Published in Towards Data Science

Pratyaksh Jain Follow
May 24, 2021 · 7 min read · Listen

Basics of CountVectorizer

Learn everything you need to know about CountVectorizer.

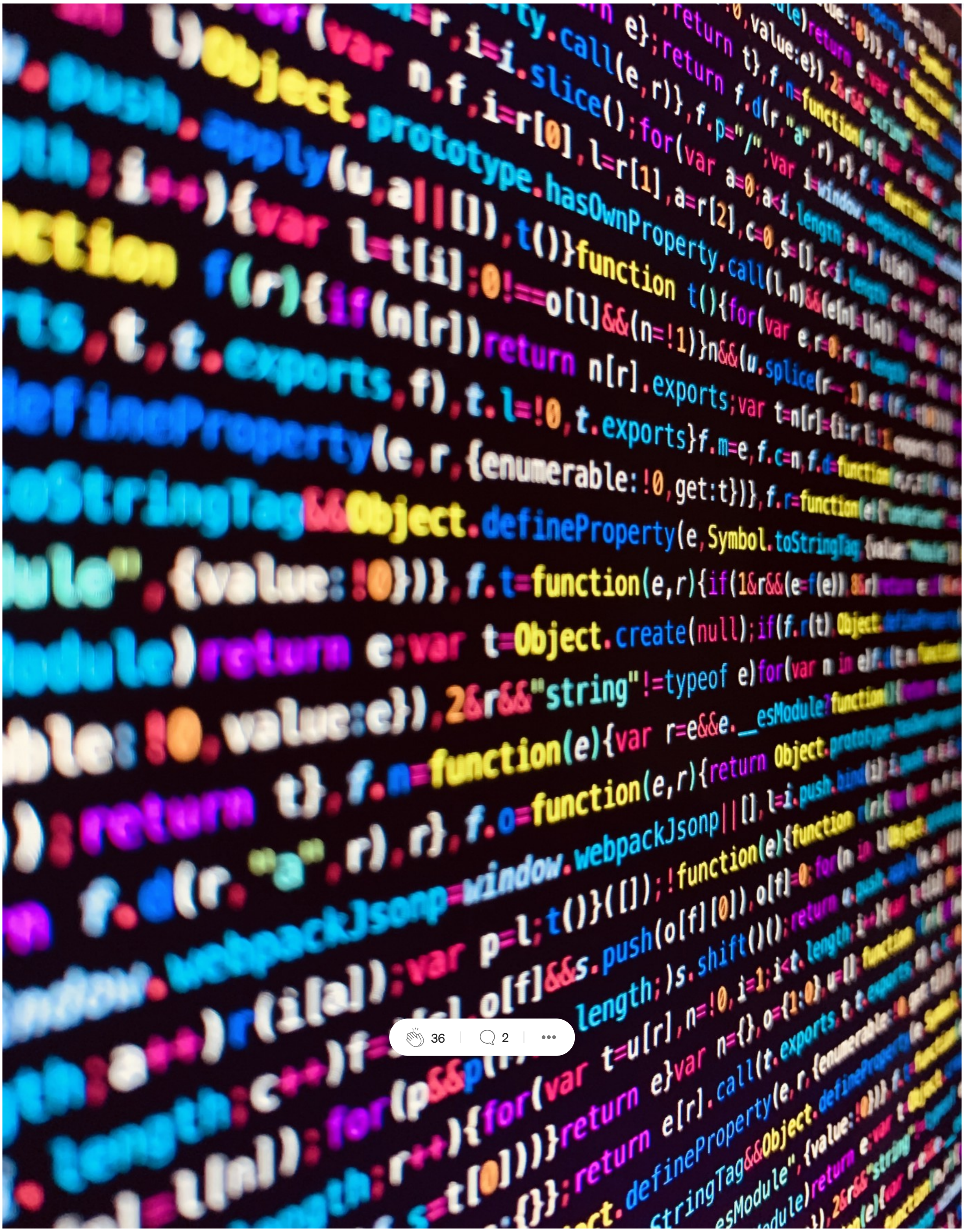
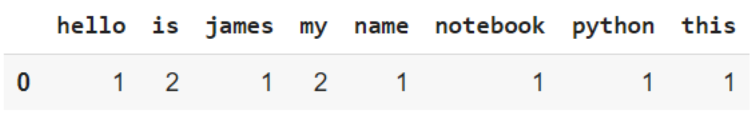


Photo by Ali Shah Lakhani on Unsplash

Machines cannot understand characters and words. So when dealing with text data we need to represent it in numbers to be understood by the machine. Countvectorizer is a method to convert text to numerical data. To show you how it works let’s take an example:

The text is transformed to a sparse matrix as shown below.



	hello	is	james	my	name	notebook	python	this
0	1	2	1	2	1	1	1	1

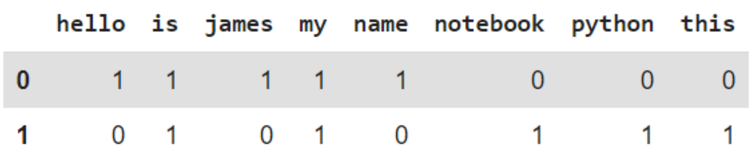
We have 8 unique words in the text and hence 8 different columns each representing a unique word in the matrix. The row represents the word count. Since the words ‘is’ and ‘my’ were repeated twice we have the count for those particular words as 2 and 1 for the rest.

Countvectorizer makes it easy for text data to be used directly in machine learning and deep learning models such as text classification.

Let’s take another example, but this time with more than 1 input:

```
text = [‘Hello my name is james' , ’this is my python notebook’]
```

I have 2 text inputs, what happens is that each input is preprocessed, tokenized, and represented as a sparse matrix. By default, Countvectorizer converts the text to lowercase and uses word-level tokenization.

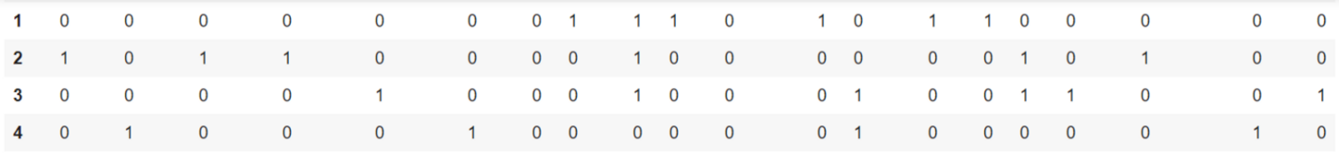


	hello	is	james	my	name	notebook	python	this
0	1	1	1	1	1	0	0	0
1	0	1	0	1	0	1	1	1

Now that we have looked at a few examples lets actually code!

We’ll first start by importing the necessary libraries. We’ll use the pandas library to visualize the matrix and the sklearn.feature_extraction.text which is a sklearn library to perform vectorization.

```
import pandas as pd

from sklearn.feature_extraction.text import CountVectorizer

text = [‘Hello my name is james’,

‘james this is my python notebook’,

‘james trying to create a big dataset’,

‘james of words to try differnt’,

‘features of count vectorizer’]

coun_vect = CountVectorizer()

count_matrix = coun_vect.fit_transform(text)

count_array = count_matrix.toarray()

df = pd.DataFrame(data=count_array,columns = coun_vect.get_feature_names())

print(df)
```



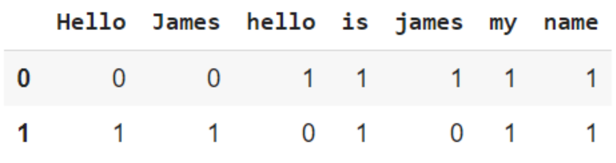
1	0	0	0	0	0	0	0	1	1	1	0	1	0	1	1	0	0	0	0	0
2	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0
3	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	1	0	0	1
4	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0

Parameters

- **Lowercase**

Convert all characters to lowercase before tokenizing. Default is set to true and takes boolean value.

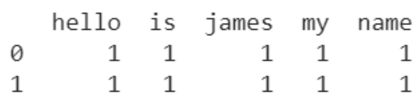
```
text = ['hello my name is james',

'Hello my name is James']

coun_vect = CountVectorizer(lowercase=False)

count_matrix = coun_vect.fit_transform(text)

count_array = count_matrix.toarray()

df = pd.DataFrame(data=count_array,columns = coun_vect.get_feature_names())

print(df)
```



	Hello	James	hello	is	james	my	name
0	0	0	1	1	1	1	1
1	1	1	0	1	0	1	1

Now lets try without using 'lowercase = False'

```
text = ['hello my name is james',

'Hello my name is James']

coun_vect = CountVectorizer()

count_matrix = coun_vect.fit_transform(text)

count_array = count_matrix.toarray()

df = pd.DataFrame(data=count_array,columns = coun_vect.get_feature_names())

print(df)
```



```
   hello  is  james  my  name
0      1   1      1   1     1
1      1   1      1   1     1
```

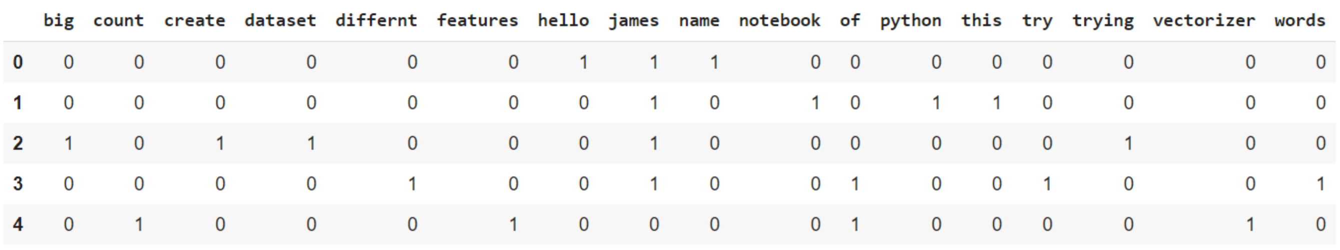
- **Stop_words**

Stopwords are the words in any language which does not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence. There are 3 ways of dealing with stopwords:

1. Custom stop words list

```
 'james this is my python notebook',

 'james trying to create a big dataset',

 'james of words to try differnt',

 'features of count vectorizer']

coun_vect = CountVectorizer(stop_words= ['is','to','my'])

count_matrix = coun_vect.fit_transform(text)

count_array = count_matrix.toarray()

df = pd.DataFrame(data=count_array,columns = coun_vect.get_feature_names())

print(df)
```

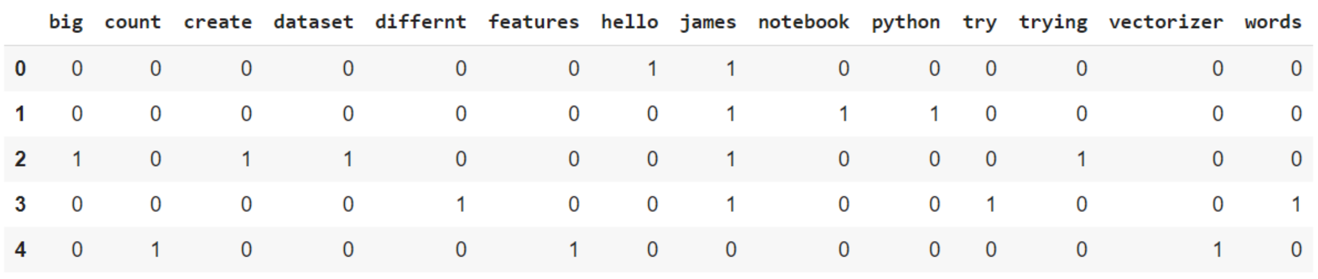
Sparse matrix after removing the words is , to and my:



	big	count	create	dataset	differnt	features	hello	james	name	notebook	of	python	this	try	trying	vectorizer	words
0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	0	1	0	1	1	0	0	0	0
2	1	0	1	1	0	0	0	1	0	0	0	0	0	0	1	0	0
3	0	0	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1
4	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0

2. sklearn built in stop words list

```
text = ['Hello my name is james',

 'james this is my python notebook',

 'james trying to create a big dataset',

 'james of words to try differnt',

 'features of count vectorizer']

coun_vect = CountVectorizer(stop_words='english')

count_matrix = coun_vect.fit_transform(text)

count_array = count_matrix.toarray()

df = pd.DataFrame(data=count_array,columns = coun_vect.get_feature_names())

print(df)
```



	big	count	create	dataset	differnt	features	hello	james	notebook	python	try	trying	vectorizer	words
0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1	1	0	0	0	0
2	1	0	1	1	0	0	0	1	0	0	0	1	0	0
3	0	0	0	0	1	0	0	1	0	0	1	0	0	1
4	0	1	0	0	0	1	0	0	0	0	0	0	1	0

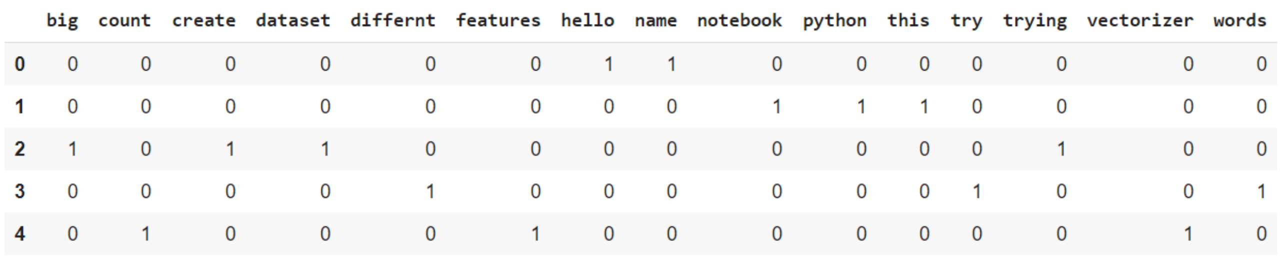
3. Using max_df and min_df (covered later)

Max_df stands for maximum document frequency. Similar to min_df, we can ignore words which occur frequently. These words could be like the word 'the' that occur in every document and does not provide and valuable information to our text classification or any other machine learning model and can be safely ignored. Max_df looks at how many documents contain the word and if it exceeds the max_df threshold then it is eliminated from the sparse matrix. This parameter can again 2 types of values, percentage and absolute.

Using absolute values:

```
text = ['Hello my name is james',

'james this is my python notebook',

'james trying to create a big dataset',

'james of words to try differnt',

'features of count vectorizer']

coun_vect = CountVectorizer(max_df=1)

count_matrix = coun_vect.fit_transform(text)

count_array = count_matrix.toarray()

df = pd.DataFrame(data=count_array,columns = coun_vect.get_feature_names())

print(df)
```

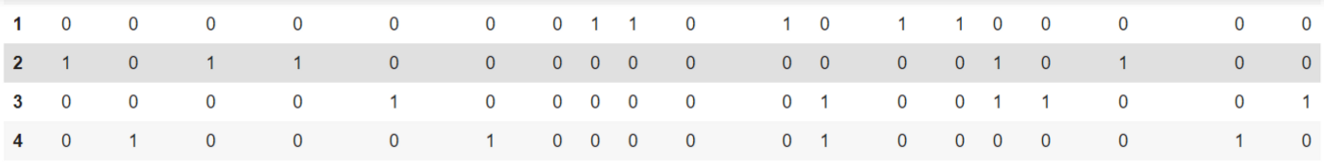
The words 'is', 'to', 'james', 'my' and 'of' have been removed from the sparse matrix as they occur in more than 1 document.



	big	count	create	dataset	differnt	features	hello	name	notebook	python	this	try	trying	vectorizer	words
0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0
2	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0
3	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1
4	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0

Using percentage:

```
text = ['Hello my name is james',

'james this is my python notebook',

'james trying to create a big dataset',

'james of words to try differnt',

'features of count vectorizer']

coun_vect = CountVectorizer(max_df=0.75)

count_matrix = coun_vect.fit_transform(text)

count_array = count_matrix.toarray()

df = pd.DataFrame(data=count_array,columns = coun_vect.get_feature_names())

print(df)
```



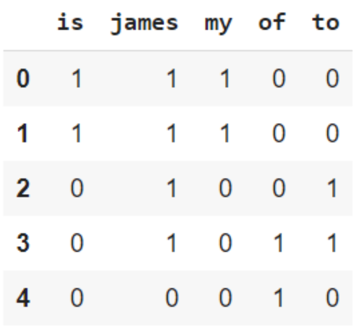
1	0	0	0	0	0	0	0	1	1	0	1	0	1	1	0	0	0	0	0
2	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
3	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	1	0	0	1
4	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0

Min_df:

Min_df stands for minimum document frequency, as opposed to term frequency which counts the number of times the word has occurred in the entire dataset, document frequency counts the number of documents in the dataset (aka rows or entries) that have the particular word. When building the vocabulary Min_df ignores terms that have a document frequency strictly lower than the given threshold. For example in your dataset you may have names that appear in only 1 or 2 documents, now these could be ignored as they do not provide enough information on the entire dataset as a whole but only a couple of particular documents. min_df can take absolute values(1,2,3..) or a value representing a percentage of documents(0.50, ignore words appearing in 50% of documents)

Using absolute values:

```
text = ['Hello my name is james',

'james this is my python notebook',

'james trying to create a big dataset',

'james of words to try differnt',

'features of count vectorizer']

coun_vect = CountVectorizer(min_df=2)

count_matrix = coun_vect.fit_transform(text)

count_array = count_matrix.toarray()

df = pd.DataFrame(data=count_array,columns = coun_vect.get_feature_names())

print(df)
```



	is	james	my	of	to
0	1	1	1	0	0
1	1	1	1	0	0
2	0	1	0	0	1
3	0	1	0	1	1
4	0	0	0	1	0

- **max_features**

The CountVectorizer will select the words/features/terms which occur the most frequently. It takes absolute values so if you set the 'max_features = 3', it will select the 3 most common words in the data.

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

coun_vect = CountVectorizer(max_features=3)

count_matrix = coun_vect.fit_transform(text)
```

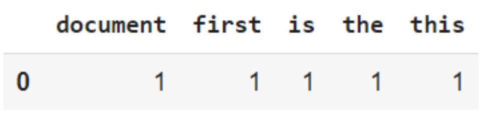
```
df
```

- **Binary**

By setting 'binary = True', the CountVectorizer no more takes into consideration the frequency of the term/word. If it occurs it's set to 1 otherwise 0. By default, binary is set to False. This is usually used when the count of the term/word does not provide useful information to the machine learning model.

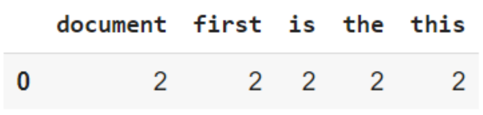
```
text = ['This is the first document. Is this the first document?' ]

coun_vect = CountVectorizer(binary=True)

count_matrix = coun_vect.fit_transform(text)

count_array = count_matrix.toarray()

df = pd.DataFrame(data=count_array,columns = coun_vect.get_feature_names())

print(df)
```

Even though all the words occur twice in the above input our sparse matrix just represents it with 1



	document	first	is	the	this
0	1	1	1	1	1

Now let's see if we used the default value:

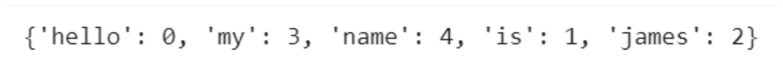


	document	first	is	the	this
0	2	2	2	2	2

- **Vocabulary**

They are the collection of words in the sparse matrix.

```
text = ['hello my name is james',

'Hello my name is James']

coun_vect = CountVectorizer()

count_matrix = coun_vect.fit_transform(text)

print(coun_vect.vocabulary_)
```



```
{'hello': 0, 'my': 3, 'name': 4, 'is': 1, 'james': 2}
```

The numbers do not represent the count of the words but the position of the words in the matrix

If you just want the vocabulary without the position of the word in the sparse matrix, you can use the method 'get_feature_names()'. If you notice this is the same method we use while creating our database and setting our columns.

```
'james this is my python notebook',

'james trying to create a big dataset']

coun_vect = CountVectorizer()

count_matrix = coun_vect.fit_transform(text)

print( coun_vect.get_feature_names())
```

```
['big', 'create', 'dataset', 'hello', 'is', 'james', 'my', 'name', 'notebook', 'python', 'this', 'to', 'trying']
```

CountVectorizer is just one of the methods to deal with textual data. Td-idf is a better method to vectorize data. I'd recommend you check out the official document of sklearn for more information.

Hope this helps :)