

Open in app ↗

Sign up

Sign In



Search Medium



Published in Towards Data Science



Patrizia Castagno

Follow

Oct 26, 2020 · 8 min read · Listen



Save



How to identify Spam using Natural Language Processing (NLP)?

Email Spam Detection using Natural Language Processing with Python

Humans master millions of words, but computationally speaking: how can we manipulate large amounts of text using programming techniques?

The idea that computers can understand ordinary languages and hold conversations with human beings has been a staple of science fiction. However, the first half of the twentieth century and was envisaged in a classic paper by Alan Turing (1950) as a hallmark of computational intelligence.

This article will focus on how computer systems can analyze and interpret texts, using the Natural Language Processing (NLP). For that, you should install Natural Language Toolkit, you can do it from <http://nltk.org>. Instructions are available on the cited website along with details of associated packages that need to be installed as well, including Python itself, which is also freely available.

What is Natural Language Processing (NLP) ?

Natural Language processing or NLP is a subset of Artificial Intelligence (AI), where it is basically responsible for the understanding of human language by a machine or a robot.



306



One of the important subtopics in NLP is *Natural Language Understanding (NLU)* and the reason is that it is used to understand the structure and meaning of human language, and then with the help of computer science transform this linguistic knowledge into algorithms of Rules-based machine learning that can solve specific problems and perform desired tasks.

LANGUAGE PROCCESS IN PYTHON

The purpose of this article is to show you how to detect spam in SMS.

For that, we use a dataset from the UCI datasets, which is a public set that contain SMS labelled messages that have been collected for mobile phone spam research. It has one collection composed by 5.574 SMS phone messages in English, tagged according being legitimate (ham) or spam.

Therefore, we will train a model to learn to automatically discriminate between ham / spam. Then we will use “test data” to test the model. Finally to evaluate if our model is efficient, we will calculate Accuracy, Classification report and Confusion Matrix.

Exploratory Data Analysis

To get started, we should first imports all the library, then load the data and rename names columns:

```
# Import library

import pandas as pd
import numpy as np
import string
import seaborn as sns
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from collections import Counter
from sklearn.metrics import classification_report, confusion_matrix
```

```
from sklearn.model_selection import GridSearchCV
%matplotlib inline

# Load data
data = pd.read_excel('data.xlsx')

# Rename names columns
data.columns = ['label', 'messages']
```

Let's see a description of our data:

```
data.describe()
```

	label	messages
count	5574	5574
unique	2	5171
top	ham	Sorry, I'll call later
freq	4827	30

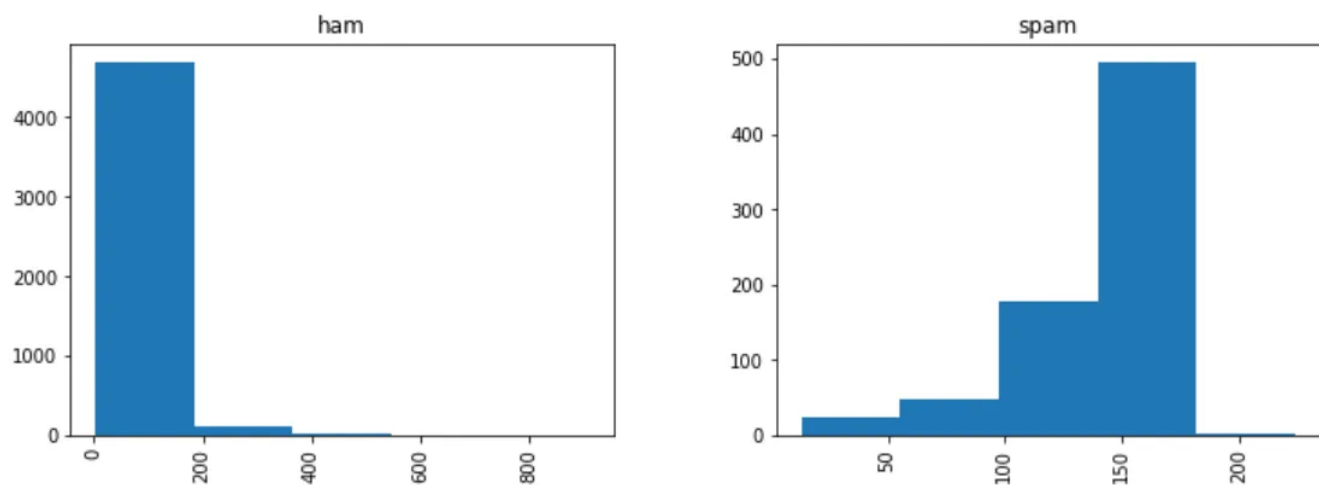
Data Description (Image by Author)

Note that our data contains a collection of 5574 SMS and also we have only 2 label: ham and spam. Now, we create a column called 'length' to know how long the text messages are and then plot it against the label:

```
data["length"] = data["messages"].apply(len)
data.sort_values(by='length', ascending=False).head(10)
```

	label	messages	length
1085	ham	For me the love should start with attraction.i...	910
1863	ham	The last thing i ever wanted to do was hurt yo...	790
2434	ham	Indians r poor but India is not a poor country...	629
1579	ham	How to Make a girl Happy? It's not at all diff...	611
2849	ham	Sad story of a Man - Last week was my b'day. M...	588
2158	ham	Sad story of a Man - Last week was my b'day. M...	588
2380	ham	Good evening Sir, hope you are having a nice d...	482
3017	ham	<#> is fast approaching. So, Wish u a v...	461
1513	ham	Hey sweet, I was wondering when you had a mome...	458
2370	ham	A Boy loved a gal. He propsd bt she didnt mind...	446

```
data.hist(column = 'length', by = 'label', figsize=(12,4), bins = 5)
```



Histogram between lenght and label (Image by Author)

Note that through the histogram, we have been able to discover that spam messages tend to have more characters.

Most likely, most of the data you have come across is numeric or categorical, but what happens when it is of type string (text format)?

As you may have noticed, our data is of type string. Therefore, we should transform it into a numeric vector to be able to perform the classification task. To do this, we use **bag-of-words** where each unique word in a text will be represented by a number. However, before doing this transformation, we should remove all punctuations and then common words like: ['I', 'my', 'myself', 'we', 'our', 'our', 'ourselves', 'you', 'are' ...]. This process is called **tokenization**. After this process, we convert our string sequence into number sequences.

1. **Remove all punctuation:** Suppose we have the following sentence:

*******Hi everyone!!! it is a pleasure to meet you.*******

and we want to remove **!!!** and **.**

First, we load the import string library and do the following:

```
message = "Hi everyone!!! it is a pleasure to meet you."
message_not_punc = []
for punctuation in message:
    if punctuation not in string.punctuation:
        message_not_punc.append(punctuation)
# Join the characters again to form the string.
message_not_punc = ''.join(message_not_punc)
print(message_not_punc)

>>> Hi everyone it is a pleasure to meet you
```

2. **Remove common words:**

To do that, we use the library nltk, i.e, **from nltk.corpus import stopwords**

Is important to know that **stopwords** have 23 languages supported by it (this number must be up to date). In this case, we use English language:

```

from nltk.corpus import stopwords

# Remove any stopwords for remove_punc, but first we should to
transform this into the list.

message_clean = list(message_not_punc.split(" "))

# Remove any stopwords

i = 0
while i <= len(message_clean):
    for mess in message_clean:
        if mess.lower() in stopwords.words('english'):
            message_clean.remove(mess)

    i = i + 1

print(message_clean)

>>> ['Hi', 'everyone', 'pleasure', 'meet']

```

Thus, with the steps 1 and 2, we can create the following function:

```

def transform_message(message):
    message_not_punc = [] # Message without punctuation
    i = 0
    for punctuation in message:
        if punctuation not in string.punctuation:
            message_not_punc.append(punctuation)
    # Join words again to form the string.
    message_not_punc = ''.join(message_not_punc)

    # Remove any stopwords for message_not_punc, but first we should
    # to transform this into the list.
    message_clean = list(message_not_punc.split(" "))
    while i <= len(message_clean):
        for mess in message_clean:
            if mess.lower() in stopwords.words('english'):
                message_clean.remove(mess)

        i = i + 1
    return message_clean

```

Now, we can apply the above function to our data analysis in the following way:

```
data['messages'].head(5).apply(transform_message)
```

```
>>>
0    [Go, jurong, point, crazy, Available, bugis, n...
1                                [Ok, lar, Joking, wif, u, oni]
2    [Free, entry, 2, wkly, comp, win, FA, Cup, fin...
3    [U, dun, say, early, hor, U, c, already, say]
4    [Nah, dont, think, goes, usf, lives, around, t...
Name: messages, dtype: object
```

Vectorization

Note that, we have the messages as lists of tokens. So, the next step is to convert each of those messages into a vector.

To do that, we use **CountVectorizer** from Scikit Learn. CountVectorizer converts a collection of documents to an array of token counts. If you want to see an example of countvectorizer in Python, we invite you to read the following article: [CountVectorizer in Python](#)

First all of, we import CountVectorizer from sklearn learn:

```
from sklearn.feature_extraction.text import CountVectorizer
```

CountVectorizer has many parameters, but we only use “**analyzer**”, which is our own previously defined function:

```
vectorization = CountVectorizer(analyzer = transform_message )
X = vectorization.fit(data['messages'])
```

Now, we should transform the entire DataFrame of the messages in a vector representation. For that, we use **transform** function:

```
X_transform = X.transform([data['messages']])
print(X_transform)
>>> :      :
      (0, 11383)      9
```

(0, 11384)	20
(0, 11385)	14
(0, 11386)	2
(0, 11387)	4
(0, 11391)	11
(0, 11393)	5
(0, 11396)	1
(0, 11397)	1
(0, 11398)	18
(0, 11399)	18
(0, 11405)	2
(0, 11408)	1
(0, 11410)	1
(0, 11411)	8
(0, 11412)	7
(0, 11413)	1
(0, 11414)	1
(0, 11415)	27
(0, 11417)	3
(0, 11418)	104
(0, 11420)	9
(0, 11422)	1
(0, 11423)	7
(0, 11424)	1

TF-IDF

After applying “CountVectorizer” on our data, we use TF-IDF. Surely, you are wondering what is TD-FT? and why should we use it? Let me explain you:

TF-IDF is the abbreviation of *Inverse document frequency* is a numerical measure that expresses how relevant a word is to a document in a collection.

The value of TF-IDF increases proportionally to the number of times a word appears in the document, and is offset by the frequency of the word in the document collection, which allows handling of the fact that some words are generally more common than others.

How to Compute TF-IDF ?

TF-IDF is composed by two terms: the first term is the **Term Frequency (TF)** and the second is the **Inverse Document Frequency (IDF)**:

Term Frequency (TF): Which measures how frequently a term occurs in a document, that is, the number of times a word appears in a document, divided by the total number of words in that document:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)

Inverse Document Frequency (IDF): Measures how important a term is, computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

IDF(t) = \log_e (Total number of documents / Number of documents with term t in it)

For more information, please refer to [“What does tf-idf mean?”](#)

Continuing with our code, we import TfidfVectorizer from `sklearn.feature_extraction.text` and then:

```
tfidf_transformer = TfidfTransformer().fit(X_transform)
```

To transform the entire bag-of-words corpus into TF-IDF corpus at once:

```
X_tfidf = tfidf_transformer.transform(X_transform)
print(X_tfidf.shape)

>>> (5572, 11425)
```

Classification Model

Having the features represented as vectors, we can finally train our spam/ham classifier. You can use any classification algorithms . Here we use Support Vector Classification (SVC) algorithm.

First, we split the data into train and test data. We take 80 % (0.80) training data and 30% (0.30) test data and then we fit the model using SVC:

```
X_train, X_test, y_train, y_test = train_test_split(X_tfidf,
data['messages'], test_size=0.30, random_state = 50)
clf = SVC(kernel='linear').fit(X_train, y_train)
```

Test model

To test the model we use X_test previously calculated:

```
predictions = clf.predict(X_test)
print('predicted', predictions)

>>> predicted ['spam' 'ham' 'ham' ... 'ham' 'ham' 'spam']
```

Is our model reliable?

The question that arises is: our model reliable across the entire data set?

For that, we can use SciKit Learn's built-in classification report, which returns

Precision, Recall, F1-Score and also **Confusion Matrix**

```
from sklearn.metrics import classification_report
print (classification_report(y_test, predictions))
```

	precision	recall	f1-score	support
ham	0.98	1.00	0.99	1469
spam	1.00	0.86	0.92	203
accuracy			0.98	1672
macro avg	0.99	0.93	0.96	1672
weighted avg	0.98	0.98	0.98	1672

Classification Report (Image by Author)

```
from sklearn.metrics import confusion_matrix  
print(confusion_matrix(y_test, predictions))
```

```
[[1469    0]  
 [  29   174]]
```

Confusion Matrix (Image by Author)

The accuracy is a good way to know if the model is efficient but this is not enough to say that our model is good, for this reason we have used Classification Report and Confusion Matrix. You can see that results obtained are quite good.

We're done !!! I hope this information has been useful to you. If you are interested in the full code, you can get it from the link below: [How to identify Spam using NLP](#)

Spam Filter

NLP

Tokenization

Tfidf Vectorizer


Bag Of Words

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.

 Get this newsletter

Get the Medium app

