1. Bag_Of_Words(Count Vectorizer) :-

- The Bag of Words (BoW) model is the simplest form of text representation in numbers. Like the term itself, we can represent a sentence as a bag of words vector (a string of numbers).
- Let's recall the three types of movie reviews we saw earlier:

```
Review 1: This movie is very scary and long
Review 2: This movie is not scary and is slow
Review 3: This movie is spooky and good
```

- We will first build a vocabulary from all the unique words in the above three reviews. The vocabulary consists of these 11 words: 'This', 'movie', 'is', 'very', 'scary', 'and', 'long', 'not', 'slow', 'spooky', 'good'.
- We can now take each of these words and mark their occurrence in the three movie reviews above with 1s and 0s. This will give us 3 vectors for 3 reviews:

	1 This	2 movie	3 is	4 very	5 scary	6 and	7 long	8 not	9 slow	10 spooky	11 good	Length of the review(in words)
Review 1	1	1	1	1	1	1	1	0	0	0	0	7
Review 2	1	1	2	0	0	1	1	0	1	0	0	8
Review 3	1	1	1	0	0	0	1	0	0	1	1	6

Vector of Review 1: [1 1 1 1 1 1 1 0 0 0 0]

Vector of Review 2: [1 1 2 0 0 1 1 0 1 0 0]

Vector of Review 3: [1 1 1 0 0 0 1 0 0 1 1]

And that's the core idea behind a Bag of Words (BoW) model.

In [1]:

```
import nltk

paragraph =  """  This movie is very scary and long.
                This movie is not scary and is slow.
                This movie is spooky and good.
                """


# Cleaning the texts
import re
sentences = nltk.sent_tokenize(paragraph)
corpus = []
for i in range(len(sentences)):
    review = re.sub('[^a-zA-Z]', ' ', sentences[i])
    review = review.lower()
    review = review.split()
    review = ' '.join(review)
    corpus.append(review)


# Creating the Bag of Words model
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features = 1500)
X = cv.fit_transform(corpus).toarray()
```

In [2]:

```
print(corpus)
X
```

```
['this movie is very scary and long', 'this movie is not scary and is slow',
'this movie is spooky and good']
```

Out[2]:

```
array([[1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1],
       [1, 0, 2, 0, 1, 1, 1, 1, 0, 1, 0],
       [1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0]], dtype=int64)
```

Here I Am not any preprocessing tool as i wish to show what exactly BOW doing. But in real word you must doing all the preprocess step then you will be apply the BOW

Drawbacks of using a Bag-of-Words (BoW) Model

In the above example, we can have vectors of length 11. However, we start facing issues when we come across new sentences:

- If the new sentences contain new words, then our vocabulary size would increase and thereby, the length of the vectors would increase too.
- Additionally, the vectors would also contain many 0s, thereby resulting in a sparse matrix (which is what we would like to avoid)
- We are retaining no information on the grammar of the sentences nor on the ordering of the words in the text.
- To Over Come this we use Term Frequency-Inverse Document Frequency (TF-IDF)

2. Term Frequency-Inverse Document Frequency (TF-IDF)

- “Term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.”

TF-IDF Vectorizer :

- TF-IDF stands for term frequency-inverse document frequency. It highlights a specific issue which might not be too frequent in our corpus but holds great importance. The TF–IFD value increases proportionally to the number of times a word appears in the document and decreases with the number of documents in the corpus that contain the word.
- It is composed of 2 sub-parts, which are :

```
1. Term Frequency (TF)
2. Inverse Document Frequency (IDF)
```

1. Term Frequency:-

- Term frequency specifies how frequently a term appears in the entire document.

```
Formula - TF =  Number of repetation word in sentence / total Number of word
```

```
### Example :-

- We will again use the same vocabulary we had built in the Bag-of-Words model to show how to calculate the TF for Review #2:

Review 2: This movie is not scary and is slow

Here,

- Vocabulary: ‘This’, ‘movie’, ‘is’, ‘very’, ‘scary’, ‘and’, ‘long’, ‘not’,  ‘slow’, ‘spooky’,  ‘good’

- Total Number of words in Review 2 = 8

TF (‘this’) = (number of times ‘this’ appears in review 2)/(number of terms in review 2) = 1/8
TF(‘movie’) = 1/8
TF(‘is’) = 2/8 = 1/4
TF(‘very’) = 0/8 = 0
TF(‘scary’) = 1/8
TF(‘and’) = 1/8
TF(‘long’) = 0/8 = 0
TF(‘not’) = 1/8
TF(‘slow’) = 1/8
TF( ‘spooky’) = 0/8 = 0
TF(‘good’) = 0/8 = 0
```

Term	Review 1	Review 2	Review 3	TF (Review 1)	TF (Review 2)	TF (Review 3)
This	1	1	1	1/7	1/8	1/6
movie	1	1	1	1/7	1/8	1/6
is	1	2	1	1/7	1/4	1/6
very	1	0	0	1/7	0	0
scary	1	1	0	1/7	1/8	0
and	1	1	1	1/7	1/8	1/6
long	1	0	0	1/7	0	0
not	0	1	0	0	1/8	0
slow	0	1	0	0	1/8	0
spooky	0	0	1	0	0	1/6
good	0	0	1	0	0	1/6

2. Inverse Document Frequency :-

- The inverse document frequency is a measure of whether a term is rare or frequent across the documents in the entire corpus. It highlights those words which occur in very few documents across the corpus, or in simple language, the words that are rare have high IDF score.

```
                        Formula - IDF =  log10(number of sentence / number of se
    ntence containing the word)
```

```
We can calculate the IDF values for the all the words in Review 2:

                                  Review 2: This movie is not scary and is slow

IDF('this') =  log(number of sentence/number of sentence containing the word 'this') =
log(3/3) = log(1) = 0


Similarly,

IDF('movie', ) = log(3/3) = 0
IDF('is') = log(3/3) = 0
IDF('not') = log(3/1) = log(3) = 0.48
IDF('scary') = log(3/2) = 0.18
IDF('and') = log(3/3) = 0
IDF('slow') = log(3/1) = 0.48
```

Term	Review 1	Review 2	Review 3	IDF
This	1	1	1	0.00
movie	1	1	1	0.00
is	1	2	1	0.00
very	1	0	0	0.48
scary	1	1	0	0.18
and	1	1	1	0.00
long	1	0	0	0.48
not	0	1	0	0.48
slow	0	1	0	0.48
spooky	0	0	1	0.48
good	0	0	1	0.48

Hence, we see that words like “is”, “this”, “and”, etc., are reduced to 0 and have little importance; while words like “scary”, “long”, “good”, etc. are words with more importance and thus have a higher value.

We can now compute the TF-IDF score for each word in the corpus. Words with a higher score are more important, and those with a lower score are less important:

$$(tf\_idf)_{t,d} = tf_{t,d} * idf_t$$

We can now calculate the TF-IDF score for every word in Review 2:

Review 2: This movie is not scary and is slow

TF-IDF(‘this’, Review 2) = TF(‘this’, Review 2) * IDF(‘this’) = 1/8 * 0 = 0

Similarly,

TF-IDF(‘movie’, Review 2) = 1/8 * 0 = 0
TF-IDF(‘is’, Review 2) = 1/4 * 0 = 0
TF-IDF(‘not’, Review 2) = 1/8 * 0.48 = 0.06
TF-IDF(‘scary’, Review 2) = 1/8 * 0.18 = 0.023
TF-IDF(‘and’, Review 2) = 1/8 * 0 = 0
TF-IDF(‘slow’, Review 2) = 1/8 * 0.48 = 0.06

Term	Review 1	Review 2	Review 3	IDF	TF-IDF (Review 1)	TF-IDF (Review 2)	TF-IDF (Review 3)
This	1	1	1	0.00	0.000	0.000	0.000
movie	1	1	1	0.00	0.000	0.000	0.000
is	1	2	1	0.00	0.000	0.000	0.000
very	1	0	0	0.48	0.068	0.000	0.000
scary	1	1	0	0.18	0.025	0.022	0.000
and	1	1	1	0.00	0.000	0.000	0.000
long	1	0	0	0.48	0.068	0.000	0.000
not	0	1	0	0.48	0.000	0.060	0.000
slow	0	1	0	0.48	0.000	0.060	0.000
spooky	0	0	1	0.48	0.000	0.000	0.080
good	0	0	1	0.48	0.000	0.000	0.080

- We have now obtained the TF-IDF scores for our vocabulary. TF-IDF also gives larger values for less frequent words and is high when both IDF and TF values are high i.e the word is rare in all the documents combined but frequent in a single document.

In [3]:

```
import nltk

paragraph =  """  This movie is very scary and long.
                 This movie is not scary and is slow.
                 This movie is spooky and good.
                 """


# Cleaning the texts
import re
sentences = nltk.sent_tokenize(paragraph)
corpus = []
for i in range(len(sentences)):
    review = re.sub('[^a-zA-Z]', ' ', sentences[i])
    review = review.lower()
    review = review.split()
    review = ' '.join(review)
    corpus.append(review)


# Creating the TF-IDF model
from sklearn.feature_extraction.text import TfidfVectorizer
cv = TfidfVectorizer()
X = cv.fit_transform(corpus).toarray()
```

In [4]:

```
X
```

Out[4]:

```
array([[0.29628336, 0.        , 0.29628336, 0.50165133, 0.29628336,
        0.        , 0.38151877, 0.        , 0.        , 0.29628336,
        0.50165133],
       [0.26359985, 0.        , 0.5271997 , 0.        , 0.26359985,
        0.44631334, 0.3394328 , 0.44631334, 0.        , 0.26359985,
        0.        ],
       [0.32052772, 0.54270061, 0.32052772, 0.        , 0.32052772,
        0.        , 0.        , 0.        , 0.54270061, 0.32052772,
        0.        ]])
```

Let me summarize what we've covered in the article:

- Bag of Words just creates a set of vectors containing the count of word occurrences in the document (reviews), while the TF-IDF model contains information on the more important words and the less important ones as well.
- Bag of Words vectors are easy to interpret. However, TF-IDF usually performs better in machine learning models.
- While both Bag-of-Words and TF-IDF have been popular in their own regard, there still remained a void where understanding the context of words was concerned. Detecting the similarity between the words 'spooky' and 'scary', or translating our given documents into another language, requires a lot more information on the documents.
- This is where Word Embedding techniques such as Word2Vec, Continuous Bag of Words (CBOW), Skipgram, etc. come in.