1_NLP_STEPS

March 23, 2021

1 Natural Language Processing (NLP)

1.1 Classifiying SMS Spam

First we have to install nltk.

Second, we download data from UCI, SMS SSpam Collection Data Set.

https://archive.ics.uci.edu/ml/datasets/sms+spam+collection

I alrady download that and put the files in the datasets folder.

1.2 The big picture, what are we going to do here?

At this point, we have learnt many classifiers. We need to train them with train data.

The train data shall have features and label. For the dataset here, we will see that they come with label already. They also provide SMS messages. The sms is string. We will try to convert these strings to something measurable and meaningful. We will create features from the message. Hopefully, the classifiers can utilize these features in the training process. This is also call **FEATURE ENGINEERING**.

2 Feature Engineering

First, let's load data and see what features we can construct from the string.

[83]:
```
import pandas as pd
import numpy as np
import seaborn as sns
```

[84]:
```
# Load data from the file into a dataframe
data = pd.read_csv('./datasets/SMSSpamCollection', sep = '\t', header=None,␣
↪names=["label", "sms"])
data.head(3)
```

[84]:
```
  label                                                sms
0   ham  Go until jurong point, crazy.. Available only ...
1   ham                      Ok lar... Joking wif u oni...
2  spam  Free entry in 2 a wkly comp to win FA Cup fina...
```

[85]:
```python
# Re-arrange the columns such that label is the last one
df = pd.DataFrame()
df["text"] = data.sms
df["label"] = data.label
```

[86]:
```python
df.head()
```

[86]:
```
                                                text label
0  Go until jurong point, crazy.. Available only ...   ham
1                      Ok lar... Joking wif u oni...   ham
2  Free entry in 2 a wkly comp to win FA Cup fina...  spam
3  U dun say so early hor... U c already then say...   ham
4  Nah I don't think he goes to usf, he lives aro...   ham
```

[87]:
```python
df.describe()
```

[87]:
```
                          text label
count                     5572  5572
unique                    5169     2
top     Sorry, I'll call later   ham
freq                        30  4825
```

[88]:
```python
# Notice the proportion of ham vs spam. If one class is too small, the training␣
↪process might not go well.
df.groupby('label').describe()
```

[88]:
```
       text
      count unique                                               top freq
label
ham    4825   4516                            Sorry, I'll call later   30
spam    747    653  Please call our customer service representativ...    4
```

2.1 Can't we just use the text as our feature?

Each sms message is a series of characters. Those characters may form sentence(s), emoji, paragraph.

Since we need numbers to be used to train the classifier, how can we convert to text to a number.

2.1.1 IDEA#1 :

Replace each text with a number (aka: a is 1, b is 2,. . . . z is 26)

1-5 6-10 11-15 16-20 21-25 26

abcde fghij klmno pqrst uvwxy z

fun : can be represented as: f->6 u->21 n->14 = 62114

This idea sounds good at first but if we look at the values. The hardly represent the original meaning.

Ex:

go : 715

went : 2351420

tasty : 201192025

yummy : 2521131325

Here, we just look at very simple words. Imagine that we have the whole sentence, the value will be greatly different even the meaning is 99% the same.

2.1.2 IDEA#2 :

We count the length of each text (sms)

[89]:
```
# Let's try with one simple word
aaa = "Hello"
len(aaa)
```

[89]:
```
5
```

[90]:
```
# Now we calculate the length of all messages
df['length']=df['text'].apply(len)
df.head()
```

[90]:
```
                                               text label  length
0  Go until jurong point, crazy.. Available only ...   ham     111
1                      Ok lar... Joking wif u oni...   ham      29
2  Free entry in 2 a wkly comp to win FA Cup fina...  spam     155
3  U dun say so early hor... U c already then say...   ham      49
4  Nah I don't think he goes to usf, he lives aro...   ham      61
```

[91]:
```
# The average length of the messages
df['length'].mean()
```

[91]:
```
80.48994974874371
```

2.2 Whitespace

The length can be misleading if there are whitespace characters. This dataset doesn't have problems with whitespace. However, let's see if we need to. clean that how could we do it.

[92]:
```
# Let's strip whitespace out
"   HELLO MUIC ".strip()
```

[92]:
```
'HELLO MUIC'
```

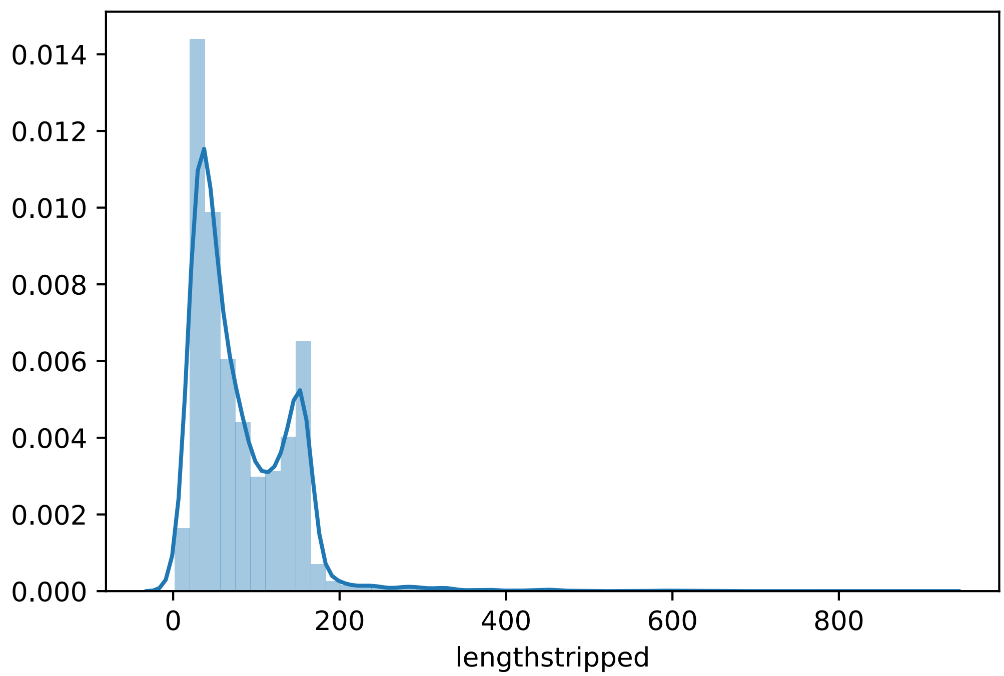
[93]:
```
df['textstripped']=df['text'].str.strip()
df['lengthstripped']=df['textstripped'].apply(len)
```

```
df['lengthdiff']=df['length']-df['lengthstripped']
df['lengthdiff'].head()
```

[93]:
```
0    0
1    0
2    0
3    0
4    0
Name: lengthdiff, dtype: int64
```

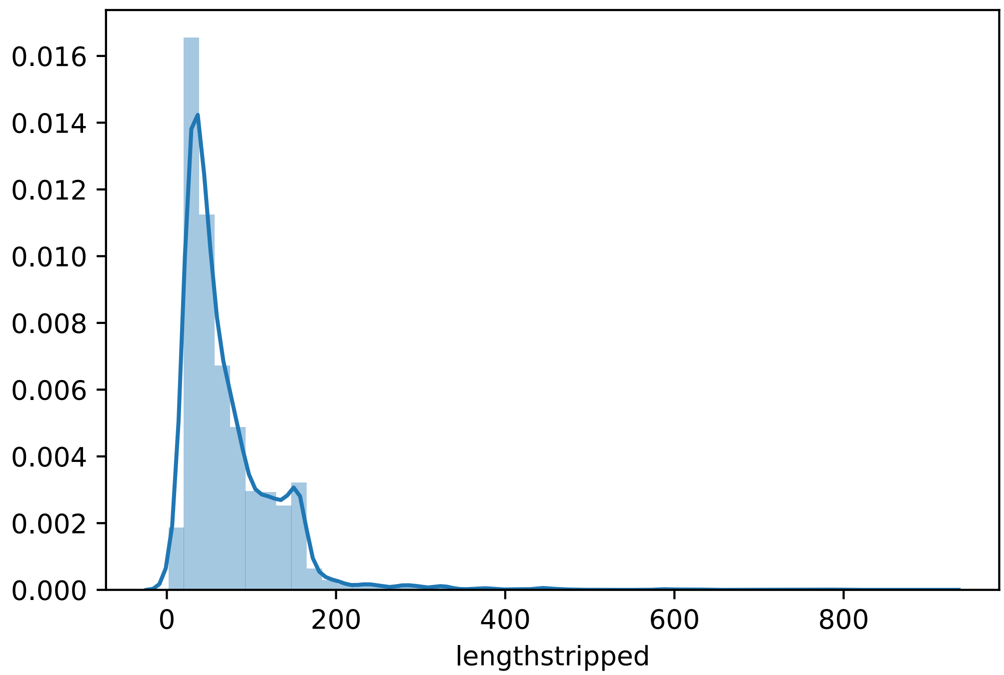
[94]:
```
# Let's visualize the data
sns.distplot(df['lengthstripped'])
```

[94]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd132bfb780>



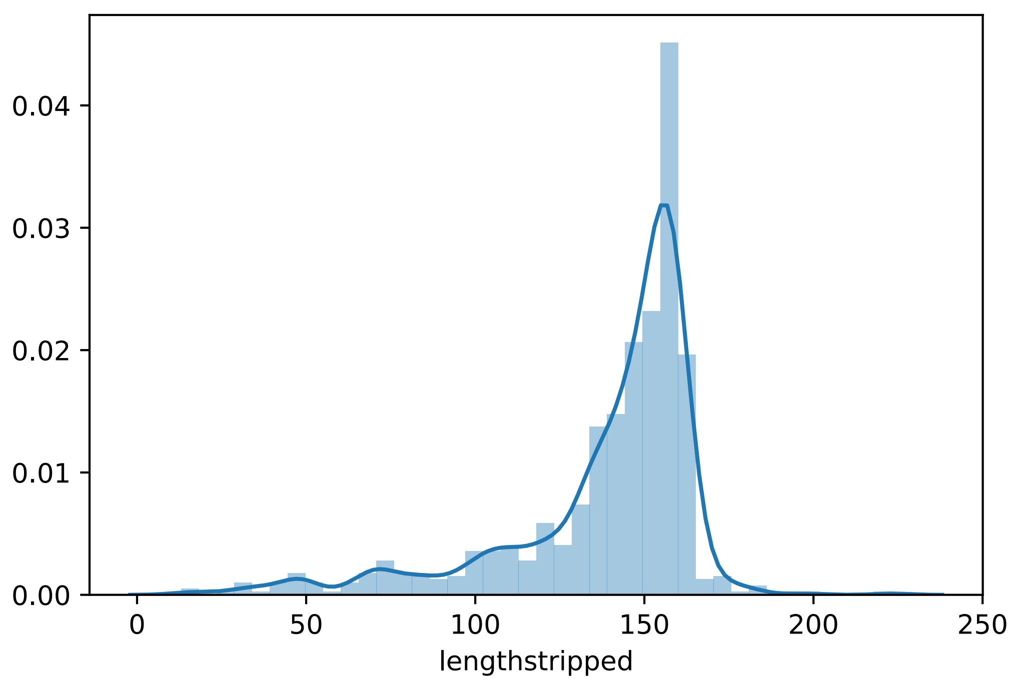
[95]:
```
ham = df[df['label']=='ham']
spam = df[df['label']=='spam']
sns.distplot(ham['lengthstripped'])
```

[95]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd0f1783278>



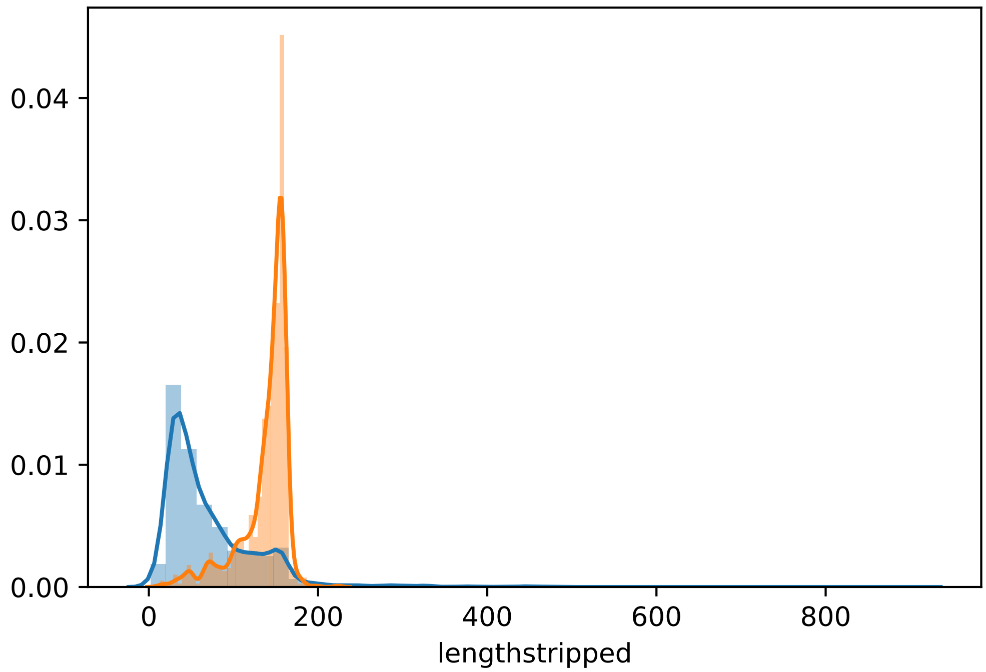
[96]:
```
sns.distplot(spam['lengthstripped'])
```

[96]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd101b8fa90>



[97]:
```
sns.distplot(ham['lengthstripped'])
sns.distplot(spam['lengthstripped'])
```

[97]: `<matplotlib.axes._subplots.AxesSubplot at 0x7fd0f1acfd30>`



At this point, the length may be used to classify the SMS messages. However, the performance may not be that good.

2.3 Let's process the data in a advanced way

[98]:
```
import nltk
import string
nltk.download('stopwords')
nltk.download('punkt') # tokenizer
```

```
[nltk_data] Downloading package stopwords to /Users/tix/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /Users/tix/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

[98]: True

2.4 Stop Words

[99]:
```
stopwords = nltk.corpus.stopwords.words('english')
len(stopwords)
```

[99]: 179

[100]:
```
# Let's take a look at some stop words
stopwords[50:60]
```

[100]: ['been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing']

2.5 Punctuation

[101]:
```
punctuation = string.punctuation
punctuation
```

[101]: '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'

2.6 Pre process data

[102]:
```
def pre_process(sms):
    # first, we move the punctuation
    remove_punct = "".join([word.lower() for word in sms if word not
                   in punctuation])
    # second, we chop the into words
    tokenize = nltk.tokenize.word_tokenize(remove_punct)
    # we remove the words that don't carry much meaning (aka: they are too␣
 ↪general)
    remove_stopwords = [word for word in tokenize if word not in
                        stopwords]
    return remove_stopwords
```

[103]:
```
#Let's test it
pre_process("I am a student.")
```

[103]: ['student']

[104]:
```
pre_process("I am a student at MUIC. I am learning about NLP today.")
```

[104]: ['student', 'muic', 'learning', 'nlp', 'today']

[105]:
```
pre_process("I am a student at MUIC. I am learning about NLP today. Now all␣
 ↪students can join.")
```

[105]: ['student', 'muic', 'learning', 'nlp', 'today', 'students', 'join']

```
[106]: temp = pre_process("I am studying this. Some studied about this before. Some␣
       ↪will study next term.")
       temp
```

```
[106]: ['studying', 'studied', 'study', 'next', 'term']
```

```
[107]: df['textstripped'].head(3).apply(pre_process)
```

```
[107]: 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...
       Name: textstripped, dtype: object
```

2.7 Vectorize

With all the remaining words, we give id to each word. So, for each message, we will represents with the IDs of the words in that message.

```
[108]: from sklearn.feature_extraction.text import CountVectorizer
```

```
[109]: # Let's create a transformer
       bow_transformer = CountVectorizer(analyzer=pre_process).fit(df['textstripped'])
```

```
[110]: len(bow_transformer.vocabulary_)
```

```
[110]: 9506
```

```
[111]: # Let's try a transformer
       # Will randomly pick one SMS from the index 555
       # Even if this is just a random message... I will call this the chosen sms !!!
       the_chosen_sms = df['textstripped'][555]
       the_chosen_sms
```

```
[111]: 'I‘ll have a look at the frying pan in case it‘s cheap or a book perhaps. No
       that‘s silly a frying pan isn‘t likely to be a book'
```

```
[112]: the_chosen_sms_bow = bow_transformer.transform([the_chosen_sms])
```

```
[113]: print(the_chosen_sms_bow)
```

```
  (0, 1713)     2
  (0, 2020)     1
  (0, 2108)     1
  (0, 3623)     2
  (0, 4996)     1
  (0, 5088)     1
  (0, 6160)     2
  (0, 6273)     1
```

```
  (0, 7467)	1
  (0, 9501)	4
```

[114]:
```
# Let's see the
bow_transformer.get_feature_names()[1713]
```

[114]:
```
'book'
```

[115]:
```
bow_transformer.get_feature_names()[9501]
```

[115]:
```
'‘'
```

2.8 TF-IDF

Words represent something deeper than just "having" the word or "not having" the word. Hans Peter Luhn suggested that we can try to identify that the word is important or not base on the context of the messages.

TF (Term Frequency) : # of that terms / # all terms in that documents

IDF (Inverse Document Frequency) : Log(number of documents / number of documents having this keyword)

TF-IDF = TF x IDF

[121]:
```
# Let's transform and get the count of all messages
sms_bow = bow_transformer.transform(df['textstripped'])
```

[122]:
```
type(sms_bow)
```

[122]:
```
scipy.sparse.csr.csr_matrix
```

[123]:
```
sms_bow.shape
```

[123]:
```
(5572, 9506)
```

[124]:
```
# let's see non-zero
sms_bow.nnz
```

[124]:
```
50198
```

[125]:
```
from sklearn.feature_extraction.text import TfidfTransformer
```

[126]:
```
tfidf_tfm = TfidfTransformer().fit(sms_bow)
```

[127]:
```
# let's transform the_sms
tfidf_the_chosen_sms = tfidf_tfm.transform(the_chosen_sms_bow)
```

[128]:
```
print(tfidf_the_chosen_sms)
```

```
  (0, 9501)     0.5936254801026182
  (0, 7467)     0.21297259302845367
  (0, 6273)     0.20330536064874724
  (0, 6160)     0.42594518605690734
  (0, 5088)     0.14686762139154752
  (0, 4996)     0.19644635011922718
  (0, 3623)     0.4066107212974945
  (0, 2108)     0.1723274286802215
  (0, 2020)     0.1649326129348811
  (0, 1713)     0.31160237154198994
```

[48]:
```python
# Now, do it for all
data_tfidf = tfidf_tfm.transform(sms_bow)
```

[132]:
```python
data_tfidf.shape
```

[132]:
```
(5572, 9506)
```

3 Train the model

[133]:
```python
from sklearn.naive_bayes import MultinomialNB
```

[134]:
```python
clf = MultinomialNB().fit(data_tfidf, df['label'])
```

[53]:
```python
clf.predict(the_chosen_sms_bow)
```

[53]:
```
array(['ham'], dtype='<U4')
```

We can do all the steps above easily using Pipeline

[54]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df['textstripped'],␣
 ↪df['label'])
```

[55]:
```python
 from sklearn.pipeline import Pipeline
```

[56]:
```python
pipeline = Pipeline([
    ('bow',CountVectorizer(analyzer=pre_process)),
    ('tfidf',TfidfTransformer()),
    ('classify',MultinomialNB())
])
```

[57]:
```python
pipeline.fit(X_train,y_train)
```

[57]:
```
Pipeline(memory=None,
     steps=[('bow', CountVectorizer(analyzer=<function pre_process at
0x7fd125908598>,
        binary=False, decode_error='strict', dtype=<class 'numpy.int64'>,
```

```
        encoding='utf-8', input='content', lowercase=True, max_df=1.0,
        max_features=None, min_df=1, ngram_range=(1, 1),
preprocessor=Non..._tf=False, use_idf=True)), ('classify',
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True))])
```

[58]:
```
predictions = pipeline.predict(X_test)
```

[59]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(predictions,y_test)
```

[59]:
```
array([[1210,   58],
       [   1,  124]])
```

[60]:
```
from sklearn.metrics import classification_report
print(classification_report(predictions, y_test))
```

```
              precision    recall  f1-score   support

         ham       1.00      0.95      0.98      1268
        spam       0.68      0.99      0.81       125

   micro avg       0.96      0.96      0.96      1393
   macro avg       0.84      0.97      0.89      1393
weighted avg       0.97      0.96      0.96      1393
```

[ ]:
```

```