SYSTEM DIAGNOSTICS

Tweets sentiment classification

29/01/2022

Rosa María Rodríguez
s191895@student.pg.edu.pl

1. Introduction

1.1 Problem Statement

Twitter is a popular social network that allows users to post short messages called *tweets.* Many customers use it every day to share opinions, complaints or finding like-minded people. However, Twitter has proved to be a great channel for enterprises to broadcast new products or finding potential customers. This is the reason why using an algorithm to analyse the sentiment of a group of tweets is a great opportunity for enterprises to keep track of what is being said about them.

Predicting the sentiments of a tweet is an example of text classification, which is a typical problem in Natural Language Processing. Basically, each tweet is a document that will have to be assigned to a class of documents (positive tweets and negative tweets).

2. The model

2.1 Input data

The dataset used for this project consists of 30.000 tweets and their sentiment (negative or positive, encoded as 0 and 1 respectively) that can be found at https://github.com/laxmimerit/twitter-data/blob/master/twitter30k_cleaned.csv. All the tweets are in lowercase and emojis and the special characters are removed.

```
yaniv75 thnks for the follow,1
nothing like getting sick at your best friends shower,0
_huny i concur w your feelings about ppl,0
a new book easyway and penny gave me a kiss x,1
thisisryanross sketch from 2007 of you,1
indianajim hey just listened to the last few epis of aoij great stuff loved the defense of tee that was pure goodness,1
charliepeacock sounds exciting have fun,1
```

Figure 1 – Extract of the dataset

The data frame we will be using has one row per tweet and only two columns: the text and the sentiment.

```
In [208]: df = pd.read_csv("C:\\Users\\rosam\\Downloads\\processed_dataset.txt")
          rows, columns = df.shape
          print(df.shape)
          df.head()
```

```
(30000, 2)
```

Out[208]:

	tweets	sentiment
0	robbiebronniman sounds like a great night	1
1	damn the person who stolde my wallet may karma...	1
2	greetings from the piano bench photo	1
3	drewryanscott i love it i love you haha forget...	1
4	kissthestars pretty pretty pretty please pakid...	0

Figure 2 – Dataframe

The distribution of the input data is exactly 50% positive tweets and 50% negative tweets. This means that the input data will not be biased towards one class.

sentiment	tweets
0	15000
1	15000

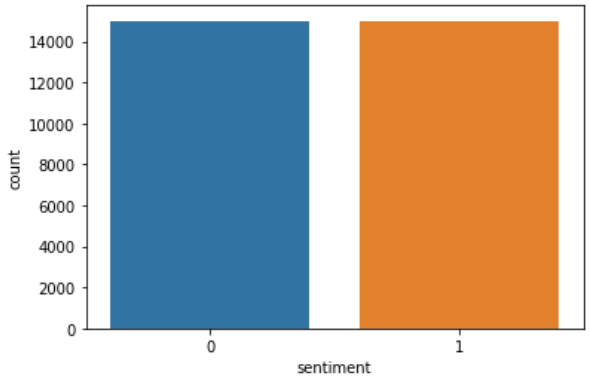


Figure 3 - Distribution of input data

2.2 Feature extraction

Feature extraction is defined as the process of transforming a text from a human understandable form into a machine-readable form.

For this project I chose TF-IDF (term frequency-inverse document frequency) since it introduces weights in the vectorization. The weights are defined by the frequency of the term, and they are correlated with the importance given to a word.

TF stands for term frequency and is a vector of the frequencies of each word of the tweet divided by the highest frequency. IDF, on the other side, stands for frequency-inverse document frequency and it specifies how common a word is amongst the whole corpus. It Is equal to the logarithm of the number of documents divided by the number of documents that contain the word. The multiplication of both TF and IDF provide the model with a matrix containing representative features of the input data.

The process of TF-IDF will be explained with a small data frame of 7 tweets.

	tweets	sentiment
0	robbiebronniman sounds like a great night	1
1	damn the person who stolde my wallet may karma...	1
2	greetings from the piano bench photo	1
3	drewryanscott i love it i love you haha forget...	1
4	kissthestars pretty pretty pretty please pakid...	0
5	really upset	0
6	lilyroseallen big pool or paddling pool might ...	1

Figure 4

First, the vectorizer is created with the function*TfidfVectorizer().* As tweets are relatively short and there are already a great deal of features, I set the ngram range to (1,1) so we will only manage unigrams, and maximum feature to 10000. Also, stop words will be deleted from each tweet to keep only the more relevant words.

The function *vectorizer.fit(X)* creates a vocabulary for the data frame X. The vocabulary consists on the words of the data frame along with their corresponding column in the *features_matrix.* As we can see, after calling the function *vectorizer.transform(X),* a matrix of size 7x77 is created. It means that there are 7 documents and 77 unique words spread among them.

```
{'robbiebronniman': 59, 'sounds': 64, 'like': 34, 'great': 20, 'night': 46, 'damn': 13, 'the': 68, 'person': 51, 'who': 73, 'st
olde': 65, 'my': 45, 'wallet': 72, 'may': 39, 'karma': 30, 'come': 12, 'back': 6, 'and': 1, 'bite': 10, 'you': 76, 'in': 26, 'a
ss': 4, 'greetings': 21, 'from': 17, 'piano': 53, 'bench': 8, 'photo': 52, 'drewryanscott': 15, 'love': 36, 'it': 28, 'haha': 2
2, 'forget': 16, 'hugyou': 24, 'should': 60, 'give': 19, 'me': 40, 'kissno': 31, 'lie': 33, 'please': 54, 'would': 75, 'be': 7,
'awesome': 5, 'if': 25, 'did': 14, 'kissthestars': 32, 'pretty': 56, 'pakidownload': 50, 'ito': 29, 'then': 69, 'reupload': 58,
'someother': 63, 'site': 61, 'mediafire': 41, 'hindi': 23, 'mgwork': 42, 'ang': 2, 'mu': 44, 'skin': 62, 'really': 57, 'upset':
71, 'lilyroseallen': 35, 'big': 9, 'pool': 55, 'or': 47, 'paddling': 49, 'might': 43, 'able': 0, 'to': 70, 'manage': 38, 'paddl
e': 48, 'surroundings': 66, 'but': 11, 'that': 67, 'is': 27, 'as': 3, 'luxury': 37, 'will': 74, 'get': 18}
(7, 77)
```

Figure 5

Printing X gives the following result. The first column is related to each word in each document, and the second is the individual TF-IDF weight.

```
(0, 64)       0.4618042361109319
(0, 59)       0.4618042361109319
(0, 46)       0.4618042361109319
(0, 34)       0.38333717539523177
(0, 20)       0.4618042361109319
(1, 76)       0.20237993754078729
(1, 73)       0.24380602367574056
(1, 72)       0.24380602367574056
(1, 68)       0.40475987508157457
(1, 65)       0.24380602367574056
(1, 51)       0.24380602367574056
(1, 45)       0.20237993754078729
(1, 39)       0.24380602367574056
(1, 30)       0.24380602367574056
(1, 26)       0.20237993754078729
(1, 13)       0.24380602367574056
(1, 12)       0.24380602367574056
(1, 10)       0.24380602367574056
(1, 6)        0.24380602367574056
(1, 4)        0.24380602367574056
(1, 1)        0.24380602367574056
```

Figure 6

After a data frame to visualize better the data, we can see that the result is a sparse matrix with all the words spread in the X-axis and the tweets in the Y-axis.

	able	and	ang	as	ass	awesome	back	be	bench	big	...	that	the	then	to	upset	wallet	who	will	w
0	0.0	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	...	0.0	0.000000	0.000000	0.0	0.0	0.000000	0.000000	0.0	0.00
1	0.0	0.243806	0.000000	0.0	0.243806	0.000000	0.243806	0.000000	0.000000	0.0	...	0.0	0.404760	0.000000	0.0	0.0	0.243806	0.243806	0.0	0.00
2	0.0	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.419257	0.0	...	0.0	0.348019	0.000000	0.0	0.0	0.000000	0.000000	0.0	0.00
3	0.0	0.000000	0.000000	0.0	0.000000	0.204592	0.000000	0.169829	0.000000	0.0	...	0.0	0.000000	0.000000	0.0	0.0	0.000000	0.000000	0.0	0.20
4	0.0	0.000000	0.206821	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	...	0.0	0.000000	0.206821	0.0	0.0	0.000000	0.000000	0.0	0.00

Figure 7

Note: at this point, I realized that I would have to process the data because some special characters were included in the input file.

:

	09	0ff	10	100	100th	101	102	1030	10am	10k	...	âã	äi	ii	ï½	ð²	ð¼ðµð½ñ	ð½ð	ðð	ðð¾	ññð¾
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 8 - Unprocessed input file

```
#delete special characters and numbers from the tweets
X = X.apply(lambda x: re.sub (r'([^a-zA-Z ]+?)', '', x))
```

Figure 9 - Applying regular expressions to delete special characters and numbers

3. Model fitting

To fit the model, the data is divided so 80% of it will be dedicated to training, ant the remaining 20% will be used in testing phase.

```
def report(model):
    test_predict = model.predict(X_test)
    print(classification_report(y_test, test_predict))
    print(accuracy_score(y_test, test_predict))
    cf_matrix = confusion_matrix(y_test, test_predict)
    sns.heatmap(cf_matrix, annot = True)
```

Figure 10 - X_test is used to get the performance of the model with unseen data

The classification report allows us to understand the how the model predicts true data. As we sill see in the next pictures, *Recall* returns the number of true positives divided by the sum of true positives plus false negatives. It is how we measure what the model is recalling from the true data. On the other side, precision is the ratio between true positives and the sum of all positives. Finally, f1 score is the harmonic mean between these two values.

On the other hand, confusion matrix is a visual way to depict the number of true positives, false positives, true negatives and false negatives. The clearer a point is, the highest the values are.

For this project I used three different functions to classify the data. First, Naïve Bayes, which calculates the class membership using probabilities using words from a dictionary with all the words in the corpus. The probabilities are used using the Bayes theorem. Then, support vector machines classifier (SVM), that uses one perceptron to build a separation line in a linearly separable problem. It is a very powerful classification tool compatible with TF-IDF because it requires vectors with fixed size. Finally, logistic regression is tested too because it is used to classify binary variables.

System diagnostics

```
84]: report(NBmodel)
```

	precision	recall	f1-score	support
0	0.73	0.77	0.75	2979
1	0.76	0.72	0.74	3021
accuracy			0.75	6000
macro avg	0.75	0.75	0.75	6000
weighted avg	0.75	0.75	0.75	6000

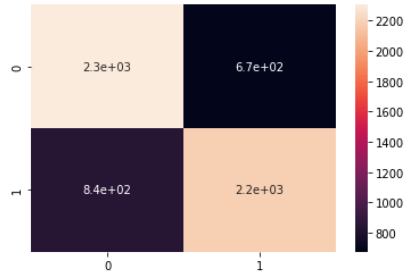


Figure 11 - Results of Naive Bayes function

```
report(SVC)
```

	precision	recall	f1-score	support
0	0.75	0.74	0.74	2979
1	0.74	0.76	0.75	3021
accuracy			0.75	6000
macro avg	0.75	0.75	0.75	6000
weighted avg	0.75	0.75	0.75	6000

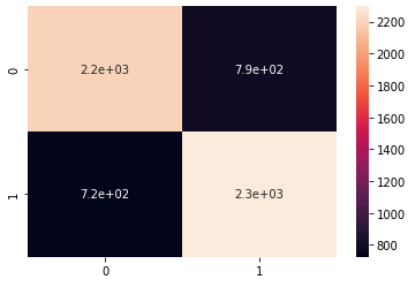


Figure 12 - results of SVM function

```
report(LRmodel)
```

	precision	recall	f1-score	support
0	0.77	0.74	0.76	3000
1	0.75	0.79	0.77	3000
accuracy			0.76	6000
macro avg	0.76	0.76	0.76	6000
weighted avg	0.76	0.76	0.76	6000

```
0.7611666666666667
```

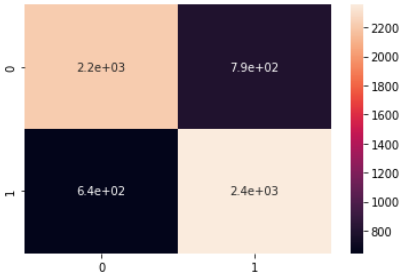


Figure 13 - Results of linear regression model

The results are similar. As we can see from the confusion matrices, the number of true positives and true negatives is way higher than false predictions. The total number of targets is equal to 6000, which corresponds to the 20% of the testing data we had split before. In all of the models, the positive tweets that were predicted correctly are around 2300, like the negative tweets.

The accuracy is also similar between the three models, but slightly higher with linear regression.

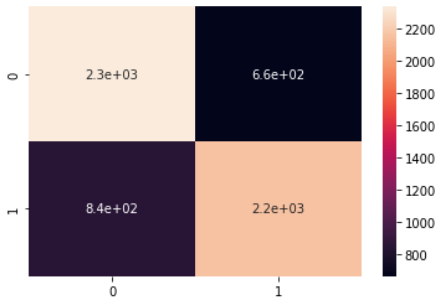
Another test was to change the ngram_range parameter in the vectorizer and set it to (1,2). This way, the features matrix will store both unigrams and bigrams. Unfortunately, I was restricted by the memory usage since the matrix was huge. For this reason, the next reports were done with half of the dataset (15000 tweets).

System diagnostics

```
: report(NBmodel)
```

	precision	recall	f1-score	support
0	0.73	0.78	0.76	3000
1	0.76	0.72	0.74	3000
accuracy			0.75	6000
macro avg	0.75	0.75	0.75	6000
weighted avg	0.75	0.75	0.75	6000

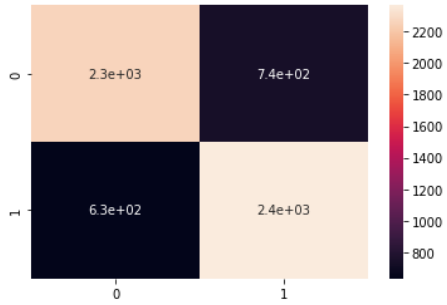
```
0.7488333333333334
```




```
]: report(SVC)
```

	precision	recall	f1-score	support
0	0.78	0.75	0.77	3000
1	0.76	0.79	0.78	3000
accuracy			0.77	6000
macro avg	0.77	0.77	0.77	6000
weighted avg	0.77	0.77	0.77	6000

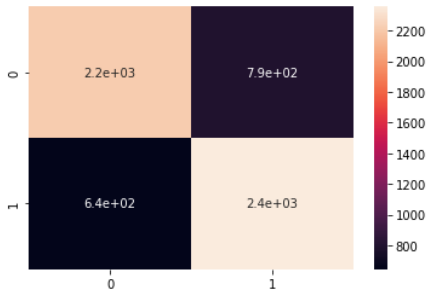
```
0.7718333333333334
```




```
]: report(LRmodel)
```

	precision	recall	f1-score	support
0	0.77	0.74	0.76	3000
1	0.75	0.79	0.77	3000
accuracy			0.76	6000
macro avg	0.76	0.76	0.76	6000
weighted avg	0.76	0.76	0.76	6000

```
0.7611666666666667
```



The results are similar as before, except for one remarkable thing which is that SVC sees an improvement of 3% in its f1-score both for true negatives and true positives.

4. Example of execution

```
: x = 'congratulations my friend'
vec = vectorizer.transform([x])
print(vec)
```

System diagnostics

```
In [309]: LRmodel.predict(vec)
Out[309]: array([1], dtype=int64)

In [310]: NBmodel.predict(vec)
Out[310]: array([1], dtype=int64)

In [311]: SVC.predict(vec)
Out[311]: array([1], dtype=int64)
```

Figure 14 - All models predict that the tweet is positive

```
: x = 'working on a friday night is so sad'
  vec = vectorizer.transform([x])
  print(vec)

    (0, 9827)     0.45729806410455925
    (0, 7936)     0.2831070691442142
    (0, 7364)     0.4129118004617147
    (0, 6106)     0.27464170996315157
    (0, 5869)     0.3883205787558402
    (0, 4320)     0.21698956256008733
    (0, 3109)     0.5166500769471422

: LRmodel.predict(vec)

: array([0], dtype=int64)

: NBmodel.predict(vec)

: array([0], dtype=int64)

: SVC.predict(vec)

: array([0], dtype=int64)
```

Figure 15 - All models predict that the tweet is negative