Using Social Media to Enhance Emergency Situation Awareness
Web Information Retrieval
Group: Kernel_Panic()

Andrea Mastropietro
1652886
Sapienza University of Rome

Umberto Mazziotta
1647818
Sapienza University of Rome

Abstract

The aim of our project is the categorization of disater-related tweets. Our work is divided into 3 tasks. Firstly, we classify disaster-related tweets into predefined classes, based on the work by (Stowe et al., 2016). Secondly, we perform bursts detection (Fung et al., 2005) in order to understand which tweets are more likely to be related to hazard events and finally, among those tweets, we perform an online clustering in order to gather tweets into groups, basing this part on the paper by (Yin et al., 2012). The results of classification and clustering can be used by emergency managers to understand how people are reacting during a disaster situation in order take proper actions both during and after the emergency[1].

1 Datasets

For our experiments we used 2 datasets from Stowe et al. consisting of tweets collected during Hurricane Sandy period, that hit New York in 2012. For the classification we have a labeled dataset of 5821 tweets (the original dimension was 7490 but some tweets are no longer available due to privacy and account deletion). The second dataset is an unlabeled dataset used for burst detection and clustering which original dimension was of 6554744 tweets, but we retrieved only 1003864. Since we only have the IDs of tweets for privacy reasons, we performed the retrieval of the text by using Twitter APIs. In the labeled dataset a tweet that is relevant for the event if it is annotated with one or more classes among *Actions*, *Sentiment*, *Preparation*, *Reporting*, *Information* and *Movement*. A non relevant tweet is annotated with *None*.

[1]Source: github.com/AndMastro/EmergencyAwareness.git

2 Classification

We trained a binary classifier for relevance classification and for finer-grained classification we trained 6 binary classifiers, one for each classes.

2.1 Model selection

In the paper by Stowe et al., they employed an SVM with unigram counts as baseline features. In our work, we tried to improve performances using a linear SVM with tf-idf after performing a preprocessing by removing stop-words and capitalization and by stemming words. We used a grid search to select the best classifier among other models (e.g SVM with RBF kernel) and to perform parameter tuning. We selected the classifier which yielded the highest F1 score on the relevance classification, i.e an SVM with linear kernel with tf-idf, unigrams, stemming (we used Porter stemmer) and minimum document frequency of 1 or 3 (it is not so relevant).

2.2 Features and feature selection

As aforementioned, we decided to use as baseline features tf-idf values. Furthermore we also used as features:

•Date: We encoded the date as a one-hot feature vector.

•Word Embeddings: this is an NLP unsupervised technique that encodes words as vectors of real values by capturing important information among words and their relations. We used a pre-trained Word2Vec model by Stanford and a model trained by us on the unlabeled dataset; both models

yielded the same results. In those models every word is encoded as a 200-entry vector. For each tweet we obtained a feature vector of 200 feature by taking the mean embedding of all words. We also tried to use the sum but it gave slightly worse results.

In the paper they worked also with some more features (such as for example the URLs in the tweet and the annotation if the tweet was a retweet or not), but, since those features where not particularly meaningful for the classification, we decided not to take them into account.
Differently from the solution proposed in the paper, we performed feature selection not using mutual information but we employed SVD and PCA. We used SVD with the tf-idf vectors in order to get rid of redundant data. We reduce the dimensionality of our features vectors from roughly 800 to 500 and, to reduce the dimensionality of Word Embeddings vectors, we employed PCA (Principal Component Analysis). With PCA we reduced the dimensionality from 200 to 150.

2.3 Classification results

Firstly we performed a 5-fold cross validation to see the mean performance of our classifiers and then we split our dataset into training set (75%) and validation set (25%) to perform the evaluation and compare our results with the ones in the paper. In the following table we also show the number of training samples for each class.

2.3.1 Baseline features

	Our model			Paper model			Samples
	F1	P	R	F1	P	R	
Relevance	.66	.83	.55	.66	.80	.56	1077
Actions	.23	.89	.13	.26	.44	.19	151
Information	.61	.67	.56	.33	.57	.24	346
Movement	.00	.00	.00	.04	.04	.04	30
Preparation	.24	.83	.14	.30	.44	.23	107
Reporting	.77	.72	.82	.52	.76	.40	701
Sentiment	.54	.68	.45	.37	.64	.26	415

Using tf-idf definitely improved performances for certain classes, such as Information and Sentiment. We observe that for Relevance, the results are almost similar. For other classes, like Action and Preparation, F1 score is similar, while for precision we do better than the paper but for recall our score is slightly lower. The improvement is due to the fact thah with tf-idf we catch the importance of the rare words that better characterize each class, differently from unigrams counts that does not give a higher weight to those terms. Regarding Movement, we perform terribly due to the scarcity of data.

2.3.2 All features

	Our model			Paper model		
	F1	P	R	F1	P	R
Relevance	.72	.82	.65	.71	.81	.64
Actions	.40	.74	.27	.39	.46	.65
Information	.65	.73	.59	.48	.57	.41
Movement	.00	.00	.00	.07	.10	.07
Preparation	.20	.50	.12	.36	.41	.32
Reporting	.76	.70	.83	.73	.71	.75
Sentiment	.55	.56	.54	.53	.58	.49

Considering also those features we see that for some classes we have a slight improvement in the scores, such as for Relevance and Information. For classes like Action and Sentiment, we notice an increase in F1 and recall but a slight decrease in precision. For Preparation, it seems that the classifier works better considering only tf-idf.

2.3.3 Best features

	Our model			Paper model		
	F1	P	R	F1	P	R
Relevance	.73	.81	.67	.72	.79	.66
Actions	.31	.69	.20	.41	.42	.40
Information	.64	.71	.59	.49	.50	.49
Movement	.00	.00	.00	.08	.10	.07
Preparation	.16	.75	.09	.36	.38	.35
Reporting	.77	.72	.83	.75	.71	.80
Sentiment	.60	.65	.55	.52	.52	.52

Performing feature selection results not being beneficial for all the classes. In fact it is to notice that for Information, Actions and Preparation we had higher scores when considering all the features. For Relevance there is a minimal improvement in F1 and recall followed by a small drop in precision. On the contrary, for Reporting and Sentiment, results are better with features selection.

2.4 Observations

To sum up, the classes that are better classified and for which we managed to achieve better results the the ones shown in the paper are Relevance, Information, Sentiment and Reporting. For Actions and

Preparation we have different behaviors according to the features considered. Finally, for Movement, the scarcity of the dataset led to terrible evaluation scores. As we noticed, the greatest contribute to classification is given by tf-idf that is able to characterize each class better than the other features.

3 Burst Detection

With the purpose to reduce the number of tweets passed to the clustering algorithm, we implemented a Burst Detection module, following the idea described in the main paper (Yin et al., 2012).

The module we implemented will return the tweets referencing any uncommon events.

A bursty tweet is a tweet containing a 'bursty feature'. A 'bursty feature' is a word that appears more frequently than it normally does.

The algorithm we tried to reproduce determine whether a feature is bursty on the basis of the probability distribution in a time window.

The probability of the number of documents containing a feature f_j in the time window W_i is modeled with a binomial distribution:

$$P(n_{i,j}) = \left(\begin{array}{c} N \\ n_{i,j} \end{array} \right) p_j^{n_{i,j}} (1 - p_j)^{N - n_{i,j}} \quad (1)$$

With N the number of documents in the time window, $n_{i,j}$ is the number of documents in the i-th time window containing the feature f_j and p_j is the expected probability of tweets that contain the feature f_j.

In order to determine whether the feature f_j is bursty it's compared the actual probability $P_o(n_{i,j})$ with the expected probability p_j of feature f_j occurring in a random window.

The expectation is computed as p_j is computed as follows:

$$p_j = \frac{1}{L} \sum_{i=0}^{L} P_o(n_{i,j}) \quad (2)$$

$$P_o(n_{i,j}) = \frac{n_{i,j}}{N} \quad (3)$$

Where L is the number of time windows containing f_j.

First of all we performed a pre-processing on the tweets, stemming the words of the tweet and removing stop-words. The result of this pre-processing reduced each tweet to a collection of features.

We then compute and compare the probability of the feature to appear in the actual time window with the expected probability of the feature to appear in a random time window; if $P_o(n_{i,j}) > p_j$ the feature is marked as bursty.

This operation is done for every feature appearing in the time window.

We used a inverted index in order to return the list the tweets containing a bursty feature. In each time window we considered as corpus the tweets of the time window, then we reduce them to basic features, then we build a dictionary using the feature as key, and we associate to each feature the list of documents in which it appears. Then when we finish to check each feature, we return the union of the lists of the features considered as bursty.

We performed the train of the module on the tweets of the unlabeled dataset. The main problem is the evaluation since we don't have labels on the tweets, so we don't have a ground truth.

So what we did is to build artificially one test set, taking one day of tweets from the retrieved dataset on which we didn't train the module. From this tweets we selected all the tweets containing some features referring to the main events happened in that period, in the specific, the Hurricane Sandy and the U.S. presidential elections.

Then we take also some random tweets to add noise in the data.

At the end for the computation of the metrics we considered as bursty only the features matching the chosen features, all the other are considered not bursty.

For the evaluation we used as metrics the "detection rate" (Recall) and the "false alarm rate" (Fall-out). Since the algorithm returned almost all the tweets inside a window, with a false alarm of around 70% we improved the algorithm checking one of the referenced papers (Fung et al., 2005).

In this paper we found a way to determine whether a feature is bursty in a better way.

Starting from the binomial distribution shown in Figure 1.

We can recognize three areas, R_A, R_B and R_C, on depending in which area $n_{i,j}$ is we can determine the probability that feature f_j is bursty.

So we can recognize three cases:

1. $n_{i,j}$ is in R_A, this implies that $P_o(n_{i,j}) \leq p_j$,

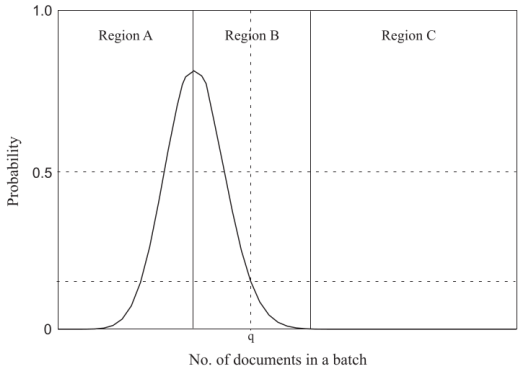

Figure 1: Binomial distribution

so f_i is not a bursty feature

2. $n_{i,j}$ is in R_C, this implies that $P_o(n_{i,j}) >> p_j$, so f_i is a bursty feature

3. $n_{i,j}$ is in R_B, in this case the feature could be either bursty or not depending to which boundary of the region R_B is closer, in this case we can compute the actual probability that the feature is bursty $P_b(i, f_j)$ in the time window W_i

The third case is the most complex, and we reduced it into checking whether the probability of the feature of being bursty is more than 0.5.

In our case the probability can be computed using a sigmoid function:

$$P_b(i, f_j) = \frac{1}{1 + e^{-x}} \tag{4}$$

$$x = P(n_{i,j}) \cdot \theta - q \tag{5}$$

Where q is the mid-point of the region R_B, and θ is the slope of the sigmoid function.

We want that the sigmoid is centered in q in such a way the probability is 0.5 when $n_{i,j} = q$. So what we do is to consider the feature bursty only when $n_{i,j} > q$.

The issue is into computing q, since we need the values for $n_{i,j}$s that are the boundaries of the three areas. For the boundaries of R_A we already know that is when $n_{i,j}/N = p_j$. For R_B we approximate it as the right boundary of the confidence interval containing the 99.9% of all the values of the distribution, so we consider as boundary the value of $n_{i,j}$ for which the right tail of the distribution contains only 0.05% of the values.

At the end, with this new check on the same test set, we get on average a detection rate of 47.16% and a false alarm rate of 14.53%. The results are more satisfying since the tweets are effectively filtered, making the module more suitable for the task of clustering.

4 Clustering

The Clustering is used to discover important topics from tweets. We tried to replicate an online incremental algorithm that groups similar tweets into topic clusters. Since we are performing an online clustering, standard clustering techniques, such as k-means, cannot be used, because we non have the whole dataset immediately available.

The algorithm takes as input a stream of tweets and uses it to form the clusters.

We represent each tweet as a vector of features, then we compute the similarity of this vector with the centroids of all the clusters, and we assign the tweet to the cluster with the closest centroid (similarly to Rocchio algorithm), but we assign the tweet to a cluster only if the similarity is above a certain threshold δ, which is application-dependent. If no suitable cluster is found, the tweet will form a new cluster by its own.

As similarity function we used a function based on the cosine similarity, which takes in account also the time distance between two tweets.

To take into account the time distance we multiply to the cosine similarity by an exponential. At first we computed the exponential using the time centroid and the time of the tweet divided the variance, but this reduced too much the value of the similarity, so we compute the squared difference with the time of the tweet and the time centroid of the cluster, and we use as variance the variance among all the times of the tweets of the cluster and the time of the current tweet. The function is the following:

$$sim(T_i, C_j) = cos(T_i, C_j) e^{-\frac{(t_{T_i} - t_{C_j})^2}{2\sigma^2}} \tag{6}$$

Where T_i is the i^{th} tweet and C_j is the j^{th} cluster.

In order to reduce the tweets to be considered, the burst detection is used as a filter, in addition the algorithm keeps two lists, one of active clusters, which are the one considered in adding new tweets, and one of inactive clusters. The algorithm

will start to consider a cluster inactive when it is no longer updated for a specific amount of time.

As features we employed both tf-idf and Word Embeddings.

4.1 Clustering evaluation and conclusions

Since, one more time, we don't have a ground truth and we don't know apriori the number of clusters, we evaluate the performances using the silhouette score.

The silhouette score gives a value in the range [-1,1], scoring how the clusters are well separated; the closer to 1 the better the separation.

Firstly, we run the clustering on few time windows using Word Embeddings and we got a great score of 0.71, against a score of 0.42 obtained in the paper.

Then we run the module on the labeled dataset on relevant tweets, checking whether the clustering algorithm identifies the same classes identified by human experts. Using tf-idf with a δ of 0.05 we obtained a silhouette score of -0.12, which means that the clustering was not able to divide the data in a proper way. So we switched to Word Embeddings with a δ of 0.01. We thus got a silhouette score of 0.76, having minimally overlapping clusters. The problem is that we have only 2 clusters. So, in order to have a more "realistic" clustering we have to drop the idea of non overlapping clusters. Using $\delta = 0.05$ we obtain a low silhouette score (-0.12). We said that this score measures how much clusters overlap. We recall that out tweets are multilabeled, so a tweet can belong to more than a class, and thus overlapping among clusters is normal. If fact, a cluster containing tweets which classes are $Information$ and $Sentiment$ will overlap with a cluster that has $Information$ and $Reporting$ tweets. The number of clusters we obtained this way is 9.

References

Gabriel Pui Cheong Fung, Jeffrey Xu Yu, Philip S Yu, and Hongjun Lu. 2005. Parameter free bursty events detection in text streams. In *Proceedings of the 31st international conference on Very large data bases*, pages 181–192. VLDB Endowment.

Kevin Stowe, Michael J Paul, Martha Palmer, Leysia Palen, and Kenneth Anderson. 2016. Identifying and categorizing disaster-related tweets. In *Proceedings of The Fourth International Workshop on Natural Language Processing for Social Media*, pages 1–6.

Jie Yin, Andrew Lampert, Mark Cameron, Bella Robinson, and Robert Power. 2012. Using social media to enhance emergency situation awareness. *IEEE Intelligent Systems*, 27(6):52–59.