Twitter Trending Topic Classification

MES: Maria Ludovica Costagliola - Emanuele De Santis - Serena Ferracci

Abstract— The Web Information Retrieval course allowed us to develope the described project. It is about classification of topics that are trending of twitter in a given moment. We have dealt with (i) text based classification using single tweets (ii) network based classification exploiting the graph structure of Twitter.
The used dataset is composed by tweets divided by trending topics. We classify the trending topics into 8 categories: *Event, Health, Movie, Music, Politics, Science, Society* and *Sport*.
The first approach gives good results in terms of precision and recall. Instead, the second approach does not give conclusive results because it requires too many resources to be implemented on a single machine.

1 Introduction

We chose this project because Twitter is one of the most popular social network. It is used every day by millions of users expressing their opinions about several fields. When an user looks for something, the first thing Twitter displays to him is the list of trending topics of the moment. Often the user can not know what the topic is about, so it has to manually search for tweets belonging to that trending topic to better understand it.

It is interesting for the user to have a way to know what the Trending Topic is about without further searches. Our work tries to replicate the results obtained by [4]. The general categories used in this project are 8, named: *Event, Health, Movie, Music, Politics, Science, Society* and *Sport*.

2 Related Work

The paper we refer to is the work done by Lee et al. for tweets classification [4]. They have used 18 general categories to classify each trending topic. They address the problem following two different approaches. The first one is a supervised learning technique, named Multinomial Naive Bayes. Instead, the second one is network based and it uses Personalized PageRank to compute the top-k influencers and then it computes the intersection between the influencers to classify the new trending topic.

All trending topics they used were downloaded from *what the trend*.

3 Dataset

Since it was not possible to get a suitable dataset for our purpose, we needed to build it manually using Twitter APIs [2] through Tweepy [3].

In `tweets_retrieve.py` we first retrieved trending topics from USA using USA WOEID (Where On Earth IDentifier, that is 23424977 for USA). Then, for each trending topic, we queried Twitter to get some tweets belonging to the given topic. We pay attention to retrieve the entire text of the tweet; in case of retweeted statuses we consider only the retweeted text. Twitter imposes some limitation on GET requests, in particular we could retrieve 180 tweets every 15 minutes. So, we have to handle exceptions fired due to the reached limit. In order to submit a request we have to set up Tweepy authorization handler with our consumer key and our access token we take from our twitter developer account. As result, we get a file for each tweet where its name is the ID of the author and the body is the text of his tweet. This file is placed in a folder whose name is composed by the name of the trending topic. At the end, we collect about 20.000 tweets belonging to 139 trending topics.

After this first phase, we had to retrieve the link structure for the users that wrote the tweets collected in the

previous phase. In order to do so, we developed two python files, named `followers_retrieve.py` and `friends_retrieve.py`. We used Tweepy to retrieve all followers and friends for each author ID being aware of all exceptions. Since the limitation, in this case, is 15 queries every 15 minutes, we had to find a way to speed up the process. To do so we decided to parallelize the computation using more than one key (precisely 6 keys), contrarily with respect to the previous retrieving phase, and working in data separation. We associated each key to a subset of IDs to be processed, this mapping was done using a simple hash function. We saved the list of followers and the list of friends for an user x in the files named `x - followers.txt` and `x - friends.txt`. These two files are saved in the same folder that contains the tweet tweeted by x.

We faced three main problems to retrieve the dataset:

- some users changed their privacy settings between the first and the second phase of our retrieving. For this reason we couldn't get neither their followers nor their friends.
- a subset of users have a plethora of followers, of the order of millions of users. It took a lot of time to complete the retrieving of their followers, sometimes almost an entire day at full capacity. It also happened that the execution stopped due to connection problems or other unpredictable events, so it was not possible to retrieve the entire list of IDs.
- there wasn't a retrievable ground-truth for the trending topics we collected.

The last problem was the only one that we could manage to solve.

We tried to automatically get a classification using some pretrained machine learning algorithms, but when we manually checked the classification done, we saw that almost all the predictions were wrong.

The solution that we came up with was to label every trending topic by hand. We took one directory at a time, we read some tweets to understand what the topic was about, we assigned to the directory a single category among the 8 defined.

4 Classification using Naive Bayes

Multinomial Naive Bayes is a very useful Machine Learning algorithm when we are dealing with text classification. Naive Bayes is based on the assumption that the probability that a document (composed of terms $(t_1,...,t_T)$) belongs to a class c_k , namely $P(t_1,...,t_T|c_k)$, can be written as $\prod_{i=1}^{T} P(t_i|c_k)$. The probability that a term is in a class is conditionally independent from the probability that another term is in the same class. Thanks to Bayes rule we can write $P(c_k|t_1,...,t_T) \propto P(c_k) \cdot \prod_{i=1}^{T} P(t_i|c_k)$.

So Naive Bayes classifier makes a prediction $\hat{y}$ using the following formula

$$\hat{y} = \underset{k\in\{1,...,K\}}{\arg\max}\ P(c_k) \cdot \prod_{i=1}^{T} P(t_i|c_k)$$

We rearranged the data in the dataset in order to have a folder that contains 8 subfolders (one for each category we considered). Each one contains all the tweets belonging to a trending topic that belongs to the category of the subfolder. This operation is done by `training_set.py` python script.

We developed also `naive_bayes_no_stemming.py`. It takes as dataset the folder yet mentioned.

`sklearn` allows to make an automatic division of the dataset, splitting it into train set and test set (25% for the test set). The training set was converted first computing the corresponding tf-idf matrix through the `TfidfVectorizer` function. This is then used to actually transform the original dataset into the term-document matrix. The term-document matrix is used by the Multinomial Naive Bayes classifier to execute the learning process.

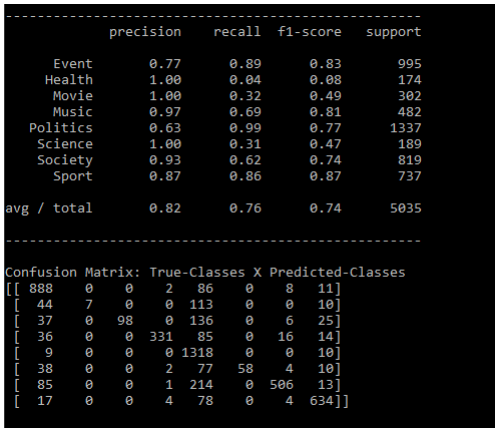
After the learning phase, we needed to evaluate the performances of the classifier: we applied the same transformation as before to the test set and we made the algorithm predict these instances. We get at the end the following results:

```
                precision    recall  f1-score   support

         Event       0.89      0.91      0.90       999
        Health       0.90      0.89      0.90       171
         Movie       0.76      0.90      0.82       279
         Music       0.91      0.92      0.91       483
      Politics       0.94      0.94      0.94      1302
       Science       0.88      0.88      0.88       201
       Society       0.87      0.81      0.83       809
         Sport       0.94      0.91      0.93       791

avg / total       0.90      0.90      0.90      5035

Confusion Matrix: True-Classes X Predicted-Classes
[[ 913    4    8    9   11    7   41    6]
 [   2  153    0    2    5    0    7    2]
 [   3    2  251    3    7    1   11    1]
 [  14    1    2  444    8    1    8    5]
 [  11    3   26    1 1229    3   15   14]
 [  10    3    0    2    1  176    6    3]
 [  57    4   28   15   34    8  652   11]
 [  13    0   16   14   10    4   13  721]]
```

Fig. 1. Classification evaluation without applying stemming

We tried, also, another version of the program, `naive_bayes_stemming.py`, where we used the stemming technique: we tokenized each word, applying

an `EnglishStemmer` from `nltk` [1]. Making the vectorization phase together with stemming, we get the results shown in 2.



```
                    precision    recall  f1-score   support

         Event       0.77      0.89      0.83       995
        Health       1.00      0.04      0.08       174
         Movie       1.00      0.32      0.49       302
         Music       0.97      0.69      0.81       482
      Politics       0.63      0.99      0.77      1337
       Science       1.00      0.31      0.47       189
       Society       0.93      0.62      0.74       819
         Sport       0.87      0.86      0.87       737

avg / total       0.82      0.76      0.74      5035

Confusion Matrix: True-Classes X Predicted-Classes
[[ 888    0    0    2   86    0    8   11]
 [  44    7    0    0  113    0    0   10]
 [  37    0   98    0  136    0    6   25]
 [  36    0    0  331   85    0   16   14]
 [   9    0    0    0 1318    0    0   10]
 [  38    0    0    2   77   58    4   10]
 [  85    0    0    1  214    0  506   13]
 [  17    0    0    4   78    0    4  634]]
```

Fig. 2. Classification evaluation applying stemming to the training set

As we can see from 1 and 2, we get lower results using stemming technique. Usually, stemming reduces the size of the particular vocabulary in order to increase performances of the classifier. In our case, this didn't happen since our tweets are written not only in English, but also in Spanish. So, it's not possible to apply this preprocessing technique since it bases its computation according to the language of the target vocabulary, that must be unique.

5 Network Based Classification

For this second approach, we exploited the structure of the social network. We assumed that if there is a significant overlap among users generating tweets on two topics, then it implies a close relationship between these two topics. Network based data modeling uses the categories that are manually labeled in the previous approach to predict the category of a new trending topic.

To compute the network based classification, we first had to construct the directed graph, that represents a small subset of the Twitter's network. In the directed graph every node represents an user and every edge represents a relationship between two users. If a node has an incoming edge, meaning it is followed by another node, we call the last one follower. Instead, if a node has an outgoing edge, meaning that it follows another node, we call the last one friend.

We build this graph using the data retrieved from Twitter: the name of the file containing a tweet is the ID of the user that wrotes it. Together with this file we have the list of the followers and of the friends of this user saved in two different files. So we first add the user ID in the graph, then we read these two additional files to add all the friends and followers IDs and to create the corresponding edges.

The original idea to perform the network based classification was to construct a single big graph for all the dataset, putting together all the users, and their friends and followers, of every category.

We first train the algorithm with a distinct graph for each category: we compute PageRank on these graphs to get the top-k influencers of each category, namely $C_t = \{i : i \in pr(k, c_t)\}$, where $pr(k, c_t)$ is the PageRank of top-k influencers run on the graph of category c_t.

To classify new instances we have to add the new trending topic IDs (so all the IDs of the users that wrotes a tweet in that trending topic together with all the followers and friends) to the single big graph mentioned above and then compute Personalized PageRank with personalization vector $\hat{P}$ such that

$$\hat{P}_i = \begin{cases} 0, & \text{if } i \notin \text{ new trending topic writers} \\ 1, & \text{otherwise} \end{cases}$$

In this way we can see all the users that are "near" to that new trending topic. We define an importance score as:

$$S_{c_t} = \sum_{i \in C_t} pr(k)_i$$

where C_t is the set of top-k influencers for class t and $pr(k)$ is the PageRank vector composed only of the first k influencers (users that had the highest PageRank score) run on the single big graph.

We can classify the new trending topic taking the top-k influencers and selecting the class that has the highest score.

$$\hat{y} = \arg\max_t S_{c_t}$$

We could not put this idea into practice because the graph reached a dimension that was too high to be managed by a single machine. To mitigate the problem, we decide to split the graph in as many components as the number of categories.

To construct the described graphs, we developed the python file `pagerank_learning.py`, in which we used the predefined library `NetworkX`. The library offers to the programmer several functions that facilitate the construction of the graph. Once the graph is finalized, we compute the PageRank using the function given by

the library itself. This function takes as parameter only the graph defined above and returns a dictionary of nodes with the PageRank score as value.

The purpose of this computation is to find the top-k influencers for each category. The top influencers are the ones that have the higher PageRank value. For every category, the program runs independently from previous executions. Each time we save in a distinct file the top-k influencers of that category. The name of the file is `topk.txt` and it is placed in the subfolder of that particular category in the training set folder. As soon as the retrieve of the influencers for each category is finished, the training is completed.

In order to perform the classification of a new trending topic we had to construct a distinct directed graph also for the new topic. The procedure is the same as the one described above. Once the graph was built, we executed PageRank on it. As result we had the top-k influencers of this particular trending topic.

At this point, we took the influencers computed in the training phase and the influencers computed in the last phase and we computed the intersection between the top-k influencers of the new topic and all other sets of influencers. The intersection will contains all the users that are influencers both in the new trending topic and in the analyzed category. Once we had completed all the intersections, we took the class which intersection has biggest cardinality as classification for the new trending topic.

It was not possible to obtain practical results, even after having applied the above modifications to the original idea, because the memory of a single machine was not sufficient to keep the graph of almost all classes and apply PageRank.

6 Results

In this project, we used two different classification approaches for Twitter trending topic classification. We used text-based classification and we tried to exploit the social network structure rather than using just textual information, which can lead to incorrect result due to the limit on the number of characters that users are allowed to use for their messages and the use of Twitter slang.

Analyzing the Naive Bayes approach, our results show that Naive Bayes without stemming performed better than Naive Bayes with stemming on our dataset. Considering retrieved tweets are not written only in English or in some cases using the regular English (the users use particular slangs or abbreviations), Naive Bayes using stemming provides worst results.

Instead, analyzing the network based classification, we were able to obtain only theoretical results. This problem is not due to errors in the proposed solution, which has been tested on a smaller dataset, but to the lack of memory available on a single machine. The original idea was to construct a single graph for all the dataset and then compute a personalized PageRank for each defined category to find the top-k influencers. To mitigate the memory problem, we decided to build a graph for each category and then compute the PageRank on these ones. Unfortunately, the graphs were still too large to make a single machine calculate the PageRank.

References

[1] Nltk 3.3 documentation. `https://www.nltk.org/#`.

[2] Standard search api - twitter developers. `https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets`.

[3] Tweepy documentation. `http://docs.tweepy.org/en/v3.5.0/`.

[4] K. Lee, D. Palsetia, R. Narayanan, M. M. A. Patwary, A. Agrawal, and A. Choudhary. Twitter trending topic classification. *2011 IEEE 11th International Conference on Data Mining Workshops*, 2011.