Identifying Twitter popular topics consistent in time

Gabriele Masina
University of Trento, 1st year CS master's degree
Matricola: 220496
gabriele.masina@studenti.unitn.it

ABSTRACT

Nowadays, social networks are observed with interest by companies and governments, since their role is more and more important in the society. In this paper we concentrate on Twitter trends, that are the most talked about topics on the social network in some period of time. We describe a method to identify popular consistent topics over time in tweets. This method is then evaluated and compared with some baseline method to identify its strengths and weaknesses.

1 INTRODUCTION AND MOTIVATION

Everyday, millions of people access online communities and social networks, producing a huge amount of data. This data, if properly analyzed, can be exploited in many different ways. In particular, in this work we focus on Twitter, a social network where the main content produced by users are tweets. Tweets are short texts, up to 280 characters, that are usually written by people to share with their followers something they think to be relevant. Tweets can contain, for instance, funny jokes, special personal events or facts and opinions about current news. The latter are particularly important, because they reflect a part of the public opinion on recent events. Therefore, they can be exploited, for example, by politicians to gain consensus or by social networks or other online platforms to recommend contents about the most trending topics to better engage their users. Another example is that by observing which concepts are frequently mentioned together, we could identify the spread of some fake news. Also, trending topics may be a good indicator of how a TV series has been welcomed by the public, and film companies can regulate their investments accordingly.

The purpose of this work is to identify which topics become popular many times over time.

We define a topic as a set of terms, that can be words, hashtags or another person's username. One could think that a trending topic is a topic that appears many times in the tweets of dataset, as in a typical market-basket scenario, where we have many baskets, in this case represented by tweets, each of which contains a small set of items, in this case represented by terms, and we are asked to identify the most frequent combinations of items appearing together in many baskets. However, in the context of Twitter, tweets have also associated a timestamp, that corresponds to the date and time the tweet has been posted.

Considering this, topics may be appear many times in some intervals of time, when they are trending, while being less frequent in others. Our objective is to identify popular consistent topics in time, i.e. topics that are frequent in many periods of time. In the next sections we will define more formally the problem and show a way to approach it. This procedure will then be evaluated and results will be analyzed.

2 RELATED WORK

In this chapter we introduce some concepts and techniques used to preprocess the dataset and to find popular consistent topics in the tweets. We also briefly introduce what other methods could be used to find popular topics.

2.1 TF-IDF

In several applications, such as the one treated in this paper, it can be useful to categorize documents by their topic. Typically, this is done by finding, for each document, the words that characterize it.

Term Frequency - Inverse Document Frequency (TF-IDF) is a numerical statistic that is intended to reflect how important a word is for a document in a collection. [6] This measure consists of a product of two components: TF and IDF.

Definition 2.1. The *Term Frequency* (*TF*) of a word i in a document j is defined as

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

where f_{ij} is the number of times i appears in j.

Definition 2.2. The *Inverse Document Frequency* (*IDF*) of a word i in a document j is defined as

$$IDF_{ij} = \log \frac{N}{n_i}$$

where N is the total number of documents and n_i is the number of documents the word i appears in.

Definition 2.3. The *Term Frequency - Inverse Document Frequency* (*TF-IDF*) of a word i in a document j is defined as

$$TF\text{-}IDF_{ij} = TF_{ij} * IDF_{ij}$$

As we discuss in Section 6, this technique can be used to preprocess the tweets in order to keep only relevant terms for each tweet. The objective is to obtain a more suitable dataset, excluding words that are used frequently but do not characterize the topic the tweets are about. We do this in order to obtain higher quality results and also to reduce the dimension of the input, so to speed up the execution time.

2.2 Finding Frequent Itemsets

A subtask of the problem consists in finding popular topics in a period of time. This task can be reduced to the more general scenario in which we are given a list of transactions, each containing a set of items. The problem to face is to find frequent itemsets, that are sets of items that appear together in many transactions, in order to find, for instance, association rules. The fraction of transactions in which an itemset i is present is called *support* of i, and an itemset is said to be frequent if its support is at least s, where s is a threshold set in advance. An algorithm that solves this task efficiently is A-Priori.

2.2.1 A-Priori. The algorithm was first introduced in 1994 by Agrawal R.S. and Srikant R. and it is a very fast algorithm to discover frequent itemsets in a list of transactions. [1]

The efficiency of the algorithm is based on the following observation: if an itemset is frequent, then all its subsets are frequent too. This property of monotonicity is exploited by the algorithm by seeing the implication in reverse way, that is itemsets whose subsets are not all frequent can not become frequent, so a great quantity of potential candidates is discarded.

The idea of A-Priori is shown in Algorithm 1.

Algorithm 1: A-Priori

Input : TRANSACTION T [ ]
FLOAT s

$k \leftarrow 1$
$C_k \leftarrow items(T)$
while $C_k \neq \emptyset$ **do**
  $L_k \leftarrow$ itemsets in C_k with a support $\geq s$
  $C_{k+1} \leftarrow$ itemsets of $k+1$ elements having all subsets of size k in L_k
  $k \leftarrow k+1$
return $\bigcup L_k$

At each iteration we have a set of candidate itemsets of size k and we find among them frequent itemsets of size k. Initially, frequent itemsets of size 1 are the sets containing a single item. At each iteration, we get the frequent itemsets of size k, which are the itemsets among the candidates with a support at least s. Then, we build the candidates of size $k+1$, by merging two frequent itemsets of size k and checking if all the subsets of size k of the new candidate are frequent. We repeat this process until the set of candidates is not empty.

Though A-Priori avoids counting many itemsets, there are numerous optimizations that make a better use of the main memory.

Among these, PCY, the Multistage and the Multihash algorithms try to further reduce the memory used to count frequent itemsets that are known an advance not to be candidates to become frequent. [4][2]

The SON algorithm applies a different heuristic, by dividing the file in chunks that can be stored in main memory to individuate the candidate frequent itemsets more rapidly. [7]

In this paper we focus just on the A-Priori algorithm, although the other algorithms could be used to deal with larger datasets.

3 PROBLEM STATEMENT

In this section, we formalize the problem introduced in Section 1, providing also some definitions to better explain the problem and the proposed solution.

First, we define some concepts used in the problem definition.

Definition 3.1. A *term* is a sequence of characters.

Definition 3.2. A *topic* is a set of at least two terms.

Definition 3.3. A *tweet* t is a pair $(time, tokens)$ where *time* is a timestamp that indicates the moment in time the tweet has been posted and *tokens* is a set of terms mentioned in the text of the tweet.

Definition 3.4. A topic q is *in* a tweet $t = (time, tokens)$ if $q \subseteq tokens$.

Definition 3.5. A *period of time* p is a pair $(start, end)$ where *start* and *end* are timestamps, such that $start \leq end$, indicating the start and the end time of p, respectively.

Definition 3.6. A tweet $t = (time, tokens)$ is *in* a period of time $p = (start, end)$ if $start \leq time < end$.

Definition 3.7. Let p be a period of time, T be a set of tweets and $s \in \mathbb{R}$ be such that $0 \leq s \leq 1$. Let $T' = \{t_1, \ldots, t_k\}$ be the set of tweets of T that are in p.

A topic q is *s-popular* in T in the period p if it is in at least $x = s * k$ tweets of T'.

Definition 3.8. Let $P = \{p_1, \ldots, p_h\}$ be a set of periods of time, $T = \{t_1, \ldots, t_k\}$ be a set of tweets and $s, r \in \mathbb{R}$ be such that $0 \leq s, r \leq 1$.

A topic q is *r-consistent-s-popular* in T in the periods P if it is s-popular in T in at least $y = r * h$ periods of time in P.

Input. As input of our problem we are given:

- The list of n tweets $T = [t_1, \ldots, t_n]$, where each tweet $t_i = (time_i, tokens_i)$;
- The two real numbers $s, r \in \mathbb{R}$ such that $0 \leq s, r \leq 1$;
- Two timestamps *start*, *end* and a number $a \in \mathbb{N}$ such that $a > 0$. The timestamps *start* and *end* indicate the start of the first period of time we consider and the end of the last one, respectively. The parameter a indicates the duration of each period of time, in seconds. The periods of time we consider are
  $P = \{(start, start + a), (start + a, start + 2a), \ldots, (start + ka, start + (k+1)a)\}$ such that $start + ka < end \leq start + (k+1)a$

Output. The output is the set of r-consistent-s-popular topics in T in the periods of time P.

4 SOLUTION

In this section, we present an algorithm to deal with the problem. Furthermore, we also describe some optimizations, whose improvement will be shown in Section 7 along with the evaluation results.

4.1 Proposed solution

The solution we propose is based on the A-Priori algorithm.

The general idea is that we first group the tweets by the period they belong to, and then, for each period, we find the popular topics in the period by exploiting the A-Priori algorithm. In this case, the transactions are represented by the tweets and the items of a transaction are the relative tokens. Consequently, the frequent itemsets returned by the algorithm are the popular topics of the period. So, we extract the popular topics in the period using the A-Priori algorithm, setting as threshold s.

By doing so, we build another dataset which will be the input for another instance of the A-Priori: in the new dataset, we have a transaction for each period of time and the items of a transaction are the popular topics in that period. As threshold for this second instance we use r.

This algorithm answer exactly the problem stated in Section 3, as it finds the topics which are s-popular in a fraction of periods of time greater than or equal to r, which is the definition of s-popular-r-consistent.

Actually, since in the second instance of the algorithm we only need to find itemsets of size 1, i.e. single topics, we only need to count in how many periods of time a topic is found to be frequent.

Thus, we only need to perform the first pass of A-Priori, without the need to build candidate pairs and so on. The complete process is shown in Algorithm 2.

Algorithm 2: Popular consistent topics

```
Input  : Tweet T[]
         Float s
         Float r
         Timestamp start
         Timestamp end
         Int a
groups ← groupByTime(T, start, end, a)
counter ← HashMap()
foreach group ∈ groups do
    frequentTopics ← APriori(group, s)
    foreach topic ∈ frequentTopics do
        counter[topic] ← counter[topic] + 1
ret ← ∅
foreach topic, count ∈ counter do
    if count ≥ r * |groups| then
        ret ← ret ∪ {topic}
return ret
```

The classical A-Priori algorithm is described in Section 2. However, if the dataset is not too big, we can exploit this fact to implement it more efficiently, by storing some additional data structures. We now proceed to describe these optimizations.

4.2 Optimizations

The idea is illustrated in Algorithm 3.

Algorithm 3: Efficient A-Priori

```
Input  : Transaction T[]
         Float s
k ← 1
C_k, indexTransactions, indexItem ← getItems(T)
result ← ∅
while C_k ≠ ∅ do
    L_k ← ∅
    foreach c ∈ C_k do
        support ← getSupport(c, indexTransactions)
        if support ≥ s * |T| then
            L_k ← L_k ∪ {c}
            if |c| > 1 then
                result ← result ∪ {c}
    C_{k+1} ← nextCandidates(L_k, k + 1)
    k ← k + 1
return mapItems(result, indexItem)
```

First, Procedure 4 is called. The procedure scans all the transactions and the items in them, building three data structures:

- *itemIndex, indexItem* used to map items to numbers and vice versa. In fact, the algorithm finds frequent itemsets of numbers corresponding to items, to be more efficient both in terms of time and memory. Once the itemsets of numbers are found, they are converted back to sets of items.
- *items* that is a set composed of the sets containing single items, i.e. the candidates of size 1.
- *indexTransactions* an array where the i-th entry is the set of the transactions containing the i-th item.

The rest of the algorithm is almost the same as the classical version, except for the computation of the support of an itemset in Procedure 5, which exploits the *indexTransactions* structure to speed up the computation. In fact, having in main memory the set of transactions containing each item allows us to get the support of the itemset fast by computing the size of the intersection of the transactions set of each item in the itemset.

Finally, once found the frequent itemsets are found, the algorithm maps back set of numbers to sets of items, as shown in Procedure 6.

Procedure 4: getItems

```
Input  : Transactions T[]
n ← 0
items ← ∅
indexTransactions ← []
indexItem ← []
itemIndex ← HashMap()
i ← 0
foreach t ∈ T do
    foreach item ∈ t do
        if item ∉ itemIndex then
            j ← n
            n ← n + 1
            items ← items ∪ {{j}}
            itemIndex[item] ← j
            indexItem.append(item)
            indexTransactions.append(∅)
        else
            j ← itemIndex[item]
        indexTransactions[j] ← indexTransactions[j] ∪ {i}
        i ← i + 1
return items, indexTransactions, indexItem
```

Procedure 5: getSupport

```
Input  : Set itemset
         Set indexTransactions[]
itemsetTrans ← ⋂_{i ∈ itemset} indexTransactions[i]
return |itemsetTrans|
```

Procedure 6: mapItems

```
Input  : Set frequentItemsets[]
         Item indexItem
result ← ∅
foreach itemset ∈ frequentItemsets do
    tmp ← ⋃_{i ∈ itemset} {indexItem[i]}
    result ← result ∪ {tmp}
return result
```

5 IMPLEMENTATION

The dataset manipulation and the algorithm have been developed and tested using Python 3.8.5, exploiting some useful tools offered by the language and the available libraries.

5.1 Dataset manipulation

As we explain in Section 6, the original dataset was precomputed to extract from each tweet the relevant terms by which it is composed. For reading and cleaning the dataset the following libraries were used:

- *Pandas* to read and write the csv;
- *HTML* to unescape html special characters;
- *Regex* to manipulate the text;
- *NLTK* to remove stop words, lemmatize words and split the text into tokens;

5.2 Algorithm implementation

The A-Priori algorithm was implemented from scratch in order to adapt it to the specific case and apply the optimizations discussed in Section 4. So, the only external library used was *Pandas*, which was used to read the csv and to group the tweets by period of time.

The complete code is available on Github. [3]

6 DATASET

The algorithm was tested over a dataset of 179 108 tweets, collected in the period from 24th July, 2020 to 30th August, 2020, a period in which the COVID-19 pandemic affected the life of everyone [5].

The original dataset is in csv format and for each tweet we are given much information, such as the date and time of publication in a timestamp format, the actual text, the hashtags and the indication if it was a retweet or not, along with information about the user who posted it.

The only fields useful to us are the date and time and the text. So, we created a new dataset, where for each tweet we kept two fields: the timestamp of publication, as it was originally given, and a set of tokens, extracted from the original text as follows:

(1) Unescape HTML sequences, such as '&' becomes '&' and transform the text to lowercase;
(2) Remove urls;
(3) Remove trunked words at the end of the text;
(4) Remove numbers. This decision was taken since the majority of numbers in tweets do not provide any relevant information about the topic treated and it is likely that the same number appears in many different tweets just by chance;
(5) Remove special symbols and emojis. In this process we remove also '#', that identifies an hashtag and '@' that identifies a person username. The purpose of it is that even if hashtags and usernames have a particular semantic in tweets, most of the times their meaning does not change if they are considered as simple words. Thus, we can count them together with their non-hashtag or non-username version to obtain better results;
(6) Use the same word to express some of the most common synonyms, such as 'new coronavirus', 'coronavirus' and 'covid' are all replaced by 'covid';
(7) Lemmatize words. Lemmatization is the process of reducing the different forms of a word to one single form, for example, reducing 'builds', 'building', and 'built' to the lemma 'build'. In this way, we count together words that are essentially referring to the exact same concept;
(8) Remove common stop words and single-letter words;
(9) Remove terms with a low TF-IDF. The threshold was chosen to remove terms with a score lower than the 10% of all terms scores. This process was done to keep for each tweet only the words that regard the topic discussed, removing some 'noise' caused by the most common words that appear in almost every tweet.
(10) The terms of a tweet are the remaining words, i.e. sequences of non-space characters.
(11) The tweets with no tokens left are discarded

7 EXPERIMENTAL EVALUATION

In this section we show the results obtained on the dataset analyzed. First, we examine the quality of the outputs through a user evaluation, then we compare them with the ones obtained with a baseline method. Lastly, we observe the execution time and discuss the scalability of the algorithm to bigger datasets.

7.1 User evaluation

Le us now show some of the results obtained, that are the consistent popular topics found by the algorithm described. Obviously, the quality and the quantity of the results depend also from the values of the parameters. The default values of the parameters have been set by hand by trying various alternatives and seeing which ones gave meaningful results. The values chosen are:

- $s = 0.01$
- $r = 0.10$
- $a = 86400$ seconds, that correspond to one day

These values have been then variate to observe how the number and the quality of results obtained vary consequently.

Tables 1, 2, 3 show the results obtained with different values for the parameters. It can be observed how as the values of the parameters vary, there are some topics that show up in each case and we can therefore say with a pretty good confidence that they are consistent popular topics over time. These are 'case new', 'case report', 'case total', 'hour last', 'mask wear' and 'positive test'. Probably, they are all related with news about the COVID-19 pandemic and it is likely they are consistent because every day most nations used to publish the daily report of the situation and thus the pandemic was certainly a much discussed topic over the period.

What can also be observed in Tables 1 and 2, is that as the values of r and s decrease, as one could expect the topics found by the algorithm are more and more. So, some interesting topics show up just with lower values. However, some topics are only found with lower values of s and some only with lower values of r, so the parameters must be tuned accurately to find best quality results.

A different matter is the a parameter, as can be seen in Table 3. In fact, the topics found with different values of a do not follow a monotonic behaviour as observed for r and s, so for instance 'case confirm' is only found when considering periods of 5 or 1 days, while it is not found when the periods considered are of 2 days or 12 hours. This is a particular observation that makes even more difficult to identify the right duration of periods.

Overall, we can state that the great majority of the results obtained are meaningful. This remark was also pointed out by the users to which we asked some opinion about the quality of

Table 1: Popular consistent topics identified, varying s

Itemset	1.25%	1.00%	0.75%	0.50%
case death	■	■	■	■
case india	■	■	■	■
case new	■	■	■	■
case report	■	■	■	■
case total	■	■	■	■
hour last	■	■	■	■
mask wear	■	■	■	■
positive test	■	■	■	■
case confirm		■	■	■
case last		■	■	■
active case			■	■
august case			■	■
case daily			■	■
case day			■	■
case hour			■	■
case hour last			■	■
case number			■	■
case spike			■	■
case tally			■	■
death india			■	■
death last			■	■
death report			■	■
help spread			■	■
new report			■	■
new total			■	■
american die				■
american realdonaldtrump				■
bring case				■
bring case total				■
bring total				■

The table shows the topics identified by the algorithm with different values of s. A cell is grey if the topic in the row has been identified by setting the parameter value corresponding to the column. Here we show the first 30 rows out of 152.

Table 2: Popular consistent topics identified, varying r

Itemset	12.5%	10.0%	7.5%	5.0%
case confirm	■	■	■	■
case death	■	■	■	■
case india	■	■	■	■
case last	■	■	■	■
case new	■	■	■	■
case report	■	■	■	■
case total	■	■	■	■
hour last	■	■	■	■
mask wear	■	■	■	■
positive test	■	■	■	■
case hour			■	■
case spike			■	■
case tally			■	■
able count				■
able count earlyvoting				■
able count earlyvoting eod				■
able count earlyvoting nov				■
able count earlyvoting person				■
able count eod				■
able count eod nov				■
able count eod nov person				■
able count eod nov willing				■
able count eod person				■
able count eod person willing				■
able count eod willing				■
able count nov				■
able count nov person				■
able count nov person willing				■
able count nov willing				■
able count person				■

The table shows the topics identified by the algorithm with different values of r. A cell is grey if the topic in the row has been identified by setting the parameter value corresponding to the column. Here we show the first 30 rows out of 147.

the results. Even if it would not be feasible to try to do the task by hand and compare the expected results with the actual ones, the major comment is that the topics found are in line with what they expected. Most of the topics are about the COVID-19, so for instance the ones that talk about face masks, cases, death and vaccines, but some others are about US electins and the Earth Overshoot Day (eod), which in 2020 was on 22nd of August. Thus, the ones found are certainly relevant, even though there might be some other interesting topics that were not identified at all.

7.2 Comparison with baseline method

We now compare the results obtained by our solution with a baseline method. As baseline, we chose to apply the A-Priori algorithm on the whole dataset, ignoring the notion of time. This result can be obtained by running the algorithm proposed and setting the duration of a period of time equal to the number of days between the oldest and the most recent tweet in the dataset.

Table 4 shows the topics identified by the proposed solution and by the baseline method. The parameters for our solution were set to their default values and for the baseline we set the minimum support to 1%, so that we compare topics found in the 1% of the tweets of the whole dataset with the topics found

Table 3: Popular consistent topics identified, varying a

Itemset	5 d	2 d	1 d	12 h
case death	■	■	■	■
case new	■	■	■	■
case report	■	■	■	■
case total	■	■	■	■
hour last	■	■	■	■
mask wear	■	■	■	■
positive test	■	■	■	■
case india	■	■	■	
case confirm	■		■	
case last		■	■	
first vaccine	■			
russia vaccine	■			

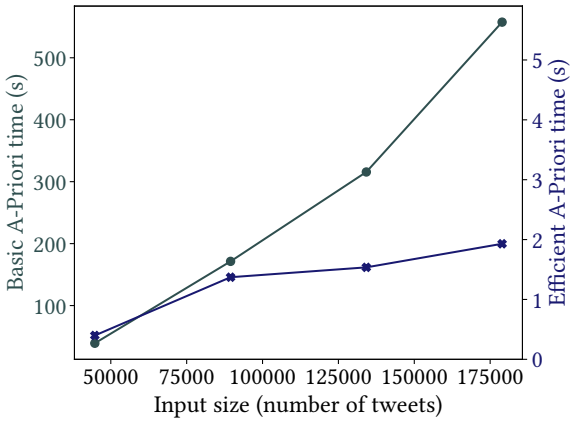
The table shows the topics identified by the algorithm with different values of a. A cell is grey if the topic in the row has been identified by setting the parameter value corresponding to the column.

Table 4: Comparison with baseline method

Itemset	Solution proposed	Baseline method
case new		
case total		
mask wear		
case confirm		
case death		
case india		
case last		
case report		
hour last		
positive test		

The table shows the topics identified by the solution proposed compared with the ones found by the baseline method. A cell is grey if the topic in the row has been identified by the method corresponding to the column.

Figure 1: Execution time with different input sizes



The plot shows how the input size is related to the execution time of two algorithms. The grey line corresponds to the time taken by the algorithm using a naive implementation of A-Priori, while the blue one represents the time taken by the more efficient implementation. The two lines are plotted with different scales, so the efficient A-Priori is almost 100 times faster than the naive one.

in the 1% of the tweets of at least 10% of the periods. It can be observed how the baseline identifies very few topics while our solution finds also topics that, even if they are not so frequent in the all dataset, are frequent in small periods of time and are thus meaningful. Similar observations can be made with different values for the parameters.

7.3 Scalability

Let us now discuss the execution time and the scalability of the proposed solution. Figure 1 shows the excecution time with different input sizes of the algorithm using a naive A-Priori implementation and using the efficient implementation described in Section 4. The times were obtained by averaging 20 executions for each input size, each with a different sample of the data.

The two lines are plotted with different scales, so the efficient implementation is roughly 100 times faster than the naive one with almost all inputs up to more than 200 times faster on the whole dataset, taking less than 2 seconds compared to more than 8 minutes on average.

On the other hand, it must be taken into consideration that the dataset analyzed was not too large and it can be stored in main memory, thus probably there is a threshold above which the time drastically increases, since the input does not fit in memory and it may be possible that the proposed optimization is not feasible. However, actually we do not need to store the whole dataset in main memory since we only analyze tweets of different periods of time independently. So, we could sort the tweets chronologically and read from the file just the tweets of the first period, find the frequent itemsets in it, then free the memory used to store the tweets, read the tweets of the second period and so on.

If the tweets in a period of time are still too much to be stored in main memory or the itemsets to be counted, it could be an option to use one of the optimizations of A-Priori listed in Section 2 to find the frequent itemsets.

8 CONCLUSIONS AND FUTURE WORK

In this paper we defined and developed a method to identify topics in tweets that are popular in different periods of time. We described the problem and the details of the solution proposed, along with some optimizations. Finally, we analyzed the results obtained both in terms of quality of topics and in terms of time performance.

The solution proved to give meaningful and consistent results, if the values of the parameters are tuned sensibly. The results in terms of performance were quite good, but it must be accounted that the dataset analyzed was not too large, so more test should be done in this direction. Finally, we discussed some possible further optimizations to deal with larger datasets.

In conclusion, we can say that the method developed is able to discover popular consistent topics in tweets. Further exploration could be done in defining alternative definitions of popular consistent topics to see the difference in terms of topics found.

REFERENCES

[1] R Srikant Agrawal and Ramakrishnan Srikant. 1994. R. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*. 487–499.

[2] Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, and Jeffrey D Ullman. 1999. Computing Iceberg Queries Efficiently.. In *Internaational Conference on Very Large Databases (VLDB'98), New York, August 1998*. Stanford InfoLab.

[3] Gabriele Masina. 2021. Popular Twitter Topics Mining. https://github.com/masinag/popular_twitter_topics_mining.

[4] Jong Soo Park, Ming-Syan Chen, and Philip S Yu. 1995. An effective hash-based algorithm for mining association rules. *Acm sigmod record* 24, 2 (1995), 175–186.

[5] G. Preda. 2020. COVID19 Tweets. Retrieved December 1, 2020 from https://www.kaggle.com/gpreda/covid19-tweets/version/24

[6] Anand Rajaraman and Jeffrey David Ullman. 2011. *Mining of massive datasets*. Cambridge University Press.

[7] Ashok Savasere, Edward Robert Omiecinski, and Shamkant B Navathe. 1995. *An efficient algorithm for mining association rules in large databases*. Technical Report. Georgia Institute of Technology.