Event summarization from news and tweets correlation

Cavallari Sandro
Giglio Marco
Morettin Paolo

ABSTRACT

In this report we consider the task of extracting a summary of the main event that caused a shift on the opinion of Twitter users. This paper presents several techniques which might be used to analyze these *sentiment shift* and to find the event which caused them. For each technique presented below some aspects are taken in account, which are important in the context of a data mining application, such as scalability and efficiency. A comparison of the different techniques is showed as well.

Introduction

This work aims to find a summary of the events which caused a *sentiment shift*.

While event extraction NLP techniques are mature, their performance on tweets inevitably degrades due to the inherent sparsity in short texts. Since tweets contain heterogeneous language structures and are at most 140 characters long, to extract event from tweets is quite tricky: instead linking tweet to news allows us to extract events from news text that has a well formed structure and contains more text with respect to tweets.

Moreover NLP document compression techniques usually exploits language dependent methods, whereas we want a methodology as language independent as possible.

This paper is structured as follows: in the next chapter we present some works which are somehow related with the one we are addressing, then we present a more formal definition of our problem and three different techniques which might be used to solve it (two of this techniques are based on the *bag of words* approach - one exploiting the *SpaceSaving algorithm* presented in [7], the other exploiting *Latent Semantic Indexing* [1] - , whereas the third one exploits *ngram graphs* [3]). After that we will show and compare the results and the performances which characterize each of these metodologies. At the end of the paper we will present some conclusions and draw some suggestions on how this work may be extended.

Related Work

We can distinguish two main subgoals in our project:

- compute the correlation between tweet and news
- create a summary of the main event that caused the sentiment shift

There are several works, in literature, which refer to similar tasks. In particular, due to Twitter's increasing popularity, in the last years there have been several works regarding the correlation and analysis of the data. Weiwei Guo et al. [4] proposed a framework to link tweets and news and to extract from the resulting correlation missing aspects of the tweet-event. In their research, rather than using a LSI technique, they propose a methodologies based on the Weighted Textual Matrix Factorization [5] [WTMF] model(the 2012 de facto standard). This approach revealed to return good correlation results, making WTMF a strong tool for baseline creation.

Some works which are complementary with respect to our might be found in the field of recommendation systems; in particular systems which address the problem of recommending news given some user's features. However, there is an important difference between this task and our, since tweets are much more heterogeneous and characterized by a much smaller length than news.

Recently *Google* presented a system for *multi-sentence compression* (MSC) [6], which aims to compress several senteces trying to mantain high readability and to not lose the semantic content of the text. Their approach explits word gram graphs, combining them with some semantic data. The combination of the two approaches allows redundancy removal while mantaining a readable sentence as output.

Problem Definition

We define *sentiment* the measurement of feelings performed using specific tools analyzing the stream of tweets. This kind of measurement is usually done using some supervised machine learning techniques together with NLP techniques.

Tools exist to track the *sentiment* toward a certain topic and to identify changes in it. In particular we can consider three major events in a sentiment timeline:

1. there is a change of sentiment from positive to negative (or viceversa)
2. there is no change of sentiment, but there is a peek in the graph describing it or its value crosses the average value
3. there is no change but in the volume of tweets/second

We define the first of these events as *sentiment shift* and the temporal window in which it happens is called *contradiction point* or *contradiction window*.

In addition we define *contradiction tweets* and *contradiction news* respectively the tweets and news published inside a contradiction window and *background tweets* and *background news* the set of all tweets and news regarding the addressed topic.

Given these definitions, the problem we aim to solve is to find, among the contradiction news, the ones which are the more representative of the contradiction tweets and to present to the user a summary which describes them.

Proposed Approach

In this section we present different approaches which might be used to solve the problem we are addressing. Of course, each approach has advantages and disadvantages which we will list while showing them.

SpaceSaving approach

Description of the algorithm

SpaceSaving [7] is an algorithm which was first presented by Metwally et al. in 2005 to efficiently compute the most frequent terms in a data stream. It allows the user to find k words which are among the most frequent in the given stream. Although this is a heuristic algorithm whose result's correctness is not guaranteed, SpaceSaving is able to specify the upper bound of the error for each word presented as output.

The classic implementation is based on a fixed size list of tuple

$$(word, occurrencies, error)$$

The stream is read word by word and at any time one of the following condition is verified:

1. the word is already in the list of frequent terms
2. the word is not among the frequent terms, but the list is not full
3. the word is not among the frequent terms and the list is full

If the first condition is verified, then the algorithm will proceed incrementing the number of occurrencies of that term; if condition 2 is verified, instead, we add that word to the list of frequent terms with number of occurrencies 1 and number of errors 0; if condition 3 is verified, then we scan the list for the term with the lowest number of occurrencies and replace it with the new word. When this replacement takes place we set the number of errors equal to the number of occurrencies of the world we just replaced, then we increment the number of occurrencies.

Our python implementation of this algorithm is slightly different from the classic one described above, since we test the word against a list of stop words and we added a hashmap to avoid useless replacement. In our implementation, then, the following steps are performed:

1. if the word is in the stop word lists, does nothing
2. otherwise compute the hash of the given word
3. increment the value associated to that hash
4. check whether the value stored in the hash map is above the number of occurrencies of the less frequent term tracked by the SpaceSaving algorithm. If so, the replacement occurs, otherwise nothing happens.

The output of this algorithm is a couple $(word, value)$ where the value is computed as

$$value = occurrencies - error$$

In order to reduce the noise we chose to use only couples where the value is above a certain threshold.

Description of the methodology

We have seen how we exploit the SpaceSaving algorithm to obtain the list of frequent terms in a text and their values.

To accomplish our task, we ran the program described above on the set of tweets inside a *contradiction pint*[CP] (point in which a sentiment shift occur), thus obtaining the list of frequent terms inside the contradictions; then, we read all the news which were published during the contradiction (a certain error between the publishing date and the contradiction window might be taken in account) and for each of them we compute its score as

$$newsScore = \sum_{w \in W} value(w)$$

where W is the list of words in the news and the value is null if that term is not in the *frequent term list*.

The news with the highest score is selected as the one causing the shift and its first words are presented as output to the user.

This approach revealed to be very fast and from its formulation we can see that it can be applied incrementally. As a drawback, we must say that it does not take in account the background coming from the topic, hence it is not able to discern whether a given frequent term is informative or not (e.g. running this program on the Obama topic it is likely that the most frequent terms will be "President", 'Barack", "Obama", "USA", hence

a news containing more repetitions of these words will have a high score, even if those words are not informative at all)

LSI

We used a bag-of-words approach exploiting a technique called Latent Semantic Analysis (also known as Latent Semantic Indexing), formalized by Scott Deerwester et al. in 1990 [1]. LSA is a statistical approach, used to find semantical similarities between sets of documents. It takes advantage of Singular Value Decomposition on a vector space of term frequencies in order to create, given a set of documents, an *index* matrix, that can be queried with other documents to find the most similar one. In order to find correlations between the sentiment shifts and the news, our approach works as follows:

- All the tweets corresponding to a CP are merged in a single document.
- Every document is preprocessed and converted in the bag-of-word representation.
- The LSA model is trained on the whole set of news for the given topic.
- The set of candidate news to be tested is chosen accordingly to a window, and it's used to create an index.
- The news are sorted in order of similarity, optionally weighted by a factor considering the time distance of the news from the CP.
- All the candidate news over a certain threshold are considered to be correlated and a set of words calculated with TF-IDF is returned.

Preprocessing

During the preprocessing, every document is filtered and converted from a string to a bag-of-word representation. The following steps are done in this phase:

1. (Optional) URLs removal using a regular expression.
2. (Optional) conversion from Unicode to ASCII.
3. The punctuation characters are substituted with a whitespace.
4. The string is tokenized.
5. Stopwords and tokens appearing less than t times in the document are removed.

LSA and candidate news selection

Once the documents are filtered and transformed in a bag-of-words representation, a dictionary is created containing the association between tokens and a positive integer identifier. This is done in order to convert on-the-fly each document D in a vector of tuples $(i, n(D, i))$ where $n(D, i)$ is the number of occurrences of the token i in the document D. The LSA model is then trained with all the news available for the given topic. Given a sentiment shift over a time interval $[T_b, T_e]$, the news can be selected using a fixed size window $[T_b - c, T_e]$ or a more sophisticated approach that considers the density of the tweets inside the shift. The latter seems reasonable since news that correlates with a shift are likely to be near the shift the more the *hype* increases.

News scoring and summarization

With the candidate news a matrix index is then created and the document representing the shift is compared. This operation produces a vector of similarities of the shift with the candidate news. These scores do not take into account the hype and the temporal distance of the news w.r.t. the sentiment shift. Unfortunately, we didn't had time to investigate this aspect and find a reasonable weighting function on those parameters. The candidate news passing a threshold are considered to be correlated with the shift, so for each of them the list of the k words with higher TF-IDF values are returned.

Implementation

The methodology described above was implemented in python 2.7 using a library called gensim [2]. Even if the time complexity is theoretically $O(2^m)$ for the compilation of a regular expression of length m, in practice step 1 is very fast due to the simplicity of the one matching the URLs. Every other step takes linear time w.r.t. the document length. Steps 2 and 3 take linear time w.r.t. the number of characters in the document, steps 4 and 5 take linear time w.r.t. the number of words in the document. Every document is preprocessed at runtime and stored in main memory

N-Gram Graph

N-Gram Graph is a NPL tool initially proposed by George Giannakopoulos [3] that use word or character n-gram in order to achieve documents summarization. The NGG tool basically slice the text in word or character n-gram and then represent them in a graph $G = \{N, E, L, W\}$ according the following structure:

- N is the set of nodes created for every different n-gram in the text
- E represent the edge of the graph; where two node are connected by an edge if they are "near" according to a window-distance
- L is a labelling function which give the label to every node and every edge (define the size of the n-gram)
- W is the weight function that gives weigh to every edge according to the number of times that two n-gram appear near one to the other

Whit this implementation NGG result to be a tool with the property to be language-neutral (make no assumption about the underlying language) and allow text manipulation trough graph operations.

In particular two operation are necessary for our goal:

- the **Intersection** operator between two graphs G_1 and G_2: which returns a resulting graph with only the common edges of G_1 and G_2 averaging the weights of the

original edges assigned as the new edge weights(example in figure 1)

- the **Normalized Value Similarity**[NVS] function that for every n-gram rank, indicating how many of the edges contained in graph G_i are also contained in graph G_j, considering also the weights of the matching edges and normalize the result respect the graph size

In particular $NVS(G_i, G_j) = \frac{VS(G_i,G_j)}{SS(G_i,G_j)}$ where:

$$VS(G_i, G_j) = \frac{\sum e \in G_i \frac{\min(w_e^i, w_e^j)}{\max(w_e^i, w_e^j)}}{\max(\mid G_i \mid, \mid G_j \mid)} \quad (1)$$

$$SS(G_i, G_j) = \frac{\min(\mid G_i \mid, \mid G_j \mid)}{\max(\mid G_i \mid, \mid G_j \mid)} \quad (2)$$

Since NGG tools allow to compute both the tweet-news correlation and summary creation, this methodology is split in different section.

0.0.0.1 Tweet-News Correlation

To obtain a correlation between contradiction tweet and news, this methodology use the base idea of exploiting the NVS similarity function as a correlation function: higher the similarity of the tweets n-gram graph representation and the news n-gram representation, higher will be the correlation.

More in detail this procedure start reading all the contradiction tweet from the json file, concatenate the tweet text it in a unique string and compute the n-gram graph representation of the concatenated single.

Due to the absence of memory lack problem, we prefer to decrease the computation time avoiding the usage of the scalable merging function proposed by George Giannakopoulos. The merging function based on a learning factor produce the same final result, but need a n-gram graph for every single tweet witch is more expensive in terms of computation time. It is strongly suggested to avoid the usage of intersection function over tweets, since tweet text contains many different language pattern determine a big chance to obtain an empty intersection graph (remember that intersection maintains only the common n-gram between the two document).

After computing the tweet graph, for every news is computed the NVS value between the tweet and news graphs. Since the NVS values have no time domain impact, a time windows is defined with the purpose of discharge from the correlation computation news that are distant in time. The correlation is computed only for news that were published in time slots defined by the contradiction point windows increased on both side by the time windows duration. The combined usage of a the truncating time windows and the NVS similarity value produce the correlation value that we look for our first goal.

Experimental Evaluation

ExperimentalEvaluation

Conclusions

1. REFERENCES

[1] Indexing by Latent Semantic Analysis, Scott Deerwester , Susan T. Dumais*, George W. Furnas, Thomas K. Landauer and Richard Harshman

[2] ŘEHŮŘEK, Radim, et al. "*Software framework for topic modelling with large corpora*". 2010.

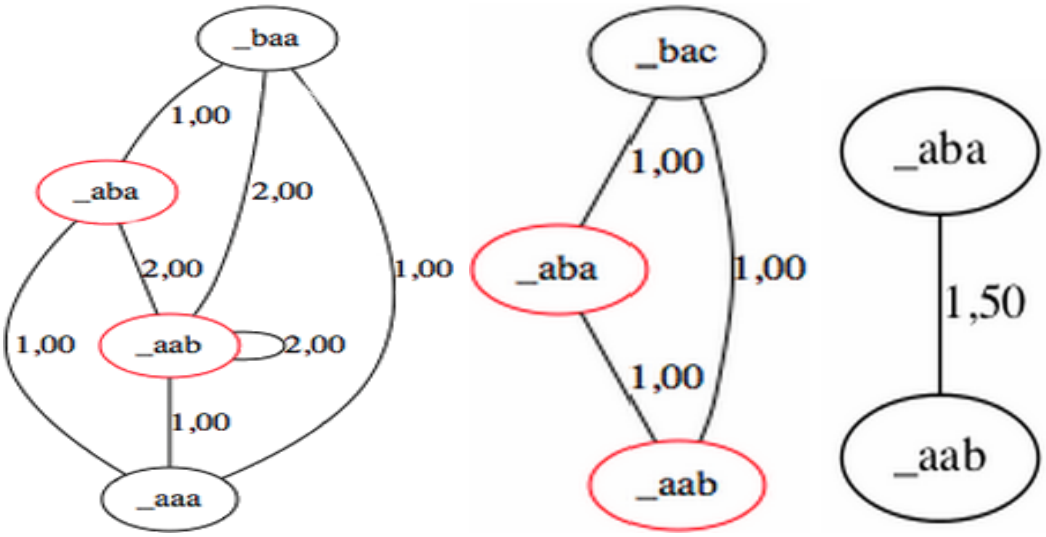
[3] Automatic Summarization from Multiple Documents : N-Gram Graph, George Giannakopoulos and Ncsr Demokritos, 2009

[4] Linking Tweets to News: A Framework to Enrich Short Text Data in Social Media, Weiwei Guo, Hao Li, Heng Ji and Mona Diab

[5] Modeling Sentences in the Latent Space, Weiwei Guo and Mona Diab , 2012

[6] Multi-Sentence Compression: Finding Shortest Paths in Word Graphs, Katja Filippova

[7] Metwally A. et al, "*Efficient Computation of Frequent and Top-k Elements in Data Streams*", Lecture Notes in Computer Science Volume 3363, 2005, pp 398-412



(a) G1="aaabaab" (b) G2="aabac" (c) Intersection Result

Figure 1: An example of the intersection operation