Natural_language_processing

June 28, 2021

1 Natural language Processing

Author: Jessica Cervi
Expected time = 2 hours
Total points = 80 points

1.1 Project Overview

This assignment provides an overview of *Natural Language Processing* (NLP) as an approach to classification problems in supervised learning involving textual data (in particular using *Naive Bayes classifiers*). In spite of the "naive" assumptions involved, Naive Bayes works very well in practice particularly for text analysis in, for instance, spam filtering or document classification. As such, for this assignment, you will build a very simple model of spam filtering to get a sense of how naive Bayes classification really works. Then, you will explore the use of tools within Scikit-Learn for NLP.

The primary goals of the current assignment are: + to become familiar with the terminology and tools available for NLP; + to practice the application of Bayes' theorem for probabilistic reasoning; and + to develop a (highly simplified) model of text analysis for spam classification using the naive Bayes classification framework.

This assignment is designed to build your familiarity and comfort coding in Python while also helping you review key topics from the module. As you progress through the assignment, answers will get increasingly complex. It is important that you adopt a data scientist's mindset when completing this assignment. **Remember to run your code from each cell before submitting your assignment.** Running your code beforehand will notify you of errors and give you a chance to fix your errors before submitting. You should view your Vocareum submission as if you are delivering a final project to your manager or client.

Vocareum Tips - Do not add arguments or options to functions unless you are specifically asked to. This will cause an error in Vocareum. - Do not use a library unless you are expicitly asked to in the question. - You can download the Grading Report after submitting the assignment. This will include feedback and hints on incorrect questions.

1.1.1 Learning Objectives

- Learn the main concepts behind the theory of Natural Language Processing
- Represent a document-term matrix in Python
- Learn the theory behind TD-IDF Vectorizer and its Python implementation
- Implement Bayes Theorem in Python

- Prearing text for analysis and distinguish between spam and ham messages
- Computing prions and likelihoods of your prediction
- Using a `Scikit-Learn` `MultinomialNB` Estimator

1.2 Index:

Natural language Processing

1.3 Natural language Processing

Natural Language Processing, usually shortened as NLP, is a branch of artificial intelligence that deals with the interaction between computers and humans using the natural language. The ultimate objective of NLP is to read, decipher, understand, and make sense of the human languages in a manner that is valuable.

In this assingment, we will guide your through your own implementation of an algorithm to analyze text.

As usual we begin by importing the necessery libraries.

```
In [23]: %matplotlib inline
         import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         import pathlib, re
```

To prepare for the eventual task of classifying text messages, here is a Python string containing the body of several text messages on separate lines:

```
In [24]: messages = '''Have a safe trip to Nigeria. Wish you happiness and very soon company to
         Well keep in mind I've only got enough gas for one more round trip barring a sudden in
         Yes i have. So that's why u texted. Pshew...missing you so much
         This school is really expensive. Have you started practicing your accent. Because its
         Sorry, I'll call later
         Anything lor. Juz both of us lor.
         Get me out of this dump heap. My mom decided to come to lowes. BORING.
         Why don't you wait 'til at least wednesday to see if you get your .
```

```
REMINDER FROM O2: To get 2.50 pounds free call credit and details of great offers pls
This is the 2nd time we have tried 2 contact u. U have won the č750 Pound prize. 2 cla
Pity, * was in mood for that. So...any other suggestions?'''
print(messages)
```

```
Have a safe trip to Nigeria. Wish you happiness and very soon company to share moments with
Well keep in mind I've only got enough gas for one more round trip barring a sudden influx of c
Yes i have. So that's why u texted. Pshew...missing you so much
This school is really expensive. Have you started practicing your accent. Because its importan
Sorry, I'll call later
Anything lor. Juz both of us lor.
Get me out of this dump heap. My mom decided to come to lowes. BORING.
Why don't you wait 'til at least wednesday to see if you get your .
REMINDER FROM O2: To get 2.50 pounds free call credit and details of great offers pls reply 2 t
This is the 2nd time we have tried 2 contact u. U have won the č750 Pound prize. 2 claim is eas
Pity, * was in mood for that. So...any other suggestions?
```

The content of the preceding string illustrates a lot of the challenges with **natural language processing** from text:

- The text needs to be split into individual *tokens* (i.e., words & punctuation).
- There can be difficulties with upper- & lower-case interspersed with numerals and punctuation characters.
- Many words add little contextual information such as articles, conjunctions, etc. These are *stop words*.
- Similar words can occur with common roots (e.g., 'go' and 'goes', 'liked' and 'likes' and 'liked', etc.
- Words can be spelled incorrectly.

Let's convert all the text to lower case and construct a list with the individual messages as a *corpus* of text.

As a reminder, **a text corpus is a large body of text**.

```
In [25]: corpus = messages.lower().split('\n')
         corpus
```

```
Out[25]: ['have a safe trip to nigeria. wish you happiness and very soon company to share momen
          "well keep in mind i've only got enough gas for one more round trip barring a sudden
          "yes i have. so that's why u texted. pshew...missing you so much",
          "this school is really expensive. have you started practicing your accent. because i
          "sorry, i'll call later",
          'anything lor. juz both of us lor.',
          'get me out of this dump heap. my mom decided to come to lowes. boring.',
          "why don't you wait 'til at least wednesday to see if you get your .",
          'reminder from o2: to get 2.50 pounds free call credit and details of great offers pl
          'this is the 2nd time we have tried 2 contact u. u have won the č750 pound prize. 2 c
          'pity, * was in mood for that. so...any other suggestions?']
```

1.4 Terminology

We'll use the following terms at various places in the assignment. More details & examples will be proveded where appropriate.

- *Stop Words* -- Specific words that are not considered important for text analysis, e.g., 'the', 'is', 'a', etc.
- *Tokenization* -- Segmentation of text into separate *tokens* (i.e., words or punctuation marks). This is a form of feature extraction.
- *Stemming* -- Reducing words to their root form by truncating characters, e.g., car, cars, car's, cars' all have *stem* 'car'.
- *Lemmatization* -- Grouping together the inflected forms of a word as a single item known as the *lemma* or dictionary form.
- *Word Embedding* -- Explicit mapping to represent sequences of tokens (words extracted from text) to vectors of real numbers.
- *n-grams* -- Sequences of words or tokens (i.e., phrases) rather than single words. Helps with better understanding of text; 'not happy' instead of 'happy,' e.g bi-gram per token. For example, the following sentence decomposes as shown into unigrams (1-grams) or bigrams (2-grams):

  Sentence: `The movie was not great.` Uni-grams: `[The, movie, was, not, great.]` Bi-grams: `[The movie, movie was, was not, not great.]`

1.5 Constructing a Word Embedding with the `CountVectorizer` class

Scikit-Learn provides many important tools for converting textual data to numerical data (as numerical data is required for most machine learning techniques). One such tool is the `CountVectorizer` class from the module `sklearn.feature_extraction.text`. The outputs of the `transform` and `fit_transform` methods of the `CountVectorizer` class are *document-term matrices* that contain word counts for each word in the corpus.

As an example, here is a (modified) excerpt from the Scikit-Learn `CountVectorizer` documentation.

```
In [26]: from sklearn.feature_extraction.text import CountVectorizer
         vectorizer = CountVectorizer()
         X = vectorizer.fit_transform(corpus)
         print(f"The object X obtained has type {type(X)}")
         print(f"The columns of X correspond to the words:\n{vectorizer.get_feature_names()}")
```

```
The object X obtained has type <class 'scipy.sparse.csr.csr_matrix'>
The columns of X correspond to the words:
['087187272008', '10p', '2nd', '4years', '50', '750', 'accent', 'and', 'any', 'anything', 'are
```

```
In [27]: X = X.toarray() # convert to dense representation
         print(X)
         print(f"After calling toarray(), the object X obtained has type {type(X)}")
```

```
[[0 0 0 ... 0 1 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 1 1 0]
 ...
 [0 0 0 ... 0 0 1]
 [1 1 1 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
After calling toarray(), the object X obtained has type <class 'numpy.ndarray'>
```

In [28]:
```python
# Represent document-term matrix using a DataFrame instead:
X = pd.DataFrame(data=X, columns=vectorizer.get_feature_names())
# Extract select columns
X[['and', 'contact', 'wednesday', 'yes', 'you']]
```

Out[28]:

	and	contact	wednesday	yes	you
0	1	0	0	0	1
1	0	0	0	0	0
2	0	0	0	1	1
3	1	0	0	0	4
4	0	0	0	0	0
5	0	0	0	0	0
6	0	0	0	0	0
7	0	0	1	0	2
8	2	0	0	0	0
9	0	1	0	0	0
10	0	0	0	0	0

Notice a few particular properties of the `CountVectorizer` class from the preceding example.

- Each column of the document-term matrix corresponds to a particular word (as displayed by the `get_feature_names` method).
- By default, the text is converted to lower case (e.g., "Wednesday" $\mapsto$ "wednesday").
- The specific vocabulary that determines the results of `get_feature_names` can be learned from some input text/corpus or predetermined when the object is instantiated (see documentation).
- Each row of the document-term matrix corresponds to a sentence from the original corpus.
- The numerical entries of the document-term matrix are nonnegative integers corresponding to counts of the occurrences of each word in the text corpus.
- The default input for the `fit` method is a list of strings or file objects corresponding to documents from which the distributions of word counts can be learned.
- The default output after applying the `fit_transform` method to text (or, equivalently, applying the `fit` method and subsequently applying the `transform` method to the same data) is a *sparse matrix* with only nonzero entries represented (i.e., to make storage more efficient). The purpose of calling the `toarray` method, then, is to transform the sparse matrix into a dense representation (i.e., by putting the zeros back in explicitly) for printing/display.

In the next exercise, you will use the Scikit-Learn `CountVectorizer` to create a document-term matrix from slightly different text. The corpus here is modelled using a list called `text` whose entries are sentences (strings), each of which is related to the topics *TV* and *radio*.

```
In [29]: text = [
                'TV programs are not interesting -- TV is annoying.',
                'Kids like TV'
                ,'We receive TV by radio waves'
                ,'It is interesting to listen to the radio'
                ,'On the waves, kids programs are rare.'
                ,'The kids listen to the radio; it is rare.'
                ]
         print(text)
         print(f'There are {len(text)} sentences.')
```

```
['TV programs are not interesting -- TV is annoying.', 'Kids like TV', 'We receive TV by radio
There are 6 sentences.
```

1.5.1 Constructing a Document-Term Matrix

Section **??**

1.5.2 Question 1:

5 points

Construct a document-term matrix from the preceding list of strings `text`. + Use an instance of the `CountVectorizer` class as in the preceding example. + You can apply the `fit_transform` method or apply the `fit` method, and then apply the `transform` method to the data `text`. + Convert the sparse array returned to a dense Numpy array. + Assign the final object obtained to the identifier `transformed_text`.

```
In [30]: ### GRADED

         from sklearn.feature_extraction.text import CountVectorizer
         count_vect = CountVectorizer()
         ### YOUR SOLUTION HERE
         vectorizer = CountVectorizer()
         X = vectorizer.fit_transform(text)
         transformed_text = X.toarray()
         ###
         ### YOUR CODE HERE
         ###
```

```
In [31]: ###
         ### AUTOGRADER TEST - DO NOT REMOVE
         ###
```

1.5.3 Using a DataFrame to Represent a Document-Term Matrix

Section **??**

1.5.4 Question 2:

5 points

Your task here is to represent the document-term matrix `transformed_text` as a Pandas `DataFrame`. In particular, adapt the construction preceding Question 01 into the body of a function `make_dtm_df`.

- Define a function signature is `make_dtm_df(corpus)` where `corpus` is a list of strings as can be used as an input to `CountVectorizeer.fit`.
- The value returned is a Pandas DataFrame.
- The rows of the DataFrame correspond to the entries of the input corpus (i.e., there are `len(corpus)` rows).
- The columns of the DataFrame correspond to the words extracted using `get_feature_names` once the `CountVectorizer` is `fit` to the input `corpus`.
- The entries of the DataFrame are the counts of each word as they occur in the entries of `corpus` (as in the document-term matrix).

```
>>> corpus = [
...     'This is the first document.',
...     'This document is the second document.',
...     'And this is the third one.',
...    'Is this the first document?' ]
>>> df = make_dtm_df(corpus)
>>> df
```

```
<tr style="text-align: right;">      <th></th>      <th>and</th>      <th>document</th>      <t
```

In [32]:
```
### GRADED

### YOUR SOLUTION HERE
def make_dtm_df(corpus):
    '''
    This function will take in a list of and return a document-term matrix as a DataF
    INPUT:
      corpus: list of sentences (strings)
    OUTPUT:
      returns DataFrame indexed by the feature names corresponding to columns of the
    '''
    vectorizer = CountVectorizer()
    X = vectorizer.fit_transform(corpus)
    transformed_text = X.toarray()
    return pd.DataFrame(X.todense(), columns=vectorizer.get_feature_names())
# corpus = ['This is the first document.',
#           'This document is the second document.',
#           'And this is the third one.',
#           'Is this the first document?' ]
# make_dtm_df(corpus)
###
### YOUR CODE HERE
###
```