 38]
{today, medical}	[1, 38]
{#covid19, #coronavirus, cases, global}	[1, 38]
{target, set}	[2, 38]
{town, july}	[2, 38]
{going, trump}	[1, 38]
{#covid19, school, go, back}	[1, 38]
{#covid19, country, #russia, russian}	[2, 38]
{#covid19, birthday}	[2, 38]
{day, crosses, lakh, tests, india, #covid19}	[2, 38]

Table 8: Topic: 0.7 Support, [7,38] Temporal Support

Itemsets, extracted from 230000 Frequent Itemsets, so these results show only a limited view of what the algorithm produces. In any case it gives the idea that without limiting the maximum Temporal Support Threshold we can see that almost all the topics are related to covid19 and they are spread over all the timespan of the dataset.

8 CONCLUSION

The aim of the work was to design a solution to get Consistent Topics in time from tweets. This is achieved treating topics as Frequent Itemsets and considering the timespan of each word to define what "frequent" means for our purpose. A modification of the FP-Growth is proposed and compared with the baseline Apriori algorithm: this solution shows good results in scalability even using a single thread.
Further improvements of this process could be an implementation in Spark of the FP-Growth algorithm: this would improve scalability if used with datasets that are bigger than the one used in this work.
Another improvement could be done in preprocessing, using NLU techniques to better extract information on raw tweets: this would improve the overall quality of the topics generated.