Word Embeddings:

- Bag of words
- TF-IDF
- Word2vec
- Glove embedding
- Fastext
- ELMO (Embeddings for Language models)

Bag of words:

The bag of words method is simple to understand and easy to implement. This method is mostly used in language modeling and text classification tasks. The concept behind this method is straightforward. In this method, we will represent sentences into vectors with the frequency of words that are occurring in those sentences.

Bag of words approach

n this approach we perform two operations.

1. Tokenization
2. Vectors Creation

Tokenization

The process of dividing each sentence into **words** or smaller parts. Here each word or symbol is called a **token**. After tokenization we will take unique words from the corpus. Here **corpus** means the tokens we have from all the documents we are considering for the bag of words creation.

Create vectors for each sentence

Here the size of the vector is equal to the number of unique words of the corpus. For each sentence we will fill each position of a vector with corresponding word frequency in a particular sentence.

Let's understand this with an example

1. This pasta is very tasty and affordable.
2. This pasta is not tasty and is affordable.
3. This pasta is very very delicious.

These **3 sentences** are example sentences, our first step is to perform tokenization. Before tokenization we have to convert all sentences to lowercase letters or uppercase letters for normalization, we will convert all the words in the sentences to lowercase.

Output of sentences after converting to lowercase

- this pasta is very tasty and affordable.
- this pasta is not tasty and is affordable.
- this pasta is very very delicious.

Now we will perform **tokenization**.

Dividing sentences into words and creating a list with all unique words and also in **alphabetical** order.

We will get the below output after the tokenization step.

[*"and", "affordable.", "delicious.", "is", "not", "pasta", "tasty", "this", "very"*]

Now what is our next step?

Creating vectors for each sentence with frequency of words. This is called a **sparse matrix**. Below is the sparse matrix of example sentences.

We can see in the above figure, every sentence converting into vectors. We can also find sentence similarities after converting sentences to vectors.

How can we find similarities ? Just calculating distance between any two vectors of sentences by using any distance measure method for example Euclidean Distance

In the above example we are just taking each word as a feature, another name for this is 1-gram representence, we can also take bigram words , tri-Gram words etc .

Examples for Bi-Gram word representation of the first sentence as below.

- this, pasta
- pasta, is
- is, very
- very, tasty
- tasty, and
- and affordable

Like this we can take more tri-gram words and n-gram words etc, here n is the number of words to split. But we can not get any semantic meaning or relation between words from the bag of words technique.

In Bag of word representation we have more zeros in the sparse matrices. The size of the matrix will be increased based on the total number of words in the corpus. In real world applications corpus will contain thousands of words. So we need more resources to build analytics models with this type of technique for large datasets. This drawback will be overcome in the next word embedding techniques.

TF :

The full form of TF is **Term Frequency** (TF). In TF , we are giving some scoring for each word or token based on the frequency of that word. The frequency of a word is dependent on the length of the document. Means in large size of document a word occurs more than a small or medium size of the documents.

So to overcome this problem we will divide the **frequency of a word** with the length of the document (total number of words) to normalize.By using this technique also, we are creating a sparse matrix with frequency of every word.

Formula to calculate Term Frequency (TF).

TF = no. of times term occurrences in a document / total number of words in a document

IDF

The full form of IDF is **Inverse Document Frequency**. Here also we are assigning a score value to a word , this scoring value explains how a word is rare across all documents. Rarer words have more IDF score.

Formula to calculate Inverse Document Frequency (IDF) :-

IDF = log base e (total number of documents / number of documents which are having term)

Formula to calculate complete TF-IDF value is

TF - IDF = TF * IDF

TF-IDF value will be increased based on frequency of the word in a document. Like Bag of Words in this technique also we can not get any semantic meaning for words.

But this technique is mostly used for document classification and also successfully used by search engines like Google, as a ranking factor for content.

Example sentences :-

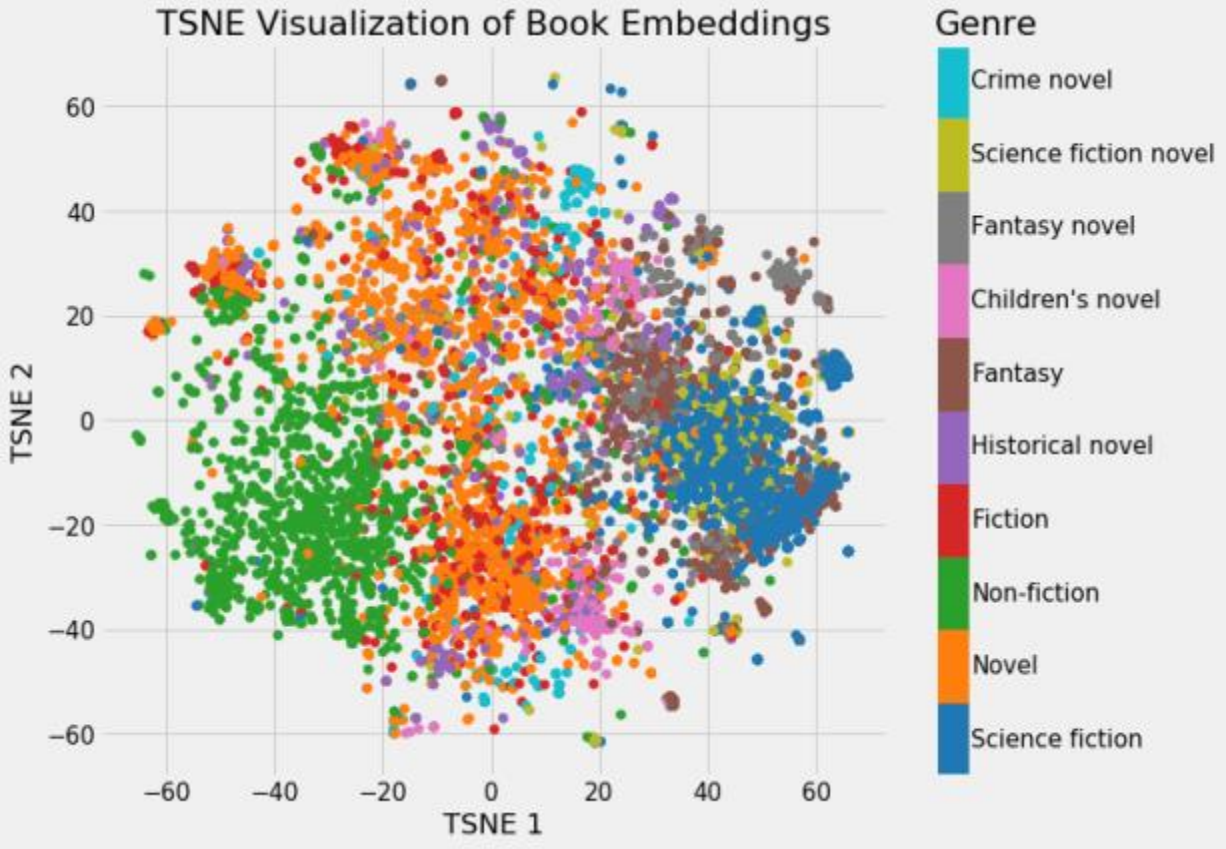
- A: This pasta is very tasty and affordable.
- B: This pasta is not tasty and is affordable.
- C: This pasta is very very delicious.

Let's consider each sentence as a document. Here also our first task is tokenization (dividing sentences into words or tokens) and then taking unique words.

Word	TF				TF * IDF		
	A	B	C		A	B	C
this	1/7	1/8	1/6	log(3/3)=0	0	0	0
pasta	1/7	1/8	1/6	log(3/3)=0	0	0	0
is	1/7	2/8	1/6	log(3/3)=0	0	0	0
Very	1/7	0	2/6	log(3/2)=0.176	0.025	0	0.058
tasty	1/7	1/8	0	log(3/2)=0.176	0.025	0.022	0
and	1/7	1/8	0	log(3/2)=0.176	0.025	0.022	0
affordable	1/7	1/8	0	log(3/2)=0.176	0.025	0.022	0
not	0	1/8	0	log(3/1)=0.477	0	0.0596	0
delicious	0	0	1/6	log(3/1)=0.477	0	0	0.079

From the above table we can observe rarer words have more score than common words.That shows us the significance of the words in our corpus.

Word2vec :



The Word2Vec model is used for learning vector representations of words called “word embeddings”. Did you observe that we didn’t get any semantic meaning from words of corpus by using previous methods? But for most of the applications of NLP tasks like sentiment classification, sarcasm detection etc require semantic meaning of a word and semantic relationships of a word with other words.

So can we get semantic meaning from words ?

Yeah exactly you got the answer , the answer is by using word2vec technique we will get what we want.

Word embeddings have a capability of capturing semantic and syntactic relationships between words and also the context of words in a document. Word2vec is the technique to implement word embeddings.

Every word in a sentence is dependent on another word or other words.If you want to find similarities and relations between words ,we have to capture word dependencies.

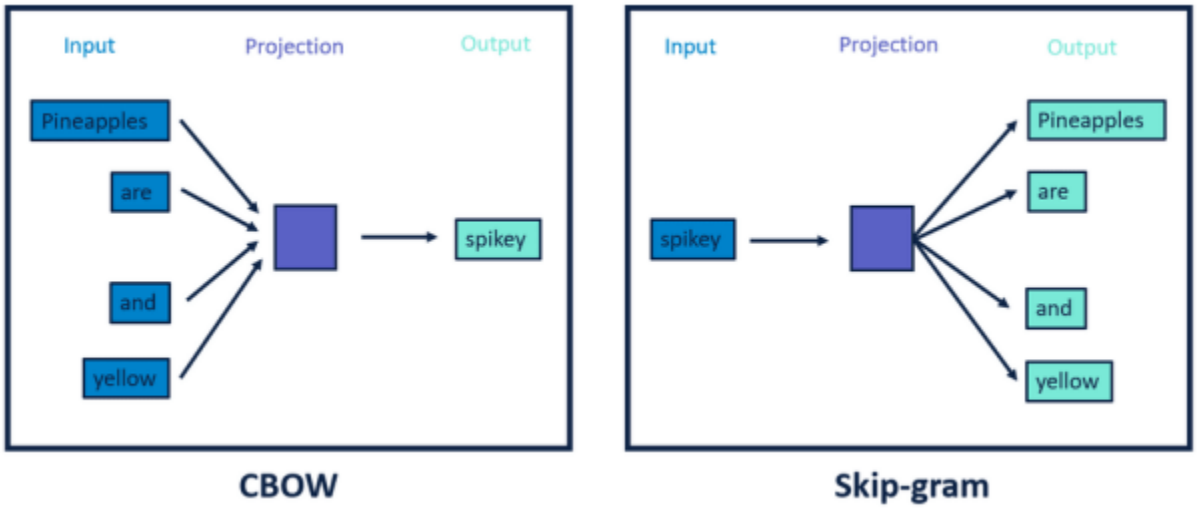
By using **Bag-of-words** and **TF-IDF** techniques we can not capture the meaning or relation of the words from vectors. Word2vec constructs such vectors called **embeddings**.

Word2vec model takes input as a **large size of corpus** and produces output to vector space. This vector space size may be in hundred of dimensionality. Each word vector will be placed on this vector space.

In vector space whatever words share context commonly in a corpus that are closer to each other. Word vector having positions of corresponding words in a vector space.

The Word2vec method learns all those types of relationships of words while building a model. For this purpose word2vec uses 2 types of methods. There are

1. Skip-gram
2. CBOW (Continuous Bag of Words)



Here one more thing we have to discuss that is window size. Did you remember the Bag-Of-words technique we discussed about 1-gram or uni-gram, bigram ,trigram ….n-gram representation of text ?

This method also follows the same technique. But here it is called **window size**.

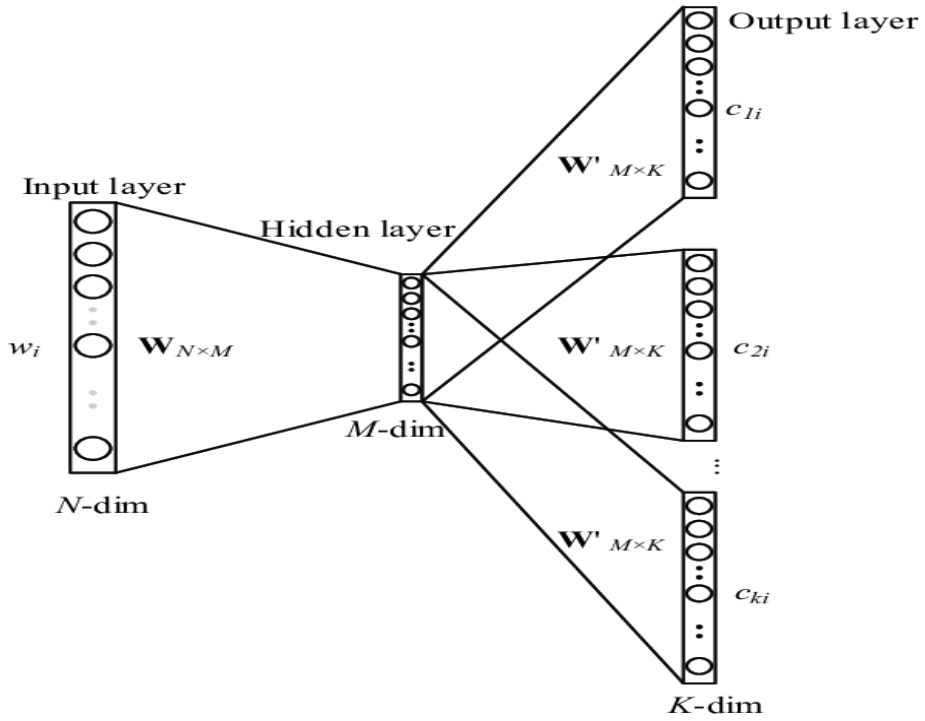
The Word2vec model will capture relationships of words with the help of window size by using skip-gram and CBow methods.

What is the difference between these 2 methods ? Do you want to know ?

That is a really simple technique. Before going to discuss these techniques , we have to know one more thing , why are we taking windows in this technique? Just to know the center word and context of the center word. (I have to add few words here like we can not use whole sentence)

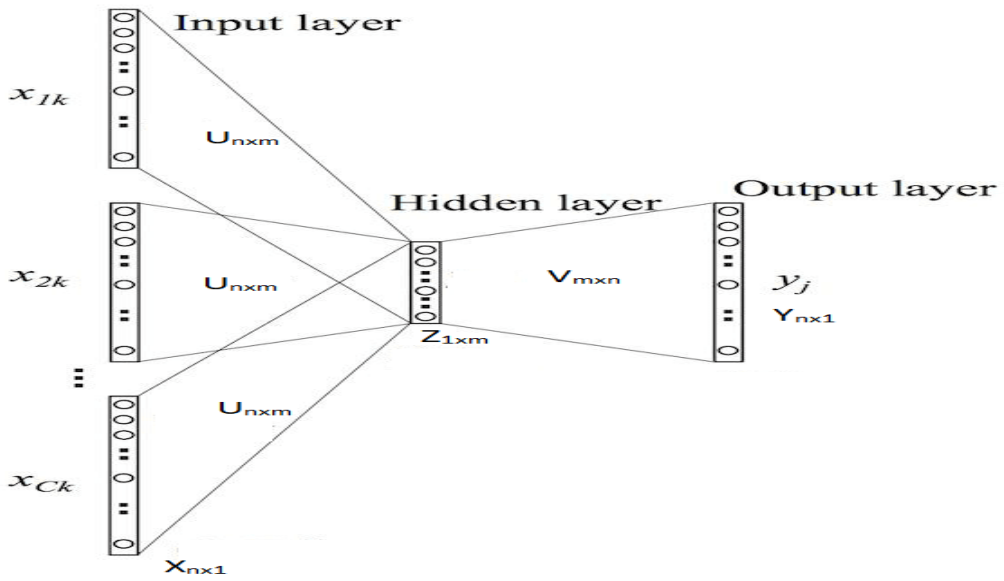
Skip-Gram

In this method , take the center word from the **window size** words as an input and context words (neighbour words) as outputs. Word2vec models predict the context words of a center word using skip-gram method. Skip-gram works well with a small dataset and identifies rare words really well.



Continuous Bag-of-words

CBow is just a reverse method of the skip gram method. Here we are taking context words as input and predicting the center word within the window. Another difference from skip gram method is, It was working **faster and better** representations for most frequency words.



difference between Skip gram & CBow

Skip gram:

- In this input is centre word and output is context words (neighbour words).
- Works well with small datasets.
- Skip-gram identifies rarer words better.

CBow:

- In this context or neighbor words are input and output is the center word.
- Works good with large datasets.
- Better representation for frequent words than rarer.

GloVe :

GloVe, an abbreviation of "global vectors," is a word embedding technique that has been developed by Stanford. It is an unsupervised learning algorithm that builds on Word2Vec. While Word2Vec is quite successful in generating word embeddings, the issue with it is that is it has a small window through which it focuses on local words

and local context to predict words. This means that it is unable to learn from the frequency of words present globally, that is, in the entire corpus. GloVe, as mentioned in its name, looks at all the words present in a corpus.

While Word2Vec is a predictive model as it learns vectors to improve its predictive abilities, GloVe is a count-based model. What this means is that GloVe learns its vectors by performing dimensionality reduction on a co-occurrence counts matrix. The connections that GloVe is able to make are along the lines of this:

king – man + woman = queen

This means it's able to understand that "king" and "queen" share a relationship that is similar to that between "man" and "woman".

These are complicated terms, so let's understand them one by one. All of these concepts come from statistics and linear algebra, so if you already know what's going on, you can skip to the activity!

When dealing with a corpus, there exist algorithms to construct matrices based on term frequencies. Basically, these matrices contain words that occur in a document as rows, and the columns are either paragraphs or separate documents. The elements of the matrices represent the frequency with which the words occur in the documents. Naturally, with a large corpus, this matrix will be huge. Processing such a large matrix will take a lot of time and memory, thus we perform dimensionality reduction. This is the process of reducing the size of the matrix so it is possible to perform further operations on it.

In the case of GloVe, the matrix is known as a co-occurrence counts matrix, which contains information on how many times a word has occurred in a particular context in a corpus. The rows are the words and the columns are the contexts. This matrix is then factorized in order to reduce the dimensions, and the new matrix has a vector representation for each word.

ELMO:

https://www.analyticsvidhya.com/blog/2019/03/learn-to-use-elmo-to-extract-features-from-text/#:~:text=Embeddings%20from%20Language%20Models%20(ELMo,NLP%20framework%20developed%20by%20AllenNLP.&text=Note%3A%20This%20article%20assumes%20you,word%20embeddings%20and%20LSTM%20architecture.

Fasttext:

https://towardsdatascience.com/word-embedding-with-word2vec-and-fasttext-a209c1d3e12c