



Documentation of the "STAD" Application

Candidati

Alice Nannini

Marco Parola

Relatori

Prof. Francesco Marcelloni

Prof. Pietro Ducange

Contents

1	Introduction	1
2	The Data	1
2.1	Retrieve the data	1
2.2	Prepare the dataset	1
3	Preprocessing	1
4	Text Elaboration	2
4.1	Tokenization, stop-word filtering and stemming	2
4.2	Stem filtering by relevant stems	2
4.3	Classification and evaluation	2

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 enough 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 (902), 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 (376)
- 1 -> the tweet is about rain or some weather condition not dangerous (224)
- 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 training 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:

$$IG(C, S_q) = H(C) - H(C|S_q)$$

where S_q is the feature corresponding to the stem s_q , $H(C)$ is the entropy of C , and $H(C|S_q)$ is the entropy of C after the observation of the feature S_q .

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 (accuracy, f-score, confusion-matrix):

Random Forest:

```
accuracy : 0.8376383763837638
f_score : 0.8169722687103137
```

NB:

```
accuracy : 0.7822878228782287
f_score : 0.7217817028974173
```

SVM:

accuracy : 0.7970479704797048

f_score : 0.7676179382644147

Decision Tree:

accuracy : 0.8191881918819188

f_score : 0.8005383709471964

k-NN:

accuracy : 0.5276752767527675

f_score : 0.38181323539413725

Adaboost:

accuracy : 0.7712177121771218

f_score : 0.7538220246286299

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 (+/- 0.10)

Accuracy Decision Tree : 0.77 (+/- 0.22)

Accuracy SVM : 0.80 (+/- 0.16)

Accuracy k-NN : 0.68 (+/- 0.17)

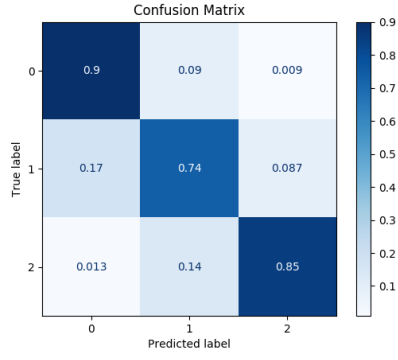
Accuracy Adaboost : 0.72 (+/- 0.22)

Accuracy Random Forest : 0.81 (+/- 0.21)

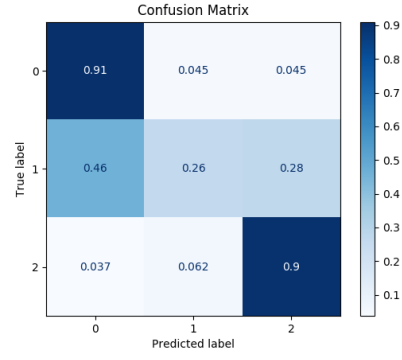
5 Experimental results

We select some significant events of strong weather condition, 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:

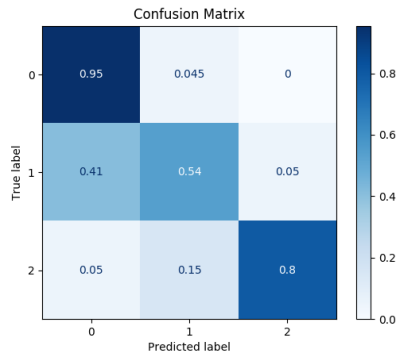
-
-



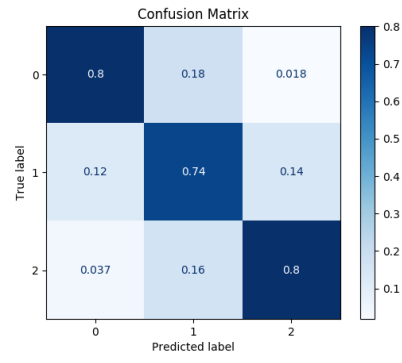
(a) *Random Forest*



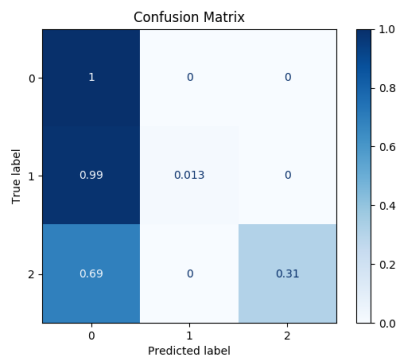
(b) *Naive-Bayes*



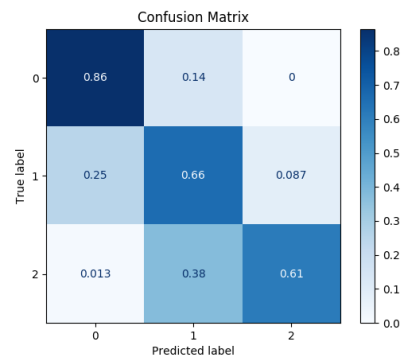
(c) *SVM*



(d) *Decision Tree*



(e) *k Nearest Neighbors ($k = 500$)*



(f) *Adaboost*

Figure 1