Identification of interesting news combinations in Twitter

Data Mining project of Academic Year 2017/2018

Damiano Chini
mat. 190937
Università degli Studi di Trento
Via Sommarive, 9
38123 Povo TN
Italy
damiano.chini
@studenti.unitn.it

Davide Corradini
mat. 190996
Università degli Studi di Trento
Via Sommarive, 9
38123 Povo TN
Italy
davide.corradini-1
@studenti.unitn.it

Alessandro Valentini
mat. 186286
Università degli Studi di Trento
Via Sommarive, 9
38123 Povo TN
Italy
alessandro.valenti-1
@studenti.unitn.it

ABSTRACT

Tweets analytics is an important data mining task that lets us get information from what users post. Gathered information can be exploited in multiple ways and for multiple purposes.
Our goal is to analyze a large amount of tweets in order to perform three kinds of operations:

1. detect topics of the tweets and group tweets talking about the same topic;
2. find a correlation between the found topics;
3. profile each user based on its topic of interest.

We present an explanation of how we managed to develop some methodologies to achieve our goal, according to the knowledge learned in the Data Mining course by professor Yannis Velegrakis at DISI UNITN.
Some of the techniques we used are Jaccard similarity, an approach based on locality-sensitive hashing, hierarchical clustering, etc.

1. INTRODUCTION

In the last decade – yes, Twitter is that old! –, microblogging platforms such as Twitter (https://twitter.com) have risen as a really popular form of social media. Everyday, Twitter's users post hundreds of millions tweets about thoughts, their lives, events that happened, stuff they are interested in.

Now, suppose to have access to this huge amount of user generated data. How can we exploit it? What kind of useful information can we retrieve from it?

Analyzing tweets is a fundamental data mining task, which is important in many areas of study. We can use data mining to extract information from tweets and discover useful and interesting things. For example, we could be interested in what people think about the next political election; if we were a company, we could be interested in the reputation of our products or in the interests of the users in order to perform a targeted advertisement campaign. There are infinite ways to exploit this kind of data.

According to the Data Mining course's project we chose, we can use data mining technologies to perform some kind of analysis on tweets in order find topic correlation among tweets. We can cluster (aggregate) tweets by similarity; for example, tweets that talk about the same topic will probably have some words in common. Then, we can find correlation between topics – finding something that is in commons among topics (clusters) – and, furthermore, tell for each user what are the topics he/she is interested in – whether he/she tweeted about them or not.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

2. RELATED WORK

To get updated on the state of the art of this kind of activity involving tweets, topic detection, etc., we looked around among various publications concerning this topic.

The work [4] tries to describe which are the measures which better approximate the similarity of text documents. The ones taken into consideration are the Euclidean Distance, the Cosine Similarity, the Jaccard Coefficient, the Pearson Correlation Coefficient and the Averaged Kullback-Leibler Divergence.

The way they do it is by calculating how well they permit to cluster text documents, which is also what we use the similarity measures for – anyway, tweets are slightly different because they are shorter, according to our data the average length is 66 characters.

They come up with the result that the worst similarity measure is the Euclidean Distance, while the other ones have very similar results in terms of clusters quality. The same conclusion was also found in the work [8]. Given these results, we chose to adopt the Jaccard index as a measure of tweets similarity – later on we will refer to it with the name of Jaccard similarity S_J.

The work [5] aims instead at identifying topics along a Twitter dataset and ranking them to extract the top trending topics. As a first step, they clean the tweets and aggressively remove those which may mislead the results, such as those tweets which have more than one hashtag or too few words. As a second step they identify the topics by applying a hierarchical clustering and cutting the resulting dendrogram with a distance threshold. They finally rank the topics, having good results.

In our work, what we are interested in is the part regarding the tweet topic detection; [5] shows that a hierarchical clustering with a carefully chosen distance threshold well addresses the problem.

Since hierarchical clustering always requires $O(n^2)$ computations, for large datasets an efficient approximation is needed. For this reason [2] shows that, by applying a locality-sensitive hashing based algorithm, – when scaling – the number of computations required to produce the clusters increases linearly with the size of the dataset, hence reducing a lot the time required with respect with the classical hierarchical clustering.

The work also shows that the cluster quality obtained with the locality-sensitive hashing (LSH) algorithm is comparable with the one of the normal hierarchical clustering.

Another publication [7], shows instead that topic detection can be significantly improved if metadata of the tweets are taken into consideration, other than the mere text content of the tweets.

The main object of the work is to try to address the problem of learning similarity functions for topic detection, using previously annotated tweets. The features adopted for training the model in fact are not only related to text, but they include also *semantic features*, *time-aware features* and *metadata*. In particular, the metadata that are taken into consideration are the *author*, the *namedusers*, the *hashtags* and the *urls*.

3. PROBLEMS STATEMENT

In this section we give some formal definitions of the problems we had to solve in order to achieve our goal. Later on – in the next chapter – our solutions are presented.

We have at disposal a huge dataset of tweets. Our goal is to find the interests of each users, and furthermore find other topics the user can be interested in but it does not tweet about.

We split our high level problem into three lower level problems:

1. **Topic detection and clustering**: we have to create clusters (groups) of tweets about the same topic;
2. **Topic correlation**: we have to find which topics (clusters) are related;
3. **User profiling**: finally, we can tell for each user what are its interests and other topics related to them.

3.1 Topic detection definition

First of all, we assume that one tweet is a document d is associated with only one topic t. This seems to be a reasonable assumption because of the very specific characteristics of a tweet – a short message usually composed by only one sentence, that is likely to involve a single topic only.

It is also to be said that a topic for us is not which is know *a priori*, in the sense that it is not something like “politics”, “sport”, etc. On the contrary, it is something that comes out the tweets themselves, since important and well defined topics like “Russiagate” or “Hurricane Irma” can not be known before their happening.

For this reason, a topic T_i is actually defined as a set of tweets, where all the tweets refer to the same topic. So, if $D = d_1, d_2, .., d_n$ is the set of all the tweet documents, we want to partition D into subsets T_i, which represent the detected topics:

$$D = \bigcup_i T_i$$

where for every i, j it holds that $T_i \cap T_j = \emptyset$ (for the fact that one tweet is only associated with one topic).

3.2 Topic correlation definition

Given the output of the topic detection problem, which is a set T of topics T_i, we want to find the set R composed by all the pairs of topics that are somehow related to each other. More formally:

$$R = \bigcup_{i,j} r(T_i, T_j)$$

where $i < j$ and $r(T_i, T_j) = \{(T_i, T_j)\}$ if topic T_i is related to T_j, and $r(T_i, T_j) = \emptyset$ otherwise.

3.3 User profiling definition

The last problem to be solved is to provide – for each Twitter user present in the dataset – a set of topics which he/she is interested in. If U is the set of all the Twitter's users, and T is the set of topics extracted in the previous steps, we want to find a function f that associates a user with all the topics he/she may be interested in: $f : U \rightarrow 2^T$

4. SOLUTIONS

In this chapter we expose the solutions we conceived to solve the problems stated above.

For each low level problem we developed more solutions. In the next chapter we will analyze which are the best choices among the solutions we conceived.

4.1 Topic detection and clustering

Our goal is to create some clusters of tweets containing topic-related tweets.

4.1.1 Getting started: data cleaning

Tweets are user generated data, therefore can contain a lot of dirty data. Before performing any operation on this data, a data cleaning is needed.

Our cleaning procedure – done on each tweet's text – is composed by:

- **Normalization**: all the letters in the text are converted to lowercase, links to pictures or videos are removed, mentions of other users are removed (words starting in `@`), and words that do not contain alphabetic characters only are removed as well.
- **Tokenization**: tokens are created from the normalization survived string. A token is a word. Each tweet is composed by a list of tokens (words).
- **Stemming**: we reduce inflected and derived words into their *stem* word, a base form. For example, words such as “interesting” and “interested” are converted to their base form “interest”. This operation is delegated to the Python library `nltk` [1] that implements a stemming algorithm.

- **Stopwords disposal**: we remove from the list of tokens all the words that do not have a semantic meaning, such as "the", "a", "of", etc. This procedure as well is done using the Python package `nltk` [1] that provides a list of stopwords. Other stopwords were added manually later after a manual analysis of the survived tokens.
- **Dirty words disposal**: some other annoying words are removed, especially words shorter than 3 characters, words longer than 20 characters and words that contain more than 3 equal consecutive letters (such as "helloooooooooo"). This latest rule was added because a lot of tweets contained such kind of words and their presence changed the behavior of our Jaccard similarity algorithm.

Now we can consider our data clean and we can use it safely for any kind of purpose.

4.1.2 Our first approach: Jaccard similarity

We think that tweets that have a high number of words in common are very likely to be related to the same topic. For this reason, the first idea we had in order to address the topic detection problem was to compute the Jaccard similarity between tweets and then perform a hierarchical clustering based – indeed – on their similarity.

The Jaccard similarity S_J between two tweets d_1 and d_2 – just to remember, a tweet is a document composed by a list of tokens (words) – is computed as the ratio of their words' intersection set size over their words' union set size:

$$S_J = \frac{|d_1 \cap d_2|}{|d_1 \cup d_2|}$$

This approach needs the computation of the Jaccard similarity for each pair of tweets; this means that the outcoming algorithm has a complexity of $O(n^2)$ because for each of our n tweets, we compute the Jaccard similarity with the other $n-1$ tweets.

This big amount of Jaccard similarities computations can become a bottleneck when working on large amounts of tweets – our case. In fact – as you will see later on – we developed another approach based on the locality-sensitive hashing (LSH) in order to reduce the size of our vector space.

Going back to our first approach, after the Jaccard similarities computation phase we continue with an hierarchical clustering algorithm in order to group tweets based on their similarity.

The Jaccard similarity algorithm is easy to implement, in fact we had not the necessity to find an implementation inside a library. We implemented it from scratch at first in Python, and then in C++ using SWIG for binding the function back to Python. We moved to C++ because the computation is much quicker and it saves a lot of time when executed billions of time – like in our case.

Concerning the clustering algorithm, the implementation is harder, therefore we searched around for Python libraries implementing it, and found the `SciPy`[6] library. Unfortunately, `SciPy`'s algorithm – and others libraries' algorithms we found – did not satisfy our requirements, so we had the necessity to implement a custom version of the clustering algorithm, inspired however by the one provided by `SciPy`.

The two main reasons why we had to build our custom version of the clustering algorithm are:

1. **Compatibility**: the `SciPy`'s algorithm accepted as parameter a distance value, while Jaccard similarity is a similarity vaule. Converting Jaccard similarity S_J to a distance value $dist$ is easy:
   $$dist = \frac{1}{S_J}$$
   but we have to deal with infinite values when the Jaccard similarity is zero – most of the times, 98% of our cases. Our implementation discards a priori those values to prevent this issue.
2. **Efficiency**: the `SciPy`'s clustering algorithm takes as input a vector of distances whose size is approximately the square of the number of documents. When working with a huge amount documents like in our case, the input vector becomes heavy on memory. We could build a dynamic algorithm that loads on memory only the distances which are useful at any given moment.

Our customized clustering algorithm brings those improvements to the known clustering algorithm called single-linkage clustering.

Single-linkage clustering is a bottom-up algorithm (agglomerative clustering) that at every step merges two clusters based on their minimum distance – more intuitively, maximum Jaccard similarity.

$$D(X, Y) = \min_{x \in X, y \in Y} dst(x, y)$$

where X and Y are clusters, x and y elements belonging to the two clusters respectively and dst the distance function.

The distance between two cluster corresponds indeed to the distance between the two closest items of the two clusters.

4.1.3 Improvements: LSH vector space reduction

The goal of the locality-sensitive hashing is to efficiently retrieve the documents which are very likely to be considered similar.

The idea behind the LSH technique is to generate signatures for every document which respect the similarity of the documents – i.e. two similar documents will also have a similar signature – and subsequently map the different pieces of signature into *buckets*.

More in details, every signature is split into b bands of r signature elements, and every band is mapped into a bucket. In this way, if two documents have a piece of signature (the band) which is identical, they will be considered as candidates to be similar, since they will be found as colliding in one bucket.

As it can be intuitively understood, the smaller the band is, the less similar the two documents have to be, in order to be candidates; so, once a document similarity threshold has been chosen, the parameters b and r must be set accordingly. In our experiments, for example, we kept $b = 50$ and $r = 2$, so that we are sure that only 3.96% of the 25% similar tweets will be false negatives.

Once we have computed these buckets of tweets, we verify that these candidates are actually similar by computing the Jaccard similarity algorithm among the tweets belonging to the same bucket.

This procedure should importantly reduce the computation time required to build the hierarchical clustering. The time required for generating the signatures and the buckets increases linearly with the number of tweets, while the number of Jaccard similarities computed is not much predictable and actually depends on how similar the tweets are in the first place.

Anyway, many experiments have been provided in the literature, which show that the candidate similar documents usually do not increase more than linearly with the number of documents in the input.

The signatures of the tweets was generated by first building a sparse matrix of boolean values where the rows represent the words (or tokens) that can be found in the tweets, while the columns represent the tweets. So the value of every cell of the matrix can be described as: $v(x, y) = 1$ if token x is present in tweet y, 0 otherwise.

The `SciPy` implementation of sparse matrices *lil_matrix* was used in order to save memory, since a great number of cells is equal to 0. For efficiency purposes, the matrix is afterwards converted into the sparse column oriented matrix *csc_matrix* provided by `SciPy`.

The matrix containing the signature of the tweets is of the form $(n_{hash} \times n_{tweets})$, where n_{hash} is the desired length of the signature and n_{tweets} is the number of tweets. This matrix is initialized with infinite values. n_{hash} hash functions are simulated for every row in the matrix by picking n_{hash} random values in the interval $[0, n_{words})$ where n_{words} is the number of words in the original matrix. Since tweets are very short documents we decided a signature with length $n_{hash} = 100$ was long enough in order to be a representative signature of the tweet.

The resulting signature matrix `sigs` will contain, for every tweet, the minimum value of each hash function computed on all the non-zero words of the original matrix, as reported in Algorithm 1.

Algorithm 1 Signature matrix generation

```
1: for each $t$ in tweets do
2:     for each non-zero $word$ in matrix[t] do
3:         for each $hashFunction_i$ do
4:             sigs[i][t] $\leftarrow$ min(sigs[i][t],$hashFunction_i(word)$)
```

Afterwards, every signature is split into b bands of r elements. For each of these bands a hashing (modulo 10^9) is computed on the tuple composed by the r elements, and this is done for every tweet. Each of these bands will give birth to a series of hashing values (one for every tweet), which represent the keys of the buckets (which are maps) into which the tweets are put.

So, at the end, the result will be b separate series of buckets.

By looking into these buckets one can find which are the tweets which are candidates to be similar to each other. So, by generating the combinations of pairs of tweets within every bucket, one can verify if the pairs of candidates are actually similar by computing the Jaccard distance between pair.

In our implementation, the generation of the pairs of tweets to be verified were generated from the buckets “on the fly”, since the storing of all the pairs of all the buckets required a huge amount of memory usage.

4.1.4 *Going deeper: using tweets’ metadata*

In order to achieve even better results, we decided to use tweets’ metadata as explained in Spina et al. [7]

In particular, we used hashtags (words starting with `#`), mentions (words starting with `@`) and URLs to external resources (words starting with `http(s)://`).

Hashtags denote topical co-occurrence more than normal words and hence two tweets containing the same hashtag should be considered as belonging to the same topic.

When a pair of tweets mentions the same named users, they are probably about the same topic.

The same concept is valid for URLs: when two tweets refer to the same URL, they probably belongs to the same cluster even if they do not have many words in common.

Considering these facts, we decided to consider these metadata during the clustering phase: if two tweets have metadata in common, then they belong to the same cluster.

4.2 Topic correlation

The second challenge we faced was to find a way to link the topics we extracted in the previous phase.

Preface: we consider as a topic each cluster we built. Tweets with the same topic will belong to the same cluster.

We immediately discarded a bunch of methodologies that involved topic correlation via similarity. Since we used similarity to build clusters (topics), we expect the clusters to be separated enough (only a few words in common at max). Using again such kind of distance measure would not lead to any interesting result.

We developed some alternative ways to achieve our goal, ’some of which’ inspired by Spina et al. [7].
In particular, they used an user-based and a time-based correlation algorithm for solving the topic detection problem and we extended it to topic correlation.

4.2.1 *User based*

We carefully built our dataset paying attention to include only users that tweeted more than once, in such a way we could develop this user-based methodology.

We consider two topics (clusters) related if k users tweeted about both topics where k is an adjustable value chosen based on the number of tweets and users we have in the dataset.

That’s because a user probably tweets about related topics. If this pattern is recurrent – therefore more users behave in this way – it means we found related topics.

For example, if

- user u_1 tweets about topics t_A, t_B and t_C
- user u_2 tweets about topics t_A, t_C and t_D

then, we can consider topics t_A and t_C related because different user have tweeted the same different topics.

4.2.2 *Time based*

Since every tweet in the dataset is associated with a timestamp, it was possible to follow a second approach to address the problem of topic correlation.

The idea of this second approach was to compare the time distribution of the tweets of the topics. In particular, we partitioned the time into bins, each of which representing a day, and we considered as *peaks* those days in which the amount of tweets posted is significantly higher than the average rate

($tweets/day$) of tweets for that topic. The threshold to consider a rate as a peak, was set to 2 times the average rate; this value seemed to be reasonably high in order to not get too many false positives, and reasonably small in order to avoid false negatives.

In this way, for each topic, we extracted the highest of the peaks and then we were able to discriminate on the correlation of two topics by looking at whether the peak is the same of not.

4.3 User profiling

The third challenge was to extract a profile for each user. The first idea was to associate to each user the topics to which his tweets belong to.

In addition to these topics we decided to include into the user profile also the topics that were found correlated to them.

Each user is hence associated with the set of topics he is interested in and a set of suggested topics that are correlated to the previous ones, which for this reason are likely to be interest of the user.

5. EXPERIMENTS

During the development of ours algorithms and methodologies we had the opportunity to experiment with optimizations and parameters tuning in order to achieve better performance and better quality algorithms.

We are dealing with a huge amount of tweets and some commodity hardware – our laptops. Since these kind of algorithms work better more tweets they analyze, we have to optimize the computation difficulty and the memory usage in order to be able to use as much tweets as possible in the shortest time possible and requiring less memory as possible.

5.1 Development environment

Inside this section it is explained how we managed our development environment.

5.1.1 Programming language

We decided to develop our software using Python as main language. However, – as you read in the previous chapters – some of the tasks that require a great amount computational power are delegated to C++ via some SWIG bindings. This fact because – as you may know – compiled C++ programs are way more efficient than interpreted Python programs, and this saved a lot of time during the execution of our scripts.

Anyway, our choice was to use Python as main programming language because around are available a lot of libraries with interesting and – especially – useful functions already implemented by others. Although Python is a multiplatform language – it works on a large amount of systems: Unix/Linux, Windows and MacOS – we tested and used our code only on our Ubuntu machines.

5.1.2 Our commodity hardware

The machine used to perform the experiment is an Ubuntu 16.04 LTS machine with an Intel® Core^{TM} i7-4700MQ CPU and a 4 GB DDR3 memory.

5.1.3 Git repository

To manage the development of our application we created a repository on GitHub where the code and other useful materials – like datasets, related works, etc. – are freely available to anyone, under the GNU General Public License v3.0. You can find our repository at https://github.com/alvalentini/DataMiningProject

5.2 Datasets

The pieces of software we wrote was executed on many different dataset that met the requirements we had during the development. For example, our first dataset was generated from data found on line and was structured this way:

```
username     tweet_text
```

where a double tab (`\t\t`) is used as separator between the username and the text of the tweet.

Meanwhile our code evolved, the necessity of a different dataset's structure arose. In particular, for the latest version of our code, the timestamp, corresponding to the time the tweet was posted, was necessary to perform an analysis based on time.

Our final datasets were generated with a Python script from a huge dataset the teacher Yannis Velegrakis gave us – about 180GB of compressed tweets.

The script takes a portion of this huge dataset and removes the tweets belonging to users that posted only 1 tweet in their history. This choice permitted us to have at least two tweets per user in order to perform an user based correlation as shown in the previous chapter.

Finally, the script saves on disk some files, each one containing a million of tweets, structured this way:

```
username     timestamp     tweet_text
```

where a double tab (`\t\t`) is used as separator both between username and timestamp and between timestamp the tweet's text.

5.3 Time and memory usage

In this section we analyze the performance of our algorithms in terms of time and memory usage, mainly focusing on the clustering algorithm, since this part is the one that requires most of the resources. In particular we compare the hierarchical clustering with the use of the LSH optimization and without it.

5.3.1 Execution time

As it can be seen in Figure 1 the clustering done without the optimization of the locality-sensitive hashing technique leads to a quadratic growth in the execution time with respect to the number of tweets taken as input. This is due to the fact that the number of Jaccard similarities to be computed is n^2 where n is the number of tweets – the algorithm's complexity is $O(n^2)$.

On the contrary the locality-sensitive hashing technique drastically reduces the number of Jaccard similarities to be computed, and the passages in order to implement the technique itself grows only linearly with the number of tweets.

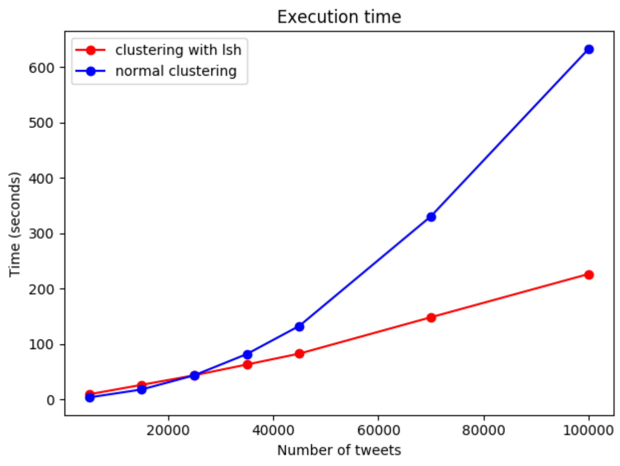

Figure 1: Plot showing the growth of time execution with the number of tweets

5.3.2 Memory usage

Figure 2 reports a plot of the maximum memory consumption detected during the execution of the algorithm.

The memory consumption of the algorithm without the locality-sensitive hashing technique is very low because thanks to the single linkage hierarchical clustering is only necessary to save the pair of tweet IDs whose similarity is greater than the specified threshold.

The memory consumption of the clustering with the locality-sensitive hashing also grows linearly, but the coefficient of the growth is a bit higher with respect to the default technique since the LSH needs to store the structures needed in order to compute the signatures of the tweets and also the buckets.

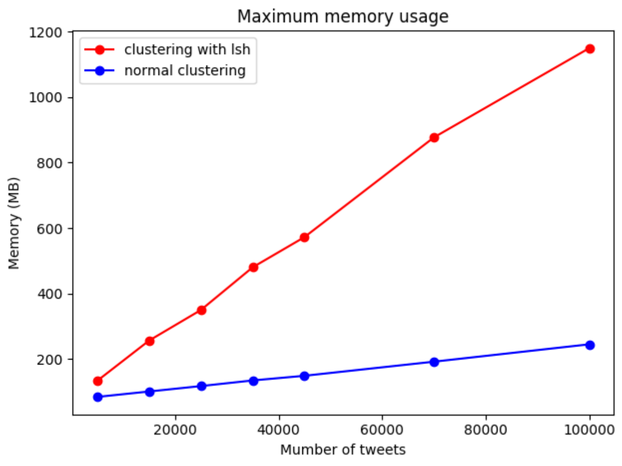

Figure 2: Plot showing the growth of memory usage with the number of tweets

5.4 Topic detection quality

Since the dataset in use does not provide any label for identifying which actually is the topic of each tweet, we could not compute an external evaluation of the clustering algorithm, and hence no external evaluation of the topic detection quality. For this reason only internal measures of the cluster could be computed.

In particular, we decided to measure the density of the clusters created and the distance between clusters with the use of the *Dunn index*[3], which describes the ratio between the minimal inter-cluster distance and the maximal intra-cluster distance. Of course the higher the index the better the quality of the cluster is.

$$I_D = \frac{\min_{1 \leq i < j \leq n} d(i,j)}{\max_{1 \leq k \leq n} d'(k)}$$

The intra-cluster distance was defined as the average distance between the elements of the cluster while the distance inter-cluster is defined for each cluster as the distance between itself and the closest cluster. In particular, for computing the distance between clusters we developed this measure: each cluster is associated to a set of tokens which are the words that appear in at least 50% of the tweets in the cluster and the distance between clusters is trivially computed with the Jaccard similarity of these sets of tokens.

We wanted to measure the cluster quality when varying the distance threshold of the hierarchical clustering algorithm, since this is most important parameter for the topic detection problem. The other parameters for the topic detection problem were set as follows:

- number of tweets: 50000

- minimum cluster size: 2

We also wanted to see measure how many clusters composed by only one tweet are produced when varying the distance. Of course a small number of cluster with only one element should be preferred, since that means that no actual topic was found for that tweet.

5.4.1 Topic detection without the use of metadata

Figure 3 shows how the Dunn index varies when we vary the distance threshold parameter. We can see that when decreasing the distance threshold the Dunn index is higher, which means that the clustering quality is better. So it seems to be advisable to use a distance threshold as low as possible.

Instead, Figure 4 shows, depending on the distance threshold, the ratio of tweets which fall into a cluster of more than one element. As it can be seen, by decreasing the distance threshold a great part of tweets result in clusters of one element. For this reason, it is to be also considered that by decreasing the distance threshold, many clusters result to be composed by only one tweet. This means that either the tweet is a topic itself, or more probably the tweet has not been associated with any topic due to the low distance requirement.

For this reason a trade-off must be considered in order to choose the right value for the distance threshold, which should be reasonably put between the values 2.0 and 2.25

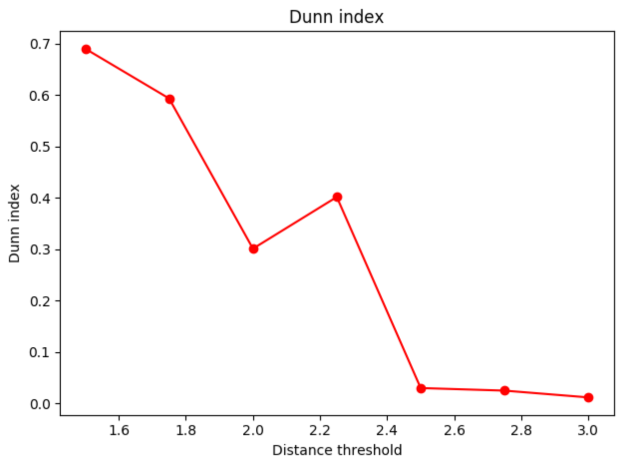

Figure 3: Plot showing the Dunn index on the variation of the distance threshold

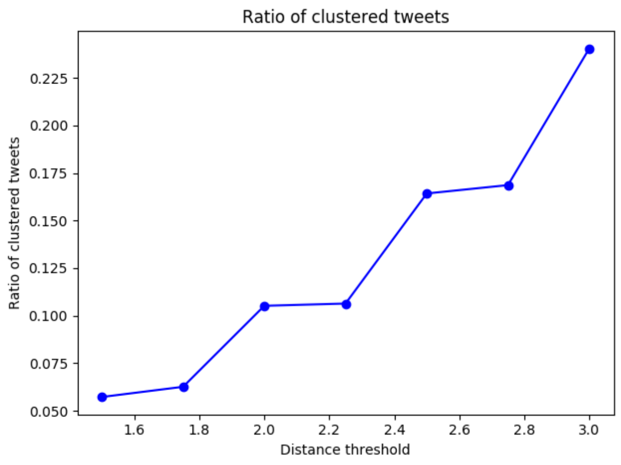

Figure 4: Plot showing the ratio of clustered tweets on the variation of the distance threshold

Since we noticed that the Dunn index seems to be not perfectly stable with the variation of the threshold distance, we tried to tackle the instability of the index by considering not the maximum intra-cluster distance, but an average of the intra-cluster distance:

$$I_{D_2} = \frac{\min_{1\leq i<j\leq n} d(i,j)}{\frac{\sum_{1\leq k\leq n} d'(k)}{n}}$$

Figure 5 shows that our index is actually more stable than the Dunn index, in the sense that the index does not oscillate anymore when varying the threshold distance. Still the general behavior of the index compared to the threshold distance is the same, so we can confirm that a distance threshold between 2.0 and 2.25 seems to be the most appropriate.

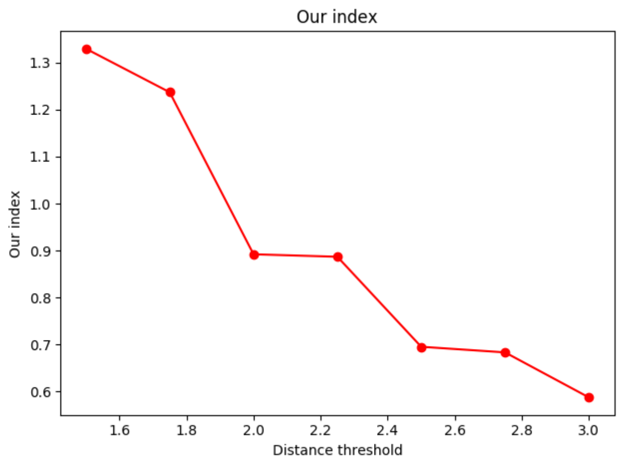

Figure 5: Plot showing our index on the variation of the distance threshold

5.4.2 *Topic detection with the use of metadata*

Figure 6 shows the ratio of tweets which are associated with a cluster containing more than one element when adopting also the metadata of the tweets. We can see that the ratio of tweets with a topic increases in a very similar way to the clustering without metadata (Figure 4), but the use of metadata permits to have a relevant increase in the absolute ratio of tweets with a topic. For example, with a threshold distance of 2.0 the clustering without metadata has a ratio of tweets with a topic of 0.1052, while with the metadata the ratio is of 0.2281.

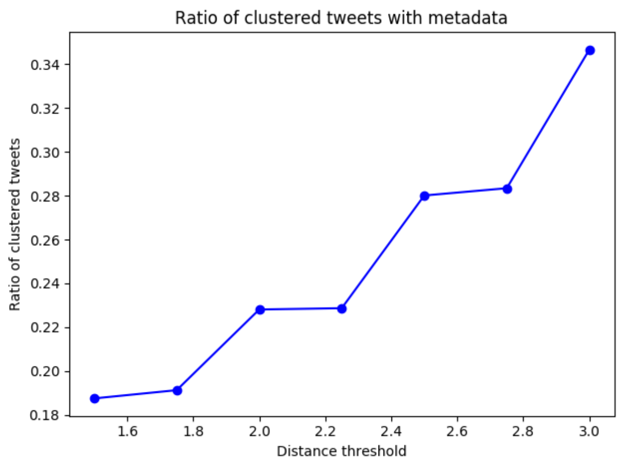

Figure 6: Plot showing the ratio of clustered tweets on the variation of the distance threshold when using also the metadata of the tweets

When adding the metadata for helping the clustering algorithm it is not much clear anymore how to compute the Dunn index, since same metadata leads two tweets with maybe no words or just few in common to belong to the same cluster and hence an actual distance between tweets and clusters is difficult to be clearly defined.

In any case, we can rely on the results provided by [7] in order to be confident on the fact that adding metadata information in the clustering algorithm leads to a more refined topic detection. Spina et al. [7] worked on a labeled dataset, where each tweet was manually associated with a topic. Having at disposal such a dataset it is easy to compute an evaluation measure comparing the clusters found by the algorithm with the topic labels provided by the dataset. In our case – unfortunately – we do not have such kind of dataset, but – as you saw – we tried to set the parameters of the algorithm in order to optimize the characteristics of the clusters, i.e. the density inside the clusters, the separation between clusters and the fact that as many tweets as possible should be associated with a topic.

5.4.3 *Detected topics*

In order to give an idea of the output of the topic detection solution, here we show the top 10 most popular topics detected, with the parameters setted as suggested previously. So, a distance threshold of 2.0 and the use of metadata enabled.
By most popular topics we mean those topic that contain the highest number of tweets. In order to understand of what the topic is actually about, each topic is represented as the set of words which appear in at least the 30% of the tweets contained in the topic (note that the words are presented as they come out after the stemming procedure). Hereafter the list of the most popular detected topics in descending order:

Table 1: Top 10 most popular topics

Topic 1	['follow']
Topic 2	['gameinsight', 'gold', 'collect', 'androi', 'androidgam', 'coin']
Topic 3	['today', 'stat', 'one', 'new', 'follow', 'unfollow']
Topic 4	['birthday', 'happi']
Topic 5	['gameinsight', 'ipa', 'collect', 'coin', 'gold', 'ipadgam']
Topic 6	['nowplay', 'win', 'listenl']
Topic 7	['sexi', 'contact', 'mywebcam']
Topic 8	['facebook', 'new', 'post', 'photo']
Topic 9	['unfollow', 'new', 'stat', 'daili', 'follow']
Topic 10	['follow', 'thousand', 'twitter', 'click', 'visit', 'websit']

5.5 Topic correlation adjustments

In this section we show some experiments done on the topic correlation problem.

The parameters used in the following experiments were set as follows:

- number of tweets: 50000
- minimum cluster size: 2
- distance threshold for the hierarchical clustering: 2.0
- usage of LSH enabled
- usage of metadata enabled

5.5.1 *Correlations on common users*

Figure 7 shows how the number of correlations between topics is affected by variation of the parameter which regulates the number users which have to be in common between two clusters in order to consider them as correlated.

It can be seen that a correlation based on only one user seems to be unreliable, since it gives birth to a big amount of correlations which immediately disappear when increasing the users in common to 2. Hence many topics found with one common user were not actually correlated.

On the contrary the number of correlations found when going from 2 to 3 users in common does not decrease much, which can mean that the topics found with 2 common users are correlated. For this reason, it seems that, with our dataset size, using 2 common users in order to consider topics to be correlated can be a good choice.

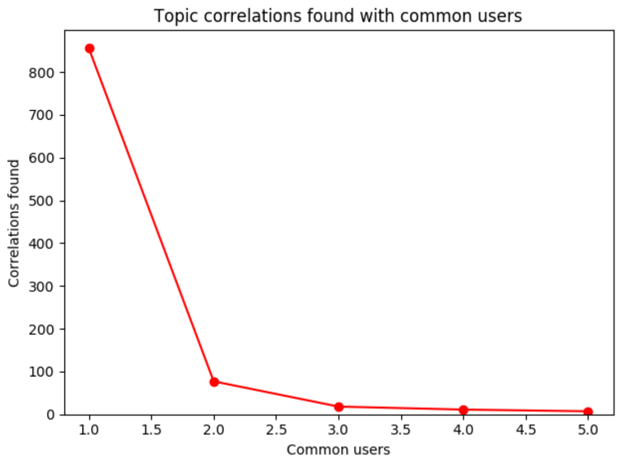

Figure 7: Plot showing the correlations found on the variation of the number of users in common

5.5.2 *Correlations on peaks in time*

We also tried to see how many correlations are found when following the approach of comparing the peaks of tweet posts on different topics.

We observed that the correlations found with this approach are 97, which is in the same order of magnitude of the number of correlations found in the approach based on common users between topics, which can be encouraging.

In order to be able to perform a deeper analysis on the experimental results of this approach a dataset spanning on a large period of time would be needed (besides a large computational power), since an analysis on peaks of tweets does not make much sense if done on a dataset spanning on few days.

6. CONCLUSIONS

We presented various solutions to the three lower level problems.

Concerning the topic detection problem, we showed that we can group tweet based on their similarity using the Jaccard index. Computing a large amount of these indexes – like in our case where we have thousands of tweets – may lead to a performance degradation. We therefore introduced a way to reduce the vector space to explore, using a locality-sensitive hashing approach. This technique reduces the amount of Jaccard similarities we compute, making the algorithm more scalable, without sacrificing the result quality. To achieve an even better result quality, we added tweets metadata as a clustering factor in our process. Now, some tweets that include common hashtags, user mentions o URLs but have only few or none words in common, get clustered in the same cluster (topic). According to [7], who tested this technique on a supervised dataset, the quality of the result improves. We could not test this fact, due to the missing of a labeled dataset.

About the topic correlation problem, our algorithm finds correlation among topics using two kind of analysis: user based and time based. The used based approach does its best when we consider as related, those topics that have in common as many users as possible; however in our case of 50000 tweets we can consider 2 users a sufficient threshold. The number of users depends on the number of tweets: a bigger dataset would generate bigger clusters hence more common users may be needed to consider two topics related. The time based approach could not be deeply tested in a definitive way because a big dataset of linearly time-distributed tweets is needed and our commodity hardware can not afford such a complex computation. However, the results we get with the time based approach are similar to the ones obtained with the user based approach, so we can consider this technique as a possibly valid way to correlate topics.

Finally, we can say that our goal has been achieved: we can provide for each user a set of topics of his/her interest and – furthermore – some more related topics in which the user could be interested in but he/she does not tweet about.

7. REFERENCES

[1] S. Bird. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pages 69–72, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[2] I. Blekanov and V. Korelin. *Hierarchical clustering of large text datasets using Locality-Sensitive Hashing*, pages 61–64. University of Aizu Press, Japan, 2015.

[3] J. C. Dunn†. Well-separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4(1):95–104, 1974.

[4] A. Huang. Similarity measures for text document clustering. pages 49–56, 2008.

[5] G. Ifrim, B. Shi, and I. Brigadir. Event detection in twitter using aggressive filtering and hierarchical tweet clustering. In S. Papadopoulos, D. Corney, and L. M. Aiello, editors, *SNOW-DC@WWW*, volume 1150 of *CEUR Workshop Proceedings*, pages 33–40. CEUR-WS.org, 2014.

[6] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[7] D. Spina, J. Gonzalo, and E. Amigó. Learning similarity functions for topic detection in online reputation monitoring. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '14, pages 527–536, New York, NY, USA, 2014.

[8] A. Strehl, J. Ghosh, and R. Mooney. Impact of similarity measures on web-page clustering. In *Proceedings of the AAAI Workshop on AI for Web Search (AAAI 2000)*, pages 58–64, Austin, TX, USA, 2000.