How NLP does this task???

Answer is Text Vectorization or Word Embedding

- **Text vectorization is a process** to convert each text document into a numeric vector.
- One hot encoding
- Word embeding

Why do we need Word Embeddings?

- As we know that many Machine Learning algorithms and almost all Deep Learning Architectures are not capable of processing strings or plain text in their raw form.

- In a broad sense, they require numerical numbers as inputs to perform any sort of task, such as classification, regression, clustering, etc.

- Also, from the huge amount of data that is present in the text format, it is imperative to extract some knowledge out of it and build any useful applications.

Text Vectorization & Transformation

Text vectorization is two types.

- Frequency-based or Statistical based Word Embedding
- Prediction based Word Embedding

Methods used for text vectorization

- Binary Term Frequency.
- Bag of Words (BoW) Term Frequency.
- TF-IDF
- Word2Vec
- Glove embedding

One-Hot Encoding (OHE)

Sentence: I am teaching NLP in Python

A word in this sentence may be “NLP”, “Python”, “teaching”, etc.
Since a dictionary is defined as the list of all unique words present in the sentence.

So, a dictionary may look like –
Dictionary: [‘I’, ’am’, ’teaching’,’ NLP’,’ in’, ’Python’]
Therefore, the vector representation in this format according to the above dictionary is

Vector for NLP: [0,0,0,1,0,0]
Vector for Python: [0,0,0,0,0,1]

Disadvantages of One-hot Encoding

1. One of the disadvantages of One-hot encoding is that the Size of the vector is equal to the count of unique words in the vocabulary.

2. One-hot encoding does not capture the relationships between different words. Therefore, it does not convey information about the context.

Count Vectorizer

- It creates a document term matrix, which is a set of dummy variables that indicates if a particular word appears in the document.

- Count vectorizer will fit and learn the word vocabulary and try to create a document term matrix in which the individual cells denote the frequency of that word in a particular document, which is also known as term frequency, and the columns are dedicated to each word in the corpus.

- Matrix Formulation

✓Consider a Corpus C containing D documents $\{d_1,d_2\ldots..d_D\}$ from which we extract N unique tokens.

✓Now, the dictionary consists of these N tokens, and the size of the Count Vector matrix M formed is given by D X N.

✓Each row in the matrix M describes the frequency of tokens present in the document D(i).

Count Vectorizer

Document-1: He is a smart boy. She is also smart.

Document-2: Chirag is a smart person.

The dictionary created contains the list of unique tokens(words) present in the corpus

Unique Words: [‘He’, ’She’, ’smart’, ’boy’, ’Chirag’, ’person’]

Here, D=2, N=6

So, the count matrix M of size 2 X 6 will be represented as –

	He	She	smart	boy	Chirag	person
D1	1	1	2	1	0	0
D2	0	0	1	0	1	1

Vector for ‘smart’ is [2,1],

Vector for ‘Chirag’ is [0, 1], and so on.

Bag-of-Words (BoW)

- This vectorization technique converts the text content to numerical feature vectors.

- Bag of Words takes a document from a corpus and converts it into a numeric vector by mapping each document word to a feature vector for the machine learning model.

- It follows two steps

✓Tokenization

✓Vectors Creation

Bag-of-Words (BoW)

- **Tokenization**

It is the process of dividing each sentence into words or smaller parts, which are known as tokens. After the completion of tokenization, we will extract all the unique words from the corpus. Here corpus represents the tokens we get from all the documents and used for the bag of words creation.

- **Create vectors for each Sentence**

Here the vector size for a particular document is equal to the number of unique words present in the corpus. For each document, we will fill each entry of a vector with the corresponding word frequency in a particular document.

Bag-of-Words (BoW)

This burger is very tasty and affordable
This burger is not tasty and is affordable
This burger is very very delicious

Corpus

this burger is very tasty and affordable.
this burger is not tasty and is affordable.
this burger is very very delicious.

Tokenization

Unique words: [“and”, “affordable.”, “delicious.”, “is”, “not”, “burger”, “tasty”, “this”, “very”]

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

	and	affordable	delicious	is	not	burger	tasty	this	very
D1	1	1	0	1	0	1	1	1	1
D2	1	1	0	2	1	1	1	1	0
D3	0	0	1	1	0	1	0	1	2

- corpus = ['the quick brown fox jumped over the brown dog.’,
  'the quick brown fox.’,
  'the brown brown dog.’,
  'the fox ate the dog.']

	ate	brown	dog	fox	jumped	over	quick	the
Document 1	0	2	1	1	1	1	1	2
Document 2	0	1	0	1	0	0	1	1
Document 3	0	2	1	0	0	0	0	1
Document 4	1	0	1	1	0	0	0	2

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

Bag-of-Words (BoW)

Disadvantages

- This method doesn’t preserve the word order.

- It does not allow to draw of useful inferences for downstream NLP tasks.

TF-IDF Vectorization

- As we discussed in the above techniques that the BOW method is simple and works well, but the problem with that is that it treats all words equally. As a result, it cannot distinguish very common words or rare words. So, to solve this problem, TF-IDF comes into the picture!

- Term frequency-inverse document frequency (TF-IDF) gives a measure that takes the importance of a word into consideration depending on how frequently it occurs in a document and a corpus.

- Term frequency (TF) : the frequency of a word in a document
- Inverse document frequency (IDF) : It measures how common a particular word is across all the documents in the corpus.

- TF(term)= $\frac{Number\ of\ times\ term\ appear\ in\ a\ document}{total\ number\ of\ items\ in\ the\ document}$

- IDF (term) = log $\frac{Total\ number\ of\ document}{Number\ of\ document\ with\ term\ in\ it}$

- TFIDF (term) = TF(term) * IDF(term)

TF – frequency of world(term) in the corpus.

Doc 1 . "I will **go** to Mumbai then go to Pune"

I	1
Will	1
Go	2
To	2
Mumbai	1
Then	1
Pune	1

NTF = total count for word/total words (normalize)

I	Wil l	Go	To	Mumbai	Then	Pune
1/7	1/7	2/7	2/7	1/7	1/7	1/7

1 . "I will go to Mumbai then go to Pune"
2. "Yesterday I went to Mumbai"

I	1/7
Will	1/7
Go	2/7
To	2/7
Mumbai	1/7
Then	1/7
Pune	1/7

Yesterday	1/5
I	1/5
Went	1/5
To	1/5
Mumbai	1/5

World frequency in all documents

I	2
Will	1
Go	2
To	3
Mumbai	2
Then	1
Pune	1
Yesterday	1
went	1

TF

word / document	I	Will	Go	To	Mumbai	Then	Pune	Yesterday	Went
TF1	0.14	0.14	0.28	0.28	0.14	0.14	0.14	0	0
TF2	0.25	0	0	0.25	0.25	0	0	0.25	0.25

1 . "I will go to Mumbai then go to Pune"
2. "Yesterday I went to Mumbai"

I	1/7
Will	1/7
Go	2/7
To	2/7
Mumbai	1/7
Then	1/7
Pune	1/7

Yesterday	1/5
I	1/5
Went	1/5
To	1/5
Mumbai	1/5

Words	Word frequency in all documents	IDF	IDF
I	2	log(2/2)	0
Will	1	log(2/1)	0.69
Go	2	log(2/1)	0.69
To	2	log(2/2)	0
Mumbai	2	log(2/2)	0
Then	1	log(2/1)	0.69
Pune	1	log(2/1)	0.69
Yesterday	1	log(2/2)	0
Went	1	log(2/1)	0.69

IDF – inverse document frequency

$$IDF = LOG\left(\frac{\textit{No of documents}}{(\text{No of documents that contains a word})}\right)$$

IDF is used to calculate weight of words.

TF-IDF

words	Word frequency in all documents	TF1	TF2	IDF	IDF	TF-IDF IDF*TF1	TF-IDF IDF*TF2
I	2	0.14	0.25	Log(2/2)	0	0	0
Will	1	0.14	0	Log(2/1)	0.69	0.1	0
Go	2	0.28	0	Log(2/1)	0.69	0.19	0
To	3	0.28	0.25	Log(2/2)	0	0	0
Mumbai	2	0.14	0.25	Log(2/2)	0	0	0
Then	1	0.14	0	Log(2/1)	0.69	0.1	0
Pune	1	0.14	0	Log(2/1)	0.69	0.1	0
Yesterday	1	0	0.25	Log(2/1)	0.69	0	0.17
went	1	0	0.25	Log(2/1)	0.69	0	0.17

Thank You