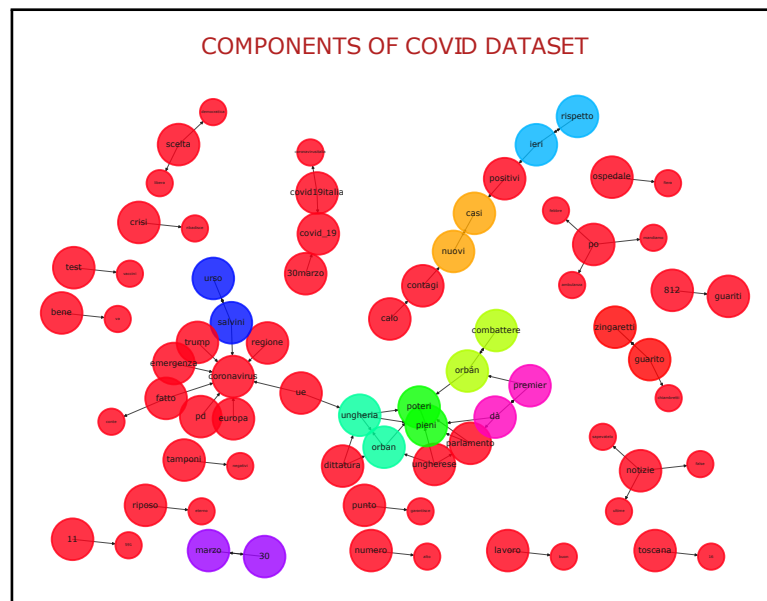




The “Covid-19” case

HOMEWORK IN COMPUTATIONAL INTELLIGENCE AND PATTERN RECOGNITION

May 19, 2020



Student: Andrea Batino

Supervisor: Enrico De Santis, PhD

Professor: Antonello Rizzi, PhD

An Intelligent Topic tracking System in Python.

The “Covid-19” case

Homework in Computational Intelligence and Pattern Recognition

Software setup

Python: [3.7](#)

[Anaconda](#), [Jupyter](#)

SO: [Manjaro 4.19](#)

Hardware setup

CPU: [4x Intel Core i5](#)

RAM: [12 GB](#)

Document version 1.0 - May 19, 2020

Summary changes:

- Rev.



Student: Andrea Batino

Supervisor: Enrico De Santis, PhD

Professor: Antonello Rizzi, PhD

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 2 |
| 1.1 | Objectives of the work | 2 |
| 2 | A brief overview of the state of the art | 3 |
| 3 | Methodology | 5 |
| 3.1 | Data acquisition | 6 |
| 3.2 | Data pre-processing | 7 |
| 3.3 | Algorithm main blocks | 8 |
| 4 | Tracking the ‘‘Covid-19’’. A case study | 13 |
| 4.1 | The data set ‘‘Covid-19’’ | 13 |
| 4.2 | Energies Evolution | 14 |
| 4.3 | Experimental settings and results | 16 |
| 4.3.1 | DAY 05-06 | 16 |
| 4.3.2 | DAY 05-03 | 18 |
| 4.3.3 | DAY 04-19 | 19 |
| 4.3.4 | DAY 04-16 | 21 |
| 4.3.5 | DAY 04-05 | 23 |
| 5 | Conclusion | 25 |
| | Appendices | 26 |
| | Appendix A Software libraries | 26 |
| | Appendix B Main functions and scripts | 26 |
| B.1 | Utils | 27 |
| B.2 | Analytics | 28 |
| | Appendix C TreeTagger Installation | 29 |
| | Appendix D User Case Examples | 29 |
| D.1 | Topic Detection | 29 |
| D.2 | Emerging terms retrieval | 29 |
| D.3 | Plot SCC graph | 30 |
| D.4 | Plot Energies over days | 30 |

1 Introduction

Twitter is a popular microblogging service that allows the users to send and read short text messages called tweets. Since it’s used worldwide by a lot of people, it defines a low level information news flashes portal and can provide a real-time system of information. Our aim in this work is extracting emerging topics expressed by the community, defined as topics extensively treated in a considered time window but rarely in the past.

To achieve this goal, starting from a bunch of tweets, we first need to retrieve info from them. This type of work belongs to the branch of text mining, an artificial intelligence (AI) technology that uses **natural language processing (NLP)** to transform the free (unstructured) text in documents and databases into normalized, structured data suitable for analysis or to drive machine learning (ML) algorithms. The main concept is to associate to each word its frequency within our set of documents (tweets) so that it’s possible to evaluate their importance.

Besides detecting a topic, our system will be able also to track them, checking their evolution and reappearance according to a term energy concept; by this, it will be possible to see for every time window what the topics behaviour is over time.

Carrying out this idea, our main reference is given by the work of *Cataldi et al.* (1); the whole process can be summarized in the following steps:

- tweets retrieval by API streaming;
- compute terms energy within intervals;
- emerging terms selection by ranking their energy values;
- create a graph which links the extracted emerging terms to their relative co-occurent terms;
- retrieve topics from graph applying **Depth First Search (DFS)**

Moving the attention to our dataset, after brief examples of usage on data retrieved at the end of 2019, the main analysis will be focused on coronavirus-related tweets: this period, although dramatic, represents a more unique than rare opportunity for this kind of work. Hence, we collected tweets everyday for 5 weeks so far, during the quarantine period in Italy.

1.1 Objectives of the work

This work is focused on extracting emerging topics day by day, having tweets and other information, such as user number of followers/followee and hashtags used. We first need to define what a topic is in our system: it can be seen as a set of words semantically related.

The first step of our processing pipeline is to retrieve data, by means of the **tweepy** python library. In order to access the twitter API, a developer account had to be created first, allowing us to be able to download json tweets and write them into a csv along-with some other useful information such as user id, number of followers/following, hashtags etc. This operation is done for every time window: at the end we’ll have several csvs.

Upon reading the csvs into a pandas dataframes, we need to preprocess them, i.e. tokenizing text, remove stopwords, extracting hashtags, so that it’s possible to perform analysis tasks. At the end of the computation, for a target day, the emerging topics will be retrieved. Note that their retrieval is not unequivocal, since there are parameters in the system which affect deeply the performances. Wrapping it up, our objective is to return a set of topics for a target day, starting from several tweets collected over days.

2 A brief overview of the state of the art

A more accurate definition of topic is a collection of events caused by a seed event; topic detection and tracking (TDT) is a study of the organization and utilization of information flows based on topical events. The topic detection work provides the raw material for topic tracking: it’s usually done using clustering and topic models.

Topic tracking is based on the definition of a topic with one or more stories, and then uses this topic to find all relevant news stories in the stories stream.

There are two main research directions in traditional topic tracking research. They are based on rules and statistics. The key technology of rule-based research is to analyze the association and inheritance relationship between text content and use the related domain knowledge to summarize the related news texts. The statistical method is mainly based on the probability distribution of text features, using mathematical statistics to determine the relevance of text to topic models.

Methodologically, general-purpose topic detection can produce two types of complementary outputs: either the documents in the collection are clustered or the most important terms or keywords are selected and then clustered. In the first method, referred to as document- pivot, a topic is represented by a cluster of documents, whereas in the latter, referred to as feature-pivot, a cluster of keywords is produced instead. This one is the one used in this work, since a topic is defined as set of words.

Both methods have advantages and disadvantages. Document-pivot methods suffer from cluster fragmentation problems and, in a streaming context, they often depend on arbitrary thresholds for the inclusion of a new document to an existing topic. On the other hand, feature-pivot methods are commonly based on the analysis of associations between terms, and often capture misleading term correlations. In general, the two approaches can be considered complementary and, depending on the application, one may be more suitable than the other.

In the topic tracking research, researchers have tried a variety of methods.

Allan et al.(2) used the Rocchio algorithm to perform topic tracking. By adding weight to specific features in the topic model, the topic model was continuously updated during the tracking of the topic.

Ren et al. (3)used K-Modes clustering to conduct adaptive topic tracking research. First, the texts were clustered. The topic model was represented by the center of each cluster, and the named entity vector was used as the cluster center. The cluster center was iterated continuously during the tracking process until the topic cluster center was stable.

Bai et al.(4) proposed a method of tracking user relationship, which first mapped the blog to the feature space, and then used the improved K-Means algorithm to perform binary clustering to track related topics.

In recent years, with the development and popularity of topic models, the topic model Latent Dirichlet Allocation (LDA) has also been frequently used in topic detection and tracking tasks. LDA is a probabilistic model for modelling text data; it can realize the dimensionality reduction representation of text in semantic space, and it models the text with the probability of vocabulary. It requires the number of topics as input ; every document is considered as a bag of terms which are the only variables in the model. The topic distribution per document and term per topic are instead hidden and have to be estimated by bayesian inference (Collapsed Variational Bayesian inference algorithm or others).

Zhang et al.(5) used the LDA topic model for topic tracking technology research. The topics of the texts were extracted by using the LDA model, and the relevant topics were tracked by *Xu et*

al.(6) calculating the correlation.

The OLDA model proposed by *Alsumaitet al.* (7) constantly adjusted the topics that had been discovered by tracking the topic and used the evolution matrix to preserve the changes of the topics. It incrementally updated the topic model at each time slice using the previously generated model as a prior and the corresponding collection of messages to guide the learning of the new generative process. This method builds an evolutionary matrix for each topic that captures the evolution of the topic over time and thus permits to detect bursty topics.

Pei et al.(8) proposed a variable online LDA (VOLDA) based on the OLDA model. The model took into account the phenomenon of alternating old and new topics that existed in the OLDA model. They designed the dynamic weight calculation method and parameter optimization method. Finally, the reliability of the method was verified in the forum data.

Gong et al.(9) proposed an adaptive topic tracking approach, which was based on an improved Single-Pass clustering algorithm with sliding time window. In the topic tracking process, a sliding time window strategy was used to guarantee the system accuracy and reduce the number of missed following stories. Finally, the experimental results showed that the approach achieved satisfying results.

3 Methodology

As in most information retrieval systems, the first step is the extraction of tweets over several days, and then preprocess them to make them suitable for the analysis.

Thus, for each time window we consider its bunch of tweets as a document and compute the Augmented Term Frequency per word, allowing us to weigh their importance.

Since we collected also information about the user, we can leverage it enveloping the TF score by a constant evaluated from the followers/followee of the relative user, in order to give more relevance to words used by "verified" users. Adding this score over tweets will result in having a nutrition score per word, defined in our system by a dictionary associating to each word appearing

in the time window the relative score.

Then, hot terms are computed by looking at the difference in terms of energy/nutrition across time slots, and see which ones received a burst in the considered slot. The terms whose differential of energy is higher than a threshold set by the user, are considered hot terms.

Next step, looking at our definition of topic, is to build a correlation vector word by word to retrieve which words are more related: with this goal, when two words appear in the same tweet is considered as positive evidence of their relation, and negative viceversa. At the end for every word we’ll have a dict associating weights between words. Since there are a lot of words, later computations might be expensive; thus, a cutoff value is applied in order to delete weak edges/relations.

By means of this correlation vector, we can build a graph and look for the **strongly connected components (SCC)**, that at the end will be our topics: a directed graph is called strongly connected if there is a path in each direction between each pair of vertices of the graph. That is, a path exists from the first vertex in the pair to the second, and another path exists from the second vertex to the first. In a directed graph G that may not itself be strongly connected, a pair of vertices u and v are said to be strongly connected to each other if there is a path in each direction between them; a strongly connected component of a directed graph G is a subgraph that is strongly connected.

Hence for each emerging keyword, we define topic as the subset of SCC including that keyword.

3.1 Data acquisition

In order to retrieve data, we first needed to create a Twitter developer account allowing us to access the Twitter API by the python library Tweepy. In this way, upon creating a listener object for the stream, it was possible to collect tweets, starting from a list of keywords set by the user.

The tweepy response is a JSON, we save different fields of it such as **text, date, user id, user followers/followee, hashtags** into a csv, ready to be read by pandas.

This process is repeated for every time window we’d like to collect.

Here under an example of retrieved dataframe is shown.


| text | id | hashtags | date | name | user_id | followers | following | lists | likes | tweets | creation_date |
|--|---------------------|---|---------------------|---|---------------------|-----------|-----------|-------|--------|--------|---------------------|
| lo ce il voglio proprio vedere i grilloni che ... | 1197517593948704769 | [] | 2019-11-21 14:10:17 | Joker__Reloaded  | 960834864672100352 | 2149 | 550 | 9 | 231 | 45012 | 2018-02-06 11:17:37 |
| @starsiriom Le sardine in piazza gridavano "OD..." | 1197517600365928448 | [] | 2019-11-21 14:10:19 | Ettore | 917304116270989312 | 488 | 131 | 0 | 76376 | 18108 | 2017-10-09 08:21:58 |
| Più chiaro di così...(severo ma giusto pure Ve... | 1197517617671606273 | [] | 2019-11-21 14:10:23 | Enjoy liburd world | 4818612196 | 4753 | 2673 | 50 | 151508 | 107957 | 2016-01-16 17:01:28 |
| @GiorgiaMeloni @FratelliItalia E tu quanti an... | 1197517620137861120 | [] | 2019-11-21 14:10:24 | Spighissimo | 1181858258182686768 | 7 | 12 | 0 | 621 | 325 | 2019-10-09 09:06:03 |
| Perché non assumeva e vendeva per procurarsela... | 1197517624567115776 | [] | 2019-11-21 14:10:25 | Giuditta | 1167372540144246785 | 11 | 19 | 0 | 204 | 1120 | 2019-08-30 09:44:55 |
| A.Mittai: Conte, tratto se azienda resta. "Dom..." | 1197517626127323137 | [{"text": "ANSA", "indices": [91, 96]}] | 2019-11-21 14:10:25 | Ansa Puglia | 1116762632 | 5917 | 8 | 67 | 0 | 29922 | 2013-01-24 13:15:05 |
| TV > Tgcom24 Luca Toccalini, Deputato Leg... | 1197517628996235264 | [] | 2019-11-21 14:10:26 | Lega - Salvini Premier | 13514762 | 96856 | 379 | 500 | 2611 | 137591 | 2008-02-15 13:34:50 |
| ...ninin Umbria cosa avevano dato come "contri..." | 1197517633651978240 | [] | 2019-11-21 14:10:27 | Sassolino66 | 2863782155 | 59 | 140 | 1 | 1087 | 4721 | 2014-11-06 12:59:45 |
| TV > Tgcom24 Luca Toccalini, | 1197517635249999879 | [] | 2019-11-21 14:10:27 | Noi con Salvini | 2900789860 | 39077 | 2 | 115 | 657 | 89494 | 2014-12-01 16:15:16 |

Figure 1: Example of not preprocessed Dataframe

3.2 Data pre-processing

Starting from the csvs, we read them by pandas, and begin preprocessing them. The steps involved are:

- Extract Hashtags: the hashtag field in the Twitter JSON comes as a string of dict, so we need to convert it into a list of hashtags
- Convert to lower case: for the analysis, it's better to not have upper case letter
- Remove links,symbols,emojis and retweets: useless to our program
- Remove stopwords: those are the words most commonly used in a language; since they include articles and conjunctions, they are not useful
- Tokenize tweets: NLP needs single words and not strings, so we tokenize every tweet
- Lemmatize (optional): similar to stemming, associates to every word its lemma
- Remove numbers (optional)

| text | id | hashtags | date | name | user_id | followers | following | lists | likes | tweets | creation_date |
|---|---------------------|-----------|------------------------|-------------------|--------------------|-----------|-----------|-------|--------|--------|------------------------|
| [sentite, pd, conf] | 1199465807346110465 | [] | 2019-11-26 23:11:48 | fran | 108373330695213056 | 180 | 304 | 1 | 9999 | 14285 | 2019-01-11 14:32:15 |
| [cmq, follower, comunisti, ce, cho, pure, lib... | 1199465817139798016 | [] | 2019-11-26 23:11:50 | nonexpedit | 763485613 | 1642 | 570 | 16 | 16129 | 19762 | 2012-08-17 10:50:51 |
| [comeva, l'anno, 2010, poco, prima, sopra, qu... | 1199465898337280000 | [spread] | 2019-11-26 23:12:09 | Barlolomew | 87950192 | 1595 | 1635 | 33 | 30128 | 20908 | 2009-11-06 14:26:23 |
| [successo, tollo, pubblicit , cartabianca, ser... | 1199465920311287811 | [] | 2019-11-26 23:12:15 | Caterina Lanza | 786608182533566464 | 10 | 94 | 0 | 73 | 131 | 2016-10-13 16:43:00 |
| [Italia, oggi, pd, 5s, pub, festeggiare, ramad... | 1199465931547787264 | [] | 2019-11-26 23:12:17 | adriano casotto | 436785388 | 704 | 716 | 1 | 104386 | 15613 | 2011-12-14 15:38:39 |
| [consuma, inchiostro, declina, manifestanti, | 1199466022862016512 | [Salvini] | 2019-11-26 23:12:39 | AnnaGrazia Quarta | 710500851270492161 | 1101 | 600 | 9 | 41170 | 20439 | 2016-03-17 16:19:38 |

Figure 2: Example of preprocessed Dataframe

3.3 Algorithm main blocks

These are the main blocks of the algorithm. The first step, as already said, is to read the csvs, collected in different days, as pandas dataframes: we create a list of them, and their lenght should be more or less the same in order to maintain a sort of word group statistics. Aiming this, during the process of creating the list, we give the user the option to shuffle each single dataframe and compute the one whose length is the shortest, and select the same number of elements for the others. At the end we preprocess each element of the list, describing the data retrieved relative to the different time slots; since it may take time to perform this operation, could be useful to serialize the result as pickle so that it’s possible to access and analyze it quickly next times.

The resulting list is then used to compute both hot terms and correlation vector.

The hot terms are computed by giving to each term in the time slots an energy value, and see which ones received a burst in the last slot. We first need to evaluate the term frequency:

$$w_{j,x} = 0.5 + 0.5 \cdot \frac{tf_{j,x}}{tf_j^{max}}$$

where $tf_{j,x}$ is the term frequency value of the x -th term in j -th tweet and tf_j^{max} returns the highest term frequency of the j -th tweet. So for each time slot we have a list of weights per tweet. Then we

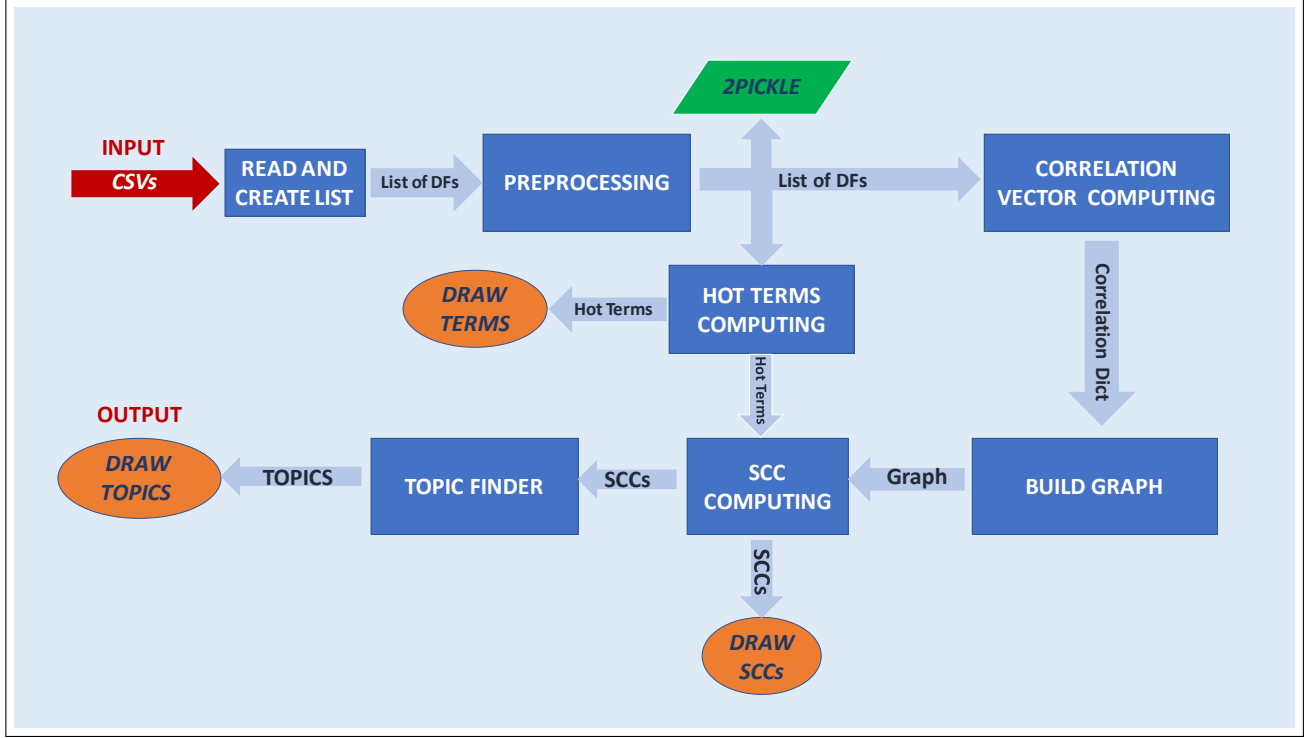


Figure 3: Analysis Block Diagram

define **user authority** as:

$$auth(u_i) = \frac{followers(u_i)}{followers(u_i) + followee(u_i)}$$

so that it’s possible to give more importance to terms used by verified users. This is a very simple case of user authority, because this two are the only ones information about the users we have; it would be interesting to apply a PageRank algorithm but due to the huge size of the twitter network it’s quite expensive computationally and not possible with the machine I have.

The authority is used to compute the nutrition score; considering a keyword $k \in K^t$ and the set of tweets $TW_k^t \in TW^t$ containing the term k at time interval I^t , the nutrition is defined as follows:

$$nutr_k^t = \sum_{tw_j \in TW_k^t} w_{k,j} * auth(user(tw_j))$$

With this nutrition score it’s easy to compute the energy as differences of nutritions:

$$energy_k^t = \sum_{x=t-s}^t (((nutr_k^t)^2 - (nutr_k^x)^2) \cdot \frac{1}{t-x})$$

where the first term is the nutrition/energy term of the considered time window and the second part of the previous ones; this allows us to evaluate the time energy increment/decrement per word.

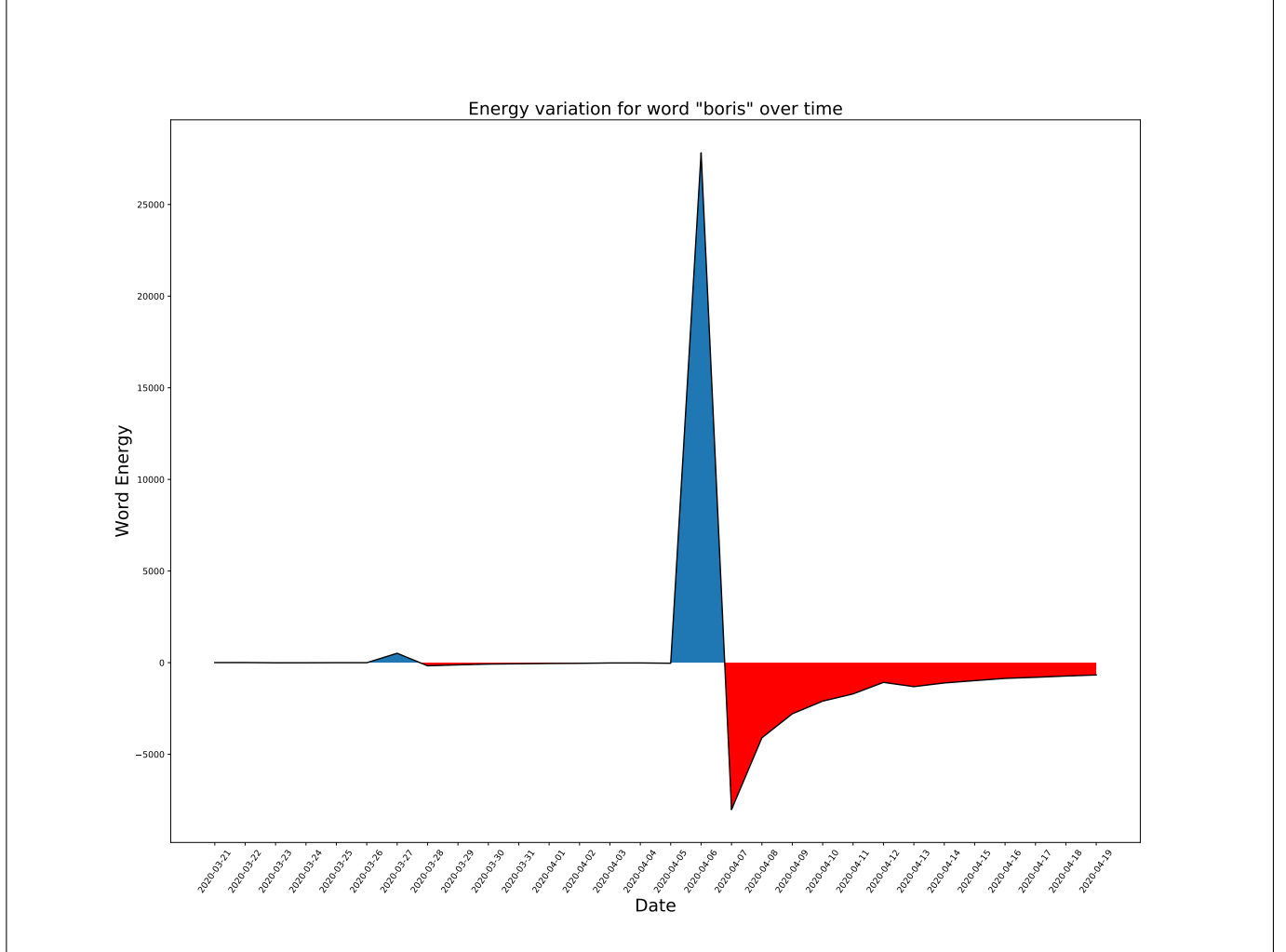


Figure 4: Example of energy

Once we retrieved the hot terms, we need to look for the correlations among terms; therefore we use this formula:

$$c_{k,z}^t = \log \frac{r_{k,z}/(R_k - r_{k,z})}{(n_z - r_{k,z})/(N - n_z - R_k + r_{k,z})} \times \left| \frac{r_{k,z}}{R_k} - \frac{n_z - r_{k,z}}{N - R_k} \right|$$

where :

- $r_{k,z}$ is the number of tweets in the interval containing both keywords k and z
- n_z is the number of tweets containing the keyword z

- R_k is the number of tweets containing k
- N is the total number of tweets

At the end for every term k we have a **correlation vector**, described in our system by a dict.

Starting from the correlation vector, we can build up a graph with:

- Words as Vertices
- Edge whose value is:

$$\rho_{k,z} = \frac{cv_k^t[z]}{\|cv_k^t\|}$$

- The graph is thinned by removing edges with values lower than a threshold
- DFS is applied in order to get the SCC (Strongly Connected Components)

This is quite easy by **networkx** python library, in other cases like implementing it by my own, it would have been really expensive in terms of computation!

Upon obtaining the SCCs, excluding the ones with just one element, we rely on our Topic Finder block to select the components with at least one emerging term within it, ranking and sorting also those components according to their whole energy to give the user insights of which topic is more important. The resulting output will be a table of this type:

It's to be noticed that in our system there are a lot of parameters that need to be tweaked, and according to the different choices the output changes. Let's enumerate all of them:

- **lemma**, flag used to decide whether or not to lemmatize: 0 : not, 1 : fast lemmatize, 2 : slow but more accurate lemmatize
- **num**, flag to decide wheter remove numbers or not
- **stop**, flag used to decide whether reading the stopwords from pickle or text file
- **shuffle**, flag used to decide whether to shuffle the dataframes and cut them to the length of the shortest one in order to preserve statistics, but losing data on the other hand
- **s**, number of previous windows to consider in the hot terms computing
- **window**, time window to be inspected (for which the topics will be retrieved)

Topics

| DATE | TOPICS |
|------------|---|
| 2019-12-02 | conte,mes,camera,meloni,riferire |
| 2019-12-02 | bene,andare |
| 2019-12-02 | presidente,consiglio |
| 2019-12-02 | parlare,parlamento,mazza,fascimo,totale |
| 2019-12-02 | comunista,guccini,francesco,mai,voto |
| 2019-12-02 | premier,tv,gt,dicembre,lunedì |
| 2019-12-02 | tranquillo,stare,cantare |

Figure 5: Example of retrieved topics

- **sigma**, dropoff value used as threshold to detect hot terms
- **cutoff**, threshold used to thin the graph (default=.25)
- **threshold**, maximum number of words per topic

The ones that affect more the algorithm are mostly **sigma** and **cutoff**.

The first is fundamental to detect hot terms, since it’s the number of times that a word energy should be higher than the average energy to be considered as hot term. So the higher **sigma** the less are the terms, and viceversa.

The latter is the one shaping the resulting graph, because edges whose value is less than this parameter will be cut out. It’s easy to understand that a small value will be affecting the system too much, because the graph could be resulting in a huge unique components, not being possible to detect topics. On the other hand a value too high will result in a not connected graph, leading us to too many components. Its choice is the most important of the whole system, and a tradeoff between the two has to be looked for. Usually values within the interval [.2, .35] are the best options.

4 Tracking the ‘‘Covid-19’’. A case study

Upon creating the software at the beginning of 2020, the world started to change dramatically due to the Coronavirus spread. This motivated us to starting collecting tweets about it, also because it monopolized the twitter nowadays. Our goal in this section, is to see how the topics evolve day by day / week by week, allowing us to monitor its behaviour worldwide since its almost beginning and how it evolves.

4.1 The data set ‘‘Covid-19’’

We started collecting tweets on March 9th, using the following keywords: **Salvini, Conte, PD, salvini, conte, pd, lega, Lega, coronavirus, Coronavirus, calcio, Calcio, sport, Sport, UE, ue, europa, Europa, USA, NBA, carceri, carcere, virus, meloni, Meloni, coni, CONI, renzi, Renzi, borsa, Borsa, Trump, NASA, ESA, scienza.** Tweets are collected everyday and saved into csvs, as well as the other info; to date April 14th we have 36 csvs with 20k tweets as average.

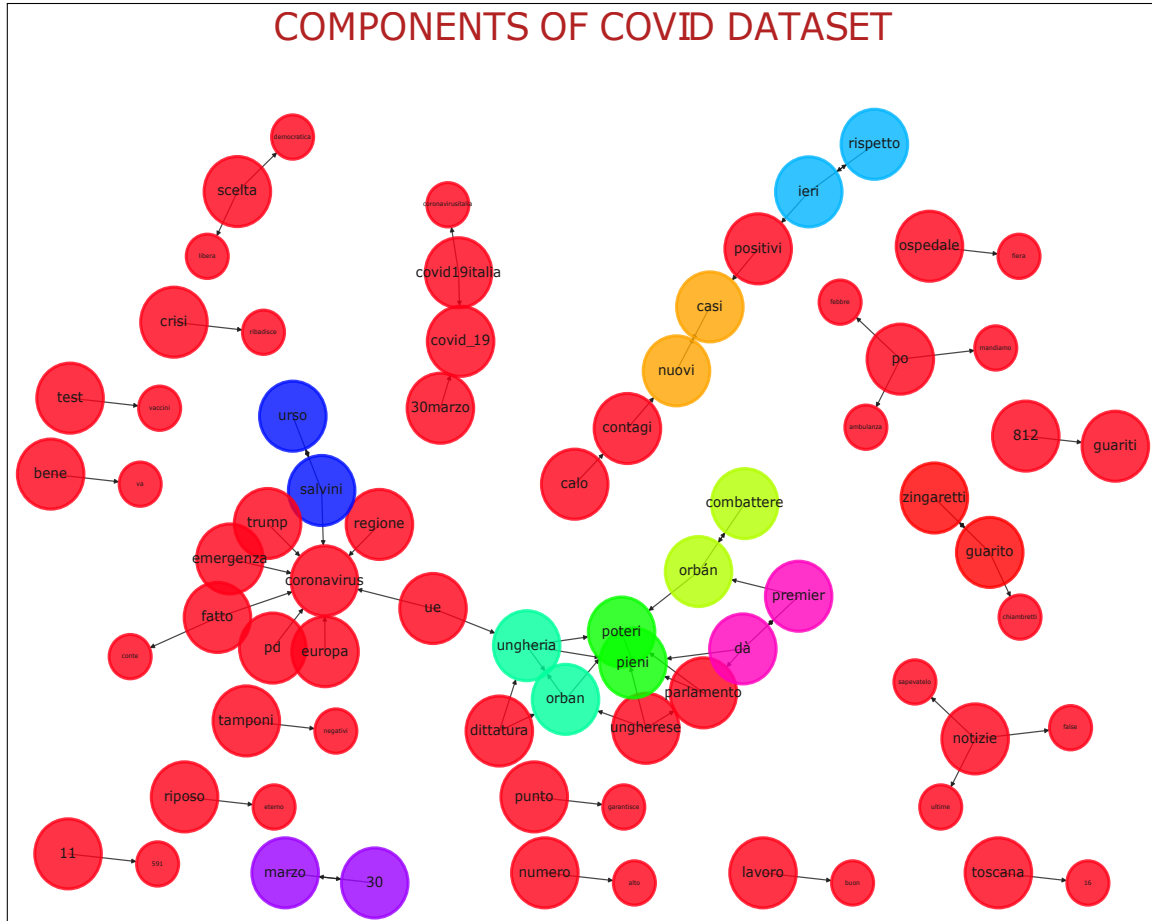


Figure 6: Example of graph

4.2 Energies Evolution

Here we show how energies of some terms varies over time, in a 30 day time window. These energies are min-max normalized and range in the interval $[0,1]$.

The energy for the word *boris*, related to the British PM "Boris Johnson", shows a spike on April 5th, day in which he has been taken to hospital due to coronavirus.

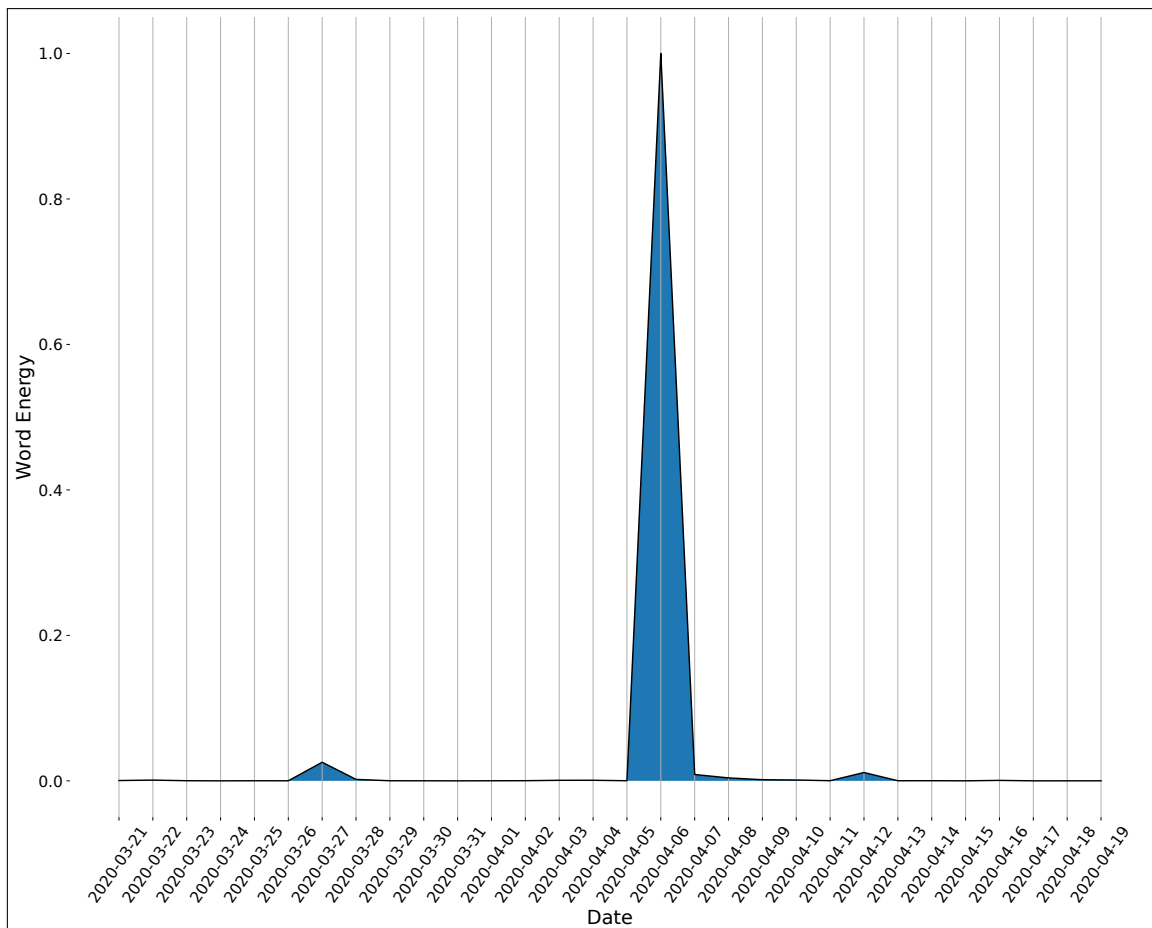


Figure 7: Nutrition for word 'boris'

Moving to *trump*, upon being not so tweeted at the end of march, when the coronavirus pandemic started spreading in the U.S. he started being a more common topic of tweets, as we can see in the last days trend.

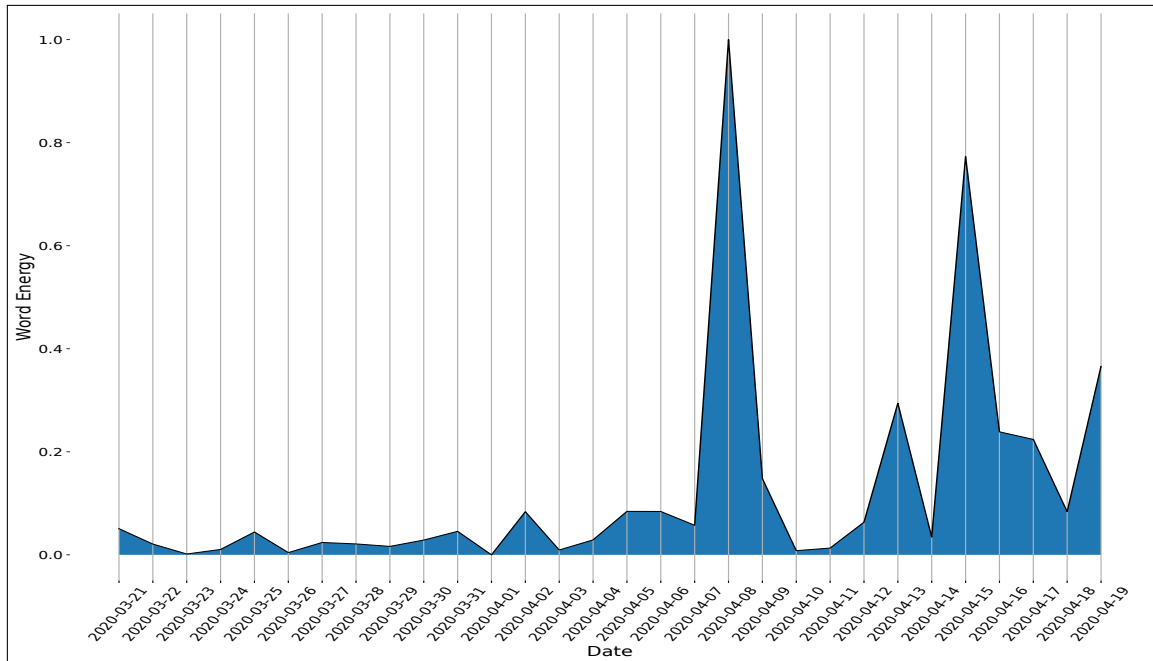


Figure 8: Nutrition for word 'trump'

The last one is the energy for the word *mes*, since it became hot topic in the during April due to the crisis the we're about to face during this harsh times.

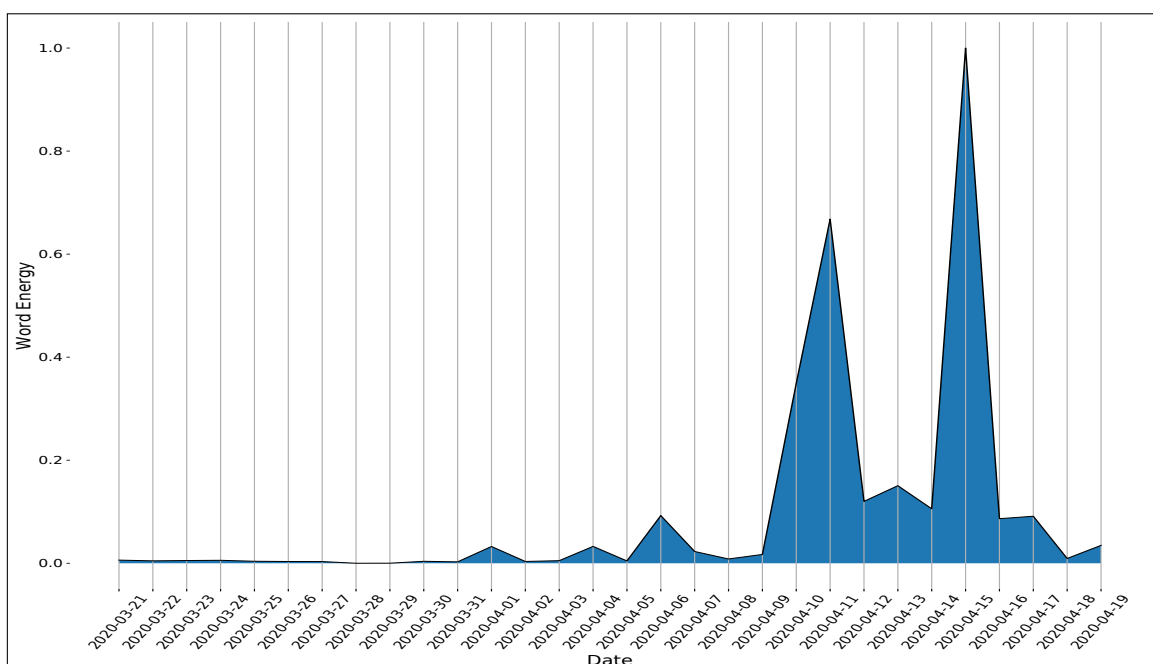


Figure 9: Nutrition for word 'mes'

4.3 Experimental settings and results

In this section the results obtained are reported. Here we are going through different days showing the topics retrieved for each of them, according to the parameters set. Note that these results are obtained by a **grid search**, and the best ones are chosen.

4.3.1 DAY 05-06

Hereunder the words energies and dropoff value are plotted: the terms whose energy is higher than it are labelled as **hot terms**.

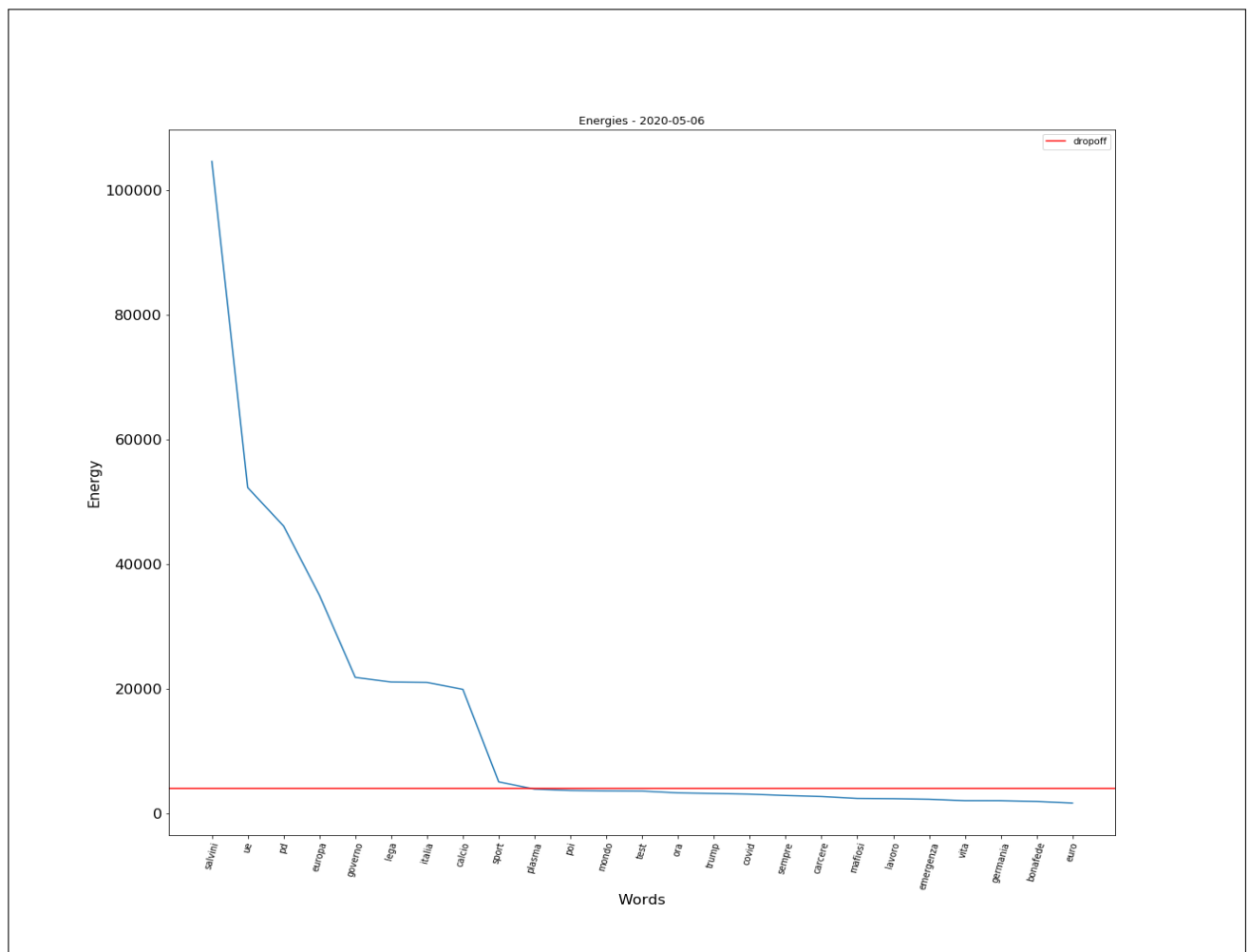


Figure 10: Hot Terms Retrieval

Parameters:

- Not lemmatized, numbers removed
- s=10
- sigma=100
- cutoff=.3
- threshold=7

| DATE | TOPICS |
|------------|--|
| 2020-05-06 | italia,pil,ue,recessione |
| 2020-05-06 | cambiare,sistema,urgenza,alimentare,europa |
| 2020-05-06 | ripresa,campionato,spadafora,data,certa,impossibile,calcio |
| 2020-05-06 | sport,puglia,nuova,aria,aperta,amatoriali,emiliano |
| 2020-05-06 | salvini,pd,chiamato,vendere,pubblicato,libro,persona |

Figure 11: Topics 05-06 (I)

Parameters:

- Not lemmatized, numbers removed
- s=10
- sigma=100
- cutoff=.35
- threshold=7

| DATE | TOPICS |
|------------|--|
| 2020-05-06 | italia,pil,ue,recessione |
| 2020-05-06 | aperta,sport,amatoriali,aria,individuali |
| 2020-05-06 | coronavirus,salvini |

Figure 12: Topics 05-06 (II)

In this day a slight difference in the cutoff value involves huge differences in the topics retrieved. Apart from the first main topic, that is caught by both configurations, the latter only retrieves two other topics where just one is similar to one of the first configuration. This is due to the fact that the cutoff value affects deeply the topology of our graph.

4.3.2 DAY 05-03

This is the day before easing the lockdown in Italy and when the U.S. Secretary of State was asked for evidence upon accusing China to have created the virus in a laboratory.

Parameters:

- **Lemmatized, numbers removed**
- `s=10`
- `sigma=100`
- `cutoff=.35`
- `threshold=7`

| DATE | TOPICS |
|------------|--------------------------------|
| 2020-05-03 | pompeo,wuhan,laboratorio,prove |
| 2020-05-03 | convivenza,comincia,domani |
| 2020-05-03 | fatto,quotidiano |

Figure 13: Topics 05-03

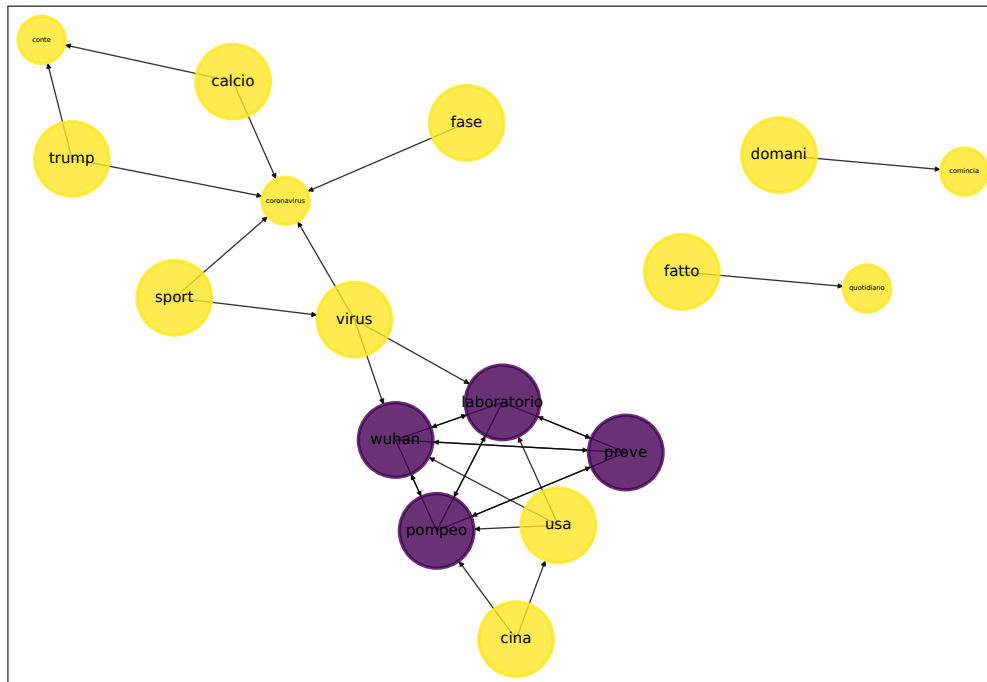


Figure 14: Subgraph 05-03

4.3.3 DAY 04-19

Here we focus on checking what are the consequences of removing numbers or not. We can notice how the main topics are retrieved in both cases, but sometimes it happens that including numbers will mess out topics since they’d be related to each other and wouldn’t be possible to find a sense to the topic.

Parameters:

- **Lemmatized, numbers removed**
- `s=15`
- `sigma=100`
- `cutoff=.4`
- `threshold=6`

| DATE | TOPICS |
|------------|---|
| 2020-04-19 | corona,virus |
| 2020-04-19 | ricciardi,trump,oms |
| 2020-04-19 | calcio,italiano,studio,gravina,becchino,fermare |
| 2020-04-19 | ancora,vittime |
| 2020-04-19 | regione,lombardia |
| 2020-04-19 | nonelarena,salvini,coronavirus,giletti |

Figure 15: Topics 04-19 (I)

Parameters:

- Lemmatized, numbers not removed
- s=15
- sigma=100
- cutoff=.4
- threshold=6

| DATE | TOPICS |
|------------|---|
| 2020-04-19 | virus,corona |
| 2020-04-19 | ricciardi,oms,trump |
| 2020-04-19 | calcio,italiano,studio,gravina,becchino,fermare |
| 2020-04-19 | regione,lombardia |
| 2020-04-19 | 19,aprile,covid |
| 2020-04-19 | giletti,coronavirus,nonelarena,salvini |

Figure 16: Topics 04-19(II)

4.3.4 DAY 04-16

Parameters:

- Not Lemmatized, numbers removed
- $s=15$
- $\sigma=100$
- $\text{cutoff}=.4$
- $\text{threshold}=6$

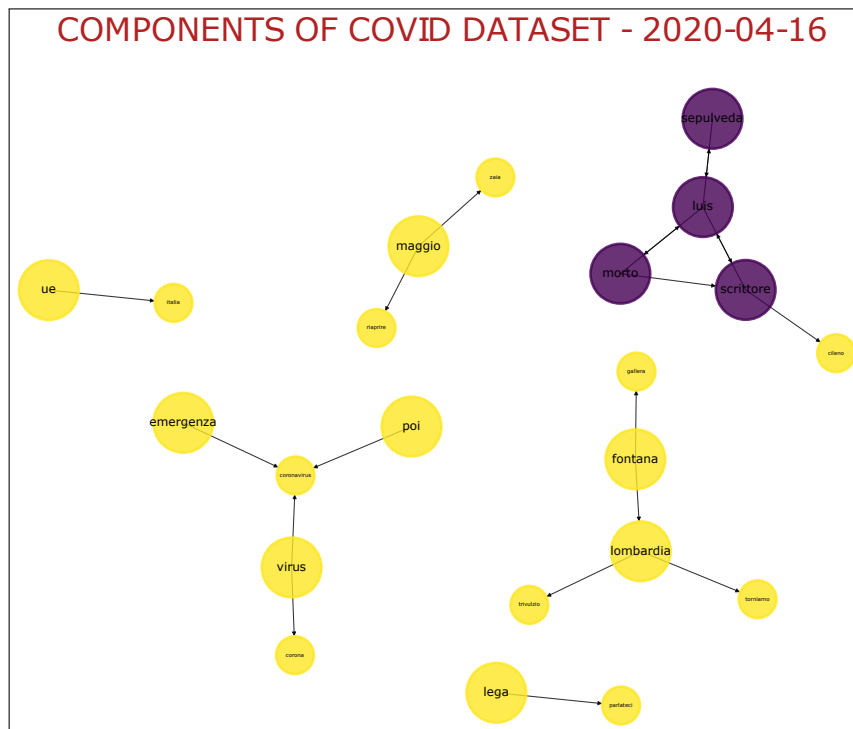


Figure 17: Subgraph 04-16 (I)

| DATE | TOPICS |
|------------|---|
| 2020-04-16 | corona,virus |
| 2020-04-16 | morto,scrittore,sepulveda,luis,cileno |
| 2020-04-16 | zaia,riaprire,maggio |
| 2020-04-16 | disgustato,fontana,sciaccallaggio,galleria,politico |

Figure 18: Topics 04-16 (I)

Notice that since the original graph is huge, here a subgraph where just terms directly correlated to hot terms are plotted. Following this graph, the word "*cileno*" shouldn't come in a topic, because not part of the SCCs of that block, but it does, due to the fact that in the original graph it's likely to be linked back to the word "*scrittore*".

- Lemmatized, numbers not removed
- s=6
- sigma=100
- cutoff=.25
- threshold=7

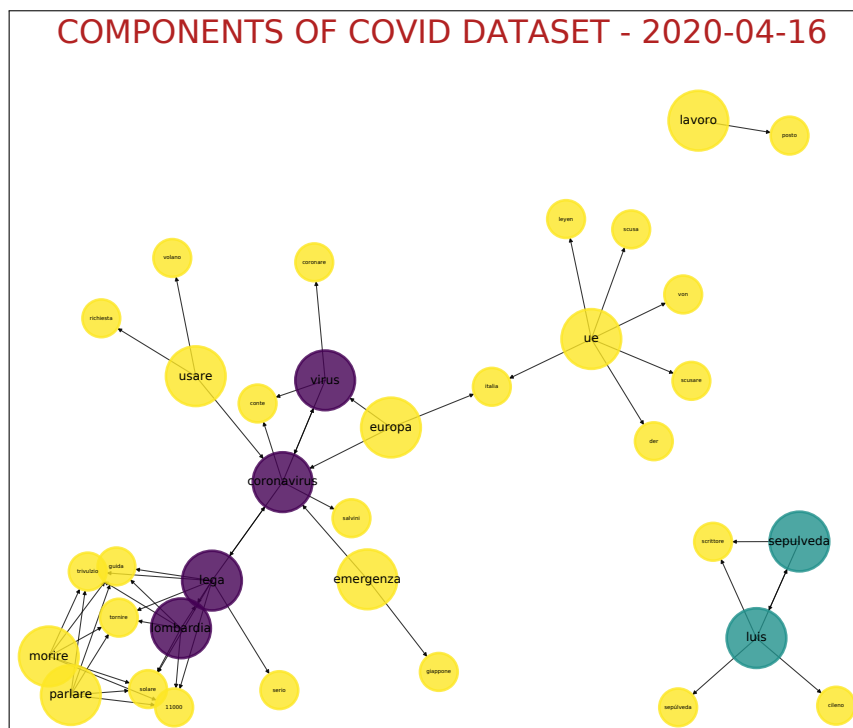


Figure 19: Subgraph 04-16 (II)

| DATE | TOPICS |
|------------|---|
| 2020-04-16 | giappone,abe,rischiare,emergenza,estendere |
| 2020-04-16 | usare,ultimo,milione,news,richiedere,disoccupazione,cronaca |
| 2020-04-16 | lavoro,assoluto,promettere,trent,sbeffeggiato,galera,cattedra |
| 2020-04-16 | ue,scusa,chiedere,scusare,von,der,leyen |
| 2020-04-16 | virus,coronavirus,lega,lombardia,luis,parlare,sepulveda |

Figure 20: Topics 04-16 (II)

Here we can see the impact of both lemmatization and cutoff. The latter changes radically topology and results in different topics retrieved from the first. Now, for example, the *sepulveda* comes along with different terms probably due to a newbie link created through *morire*.

4.3.5 DAY 04-05

For this day we can see how the results change for different configurations. In particular, looking at the last one we can notice how increasing `sigma` yields less topics because there will be less emerging terms.

Parameters:

- **Lemmatized, numbers removed**
- `s=15`
- `sigma=100`
- `cutoff=.25`
- `threshold=6`

| DATE | TOPICS |
|------------|---|
| 2020-04-05 | mascherina,obbligo,obbligatorio,toscana,distribuire |
| 2020-04-05 | restare,casa,riposo |
| 2020-04-05 | sky,sport |
| 2020-04-05 | virus,aprire,fiorello,pregare,provare,fedele |

Figure 21: Topics 04-05 (I)

Parameters:

- Not lemmatized, numbers not removed
- `s=15`
- `sigma=100`
- `cutoff=.4`
- `threshold=6`

| DATE | TOPICS |
|------------|---|
| 2020-04-05 | galleria,lombardia |
| 2020-04-05 | elisabetta,regina,discorso |
| 2020-04-05 | trump,america |
| 2020-04-05 | virus,salvini,aperte,pasqua,corona,chiede |

Figure 22: Topics 04-05 (II)

Parameters:

- Not lemmatized, numbers not removed
- `s=15`
- `sigma=1000`
- `cutoff=.4`
- `threshold=6`

| DATE | TOPICS |
|------------|---|
| 2020-04-05 | virus,aperte,corona,pasqua,chiede,salvini |

Figure 23: Topics 04-05 (III)

5 Conclusion

This work has been interesting and allowed me to learn a lot of new things in a field I didn’t tackle before. Starting with all the preprocessing, it is quite basic NLP preprocessing and thanks to the libraries was easy to implement, even though most of them suits well with English language and thus there are some lack of performances due to our choice to implement it for Italian language. In particular the lemmatization process doesn’t add any ease to the late analysis because of the hard grammar of our language.

Even numbers are not really useful because it happens that some of them are related, meaning that if they appear in a topic list is really difficult to extract insight from them. Hence the best preprocessing configuration seems to be no lemmatization and numbers removed.

Going through the analysis, using `sigmas` lower than 100 is bad practice, since a lot of terms are labelled as emerging, and `cutoff` values within range `[.25,.4]` are preferred. The hashtag burst parameter is also important because it increases energies of terms, so if set too high maybe too many hashtags will end up as emerging terms, giving life to some weird things due to the grammar free property of them; for instance the ‘covid19italia’ hashtag could be inserted as topic even though maybe both covid19 and italia are. The default value of `.1` seems a good tradeoff so that no too much importance is given to them.

The results are in general good, but sometimes it happens that two topics are merged if the wrong configuration is chosen.

An enhancement that can be done is to add an unsupervised selection algorithm to decide dynamically topic by topic the maximum number of words, instead of choosing it a priori. By the way, this system remains unsupervised, and one of the main issues is the lack of an index of error to optimize, such as an MSE for classical ML algorithms; for this reason the parameters tuning and evaluation of results sticks to the user at the moment, not being possible to automatize the process.

A further development can be moving to topic detection in documents, using sentences as tweets and then proceeding as explained above; this can be used to detect topics as we go through the document, and besides a sentiment analysis tracking can also be introduced.

Appendix A Software libraries

Below in Tab. 1 are reported the main Python libraries used.

Table 1: Main Python libraries.

| Library | version | notes |
|----------------------------|---------|---|
| tweepy | 3.8 | Twitter API access |
| pandas | 1.0.3 | DataFrame handling |
| nltk | 3.5 | Natural Language Toolkit |
| plotly | 4.6.0 | Interactive Visualization |
| numpy | 1.18.2 | Array Computing |
| matplotlib | 3.2.1 | Data Visualization |
| NetworkX | 2.4 | Graph Analysis |

Appendix B Main functions and scripts

Here we describe how to use the scripts attached to this work. It comes in a folder called topic detection, whose elements are a main function and 3 other folders: ‘Utils’, ‘Analytics’ and ‘Results’.

The *main* function is the one wrapping the whole work up: upon setting up its parameters it gives a table with all the topics for the target day as output, saved in the folder results along-with a text file with the setting printed.

- System Parameters:
 - **preprocess**, parameter to decide whether reading the csvs and preprocessing them or reading the already preprocessed list from a pickle right away
 - **stop** flag to read either read the stopwords file from pickle (0, default) or from txt (1). Note that both files’names should be of the type ‘italian_stopwords.txt/pkl’ and inside the folder Utils
 - **lemma** flag within [0,1,2] to decide whether to apply lemmatization or not (default = 0, no lemmatization). 1 applies lemmatization after tokenizing words, losing in such a way sort of speech insight, whilst 2 applies lemmatization when the tweets are still strings and keeping the part of speech meaningful, it is much more computational expensive on the other hand

- **num**, boolean flag used to remove (default, True) or not numbers (False)
- **shuffle**, flag used to decide whether to shuffle the dataframes and cut them to the length of the shortest one in order to preserve statistics, but losing data on the other hand
- Algorithm Parameters
 - **date**, string of type '2020-19-04' being the target of the detection
 - **s**, number of previous time windows used to compute the emerging/hot terms
 - **sigma**, drop-off value indicating how many times a word energy should be higher than average's to being considered hot term
 - **hashtag_burst**, energy increase percentage for terms included in hashtag list (default=.1)
 - **cutoff**, edge-thinning value used to remove edges whose connection is feeble
 - **threshold**, number of max words per topic

B.1 Utils

Here we have 3 other folders: 'data', where the csvs are stored, data_pickled, where there are 4 already preprocessed, with different settings, lists of 30 dataframes each, and Treetagger which contains the lemmatization related files. Note that it needs a dedicated installation and it will be explained in Appendix C. Hereunder there is a description of the three scripts:

- **tweet_retrieval**, script used to retrieve tweets from the API, streaming them by keyword filtering. The keywords list is the unique parameter indeed.
- **preprocessing**, used to preprocess twitter dataframe. To run it set the file path and the parameters, such as lemma, stop and num.
- **merge_df**, used to preprocess and fetch into single list a pre-set number of csvs. Its parameters are:
 - **folder**, path to folder containing csvs
 - **lemma**, default=0
 - **stop**, default=0
 - **num**, default=True
 - **shuffle**, default=True
 - **window**, index of the csvs list to start from , default=0

- `n`, number of csvs to merge, default=0
- `to_pickle`, boolean flag to serialize list into pickle

B.2 Analytics

This is the folder containing analysis scripts:

- **hot_terms_computing**, used for compute the emerging terms in a day. Its parameters are:
 - `dfs`, list of dataframes
 - `s`
 - `window`
 - `sigma`
 - `hashtag_burst`, energy increase percentage for terms included in hashtag list (default=.1)
 - `draw`, boolean flag to decide whether or not to draw terms in a table, saved in the emerging terms folder

and as output the hot_terms list, energy dict and average energy.

- **corr**, used to evaluate the normalized correlation vector between words in a list of tweets, described as a dict of dict. Its input are the dataframe and the cutoff value.
- **scc**, used to build the graph starting from the correlation vector and compute the SCCs, defined as list of lists. Moreover there is a function named **draw_components** that allows the user to plot the components starting from the list of dataframes and other inputs such as `s`, `sigma`, `window` etc.
- **topic_finder**, script that envelopes all the previous function and creates the final function to detect topics starting from the list of dataframes.
- **collect_energies**, script that collects energies over different days. It takes as input a list of dataframes and an integer indicating how many days. The output is a list of dictionaries saved in a pickle file in the folder *pickled_energies* (if flag `to_pickle` activated). This script has to be used before *plot_energy*.
- **plot_energy**, script used to plot energy over time for a target word; it takes as input a list of dictionaries and the target, and returns a plot saved in **energies**. In its main the list of dicts is read from a pickle, change path if need different time window (e.g. days).

Appendix C TreeTagger Installation

Follow the instructions at this link: [TreeTagger instructions](#)

Note that this files should be downloaded or moved after installation to the Treetagger folder in Utils. My advice is to download them directly in the folder Treetagger and run the installation process from there.

Appendix D User Case Examples

D.1 Topic Detection

The reference script to perform topic detection is the *main*. Here we can choose between two different paths according to the flag of `preprocess`: if set to False, extract the already preprocessed data from a pickle file in the folder `pickled_data`, on the other hand if True perform preprocessing during the main, adding computational complexity.

- False, you need to tune the following parameters: `date`, `sigma`, `cutoff`, `threshold` and set `name` of the pickle you want
- True, besides those parameters you need to tune also the system ones, such as `shuffle`, `lemma`, `stop_words_f` and `num_flag`

Once you did this, you can run the script: go to `topic_detection` folder and run the following command:

```
$ python3 main.py
```

The output table is stored in the results folder.

D.2 Emerging terms retrieval

Go to `hot_terms_computing` and set the following parameters: `window`, `s`, `sigma`, `burst`. Then open the terminal and run this command within `topic_detection` folder¹:

```
$ python3 -m Analytics.hot_terms_computing
```

The table is stored in the `emerging_terms` folder

¹-m is the command to open sub-folder in terminal.

D.3 Plot SCC graph

Go to **scc** and set the following parameters: **window**, **s**, **sigma**, **cutoff**, **flag** and the **name** of the pickle to retrieve data from. Then open the terminal and run this command within **topic-detection** folder:

```
$ python3 -m Analytics.scc
```

The resulting graph is stored in the **draw_graph** folder

D.4 Plot Energies over days

Performing this operation is composed of two steps: the first one is to collect the energies and serialize them in a pickle file. Go to **collect_nutritions** and set **name** and **n**, respectively name of the pickle file to extract the list of dataframes from and the number of days which the energies will be collected for. Run:

```
$ python3 -m Analytics.collect_nutritions
```

The retrieved energies are stored in the **pickled_energies file**. Then go to **plot_energy** and set **name** of the pickle to retrieve energies from and **words** to plot; then run:

```
$ python3 -m Analytics.plot_energy
```

The resulting plots are stored in **energies** folder.

References

- [1] M. Cataldi, L. Di Caro, and C. Schifanella, “Emerging topic detection on twitter based on temporal and social terms evaluation,” in *Proceedings of the tenth international workshop on multimedia data mining*, pp. 1–10, 2010.
- [2] J. Allan, *Topic Detection and Tracking: Event-Based Information Organization*, vol. 1. 01 2002.
- [3] G. Ren and T. Hong, “Investigating online destination images using a topic-based sentiment analysis approach,” *Sustainability*, vol. 9, p. 1765, 09 2017.
- [4] W.-Y. Bai, C. Zhang, K.-F. Xu, and Z.-M. Zhang, “A self-adaptive microblog topic tracking method by user relationship,” *Tien Tzu Hsueh Pao/Acta Electronica Sinica*, vol. 45, pp. 1375–1381, 06 2017.
- [5] Z. Tong and H. Zhang, “A text mining research based on lda topic modelling,” vol. 6, pp. 201–210, 05 2016.
- [6] G. Xu, Y. Meng, Z. Chen, X. Qiu, C. Wang, and H. Yao, “Research on topic detection and tracking for online news texts,” *IEEE Access*, vol. PP, pp. 1–1, 04 2019.
- [7] L. Alsumait, D. Barbara, and C. Domeniconi, “On-line lda: Adaptive topic models for mining text streams with applications to topic detection and tracking,” pp. 3–12, 12 2008.
- [8] Q. He, B. Chen, J. Pei, B. Qiu, P. Mitra, and C. Giles, “Detecting topic evolution in scientific literature: How can citations help?,” pp. 957–966, 01 2009.
- [9] Gong Zhe, Jia Zhe, Luo Shoushan, Tian Bin, Niu Xinxin, and Xin Yang, “An adaptive topic tracking approach based on single-pass clustering with sliding time window,” in *Proceedings of 2011 International Conference on Computer Science and Network Technology*, vol. 2, pp. 1311–1314, 2011.