Spam Detector

Project for Data Mining and Machine Learning

Francesco Campilongo

Contents

Chapter 1

Introduction

Since quite a while now, spam is a popular term that specific messages not desired, and now days this kind of messages are always more frequent by electronic mail or by SMS, these two are the use case token in consideration in this paper. Purpose of this documentation is to prove the machine learning algorithms utility for the detection of spam, useless messages, into dataset of E-Mails and SMS.

1.1 Datasets

The datasets used for this project are two:

- E-mail dataset
- SMS dataset

Those two datasets where found on Kaggle.com both of them in a .csv format.

1.2 Data Preprocessing

The datasets found were already pretty good for the type of result this project wants to achieve. The E-mail dataset is characterize from three column a "label", "text" and "label_num", the second column contains the actual message, the first and third column are the same, but in the first one there are the actual words "ham", for the messages which are not classified as spam, and "spam", instead in the third column there is there are "0" for the ham messages and "1" for the spam messages;
The first column has been discharged since not useful for the execution of the classification algorithm.
The SMS file had need a little more work to obtain a dataset usable, because the it has five columns named "v1", "v2", "", "" and "", the first contains the words "ham", for the messages which are not classified as spam and "spam"; the second column contains the actual messages and the last three columns are basically blank, so in order to obtain a good dataset to work with the last three columns has been discharged the first two are taken in consideration, but in the first one the "ham" word is changed with a "0" and the "spam" word with a "1", in order to have the same kind of datasets between the E-mail and the SMSs.

1.3 Features Vector

In order to gather features from a collection of raw documents, the e-mail and SMS text, a function available on the Scikit-learn has been used, TfidfVectorizer, a function that let the user convert a collection of raw documents to a matrix of TF-IDF features. TfidfVectorizer has some parameters to optimize the conversion, here three parameters were utilized:

- min_df = 1 -> When building the vocabulary ignore terms that have a document frequency strictly lower than the given threshold.
- stop_words = 'english' -> In order to ignore the english stop_words.
- lowercase = True -> Convert all characters to lowercase before tokenizing.

After this function call from string e-mail and SMS, TfidfVectorizer gave back, as mentioned earlier, a matrix of TF-IDF which is a numeric (real) matrix.
To better clarify from this kind of data set:

```
3075     Mum, hope you are having a great day. Hoping t...
1787                               Yes:)sura in sun tv.:)lol.
1614     Me sef dey laugh you. Meanwhile how's my darli...
4304                        Yo come over carlos will be here soon
3266                             Ok then i come n pick u at engin?
...
```

To this features vector:

```
(0, 741)        0.3219352588930141
(0, 3979)       0.2410582143632299
(0, 4296)       0.3891385935794867
(0, 6599)       0.20296878731699391
(0, 3386)       0.3219352588930141
(0, 2122)       0.38613577623520473
(0, 3136)       0.440116181574609
...
```

In order to implement all the classifiers took in consideration in the best way possible to get the best results.

1.3.1 TF-IDF

Short for Term Frequency - Inverse Document Frequency, is a numeric statistic that is intended to reflect how important is a word in a document. The idea is to give importance to the terms who appear into the document but usually are not that frequent.

Chapter 2

Design

The application design is pretty basic, it is written in python (version 3) does not involve any classes, just the creation of some functions in order to keep the code clean and more understandable.

2.1 Implementation

2.1.1 Libraries

The libraries involved into this application are two:

- Pandas
- Scikit-learn

Pandas gives the possibility to use a dataset in .csv (and many others) from the disk, implementing it on a dataframe into the main memory to use all the machine learning algorithm that are imported from Sklearn.

2.1.2 Functions

The functions are meant to implement all the classifiers used, which are the following:

- Support Vector Machine
- Näive Bayes
- K Nearest Neighbor
- Random Forest
- AdaBoost

These functions are written to use them with different datasets, the SMS dataset and the E-Mail dataset as already mention earlier.
There is also another function that is called to show the result of all the algorithms.

2.1.3 Main code

This section is used to show some insight of the main code.
Here will be shown a bit of the dataset cleaning and processing that has been made, and described into the Data Preprocessing chapter.

```
df2 = pd.read_csv("sms_spam.csv", encoding = "ISO-8859-1")
dfsp = df2[['v1', 'v2']]
dfsp.loc[dfsp["v1"] == 'ham', "Category"] = 0
dfsp.loc[dfsp["v1"] == 'spam', "Category"] = 1
dfsp = dfsp.rename(columns={'v2': 'Content'})
dfs = dfsp[['Content', 'Category']]
```

The following code will give a look at one of the function, the one that implement the Support Vector Machine. Since these functions are quite similar there is no need to show them all here.

```
def SVM(x_train, y_trainFeat, y_testFeat):
  classifier = LinearSVC()
  classifier.fit(y_trainFeat, x_train)
  predRes = classifier.predict(y_testFeat)
  return predRes
```

The following code lines will show the functions calls, also in this case in order to avoid repetitions only few functions call will be shown.

```
# KNN
predResMailKNN = KNN(x_train, y_trainFeat, y_testFeat)
# Metrics and results
print("\tK Nearest Neighbors RESULTS")
print("Neighbors Number: 1")
show_res(actual_x, predResMailKNN)

# RF
predResMailRF = RF(x_train, y_trainFeat, y_testFeat)
# Metrics and results
print("\tRandom Forest RESULTS")
show_res(actual_x, predResMailRF)
```

Chapter 3

Results

Now some considerations based on results per every classifier utilized.
All the results are visible in the next page.

Support Vector Machine

Support Vector Machine is a classifier which use hyperplane creation in order to separate different classes. As the result suggests this was a extremely good classifier for this type of datasets. This was expected since there is already some literature that's evidence that this classifier for text classification problem is one of the best if not the best.

Näive Bayes

A statistical classifier which performs probabilistic prediction on the class membership, based on the Bayes' Theorem. The results here are pretty good, in this particular experiments the version of the Näive Bayes classifier utilized was the Multinomial Näive Bayes which is the best version for text classification problems.

K Nearest Neighbor

The nearest neighbor, defined in terms of euclidean distance, k nearest neighbor returns the most common value among the k training examples nearest to the instance that need to be classified. For this kind of problem (text classification for spam detection) the best value for k was 1. The results were very good all calculated with k = 1.

Random Forest

An algorithm characterized by an ensemble of classifier where each one is a decision tree classifier generated using a random selection of attributes at each node to determine the split, every tree gets to vote and the most popular class is returned. The metrics tells very good results for this algorithm in this kind of classification problem.

AdaBoost

A boosting algorithm which uses an ensemble of classifier, where each one gives a weighted vote, based on how well the classifier performs. The lower a classifier error rate the more accurate it is and so the higher is the weight for its vote. Also in this case the metrics for this algorithm are very good.

Here the results for what concern the E-mail dataset:
The metrics that shows the results goodness are: Accuracy, F Score and Confusion Matrix; Chosen in order to give a deep look at the algorithms behavior.

```
/--------------------SpamDetector for e-Mail--------------------/
Support Vector Machine results
Accuracy Score: 98.6473
F Score:  98.3941
Confusion Matrix:
[[716  13]
[  1 305]]

Multinomial Naive Bayes results
Accuracy Score: 91.3043
F Score:  88.5304
Confusion Matrix:
[[727   2]
[ 88 218]]

K Nearest Neighbors results
Neighbors Number: 1
Accuracy Score: 95.8454
F Score:  94.9084
Confusion Matrix:
[[718  11]
[ 32 274]]

Random Forest results
Accuracy Score: 95.4589
F Score:  94.5432
Confusion Matrix:
[[706  23]
[ 24 282]]

Adaboost results
Estimators Number: 100
Accuracy Score: 96.9082
F Score:  96.3159
Confusion Matrix:
[[709  20]
[ 12 294]]
```

Here the results for what concern the SMS dataset:

```
/---------------------SpamDetector for SMS--------------------/
Support Vector Machine results
Accuracy Score: 98.5650
F Score:  96.8657
Confusion Matrix:
[[960   0]
 [ 16 139]]

Multinomial Naive Bayes results
Accuracy Score: 97.1300
F Score:  93.4249
Confusion Matrix:
[[960   0]
 [ 32 123]]

K Nearest Neighbors results
Neighbors Number: 1
Accuracy Score: 94.7982
F Score:  87.0259
Confusion Matrix:
[[960   0]
 [ 58  97]]

Random Forest results
Accuracy Score: 97.4888
F Score:  94.3510
Confusion Matrix:
[[959   1]
 [ 27 128]]

Adaboost results
Estimators Number: 100
Accuracy Score: 97.7578
F Score:  95.0885
Confusion Matrix:
[[956   4]
 [ 21 134]]
```