Tweet Categorization

A case of document categorization in topics

[redacted]

[redacted]

ABSTRACT

Nowadays, automatic topic recognition in text and documents has undoubtedly become an important task in many fields: from search engines to recommendation systems, from content filtering to sentiment analysis in social networks.

Twitter is one of the most known social networking services and therefore the analysis of the posted content and the research of underlying topics is a challenging task. Contributing in making this task even more challenging is the structure of the tweets themselves, being it different in length, content and linguistic register from other texts available online. Therefore effort has to be put in order to devise ad-hoc solutions aiming at overcoming inefficiencies caused by these peculiarities.

In this paper, techniques used in order to analyze tweets combined with geographic information are presented and discussed in order to provide insight about the most popular topics in the geographic areas of interest.

KEYWORDS

Tweet, Tweet Classification, Topics Mining, Tweet Mining, LDA, Clustering, TF-IDF, Distributed Computing

1 INTRODUCTION

Twitter is an online news and social networking service where users post and interact with messages, known as *tweets*. Since the first tweet, sent on March 21, 2006 by Jack Dorsey, the creator of Twitter, the number of messages sent daily increased dramatically, reaching 500 000 000 tweets per day in 2013[1]. The intermediate steps of this growth, shown in Figure 1, consisted of 5 000 tweets per day in 2007[2], 300 000 tweets per day in 2008[2], 2.5 million tweets per day in 2009[2], 35 million tweets per day in 2010[2], 200 million tweets per day in 2011[3], and 340 million tweets per day when Twitter celebrated its sixth year on March 21, 2012[4].

Due to the way tweets are written, they represent a challenging Natural Language Processing (NLP) task. The main reasons of such complexity are connected with the three main characteristics of tweets, namely the facts that they are

(1) short, up to 140 characters
(2) enriched by contextual clues such as URLs, tags, Twitter names
(3) written in an informal language with misspellings, acronyms, and abbreviations.

This results in three main challenges, namely the very limited word co-occurrence information availability, the less discriminating role played by the word frequency and the limited context, which makes the discrimination between ambiguous words more difficult [5].

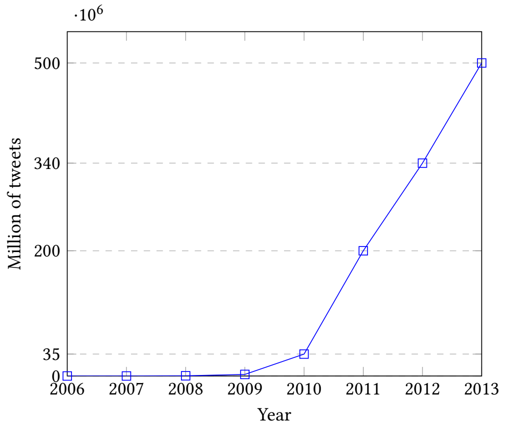


Figure 1: *Growth in the number of tweets sent each day from the birth of Twitter*

In general, the extraction of topics from short texts remains a challenging research problem [6, 7]. Moreover, lately effort has been put in the collection and analysis of short texts gathered from social networks such as Twitter, in order to analyze them to find latent topics. This analysis might have several usages, such as event tracking [8], influential user prediction [9], and content recommendation [10]. Furthermore, this type of analysis, combined with the geographic information of the data would allow to identify and show the popular topics of geographical areas of any size.

With this challenge in mind, the final purpose of this research is to develop an efficient algorithm, able to give insights on the most popular topics in the geographic areas of interest, by finding the most effective approach among several implemented techniques.

In order to tackle this problem, it is, beforehand, necessary to gather the right data that would allow a comprehensive analysis, development and testing of the proposed algorithms. To do so, part of this research is devoted to data collection that, later, allow the comparison of the different proposed approaches.

After the selection of the right dataset candidate algorithms, implementing the different techniques, to be used to compute the topics in the data need to be found. The developed algorithms are based on, nowadays popular, **statistical models** like:

- Naïve Term Count
- TF-IDF
- LDA - Latent Dirichlet Allocation

In this paper also a novel method that is supposed to improve performances for geo-tagged tweets is also proposed. This will be tested against the just listed statistical models in order to appraise its efficiency in solving the proposed task.

Furthermore, for each model **two implementations** are provided:

- an implementation in **pure Python** and
- an implementation in a *distributed fashion* over the cluster-computing engine **Apache Spark** through the pyspark library

The latter approach enables to break the performance barriers of single CPU, allowing the processing of huge quantity of data, thanks to the distributed computation over a large number of Computation Units that share the same network.

This combination of statistical methods and implementations can give a more complete exploration of approaches to the problem, as well as yielding a more interesting comparison and guidelines to a more trustworthy solution.

2 RELATED WORK

The analysis of text aiming at understanding the underlying topics is a field in which a lot of research has been carried on. Although at first this analysis was mainly focused on long texts, nowadays the focus is mainly short ones, such as Facebook statuses, tweets, short messages exchanged trough chats, etc. These forms of communication, that no longer than a dozen years ago were a rare and negligible, now represent an extremely vast part of the content exchanged and published online. Therefore, it is vital that the research follows this trend and focuses to analyze this new form of expression.

The work described in this paper is based on tweets, and therefore also the related work revised in a preliminary step is based on tweets. This preliminary work is now reported in order to give also to the reader an understanding on the state-of-art of the topic recognition in tweets.

One of the most used techniques, both for short and long texts, is Latent Dirichlet Allocation (LDA). However, the naïve application of LDA to Twitter content is considered to be not effective in terms of topic identification. The unsuitability of the LDA method for model tweets has been already proven by Zhao et Al. [11]. However, this technique can be useful during performance comparison as a reference and in order to state where the other approaches stands respectively. In fact, it is rare to find a state-of-art article in which LDA is not mentioned and used although it has been proven not effective.

Moreover, effort has been put in order to exploit heuristic in order to overcome the problem of the limited word co-occurrence information that is available in short text and therefore improve the performances of LDA. Some of these works took advantage of ties among short text to aggregate them into long pseudo-documents before topic inference [5, 12]. Examples of such heuristic techniques include the aggregation of tweets by author [13] or by hashtag [12]. Other authors suggest to aggregate short texts into long pseudo-texts through clustering methods [5].

All this heuristic versions of LDA have been proven effective in topic identification to a certain extent. However, they require either a labeled dataset, a dataset restricted to a certain number of topics or at least a huge set of tweets on which to perform training.

Therefore, in the proposed solution state-of-the-art techniques will not be applied directly, but meditated and adapted in order to make them suitable for the problem at stake.

Another important research field in literature is the research of methods that overcomes the scalability problem, i.e., the capability of a system, network, or process to handle a growing amount of work [14]. Over the years many approaches to this problem were proposed. Starting from shared memory multiprocessing arriving to the more modern distributed memory multiprocessing, there has been a lot of hardware and software models that tried to achieve the scalability goal. Nowadays, the rise of MapReduce paradigm [15] allows, with little effort, to write programs that can be parallelized and executed on a large cluster of commodity machines. This paradigm has been taken as building block by many later solutions and since its birth it has been extended and improved.

The most popular and widely known open source solutions that are based on the MapReduce paradigm, as today, are two Apache Foundation products: Apache Hadoop and Apache Spark. Although being both well known and used, it is important to note that, according to the recent researches, **Spark outperforms** its old brother **Hadoop** in term of speed [16], thanks to its advanced Directed Acyclic Graph engine and **in-memory computing**. Furthermore, Spark includes, at its core, a powerful stack of libraries, including `SQL` and `DataFrames`, `MLlib` for machine learning, `GraphX`, and `Spark Streaming`. By combining these libraries, complex distributed applications can be obtained.

These functionalities were all pondered and kept in mind during the development of the solution of the problem that will be presented in detail in the following section.

3 PROBLEM STATEMENT

Before explaining how the problem at stake was solved, a formal definition of the issue has to be provided.

The general idea is that, given a set of tweets for which the sending location is known, the best way, efficiency wise and effectiveness wise, to recognize the topics of the said tweets has to be identified.

A **topic** is defined as a class of words that can be grouped together and can be associated under a general field of considerations from which arguments can be drawn.

In order to do this, an algorithm that computes such topics had to be devised. The said algorithm should be able to identify topics in different regions (defined as squares in a grid) of the map. Furthermore, the algorithm that can be considered satisfactory of the efficiency and effectiveness constraints has to perform better than others considered algorithms in both time and topic identification correctness.

More formally, given the following inputs

- a set U of pairs `<tweet, location>`
- an area A of the map determined by a top left and a bottom right point
- a step S (e.g. 2km)

create a grid by dividing the area A in squares of size $S \times S$ and identify and show the popular topics in each such square. This has to be done over the assumption that the grid changes every time.

Moreover, two versions of the said algorithm had to be devised in order to compare performances of the single core version of the algorithm with the performances of the distributed computation version.

Optimization have to be put in practice in order to avoid re-computation and ideally improve the performances of the algorithm itself.

4 SOLUTION

After a first stage in which existing methods were revised, a solution specific to the problem at stake was formulated.

The algorithm devised in order to solve the problem described in Section 3 consists of five parts, namely:

(1) **dataset collection**: the gather of tweets among which the development, validation and comparison of the algorithm are done
(2) **grid map design**: the generation of the tweet map based on the chosen geographic area, the dataset and specified cell step S.
(3) **data preprocessing**: the procedure of tweet loading and cleaning finalized to ease topic computation.
(4) **topic computation**: the collection of procedures that yield to topic computation for each cell of the grid.
(5) **distributed topic computation**: the procedure of tweets parallel loading, cleaning and topic computation, for each cell of the grid.

A section will now be devoted to each of these parts.

4.1 Dataset collection

Finding a good solution, in this case an algorithm that performs topic recognition effectively, is, in the first place, related to the fact that the data has to be suitable for the constraints defined. Furthermore, data have to be of good quality, not too noisy or local. If the data is either too noisy or local, in fact, topics would not be identified correctly due to disturbs and fluctuations. As an example, if the geographic area analyzed is too small (local) it will be likely that in the area only a limited number of tweets is posted and, consequently, the computation of topics over very few tweets, or no tweets at all, will be ineffective.

For this purpose, several datasets are available online. However, for a specific goal like the one proposed, it is surely better to gather fresh data from Twitter. Through the Twitter API, in fact, it is possible to require data that has been tweeted from a specific geographic area (*geobox*) and also ask for the given tweets to be geo-tagged with GPS coordinates.

The first step was therefore to choose an area of interest, and, once this is chosen, start requesting data to the Twitter API. For this purpose, during the development, a Python script was developed. This script was in charge of opening a connection to the APIs and requesting a custom stream of tweets, which had to:

- have GPS coordinates
- belong to the defined geobox

Figure 2: *Selected Geobox with the map of Trentino*

Depending on the size of the selected area the incoming stream can vary in flow.

In this case, for conducting the research, the selected geographic area was a square box around the *Trentino* region and some near neighbors (see Figure 2). The amount of raw tweets collected was around 35,000 in about three weeks of incoming stream listening.

It is needless to say that captured data were really *raw* and therefore a stage of preprocessing was needed before being ready to be mined.

Another script was devoted to this task. The aim of this script was to load the raw tweets, remove the malformed ones and transform the content in a valid *JSON array* [17] of tweet objects.

The obtained structure of tweets is the following:

```
tweet : <id, text, coordinates, user, . . . >
```

Analyzing the results of the cleaned tweets, it was possible to notice that most of them were still, even if coming from the geobox, without coordinates. As stated before, it is very important for the research that the whole dataset has this kind of location property, therefore, in order to overcome this problem, random coordinates were generated and associated with the data that didn't have this property.

A further step that had to be performed in order to have a better data from a qualitative point of view, was to only select tweets marked as belonging to the Italian language. In fact, although the area selected is in Italy the quantity of tweets written only in English was high enough that it would have caused a noisy topic identification.

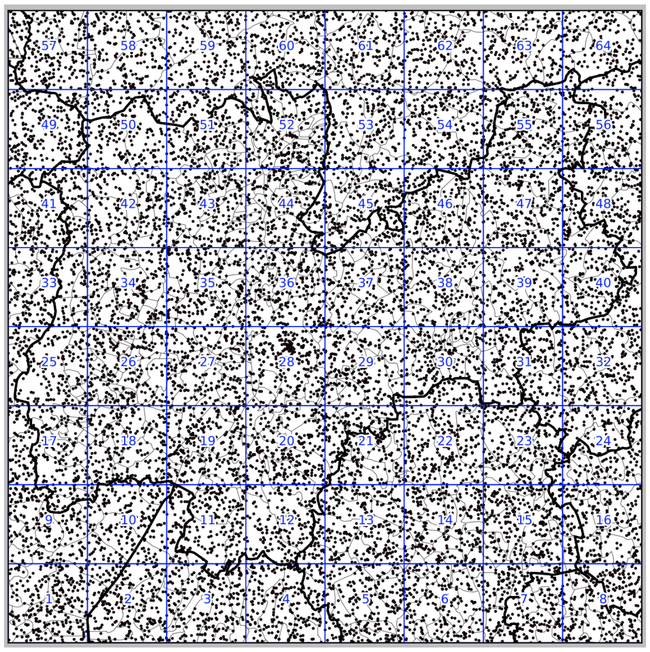


Figure 3: *Tweet map with a 8x8 grid*

4.2 Grid map design

Once the tweets were cleaned and validated, the next step was to check the actual correctness of the collected data, as well as providing visual insight on the dataset. This has been done by devising a map that provides a view over the defined geographical area.

In order to later identify topics in the subsections of the geobox, the map was split into smaller $S \times S$ squares with S equal to the desired size of the area. The S parameter can be specified at each execution, therefore allowing the user to select the level of details of the computed topics.

Once rendered the grid, the geo localized tweets are printed onto the map and the result helped validating the correctness of the collected data. An example of how the rendered grid with the addition of dots in correspondence of the geographical location of each tweet can be see in Figure 3.

Moreover, in this phase, each cell generated by the map splitting was identified by a *cell id* and the various tweets in the dataset were subdivided in their belonging cells accordingly to their coordinates, attaching the cell_id to each of them.

The resulting new tweet structure, with cell_id can be expressed as follows:

```
tweet : < cell_id, id, text, coordinates, user, ..>
```

4.3 Data Pre-processing

Once the tweets have been validated and assigned to one of the cells the map is subdivided into, they are ready to be used as input in the computation phase. Before being processed by the various topic mining algorithms, however, the text in the tweets has to pass through various steps.

First of all, **text extraction** has to be performed. This step consists in loading each tweet object and extract the text that forms it.

Then each of the retrieved texts has to undertake a **cleaning** procedure. This consists in removing all the parts that are not relevant for the topic computation. These include, for instance, URLs, HTML tags, email addresses, emojis (e.g. *:)*, *=-P*, etc.), and hastags. The latter consists of words or unspaced sequences of words preceded by a # symbol. These, although in other researches have been proven useful [12], in our case might as well consist of a source of noise, especially in case of long unspaced sequences of words and were therefore removed. Moreover, also unknown characters, hex codes, and other non-alphanumeric characters were removed from the cleaned texts.

The cleaned texts have to be **tokenized**, i.e. undertake a procedure in which each text string is split in smaller strings that include a single word/term, called *tokens*. This operation also takes care of removing punctuation.

The last step consists in **stopwords removal**. Formally, a *stopword* is a word $w \in B$ where B is the set of the most common words in a language. Typically, stopwords include articles, conjunctions, lexical words, common verbs (e.g. auxiliary verbs).

In this specific case three sets of stopwords have been used:

(1) the list of stopwords for Italian language provided by a Python library;
(2) the list of stopwords for English language also provided by Python;
(3) a list manually generated Italian stopwords.

The reason for the usage of the first set is straightforward, while the others require a brief discussion.

English stopwords had to be removed because, despite all the used text having Italian as main language, in some cases, they still contain parts written in English. This removal does not influence performance as none of the so discarded words is fundamental to topic identification.

After a few preliminary tests, it was also noted that a set of other words, not contained in the first two sets, were causing noise in the topic identification. Such words were therefore inserted in a manually generated Italian stopword list. This list is not only derived from these preliminary tests, but also contains words regarded by various online sources as stopwords.

Furthermore, from the preliminary tests, the fact that shorter words are also to be considered irrelevant emerged. This phenomenon has already been shown in literature [18], as in human languages the frequency of words follows a Zipf's law [19], where frequent words are very short and longer words are less frequent. This phenomenon is known as the Less Effort Principle [20].

Since these short words, as already said, appeared to be irrelevant to the end of topic computation, it was decided to remove also all the words shorter than four characters from the texts.

Once all these steps, which only take a few seconds, are done, the tokenized tweets are ready to be mined.

4.4 Topic Computation

As stated before, a topic is a vector of words that appear together very often. In order to compute such a vector several techniques have been used.

First of all it was decided to use some of the most used standard extraction techniques for topic computation in the literature, namely *Naïve Term Count*, *TF-IDF* and *LDA*.

These first three techniques were used not only for simple topic recognition, but also as a validation method for a novel topic computation algorithm (*Geotag Cross-Document Ranking*) devised to try to improve results for the task formally defined in Section 3.

At first also another technique was used: *Clustering+LDA*. As already said, since only very limited word co-occurrence information is available in short texts, heuristic should be exploited in order to aggregate texts into long pseudo-documents before topic inference [5, 12]. Among the revised techniques the one that seemed more suitable to the task at stake appeared to be the one proposing to aggregate tweets in clusters. More specifically, the Clustering+LDA method consisted in exploiting K-means clustering technique in order to aggregate tweets, and then use each of the obtain clusters as a document for LDA. However, this method produced unsatisfying results, and was therefore discarded in favour of the Geotag Cross-Document Ranking method.

4.4.1 Naïve Term Count

The most naïve and trivial way to approach the problem of mining the most popular topics over a set of documents, or in this case tweets, is to *count* the occurrence of terms over the texts. Despite being fairly raw, this method can provide a good starting point to use also as a comparison measure with other, more complex, methods.

The algorithm, that can be seen in Algorithm 1, is rather simple: given the step size *S*, the grid size is computed. Then a loop is used to evaluate, each step of the loop, a new cell of the map grid. During the cell evaluation, the tokens in the tweets belonging to that cell are counted and the *top ten* are saved to a variable. After this step the results obtained can be stored and/or printed.

Although very trivial, this method can provide significant insight about the topics that are mostly tweeted in the selected area.

Algorithm 1 Naïve Term Count

```
1: grid_size ← S × S
2: for i ← 1 to grid_size do
3:     term_count ← count_tokens(tweets)
4:     cell_top_10 ← top(10, term_count)
```

4.4.2 TF-IDF

In information retrieval, *TF-IDF*, short for *Term Frequency-Inverse Document Frequency*, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus[21]. The TF-IDF value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust to the fact that some words appear more frequently in general.

This method was therefore used in order to appraise how the said offset impact the results, above all with respect to word count.

4.4.3 LDA

Latent Dirichlet Allocation (LDA)[22] is one of the most common probabilistic topic models. Topics learned from LDA are formally a multinomial distribution over words, and, by convention, the top ten words are used to identify the subject area or give an interpretation of a topic.

In order to compute LDA, tokenized tweets are first of all turned into an *id-term dictionary*, which is then used to compute a *document-term matrix*. Then the LDA model is computed using gensim's *ldamodel*. This Python module, based on `onlineldavb.py` script by M. Hoffman[23] allows LDA model estimation from a training corpus. The model can also be updated with new documents in order to support online training. The latter function will be useful for the dynamic computation (see Section 4.5).

The main drawback of this approach is the fact that the number of topics on which to train the LDA model need to be specified. Therefore tuning is required in order to find the best topic number.

Moreover, it is important to note that randomness intrinsic in the LDA computation causes the topic returned by the algorithm to slightly differ from one run to the other.

4.4.4 Geotag Clustering Selection

It was decided to implement a novel method for topic recognition in order to exploit knowledge regarding the data available with the aim of improving the quality of the results obtained as well as the computational time.

The assumption that led to the development of the *Geotag Cross-Document Ranking* algorithm is simple: people writing from the same location are likely to be writing about the same topic. This assumption seems reasonable, especially in rural areas, which constitute the majority of the analyzed territory.

It is important to note that the assumption this method is based on only works for topics close in time. For instance, in mountain areas a hot topic might be related to snow and skiing in winter, and holidays and walks in summer. Moreover, big events might influence the topic. However, this issue is not part of the current research, as the analyzed tweets were collected within three weeks, which seems to be a time short enough for the topic not to change consistently.

Another assumption that constitutes the backbone of this method is the following: words regarding an *"hot topic"* are likely to appear on multiple tweets. Thus, the higher the number of tweets that a token appears in, the higher the probability of it to belong to a relevant topic in a specific area.

The actual implementation consists in the three steps now described.

(1) First of all, tweets has to be **clustered with respect to their location**. For simplicity, it was decided to merge tweets belonging to the same cells in Figure 3. This makes sense, as for the run experiments this is the maximum grid size used, and thus this is the minimum level of detail needed. As shown in the lines 2 to 5 of Algorithm 2, the actual implementation of this step consists in getting, for each cell, the list of tokens used in that area.
(2) Once retrieved the list of tokens, for each of these, a **rank is computed**. The rank is obtained by dividing the number

of documents containing the token by the number of documents contained in the map grid cell at issue. Moreover, as can be seen in lines 6 to 15 of Algorithm 2, tokens that do not appear in at least three documents are considered irrelevant and therefore discarded. It is important to note that number of documents containing a token might differ also significantly from the value obtained using the naïve word count, since a single tweet might contain repetitions of a word. The ranking technique just discussed gives in fact an higher importance to the tokens that appear in the highest number of documents, rather than the actual frequency of the token itself.

(3) Finally, **topic computation** is performed for each cell. This simply consists of ordering the tokens by rank and returning the top ten of each cell.

Algorithm 2 Geotag Clustering Selection

```
1:  function COMPUTE_TOPIC (CELL_ID, TWEETS)
2:      token_set ← set[]
3:      for tweet ∈ tweets[cell_id] do
4:          for token ∈ tweet do
5:              token_set.append(token)
6:      token_rank ← dict{}
7:      n_docs ← length(tweets[cell_id])
8:      for token ∈ token_set do
9:          n_found ← 0
10:         for tweet ∈ tweets[cell_id] do
11:             if token ∈ tweet then
12:                 n_found ← n_found+1
13:         if n_found > 2 then
14:             value ← n_found/n_docs
15:             token_rank.append(token : value)
16:     sorted_token_rank ← sort_by_value(token_rank)
17:     return top(sorted_token_rank, 10)
```

Heuristic is also in practice to avoid recomputation when the map is dynamically rescaled. This is allowed by the fact that the values of `sorted_token_rank` for each cells are kept in memory and not discarded. Details regarding how the rescaling is made efficient are provided in the next section.

4.5 Dynamic Topic Computation

Avoiding recomputation on map or cell resizing is a key point for the project at stake, whose importance is straightforward: a topic computation algorithm, in order to be usable, needs to perform well also with respect to time and memory usage. The efficiency does not depend only on the method itself, but also on mechanisms put in practice to avoid unnecessary computations.

The devised solution needs an explanation, also because different reasonings are in place for map rescaling and for grid rescaling.

4.5.1 Map rescaling

Map rescaling is used to allow user to zoom in and out the map in order to focus on a specific area of the map. It was decided to use the following simplification: resizing can only be done with a factor of two. This means that we can only move in or out grid

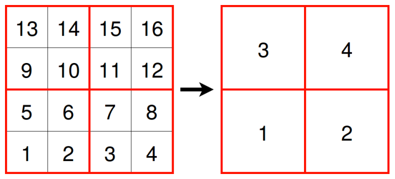


Figure 4: *Resize of a 4x4 map to a 2x2*

squares. For instance, if we are visualizing the a map similar to the left one in Figure 4, we can only move to one of the four red bordered squares. Likewise, if we are in one of these squares, say square with cell id 1, we can return to the previous configuration.

In general map rescaling does not require re-computations, except for the first time an area is visited, as the topics for each smallest-grid-size cells and current-grid-size cells are stored. Therefore, map rescaling simply consists in displaying only the topics for the requested cells. Moreover, in case of bigger maps or frequent resizes also intermediate values for different cell sizes (see Section 4.5.2) can be memorized in order to allow an even faster update of the results and avoid to repeat computations.

4.5.2 Grid rescaling

Algorithm 3 shows how effective re-computation of topics when the size of the grid changes is performed. Also this algorithm works under the simplifying assumption that cells can be only rescaled by a factor of two. For instance, the maximum size used in the tests is 8x8, which can be rescaled in a grid 4x4 or 2x2. Under this assumption the algorithm, taking as input the old grid size S, merges 4 adjacent cells into one, starting from the bottom-left one. For clarity reasons the an example of such a reduction from a 4x4 to a 2x2 grid is provided in Figure 4.

As can be seen in Algorithm 3 the algorithm is really simple: it loops through all the cells of the grid and it updates topic using the old cell index (*cell_id*) and the new cell id of the resulting grid (*bias+i/2*). The rest of the algorithm is used to ensure the correct mapping between old and new cell id. Furthermore, in case grid and cell rescaling are performed together, a variation of Algorithm 3 with the addition of a constraint to limit computation to only the cells in which one is interested can be used.

Algorithm 3 Grid rescaling

```
1:  procedure UPDATE_CELLS (S)
2:      i ← 0
3:      bias ← 0
4:      for cell_id ← 0 to $s^2$ do
5:          update_topics(cell_id, bias+i/2)
6:          if i < s-1 then
7:              i ← i+1
8:          else
9:              i ← 0
10:             if x+1  mod (s · 2) = 0 then
11:                 bias ← bias + s/2
```

The `update_topics` function at line 5 takes care of updating topics for each method. This operation is not done by performing

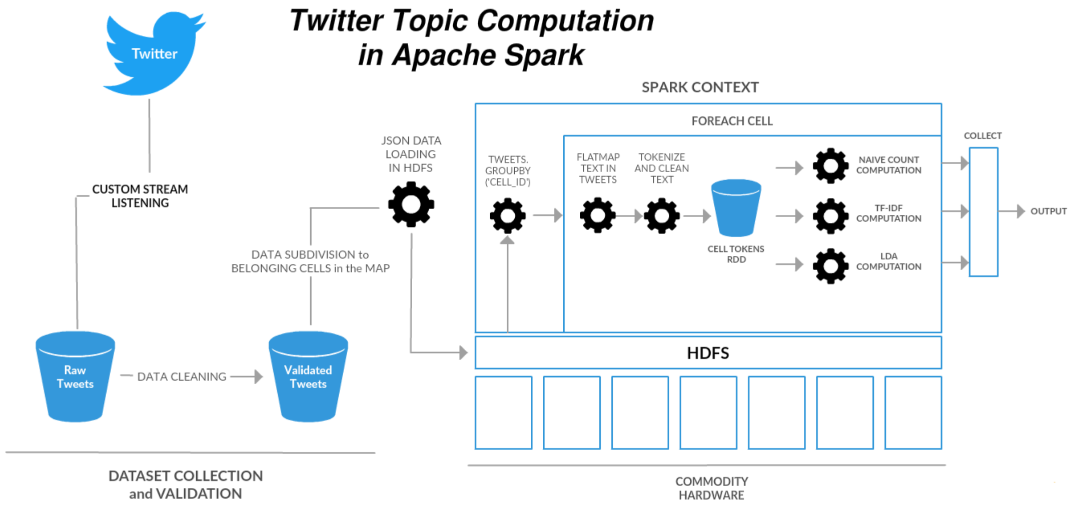


Figure 5: *Topic Computation in Apache Spark*

a re-computation, but, on the contrary, heuristic is used to make this step as fast and as computationally light as possible. For the standard topic extraction this was straightforward as they have build-in functions that allow update, while for the Geotag Cross-Document Ranking an update procedure had to be devised.

In this case, it was decided to base topic update on the average token rank over the documents to be merged. The rank in this case is simply considered as the position each token has on the sorted list of each cell. For instance, for a given cell, the first topic word will have rank 0, the second rank 1, and so on until topic $n - 1$, where n is the length of the sorted token list.

Algorithm 4 shows how the update is actually performed. As can be seen the auxiliary function `get_index` is used to return the position of the token. In case the token is not present in the list being checked, a value corresponding to the rank of the last token in that list plus 1 (i.e., the length of the list) is used to downgrade the rank

In this case ranks should be sorted in ascending order, as lower ranks belong to tokens that have scored higher in the original computation.

Algorithm 4 Geotag Clustering update

```
1: function GET_INDEX (TOKEN, LIST)
2:     if token ∈ list then
3:         return x.index(token)
4:     else
5:         return len(x)
6: function UPDATE(LISTS, TOKENS)
7:     token_rank ← {}
8:     for t ∈ tokens do
9:         rank ← 0
10:        for l ∈ lists do
11:            rank ← rank + get_index(t, l)
12:        token_rank.update(t : rank/4)
13:    sorted_tokens ← sort_by_value(token_rank, asc)
14:    return top(sorted_tokens, 10)
```

4.6 Distributed topic computation

4.6.1 Overview

While for a small dataset (around ~`36,000 lines` of text, `120MB`) like the one used for this experiment a single modern CPU can perform all the computations described in previous sections with relative ease, this is not entirely true if the data to be processed grow by several orders of magnitude in size and, therefore, speed.

An idea that could quickly arise would be to solve this problem by improving the hardware resources of the system that is in charge of executing the computations. But improving the resources of a system by choosing the highest performing components for it, helps only up to a certain point. It is necessary to find a way that would allow the system to grow indefinitely in computational capabilities, potentially following the size of the data toward infinity. To achieve the goal of building this kind of system, multiprocessing should be considered.

There can be several ways of constructing multiprocessor systems, but the distributed memory multiprocessing architecture seems a more flexible way of building the system compared to shared memory multiprocessing model. These distributed memory multiprocessing architectures are nowadays achieved by consumer-grade (commodity) hardware, interconnected with a high speed network. This allows to keep costs contained and to easily replace the faulty hardware, so that the actual hardware architecture is kept simple and the complexity of the system relies on software. But computing in a distributed system is significantly different from working with a single CPU and algorithms should be carefully crafted in order to exploit its whole potential.

As previously observed, there are interesting works in this field that allow the exploitation of these systems' potential. Surely one of the most successful ideas which is widely used in this area is the MapReduce Paradigm. In the MapReduce philosophy there are both a programming model and a framework for processing data in a distributed way. The programming model basically consists of two distinct steps that are customizable and performed in sequence: the **map** function and the **reduce** function.

- **Map**: a key/value pair is processed in an arbitrary way (our desired transformation function) to generate a set of intermediate key/value pairs
- **Reduce**: merges all intermediate values associated with the same intermediate key

The algorithms expressed in this functional style are automatically parallelized by the framework and executed on a large cluster of commodity machines.

This is possible because, even if the MapReduce interface exposed to the user is pretty simple, the actual implementation of the MapReduce framework backend hides away the hard details of parallelization, fault-tolerance, data distribution and load balancing. Having this powerful tool available it is possible to focus on the real solution of the problem rather than get lost in developing really complex mechanisms for data and computation distribution over the cluster.

So in order to find a solution to overcome the scalability problem, the topic computation algorithm was also developed using the MapReduce paradigm.

4.6.2 *Distributed Topic Computation Algorithm*

The adopted solution consists in an algorithm able to run over a cluster of computers, so that it is possible to distribute the workload between multiple CPUs. This allows a theoretically unlimited possibility of growth and scalability. The solution, shown in Algorithm 5, follows the steps described in the previous sections integrated with the MapReduce framework.

4.6.3 *Apache Spark Code Strategy Overview*

Given the fact that the described algorithm tries to solve the problem in a way that allow scalability over a distributed cluster of computers, in order to verify the effective correctness of the proposed solution an implementation of the algorithms described in the previous subsection was therefore developed for Apache Spark. In order to be consistent and better compare the two different types of implementations (the distributed and non distributed one) the Python language was chosen also for this task. Python in fact is one of the languages that are supported by the Apache Spark framework. The Spark ecosystem in fact provides a client library for Python called **PySpark**, with which is possible to write programs that exploit the MapReduce paradigm and that can be distributed by the Spark computation engine over a cluster of Spark workers.

The strategy of computation execution on the Spark Cluster is a key factor for obtaining a high-performance algorithm. In fact, even if an entire cluster is available, the operations have to be efficiently organized and coded in order to avoid waste of computational resources. This is even more important with huge quantities of data, because if there is a code inefficiency the repercussions of that inefficiency grow as the size of the data grow.

Algorithm 5 Distributed Topic Computation with MapReduce paradigm

```
function MAIN()
    loads a list of stop words from a file
    stop_words ← load_stopwords(stopwordfile)
    partition and load chunks of the file in each cluster's node with the help of the specific MapReduce framework api (e.g. Apache Spark: sc.wholeTextFiles() )
    dist_tweets_file ← dist_load('tweets_data.json')
    distributely parse each row of the file as json and load it in a list of tweet objects
    distributed_tweets ← map(func row: json(row), dist_tweets_file)
    group the tweets by geographical cell_id, which in MapReduce is expressed as shuffle and reduce by key(cell_id) the tweets (e.g. Apache Spark: rdd.groupBy(lambda x: x['cell_id']).map(lambda y : (y[0],list(y[1])))
    dist_geoclustered_tweets ← reduce('cell_id' , distributed_tweets)
    distributely compute relevant tweets' topics for each cell in the grid
    results ← map(func cell: compute_topics(cell) , dist_geoclustered_tweets)

compute topics for each single geographical cell
function COMPUTE_TOPICS(CELL)
    map a function on each tweet in the cell, that extract the text property from the tweet object and split it into words
    cell_tweets_words ← map(func tweet: tweet["text"].split() , cell)
    map a cleanup function on each word that clean it and remove stop words
    clean_tweets_words ← map(func word: cleanup(word) , cell_tweets_words)
    flatten the list of words in tweets to a simple list of words (e.g. Apache Spark: rdd.flatMap(lambda tweet: tweet)
    cell_words ← map(func tweet: flatten(tweet) , clean_tweets_words)
    ## NAIVE WORD COUNT COMPUTATION ##
    counts number of occurrencies of words in tweets to find out the most relevant topics
    mapped_words ← map(func word: (word, 1), clean_tweets_words)
    data is reduced in order to merge together the individual results
    term_counts ← reduce(func x,y: x + y, mapped_words)
    ## TFIDF and LDA COMPUTATIONS ##
    These computations are directly implemented with the help of the Apache Spark MLlib
    tfidf_cell_topics ← cell_tfidf_computation()
    lda_cell_topics ← cell_lda_computation()
```

Table 1: *Results of test on number of topics*

	5 topics		10 topics		20 topics		50 topics		100 topics	
	word	prob	word	prob	word	prob	word	prob	word	prob
Sample topic	madonna	0.006	auguri	0.013	pomeriggio	0.031	vedo	0.105	parte	0.140
	campiglio	0.006	parte	0.012	impestato	0.014	figlio	0.049	stagione	0.042
	freddo	0.005	oggi	0.010	storia	0.012	lago	0.036	dura	0.036
	iniziare	0.004	vite	0.007	puoi	0.011	mesi	0.026	nessuna	0.030
	lago	0.004	segui	0.007	stasera	0.011	piano	0.026	colori	0.028
	coglioni	0.004	fate	0.007	just	0.010	grost	0.024	pensionati	0.025
	chiama	0.004	trento	0.007	posted	0.010	associazioni	0.022	gian	0.023
	uscire	0.004	diretta	0.006	photo	0.010	spettacolo	0.018	sportiva	0.022
	delinquenti	0.004	tanti	0.006	associazioni	0.009	natura	0.017	cambiare	0.023
	parole	0.003	storia	0.006	crisi	0.008	questanno	0.015	unico	0.018
Time (sec)	~10		~15		~20		~30		~55	

5 EXPERIMENTS

In order to validate the solution presented in the previous section, several experiments have been run. Before this, however, some preliminary tests were executed in order to tune parameters and evaluate general performances of the algorithms. The aim of this section is therefore to analyze the purpose and the results of each preliminary or performance-driven test.

5.1 Preliminary tests

5.1.1 *Preliminary test 1: number of topics*

The first experiment aimed at understanding how the problem of the unknown number of topics discussed among the tweets influences the performances of the devised solution. To do this, it was decided to use LDA and compare the results obtained with different number of topics in terms of coherence and computation time. Table 1 shows the results of the LDA model computation with a number of topics equal to 5, 10, 20, 50, and 100. For each of the said number of topics, the table shows, for a sample topic, the top ten words with respect to the probability of each of these words, which is also listed in the table. As can be seen, the precision slightly increases with a higher number of topics, as shown by the probability of each word. However, from a human-evaluation point of view, the topics seem more or less equally coherent. This is not only due to the fact that LDA in not suitable to analyze tweets, but also to the low number of tweets collected.

Moreover, Table 1 also list the approximate time used by each LDA model computation. The value is only approximate since the actual time can vary from one execution to the other due to the fact that the LDA computation uses randomness in the training steps.

Since it is not possible to identify a better number of topics, for the following tests it was decided to set the number of topics equal to $k = \sqrt{\frac{n}{2}}$, which is a rule of thumb for choosing k, where n is the number of words in the model. Note that the maximum value that k can assume in the experiments at stake is 110, and therefore results are comparable to the ones of the last column of Table 1.

5.1.2 *Preliminary test 2: general performances*

A fundamental experiment is the one to aiming at understanding how suitable each of the used algorithm is for the task of recognizing the topics of the analyzed tweets. This is relevant with respect to the state-of-the-art. In fact the task at stake is peculiar and differs from the state-of-the-art as it aims at finding topics for relatively small areas as well as understanding how these topics vary from area to area. While the latter task will be discussed in Section 5.2, this section will only be devoted to analyzing results of the application the algorithms to the whole map.

The reason behind the choice of performing this test is to avoid the problem of the limited number of tweets that are found in each grid section.

This preliminary test does not include the novel Geotag Cross-Document Ranking method, because, for clarity reasons, it was decided to have a test devoted only to the validation of this new method.

The fist two methods applied to tweets were naïve word count and TF-IDF. For each of these algorithms the ten best results were then returned. The evaluation of tokens as already discussed is different for the two methods, the former algorithm uses a simple count of words, while the latter uses the ranking discussed in Section 4.4.2. Therefore, the results obtained with the two algorithms are close to opposite. Table 2 shows the results of the algorithms. As can be seen, no word appears in the top words for both algorithms. Moreover, a further test revealed that the first ten word of naïve word count are the last ten for TF-IDF.

From these data it seems that the naïve word count performs better in identifying a topic. For TF-IDF on the other hand is hard to find something that relates the words among them. On the contrary, for the naïve word count it is easy to identify the topic, that is coherent with the period in which topics were collected: Christmas. This is also clearly visible in the word cloud in Figure 6, which graphically show an extension of the results of Table 2 for naïve word count.

The second part of this test consisted in running, always for the whole map, LDA algorithm. This algorithm works well, presenting coherent topic with an associated word probability included between 0.4 and 0.05 for the first word of each topic. However, LDA is slower, as it takes around a minute to run for the whole map, while the first two methods are faster: 0.04 and 0.3 seconds respectively.

Table 2: *Results of naïve word count and TF-IDF on whole map*

Naïve word count		TF-IDF	
word	count	word	rank
buon	709	motivazioni	10
natale	573	lodare	10
grazie	548	orchidea	10
auguri	402	bottoni	10
oggi	276	peculiare	10
buongiorno	271	nazionalizzata	10
vita	231	four	10
fatto	222	seminario	10
casa	211	fotodomani	10
giorno	196	rinunciarci	10

Figure 6: *Word cloud representing most frequent words in the dataset*

Although not performing much worse than LDA from the computational time point of view (only around 15 seconds more), the LDA and K-means clustering algorithm was discarded due to its lack of improvement in topic quality. In some cases not only did this algorithm not improve topic recognition, but event worsen it.

This decrease in performance might be due to several factors. Among the most likely causes of this behavior is possible to list the following three.

(1) The low number of tweets on which perform clustering. In this specific case results obtained are worse than with classic LDA since this method is not only affected directly by a limited training set, but also indirectly by a poor quality of the cluster itself.
(2) The cluster quality, which might not only be affected by the lack of an adequate number of tweets on which to perform k-means, but also by the choice of the number of cluster and the parameters used to run the algorithm.
(3) The choice of the clustering method is another factor that might have bias the performance of the algorithm. Other clustering techniques such as the ones reviewed in Section 2 or other not yet tested might be more suitable to the case.

Nevertheless, the list of factors just provided is a mere supposition drawn up on the observation of the data and knowledge provided by the experiments and research revised in Section 2. Moreover, the three causes might also co-occur in the drop of performances of this algorithm.

5.2 Test 1: map rescaling

The main difference between this work and the state of art is the possibility to analyze topics with respect to each geographical area. As already said, it is in fact possible to select, within a map, a number of cells in which this can be divided by specifying a parameter S.

Therefore, it is vital to test how well the algorithm works when the size of S is changed during the computation. In order to do this, a test is run starting from a map divided into a 8×8 grid (see Figure 3). By using the algorithm described in section Section 4.5, the grid is then scaled into a 4×4 grid.

The algorithm used for this test is only LDA and count, as from the previous test it seemed that those worked better both for topic identification and in terms of computational time. Furthermore, map rescaling for Geotag Cross-Document Ranking is tested in Section 5.3, where the said method is validated.

The results obtained does not add much in terms quality of topics. Nevertheless, some consideration can be drawn. For instance, the Christmas topic was equally present in all the grid cells. Moreover, for smaller cells, such as the ones in the 8x8 grid case, this topic is also highlighted by the LDA algorithm.

From further test with different map size, the results obtained at the end of the computation are coherent from the ones obtained in the previous test.

A more interesting analysis concerns the computation time. This test is in fact crucial in order to match the request of the problem.

What emerged is that the algorithm is actually effective in updating the word count and the LDA model while the map is scaled. To give an idea, the first step, i.e. the computation of LDA and word count for each of the 64 grid cells took almost an hour, while merging the said result in the 4×4 grid took less than a minute.

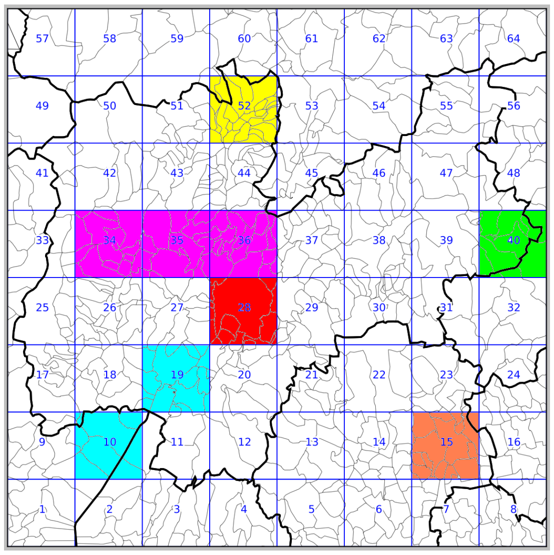
It is also possible to use the algorithm in an iterative way, by applying the algorithm recursively in order to move from a 8×8 grid, to a 4×4, to a 2×2 to the full map.

A similar reasoning can be done for the map rescaling, i.e. what happens when the user zooms in or out the map. The pre-processing phase in which topics will be computed for each grid cell has still to be done, not strictly at the same time, once for all the possible resizing of grid an map size. However, once this is done, map rescaling is performed in a matter of seconds.

This shows that the algorithm used to avoid re-computation of LDA and word count is highly effective.

5.3 Test 2: Geotag Cross-Document Ranking validation

This test aimed at validating the Geotag Cross-Document Ranking algorithm devised for the exact scope of improving performances in terms of quality of the output and computation time.



- ● `campiglio, madonna, trentinoalto, italy, adige, pinzolo - madonna, campiglio, dolomites, andalo, paganella, grost - rifugio, dosson, andalo`
- ● `italia, garda - garda, riva, lago, arco, italia`
- ● `ponte, legno, tonale, passo, tonale, presena`
- ● `bassano, grappa, centro, storico`
- ● `trento, piazza, duomo, mercatino, natale`
- ● `pieros, bakery, banana, crema, cacao, avena`

Figure 7: *Some interesting results obtained with Geotag Clustering algorithm*

The test consisted in running the topic identification algorithm described in Section 4.4.4 for each of the 64 cells of Figure 3, appraise the results obtained, and then also evaluate the results obtained by resizing the map grid.

The topics obtained from most of the grid cells are related to the Christmas festivity, as emerged also with the standard topic extraction algorithm. This confirms the initial hypothesis that the method should be used for only short periods of times in order not to distort the results.

For some cells, however, the result was very specific and related to the location. Figure 7 show some of the most interesting results that emerged. As can be seen, the among the top ten words that represent a topic, many are strongly related to the location from which tweets were sent as they either include the name of location, attractions or activities near the geographical location from which tweets were posted.

This result is even more satisfactory thinking that noise was introduced by randomizing the geotag position of tweets which are not provided with the position by the API.

Not only is this method very effective in topic detection, but is also very fast: less than two seconds to compute the topics for all 64 cell grids.

Map resizing proved again the efficiency and efficacy of the Geotag Cross-Document Ranking method. In fact, updating the values of a map scaling form a 8×8 grid to a 4×4 grid only took 0.005 seconds. The result is even better than the one obtained with naïve word count. This is also due to the fact that, as one would expect when the level of detail is removed by zooming out of the map, words that are too local are discarded.

Moreover, the fact that the tokens related to Christmas are still the top ranked one, showing that the algorithm works as expected.

5.4 Test 3: distributed computation

The last test consisted in the evaluation of performances of the algorithms in the case of distributed computation both as a standalone and with respect to the previous computations.

For the comparison, both Computation Time and Topic Computation Effectiveness are taken in consideration.

5.4.1 Time Comparison

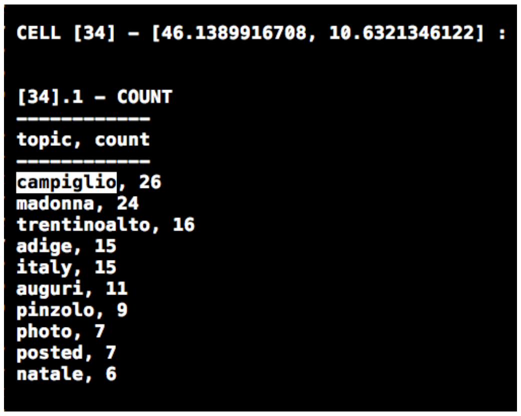
In terms of time, the distributed computations over Apache Spark have outperformed the ones in the pure Python implementation (up to x10 times faster) also with Spark running on a single machine. This, in our opinion, can be due to two facts:

(1) the computation running in the Spark context is highly optimized to work with data manipulation and the pyspark code that later runs in the Spark JVM is run more efficiently that with the python code interpreter.
(2) the computation that run in a cluster with multiple CPUs run naturally faster that the one that runs on a similar single CPU.

5.4.2 Effectiveness Comparison

The results obtained on the whole map in the Apache Spark implementation were the same as the one obtained for the non distributed algorithm for the Naïve word count and TF-IDF implementations, since they are a sort of deterministic operations. On the single cells the topic identification was pretty accurate. For example, in the Madonna di Campiglio cell the topics were identified correctly, given the fact that it is a ski area, as can be seen here Figure 8.

This is not the case for the LDA implementation, which in Apache Spark did not perform as well as in the Pure Python implementation. In the obtained results, for different cells, the topics were very similar. This may be due to the fact that there was some noise in the dataset because the data were collected during the Christmas festivity.



```
CELL [34] - [46.1389916708, 10.6321346122] :

[34].1 - COUNT
------------
topic, count
------------
campiglio, 26
madonna, 24
trentinoalto, 16
adige, 15
italy, 15
auguri, 11
pinzolo, 9
photo, 7
posted, 7
natale, 6
```

Figure 8: *Campiglio Topics Identification with Word Count*

6 CONCLUSION

All the methodologies analyzed in this report yield to rather satisfactory results. On the one hand, we discovered that, for this specific case, naïve word count and the classic LDA approaches are more suitable than other techniques used as a black box in order to understand which are the underlying topics discussed with Twitter in a set geographical location.

On the other hand, the most satisfactory results were obtained by using the Geotag Cross-Document Ranking method, that was developed ad-hoc to solve the problem at issue in this report. Not only did this perform better in terms of quality of the output, but also from a computational point of view. In fact it reduced by almost 1/30 the computational time for the recognition of topics for all the cells for the minimum value of S, while the time for updating the topics on map resize was reduced from a few seconds to some milliseconds.

The latter fact shows the validity of both the algorithms devised: the one to cluster tweets with respect to their position and the one to update the topics when the map is rescaled.

From a performance point of view in computation timing, the distributed implementations were undoubtedly faster (this may vary depending on the computing cluster) and potentially allow the treatment of much larger quantities of data. This is very important, especially in this field and considering the fact that twitter, but more generally social network data analysis implies very high and fast-growing dimensionality of data.

More generally, it was found that topic detection in tweets is a very challenging activity and the traditional techniques of topic computation over documents (LDA, TF-IDF) are not so effective on very short types of documents like tweets can be. It follows that more custom and tailored techniques, like the one we proposed (Geotag Cross-Document Ranking), can give much better results under these conditions.

REFERENCES

[1] Raffi Krikorian. New tweets per second record, and how. *Twitter Engineering Blog*, 16, 2013.
[2] Kevin Weil. Measuring tweets. *Twitter Offical Blog. Februari*, 22:2010, 2010.
[3] Twitter Blog. 200 million tweets per day. *Blog. twitter. com, June*, 30, 2011.
[4] Twitter Team. Twitter turns six. *Twitter Blog*, pages 21–3, 2012.
[5] Xiaojun Quan, Chunyu Kit, Yong Ge, and Sinno Jialin Pan. Short and sparse text topic modeling via self-aggregation. In *IJCAI*, pages 2270–2276, 2015.
[6] Jey Han Lau, Nigel Collier, and Timothy Baldwin. On-line trend analysis with topic models:\# twitter trends detection topic model online. In *COLING*, pages 1519–1534, 2012.
[7] Xin Wang, Ying Wang, Wanli Zuo, and Guoyong Cai. Exploring social context for topic identification in short and noisy texts. In *AAAI*, pages 1868–1874, 2015.
[8] Cindy Xide Lin, Bo Zhao, Qiaozhu Mei, and Jiawei Han. Pet: a statistical model for popular events tracking in social communities. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 929–938. ACM, 2010.
[9] Jianshu Weng, Ee-Peng Lim, Jing Jiang, and Qi He. Twitterrank: finding topic-sensitive influential twitterers. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 261–270. ACM, 2010.
[10] Owen Phelan, Kevin McCarthy, and Barry Smyth. Using twitter to recommend real-time topical news. In *Proceedings of the third ACM conference on Recommender systems*, pages 385–388. ACM, 2009.
[11] Wayne Xin Zhao, Jing Jiang, Jianshu Weng, Jing He, Ee-Peng Lim, Hongfei Yan, and Xiaoming Li. Comparing twitter and traditional media using topic models. In *European Conference on Information Retrieval*, pages 338–349. Springer, 2011.
[12] Rishabh Mehrotra, Scott Sanner, Wray Buntine, and Lexing Xie. Improving lda topic models for microblogs via tweet pooling and automatic labeling. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 889–892. ACM, 2013.
[13] Mark Steyvers, Padhraic Smyth, Michal Rosen-Zvi, and Thomas Griffiths. Probabilistic author-topic models for information discovery. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 306–315. ACM, 2004.
[14] André B Bondi. Characteristics of scalability and their impact on performance. In *Proceedings of the 2nd international workshop on Software and performance*, pages 195–203. ACM, 2000.
[15] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, 2004.
[16] Satish Gopalani and Rohan Arora. Comparing apache spark and map reduce with performance analysis using k-means. *International Journal of Computer Applications*, pages 0975–8887, 2015.
[17] Json notation. http://www.json.org/json-it.html. Accessed: 2017-01-20.
[18] Steven T Piantadosi. Zipffis word frequency law in natural language: A critical review and future directions. *Psychonomic bulletin & review*, 21(5):1112–1130, 2014.
[19] George Kingsley Zipf. The psycho-biology of language. 1935.
[20] Logan Wilson and George K Zipf. Human behavior and the principle of least effort, 1949.
[21] Jeffrey D Ullman, Jure Leskovec, and Anand Rajaraman. Mining of massive datasets, 2011.
[22] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.
[23] Matthew Hoffman, Francis R Bach, and David M Blei. Online learning for latent dirichlet allocation. In *advances in neural information processing systems*, pages 856–864, 2010.