A friendly guide to NLP: Bag-of-Words with Python example

Eugenia Anello — Published On August 19, 2021 and Last Modified On August 2nd, 2022

Advanced NLP Project Python Text Unstructured Data

This article was published as a part of the Data Science Blogathon

This is the second post of the NLP tutorial series. This guide will let you understand step by step how to implement Bag-Of-Words and compare the results obtained with the already implemented Scikit-learn's CountVectorizer.

In the previous post of the series, I showed how to deal with text pre-processing, which is the first phase before applying any classification model on text data. But the cleaned text isn't enough to be passed directly to the classification model. The features need to be numeric, not strings. There are many state-of-art approaches to extract features from the text data.

The most simple and known method is the Bag-Of-Words representation. It's an algorithm that transforms the text into fixed-length vectors. This is possible by counting the number of times the word is present in a document. The word occurrences allow to compare different documents and evaluate their similarities for applications, such as search, document classification, and

The reason for its name, “Bag-Of-Words”, is due to the fact that it represents the sentence as a bag of terms. It doesn’t take into account the order and the structure of the words, but it only checks if the words appear in the document.

Table of Content:

1. A Quick Example

Let’s look at an easy example to understand the concepts previously explained. We could be interested in analyzing the reviews about Game of Thrones:

Review 1: Game of Thrones is an amazing tv series!

Review 2: Game of Thrones is the best tv series!

Review 3: Game of Thrones is so great

In the table, I show all the calculations to obtain the Bag-Of-Words approach:

	amazing	an	best	game	great	is	of	series	so	the	thrones	tv
0	1	1	0	1	0	1	1	1	0	0	1	1
1	0	0	1	1	0	1	1	1	0	1	1	1
2	0	0	0	1	1	1	1	0	1	0	1	0

Each row corresponds to a different review, while the rows are the unique words, contained in the three documents.

2. Implementation with Python

Let’s import the libraries and define the variables, that contain the reviews:

```
import pandas as pd
import numpy as np
import collections
```

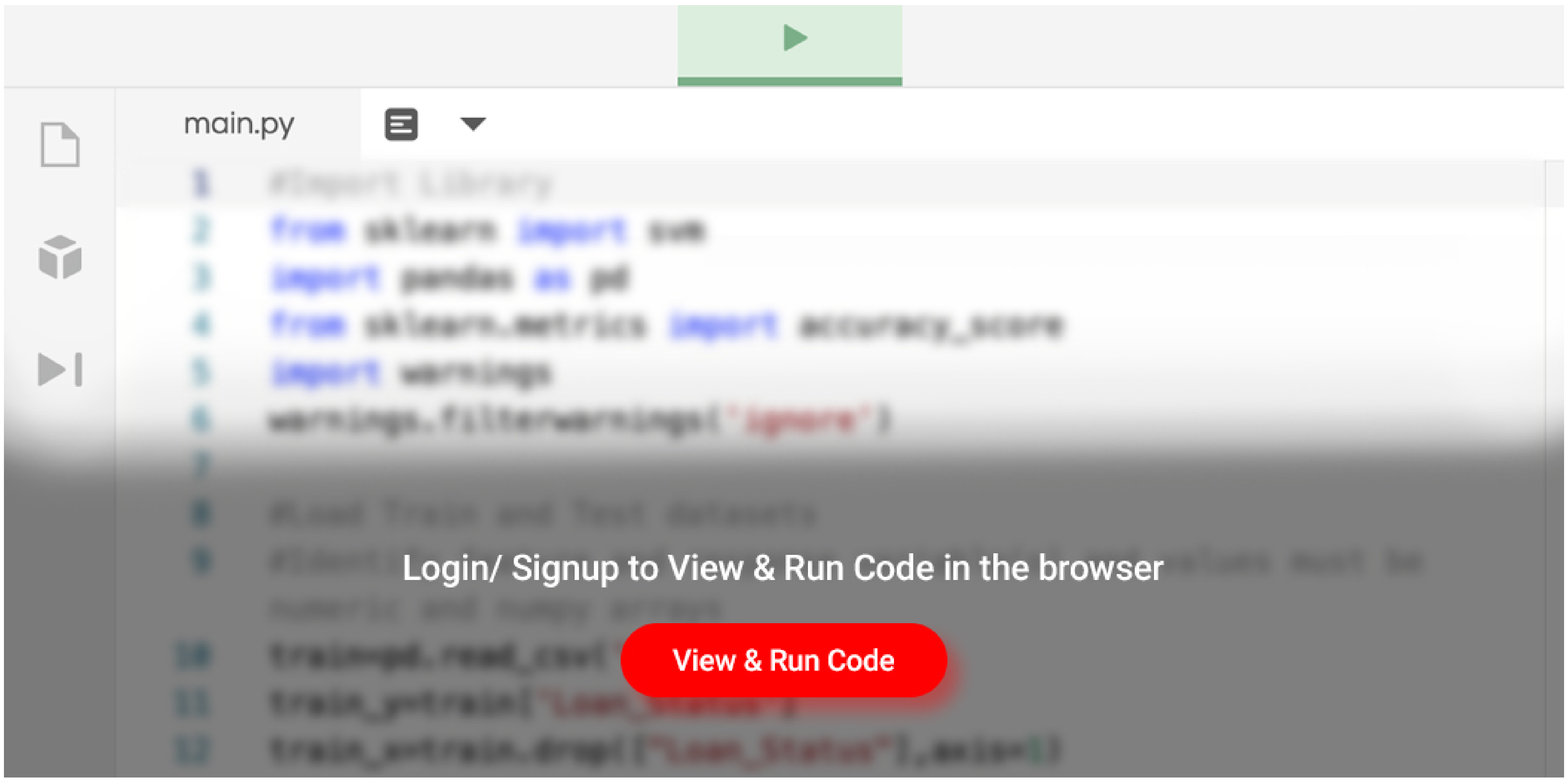
```
doc1 = 'Game of Thrones is an amazing tv series!'
doc2 = 'Game of Thrones is the best tv series!'
doc3 = 'Game of Thrones is so great'
```

We need to remove punctuations, one of the steps I showed in the previous post about the text pre-processing. We also transform the string into a list composed of words.

```
l_doc1 = re.sub(r"[^a-zA-Z0-9]", " ", doc1.lower()).split()
l_doc2 = re.sub(r"[^a-zA-Z0-9]", " ", doc2.lower()).split()
l_doc3 = re.sub(r"[^a-zA-Z0-9]", " ", doc3.lower()).split()
```

After we achieve the Vocabulary, or wordset, which is composed of the unique words found in the three reviews.

Python Code:



```
#Import Library
from sklearn import svm
import pandas as pd
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')

#Load Train and Test datasets
#Identify [illegible] values must be
numeric and numpy arrays
train=pd.read_csv([illegible]
train_y=train['Loan_Status']
train_x=train.drop(["Loan_Status"],axis=1)
```

We can finally define the function to extract the features in each document. Let's explain step by step:

- we define a dictionary with the specified keys, which corresponds to the words of the Vocabulary, and the specified value is 0.
- we iterate over the words contained only in the document and we assign to each word its frequency within the review.

```
def calculateBOW(wordset,l_doc):
  tf_diz = dict.fromkeys(wordset,0)
  for word in l_doc:
      tf_diz[word]=l_doc.count(word)
  return tf_diz
```

We can finally obtain the Bag-of-Words representations for the reviews. In the end, we obtain a data frame, where each row corresponds to the extracted features of each document.

```
bow1 = calculateBOW(wordset,l_doc1)
bow2 = calculateBOW(wordset,l_doc2)
bow3 = calculateBOW(wordset,l_doc3)
df_bow = pd.DataFrame([bow1,bow2,bow3])
df_bow.head()
```

	amazing	an	best	game	great	is	of	series	so	the	thrones	tv
0	1	1	0	1	0	1	1	1	0	0	1	1
1	0	0	1	1	0	1	1	1	0	1	1	1
2	0	0	0	1	1	1	1	0	1	0	1	0

Results obtained with the implementation from scratch

Didn't it seem one of the boring exercises given during a programming course? It's like that but applied in a real dataset. Great! We obtained what we wanted.

In the previous section, we implemented the representation. Now, we want to compare the results obtaining, applying the Scikit-learn's CountVectorizer. First, we instantiate a CountVectorizer object and later we learn the term frequency of each word within the document. In the end, we return the document-term matrix.

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
print(vectorizer.get_feature_names())
```

```
['amazing',
 'an',
 'best',
 'game',
 'great',
 'is',
 'of',
 'series',
 'so',
 'the',
 'thrones',
 'tv']
```

CountVectorizer provides the get_features_name method, which contains the uniques words of the vocabulary, taken into account later to create the desired document-term matrix X. To have an easier visualization, we transform it into a pandas data frame.

```
X = vectorizer.fit_transform([doc1,doc2,doc3])
df_bow_sklearn = pd.DataFrame(X.toarray(),columns=vectorizer.get_feature_names())
df_bow_sklearn.head()
```

	amazing	an	best	game	great	is	of	series	so	the	thrones	tv
0	1	1	0	1	0	1	1	1	0	0	1	1
1	0	0	1	1	0	1	1	1	0	1	1	1
2	0	0	0	1	1	1	1	0	1	0	1	0

Results obtained with Sklearn

We compare it with the output obtained before.

	amazing	an	best	game	great	is	of	series	so	the	thrones	tv
0	1	1	0	1	0	1	1	1	0	0	1	1
1	0	0	1	1	0	1	1	1	0	1	1	1
2	0	0	0	1	1	1	1	0	1	0	1	0

Results obtained with the implementation from scratch

So, the results match and the task is solved!

Until now I kept the stop words to keep the tutorial simple. But there is also the possibility to remove the stop words without adding any line of code in Sklearn. We only need to add an argument in the CountVectorizer function:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(stop_words='english')
X = vectorizer.fit_transform([doc1,doc2,doc3])
df_bow_sklearn = pd.DataFrame(X.toarray(),columns=vectorizer.get_feature_names())
df_bow_sklearn.head()
```

	amazing	best	game	great	series	thrones	tv
0	1	0	1	0	1	1	1
1	0	1	1	0	1	1	1
2	0	0	1	1	0	1	0

We can also do another experiment. One possibility is to take into account the bigrams, instead of the unigrams. For example, the two words, "tv series", match very well together and are repeated in every review:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(stop_words='english',ngram_range=(2,2))
X = vectorizer.fit_transform([doc1,doc2,doc3])
df_bow_sklearn = pd.DataFrame(X.toarray(),columns=vectorizer.get_feature_names())
df_bow_sklearn.head()
```

	amazing tv	best tv	game thrones	thrones amazing	thrones best	thrones great	tv series
0	1	0	1	1	0	0	1
1	0	1	1	0	1	0	1
2	0	0	1	0	0	1	0

Aren't the combination of words interesting? It seems to make sense for "tv series", while "game thrones" bigram loses the meaning and the word "of" since it's a stop word. So, in some context, remove all the stop words isn't always convenient.

Final thoughts:

That's it! Bag-Of-Words is quite simple to implement as you can see. Of course, we only considered only unigram (single words) or bigrams(couples of words), but also trigrams can be taken into account to extract features. Stop words can be removed too as we saw, but there are still some disadvantages. The order and the meaning of the words are lost using this method. For this reason, other approaches are preferred to extract features from the text, like TF-IDF, which I will talk about in the next post of the series. Thanks for reading. Have a nice day!

Author Bio:

Eugenia Anello has a statistics background and is pursuing a master's degree in Data Science at the University of Padova. She enjoys writing data science posts on Medium and on other platforms. Her purpose is to share the knowledge acquired in a simple and understandable way.

You can follow her on Linkedin and Medium.

The media shown in this article are not owned by Analytics Vidhya and are used at the Author's discretion.

bag of words   blogathon

Analytics Vidhya
DATA SCIENCE
BLOGATHON
8th October to 31st October 2022
Assured Rs 2000* ($26) for every published article!
A chance to win an iPad* T&C apply
Register Now

About the Author

[Eugenia Anello](#)

Our Top Authors

view more

Download

Analytics Vidhya App for the Latest blog/Article

Previous Post

[A friendly guide to NLP: Text pre-processing with Python Example](#)

Next Post

[Complete guide on How to learn Scikit-Learn for Data Science](#)

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

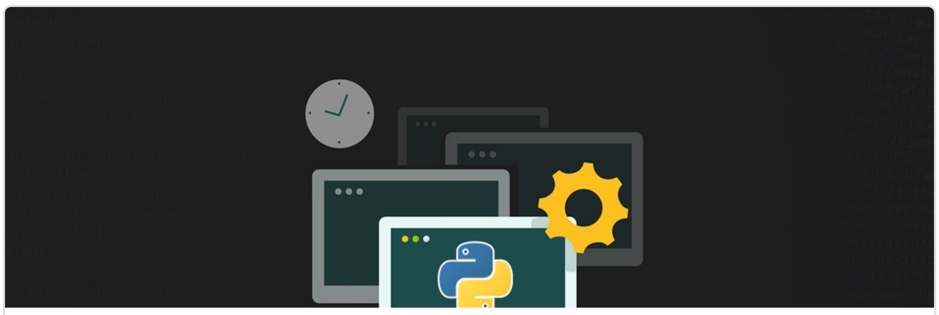
Name*	Email*
Website	

☑ Notify me of follow-up comments by email.

☑ Notify me of new posts by email.

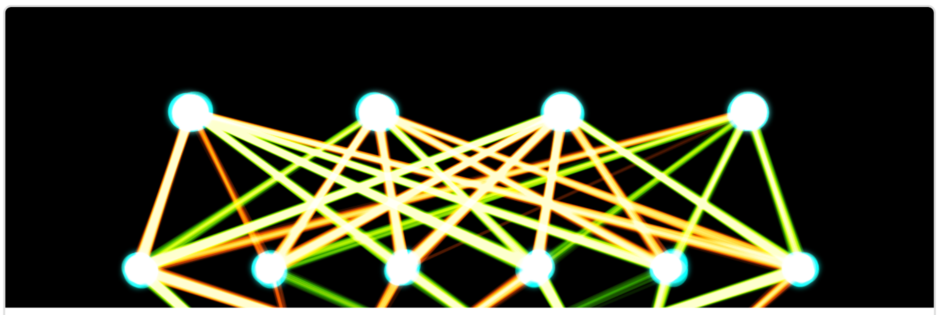
Submit

Top Resources



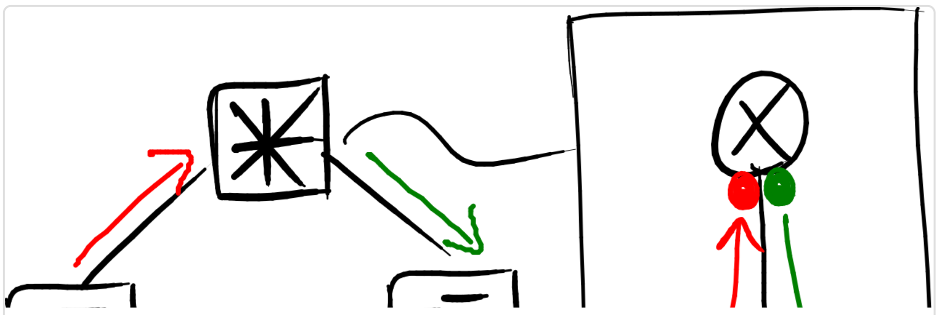
Python Tutorial: Working with CSV file for Data Science

Harika Bonthu - AUG 21, 2021



Boost Model Accuracy of Imbalanced COVID-19 Mortality Prediction Using GAN-based..

Bala Gangadhar Thilak Adiboina - OCT 07, 2020



Introductory guide on Linear Programming for (aspiring) data scientists

avcontentteam - FEB 28, 2017

Understanding Random Forest

Sruthi E R - JUN 17, 2021