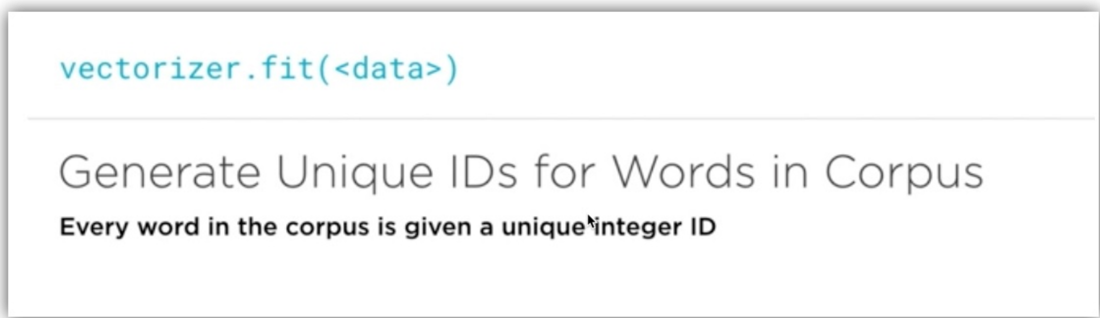
TextFeatureExtraction

November 2, 2021

[25]:
```
from IPython.display import Image
import pandas as pd
```

[4]:
```
'''
Every vectorizer implements a fit method to which we pass in a corpus of␣
 ↪documents.
This method generates unique IDs for all words in the corpus.

This is where the vectorizer learns the vocabulary of our input data set.
'''

Image("/Users/subhasish/Documents/APPLE/SUBHASISH/Development/GIT/Interstellar/
 ↪SB-AI-DEV/ML/SB/Classification/Janani Ravi/Building Machine Learning Models␣
 ↪in Python with scikit-learn/Image/2021-11-02_19-14-33.jpg")
```

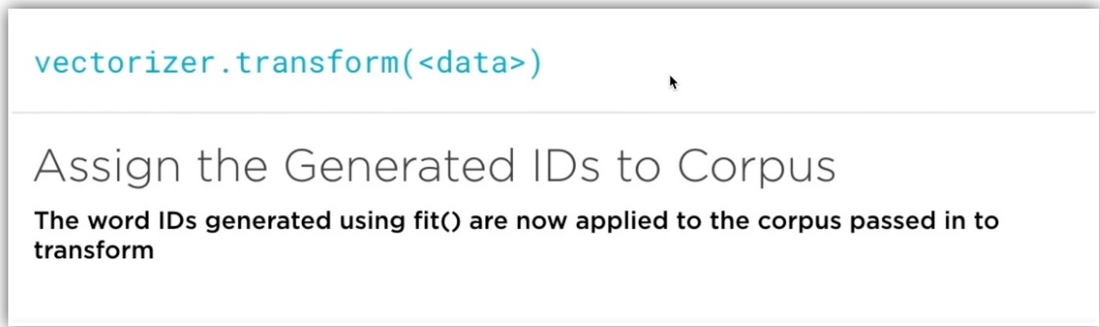
[4]:



[5]:
```
'''
A vectorizer also implements the transform method to which you pass in some␣
 ↪input data.
This is used to assign the generated unique IDs to words in the corpus that we␣
 ↪just passed in as an input argument.


Using the fit method and the transform method separately makes sense if you␣
 ↪want to generate unique
IDS using one corpus, and assign those IDs to another corpus.
```

```
'''
Image('/Users/subhasish/Documents/APPLE/SUBHASISH/Development/GIT/Interstellar/
↪SB-AI-DEV/ML/SB/Classification/Janani Ravi/Building Machine Learning Models␣
↪in Python with scikit-learn/Image/2021-11-02_19-16-54.jpg')
```

[5]:



[6]:
```
'''
If you're working on just one data set, you'll typically use fit and transform␣
↪to generate unique IDs
and assign it to words in that corpus.
'''

Image('/Users/subhasish/Documents/APPLE/SUBHASISH/Development/GIT/Interstellar/
↪SB-AI-DEV/ML/SB/Classification/Janani Ravi/Building Machine Learning Models␣
↪in Python with scikit-learn/Image/2021-11-02_19-19-02.jpg')
```

[6]:

[7]:
```
'''
The first is the CountVectorizer. This is a basic frequency-based␣
↪representation of words in documents.
'''
from sklearn.feature_extraction.text import CountVectorizer
```

[8]:
```
'''
This corpus has four documents, which we've specified in the form of an array. I
'''
corpus = ['This is the first document.',
          'This is the second document.',
          'Third document. Document number three',
          'Number four. To repeat, number four']
```

[9]:
```
'''
 Initialize the CountVectorizer class and generate a bag of words in numeric␣
 ↪form.
 Simply set up the CountVectorizer, and then call fit_transform on our corpus␣
 ↪of data.
 The bag of words is a sparse 4 by 12 matrix, four documents and a total␣
 ↪vocabulary of 12 words.
'''
vectorizer=CountVectorizer()
bag_of_words=vectorizer.fit_transform(corpus)
bag_of_words
```

[9]:
```
<4x12 sparse matrix of type '<class 'numpy.int64'>'
        with 18 stored elements in Compressed Sparse Row format>
```

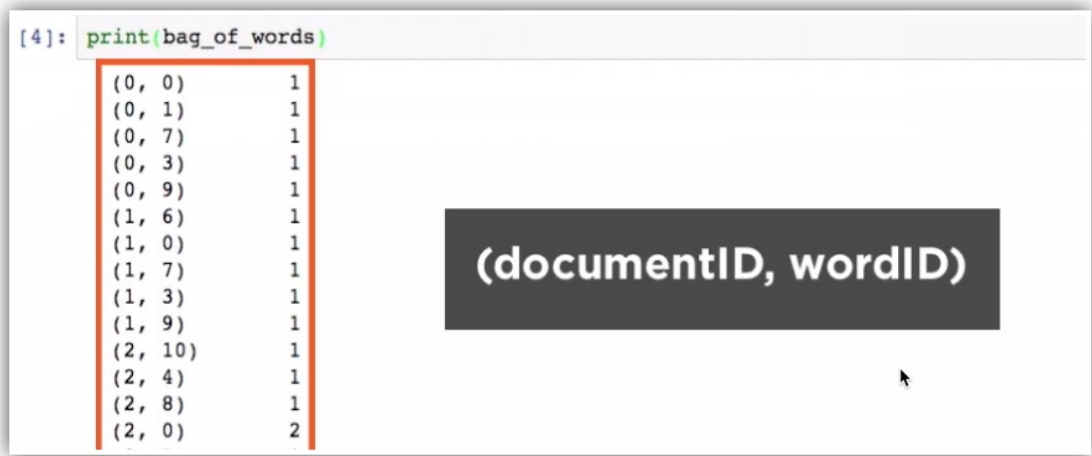
[10]:
```
'''
Print out the contents of this bag of words.
Notice that every word is identified by the document in which that word␣
 ↪occurred.

'''
print(bag_of_words)
```

```
  (0, 9)        1
  (0, 3)        1
  (0, 7)        1
  (0, 1)        1
  (0, 0)        1
  (1, 9)        1
  (1, 3)        1
  (1, 7)        1
  (1, 0)        1
  (1, 6)        1
  (2, 0)        2
  (2, 8)        1
  (2, 4)        1
  (2, 10)       1
  (3, 4)        2
  (3, 2)        2
```

```
(3, 11)       1
(3, 5)        1
```

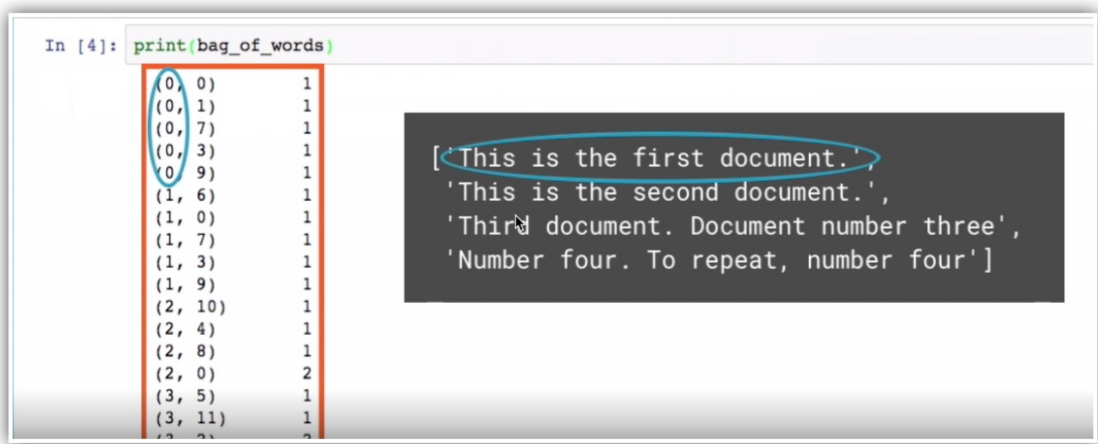
[11]:
```
Image('/Users/subhasish/Documents/APPLE/SUBHASISH/Development/GIT/Interstellar/
↪SB-AI-DEV/ML/SB/Classification/Janani Ravi/Building Machine Learning Models␣
↪in Python with scikit-learn/Image/2021-11-02_19-34-45.jpg')
```

[11]:



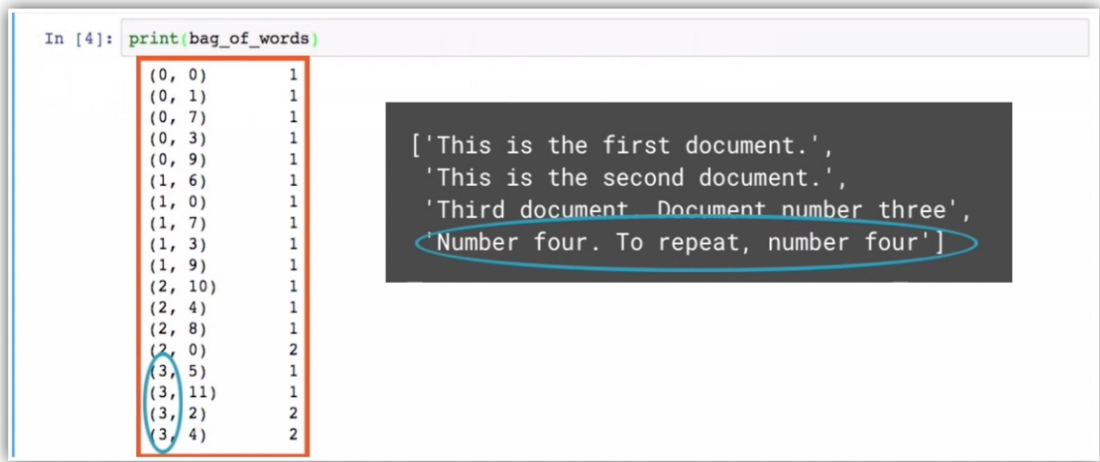
[12]:
```
'''
There are 4 documents in our input corpus, which is why every document is given␣
↪a unique
ID from zero all the way to three.
'''
Image('/Users/subhasish/Documents/APPLE/SUBHASISH/Development/GIT/Interstellar/
↪SB-AI-DEV/ML/SB/Classification/Janani Ravi/Building Machine Learning Models␣
↪in Python with scikit-learn/Image/2021-11-02_19-35-13.jpg')
```

[12]:



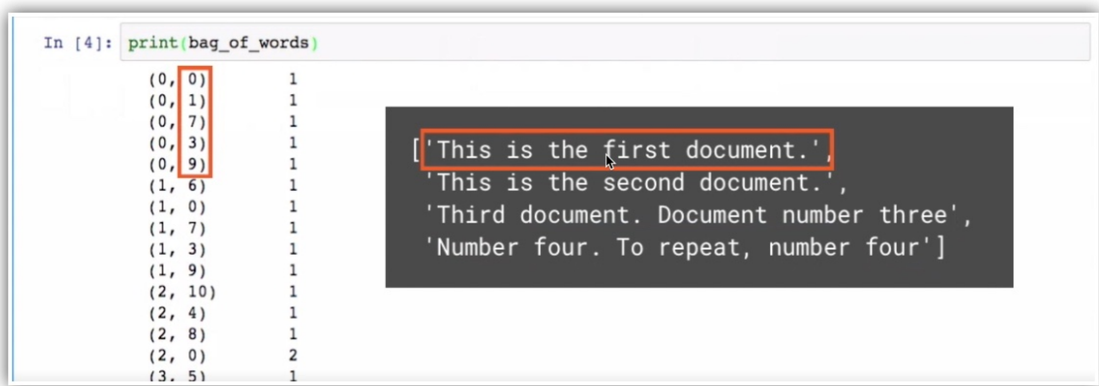
[13]:
```python
Image('/Users/subhasish/Documents/APPLE/SUBHASISH/Development/GIT/Interstellar/
↪SB-AI-DEV/ML/SB/Classification/Janani Ravi/Building Machine Learning Models␣
↪in Python with scikit-learn/Image/2021-11-02_19-36-27.jpg')
```

[13]:



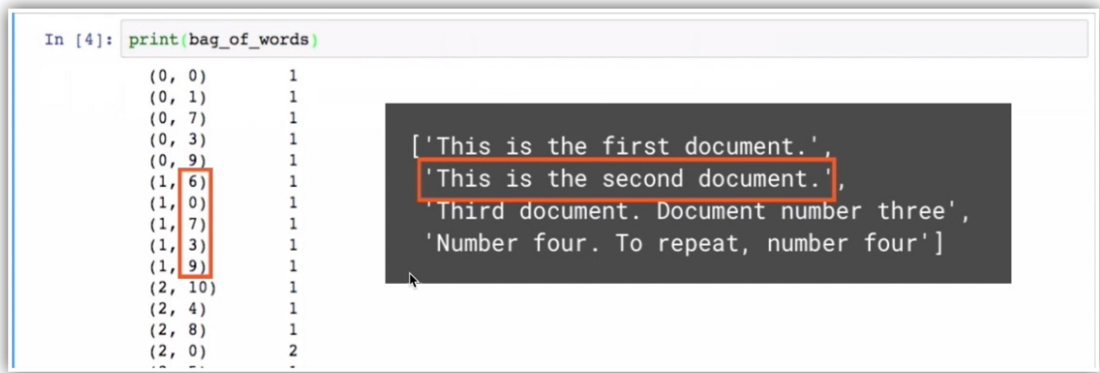
[14]:
```python
'''
Every word in our document corpus also has a unique individual ID.
Here are the five words in document one, the five words in document two
'''
Image('/Users/subhasish/Documents/APPLE/SUBHASISH/Development/GIT/Interstellar/
↪SB-AI-DEV/ML/SB/Classification/Janani Ravi/Building Machine Learning Models␣
↪in Python with scikit-learn/Image/2021-11-02_19-39-51.jpg')
```

[14]:



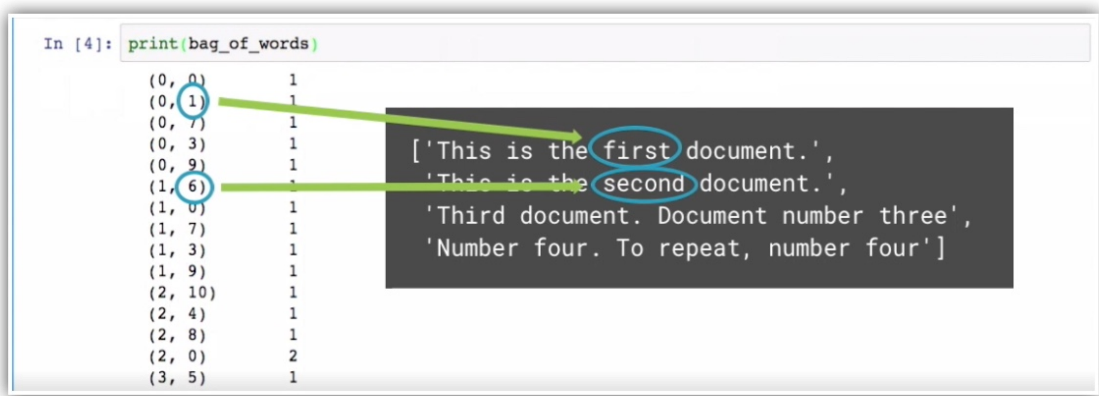
[15]:
```python
Image('/Users/subhasish/Documents/APPLE/SUBHASISH/Development/GIT/Interstellar/
↪SB-AI-DEV/ML/SB/Classification/Janani Ravi/Building Machine Learning Models␣
↪in Python with scikit-learn/Image/2021-11-02_19-40-24.jpg')
```

[15]:



[16]:
```python
'''
If you look closely at document one and two, you'll see that the only word that is different
is the word "first" and "second".
Those are represented by different integers. The other words in these two documents are the same.
Their integer representations are also the same.
'''
Image('/Users/subhasish/Documents/APPLE/SUBHASISH/Development/GIT/Interstellar/SB-AI-DEV/ML/SB/Classification/Janani Ravi/Building Machine Learning Models in Python with scikit-learn/Image/2021-11-02_19-41-04.jpg')
```

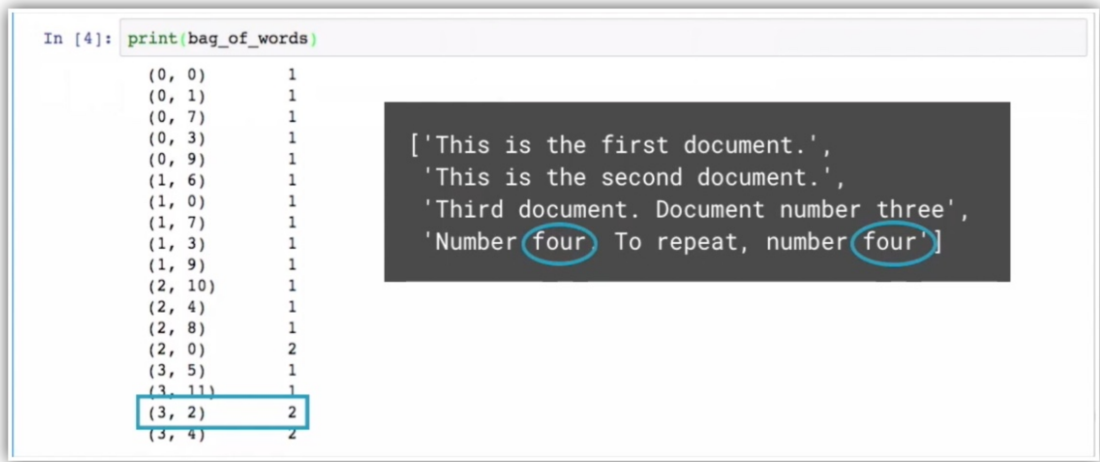
[16]:



[17]:
```python
'''
 Notice that word frequencies are captured in this bag of words representation.
 The word "four" appears twice in the last document. It's frequency is two.
'''
```

```
Image('/Users/subhasish/Documents/APPLE/SUBHASISH/Development/GIT/Interstellar/
↪SB-AI-DEV/ML/SB/Classification/Janani Ravi/Building Machine Learning Models␣
↪in Python with scikit-learn/Image/2021-11-02_19-43-16.jpg')
```

[17]:

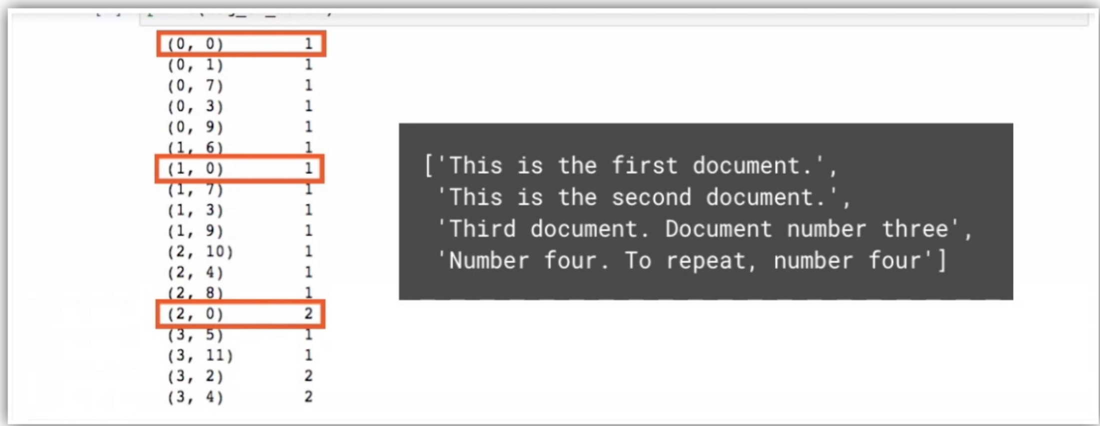


[20]:
```
'''
You can access the ID that corresponds to a particular word by calling␣
↪vectorizer.vocabulary.get on that word.
The word document corresponds to ID zero.
'''
vectorizer.vocabulary_.get('document')
```

[20]: 0

[22]:
```
'''
Typically the most frequently used words are assigned lower IDs.
The word document appears four times across our corpus. It's the most␣
↪frequently used word.
'''
Image('/Users/subhasish/Documents/APPLE/SUBHASISH/Development/GIT/Interstellar/
↪SB-AI-DEV/ML/SB/Classification/Janani Ravi/Building Machine Learning Models␣
↪in Python with scikit-learn/Image/2021-11-02_19-46-20.jpg')
```

[22]:



[23]:
```python
'''
Vectorizer. vocabulary will give you access to all words in our vocabulary.␣
↪Here are the 12 different words.
'''
vectorizer.vocabulary_
```

[23]:
```
{'this': 9,
 'is': 3,
 'the': 7,
 'first': 1,
 'document': 0,
 'second': 6,
 'third': 8,
 'number': 4,
 'three': 10,
 'four': 2,
 'to': 11,
 'repeat': 5}
```

[28]:
```python
'''
Let's view this bag of words in a tabular format in order to understand how␣
↪exactly it is set up.
We'll use the pandas dataframe for this.
This dataframe will be in the form of a 2D array where the rows are the␣
↪individual documents
and the columns are the words in our vocabulary.

The numbers which are present in the cells of this tabular format represent the␣
↪frequency of individual words
in each document.
```

```
['This is the first document.',
          'This is the second document.',
          'Third document. Document number three',
          'Number four. To repeat, number four']

'''
pd.DataFrame(bag_of_words.toarray(),columns=vectorizer.get_feature_names())
```

[28]:

```
   document  first  four  is  number  repeat  second  the  third  this  three  \
0         1      1     0   1       0       0       0    1      0     1      0
1         1      0     0   1       0       0       1    1      0     1      0
2         2      0     0   0       1       0       0    0      1     0      1
3         0      0     2   0       2       1       0    0      0     0      0

   to
0   0
1   0
2   0
3   1
```

[29]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

[30]:

```
'''
Let's now look at the TF-IDF vectorizer.
This associates scores with every word in our document corpus.
Here is a bag of words representation which is the output of the TF-IDF␣
 ↪vectorizer.
Every word in every document is associated with a score.
'''
vectorizer=TfidfVectorizer()
bag_of_words=vectorizer.fit_transform(corpus)
```

[31]:

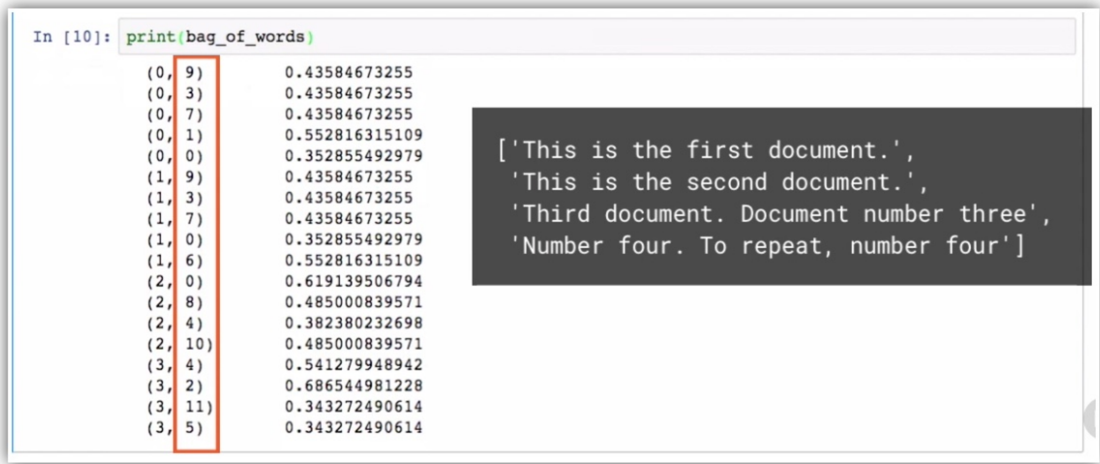
```
print(bag_of_words)
```

```
  (0, 0)        0.3528554929793508
  (0, 1)        0.5528163151092931
  (0, 7)        0.43584673254990375
  (0, 3)        0.43584673254990375
  (0, 9)        0.43584673254990375
  (1, 6)        0.5528163151092931
  (1, 0)        0.3528554929793508
  (1, 7)        0.43584673254990375
  (1, 3)        0.43584673254990375
  (1, 9)        0.43584673254990375
  (2, 10)       0.4850008395708102
  (2, 4)        0.3823802326982809
  (2, 8)        0.4850008395708102
```

```
  (2, 0)        0.6191395067937654
  (3, 5)        0.3432724906138499
  (3, 11)       0.3432724906138499
  (3, 2)        0.6865449812276998
  (3, 4)        0.5412799489419371
```

[32]:
```python
'''
Every document has a unique ID. Every word has a unique ID as well, and a␣
 ↪document ID word ID
combination is associated with a score.
'''
Image('/Users/subhasish/Documents/APPLE/SUBHASISH/Development/GIT/Interstellar/
 ↪SB-AI-DEV/ML/SB/Classification/Janani Ravi/Building Machine Learning Models␣
 ↪in Python with scikit-learn/Image/2021-11-02_19-54-48.jpg')
```

[32]:



[33]:
```python
'''
 You can access the IDs assigned to individual words just like you did before.
'''
vectorizer.vocabulary_.get('document')
```

[33]: 0

[34]:
```python
'''
You can also represent this in a dataframe format.
The cells of this dataframe contain TF-IDF scores and not word frequencies.
'''
pd.DataFrame(bag_of_words.toarray(),columns=vectorizer.get_feature_names())
```

[34]:

```
   document     first      four        is   number    repeat    second  \
0  0.352855  0.552816  0.000000  0.435847  0.00000  0.000000  0.000000
1  0.352855  0.000000  0.000000  0.435847  0.00000  0.000000  0.552816
```

```
2  0.619140  0.000000  0.000000  0.000000  0.38238  0.000000  0.000000
3  0.000000  0.000000  0.686545  0.000000  0.54128  0.343272  0.000000

        the     third      this     three        to
0  0.435847  0.000000  0.435847  0.000000  0.000000
1  0.435847  0.000000  0.435847  0.000000  0.000000
2  0.000000  0.485001  0.000000  0.485001  0.000000
3  0.000000  0.000000  0.000000  0.000000  0.343272
```

[35]:

```python
'''
 And finally here is our complete vocabulary, all of 12 words.
'''
vectorizer.vocabulary_
```

[35]:

```
{'this': 9,
 'is': 3,
 'the': 7,
 'first': 1,
 'document': 0,
 'second': 6,
 'third': 8,
 'number': 4,
 'three': 10,
 'four': 2,
 'to': 11,
 'repeat': 5}
```

[36]:

```python
'''
If you have a very large vocabulary of words, we can choose to use the
HashingVectorizer rather than the CountVectorizer.
'''

from sklearn.feature_extraction.text import HashingVectorizer
```

[37]:

```python
'''

The use of hashing buckets to represent words allows us to scale large data␣
 ↪sets when we use the HashingVectorizer.

The input argument to this vectorizer is the number of hash buckets,
which in our case is set to 8.

The result you see on screen is the numeric representation of all the words in␣
 ↪our four documents.
Notice that word IDs are from zero to seven because we have a total of eight␣
 ↪buckets.

```

```
Because the size of our vocabulary is larger than the number of buckets, which␣
 ↪is how it should be,
multiple words can hash to the same bucket.

One disadvantage of the HashingVectorizer is that there is no way to get back␣
 ↪to the
original word from its hash bucket value.

The frequencies of each token is not represented in raw number form. This is␣
 ↪some kind of normalized form.
'''
vectorizer=HashingVectorizer(n_features=8)
feture_vector=vectorizer.fit_transform(corpus)
print(feture_vector)
```

```
  (0, 0)        -0.8944271909999159
  (0, 5)        0.4472135954999579
  (0, 6)        0.0
  (1, 0)        -0.5773502691896258
  (1, 3)        0.5773502691896258
  (1, 5)        0.5773502691896258
  (1, 6)        0.0
  (2, 0)        -0.7559289460184544
  (2, 3)        0.3779644730092272
  (2, 5)        0.3779644730092272
  (2, 7)        0.3779644730092272
  (3, 0)        0.31622776601683794
  (3, 3)        0.31622776601683794
  (3, 5)        0.6324555320336759
  (3, 7)        0.6324555320336759
```

0.0.1 Hashing Vectorizer

- One issue with CountVectorizer and TF-IDF Vectorizer is that the number of features can get very large if the vocabulary is very large
- The whole vocabulary will be stored in memory, and this may end up taking a lot of space
- With Hashing Vectorizer, one can limit the number of fe/atures, let's say to a number n
- Each word will be hashed to one of the n values
- There will collisions where different words will be hashed to the same value
- In many instances, peformance does not really suffer in spite of the collisions

[ ]:
```

```