# Rafael_Claro_Ito_CountVectorizer

March 18, 2020

## 1 Nome: Rafael Ito

Objetivo desse experimento é conhecer o CountVectorizer do scikit-learn, usando-o numa pequena amostra do dataset IMDB e codificando funções equivalente no Python.

Funções a serem implementadas:

1. vocab = build_vocab(corpus)
2. corpus_tok = tokenizer(corpus, vocab)
3. doc_term = feature(corpus_tok)

Enquanto está depurando o seu programa, utilize um corpus bem pequeno, com poucos exemplos e depois de depurado, rode ele nos 1000 exemplos do imdb_sample.

### 1.1 Usando o exemplo do scikit-learn:

[0]:
```
from sklearn.feature_extraction.text import CountVectorizer
import re
import torch
import numpy as np
```

[0]:
```
corpus = [
    'This is the first document.',
    'This document is the second document.',
    'And this is the third one.',
    'Is this the first document?',
]
```

[3]:
```
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
vocab = vectorizer.get_feature_names()
print(vocab)
```

```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
```

### 1.2 Mostrando o Document-term também denominado de “bag of words”

[4]:
```
print(X.toarray())
```

```
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
```

### 1.3 Minha implementação de um tokenizador simples usando o vocabulário já extraído pelo scikit-learn

Primeira versão: usando for simples

[5]:
```
list_word_based = []
list_token_based = []
for amostra in corpus:
    amostra = re.sub(r'\W',' ',amostra).strip().lower()
    list_words = amostra.split(' ')
    list_tokens = []
    for word in list_words:
        list_tokens.append(vocab.index(word))
    list_word_based.append(list_words)
    list_token_based.append(list_tokens)
list_word_based, list_token_based
```

[5]:
```
([['this', 'is', 'the', 'first', 'document'],
  ['this', 'document', 'is', 'the', 'second', 'document'],
  ['and', 'this', 'is', 'the', 'third', 'one'],
  ['is', 'this', 'the', 'first', 'document']],
 [[8, 3, 6, 2, 1], [8, 1, 3, 6, 5, 1], [0, 8, 3, 6, 7, 4], [3, 8, 6, 2, 1]])
```

Segunda versão: for com list comprehension

[6]:
```
list_word_based = []
list_token_based = []
for amostra in corpus:
    amostra = re.sub(r'\W',' ',amostra).strip().lower()
    list_words = amostra.split(' ')
    list_tokens = [vocab.index(word)   for word in list_words]
    list_word_based.append(list_words)
    list_token_based.append(list_tokens)
list_word_based, list_token_based
```

[6]:
```
([['this', 'is', 'the', 'first', 'document'],
  ['this', 'document', 'is', 'the', 'second', 'document'],
  ['and', 'this', 'is', 'the', 'third', 'one'],
  ['is', 'this', 'the', 'first', 'document']],
 [[8, 3, 6, 2, 1], [8, 1, 3, 6, 5, 1], [0, 8, 3, 6, 7, 4], [3, 8, 6, 2, 1]])
```

## 2 Download do dataset do IMDB_sample (apenas 1000 exemplos)

O dataset está sendo carregado dos datasets disponibilizados pelo curso fast.ai: https://course.fast.ai/datasets.html

O comando wget busca o arquivo imdb.tgz O comando tar descomprime o arquivo no diretório local

[7]:
```
!wget -nc http://files.fast.ai/data/examples/imdb_sample.tgz
!tar -xzf imdb_sample.tgz
```

```
--2020-03-19 01:42:25--  http://files.fast.ai/data/examples/imdb_sample.tgz
Resolving files.fast.ai (files.fast.ai)… 67.205.15.147
Connecting to files.fast.ai (files.fast.ai)|67.205.15.147|:80… connected.
HTTP request sent, awaiting response… 200 OK
Length: 571827 (558K) [application/x-gtar-compressed]
Saving to: ‘imdb_sample.tgz’

imdb_sample.tgz     100%[===================>] 558.42K  --.-KB/s    in 0.07s

2020-03-19 01:42:25 (8.24 MB/s) - ‘imdb_sample.tgz’ saved [571827/571827]
```

O diretório descomprimido tem um arquivo no formato csv:

[8]:
```
!ls imdb_sample
```

```
texts.csv
```

[0]:
```
import pandas as pd
```

[10]:
```
df = pd.read_csv('imdb_sample/texts.csv')
df.shape
```

[10]:
```
(1000, 3)
```

[11]:
```
df.head()
```

[11]:

| | label | text | is_valid |
|---|---|---|---|
| 0 | negative | Un-bleeping-believable! Meg Ryan doesn't even … | False |
| 1 | positive | This is a extremely well-made film. The acting… | False |
| 2 | negative | Every once in a long while a movie will come a… | False |
| 3 | positive | Name just says it all. I watched this movie wi… | False |
| 4 | negative | This movie succeeds at being one of the most u… | False |

[0]:
```

```

## 2.1 Pre-processing

[0]:
```
def pre_processing(corpus):
    corpus_pp = []
    for sentence in corpus:
        new_sentence = sentence.lower()                          # convert␣
↪to lowercase
        new_sentence = re.sub("[^\w]", " ",  new_sentence).split()  # match␣
↪word characters [a-zA-Z0-9_]
        corpus_pp.append(new_sentence)
    return corpus_pp
```

[13]:
```
corpus_pp = pre_processing(corpus)
corpus_pp
```

[13]:
```
[['this', 'is', 'the', 'first', 'document'],
 ['this', 'document', 'is', 'the', 'second', 'document'],
 ['and', 'this', 'is', 'the', 'third', 'one'],
 ['is', 'this', 'the', 'first', 'document']]
```

## 2.2 Vocabulary

### 2.2.1 function

[0]:
```
def build_vocab(corpus):
    vocab = " ".join(corpus)                        # join elements
    vocab = vocab.lower()                           # convert to lowercase
    vocab = re.sub("[^\w]", " ",  vocab).split()    # match word characters␣
↪[a-zA-Z0-9_]
    vocab = list(set(vocab))                        # remove duplicates
    vocab.sort()                                    # sort elements
    return vocab
```

### 2.2.2 testing

[15]:
```
vocab = build_vocab(corpus)
vocab
```

[15]:
```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
```

### 2.2.3 comparing with scikit-learn

[16]:
```
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
sk_vocab = vectorizer.get_feature_names()
print(sk_vocab)
```

```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
```

[17]:
```
vocab == sk_vocab
```

[17]:
```
True
```

## 2.3 Tokenizer

### 2.3.1 function

[0]:
```
def tokenizer(corpus, vocab):
    # first, a dictionary is created with the keys being the words in vocab,␣
 ↪and the values being the index
    dict = {vocab[i] : i for i in range(len(vocab))}
    corpus_pp = pre_processing(corpus)
    corpus_tok = []
    for idx, sentence in enumerate(corpus_pp):
        tokens = [dict[word] for word in sentence]
        corpus_tok.append(tokens)
    return corpus_tok
```

### 2.3.2 testing

[19]:
```
corpus_tok = tokenizer(corpus, vocab)
corpus_tok
```

[19]:
```
[[8, 3, 6, 2, 1], [8, 1, 3, 6, 5, 1], [0, 8, 3, 6, 7, 4], [3, 8, 6, 2, 1]]
```

## 2.4 Bag of Words

### 2.4.1 function

[0]:
```
def feature(corpus_tok):
    # create tensor with zeros of the correct size
    size = max([max(sublist) for sublist in corpus_tok]) + 1
    doc_term = torch.zeros(len(corpus_tok), size, dtype=torch.int64)
    for line, tok in enumerate(corpus_tok):
        for column in tok:
            doc_term[line][column] += 1
    return doc_term
```

### 2.4.2 testing

[21]:
```
doc_term = feature(corpus_tok)
doc_term
```

[21]:
```
tensor([[0, 1, 1, 1, 0, 0, 1, 0, 1],
        [0, 2, 0, 1, 0, 1, 1, 0, 1],
        [1, 0, 0, 1, 1, 0, 1, 1, 1],
        [0, 1, 1, 1, 0, 0, 1, 0, 1]])
```

#### 2.4.3 comparing with scikit-learn

[22]:
```
print(X.toarray())
```

```
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
```

[23]:
```
doc_term_np = doc_term.numpy()
print(doc_term_np)
```

```
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
```

[24]:
```
np.array_equal(doc_term_np, X.toarray())
```

[24]:
```
True
```

### 2.5 IMDb

#### 2.5.1 Filter data

[25]:
```
# getting only the 'text' column
imdb_corpus = df['text']
imdb_corpus.shape
```

[25]:
```
(1000,)
```

#### 2.5.2 Vocabulary

[26]:
```
# build_vocab
imdb_vocab = build_vocab(imdb_corpus)
len(imdb_vocab)
```

[26]:
```
18705
```

[27]:
```
# scikit-learn comparison
vectorizer = CountVectorizer()
Y = vectorizer.fit_transform(imdb_corpus)
```

```
sk_imdb_vocab = vectorizer.get_feature_names()
len(sk_imdb_vocab)
```

[27]: 18668

#### 2.5.3 Tokenizer

[28]:
```
imdb_corpus_tok = tokenizer(imdb_corpus, imdb_vocab)
len(imdb_corpus_tok)
```

[28]: 1000

#### 2.5.4 Bag of Words

[29]:
```
imdb_doc_term = feature(imdb_corpus_tok)
imdb_doc_term.shape
```

[29]: torch.Size([1000, 18705])

[30]:
```
# scikit-learn comparison
Y.toarray().shape
```

[30]: (1000, 18668)

#### 2.5.5 Comments:

O tamanho do vocabulário com a implementação do scikit-learn foi ligeiramente menor do que a minha implementação (scikit-learn: 18668, minha implementação: 18705). Isso ocorre devido a diferença de filtragem inicial. No meu caso, apenas troquei os caracteres para minúsculo e depois selecionei palavras que começam com os caracteres [a-zA-Z0-9_].