Finding consistent topics in tweets

Andrea Fox

1 INTRODUCTION & MOTIVATION

The goal of this work is to find consistent topics in a dataset containing more than 170 thousand Covid-19 related tweets on the period of time between 24 of July 2020 and 30 of August of the same year.
The main piece used to find the topics consistent in time is going to be the A-Priori algorithm, which is in general used for finding frequent itemsets in a list of baskets: indeed, our problem can be reformulated as the search for frequent sets of words inside a list of baskets, each of those containing the words in a single tweet.
At first the dataset is going to be divided into 26 smaller subsets: one for all the tweets written in a specific day and on each of this subsets we are going to apply the algorithm for finding frequent itemsets chosen. Moreover we will individuate 7 longer periods of several days each: a tweet is going to be considered consistent throughout the whole time frame if it appears inside the set of frequent itemsets in at least a certain number of days, across a certain number of different periods. So, we might say that our goal is to find those sets of words which can be found frequently over the whole period studied, or at least in different periods over the whole period studied. This means that for the purpose of our analysis, those topics which appear very often in just a short span of days might not be considered consistent (depending on the parameter that the user chose).
Clearly, to follow what just described, the number of periods across which a term has to appear as frequent to be considered consistent in time, has to be sufficiently high.

Even though people who write on Twitter are not a complete representation of the world's population, these tweets can give some insights on how different people around the globe responded to an unprecedented situation like the pandemic that characterized the whole 2020. This kind of analysis might be very useful to understand how the mental health of different people evolved during a situation which caught all of us unprepared. Moreover it is even more clear how the pandemic has changed (in general in worse) the economic condition of several people and by looking at this data we might have a better idea of which areas are the one that have been hit the most. Finally, this kind of analysis might simply be useful to understand what topics have been considered more important by the bubble of tweeters considered, and future analysis might compare these results to the one obtained by making a survey among other people.

2 RELATED WORK

As previously stated, the problem can be reformulated into an application of the frequent itemsets problem on the *market-basket* model. On one hand, we are going to have the items (in our case the possible words of the vocabulary) and on the other we have the baskets, which are small sets of items. This last group, in the case analyzed, consists in the tweets, which clearly can be seen as set of words. Intuitively, a set of items that appears in many baskets is said to be *frequent.* To be formal, we are going to assume a value s, called the *support threshold*, and a set of items I is going to be said frequent if its support has cardinality bigger than s.
The algorithm chosen to solve this problem is the *A-Priori algorithm*, which is based on the idea of *monotonicity of the itemsets.* It states that if a set I of items is frequent, then so is every subset of I.
So, at first, we only focus on the task of counting frequent pairs and in that case the *A-Priori algorithm* can be divided into two steps, which will indicate the two passes over the whole data.
In the first pass, we are going to create two different tables: the first one translates items names into integers from 1 to n, while the other is an array of counts, where the i-th element corresponds to the occurrences of item i. During the first pass, as we look at each item in the baskets, we translate its name to an integer and then increase the corresponding count.
Then, between the first and second pass, we examine the counts of the items to determine which of them are frequent as singletons. Finally, to conclude the preparation for the second pass, we create a new numbering system from 1 to m, just for the items whose count is above the threshold previously defined.
During the second pass we count all the pairs that consist of two frequent items: as the monotonicity of the itemsets says, a pair cannot be frequent unless both its members are frequent. The space required in the second pass is going to be $2m^2$ bytes if we use a triangular matrix for counting.
The mechanics for the second pass are as follows:

(1) for each basket, look in the frequent-items table to see which of its items are frequent
(2) in a double loop, generate all pairs of frequent items in that basket
(3) for each pair that verifies the conditions in point 2, add one to its count in the data structure used to store count

Finally, at the end of this pass we examine the structure used for keeping the occurrences of each couple, to determine which pairs are frequent.
In order to define the procedure for obtaining frequent sets of size $k + 1$ given all those of size less than k, we need to define two sets:

- C_k, the set of candidate itemsets of size k
- L_k, the set of truly frequent itemsets of size k

Given this two definitions, we can follow this procedure:

(1) define C_{k+1} as the set containing all those sets of terms of size $k + 1$, for which all the subsets of size k are in L_k
(2) find L_{k+1} by making a pass through all the baskets and counting all and only the itemsets of size $k + 1$ that are in C_{k+1}: those whose final count is going to be higher than the threshold t are going to end up in L_{k+1}

In the case studied, the items considered are going to be the words of the vocabulary and the baskets are the set of words contained in the tweets. Using this algorithm, we are going to end up with 26 different sets of frequent words that appear together, one for each of the day analyzed.
So, in the proposed algorithm for studying the consistent in time topics, this work is just the first part: after creating all the sets of the frequent sets of words, it will be necessary to check which are the frequent itemsets which appear in more days and finally check if they also appear in a sufficiently high number of longer

periods.

3 PROBLEM STATEMENT

We will now try to give a more formal representation of the problem studied. To do so, we need to define the following sets:

- W = set of words in the vocabulary
- D = set of days
- P = set of periods (each period is going to be a set of days)
- T = set of tweets

We might also notice that the set of tweets T is a subset of $\mathcal{P}(W)$, where $\mathcal{P}$ indicates the power set of W, the set of all subsets of W, including the empty set and W itself. Moreover, each element of D has to appear in exactly one element of P. This means that each day has to be assigned to exactly one period.

To describe formally the algorithm used to obtain the consistent sets throughout time in the set of tweets studied, we need to define four different functions.

The first one is very simple: it takes as input a tweet $t_i \in T$ and gives the day in which it was written:

$$\mathscr{D} : T \longrightarrow D$$
$$\mathscr{D}(t_i) = d_j$$

The second one takes as an input an itemset x ($x \subset W$) and a day $d_i \in D$ and gives back a natural number which corresponds to the cardinality of the set of tweets written in day d_i which contain x. More formally:

$$f_1 : \mathcal{P}(W) \times D \longrightarrow \mathbb{N}$$
$$f_1(x, d_i) = |\{t_j \in T \mid \mathscr{D}(t_j) = d_i, x \subset t_j\}|$$

With this function we will be able to define whether a set of words can be considered a frequent itemset for day d_i: it will happen if $f_1(x, d_i) \geq frequency\_threshold$.

The third function takes account of counting the number of days in which the itemset x is considered frequent. A formal definition of this function is the following:

$$f_2 : \mathcal{P}(W) \longrightarrow \mathbb{N}$$
$$f_2\,(x) = |\{d_i \in D \mid f_1(x, d_i) \geq frequency\_threshold\}|$$

Finally, the last function tells us the number of periods in which there exists at least a day when the itemset x can be considered frequent. It can be written as:

$$f_3 : \mathcal{P}(W) \longrightarrow \mathbb{N}$$
$$f_3(x) = |\{p_i \in P \mid \exists d_j \in p_i : f_1(x, d_j) \geq frequency\_threshold\}|$$

So, using the function described above a set of words, or to use a more specific term an itemset $x \subset W$, is going to be considered *consistent in time* if and only if the following conditions are verified:

(1) $f_2(x) \geq days\_frequent\_threshold$
(2) $f_3(x) \geq period\_consistent\_threshold$

Our final goal will be to find all those itemsets x that verify this two conditions, for suitable values of the thresholds used above: the final output of the algorithm is going to be the set of consistent itemsets C:

$$\{x \in \mathcal{P}(W) \mid \text{(1) and (2) are verified}\}$$

4 SOLUTION

As indicated in the previous section, the goal is to find those itemsets which are considered frequent in a subset of the days spread out throughout the whole time span considered. The set of days D is going to be $D = \{d_1 = 24/07/20, \ldots, d_{26} = 30/08/20\}$, as we are going to consider tweets written in those period. Can be noticed how there should be more days between the first and the last one (more precisely 38), however the dataset studied doesn't have tweets for certain days, which were consequently ignored.

The periods considered are going to be 6: as the number of days is not divisible by that number we won't have period of the same length. Moreover we also take account of the days which are skipped in the dataset. The division of the days into the periods is going to be the following:

p_1 : is going to contain the tweets written between the 24/07/20 and the 27/07/20

p_2 : is going to contain the tweets written between the 28/07/20 and the 31/07/20

p_3 : is going to contain the tweets written between the 1/08/20 and the 6/08/20, however there are no tweets written on the 3/08/20 and on the 5/08/20

p_4 : is going to contain the tweets written between the 7/08/20 and the 10/08/20

p_5 : is going to contain the tweets written between the 11/08/20 and the 14/08/20,

p_6 : is going to contain the tweets written between the 16/08/20 and the 18/08/20, and those written on the 22/08/20

p_7 : is going to contain the tweets written on the 29/08/20 and on the 30/08/20

The number of groups has been chosen in order to have sets of days of the same size, apart from the last one which contains days far from the others, and at the same time have sets of sufficiently high sizes. Clearly this choice is arbitrary and specific for the dataset considered in this example.

At first we are going to divide the dataset into 26 smaller sets: each one will contain all the tweets written in the corresponding day. On each of those daily datasets the *A-priori* algorithm will be applied and the result is going to be the corresponding set of frequent itemsets. At the end of this phase we are going to end up with a family of sets $\mathcal{F}$ which will contain the frequent itemsets for each of the days in the dataset. The number of elements in each frequent itemsets set is going to depend on the value of $frequency\_threshold$: in the last section the results for different values will be compared, however it is clear how a larger value of this threshold gives less results.

In this first phase we also create the set of the frequent itemsets in each period, by computing the union set of the corresponding elements of the family of frequent itemset sets.

The goal of the second phase is simply to find which itemsets appear at least $days\_frequent\_treshold$ times. To do so, one might simply count the occurrences of each itemsets in $\mathcal{F}$: indeed each element of $\mathcal{F}$ is a set, hence its elements are not repeated inside it, but at the same time there might be repeated sets of words when looking at different elements of the family of the frequent itemsets set.

Hopefully we are going to find some sets of words which are considered frequent in at least $days\_frequent\_treshold$ days analyzed: those are going to be stored in the new set of *candidate consistent itemsets.*

In last phase of the algorithm we need to find which of the elements in the *candidate consistent itemsets* set appear in at least *period_consistent_threshold* periods. We have to remember that in the first phase we also had defined, for each period, a set of the frequent itemsets for the days in that time span: basically we have created a set of the terms which are frequent itemsets at least once during a period.
One possible way to solve our problem, is simply counting the occurrences of each candidate consistent itemset on those sets and then save in the set of *consistent itemsets* those which appear at least *period_consistent_threshold* times.
So, after completing also this process we should have found all those sets of words which appear frequently together throughout time, following the definition given in the introduction.

5 IMPLEMENTATION

To run the code necessary to achieve these results, Python 3.8.5 was used.
Before going into the details of the implementation of the algorithm we need to transform the string that contains the words in each tweet into a set.
To solve the first phase, the one where we look for the frequent itemsets in each day, it was used the function `apriori` from package `efficient_apriori`, whose documentation can be found at https://pypi.org/project/efficient-apriori/.
This function required a list of tuples as an input. To do so, we created a list which contains, for each day, another list with the tuples containing the words of each tweet. To obtain a tuple starting from a set of words, the built-in function `tuple()` has been used.
As previously said, to obtain the frequent itemsets of each day, the function `apriori()` has been used. The input of this function contains, apart from the list of tuples, also a parameter called `min_support`, which indicates the frequency of which the items in the itemset have to appear together in order to be considered frequent. This parameter is going to be between 0 and 1, as it indicates a frequency, and corresponds to the value of $frequency\_threshold$ indicated in the Problem Statement section.
Each time the function `apriori` is applied, the output is a dictionary having as keys a set of natural number and, as value, new dictionaries. This last objects have, as key, the frequent itemsets of size equal to natural number which indicates this dictionary in the original one, and as values the corresponding number of occurrences in the tweets of the day considered. Since we have no interest nor in the frequent single words nor in the number of occurencies of each frequent itemset, we are going to keep only the sets of (multiple) words which are considered frequent by the A-Priori algorithm and save them in a new set. Then we can put each set, corresponding to each day, in a new list for all the sets of daily frequent itemsets, which in the code is going to be called `list_of_frequent_itemsets`.
During this first phase we also create, for each period of time, a set which is going to contain all the frequent itemsets in that particular time span: clearly, as we are talking about a set, there will be no repeated elements.
The second step, which as explained previously is counting the occurrences of each frequent set of words, can be achieved by creating a new dictionary, which will have as key all the frequent itemsets in the list described above and as value its number of occurrences throughout the whole list and across the different sets. After completing the count of occurrences, we can read once more the whole dictionary and save in a new list, called `candidate_consistent_itemsets`, only the keys (in this case sets of words) which have a value higher than *days_frequent_threshold*.
The last step which will give the list of consistent items, is handled with a single pass on the list of candidate consistent items: for each one is sufficient to count the number of occurrences in each of the sets of every period created before. If this number is higher than *period_consistent_threshold*, then the corresponding itemset is going to be added to the list `consistent_itemsets`, which is also going to be the final output.

6 DATASET

The dataset, which can be freely downloaded at https://www.kaggle.com/gpreda/covid19-tweets, had to be modified in order to suit the algorithms developed for this work.
In particular, some tweets contained special characters, such as *\t* and *\n*, which created several empty lines. Fortunately, by eliminating those characters it was possible to eliminate this issue and create a new modified version of the dataset containing one tweet (and all the information related) in each line.
Due to the specific nature of the analysis needed, many information contained in the dataset were eliminated. In particular, at the beginning it contained several information about the person who wrote each tweet (such as his user name, his description, the number of followers and others) which were discarded, as they do not provide useful information for our analysis. Also links or truncated words contained in the text of the tweet were eliminated.
In order to reduce the number of operations needed to perform the algorithm several stop words have been eliminated from the texts studied and for each word considered only the letters of the alphabet, the numbers and a few elements of punctuation have been mantained. By eliminating this terms, which include words such as *and*, *don't* or *you*, or even emojis, many operations in the search for the frequent itemsets have been eliminated and at the same time no relevant information has been lost.
Moreover, all the letters have been tranformed into their lowercase character, to eliminate issues regarding *case-sensitive* algorithms.
It has been noticed that the tweets in the dataset are truncated after 140 characters: this issue might limit the significance of the results, as some terms will be ignored by the algorithm.
Finally, for each tweet, have been considered its date, a set containing all the remaining words in the tweet and a set of the hashtags contained in the tweet.

7 EXPERIMENTAL EVALUATION

Before talking about the actual results obtained during the analysis, we need to describe the four parameters that determine the results. Three were already defined in the third section and these are

- *frequency_threshold*
- *days_frequent_threshold*
- *period_consistent_threshold*

There is also an additional parameter, a boolean, called *consider_hashtags*, which indicates whether we also have to add also the hashtags into the set of words to consider as baskets.
In the case studied a very high number of tweets was characterized by the hashtag *covid19* (either in lowercase or uppercase) and this translates into the results: we can see how including the hashtags the set of consistent itemsets is mainly made by sets which contain the mentioned term.
For example, by choosing as parameters

- *frequency_threshold* = 0.005
- *days_frequent_threshold* = 10
- *period_consistent_threshold* = 4

when considering the hashtags, the number of consistent itemsets is 133, while in the case with the boolean value set to *False* this quantity decreases to 31.
One can find in the file `005_10_4_hashtags.txt` all the consistent itemsets found with the parameter set to True, while the results obtained with `consider_hashtags = False` are found in `005_10_4.txt`. Both files are in the bin folder.
Due to this behaviour, which can be observed for all the values of the parameters, from now on the hashtags will not be considered in the analysis, as they do not provide any additional meaningful information and increase the execution time, as the frequent terms to consider would be much more.

The algorithm was tested using all possible triplets of the following values for the three parameters described before:

(1) *frequency_threshold* $\in \{0.003, 0.005, 0.01\}$
(2) *days_frequent_threshold* $\in \{4, 7, 10, 13\}$
(3) *period_consistent_threshold* $\in \{3, 5, 7\}$

The results of each attempt are summarized in the following tables, which contain the number of consistent itemsets through time for all the possible value of the parameter written above.
The first table contains the number of set for *frequency_threshold* = 0.003.
On the rows we have different values of *period_consistent_threshold* and on the columns the values of *days_frequent_threshold*

	4	7	10	13
3	1110	170	90	49
5	168	156	90	49
7	60	60	59	39

Table 1: *frequency_threshold* = 0.003

This table can be obtained also for the other values of *frequency_threshold*. The one for 0.005 is

	4	7	10	13
3	151	59	31	22
5	70	58	31	22
7	29	29	22	17

Table 2: *frequency_threshold* = 0.005

while the one for *frequency_threshold* = 0.01 is

	4	7	10	13
3	23	10	6	3
5	23	10	6	3
7	6	6	3	2

Table 3: *frequency_threshold* = 0.01

These first three tables give a first indication on the number of of consistent itemsets that one might find by running the algorithm presented before. As expected, a lower value of *frequency_threshold* gives back a much higher number of elements, while an higher values for *days_frequent_threshold* and *period_consistent_threshold* translate into a lower number of consistent itemsets. This is totally expected, as imposing that a frequent itemset has to appear in an higher number of periods reduces (and by a lot) the possible sets to consider. clearly, this also increases the execution time: due to the idea behind the *A-priori* algorithm, having an higher number of frequent sets of size k increases the number of operations necessary to retrieve the frequent sets of size $k + 1$.
It can also be observed that for fixed values of *days_frequent_threshold*, there is a big difference between the numerical results obtained when considering *period_consistent_threshold* 5 and 7, while the values are generally similar when comparing the output obtained when the same parameter equal is to 3 or 5. This is probably due to the nature of the division into periods of the dataset analyzed. In order to have a division that reflected the temporal dimension of the tweets, the last period is made only of a couple of days, while the others have 4. Moreover the last period is very distant in time to the others: if we exclude the 22^{nd} of August contained in the sixth period, the distance between the first day of the last period and the second to last day of the sixth time period, is of 11 days. This probably implies that the topics talked about in the last days might be slightly different from the ones of the rest of the days and this translates into the result described above.
We might also study the nature of the terms found by executing the algorithm for particular values of the parameters: can be observed that in all the cases, the majority of the itemsets contain terms that in some way are related to the pandemic situation, such as *cases*, or *deaths*. For example, looking at the list of consistent itemsets obtained using the parameters which give the lowest number of consistent itemsets (*frequency_threshold* = 0.01, *days_frequent_threshold*= 13 and *period_consistent_threshold* = 7) we get only the following result:

- ('cases', 'total')
- ('cases', 'deaths')

By analyzing a more complete list of numerous items, like the one obtained with *frequency_threshold* = 0.005, *days_frequent_threshold*= 10 and *period_consistent_threshold* = 5 which can be found in the file `005_10_5.txt`, we find sets like the following:

- ('cases', 'deaths', 'total')
- ('mask', 'wear')
- ('positive', 'tested')
- ('cases', 'today')
- ('cases', 'coronavirus')
- ('cases', 'deaths', 'reported'),
- ('august', 'cases')
- ('2020', 'august')

Apart from the last one, we can see how all the other consistent in time itemsets found are, somehow, related to the covid19 pandemic. This might be an indication that the set of tweets studied

contains many tweets regarding the pandemic situation that have characterized 2020.
We might repeat the same analysis on all the other sets obtained with different values of the parameters and look for the terms that we have obtained, however the results wouldn't change by much. By looking at all the other files contained in the `bin` folder, one for each possible triple if parameters, we can observe how the lists of consistent itemsets are very similar to each other.

We will now compare the method just introduced with a baseline model where we do not consider the periods, but rather just look for itemsets that appear as frequent at least a certain number of times among all days. This approach doesn't ensure us that the sets that appear are consistent, where with *consistency* we mean that the set is frequent in most of the periods which constitute the whole time span considered, however, we can observe that in most cases the terms which are found by this simplified algorithm are not much more then the one found with the approach described up to this point.
So, to use the formal approach described in the third section, the baseline model considers as consistent all those sets which satisfy only the following property:

$$f_2(x) \geq days\_frequent\_threshold$$

To show clearly how the number of such elements in many cases is not that different from the results obtained with the other algorithm, we can represent the same tables as before, but with an added row which represents the number of itemsets which are considered frequent in at least the number of days indicated by the value of the column. We might also notice that this approach is equivalent to asking that the sets considered are frequent in at least one period, which is always true as long as an itemset is frequent in at least one day. In the actual implementation of the baseline method we do not need to define the sets of frequent itemsets for each period, and hence the code is a bit faster.
The first table contains the number of set for *frequency_threshold* = 0.003.

	4	7	10	13
Baseline	**1120**	**171**	**90**	**49**
3	1100	170	90	49
5	168	156	90	49
7	60	60	59	39

Table 4: *frequency_threshold* = 0.003

The following table contains the quantities for *frequency_threshold* = 0.005

	4	7	10	13
Baseline	**153**	**59**	**31**	**22**
3	151	59	31	22
5	70	58	31	22
7	29	29	22	17

Table 5: *frequency_threshold* = 0.005

while the one for *frequency_threshold* = 0.01 is

	4	7	10	13
Baseline	**23**	**10**	**6**	**3**
3	23	10	6	3
5	23	10	6	3
7	6	6	3	2

Table 6: *frequency_threshold* = 0.01

From these three tables we can quickly see how the number of frequent itemsets obtained with the baseline method is in most cases very similar to the one obtained when considering 3 or even 5 periods, while it is significantly lower when considering only the sets which are frequent in at least one of the days of each period. This kind of behaviour is expected as it is quite unlikely that, for example, a set is considered frequent in more than 13 days (so, more than the half of the whole period) and, at the same time, those days are spread in less than 5 periods. Indeed, this would mean having a set of items which is frequent in at least 13 days spread across 16 or less days concentrated in just 4 periods.
It can also be observed that the number of elements obtained in the case without periods and the number obtained when considering elements which appear as frequent in at least three periods is always equal when *days_frequent_threshold* is either equal to 10 or 13. This is due to the fact that it is impossible to have that a set is frequent for 10 or 13 days in at most two periods, as they correspond, in the best case, to just 8 days.
On the other hand, it might happen that a topic is frequent in many days, but not in a couple of periods: in particular, due to the particular distribution of the days, which are not entirely consecutive, it might happen that the seventh period, which contains days far from the others, has slightly different frequent itemsets.
To conclude this part, we might say that the taking account of the periods (while imposing an high value of *period_consistent_threshold*) is a better choice than simply considering the itemsets which are frequent in at least a certain quantity of the days, if our goal is to look for those sets of words which are frequent over the whole time span studied. Indeed, if we impose *period_consistent_threshold* = 7, this method ensures that we are looking only at those sets of words which are frequent during the whole time frame and not just in a shorter period.
There is also a downside of considering the periods: in order to apply the algorithm described we need to save somewhere in the main memory a number of sets equal to the number of periods. These are going to contain all those terms which appear at least once among the frequent itemsets of the days in the period: this means that for large quantities of tweets or for long periods, each set might get quite big. Moreover, if we considered a dataset containing tweets spread out in a very long time span, we would have to define a huge number of periods and hence a corresponding number of sets of frequent itemsets. This might become a problem as these sets might even fill the whole memory available.
When taking into consideration the dataset studied in this case, this problem does not arise. Moreover, the number of operation caused by the additional operations in the original algorithm, which are looking for the number of occurrences of each member of the candidate itemsets set inside the sets of frequent itemsets of each period and creating these sets, are a negligible quantity. As we can see in the tables 4, 5 and 6, the number of candidate itemsets is quite small and for each element we simply need to check if it belongs to 7 sets. Creating the sets of frequent itemsets

for each period is not very expensive either, and the time taken by both algorithm is in general the same.
The situation might change in a different setting with a much bigger dataset that does not fit in the main memory. It can be observed that the two most time consuming parts of the execution of the program are creating the list of baskets to feed the `apriori` function and applying that function on the baskets of each day. Both this operations can be optimized by creating a distributed system, made of a number of nodes equal to the number of days in the dataset. Each node will then take as input the part of the dataset specific for that day and transform the text of the tweets into a list of tuples, as required by the function `apriori`. Then, finally, each node will compute the list of the frequent itemsets of the day assigned applying the A-priori algorithm or some other frequent itemsets algorithm.
At this point, the number of elements which can be considered *candidate consistent itemsets* is already very limited if the threshold for the frequency is sufficiently high.
To take account of the additional operations of the original algorithm, in a distributed environment it might be more convenient to not create the sets. Instead, after eliminating all those itemsets which are not frequent in a sufficiently high number of days, we might simply use a sufficiently high number of *or* conditions to understand if an itemsets is frequent at least once during a period. This operation could also be parallelized in order to save even more time.
We can then conclude that, even when considering bigger datasets, the baseline algorithm is a bit faster, however its results might not be as precise as the one found with the other algorithm.
Clearly, the choice of using a distributed system depends on the size of the dataset. In the case considered it is relatively small, as it contains less than 200000 tweets, and the use of a distributed system is not necessary as all the computations can be completed relatively fast even with a generic consumer laptop directly in the main memory.

Before concluding, there is one last observation that might influence the algorithm based on the idea of periods. Until now, we have always assumed a fixed number of periods (7) that divide the whole time span. However, this number was chosen in an arbitrary way, as it seemed the more suitable for the distribution of the day provided and in different situation it might be a good idea to choose other quantities if periods. For example, if one had all the tweets written during a whole year, it might be a good idea to define 12 periods (one for each month) or even 52 (one for each week). In particular, with 52 periods, we might be able to incorporate some weekly trends inside our periods: for example the fact that in the weekend many tweets might talk about typical (maybe not in the middle of a pandemic) weekend activities, like watching some sport live or going for an hike.

Finally, to conclude, we might say that, with the right parameters, the method described is definitely able to detect all and only those itemsets which appear frequently throughout the whole period analyzed. Indeed, in the case studied, setting *period_consistent_threshold* = 7, will certainly give as result only those itemsets which are considered frequent at least once in all the periods analyzed. This is clearly much better than simply looking for those sets of terms which are frequent in a large fraction of the days, as this last method doesn't guarantee the frequency throughout the whole period but only on a certain number of days (unless we ask for itemsets frequent every single day).
So, taking on consideration all these variables, it can be concluded that, depending on the characteristics of the dataset, the algorithm proposed might be a good idea for finding those sets of words which are frequent throughout the whole time span.