Data mining and machine learning project

DEVELOPMENT OF AN APPLICATION TO ANALIZE REAL-TIME TWEETS ABOUT EARTHQUAKES

Giacomo Mantovani
Stefano Poleggi

Artificial Intelligence and Machine Learning

2019/2020

Motivation

Social network have entered our lives massively. The idea behind our work is to analyze tweets and to use them as a sort of sensor to detect study and detect their usage in case of natural disaster like earthquakes.

We developed an application for real-time monitoring of serious earthquake event detection. To do that, we analyzed tweets related to that topic using text analysis techniques; at the end we created a model able to classify tweets.

Dataset (I)

In order to build our classifier, raw tweets have been scraped using a twitter scraping tool called Twint written in Python. Given location, tags to search for and a timestamp (or a date interval), returns the informations about the tweets matching those parameters.
To build the training set, two different queries have been used:

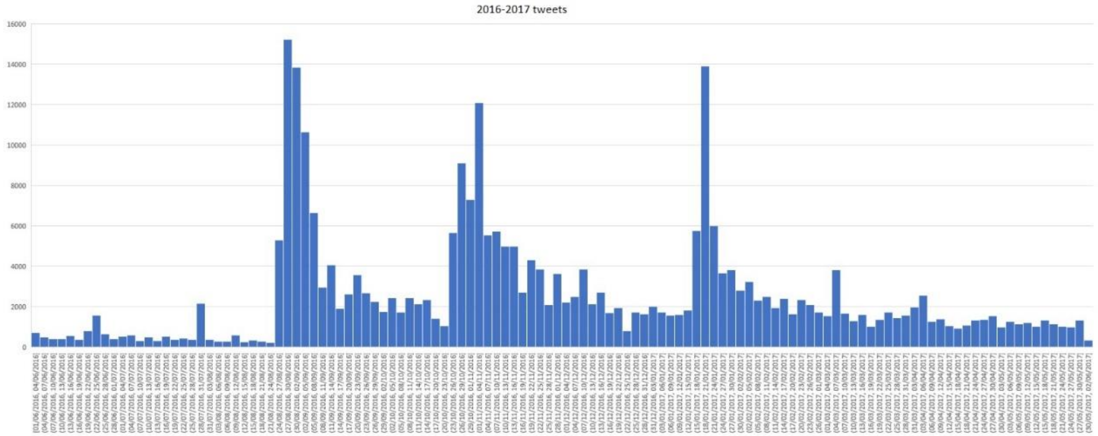
- The first query selects tweets in Italy matching the keyword “terremoto”.
- The second query selects the tweets in Italy without tags.

Another database was created by collecting tweets between the dates 2017-06-30 and 2016-06-01.

Dataset (II)

Once the database about tweets of 2016-2017 were created, we realized an histogram to check if there are spikes of tweets corresponding to catastrophic earthquakes in order to perform an analysis.

As we can see, there 3 major spikes relative to the 24-08-2016, 30-10-2016 and 18-01-2017 huge earthquakes.



Training set

All labels (earthquakes, non-earthquakes) on tweets have been added manually. We ended up having a balanced data set of 608 tweets (304 per class).

Earthquake class: tweets relative to a serious earthquake.

Non-earthquake class: tweets NOT relative to serious earthquakes (minor earthquakes are labeled as non-earthquake too)

Pre-processing

After the tweets have been fetched, all of them have been pre-processed in order to extract only the raw text and remove all meta-information associated. All useless information have been discarded, like user id, retweet flag etc. After that a regular expression filter to the text of the tweet is applied and all the characters have been converted to lower case.

Text mining process

Before creating the model we needed to apply a text mining process in order to obtain, from a set of strings, a set of numeric vectors that will be elaborated in the classification step.

Text mining steps:

- Tokenization
- Stop-word filtering
- Stemming
- Stem-filtering
- Feature representation

Classification

Once the raw tweets have been elaborated during text mining steps, we used a classification model to obtain a label for each tweet.

Different classifiers have been compared, in particular we focused on SMO (Sequential Minimal Optimization), J48 (weka’s implementation of the C4.5 algorithm), kNN (k-Nearest Neighbors, we focused on two kNN models, with k equal to 1 and 3) an the NB (the naïve Bayesian classifier, based on the Bayes’ s theorem).

Classification model (I)

- SMO

Correctly Classified Instances	551	90.625	%
Incorrectly Classified Instances	57	9.375	%

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,832	0,020	0,977	0,832	0,899	0,822	0,906	0,897	earthquake
	0,980	0,168	0,854	0,980	0,913	0,822	0,906	0,847	non-earthquake
Weighted Avg.	0,906	0,094	0,915	0,906	0,906	0,822	0,906	0,872	

- J48

Correctly Classified Instances	536	88.1579	%
Incorrectly Classified Instances	72	11.8421	%

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,868	0,105	0,892	0,868	0,880	0,763	0,952	0,934	earthquake
	0,895	0,132	0,872	0,895	0,883	0,763	0,952	0,959	non-earthquake
Weighted Avg.	0,882	0,118	0,882	0,882	0,882	0,763	0,952	0,946	

Classification model (II)

- 1NN

Correctly Classified Instances	547	89.9671 %
Incorrectly Classified Instances	61	10.0329 %

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,875	0,076	0,920	0,875	0,897	0,800	0,960	0,953	earthquake
	0,924	0,125	0,881	0,924	0,902	0,800	0,960	0,958	non-earthquake
Weighted Avg.	0,900	0,100	0,901	0,900	0,900	0,800	0,960	0,956	

- 2NN

Correctly Classified Instances	536	88.1579 %
Incorrectly Classified Instances	72	11.8421 %

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,822	0,059	0,933	0,822	0,874	0,769	0,971	0,963	earthquake
	0,941	0,178	0,841	0,941	0,888	0,769	0,971	0,975	non-earthquake
Weighted Avg.	0,882	0,118	0,887	0,882	0,881	0,769	0,971	0,969	

Classification model (III)

- 3NN

Correctly Classified Instances	521	85.6908 %
Incorrectly Classified Instances	87	14.3092 %

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,780	0,066	0,922	0,780	0,845	0,723	0,967	0,957	earthquake
	0,934	0,220	0,809	0,934	0,867	0,723	0,966	0,974	non-earthquake
Weighted Avg.	0,857	0,143	0,866	0,857	0,856	0,723	0,966	0,965	

- NB

Correctly Classified Instances	511	84.0461 %
Incorrectly Classified Instances	97	15.9539 %

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,734	0,053	0,933	0,734	0,821	0,697	0,949	0,929	earthquake
	0,947	0,266	0,780	0,947	0,856	0,697	0,949	0,942	non-earthquake
Weighted Avg.	0,840	0,160	0,857	0,840	0,839	0,697	0,949	0,935	

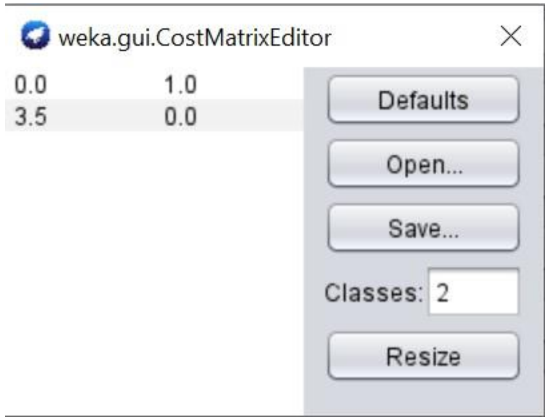
Classification model (IV)

To evaluate each classification model, we used an n-fold cross-validation (with n = 10).
The results obtained about accuracy for each classifier are summarized here:

Classifier	Accuracy (%)
SMO	90.62%
J48	88.16%
1NN	89.97%
2NN	88.16%
3NN	85.70%
NB	84.05%

Note about classifier

We performed a cost sensitive classification schema, because we wanted to give an higher misclassification error cost for the “earthquake” class rather than “non-earthquake” class.



Classifier choice

We ended up choosing the SMO classifier since it has the higher accuracy and his false negative rate is fairly low, which is good in our case since we don’t want to classify non-earthquake tweets as earthquake, but it is acceptable to have a slightly higher number of false positive (this is the reason for applying the cost sensitive classifier to our model).

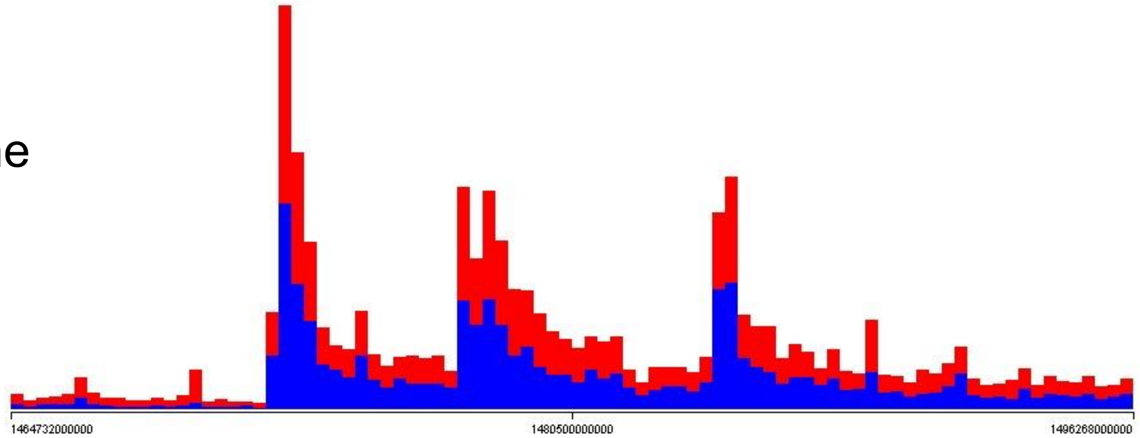
```
=== Confusion Matrix ===

   a   b   <-- classified as
 253  51 |   a = earthquake
   6 298 |   b = non-earthquake
```

Using the classifier on the database

To exploit a further test, we applied the model on the database containing the 2016-2017 tweets.

We can see that the classifier has labeled a good amount of tweets as “earthquake” at the peaks seen previously.



Toward the application

Once we found the right classifier, we started implementing the application using java language. The application fetches the last 10 minutes tweets containing the keyword “terremoto” and performs a real-time classification of each fetched tweet.

Two thresholds are set: warningThreshold and emergencyThreshold.
If N tweets (of the last 10 minutes) are classified as “earthquake”:

- If N > warningThreshold: then a warning message is shown.
- If N > emergencyThreshold: then an emergency message is shown (earthquake recognized).

Threshold values

To choose the threshold values we counted how many tweets are labeled as positive (in a 10 minute span) in three different cases and compared those values:

- tweets relative to a catastrophic earthquake.
- tweets relative to a non-catastrophic earthquake.
- tweets relative to a date without earthquake.

Considering those thresholds, if an earthquake has been recognized, its informations are stored into a sql database and showed on the application.