Mining Locally Frequent Topics in Twitter Data

Ayushi Shah
University of Trento,

2^{nd} year, EIT Digital (AUS)
Trento, Italy
ayushianil.shah@unitn.it

Figure 1: photo:unsplash

1 INTRODUCTION

Twitter is a microblogging platform which enables us to write our thoughts and express ourselves in few words. This data is public and available in the form of tweets and retweets. As twitter is a user-base that covers people from all the educational as well as geographical backgrounds, there are millions of threads of conversations on different topics like health, sports, technology, business and many more. This data can be thought of as a gold-mine of information. But there is no use of it if we do not know how to mine the relevant data out of these conversations. One example can be online news media. Online news channels need to generate rich and timely information everyday which is possible through the twitter data. But, to find relevant information from the tweets and retweets will require large amount of efforts. There is a huge need to monitor and summarize this data. The concern is that the twitter data is a temporal data and thus the popular topics changes after a period of time. Thus we may find important topics which are popular for certain time intervals but not throughout the data set. We call here these popular topics as the frequent item sets which are locally frequent in the data set. Normally these locally frequent sets are periodic in nature. The aim of the project is to find these locally popular topics in the twitter data. It proposes a modification to Apriori algorithm to compute locally frequent sets.

2 RELATED WORK

Following are the concepts that will be used in the project:

Association rule learning

It is a rule-based machine learning method for discovering interesting relations between variables in large databases. The aim is to identify strong rules in the database using some measures of interestingness. For example: to identify items that are bought together by most of the customers, we can analyse the sales data to find dependencies among the items. Here one of the association rule for analysing data can be: If someone buys diaper and milk, then he/she is likely to buy beer. Thus we are looking for connections in the data items. It can be written as $\{i_1, i_2, ..., i_k\} \rightarrow j$. Which means if basket contains all of $i_1, i_2, ..., i_k$ then it is likely to contain j.

In practice there would be many rules but we need to find significant and interesting ones, for which contraints on various measures of significance and interest are used. The best-known constraints are minimum thresholds on support and confidence.

1. Support of itemset I: Number of items containing all items in I. It is also expressed as a function of total number of baskets

2. Confidence of the association rule is the probability of j given $I=i_1, i_2, ..., i_k$

$$conf(I \rightarrow j) = support \frac{I \cup j}{support(I)} \tag{1}$$

3. Interest of an association rule I $\rightarrow$ j is difference between its confidence and the fraction of the baskets that contain j. Interesting rules are those with high positive or negative interest values (usually above 0.5) $I=i_1, i_2, ..., i_k$

$$Interest(I \rightarrow j) = conf(I \rightarrow j) - P_r[j] \tag{2}$$

2.1 Apriori Algorithm

Given a set I, of items and a large collection D of transactions involving the items, the problem is to find relationships among the items. A transaction t is said to support an item if that item is present in t. A transaction t is said to support an itemset if t supports each of the items present in the itemset. An association rule is an expression of the form $X \Rightarrow Y$ where X and Y are subsets of the item set I. The rule holds with confidence τ if $\tau\%$ of the transaction in D that supports X also supports Y. The rule has support σ if $\sigma\%$ of the transactions supports $X \cup Y$. A method for the discovery of association rules was given in [1] is known as the *A-priori* Algorithm.

Steps for A-priori Algorithm:

- We begin with C1, which is all singleton itemsets, i.e., the set of all the unique items. Length of each item set being 1. This is the construction step 1, which is before taking a pass through the dataset.
- The first filter step is to count all items, and those whose counts are greater than or equal to the support threshold s form the set L1 of frequent items.
- Next is again a construction step for C2. The set C2 consists of candidate pairs. Here we take the combinations of all the itemsets from L1 (frequent itemsets of size-1) and form all the possible items sets of size-2.
- The second pass of the A-Priori Algorithm which is also the second filter step, counts all the candidate pairs and determines which appear at least s times. These pairs form L2, the frequent pairs.
- Similar construction and filter methods are used for obtaining the frequent itemsets of size-3. But for this project we will only need to generate the frequent itemsets of size-1 and size-2.

2.2 Text Preprocessing Techniques

We need to convert text from human language to the machine readable format for further applying data mining algorithms, this is called as text pre-processing. Following are the Natural language processing terms which are extremely useful for the process of text preprocessing.

1. Tokenization

Tokenization in text mining is the process of chopping up a given stream of text or character sequence into words, phrases, symbols, or other meaningful elements called tokens which are grouped together as a semantic unit and used as input for further processing such as parsing or text mining.

2. Stop Word Removal

Sometimes a very common word, which would appear to be of little significance in helping to select documents matching user's need, is completely excluded from the vocabulary. These words are called "stop words" and the technique is called "stop word removal". Some of the examples of stop-word are: a, an, the, and, are, as, at, be, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with etc.

3. Lemmatization

Lemmatization (or lemmatisation) in linguistics, is the process of reducing the the derived forms of a word to its base form so that they can be analysed as a single term. For instance: If operated on a token "saw", lemmatization will go through the morphological analysis and return see or saw depending upon the use of word "saw" as a verb or noun in the sentence.

3 PROBLEM STATEMENT

Before defining the problem statement, lets get familiar with the list of parameters, their notations and definitions which are required for framing the problem statement. This section describes the parameters with reference to market-basket model as it is familiar and easy to understand concept. Also, analogy with the twitter data is explained.

- Let T = <*to, t1,*.....> be a sequence of time stamps where i<j, then $ti < tj$, which means ti denotes a time which is earlier than tj.
- Let I be a finite set of items. We can also look at it as a set of unique words.
- Let D be the transaction database , which is a collection of transactions where each transaction has a part which is a subset of the item set I and the other part is a time-stamp indicating the time in which the transaction had taken place. In the case of the project, each transaction can be each tweet and its corresponding time-stamp. We make an assumption that the database is sorted in the ascending order of the date and time in the time-stamp.
- Next parameter is time intervals. Here we consider closed intervals of the form [*t1, t2*] where *t1* and *t2* are time

stamps. We can say that a transaction is in the time interval $[t1, t2]$ if the time stamp of the current transaction t occurs in the closed interval such that $t1 \leq t \leq t2$. We define the local support of an item set in a time interval $[t1, t2]$ as the ratio of the number of transactions in the time interval $[t1,t2]$ containing the item set to the total number of transactions in $[t1, t2]$ for the whole database D.

- Another important parameter is the *Support*. We use the notation $Sup_{[t1,t2]}(X)$ to denote the support of the itemset X in the time interval $[t1, t2]$, which can also be referred as local support. Given a threshold σ we say that an itemset X is frequent in $[t1,t2]$ if $Sup_{[t1,t2]}(X) \geq (\frac{\sigma}{100}) * tc$ where tc denotes the total number of transactions in D that are in the time interval $[t1,t2]$.
- Last parameter is time-span in days. It is the minimum threshold that defines a minimum period length for the word to be locally frequent. It is denoted by *minthd2*
- Output parameter *L1* is the map of singleton frequent item sets and its corresponding time range list.
  L1 = {*I1: t[I1]*}
- Output parameter *L2* is the map of frequent item sets with 2 items and its corresponding time range list.
  L2 = {*I2: t[I2]*}

Formal Problem Statement
Problem: Find Locally frequent topics in the text of twitter data set
Input $\Rightarrow$ (D, σ, *minthd2*)
Output $\Rightarrow$ *L1*, *L2*

4 DATA SET

For the project the COVID19 Tweets data set available on Kaggle is being used. This data set contains the tweets collected using Twitter API and a python script. It is collected by running a query for high-frequency hashtag (#covid19) on a daily basis from the date 24-07-2020 to 30-08-2020. Thus we have a huge data from all around the globe. Link for the dataset is here:
(https://www.kaggle.com/gpreda/covid19-tweets)
In this step of the project cleaning if data is performed. All the text preprocessing steps mentioned in the section 2 are performed on the huge data. After preprocessing we have got a new data set with 3 feature vectors, 1. date, 2. text_clean and 3. hashtags.
1. date: This feature gives information about data and time at which the particular tweet was queried.
2. text_clean: This is a feature of type array. After cleaning the data we have an array of relevant words for reach tweet. Just like the market-basket model where each record is items in each basket, here each record is described as words in each tweet.
3. hashtags: This is a feature of type array again, which is used to store the hashtags used in ever tweet.

5 SOLUTION

This section describes the algorithm used in the project. The algorithm is used in [2] for finding locally frequent item sets is used.

5.1 Concept of tracking time-interval for locally frequent itemsets

Generating locally frequent item sets requires to maintain a list of time-intervals in which the word is used frequently. Here we use 2 thresholds *minthd1* and *minthd2* to be able to maintain the time interval list.

minthd1: while making a pass through the database, if for a particular item set the time difference between its current timestamp and the time when it was last seen is less than the value of *minthd1* then the current transaction is included in the current time-interval under consideration. Otherwise a new time-interval is started with the starting timestamp equal to the current timestamp. Then the support of the item set in the previous time interval is checked, if it is frequent then the item set is added to the list which is maintained for that particular item set.
minthd2: There is another time threshold which is given by user as input. The locally frequent sets with minimum period length greater than or equal to this value are kept. If we do not give this threshold then an item or word appearing once in the whole database will also become locally frequent.

5.2 Computing L1: A set of locally frequent items sets of size 1

Variables for time-stamp tracking:

- *tm*: In a database every transaction(vector of words in the text of tweets) is accociated with a time stamp and while passing through the database the the timestamp of the current transaction is denoted by *tm*.
- *firstseen*: Corresponds to the time when the item/ word was first seen in the database.
- *lastseen*: Corresponds to time when the item/word was last seen. When an item is found in a transaction and the time difference between *lastseen* and *tm* is greater than the minimum threshold given, then a new time interval is started by setting *start* of the new time interval as *tm* and *end* of the previous time interval as *lastseen*.The previous time interval is added to the list of time intervals for that item if the support of the item in that time interval is greater than or equal to the minimum local threshold provided by the user. Otherwise *lastseen* is set to *tm*, the counters maintained for counting transactions are increased and the process is continues.

Counter Variables: There are 3 counts maintained for each item which are *icount*, *ctcount*, *ptcount*. All the counts are initialized to 1 when an item is first seen.

- *icount*: For each item, while making a pass through the dataset when a transaction containing the item is found then *icount* for that item is incremented by 1. Next we need to see whether an item is frequent in a time interval. For this total transactions in that interval will be needed to be counted.
- *ctcount*:The value of *ctcount* increases with each transaction
- *ptcount*: The value of *ptcount* changes only when a transaction containing an item is found within *minthd1* from current value of *lastseen* and then it takes the value of *ctcount*. When an item is not seen for more than *minthd1* time distance from *lastseen* then the value of *ptcount* is used to compute the percentage *support* of the item between *firstseen* and lastseen. If the *support* percentage of an item in a time interval is greater than the minimum local support then only the set is considered as a locally frequent set and the locality is the time interval. When a new interval is started for an item then the three counts

again start from 1. Algorithm 1 describes the procedure to compute L1.

Algorithm 1: Locally Frequent Itemsets of Size-1

Input: D, σ,local_support, minthd1, minthd2
Output: L1
$C_1 = \{(i_k, t_p[k]) : k = 1, 2, \ldots, n\}$ where i_k is the k-th item and $t_p[k]$ points to a list of time intervals initially empty
for *k = 1 to n* **do**
    set lastseen[k], icount[k] ctcount[k] and ptcount[k] to zero
for *each transaction t in the database with time stamp tm* **do**
    for *k = 1 to n* **do**
        if $i_k \subseteq t$ **then**
            if *lastseen[k] == 0* **then**
                lastseen[k] = firstseen[k] = tm
                icount[k] = ptcount[k]=ctcount[k] =1
            else if *(|tm − lastseen[k]| < minthd1* **then**
                lastseen[k]=tm itemcount[k]++
                ctcount[k]++ ptcount[k]=ctcount[k]
            else
                if *((|lastseen[k] − firstseen[k]| ≥ minthd2) && (icount[k]/ptcount[k]*100 ≥ σ s))* **then**
                    add(firstseen[k], lastseen[k]) to tp[k]
                    icount[k] = ptcount[k] = ctcount[k]= 1
                    lastseen[k] = firstseen[k] = tm
        else
            ctcount[k]++
    for *k = 1 to n* **do**
        if *((|lastseen[k] − firstseen[k]| ≥ minthd2) && (icount[k]/ptcount[k]*100 ≥ σ s))* **then**
            add (firstseen[k], lastseen[k]) to tp[k]
        if($t_p[k]$! = 0) add $(i_k, t_p[k])$ to L1

5.3 Computing L2: A set of locally frequent items sets of size 2

We have frequent itemsets of size-1, next step is to generate locally frequent itemsets of size-2. For this A-priori candidate generation algorithm is used to find candidate frequent sets of size-2 and then pruning is applied. Again here With each candidate frequent set of size-2 we associate a list of time intervals. IN the beginning of the algorithm we simply take the frequent item sets of size-1 and generate all the possible itemsets of size-2 irrespective of its time interval. we can say refer this list as C_2. In the pruning phase we generate the list time intervals associated with the itemsets in C_2. Now on in this section, by 'itemsets' we refer to the itemsets of C_2,in which each itemset contains 2 items. The procedure of generation is that we take the time-intervals associated with each of the two elements in the itemset and a pair-wise intersection of the interval lists are taken. If in an interval say [*t, t'*] an itemset say A,B is frequent then there exits two time periods [*t1, t1'*] and [*t2, t2'*] in which the itemsets A and B are respectively frequent and [*t, t'*] ⊆ [t1, t1'] ∩ [t2, t2']. Using this concept there is a modification made in the Apriori Algorithm. Algorithm Algorithm 2 explains the procedure for generating L2. Modification of Apriori algorithm is in its prune function, which is described in Algorithm 3.

Algorithm 2: Locally Frequent Itemsets of Size-2

Input: D, σ,local_support, minthd1, minthd2
Output: L2
$C_2 = \{(i_{k2}, t_{p2}[k]) : k = 1, 2, \ldots, n\}$ where i_{k2} is the k-th item of size 2 and $t_{p2}[k]$ points to a list of time intervals initially empty
C_2=apriori_gen(L1, minthd2)
L2=level_two_frequent(C_2, local_support, minthd1, minthd2) compute L2
`// L2 can be computed from C_k using the same procedure as mentioned for L1`

Algorithm 3: Apriori generation algorithm

Input: L1, minthd2
Output: time_interval for each itemset of size-2
possible_item_sets= join_step(itemsets, 2)
`// All possible combinations of frequently occurring singleton words`
for *each itemset in possible_item_sets* **do**
    // Pruning step
    time_interval[itemset]=intersection_of(time_interval[item[0]] and time_interval[item[1]])
    if *time_interval[itemset] is empty* **then**
        remove itemset from list of possible itemsets

6 IMPLEMENTATION

All the algorithms mentioned above are implemented in a very simple way in the language Python. It is a naive implementation which is implemented using Jupyter notebooks as it provides the benefit of being powerful and versatile. Implementing code bit by bit and being able to debug it step by step is very important and with jupyter notebooks we can do that easily. After the implementation the code was converted to python script.
1.As the project uses text data, the part one of the project is about cleaning the data. Here we use natural language processing libraries provided by python. Libraries used are:

- NLTK (Natural Language Toolkit):
  nltk.download('stopwords')
  nltk.download('wordnet')
  nltk.tokenize: TweetTokenizer is used to tokenize the data
- tweet-preprocessor: It is a preprocessing library for tweet data. It is used for cleaning the data of any unwanted symbol or emoticons.
- Ekphrasis: It is a Text processing tool, which is specially used for text from social networks, such as Twitter or Facebook. Ekphrasis performs tokenization, word normalization, word segmentation (for splitting hashtags) and spell correction. Here in the project we use it for the purpose of segmention of hashtags in text.
- Re module: For regular expression matching operations.

2. Next part of the project is about algorithm implementation. Following libraries are used for the same:

Pandas, itertools, datetime and DateTimeRange (to perform operations on the variables of type datetime)
3. Last part is about data visualization: In the data visualization part We are creating wordcloud for the frequently occurring singleton words in the data set. For this we are using the following libraries: matplotlib and wordcloud

7 RESULTS AND ANALYSIS

This section describes the results obtained in various experiments and their analysis. We know that to get best results we need to make multiple runs through the script by passing different parameter values. Here the main parameters that we change every time during the run are 'local_support', 'minthd1','minthd2'. We have made 4-5 runs through the whole data set and recieved 2 results which can be on the basis of which a clear comparison of outputs on different parameter values can be made. Also these results will help us analyse the properties of the dataset. In the following sections we describe Run 1 and Run 4:

7.1 Run 1 and Run 4 comparision

Parameters Run 1:
1. 'local_support'→ 2%
2. 'minthd1'→ 2
3. 'minthd2'→ 3
Parameters Run 4:
1. 'local_support'→ 1%
2. 'minthd1'→ 2
3. 'minthd2'→ 4

Following are the observations from the outputs:

- Figure 2 and Figure 5 shows the results of Run 1 and 4 respectively. The whole outputs are in the files whole_data1.csv and whole_data4.csv in the output folder. The first thing we observe from both the outputs is that we get frequent itemsets in the periodic date intervals with varying times. There are 2 main identified intervals '2020-07-24 to 2020-08-04' and '2020-08-06 to 2020-08-18' each consisting a gap of 12 days. So we can say that the algorithm identifies frequent itemsets in the periodic intervals.
- Let us consider outputs in the time interval 2020-07-24 -2020-08-04 for both the runs. We find that the number of frequent itemsets of size-1 are much more than the number of frequent itemsets of size-2. What could be the possible reason for this? If we look at the data set we will find that there are thousands of transactions for a single date with varying times. So there is a high chance that 2 items each frequent and having support greater than the local_support, also they might appear for same number of times in a given date intervals, but at different time intervals. Bye same date and different time intervals, we mean different transactions on the same day. For example: Let us consider items "people" and "pandemic", each of these two words are frequent in the same date interval, yet do not show up in the output of L2. The reason being these two items do not appear in the same transactions often for them to be considered as frequent itemsets of size-2.
- We observe that local_support parameter for Run 1 was 2% and for Run 4 was 1%. What can be the possible effect of this? Let us consider the same date interval. Figure 3 and Figure 4 shows the wordclouds of Run1 and 4 respectively. These wordclouds show all the singleton items that are frequent in the date interval. The words are ranked according to their support values in that date interval. We notice that there are many more words in the wordcloud for Run 4 as compared to wordcloud for Run 2. This happened because of low value of local_support for Run 4.
- If we look at the Figure 4 showing the wordclouds of Run 4. We will find many words that are not at all relevant. For example: 'every', 'even','dont' and many more. These words have very less support in the data set. When we set our local_support extremely small then we get the unwanted outputs. Also counter to this situation if we increase support to 7 or 8 we will miss out on the correct results and end up getting a very small set of frequent itemsets. So it is very important to choose appropriate support values. After performing various runs by changing different values of support, it eas observed that the algorithm wored just fine with local_support= 2.
- On the large datasets the algorithm takes huge amount execution time and thus needs improvisations like use of Map Reduce.

Items	Time-interval	Support (%)
case	Timestamp(2020-07-24 23:47:00), Timestamp(2020-08-04 07:56:00), Timestamp(2020-08-06 14:48:00), Timestamp(2020-08-18 18:27:00)	9.40, 9.17
pandemic	Timestamp(2020-07-24 23:47:00), Timestamp(2020-08-04 07:56:00), Timestamp(2020-08-06 14:48:00), Timestamp(2020-08-18 18:27:00)	4.45, 4.73
state	Timestamp(2020-07-24 23:51:00), Timestamp(2020-08-04 07:56:00)	2.21
july	Timestamp(2020-07-24 23:48:00), Timestamp(2020-08-04 07:53:00)	2.05
positive	Timestamp(2020-07-24 23:49:00), Timestamp(2020-08-04 07:56:00), Timestamp(2020-08-06 14:51:00), Timestamp(2020-08-18 18:26:00)	3.10, 2.30
august	Timestamp(2020-08-06 14:48:00), Timestamp(2020-08-18 18:27:00)	2.20
today	Timestamp(2020-07-24 23:49:00), Timestamp(2020-08-04 07:56:00), Timestamp(2020-08-06 14:48:00), Timestamp(2020-08-18 18:26:00)	2.45, 2.72
need	Timestamp(2020-07-24 23:48:00), Timestamp(2020-08-04 07:55:00), Timestamp(2020-08-06 14:48:00), Timestamp(2020-08-18 18:27:00)	2.77, 2.65
(coronavirus, covid19)	Timestamp(2020-07-24 23:47:00), Timestamp(2020-08-04 07:57:00), Timestamp(2020-08-06 14:48:00), Timestamp(2020-08-18 18:26:00)	5.40, 5.15
(covid, covid19)	Timestamp(2020-07-24 23:47:00), Timestamp(2020-08-04 07:54:00), Timestamp(2020-08-06 14:48:00), Timestamp(2020-08-18 18:26:00)	3.64, 3.53
(covid19, case)	Timestamp(2020-07-24 23:47:00), Timestamp(2020-08-04 07:56:00), Timestamp(2020-08-06 14:48:00), Timestamp(2020-08-18 18:27:00)	7.05, 7.03
(covid19, death)	Timestamp(2020-07-24 23:48:00), Timestamp(2020-08-04 07:54:00), Timestamp(2020-08-06 14:48:00), Timestamp(2020-08-18 18:27:00)	3.56, 3.62
(covid19, pandemic)	Timestamp(2020-07-24 23:47:00), Timestamp(2020-08-04 07:54:00), Timestamp(2020-08-06 14:48:00), Timestamp(2020-08-18 18:27:00)	2.85, 3.01

Figure 2: Output Run 1

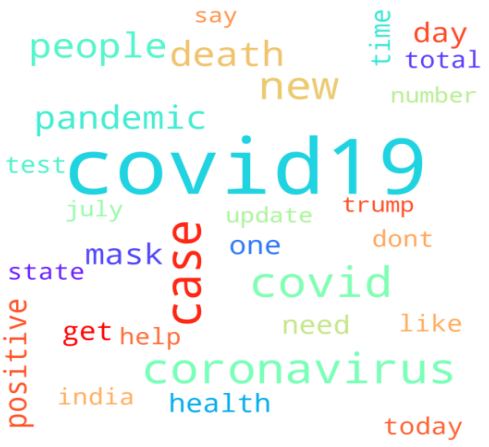

Figure 3: Wordcloud for singleton itemsets for the date interval '2020-07-24 -2020-08-04' for Run:1

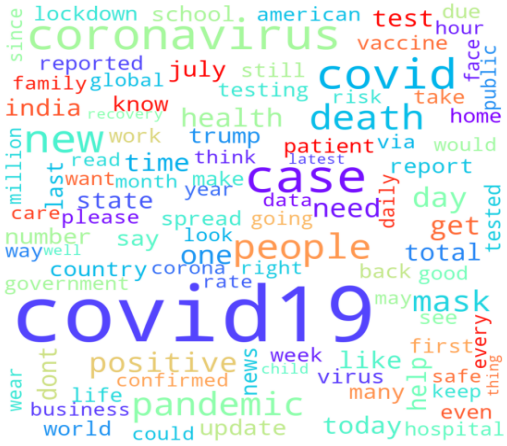

Figure 4: Wordcloud for singleton itemsets for the date interval '2020-07-24 -2020-08-04' for Run:4

Items	Time-interval	Support %
(case, covid)	Timestamp(2020-07-24 23:47:00), Timestamp(2020-08-04 07:46:00), Timestamp(2020-08-06 14:50:00), Timestamp(2020-08-18 18:25:00)	1.25, 1.20
(case, covid19)	Timestamp(2020-07-24 23:47:00), Timestamp(2020-08-04 07:56:00), Timestamp(2020-08-06 14:48:00), Timestamp(2020-08-18 18:27:00)	7.05, 7.03
(case, india)	Timestamp(2020-07-25 00:13:00), Timestamp(2020-07-29 16:08:00)	1.57
(case, new)	Timestamp(2020-07-24 23:49:00), Timestamp(2020-08-04 07:56:00), Timestamp(2020-08-06 14:48:00), Timestamp(2020-08-18 18:26:00)	3.37, 3.49
(case, positive)	Timestamp(2020-07-24 23:52:00), Timestamp(2020-08-04 07:56:00)	1.01
(case, reported)	Timestamp(2020-07-25 00:01:00), Timestamp(2020-08-04 07:54:00), Timestamp(2020-08-06 14:48:00), Timestamp(2020-08-18 18:26:00)	1.12, 1.37
(covid, coronavirus)	Timestamp(2020-07-24 23:54:00), Timestamp(2020-08-04 07:48:00), Timestamp(2020-08-06 14:51:00), Timestamp(2020-08-18 18:25:00)	1.25, 1.24
(covid, covid19)	Timestamp(2020-07-24 23:47:00), Timestamp(2020-08-04 07:54:00), Timestamp(2020-08-06 14:48:00), Timestamp(2020-08-18 18:26:00)	3.64, 3.53
(covid19, coronavirus)	Timestamp(2020-07-24 23:47:00), Timestamp(2020-08-04 07:57:00), Timestamp(2020-08-06 14:48:00), Timestamp(2020-08-18 18:26:00)	5.40, 5.15

Figure 5: Output showing itemsets of size-2 for Run-4

8 EXPERIMENTAL EVALUATION

This section describes the experimental evaluation of the algorithm used in the project. A new data set is created for this part with known parameters and we run our script on the new data set to test the working of the project.

8.1 Data Set

8.1.1 Evaluation Phase 1. For this evaluation phase we create a small data set of about 319 records. It is the file named 'custom_data_set.csv' which is stored in the data folder of the project. This data set can be referred as the retail data set where we have 9 unique items: {'bread', 'wine', 'eggs', 'meat', 'cheese', 'pencil', 'diaper', 'bagel', 'milk'}. Various combinations of these items are the transactions in the database. Each transaction has a time stamp associated with it. Thus we can say that the data set is temporal. It can be described as the transactions taking place at a retail store every day and we are keeping the track of items as well as the day and time of the purchase. It contains transactions from the date "25-07-2020" to "22-08-2020" sorted in the ascending order of the date. The data set is created such that we know the temporal association rules and support of each item already. Following are the rules:

{bread}→{wine}⇒'25-07-2020'-'29-07-2020'; local_support>50%
{eggs}→{meat}⇒'30-07-2020'-'03-08-2020'; local_support>50%
{bread}→{meat}⇒'04-08-2020'-'08-08-2020'; local_support>50%
{cheese}→{pencil}⇒'9-08-2020'-'13-08-2020'; local_support>50%
{diaper}→{bagel}⇒'14-07-2020'-'18-08-2020'; local_support>50%
{cheese}→{bagel}⇒'19-08-2020'-'22-08-2020'; local_support>50%
This data set was created for exact evaluation of the algorithm.

8.1.2 Evaluation Phase 2. for this phase of evaluation a random data set with the same 9 unique items mentioned in the above section was created. Every transaction had random combinations of the 9 unique items and its associated timestamp. This data set was created by a student studying at Gerogia Tech in the US.

8.2 Results

8.2.1 Evaluation phase 1. In the phase one of the evaluation extremely accurate results were received with approximately 100% accuracy with high speed of execution. After several experiments the best parameters were as follows:
local_support=20
minthd1=2
minthd2=3
Figure 4 shows the output of this evaluation phase

Items	Time-interval	Support (%)
diaper	Timestamp('2020-08-14 12:26:00'), Timestamp('2020-08-18 12:26:00')	81.13'
wine	Timestamp('2020-07-25 12:27:00'), Timestamp('2020-07-29 12:26:00')	58.18'
pencil	Timestamp('2020-08-09 12:26:00'), Timestamp('2020-08-13 12:26:00')	81.48'
meat	Timestamp('2020-07-30 12:26:00'), Timestamp('2020-08-08 12:26:00')	77.98'
bread	Timestamp('2020-07-25 12:27:00'), Timestamp('2020-07-29 12:26:00'), Timestamp('2020-08-04 12:26:00'), Timestamp('2020-08-08 12:26:00')	58.18', '79.25'
bagel	Timestamp('2020-08-14 12:26:00'), Timestamp('2020-08-22 12:26:00')	63.16'
eggs	Timestamp('2020-07-30 12:26:00'), Timestamp('2020-08-03 12:26:00')	78.18'
cheese	Timestamp('2020-08-09 12:26:00'), Timestamp('2020-08-13 12:26:00'), Timestamp('2020-08-19 12:26:00'), Timestamp('2020-08-22 12:26:00')	81.48', '41.46'
('bagel', 'cheese')	Timestamp('2020-08-19 12:26:00'), Timestamp('2020-08-22 12:26:00')	41.46'
('diaper', 'bagel')	Timestamp('2020-08-14 12:26:00'), Timestamp('2020-08-18 12:26:00')	81.13'
('meat', 'bread')	Timestamp('2020-08-04 12:26:00'), Timestamp('2020-08-08 12:26:00')	79.25'
('meat', 'eggs')	Timestamp('2020-07-30 12:26:00'), Timestamp('2020-08-03 12:26:00')	78.18'
('pencil', 'cheese')	Timestamp('2020-08-09 12:26:00'), Timestamp('2020-08-13 12:26:00')	81.48'
('wine', 'bread')	Timestamp('2020-07-25 12:27:00'), Timestamp('2020-07-29 12:26:00')	58.18'

Figure 6: Outputs of evaluation phase 1

8.2.2 Evaluation phase 2. In this phase of evaluation the results were verified by the same student from the Gerogia tech, US, the one mentioned in the Data Set section. Again we got the accuracy of approximately 100% with high speed of execution Parameters for the best results were same as mentioned in the above section. Figure 7 shows the results of evaluation phase 2, we can see clearly identified time intervals for various itemsets of size-2. So, the algorithm works perfectly to generate locally frequent itemsets and also it gives its specific time intervals. The advantage being the evaluation dataset is small in size which helped to better judge the algorithm by manually checking it.

9 CONCLUSION

We come to the end of the report and there are various points that can be concluded from the project

Items	Time-interval	Support %
(Cheese, Milk)	Timestamp(2020-07-25 12:27:00), Timestamp(2020-08-03 12:26:00), Timestamp(2020-08-05 12:26:00), Timestamp(2020-08-21 12:26:00)	36.70, 30.27
(Cheese, Wine)	Timestamp(2020-07-25 12:27:00), Timestamp(2020-08-22 12:26:00)	27.16
(Diaper, Bagel)	Timestamp(2020-07-25 12:27:00), Timestamp(2020-07-31 12:26:00)	25.00
(Diaper, Bread)	Timestamp(2020-07-25 12:27:00), Timestamp(2020-07-31 12:26:00), Timestamp(2020-08-09 12:26:00), Timestamp(2020-08-18 12:26:00)	32.47, 35.29
(Diaper, Cheese)	Timestamp(2020-07-25 12:27:00), Timestamp(2020-07-31 12:26:00), Timestamp(2020-08-12 12:26:00), Timestamp(2020-08-18 12:26:00)	28.57, 28.99
(Diaper, Meat)	Timestamp(2020-07-25 12:27:00), Timestamp(2020-07-31 12:26:00), Timestamp(2020-08-09 12:26:00), Timestamp(2020-08-18 12:26:00)	28.38, 23.96
(Diaper, Milk)	Timestamp(2020-07-25 12:27:00), Timestamp(2020-07-31 12:26:00), Timestamp(2020-08-14 12:26:00), Timestamp(2020-08-18 12:26:00)	23.94, 25.00
(Diaper, Pencil)	Timestamp(2020-07-30 12:26:00), Timestamp(2020-08-02 12:26:00), Timestamp(2020-08-09 12:26:00), Timestamp(2020-08-14 12:26:00), Timestamp(2020-08-17 12:26:00), Timestamp(2020-08-20 12:26:00)	28.21, 30.19, 28.21
(Diaper, Wine)	Timestamp(2020-07-25 12:27:00), Timestamp(2020-08-07 12:26:00), Timestamp(2020-08-14 12:26:00), Timestamp(2020-08-22 12:26:00)	24.18, 26.88

Figure 7: Outputs of evaluation phase 2

- Analysing accuracy of the algorithm we get highly accurate results.
- Observing the results we find that the results are periodic in the nature.
- When we try running the algorithm on the large data set, the execution time is extremely high. This is the huge setback of the algorithm
- Another cons being that the naive implementation of the modified Apriori algorithm has huge computational cost.

10 FUTURE SCOPE

In the future scope of the project we can try using the data structures like hash tree and trie data structure implementation mentioned in the [3] to decrease the execution time. Another thing that can be useful will be implementing the algorithm using Map Reduce. Parallel computing will definitely decrease the execution time. Currently in the project we are not considering the concept of "confidence" and "interest" and ended up getting many irrelevant item sets. But in the future scope for extremely large data sets it is recommended to incorporate these concepts to get only those frequent item sets in the output which are of high importance.

REFERENCES

[1] Rakesh Agrawal and Ramakrishnan Srikant. 2000. Fast Algorithms for Mining Association Rules. *Proc. 20th Int. Conf. Very Large Data Bases VLDB* 1215 (08 2000).

[2] A. Kakoti Mahanta, F. A. Mazarbhuiya, and H. K. Baruah. 2005. Finding Locally and Periodically Frequent Sets and Periodic Association Rules. In *Pattern Recognition and Machine Intelligence*, Sankar K. Pal, Sanghamitra Bandyopadhyay, and Sambhunath Biswas (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 576–582.

[3] Fokrul Mazarbhuiya, Mohamed Shenify, and Husamuddin Mohammed. 2014. An Efficient Algorithm for Mining Locally Frequent Itemsets.