Implementing Bag of Words

Fit method:

1. With this function, we will find all unique words in the data and we will assign a dimension-number to each unique word.

2. We will create a python dictionary to save all the unique words, such that the key of dictionary represents a unique word and the corresponding value represent it's dimension-number.

3. For example, if you have a review, **___'very bad pizza'___** then you can represent each unique word with a dimension_number as,
dict = { 'very' : 1, 'bad' : 2, 'pizza' : 3}

In [3]:

```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
from tqdm import tqdm
import os
```

In [2]:

```
from tqdm import tqdm # tqdm is a library that helps us to visualize the runtime of for loop. refe
r this to know more about tqdm
#https://tqdm.github.io/

# it accepts only list of sentances
def fit(dataset):
    unique_words = set() # at first we will initialize an empty set
    # check if its list type or not
    if isinstance(dataset, (list,)):
        for row in dataset: # for each review in the dataset
            for word in row.split(" "): # for each word in the review. #split method converts a str
ing into list of words
                if len(word) < 2:
                    continue
                unique_words.add(word)
        unique_words = sorted(list(unique_words))
        vocab = {j:i for i,j in enumerate(unique_words)}
        return vocab
    else:
        print("you need to pass list of sentance")
```

In [4]:

```
vocab = fit(["abc def aaa prq", "lmn pqr aaaaaaa aaa abbb baaa"])
print(vocab)
```

```
{'aaa': 0, 'aaaaaaa': 1, 'abbb': 2, 'abc': 3, 'baaa': 4, 'def': 5, 'lmn': 6, 'pqr': 7, 'prq': 8}
```

What is a Sparse Matrix?

1. Before going further into details about Transform method, we will understand what sparse matrix is.

2. Sparse matrix stores only non-zero elements and they occupy less amount of RAM comapre to a dense matrix. You can refer to this link.

3. For example, assume you have a matrix,

```
[[1, 0, 0, 0, 0],
[0, 0, 0, 1, 0],
```

```
       [0, 0, 4, 0, 0]]
```

In [0]:

```
from sys import getsizeof
import numpy as np
# we store every element here
a = np.array([[1, 0, 0, 0, 0], [0, 0, 0, 1, 0], [0, 0, 4, 0, 0]])
print(getsizeof(a))

# here we are storing only non zero elements here (row, col, value)
a = [ (0, 0, 1), (1, 3, 1), (2,2,4)]
# with this way of storing we are saving alomost 50% memory for this example
print(getsizeof(a))
```

```
172
88
```

How to write a Sparse Matrix?:

1. You can use csr_matrix() method of scipy.sparse to write a sparse matrix.
2. You need to pass indices of non-zero elements into csr_matrix() for creating a sparse matrix.
3. You also need to pass element value of each pair of indices.
4. You can use lists to save the indices of non-zero elements and their corresponding element values.
5. For example,
   - Assume you have a matrix,

     ```
     [[1, 0, 0],
     [0, 0, 1],
     [4, 0, 6]]
     ```

   - Then you can save the indices using a list as,
     list_of_indices = [(0,0), (1,2), (2,0), (2,2)]
   - And you can save the corresponding element values as,
     element_values = [1, 1, 4, 6]
6. Further you can refer to the documentation here.

Transform method:

1. With this function, we will write a feature matrix using sprase matrix.

In [5]:

```
from collections import Counter
from scipy.sparse import csr_matrix
test = 'abc def abc def zzz zzz pqr'
a = dict(Counter(test.split()))
for i,j in a.items():
    print(i, j)
```

```
abc 2
def 2
zzz 2
pqr 1
```

In [10]:

```
# https://stackoverflow.com/questions/9919604/efficiently-calculate-word-frequency-in-a-string
# https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.sparse.csr_matrix.html
# note that we are we need to send the preprocessing text here, we have not inlcuded the processin
g

def transform(dataset,vocab):
```

```
    rows = []
    columns = []
    values = []
    if isinstance(dataset, (list,)):
        for idx, row in enumerate(tqdm(dataset)): # for each document in the dataset
            # it will return a dict type object where key is the word and values is its frequency,
{word:frequency}
            word_freq = dict(Counter(row.split()))
            # for every unique word in the document
            for word, freq in word_freq.items():  # for each unique word in the review.

                if len(word) < 2:
                    continue
                # we will check if its there in the vocabulary that we build in fit() function
                # dict.get() function will return the values, if the key doesn't exits it will retu
rn -1
                    col_index = vocab.get(word, -1) # retreving the dimension number of a word
                # if the word exists
                print(col_index)
                if col_index !=-1:
                    # we are storing the index of the document
                    rows.append(idx)
                    # we are storing the dimensions of the word
                    columns.append(col_index)
                    # we are storing the frequency of the word
                    values.append(freq)
        return csr_matrix((values, (rows,columns)), shape=(len(dataset),len(vocab)))
    else:
        print("you need to pass list of strings")
```

In [11]:

```
strings = ["the method of lagrange multipliers is the economists workhorse for solving
optimization problems",
           "the technique is a centerpiece of economic theory but unfortunately its usually taught
poorly"]
vocab = fit(strings)
print(list(vocab.keys()))
print(transform(strings, vocab).toarray())
```

```
['but', 'centerpiece', 'economic', 'economists', 'for', 'is', 'its', 'lagrange', 'method',
'multipliers', 'of', 'optimization', 'poorly', 'problems', 'solving', 'taught', 'technique',
'the', 'theory', 'unfortunately', 'usually', 'workhorse']
```

```
  0%|                                                    | 0/2 [00:00<?, ?it/s]
```

```
17
8
10
7
9
5
3
21
4
14
11
13
17
16
5
1
10
2
18
0
19
6
20
15
12
```

```
100%|████████████████████████████████████████████| 2/2 [00:00<00:00, 28.98it/s]
```

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

Comparing results with countvectorizer

In [0]:

```
from sklearn.feature_extraction.text import CountVectorizer

vec = CountVectorizer(analyzer='word')

vec.fit(strings)
feature_matrix_2 = vec.transform(strings)
print(feature_matrix_2.toarray())
```

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

In [0]:

```

```