University of Pisa

SCUOLA DI INGEGNERIA

Corso di Laurea in Artificial Intelligence and Data Engineering

Documentation of the "STAD" Application

Candidati

Alice Nannini

Marco Parola

Relatori

Prof. Francesco Marcelloni

Prof. Pietro Ducange

Anno Accademico 2019–2020

Contents

1 Introduction

The goal of this application is to prevent and detect situations potentially dangerous, caused by huge amounts of rain, scraping twitter and analyzing each tweet, in order to discover some tweets containing information related to these critical situations.
We analyze the data and develop the application using Python and Sklearn library.

2 The Data

2.1 Retrieve the data

The data, on which this application works, are tweets. In order to collect enought tweets, we scraped twitter, using **twint**.
Twint is an advanced opensource Twitter scraping tool, written in Python, thanks to which it's very easy to collect data, according to some criteria, and store them in csv files. For more information about *twintproject* visit the Github repository: `https://github.com/twintproject`.

```
twint -s <WORDS> -o tweetPioggia.csv --csv

<WORDS> : 'pioggia', 'piove', 'allerta', 'meteo', 'alluvione', 'maltempo'
```

Examples:

- "Ora piove a dirotto per la gioia di yuki che non può andare al parco"
- "Ma dai, ma piove sul bagnato! Povera Antonella!!!!! #GFVIP"

Moreover we added to the dataset some posts randomly downloaded, not related to any weather phenomenons (without specifying any keywords).

2.2 Prepare the dataset

After collectioning the tweets (864), we assigned each of them to a class, in order to prepare the dataset, thanks to which we can build some classifiers.
We decided to map the tweets in 3 classes:

- 0 -> the tweet is not related to a weather condition (348)
- 1 -> the tweet is about rain or some weather condition not dangerous (214)
- 2 -> the tweet is about some dangerous situation caused by the rain (302)

3 Preprocessing

In this phase, we first extract from the *csv* file, created by *twint*, just the tweets' texts, discarding all the meta-information associated with them (id, user, etc.).
Then we delete some tweets, in order to manage only the italian tweets. The deleting is made using the *guess_language* package.
Finally, we clean the text of each tweet removing eventual URLs: to do so, we use a Regular Expression filter.

4 Supervised learning stage

After the preprocessing phase, we follow the standard steps to build our classification model, managing the tweets' texts:

- tokenization, stop-word filtering and stemming
- stem filtering by relevant stems, selected by the tf-idf computation
- classification and evaluation

4.1 Tokenization, stop-word filtering and stemming

After stratified splitting the dataset in tranining and test set, we tokenize each tweet and we apply an italian stemming filter, in order to find more general words.

```
italian_stemmer = SnowballStemmer('italian')
class StemmedCountVectorizer(CountVectorizer):
   def build_analyzer(self):
      analyzer = super(StemmedCountVectorizer, self).build_analyzer()
      return lambda doc: ([italian_stemmer.stem(w) for w in analyzer(doc)])
```

4.2 Stem filtering by relevant stems

All the stems are united in one vector and are weighted using the IDF index (Inverse Document Frequency).
Then, each training tweet is represented as a vector of features, dimension equal to the number of stems, and the i-th feature is calculated as the frequency of the i-th stem in the tweet per the weight of that stem.
Finally each stem is evaluated by the Information Gain (IG) value between the corresponding feature and the possible class labels:

```
IG(C, Sq) = H(C) - H(C|Sq)
```

where Sq is the feature corresponding to the stem s_q, $H(C)$ is the entropy of C, and $H(C|Sq)$ is the entropy of C after the observation of the feature Sq.
Then, the stems are ranked in descending order and F stems, with $F \leq Q$, are selected among these.

```
tfidf_transformer = TfidfTransformer(smooth_idf=True,use_idf=True)
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
```

4.3 Classification and evaluation

After the tfidf_transformer fitting phase, we use it to transform the test set and we build and test different classifiers and compute different metrics to compare them (accurancy, f-score, confusion-matrix):

```
Multinomial NB:
accuracy : 0.8076923076923077
f_score : 0.7707147012774636

Decision Tree:
accuracy : 0.8423076923076923
f_score : 0.8328284726675067

SVM:
accuracy : 0.8653846153846154
f_score : 0.8530622512329255

k-NN (k = 5) :
accuracy : 0.4846153846153846
f_score : 0.3531361519385472

Adaboost:
accuracy : 0.7
f_score : 0.6895601571827289

Random Forest:
accuracy : 0.8692307692307693
f_score : 0.8609699374903657
```

4.4 Cross-validation

We try to evaluate all the previous phases of elaboration and supervised learning, based on our dataset, with the method of cross-validation. Using 10 folds, these are our results for each classifier involved:

```
Accuracy MultinomialNB : 0.75 (+/- 0.15)
[0.56 0.84 0.82 0.74 0.71 0.73 0.83 0.80 0.77 0.72]

Accuracy Decision Tree : 0.79 (+/- 0.25)
[0.71 0.93 0.86 0.75 0.72 0.79 0.94 0.94 0.77 0.51]

Accuracy SVM : 0.82 (+/- 0.16)
[0.74 0.90 0.84 0.74 0.78 0.86 0.92 0.92 0.79 0.69]

Accuracy k-NN : 0.69 (+/- 0.16)
[0.68 0.77 0.79 0.63 0.67 0.63 0.76 0.74 0.66 0.52]

Accuracy Adaboost : 0.68 (+/- 0.15)
[0.67 0.68 0.84 0.59 0.67 0.66 0.67 0.72 0.76 0.55]

Accuracy Random Forest : 0.86 (+/- 0.21)
[0.76 0.91 0.86 0.84 0.90 0.95 0.99 0.94 0.78 0.63]
```

Given the results obtained both in single evaluation and in cross-validation, we conclude that the SVM classifier is the best for our application. Indeed, it has the higher accuracy, second just to the Random Forest's one. Moreover, this last algorithm has larger confidence intervals ($\pm 2std$) than the SVM, that is to prefer.

5 Experimental results

We select some significant events of strong weather conditions, involving rain and flooding, happened in Italy during the past years. We collect the tweets posted on the social network during the hours of each event and then we pass these data to our new constructed model. The selected events are:

- flood in Casteldaccia, province of Palermo, 2018-11-03

```
twint -g="38.1405227,13.2870764,50km" --since 2018-11-02 --until 2018-11-05
-o palermo2018.csv --csv
```

```
twint -g="44.561806,8.595268,30km" --since 2011-11-01 --until 2011-11-06
-o genova2011.csv --csv

twint -g="44.701608,10.5680563,25km" --since 2017-12-08 --until 2017-12-13
-o reggioemilia2017.csv --csv

twint --near Pisa --since 2019-08-05 --until 2019-08-08 -o pisa2019.csv --csv

twint --near Firenze --since 2019-07-01 --until 2019-07-06  -o firenze2019.csv --csv
```

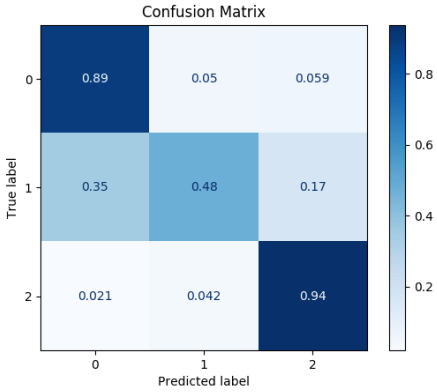
- flood in Piemonte, 2016-11-21/25

```
twint -g="45.0704900,7.6868200,50km" --since 2016-11-23 --until 2016-11-26
-o piemonte2016.csv --csv
```

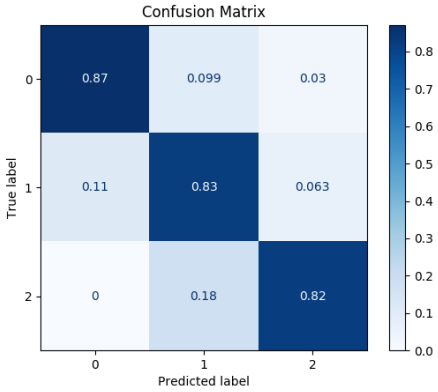
$weightedP = \frac{(N_1+2N_2)}{N_{tot}}$

Then, we select also a day with no significant weather conditions, and we apply the model on the data retrieved also from that day:

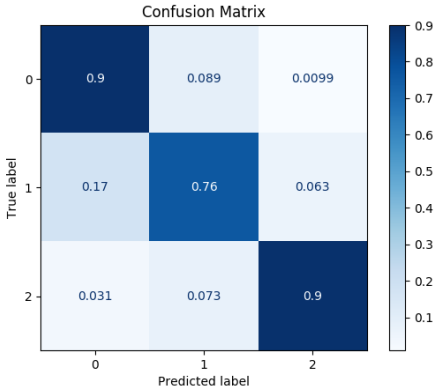
```
twint --near Pisa --since 2019-08-10 --until 2019-08-11 -o pisa2019.csv --csv
```



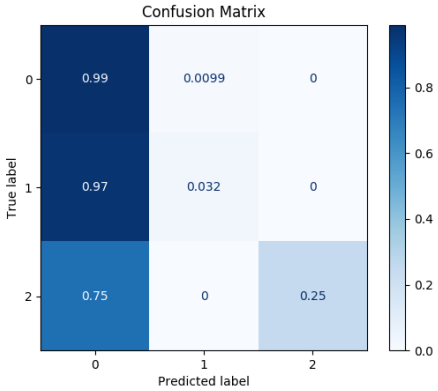
(a) *Multinomial Naive-Bayes*



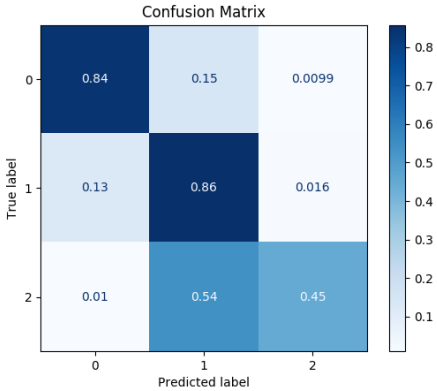
(b) *Decision Tree*



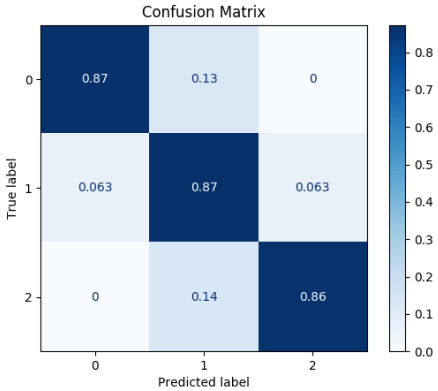
(c) *SVM*



(d) *k Nearest Neighbors (k = 5)*



(e) *Adaboost*



(f) *Random Forest*

Figure 1