NLP — Text Encoding: A Beginner's Guide

Bishal Bose · Follow
9 min read · Nov 7, 2020

— In this blog we will understand **WHAT** Text Encoding is? **HOW** to perform it? And rather **WHY** to perform it?

Source: Google

Let's try to understand a few basic rules first…

> *1. Machine doesn't understand characters, words or sentences.*
>
> *2. Machines can only process numbers.*
>
> *3. Text data must be encoded as numbers for input or output for any machine.*

WHY to perform text encoding?

As mentioned in the above points we cannot pass raw text into machines as input until and unless we convert them into numbers, hence we need to perform text encoding.

WHAT is text encoding?

Text encoding is a process to convert meaningful text into number / vector representation so as to preserve the context and relationship between words and sentences, such that a machine can understand the pattern associated in any text and can make out the context of sentences.

HOW to encode text for any NLP task?

There are a lot of methods to convert Text into numerical vectors, they are:

> \- Index-Based Encoding
>
> \- Bag of Words (BOW)
>
> \- TF-IDF Encoding
>
> \- Word2Vector Encoding
>
> \- BERT Encoding

As this is a basic explanation of NLP text encoding hence we will be skipping the last 2 methods, i.e. Word2Vector and BERT as they are quite complex and powerful implementations of Deep Learning method based Text Embedding to convert text into vector encoding.

You can find in-depth information about Word2Vector in my other blog stated here: *NLP — Text Encoding: Word2Vec*

Before we deep dive into each method let's set some ground examples so as to make it easier to follow through.

Document Corpus: This is the whole set of text we have, basically our text corpus, can be anything like news articles, blogs, etc.… etc.…

Example: We have 5 sentences namely, ["this is a good phone" , "this is a bad mobile" , "she is a good cat" , "he has a bad temper" , "this mobile phone is not good"]

Data Corpus: It is the collection of unique words in our document corpus, i.e. in our case it looks like this:

["a" , "bad" , "cat" , "good" , "has" , "he" , "is" , "mobile" , "not" , "phone" , "she" , "temper" , "this"]

We will stick to these sentences to understand each embedding method.

This will make it easier to understand and grasp the intuition behind these methods.

So let's try to understand each of them one by one:

1. Index-Based Encoding:

As the name mentions, Index based, we surely need to give all the unique words an index, like we have separated out our Data Corpus, now we can index them individually, like…

a : 1

bad : 2

...

this : 13

Now that we have assigned a unique index to all the words so that based on the index we can uniquely identify them, we can convert our sentences using this index-based method.

It is very trivial to understand, that we are just replacing the words in each sentence with their respective indexes.

Our document corpus becomes:

[13 7 1 4 10] , [13 7 1 2 8] , [11 7 1 4 3] , [6 5 1 2 12] , [13 8 10 7 9 4]

Now we have encoded all the words with index numbers, and this can be used as input to any machine since machine understands number.

But there is a tiny bit of issue which needs to be addressed first and that is the consistency of the input. Our input needs to be of the same length as our model, it cannot vary. It might vary in the real world but needs to be taken care of when we are using it as input to our model.

Now as we can see the first sentence has 5 words, but the last sentence has 6 words, this will cause an imbalance in our model.

So to take care of that issue what we do is max padding, which means we take the longest sentence from our document corpus and we pad the other sentence to be as long. This means if all of my sentences are of 5 words and one sentence is of 6 words I will make all the sentences of 6 words.

Now how do we add that extra word here? In our case how do we add that extra index here?

If you have noticed we didn't use 0 as an index number, and preferably that will not be used anywhere even if we have 100000 words long data corpus, hence we use 0 as our padding index. This also means that we are appending nothing to our actual sentence as 0 doesn't represent any specific word, hence the integrity of our sentences are intact.

So finally our index based encodings are as follows:

[13 7 1 4 10 0] ,

[13 7 1 2 8 0] ,

[11 7 1 4 3 0] ,

[6 5 1 2 12 0] ,

[13 8 10 7 9 4]

And this is how we keep our input's integrity the same and without disturbing the context of our sentences as well.

Index-Based Encoding considers the sequence information in text encoding.

2. Bag of Words (BOW):

Bag of Words or BoW is another form of encoding where we use the whole data corpus to encode our sentences. It will make sense once we see actually how to do it.

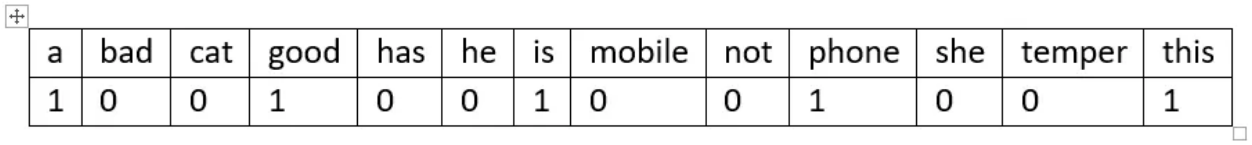
Data Corpus:

[“a” , “bad” , “cat” , “good” , “has” , “he” , “is” , “mobile” , “not” , “phone” , “she” , “temper” , “this”]

As we know that our data corpus will never change, so if we use this as a baseline to create encodings for our sentences, then we will be on an upper hand to not pad any extra words.

Now, 1st sentence we have is this : “this is a good phone”

How do we use the whole corpus to represent this sentence?



a	bad	cat	good	has	he	is	mobile	not	phone	she	temper	this
1	0	0	1	0	0	1	0	0	1	0	0	1

So our first sentence becomes a combination of all the words we have and we do not have.

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

This is how our first sentence is represented.

Now there are 2 kinds of BOW:

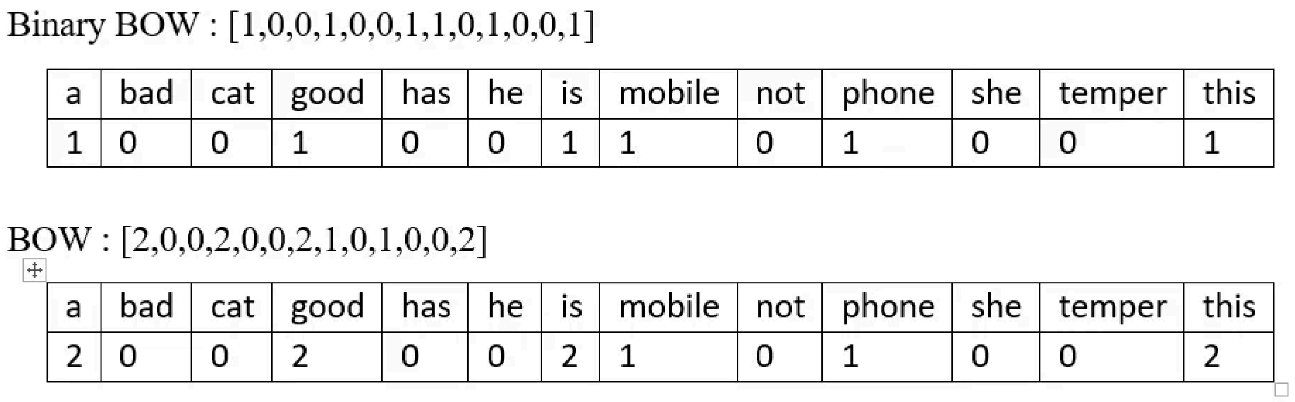
1. Binary BOW.

2. BOW

The difference between them is, in Binary BOW we encode 1 or 0 for each word appearing or non-appearing in the sentence. We do not take into consideration the frequency of the word appearing in that sentence.

In BOW we also take into consideration the frequency of each word occurring in that sentence.

Let's say our text sentence is "this is a good phone this is a good mobile" (FYI just for reference)



Binary BOW : [1,0,0,1,0,0,1,1,0,1,0,0,1]

a	bad	cat	good	has	he	is	mobile	not	phone	she	temper	this
1	0	0	1	0	0	1	1	0	1	0	0	1

BOW : [2,0,0,2,0,0,2,1,0,1,0,0,2]

a	bad	cat	good	has	he	is	mobile	not	phone	she	temper	this
2	0	0	2	0	0	2	1	0	1	0	0	2

If you see carefully, we considered the number of times the words "this", "a", "is" and "good" have occurred.

So that's the only difference between Binary BOW and BOW.

BOW totally discards the sequence information of our sentences.

3. TF-IDF Encoding:

Term Frequency — Inverse Document Frequency

As the name suggests, here we give every word a relative frequency coding w.r.t the current sentence and the whole document.

Term Frequency: Is the occurrence of the current word in the current sentence w.r.t the total number of words in the current sentence.

Inverse Data Frequency: Log of Total number of words in the whole data corpus w.r.t the total number of sentences containing the current word.

TF:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

Term-Frequency

IDF:

$$idf(w) = log(\frac{N}{df_t})$$

Inverse-Data-Frequency

One thing to note here is we have to calculate the word frequency of each word for that particular sentence, because depending on the number of times a word occurs in a sentence the TF value can change, whereas the IDF value remains constant, until and unless new sentences are getting added.

Let's try to understand by experimenting it out:

Data Corpus: ["a" , "bad" , "cat" , "good" , "has" , "he" , "is" , "mobile" , "not" , "phone" , "she" , "temper" , "this"]

TF-IDF : "this" in sentence1 : Number of "this" word in sentence1 / total number of words in sentence1

IDF : log(total number of words in the whole data corpus / total number of sentences having "this" word)

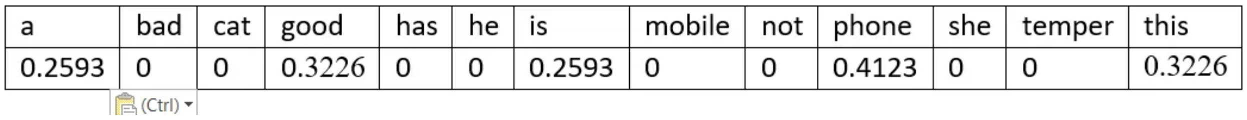
TF : 1 / 5 = 0.2

IDF : loge(13 / 3) = 1.4663

TF-IDF : 0.2 * 1.4663 = 0.3226

So we associate "this" : 0.3226; similarly we can find out TF-IDF for every word in that sentence and then rest of the process remains same as BOW, here we replace the word not with the frequency of its occurrence but rather with the TF-IDF value for that word.

So let's try to encode our first sentence: "this is a good phone"



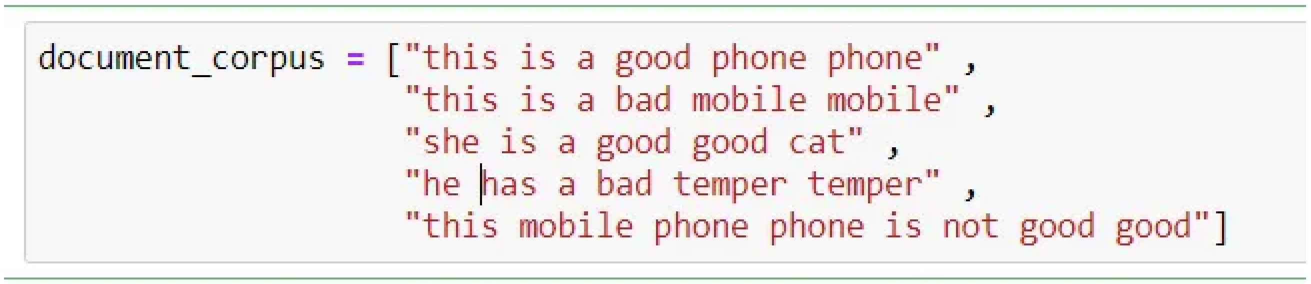
a	bad	cat	good	has	he	is	mobile	not	phone	she	temper	this
0.2593	0	0	0.3226	0	0	0.2593	0	0	0.4123	0	0	0.3226

As we can see that we have replaced all the words appearing in that sentence with their respective tf-idf values, one thing to notice here is, we have similar tf-idf values of multiple words. This is a rare case that has happened with us as we had few documents and almost all words had kind of similar frequencies.

And this is how we achieve TF-IDF encoding of text.

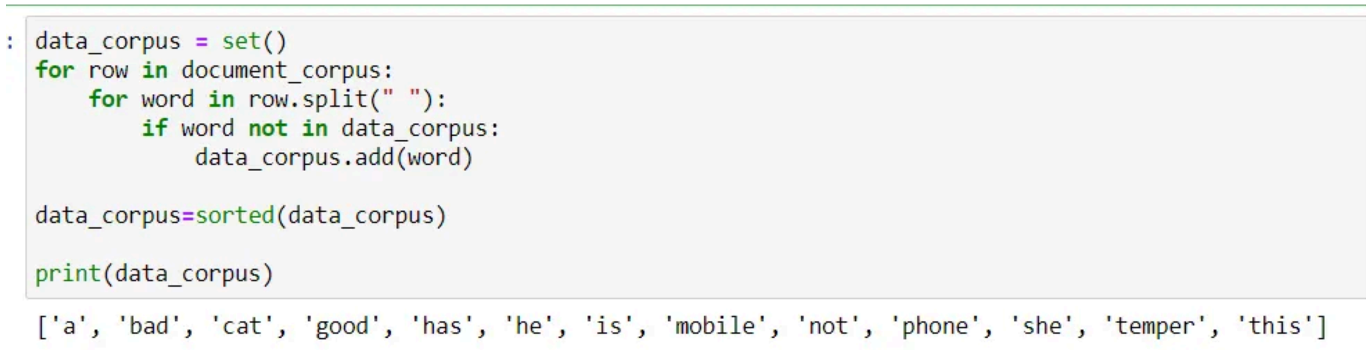
Self — Implementation:

Now we will try to implement them on our own:



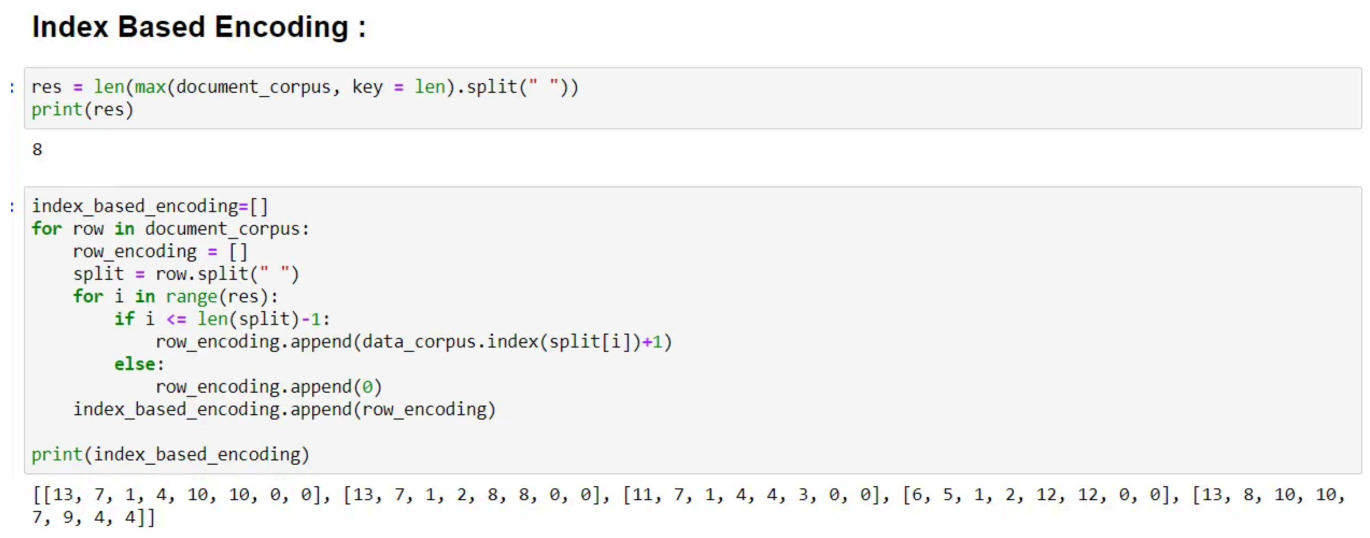
```
document_corpus = ["this is a good phone phone" ,
                   "this is a bad mobile mobile" ,
                   "she is a good good cat" ,
                   "he has a bad temper temper" ,
                   "this mobile phone phone is not good good"]
```

This is our document corpus as mentioned above. I have tweaked the data so as to make it more understandable, it will give more sense of the encodings.



```
: data_corpus = set()
  for row in document_corpus:
      for word in row.split(" "):
          if word not in data_corpus:
              data_corpus.add(word)

  data_corpus=sorted(data_corpus)

  print(data_corpus)

  ['a', 'bad', 'cat', 'good', 'has', 'he', 'is', 'mobile', 'not', 'phone', 'she', 'temper', 'this']
```

This is how we are creating our data corpus based on any document corpus we have.



Index Based Encoding :

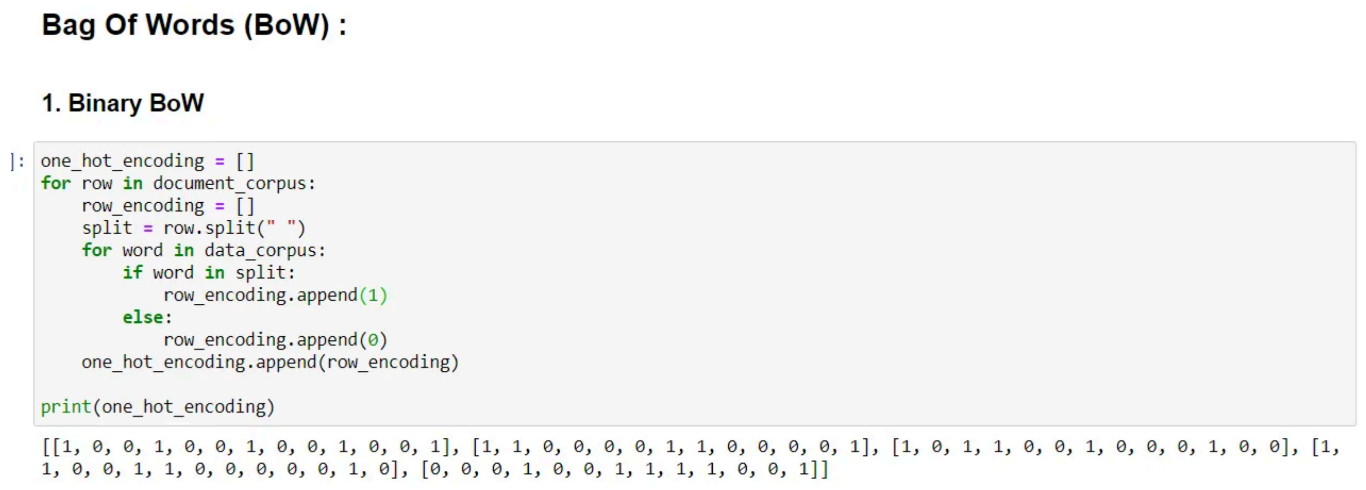
```
res = len(max(document_corpus, key = len).split(" "))
print(res)
```

```
8
```

```
index_based_encoding=[]
for row in document_corpus:
    row_encoding = []
    split = row.split(" ")
    for i in range(res):
        if i <= len(split)-1:
            row_encoding.append(data_corpus.index(split[i])+1)
        else:
            row_encoding.append(0)
    index_based_encoding.append(row_encoding)

print(index_based_encoding)
```

```
[[13, 7, 1, 4, 10, 10, 0, 0], [13, 7, 1, 2, 8, 8, 0, 0], [11, 7, 1, 4, 4, 3, 0, 0], [6, 5, 1, 2, 12, 12, 0, 0], [13, 8, 10, 10,
7, 9, 4, 4]]
```

First we find out the max length of a sentence and then using our data corpus we encode all our text with index based scheme.



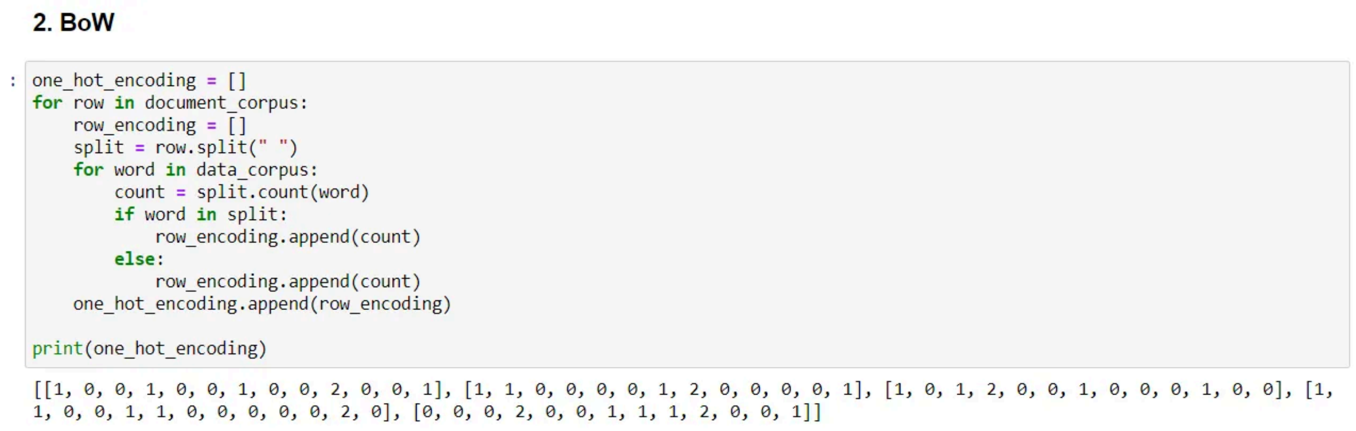
Bag Of Words (BoW) :

1. Binary BoW

```
one_hot_encoding = []
for row in document_corpus:
    row_encoding = []
    split = row.split(" ")
    for word in data_corpus:
        if word in split:
            row_encoding.append(1)
        else:
            row_encoding.append(0)
    one_hot_encoding.append(row_encoding)

print(one_hot_encoding)
```

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

Implementing Binary BoW, where we place 1 for every word encountered in the sentence in the data corpus and 0 for the rest.

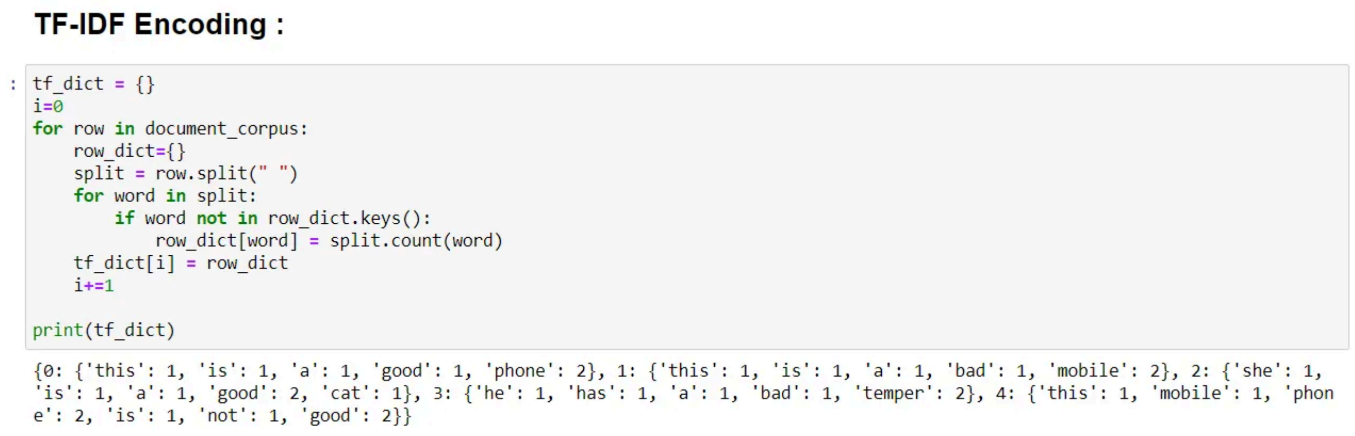


2. BoW

```
one_hot_encoding = []
for row in document_corpus:
    row_encoding = []
    split = row.split(" ")
    for word in data_corpus:
        count = split.count(word)
        if word in split:
            row_encoding.append(count)
        else:
            row_encoding.append(count)
    one_hot_encoding.append(row_encoding)

print(one_hot_encoding)
```

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

Implementing BoW, here we have to encode the count of each occurrences of the word in that particular sentence to the data corpus, and 0 for the rest.

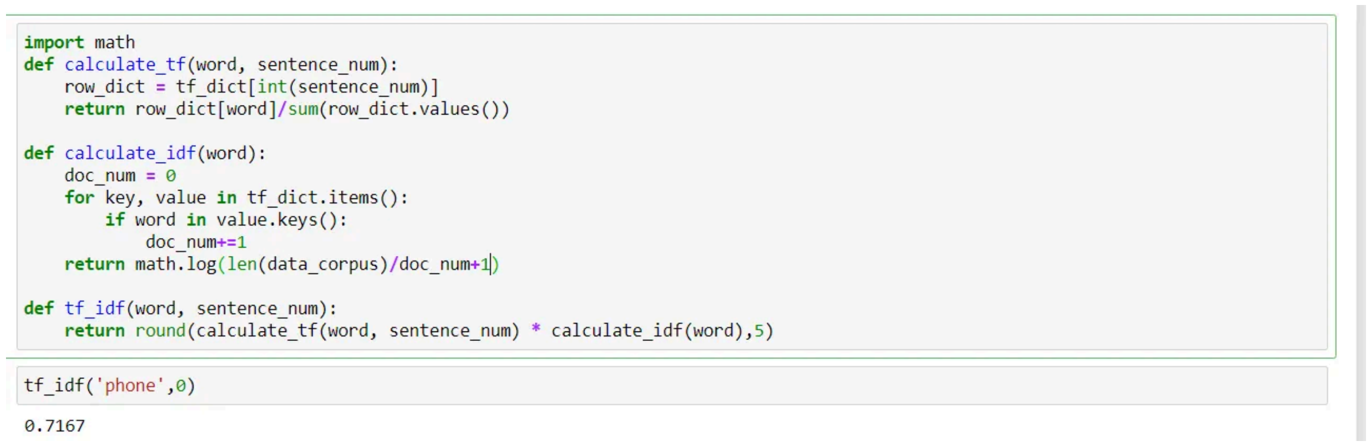


TF-IDF Encoding :

```
tf_dict = {}
i=0
for row in document_corpus:
    row_dict={}
    split = row.split(" ")
    for word in split:
        if word not in row_dict.keys():
            row_dict[word] = split.count(word)
    tf_dict[i] = row_dict
    i+=1

print(tf_dict)
```

```
{0: {'this': 1, 'is': 1, 'a': 1, 'good': 1, 'phone': 2}, 1: {'this': 1, 'is': 1, 'a': 1, 'bad': 1, 'mobile': 2}, 2: {'she': 1,
'is': 1, 'a': 1, 'good': 2, 'cat': 1}, 3: {'he': 1, 'has': 1, 'a': 1, 'bad': 1, 'temper': 2}, 4: {'this': 1, 'mobile': 1, 'phon
e': 2, 'is': 1, 'not': 1, 'good': 2}}
```

First calculating frequency of every element w.r.t the sentences.

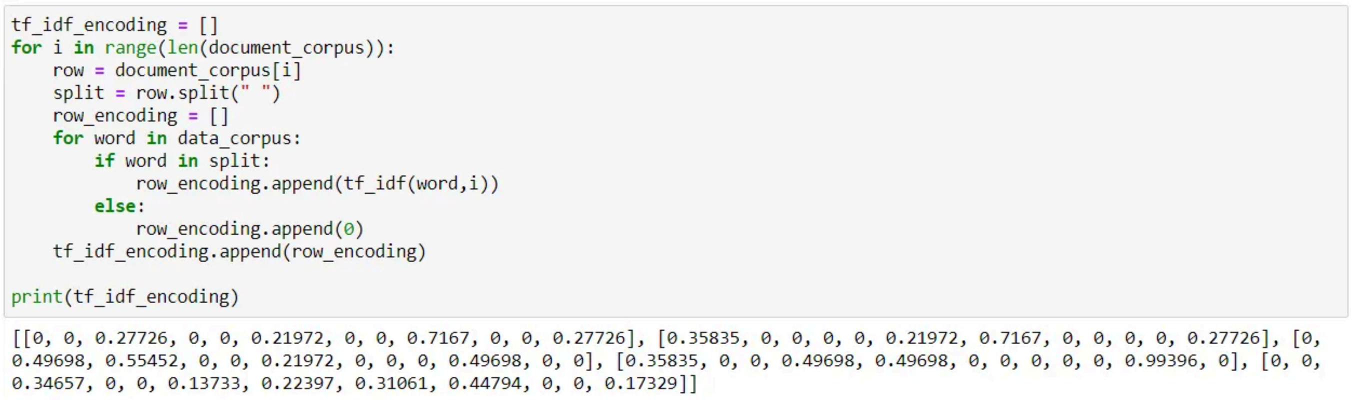


```
import math
def calculate_tf(word, sentence_num):
    row_dict = tf_dict[int(sentence_num)]
    return row_dict[word]/sum(row_dict.values())

def calculate_idf(word):
    doc_num = 0
    for key, value in tf_dict.items():
        if word in value.keys():
            doc_num+=1
    return math.log(len(data_corpus)/doc_num+1)

def tf_idf(word, sentence_num):
    return round(calculate_tf(word, sentence_num) * calculate_idf(word),5)
```

```
tf_idf('phone',0)
```

```
0.7167
```

Creating function to calculate tf-idf for every word in a particular sentence, which takes reference of the above created frequencies.

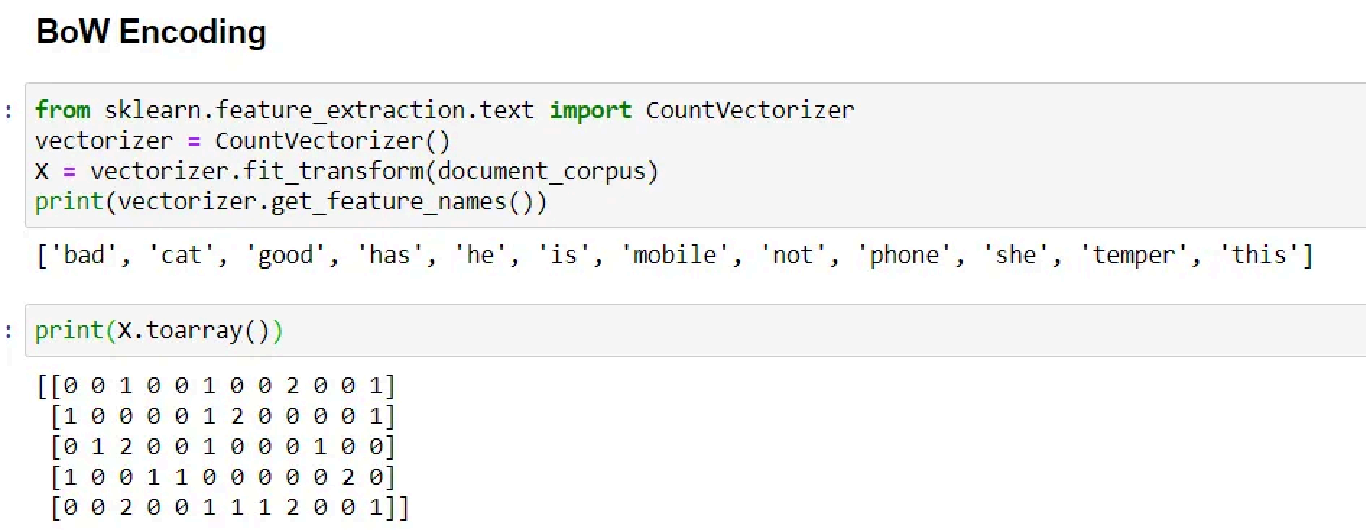


```
tf_idf_encoding = []
for i in range(len(document_corpus)):
    row = document_corpus[i]
    split = row.split(" ")
    row_encoding = []
    for word in data_corpus:
        if word in split:
            row_encoding.append(tf_idf(word,i))
        else:
            row_encoding.append(0)
    tf_idf_encoding.append(row_encoding)

print(tf_idf_encoding)
```

```
[[0, 0, 0.27726, 0, 0, 0.21972, 0, 0, 0.7167, 0, 0, 0.27726], [0.35835, 0, 0, 0, 0, 0.21972, 0.7167, 0, 0, 0, 0, 0.27726], [0,
0.49698, 0.55452, 0, 0, 0.21972, 0, 0, 0, 0.49698, 0, 0], [0.35835, 0, 0, 0.49698, 0.49698, 0, 0, 0, 0, 0, 0.99396, 0], [0, 0,
0.34657, 0, 0, 0.13733, 0.22397, 0.31061, 0.44794, 0, 0, 0.17329]]
```

Finally creating the tf-idf vectors for our module.

Python Library Implementation:



BoW Encoding

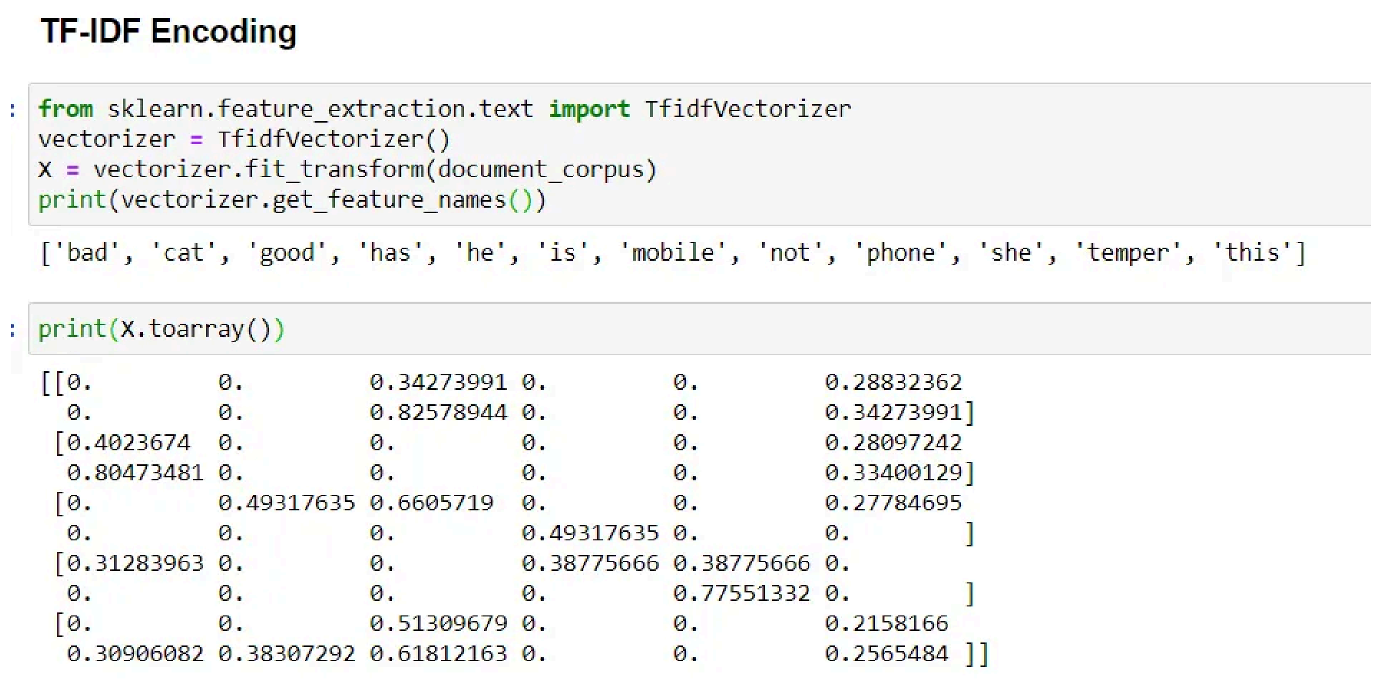
```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(document_corpus)
print(vectorizer.get_feature_names())
```

```
['bad', 'cat', 'good', 'has', 'he', 'is', 'mobile', 'not', 'phone', 'she', 'temper', 'this']
```

```
print(X.toarray())
```

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

CountVectorizer from Scikit-Learn module gives us the BoW representation of text. We can use different parameters to calculate Binary BoW or BoW and much more customizations.



TF-IDF Encoding

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(document_corpus)
print(vectorizer.get_feature_names())
```

```
['bad', 'cat', 'good', 'has', 'he', 'is', 'mobile', 'not', 'phone', 'she', 'temper', 'this']
```

```
print(X.toarray())
```

```
[[0.         0.         0.34273991 0.         0.         0.28832362
  0.         0.         0.82578944 0.         0.         0.34273991]
 [0.4023674  0.         0.         0.         0.         0.28097242
  0.80473481 0.         0.         0.         0.         0.33400129]
 [0.         0.49317635 0.6605719  0.         0.         0.27784695
  0.         0.         0.         0.49317635 0.         0.        ]
 [0.31283963 0.         0.         0.38775666 0.38775666 0.
  0.         0.         0.         0.         0.77551332 0.        ]
 [0.         0.         0.51309679 0.         0.         0.2158166
  0.30906082 0.38307292 0.61812163 0.         0.         0.2565484 ]]
```

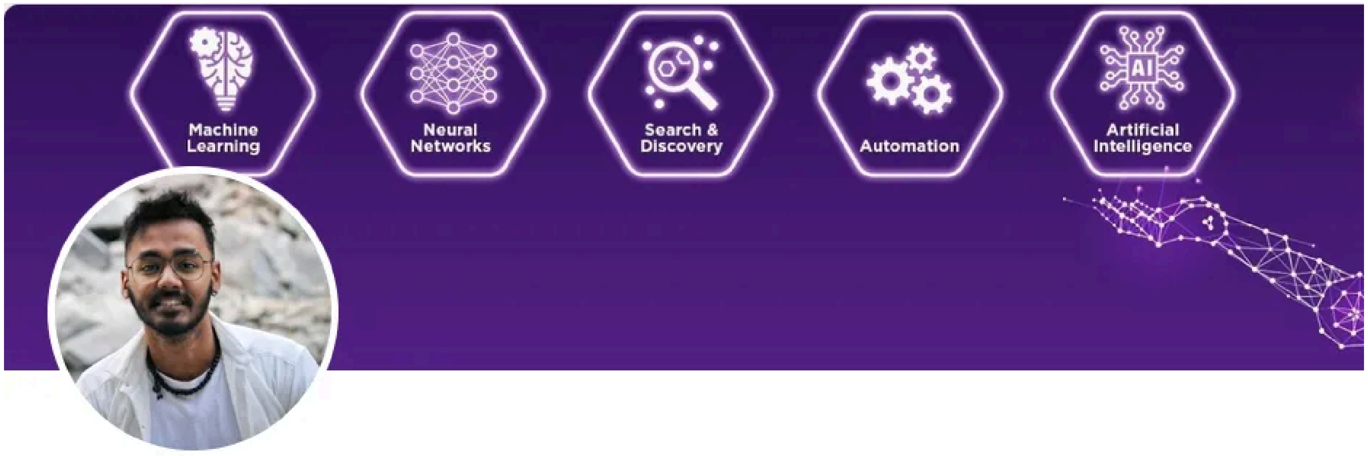
TfidfVectorizer from Scikit-Learn module gives us the TF-IDF representations of the text encoding, similar to CountVectorizer we can set a lot of parameters for conversion.

There will be slight difference in values, as we implemented the basic version of TF-IDF. In Scikit-learn library they implement TF-IDF with different methods, hence we see such differences, otherwise, more or less things are the same.

So there you have it: basic NLP text encoding implementations.

You can find the code in my GitHub link **here**.

Next Section: *NLP — Text Encoding: Word2Vec*



Stay updated with all my blogs & updates on Linked In. Welcome to my network. Follow me on Linked In Here — -> https://www.linkedin.com/in/bishalbose294/

Machine Learning | Naturallanguageprocessing | NLP | Deep Learning | Bishal Bose

Written by Bishal Bose

150 Followers · 1 Following

Follow

Senior Lead Data Scientist @ MNC | Applied & Research Scientist | Google & AWS Certified | Gen AI | LLM | NLP | CV | TS Forecasting | Predictive Modeling

Responses (3)

Write a response

What are your thoughts?

Medo Sas
Nov 30, 2022

Excellent article and amazing explanation, thank you so much Bishal

1 Reply

Kichi Alae
Jul 2, 2021

Good article I love it ! juste one thing for TF-IDF i think it's term frequency inverse document frequency and not data frenquency.

1 reply Reply

Walter Coleman
Apr 13, 2021

Hi, I am processing bank statements, and due to the nature of the memo structure, I have about 800 unique words in my corpus.

There are some ways I am narrow this down, but if we have far more than 100 unique words, what are our options?

1 reply Reply

More from Bishal Bose

Bishal Bose

Artificial Intelligence: A Boon or Curse ?

What is Artificial Intelligence ?

Jan 12, 2023 1.2K 2

Bishal Bose

Introduction to Basic Computer Vision & Image Processing

Q: What is Computer Vision?

Apr 27, 2022 1.1K

Bishal Bose

NLP — Text Encoding: Word2Vec

— In this blog we will understand the working and intuition of Word2Vec.

Jul 10, 2021 1.5K 2

Bishal Bose

Optimizing Retrieval-Augmented Generation (RAG): From...

Retrieval-Augmented Generation (RAG) has revolutionized how Large Language Models...

Mar 15 651

See all from Bishal Bose

Recommended from Medium

Vipra Singh

LLM Architectures Explained: BERT (Part 8)

Deep Dive into the architecture & building real-world applications leveraging NLP...

Nov 17, 2024 277 2

Abhijat Sarari

Next Sentence Prediction using BERT

In the world of Natural Language Processing (NLP), predicting relationships between...

Nov 8, 2024 10

In about ai by Edgar Bermudez

Understanding Embedding Models in the Context of Large Language...

Large Language Models (LLMs) like GPT, BERT, and similar architectures have...

LM Po

Understanding Tokenization

BPE, WordPiece, and SentencePiece in NLP

Jan 28 1

Jan 12 8

Karthikeyan Dhanakotti

Choosing the Best Approach for Multi-Class Text Classification: A...

Text classification is one of the foundational tasks in natural language processing (NLP),...

Jan 9 1

Muneeb S. Ahmad

Mastering NLP with GloVe Embeddings: Word Similarity,...

Introduction

Oct 21, 2024

See more recommendations