STAD: ”Scraping Twitter And Detect” on flooding

Data Mining and Machine Learning 2019/2020

Nannini Alice
Parola Marco

What is STAD

The goal of our application is to prevent and detect potentially dangerous situations, caused by strong weather conditions that involve rain and flooding.

It’s based on scraping twitter and analyzing each tweet, in order to discover some tweets that contain information related to these critical situations.

We analyze the data and develop the application using **Python** and **Sklearn** library.

Getting the data

The tweets are collected using **twint**:

- 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.
- https://github.com/twintproject

Some search criteria are setted, e.g. geographic coordinates, to find the data related to the area that is to be monitored.

```
twint --near Pisa -o data.csv --csv
```

Pre-processing the data

- We extract from the csv file, created by twint, the tweets' texts, discarding all the meta-information associated with them (id, user, etc.).
- We filter only the tweets, written in Italian, by using the *guess_language* package.
- We clean the text of each tweet removing eventual URLs: to do so, we use a Regular Expression filter.

Text elaboration

We follow the standard steps of text elaboration, in order to be able to represent each tweet as a numerical vector:

- tokenization, stop-word filtering and stemming
- stem filtering by relevant stems
- feature representation

Then the classification model is applied on the vectorized tweets.

The training set

We've collected tweets from September 2019 to February 2020, filtering by some context-related keywords:

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

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

We've also scraped some posts randomly, not related to any weather phenomenons (without specifying any keywords).

The training set

We've selected and manually labelled 864 tweets, from the collected ones. We've decided to map them 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)

The training set

Class 0:

"Ma si certo. L'errore e la stupidaggine di questi ragazzi sono da condannare, anche perché c'erano delle direttive ferree vista la situazione. Su questo non ci **piove**. Spero di essermi chiarito."

Class 2:

"Ora **piove** a dirotto per la gioia di yuki che non può andare al parco"

Class 2:

"Maremma, il **maltempo** devasta la riserva della Feniglia, mille pini sradicati"

Class 0:

"Ma dai, ma **piove** sul bagnato! Povera Antonella!!!!! #GFVIP"

Class 1:

"Ma oggi **piove** pure e io sono a casa a non saper che fare"

Supervised learning stage

In order to build our model, we've previously carried on a supervised learning phase, in which we've grouped all the elaboration actions into a single **pipeline**. The pipeline is executed on the training set multiple times, each time using a different classification algorithm.

```
text_clf = Pipeline([
    ('vect', stemming.StemmedCountVectorizer(analyzer='word',
        stop_words=set(stopwords.words('italian')))),
    ('tfidf', TfidfTransformer(smooth_idf=True,use_idf=True)),
    ('clf', Classifier()),
])
```

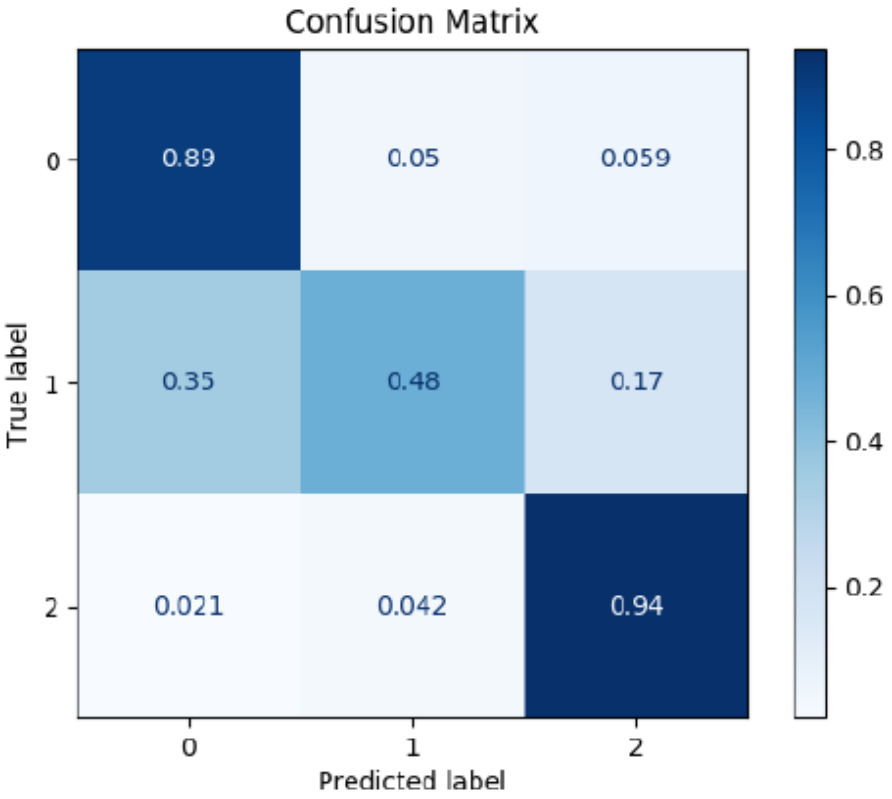
Classification and evaluation

After stratified splitting the dataset in training set (70%) and test set (30%), we've tried an **holdout** validation to compute for each classifier the accuracy, the f-measure and the confusion matrix.

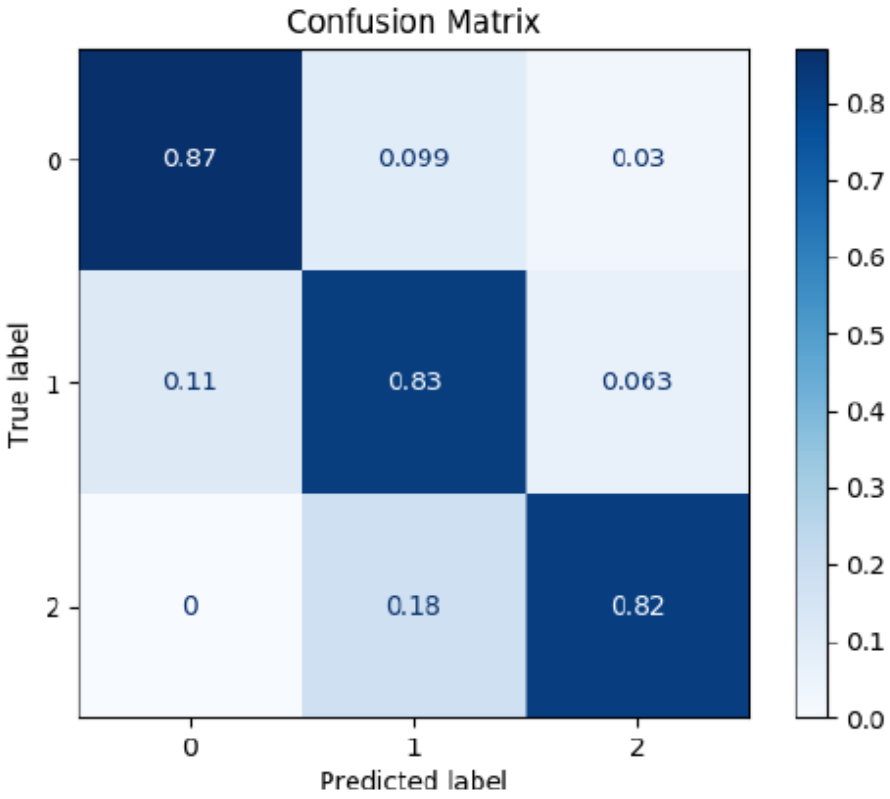
```
Multinomial NB:
accuracy : 0.8076923076923077
f_score : 0.7707147012774636

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
```

Classification and evaluation

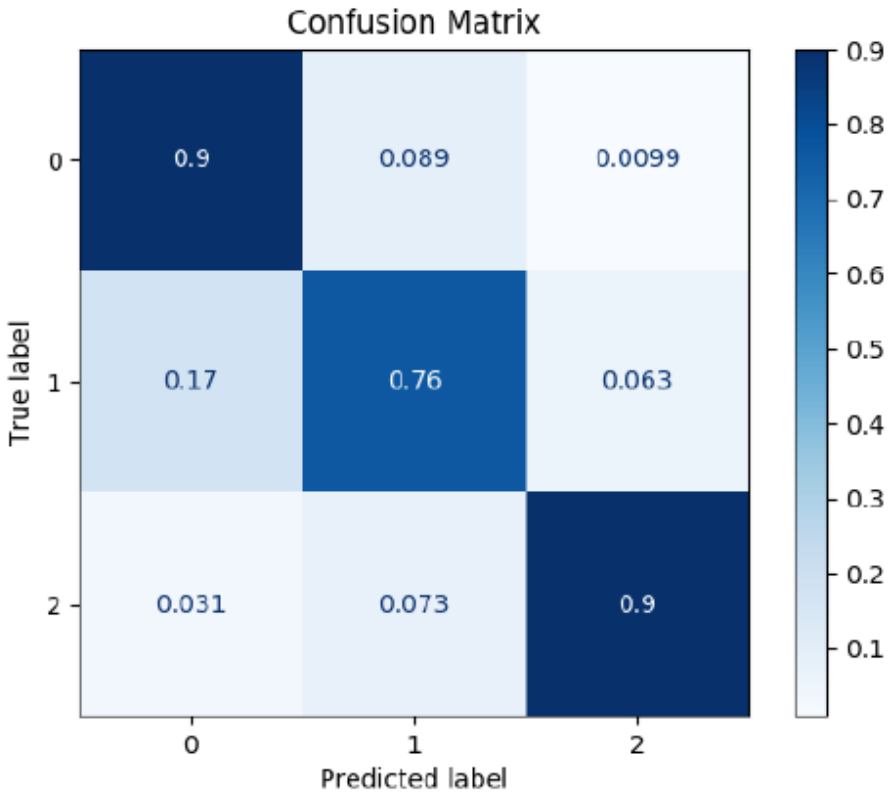


(a) *Multinomial Naive-Bayes*

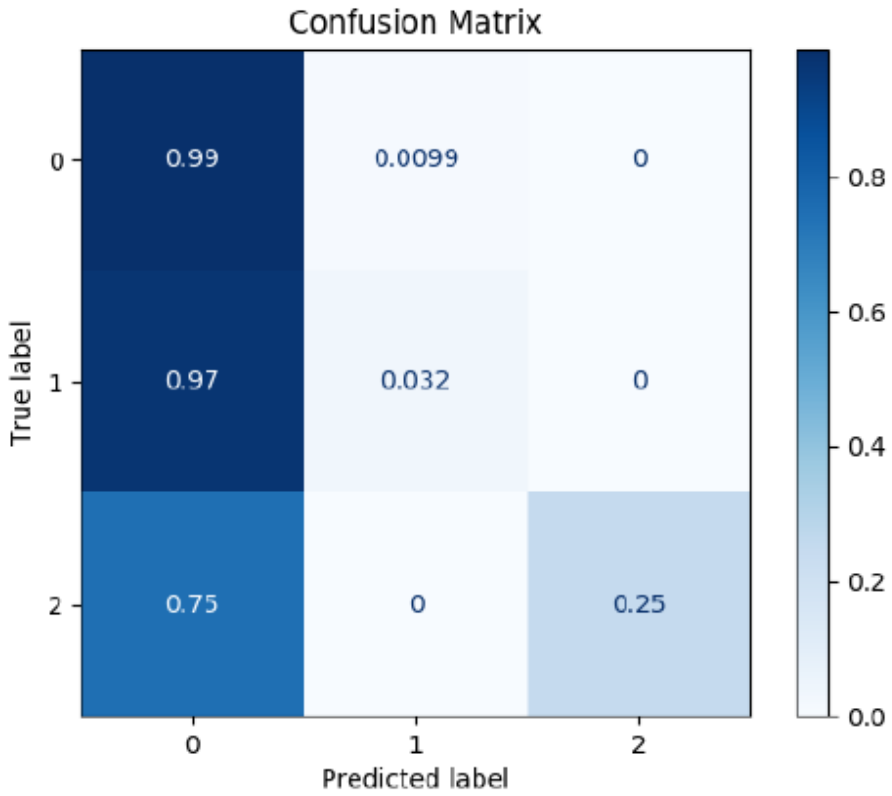


(b) *Decision Tree*

Classification and evaluation

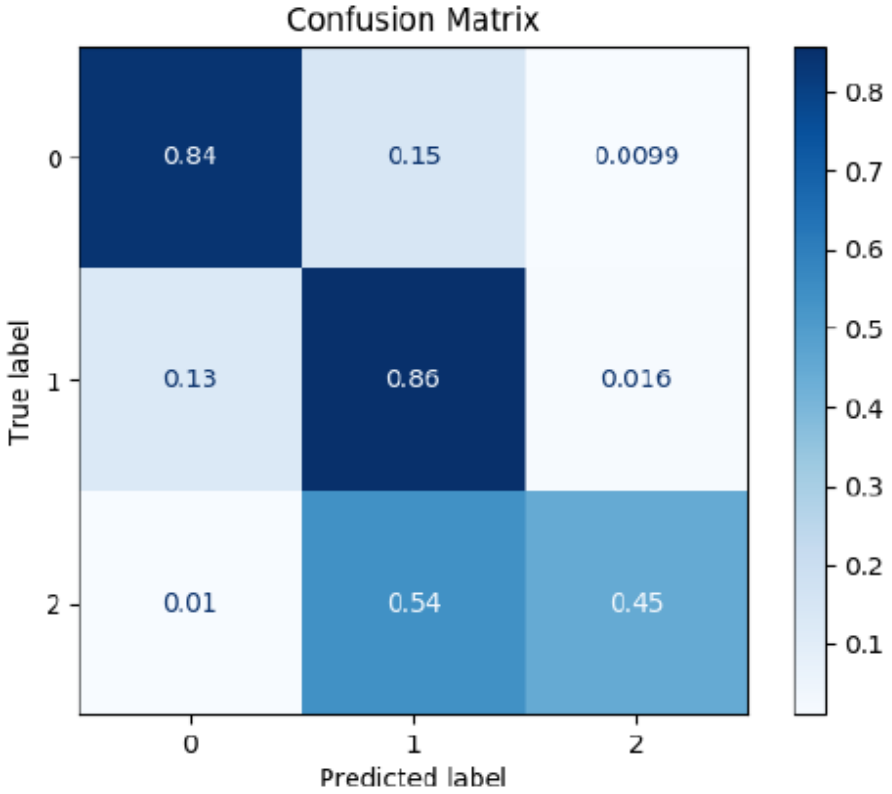


(c) *SVM*

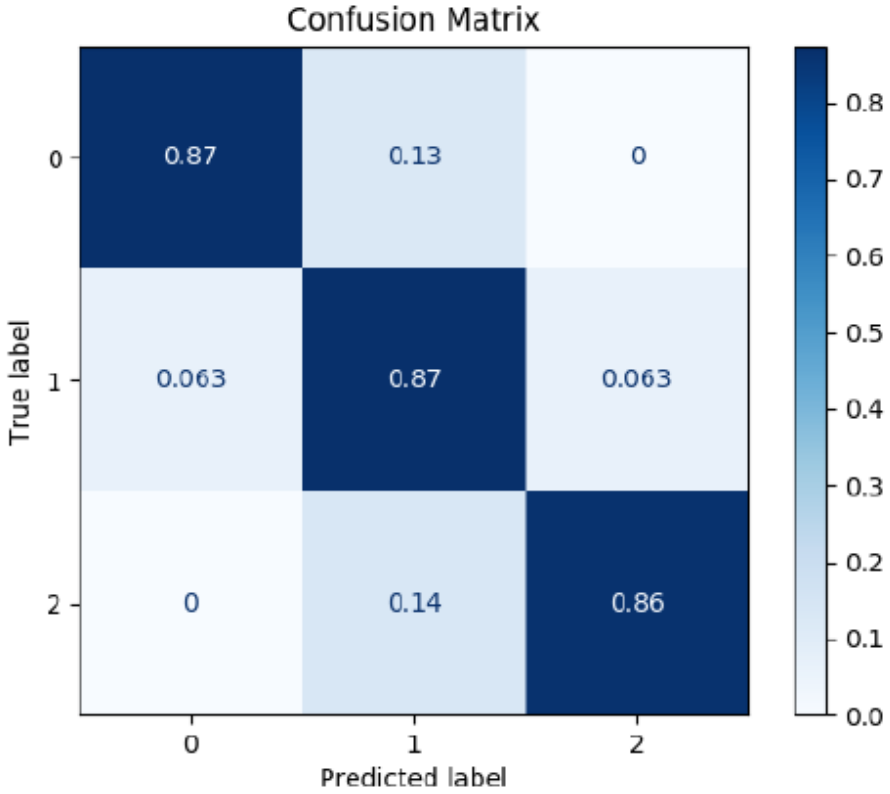


(d) *k Nearest Neighbors (k = 5)*

Classification and evaluation



(e) *Adaboost*



(f) *Random Forest*

Classification and evaluation

We've also tried to evaluate all the previous phases of elaboration and supervised learning, based on our dataset, with the method of **cross-validation** (using 10 folds).

```
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]
```

Classification and evaluation

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 (±2std) than the SVM, that is to prefer.

```
Accuracy SVM : 0.82 (+/- 0.16)
Accuracy Random Forest : 0.86 (+/- 0.21)
```

Classification and evaluation

We've applied the non-parametric **Wilcoxon** test (with α=0.1) between the accuracy distributions, taken from two cross-validation executions, of the SVM classifier and the accuracy distributions of the other classifiers.

All the tests rejected the null-hypothesis, so we've been positive in choosing the SVM as our classifier.

Testing the model

We've selected some significant events of strong weather conditions, involving rain and flooding, happened in Italy during the past years. We've collect the tweets posted on the social network during an interval of 5 days around each event, and then we've passed these data to our new constructed model.

Some examples:

- 2011-11-04 flood in Genova (6 deaths)

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

- 2017-12-12 flood in Lentigione, Reggio Emilia (1 death)

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

Testing the model

where	radius	when	tot tweets	class 0	class 1	class 2	weighted%
Genova	30km	2011-11-01/05	179	151	14	14	23,46
		01/11/11	26	26	0	0	0
		02/11/11	16	16	0	0	0
		03/11/11	27	24	2	1	14,8
		04/11/11	**44**	**37**	**6**	**1**	**18,18**
		05/11/11	66	48	6	12	45,46
Reggio Emilia	25km	2017-12-08/12	166	151	11	4	11,45
		08/12/17	38	36	2	0	5,26
		09/12/17	30	29	1	0	3,33
		10/12/17	42	36	4	2	19,04
		11/12/17	34	31	2	1	11,76
		12/12/17	**22**	**19**	**2**	**1**	**18,2**

Testing the model

We've also selected some days with no significant weather conditions, and applied the model on the data retrieved.

Some examples:

- `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`

Testing the model

where	when	tot tweets	class 0	class 1	class 2	weighted%
Pisa	2019-08-05/07	438	427	10	1	2,74
	2019-08-05	133	130	3	0	2,25
	2019-08-06	157	152	4	1	3,83
	2019-08-07	148	145	3	0	2,03
Firenze	2019-07-01/05	3292	3194	89	9	3,24
	2019-07-01	692	673	17	2	3,04
	2019-07-02	668	643	24	1	3,89
	2019-07-03	747	722	23	2	3,62
	2019-07-04	616	600	14	2	2,91
	2019-07-05	569	556	11	2	2,63

Testing the model

Given the various tests on the model, we've decided to set a threshold for launching the alert when the application detects more than **10%** on the collected tweets classified as 1 or 2.

The percentage is calculated weighting the class 2 more than the class 1:

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

The GUI

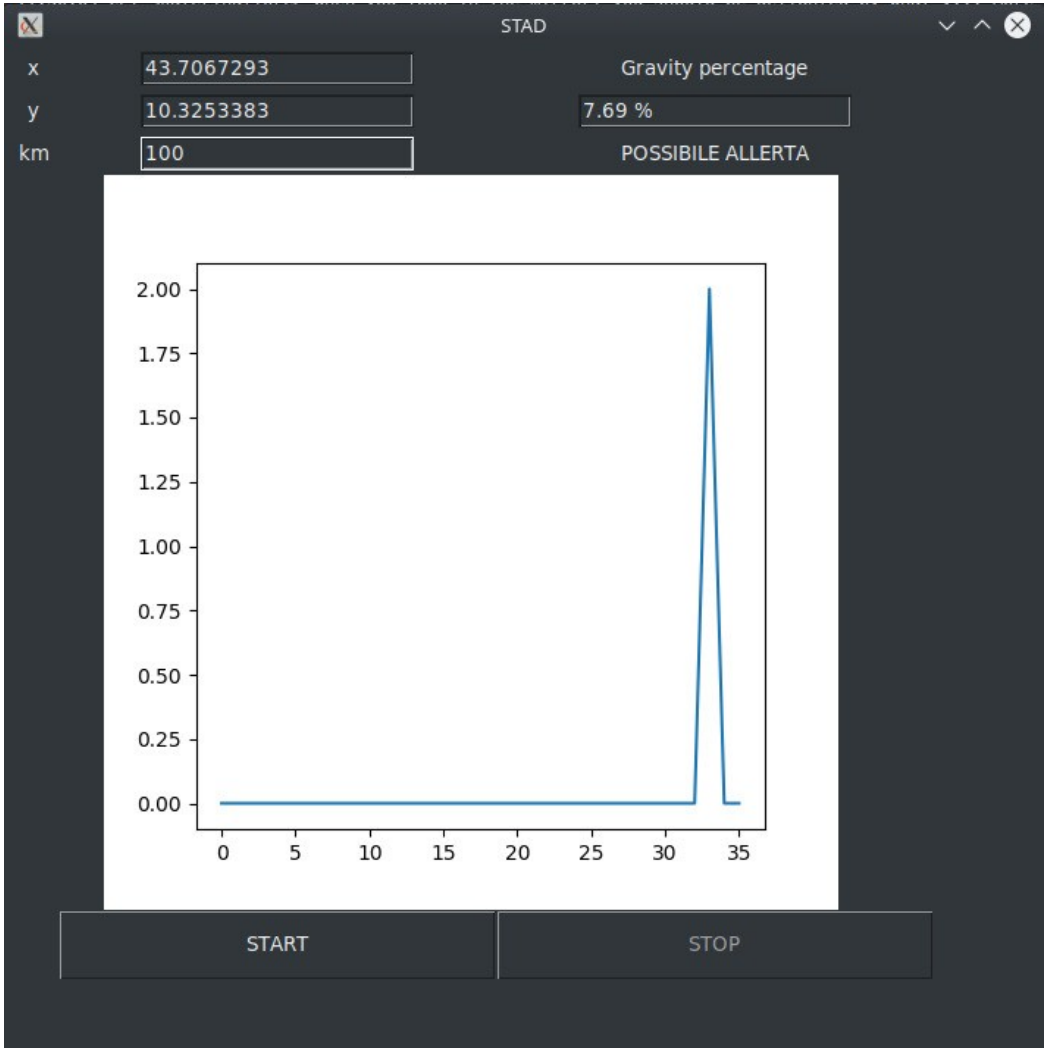
After the analysis phase, we've trained an SVM classifier and exported it in a file.

We've implemented a simple demo, requiring 3 parameters: x coordinate, y coordinate, and radius of the area.

Every 10 minutes a scraping phase is scheduled, in order to retrieve the tweets published in the last 10 minutes. These tweets are elaborated and classified by the imported SMV classifier.

We plot the data related to the last 6 hours, and we check if in this interval the percentage of the tweets about rain is greater than 10%.

The GUI


STAD
x
43.7067293
Gravity percentage
y
10.3253383
7.69 %
km
100
POSSIBILE ALLERTA
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
5
10
15
20
25
30
35
START
STOP