An Introduction to Bag of Words (BoW) | What is Bag of Words?

Bag of words is a Natural Language Processing technique of text modelling. Through this blog, we will learn more about why Bag of Words is used, we will understand the concept with the help of an example, learn more about it's implementation in Python, and more.

Using Natural Language Processing, we make use of the text data available across the internet to generate insights for the business. In order to understand this huge amount of data and make insights from them, we need to make them usable. Natural language processing helps us to do so.

What is a Bag of Words in NLP?

Bag of words is a Natural Language Processing technique of text modelling. In technical terms, we can say that it is a method of feature extraction with text data. This approach is a simple and flexible way of extracting features from documents.

A bag of words is a representation of text that describes the occurrence of words within a document. We just keep track of word counts and disregard the grammatical details and the word order. It is called a "bag" of words because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document.

Why is the Bag-of-Words algorithm used?

So, why bag-of-words, what is wrong with the simple and easy text?

One of the biggest problems with text is that it is messy and unstructured, and machine learning algorithms prefer structured, well defined fixed-length inputs and by using the Bag-of-Words technique we can convert variable-length texts into a fixed-length **vector.**

Also, at a much granular level, the machine learning models work with numerical data rather than textual data. So to be more specific, by using the bag-of-words (BoW) technique, we convert a text into its equivalent vector of numbers.

Understanding Bag of Words with an example

Let us see an example of how the bag of words technique converts text into vectors

Example(1) without preprocessing:

Sentence 1: "Welcome to Great Learning, Now start learning"

Sentence 2: "Learning is a good practice"

Sentence 1	Sentence 2
Welcome	Learning
to	is
Great	a
Learning	good
,	practice
Now	
start	
learning	

Step 1: Go through all the words in the above text and make a list of all of the words in our model vocabulary.

- Welcome
- To
- Great
- Learning
- ,
- Now
- start
- learning
- is
- a
- good
- practice

Note that the words 'Learning' and ' learning' are not the same here because of the difference in their cases and hence are repeated. Also, note that a comma ' , ' is also taken in the list.

Because we know the vocabulary has 12 words, we can use a fixed-length document-representation of 12, with one position in the vector to score each word.

The scoring method we use here is to count the presence of each word and mark 0 for absence. This scoring method is used more generally.

The scoring of sentence 1 would look as follows:

Word	Frequency
Welcome	1
to	1
Great	1
Learning	1
,	1
Now	1
start	1
learning	1
is	0

a	0
good	0
practice	0

Writing the above frequencies in the vector

Sentence 1 → **[1,1,1,1,1,1,1,1,0,0,0]**

Now for sentence 2, the scoring would like

Word	**Frequency**
Welcome	0
to	0
Great	0
Learning	1
,	0
Now	0
start	0
learning	0
is	1
a	1
good	1
practice	1

Similarly, writing the above frequencies in the vector form

Sentence 2 → **[0,0,0,0,0,0,0,1,1,1,1,1]**

Sentence	Welcome	to	Great	Learning	,	Now	start	learning	is	a	good	practice
Sentence1	1	1	1	1	1	1	1	1	0	0	0	0
Sentence2	0	0	0	0	0	0	0	1	1	1	1	1

But is this the best way to perform a bag of words. The above example was not the best example of how to use a bag of words. The words Learning and learning, although having the same meaning are taken twice. Also, a comma ',' which does not convey any information is also included in the vocabulary.

Let us make some changes and see how we can use 'bag of words in a more effective way.

Example(2) with preprocessing:

Sentence 1: "Welcome to Great Learning, Now start learning"

Sentence 2: "Learning is a good practice"

Step 1: Convert the above sentences in lower case as the case of the word does not hold any information.

Step 2: Remove special characters and stopwords from the text. Stopwords are the words that do not contain much information about text like 'is', 'a','the and many more'.

After applying the above steps, the sentences are changed to

Sentence 1: "welcome great learning now start learning"

Sentence 2: "learning good practice"

Although the above sentences do not make much sense the maximum information is contained in these words only.

Step 3: Go through all the words in the above text and make a list of all of the words in our model vocabulary.

- welcome
- great
- learning
- now
- start
- good
- practice

Now as the vocabulary has only 7 words, we can use a fixed-length document-representation of 7, with one position in the vector to score each word.

The scoring method we use here is the same as used in the previous example. For sentence 1, the count of words is as follow:

Word	**Frequency**
welcome	1
great	1
learning	2
now	1
start	1
good	0

practice	0

Writing the above frequencies in the vector

Sentence 1 → **[1,1,2,1,1,0,0]**

Now for sentence 2, the scoring would be like

Word	**Frequency**
welcome	0
great	0
learning	1
now	0
start	0
good	1
practice	1

Similarly, writing the above frequencies in the vector form

Sentence 2 → **[0,0,1,0,0,1,1]**

Sentence	welcome	great	learning	now	start	good	practice
Sentence1	1	1	2	1	1	0	0
Sentence2	0	0	1	0	0	1	1

The approach used in example two is the one that is generally used in the Bag-of-Words technique, the reason being that the datasets used in Machine learning are tremendously large and can contain vocabulary of a few thousand or even millions of words. Hence, preprocessing the text before using bag-of-words is a better way to go.

There are various preprocessing steps that can increase the performance of Bag-of-Words. Some of them are explained in great detail in this blog.

In the examples above we use all the words from vocabulary to form a vector, which is neither a practical way nor the best way to implement the BoW model. In practice, only a few words from the vocabulary, more preferably most common words are used to form the vector.

Implementing Bag of Words Algorithm with Python

In this section, we are going to implement a bag of words algorithm with Python. Also, this is a very basic implementation to understand how bag of words algorithm work, so I would not recommend using this in your project, instead use the method described in the next section.

```
def vectorize(tokens):
    ''' This function takes list of words in a sentence as input
    and returns a vector of size of filtered_vocab.It puts 0 if the
    word is not present in tokens and count of token if present.'''
    vector=[]
    for w in filtered_vocab:
        vector.append(tokens.count(w))
    return vector
def unique(sequence):
    '''This functions returns a list in which the order remains
    same and no item repeats.Using the set() function does not
    preserve the original ordering,so i didnt use that instead'''
    seen = set()
    return [x for x in sequence if not (x in seen or seen.add(x))]
#create a list of stopwords.You can import stopwords from nltk too
stopwords=["to","is","a"]
#list of special characters.You can use regular expressions too
special_char=[",",":"," ",";",".","?"]
#Write the sentences in the corpus,in our case, just two
string1="Welcome to Great Learning , Now start learning"
string2="Learning is a good practice"
#convert them to lower case
string1=string1.lower()
string2=string2.lower()
#split the sentences into tokens
tokens1=string1.split()
tokens2=string2.split()
print(tokens1)
print(tokens2)
#create a vocabulary list
vocab=unique(tokens1+tokens2)
print(vocab)
#filter the vocabulary list
filtered_vocab=[]
for w in vocab:
    if w not in stopwords and w not in special_char:
        filtered_vocab.append(w)
print(filtered_vocab)
#convert sentences into vectords
vector1=vectorize(tokens1)
print(vector1)
vector2=vectorize(tokens2)
print(vector2)
```

Output:

```
['welcome', 'to', 'great', 'learning', ',', 'now', 'start', 'learning']
['learning', 'is', 'a', 'good', 'practice']
['welcome', 'to', 'great', 'learning', ',', 'now', 'start', 'is', 'a', 'good', 'practice']
['welcome', 'great', 'learning', 'now', 'start', 'good', 'practice']
[1, 1, 2, 1, 1, 0, 0]
[0, 0, 1, 0, 0, 1, 1]
```

Create a Bag of Words Model with Sklearn

We can use the **CountVectorizer()** function from the Sk-learn library to easily implement the above BoW model using Python.

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

sentence_1="This is a good job.I will not miss it for anything"
sentence_2="This is not good at all"



CountVec = CountVectorizer(ngram_range=(1,1), # to use bigrams ngram_range=(2,2)
                           stop_words='english')
#transform
Count_data = CountVec.fit_transform([sentence_1,sentence_2])

#create dataframe
cv_dataframe=pd.DataFrame(Count_data.toarray(),columns=CountVec.get_feature_names())
print(cv_dataframe)
```

Output:

What are N-Grams?

Again same questions, what are n-grams and why do we use them? Let us understand this with an example below-

Sentence 1: “This is a good job. I will not miss it for anything”

Sentence 2: ”This is not good at all”

For this example, let us take the vocabulary of 5 words only. The five words being-

- good
- job
- miss
- not
- all

So, the respective vectors for these sentences are:

“This is a good job. I will not miss it for anything”=**[1,1,1,1,0]**

”This is not good at all”=**[1,0,0,1,1]**

Can you guess what is the problem here? Sentence 2 is a negative sentence and sentence 1 is a positive sentence. Does this reflect in any way in the vectors above? Not at all. So how can we solve this problem? Here come the N-grams to our rescue.

An N-gram is an N-token sequence of words: a 2-gram (more commonly called a bigram) is a two-word sequence of words like “really good”, “not good”, or “your homework”, and a 3-gram (more commonly called a trigram) is a three-word sequence of words like “not at all”, or “turn off light”.

For example, the bigrams in the first line of text in the previous section: “This is not good at all” are as follows:

- “This is”
- “is not”
- “not good”
- “good at”
- “at all”

Now if instead of using just words in the above example, we use bigrams (Bag-of-bigrams) as shown above. The model can differentiate between sentence 1 and sentence 2. So, using bi-grams makes tokens more understandable (for example, “HSR Layout”, in Bengaluru, is more informative than “HSR” and “layout”)

So we can conclude that a bag-of-bigrams representation is much more powerful than bag-of-words, and in many cases proves very hard to beat.

What is Tf-Idf (term frequency-inverse document frequency)?

The scoring method being used above takes the count of each word and represents the word in the vector by the number of counts of that particular word. What does a word having high word count signify?

Does this mean that the word is important in retrieving information about documents? The answer is NO. Let me explain, if a word occurs many times in a document but also along with many other documents in our dataset, maybe it is because this word is just a frequent word; not because it is relevant or meaningful.

One approach is to rescale the frequency of words by how often they appear in all documents so that the scores for frequent words like “the” that are also frequent across all documents are penalized. This approach is called term frequency-inverse document frequency or shortly known as Tf-Idf approach of scoring.TF-IDF is intended to reflect how relevant a term is in a given document. So how is Tf-Idf of a document in a dataset calculated?

TF-IDF for a word in a document is calculated by multiplying two different metrics:

The **term frequency (TF)** of a word in a document. There are several ways of calculating this frequency, with the simplest being a raw count of instances a word appears in a document. Then, there are other ways to adjust the frequency. For example, by dividing the raw count of instances of a word by either length of the document, or by the raw frequency of the most frequent word in the document. The formula to calculate Term-Frequency is

$$TF(i,j)=n(i,j)/\Sigma\ n(i,j)$$

Where,

$n(i,j)$ = number of times nth word occurred in a document

$\Sigma n(i,j)$ = total number of words in a document.

The **inverse document frequency(IDF)** of the word across a set of documents. This suggests how common or rare a word is in the entire document set. The closer it is to 0, the more common is the word. This metric can be calculated by taking the total number of documents, dividing it by the number of documents that contain a word, and calculating the logarithm.

So, if the word is very common and appears in many documents, this number will approach 0. Otherwise, it will approach 1.

Multiplying these two numbers results in the TF-IDF score of a word in a document. The higher the score, the more relevant that word is in that particular document.

To put it in mathematical terms, the TF-IDF score is calculated as follows:
`IDF=1+log(N/dN)`

Where
`N=Total number of documents in the dataset`
`dN=total number of documents in which nth word occur`

Also, note that the 1 added in the above formula is so that terms with zero IDF don't get suppressed entirely. This process is known as IDF smoothing.

The TF-IDF is obtained by
`TF-IDF=TF*IDF`

Does this seem too complicated? Don't worry, this can be attained with just a few lines of code and you don't even have to remember these scary formulas.

Feature Extraction with Tf-Idf vectorizer

We can use the **TfidfVectorizer()** function from the Sk-learn library to easily implement the above BoW(Tf-IDF), model.

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

sentence_1="This is a good job.I will not miss it for anything"
sentence_2="This is not good at all"



#without smooth IDF
print("Without Smoothing:")
#define tf-idf
tf_idf_vec = TfidfVectorizer(use_idf=True,
                         smooth_idf=False,
                         ngram_range=(1,1),stop_words='english') # to use only  bigrams ngram_range=(2,2)
#transform
tf_idf_data = tf_idf_vec.fit_transform([sentence_1,sentence_2])

#create dataframe
tf_idf_dataframe=pd.DataFrame(tf_idf_data.toarray(),columns=tf_idf_vec.get_feature_names())
print(tf_idf_dataframe)
print("\n")

#with smooth
tf_idf_vec_smooth = TfidfVectorizer(use_idf=True,
                         smooth_idf=True,
                         ngram_range=(1,1),stop_words='english')


tf_idf_data_smooth = tf_idf_vec_smooth.fit_transform([sentence_1,sentence_2])

print("With Smoothing:")
tf_idf_dataframe_smooth=pd.DataFrame(tf_idf_data_smooth.toarray(),columns=tf_idf_vec_smooth.get_feature_names())
print(tf_idf_dataframe_smooth)
```

Output:

Limitations of Bag-of-Words

Although Bag-of-Words is quite efficient and easy to implement, still there are some disadvantages to this technique which are given below:

1. The model ignores the location information of the word. The location information is a piece of very important information in the text. For example "today is off" and "Is today off", have the exact same vector representation in the BoW model.
2. Bag of word models doesn't respect the semantics of the word. For example, words 'soccer' and 'football' are often used in the same context. However, the vectors corresponding to these words are quite different in the bag of words model. The problem becomes more serious while modeling sentences. Ex: "Buy used cars" and "Purchase old automobiles" are represented by totally different vectors in the Bag-of-words model.
3. The range of vocabulary is a big issue faced by the Bag-of-Words model. For example, if the model comes across a new word it has not seen yet, rather we say a rare, but informative word like Biblioklept(means one who steals books). The BoW model will probably end up ignoring this word as this word has not been seen by the model yet.

This brings us to the end of this article where we have learned about Bag of words and its implementation with Sk-learn. If you wish to learn more about NLP, take up the Introduction to Natural Language Processing Free Online Course offered by Great Learning Academy and upskill today.

Further Reading