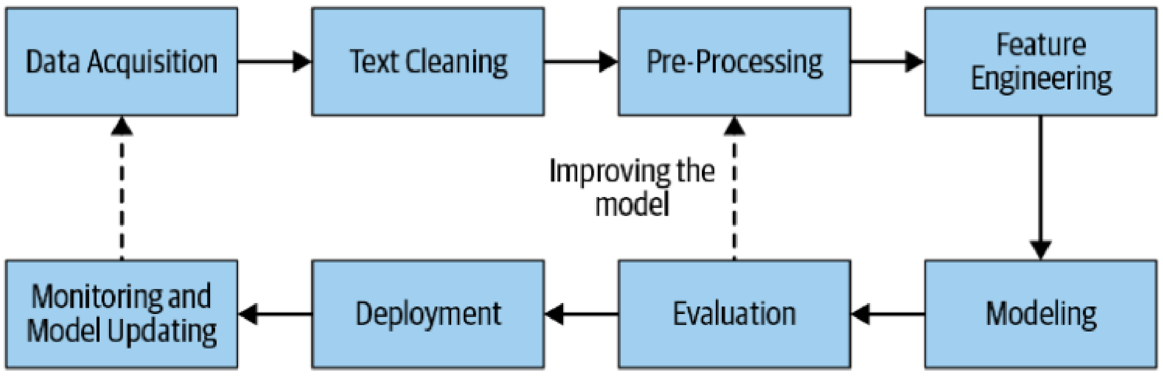
Module - 2(Pre-processing and Representation Models)

NLP System Pipeline--Steps--Data Acquisition, Text Extraction and Clean-up, Pre-processing, Feature Engineering, Modelling, Evaluation, Post-Modelling Phases Text Representation--Vector Space Models--Basic Vectorization Approaches--One-Hot Encoding, Bag of Words, Bag of N-Grams TF-IDF; Distributed Representations-- Word Embeddings, Doc2Vec.

NLP System Pipeline--Steps

NLP Pipeline is a set of steps followed to build an end to end NLP software.

Before we started we have to remember this things pipeline is not universal, Deep Learning Pipelines are slightly different, and Pipeline is non-linear.



1. Data Acquisition

In the data acquisition step, these three possible situations happen.

1. Data Available Already

A. Data available on local Machine – If data is available on the local machine then we can directly go to the next step i.e. Data Preprocessing.

B. Data available in Database – If data is available in the database then we have to communicate to the data engineering team. Then Data Engineering team gives data from the database. data engineers create a data warehouse.

C. Less Data Available – If data is available but it is not enough. Then we can do data Augmentation. Data augmentation is to making fake data using existing data. here we use Synonyms, Bigram flip, Back translate, or adding additional noise.

2. Data is not available in our company but is available outside. Then we can use this approach.

A. Public Dataset – If a public dataset is available for our problem statement.
B. Web Scrapping – Scrapping competitor data using beautiful soup or other libraries
C. API – Using different APIs. eg. Rapid API.

3. Data Not Available – Here we have to survey to collect data. and then manually give a label to the data.

2. Text Preprocessing

So Our data collection step is done but we can not use this data for model building. we have to do text preprocessing.

This text preprocessing I have already explained in my previous blog.
Steps –
1. Text Cleaning – In-text cleaning we do HTML tag removing, emoji handling, Spellingchecker,etc.
2. Basic Preprocessing — In basic preprocessing we do tokenization(word or sent tokenization, stop word removal, removing digit, lower casing.)
3. Advance Preprocessing — In this step we do POS tagging, Parsing, and Coreference resolution.

3. Featured Engineering

Feature Engineering means converting text data to numerical data. but why it is required to convert text data to numerical data?. because our machine learning model doesn't understand text data then we have to do feature engineering. This step is also called Feature extraction from text.

In this step, we use multiple techniques to convert text to numerical vectors.

1. One Hot Encoder
2. Bag Of Word(BOW)
3. n-grams
4. Tf-Idf
5. Word2vec

4. Modelling/Model Building

In the modeling step, we try to make a model based on data. here also we can use multiple approaches to build the model based on the problem statement.

Approaches to building models

1.Heuristic approach Approach 2. Machine Learning Approach 3. Deep Learning Approach 4. Cloud API

Here comes one question, Which approach do we have to use? Right? then this is based on two things,

1. Amount of data

2. Nature of the problem.

If we have very less data then we cannot use ML or DL approach then we have to use the heuristic approach. but if we have a good amount of data then we can use a machine learning approach and if we have a large amount of data then we can use a deep learning approach.

second, based on the nature of the problem, we have to check which approach gives the best solution because if the nature of the problem changes all things get changed.

5. Model Evaluation

In the model evaluation, we can use two metrics Intrinsic evaluation and Extrinsic evaluation.

Intrinsic evaluation – In this evaluation, we use multiple metrics to check our model such as Accuracy, Recall, Confusion Metrics, Perplexity, etc.

Extrinsic evaluation — This evaluation is done after deployment. This is the business-centric approach.

6. Deployment

In the deployment step, we have to deploy our model on the cloud for the users. and users can use this model. deployment has three stages deployment, monitoring, and retraining or model update.

Three stages of deployment –
1. Deployment – model deploying on the cloud for users.

2. Monitoring – In the monitoring phase, we have to watch the model continuously. Here we have to create a dashboard to show evaluation metrics.
3. Update- Retrain the model on new data and again deploy.

- NLP pipeline is very important to building any kind of NLP problem.
- Text preprocessing step is the most important step in the NLP pipeline.
- We can use multiple techniques for feature extraction such as a bag of words, Tf-idf, n-grams, and word2vec.

- In model evaluation, we have to build multiple models and select the best model based on evaluation metrics.
- Deployment has three stages deployment, monitoring, and retraining.

TEXT CLEANING

Most Common Methods for Cleaning the Data

- Removing HTML tags
- Removing emojis
- Removing & Finding URL
- Removing & Finding Email id
- Standardizing and Spell Check
- Chat word correction
- Remove the frequent words
- Removing the less frequent words

Clean text is human language rearranged into a format that machine models can understand. Text cleaning can be performed using simple Python code that eliminates stopwords, removes unicode words, and simplifies complex words to their root form.

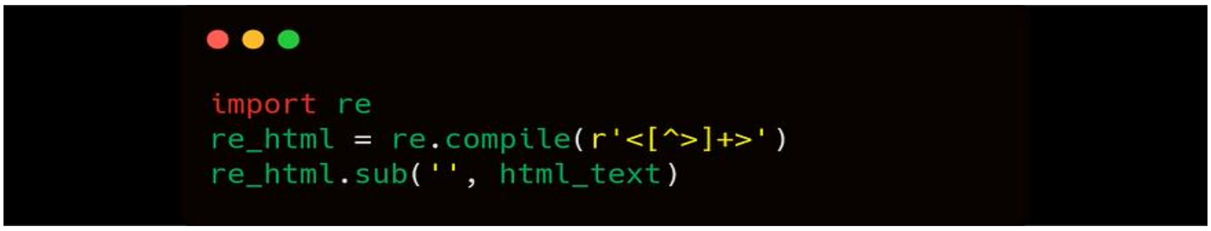
Removing HTML tags

Most of the times when you want to process a tonne of html files in your corpus,

1. Using Regex

Regular expressions are the most popular and powerful method for any of the complex string extraction process you want to carry out. regex can be easily employed in searching for string patters between HTML tags.

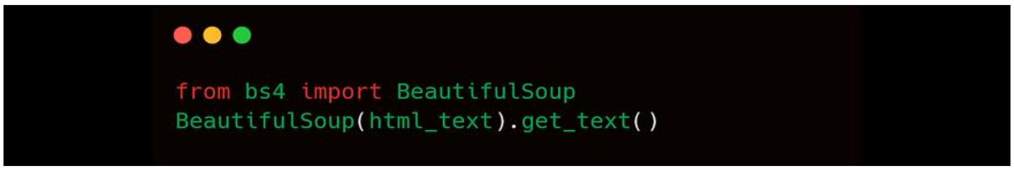
Eg: html_text = "<HTML><HEAD> **This is HEAD** <INSIDE> **The is inside tag** </INSIDE></HEAD> <BODY> **This is BODY** </BODY></HTML>" #the html text you want to process



```
import re
re_html = re.compile(r'<[^>]+>')
re_html.sub('', html_text)
```

The output :
This is HEAD The is inside tag This is BODY
2. Using Beautiful Soup

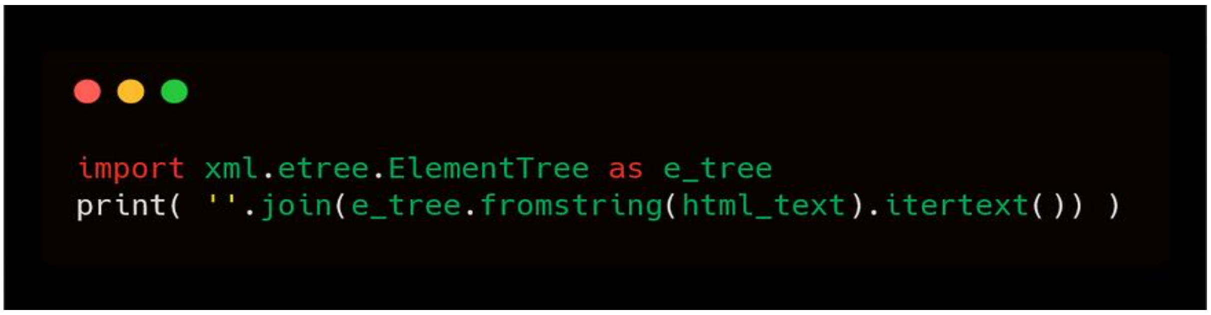
Beautiful Soup is a package widely used to scrape text from webpages. It has very powerful methods that can parse different DOM structures. Here we will use that to parse the HTML formatted text.



```
from bs4 import BeautifulSoup
BeautifulSoup(html_text).get_text()
```

3. Using XML's ElementTree

The itertext() method in XML's ElementTree module can be used to pull out any text as long as it conforms to an XML structure.



```
import xml.etree.ElementTree as e_tree
print( ''.join(e_tree.fromstring(html_text).itertext()) )
```

Removing emojis

Both **emoji** and **emoticon** convey emotional expression in a text message. The main difference between emoji and emoticon: an **emoji** is a small actual image that is used to express emotions or idea in text messages. where an **emoticon** is a facial expression representation using keyboard characters and punctuations. For example, ? is an emoji and **':)'** is an emoticon that represents a happy face.

```
emoji_pattern=re.compile(ur" " " [\U0001F600-\U0001F64F] # emoticons \
                             |\
                             [\U0001F300-\U0001F5FF] # symbols & pictographs\
                             |\
                             [\U0001F680-\U0001F6FF] # transport & map symbols\
                             |\
                             [\U0001F1E0-\U0001F1FF] # flags (iOS)\
                       " " ", re.VERBOSE)

emoji_pattern.sub('', word)
```

Standardizing and Spell Check

spelling correction is the process of correcting word's spelling for example "lisr" instead of "list". Word Lengthening is also a type of spelling mistake in which characters within a word are repeated wrongly for example "awwwwsome" instead of "awesome".

```
from autocorrect import Speller

spell = Speller(lang='en')

print(spell('caaaar'))
print(spell('mussage'))
print(spell('survice'))
print(spell('hte'))
```

PREPROCESSING

Data Preprocessing is the most essential step for any Machine Learning model. How well the raw data has been cleaned and preprocessed plays a major role in the performance of the model.

The various preprocessing steps that are involved are :

1. Tokenization
2. Lower Casing
3. Punctuation Mark Removal
4. Stop Word Removal
5. Stemming
6. Lemmatization

1.Tokenization

Tokenizationis the process of breaking up the paragraph into smaller units such as sentences or words. Each unit is then considered as an individual token. The fundamental principle of Tokenization is to try to understand the meaning of the text by analyzing the smaller units or tokens that constitute the paragraph.

Sentence Tokenize

Take a paragraph as input and tokenize it into its constituting sentences. The result is a list stored in the variable 'sentences'. It contains each sentence of the paragraph. The length of the list gives us the total number of sentences.

```
from nltk.tokenize sent_tokenize

sentences=sent_tokenize(text)

print(sentences)
```

Word Tokenize

Tokenize the paragraph into words. The result is a list called 'words', containing each word of the paragraph. The length of the list gives us the total number of words present in our paragraph.

```
from nltk.tokenize word_tokenize

words = nltk.word_tokenize(paragraph.lower())
print (words)
print (len(words))
```

2.Lower Casing

The same words written in different cases are considered as different entities by the computer. For example: '*Girl*' and '*girl*' are considered as two separate words by the computer even though they mean the same.

In order to resolve this issue, we must convert all the words to lower case. This provides uniformity in the text.

```
sentence = "This text is used to demonstrate Text Preprocessing in NLP."
sentence = sentence.lower()
print(sentence)
```

3. Punctuation Mark Removal

we can remove all the punctuation marks from our list of words by excluding any alphanumeric element. This can be easily done in this manner.

```
New_words= [word for word in words if word.isalnum()]
```

4.Stop word removal

Stop words are a collection of words that occur frequently in any language but do not add much meaning to the sentences. These are common words that are part of the grammar of any language. Every language has its own set of stop words. For example some of the English stop words are "the", "he", "him", "his", "her", "herself" etc.

```
from nltk.corpus import stopwords
nltk.download('stopwords')
WordSet = []
for word in new_words:
  if word not in set(stopwords.words("english")):
    WordSet.append(word)
print(WordSet)
```

5.Stemming

Stemming is the process of reduction of a word into its root or stem word. The word affixes are removed leaving behind only the root form or *lemma*.

For example: The words *"connecting", "connect", "connection", "connects"* are all reduced to the root form *"connect"*. The words *"studying", "studies", "study"* are all reduced to *"studi"*.

```
from nltk.stem import PorterStemmer
ps = PorterStemmer()
WordSetStem = []
for word in WordSet:
  WordSetStem.append(ps.stem(word))
print(WordSetStem)
```

6.Lemmatization

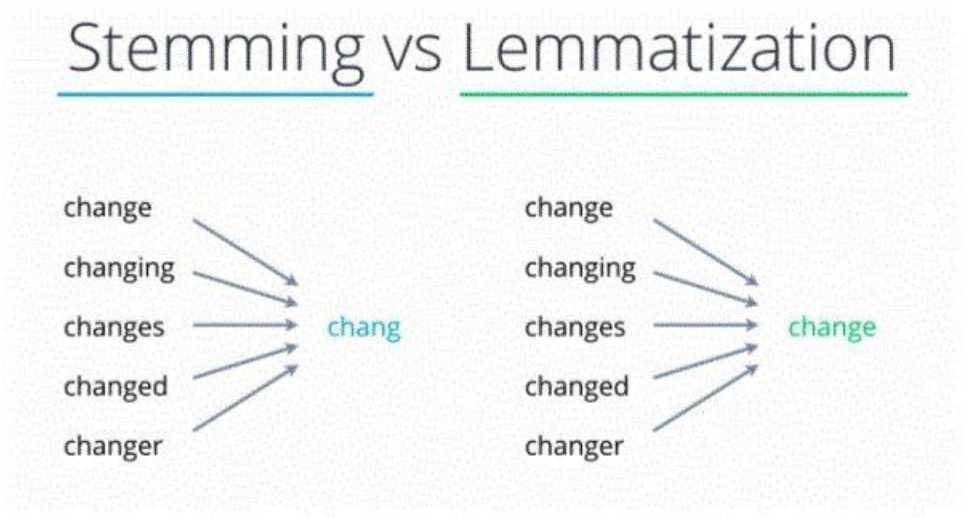
Stemming does not always result in words that are part of the language vocabulary. It often results in words that have no meaning to the users. In order to overcome this drawback, we shall use the concept of Lemmatization.

Lemmatization is a text normalization technique used in Natural Language Processing (NLP), that switches any kind of a word to its base root mode. Lemmatization is responsible for grouping different inflected forms of words into the root form, having the same meaning.

```
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
sentence="Hello Guru99, You have to build a very good site and I love visiting your site."
words = word_tokenize(sentence)
ps = PorterStemmer()
for w in words:
        rootWord=ps.stem(w)
        print(rootWord)
```

- Lemmatization and Stemming, both are used to generate root form of derived (inflected) words. However, lemma is an actual language word, whereas stem may not be an actual word.
- Lemmatization uses corpus for stop words and WordNet corpus to produce lemma. Moreover, parts-of-speech also had to be defined to obtain correct lemma.

For instance, stemming the word 'Caring' would return 'Car'. For instance, lemmatizing the word 'Caring' would return 'Care'.



Stemming is a process that stems or removes last few characters from a word, often leading to incorrect meanings and spelling. Lemmatization considers the context and converts the word to its meaningful base form, which is called Lemma.

<u>**TEXT REPRESENTATION**</u>

Text Representation is a way to convert text in its natural form to vector form .This is the step in an NLP pipeline after Text Pre-processing.

<u>Vector space models:</u>

Vector space models are **algebraic models that are often used to represent text**. Vector space models are to consider the relationship between data that are represented by vectors. It is popular in information retrieval systems but also useful for other purposes.

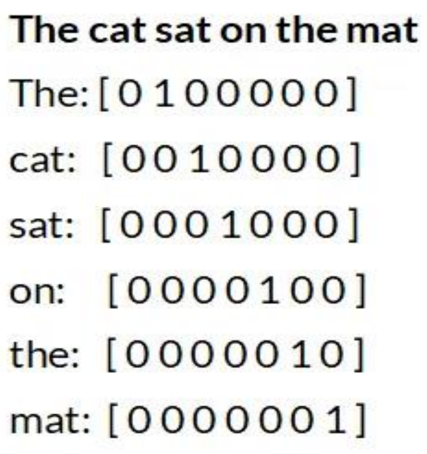
The Vector Space Model (VSM) is a mathematical framework used in Natural Language Processing (NLP) to represent text documents as numerical vectors. It is based on the principle that words in a document can be represented as a set of features, and each feature can be given a weight that represents its importance in the document.

In the VSM, a text document is represented as a vector of numbers, where each dimension of the vector represents a unique term or word in the document. The value of each dimension is the frequency or weight of the corresponding term in the document. This way, each document can be represented as a point in a high-dimensional space, where each dimension represents a different term. The steps involved in building a VSM are as follows:

1. Corpus Preparation: The first step is to prepare a corpus, which is a collection of text documents that will be used to create the VSM. The corpus should be cleaned by removing stop words, punctuations, and other irrelevant characters.
2. Tokenization: Next, the documents in the corpus are tokenized, which means splitting each document into individual words or terms. This is typically done using a tokenizer, which can handle various types of tokenization, such as word-based or character-based tokenization.
3. Vocabulary Creation: After tokenization, the next step is to create a vocabulary, which is a list of unique words or terms that occur in the corpus. The vocabulary is created by collecting all the unique tokens from all the documents in the corpus.
4. Document-Term Matrix: Once the vocabulary is created, a Document-Term Matrix (DTM) is constructed. In this matrix, each row corresponds to a document in the corpus, and each column corresponds to a term in the vocabulary. The cells of the matrix contain the frequency of each term in each document.
5. Term Weighting: The next step is to weight the terms in the DTM, which involves assigning a weight to each term based on its importance or relevance. Common term weighting methods include Term Frequency-Inverse Document Frequency (TF-IDF), which assigns higher weights to terms that occur frequently in a document but infrequently across the corpus.
6. Vector Representation: Finally, the documents in the corpus are represented as vectors in a high-dimensional space. Each document vector represents the frequency or weight of each term in the vocabulary. The resulting vectors can then be used for various NLP tasks such as document classification, information retrieval, and clustering.

1.One hot encoding

In one hot encoding, every word (even symbols) which are part of the given text data are written in the form of vectors, constituting only of 1 and 0 . So one hot vector is a vector whose elements are only 1 and 0. Each word is written or encoded as one hot vector, with each one hot vector being **unique**. This allows the word to be identified uniquely by its one hot vector.and vice versa, that is no two words will have same one hot vector representation.



The cat sat on the mat

The: [0100000]

cat: [0010000]

sat: [0001000]

on: [0000100]

the: [0000010]

mat: [0000001]

It's relatively straightforward to implement and enables us to apply machine-learning algorithms to data with categorical columns.

The problem is that it **increases dimensionality so training becomes slower and more complex**. It can also create sparse data since most entries in the new columns will be zero. Additionally, one-hot encoding takes more space but adds no new information since it only changes data representation.

2.Bag-of-words (BOW)

The bag-of-words model is a simplifying representation used in natural language processing and information retrieval (IR). In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity. The bag-of-words model is commonly used in methods of document classification where the (frequency of) occurrence of each word is used as a feature for training a classifier. By using the bag-of-words (BoW) technique, we convert a text into its equivalent vector of numbers.

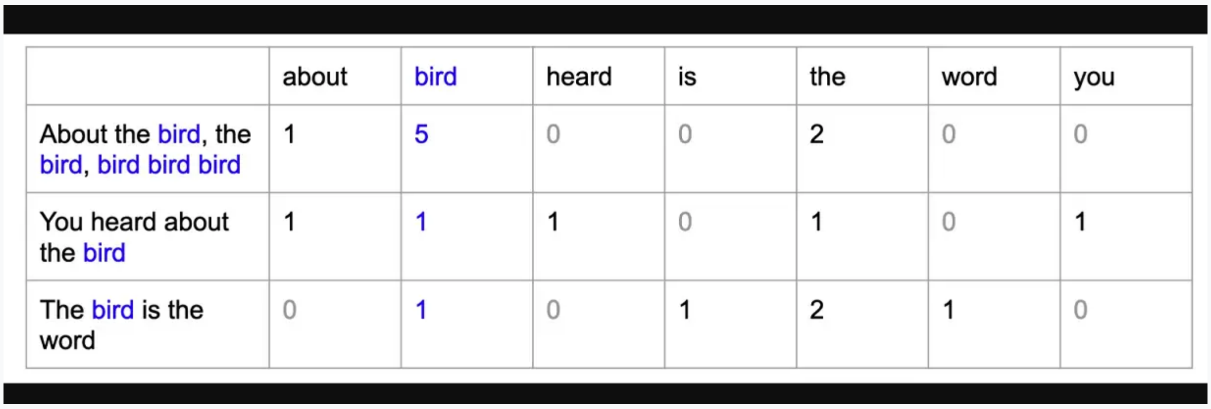
The **bag-of-words** (BOW) model is a representation that turns arbitrary text into **fixed-length vectors** by counting how many times each word appears. This process is often referred to as **vectorization**. where the (frequency of) occurrence of each word is used as a feature for training a classifier.

```
1) John likes to watch movies. Mary likes movies too.
BoW1={"John":1,"likes":2,"to":1,"watch":1,"movies":2,"Mary":1,"too":1};



(2) Mary also likes to watch football games.
```

```
BoW2={"Mary":1,"also":1,"likes":1,"to":1,"watch":1,"football":1,"games"
:1};
```



	about	bird	heard	is	the	word	you
About the bird, the bird, bird bird bird	1	5	0	0	2	0	0
You heard about the bird	1	1	1	0	1	0	1
The bird is the word	0	1	0	1	2	1	0

Disadvantages:

1)Consider deploying the Bag of Words method to generate vectors for large documents. The resultant vectors will be of large dimension and will contain far too many null values resulting in sparse vectors.

2) Resulting in sparse representations.

3)Bag of Words does a poor job in making sense of text data. For example, consider the two sentences: "I love playing football and hate cricket" and it's vice-versa "I love playing cricket and hate football". Bag of Words approach will result in similar vectorized representations although both sentences carry different meanings.

4)OOV(out of vocabulary)

3.**Bag-of-ngrams**

Continual word, symbol, or token sequences are known as **n-gram representations**. They are the adjacent groups of items in a document. In natural language processing (NLP) tasks, they are relevant when we deal with textual data.

n is a positive integer variable that can have values like 1, 2, 3, 4, and so on.

Depending on the value of n, n-grams have the following different types or categories:

1. Unigram
2. Bigram
3. Trigram
4. n-gram

Unigram

Unigrams are a type of n-gram where the value of `n` is `1`. Unigram means taking only one word or token at a time.

Example:

Text = “Educative is the best platform”

The unigram for the above text is as follows:

[“Educative”, “is”, “the”, “best”, “platform”]

Bigram

Bigrams are a type of n-gram where the value of `n` is `2`. Bigram means taking two words or tokens at a time.

Example:

Text = “Educative is the best platform”

The bigram for the above text is as follows:

[“Educative is”, “is the”, “the best”, “best platform”]

Trigram

Trigrams are a type of n-gram where the value of `n` is `3`. Trigram means taking three words or tokens at a time.

Example:

text = “Educative is the best platform”

The trigram for the above text is as follows:

[“Educative is the”, “is the best”, “the best platform”]

n-gram

n-grams can be defined for any given value of `n`.

Let us consider `n` to be `4`. This means taking fours words or tokens at a time.

Example:

text = “Educative is the best platform”

The 4-gram for the above text is as follows:

[“Educative is the best”, “is the best platform”]

Disadvantages

1. The length of the vocabulary is large, resulting in a large vector length of the word(increases dimensions)
2. The co-occurrence matrix is also a sparse matrix .
3. OOV
4. Doesn’t capture semantic meaning of the texts

4.TF-IDF

The TF-IDF algorithm is used to weigh a keyword in any content and assign importance to that keyword **based on the number of times it appears** in the document. More importantly, it checks how relevant the keyword is throughout the web, which is referred to as *corpus*.

For a term **t** in document **d**, the weight **Wt,d** of term t in document d is given by:

W(t,d) = TF(t,d)*log (N/DFt)

Where:

- TF(t,d) is the number of occurrences of t in document d.
- DFt is the number of documents containing the term t.
- N is the total number of documents in the corpus.

TF-IDF is scored between 0 and 1. The higher the numerical weight value, the rarer the term. The smaller the weight, the more common the term.

TF (term frequency) example

The TF (term frequency) of a word is the frequency of a word (i.e., number of times it appears) in a document. When you know TF, you’re able to see if you’re using a term too much or too little.

When a 100-word document contains the term “cat” 12 times, the TF for the word ‘cat’ is

TFcat = 12/100 i.e. 0.12

IDF (inverse document frequency) example

The IDF (inverse document frequency) of a word is the measure of how significant that term is in the whole corpus (a body of documents).

Let’s say the size of the corpus is 10,000,000 million documents. If we assume there are 0.3 million documents that contain the term “cat”, then the IDF (i.e. log {DF}) is given by the total number of documents (10,000,000) divided by the number of documents containing the term “cat” (300,000).

IDF (cat) = log (10,000,000/300,000) = 1.52

TF-IDF Calculation

Put the TF and IDF calculations together to get a TF IDF score.

∴ Wcat = (TF*IDF) cat = 0.12 * 1.52 = 0.182

A TF-IDF score of 0.182 is much closer to 0 than 1.

Advantages:
- Easy to compute
- You have some basic metric to extract the most descriptive terms in a document
- You can easily compute the similarity between 2 documents using it

Disadvantages:
- TF-IDF is based on the bag-of-words (BoW) model, therefore it does not capture position in text, semantics, co-occurrences in different documents, etc.
- For this reason, TF-IDF is only useful as a lexical level feature
- Cannot capture semantics (e.g. as compared to topic models, word embeddings)

Word Embedding

Word Embeddings in NLP is a technique where individual words are represented as real-valued vectors in a lower-dimensional space and captures inter-word semantics. Each word is represented by a real-valued vector with tens or hundreds of dimensions.

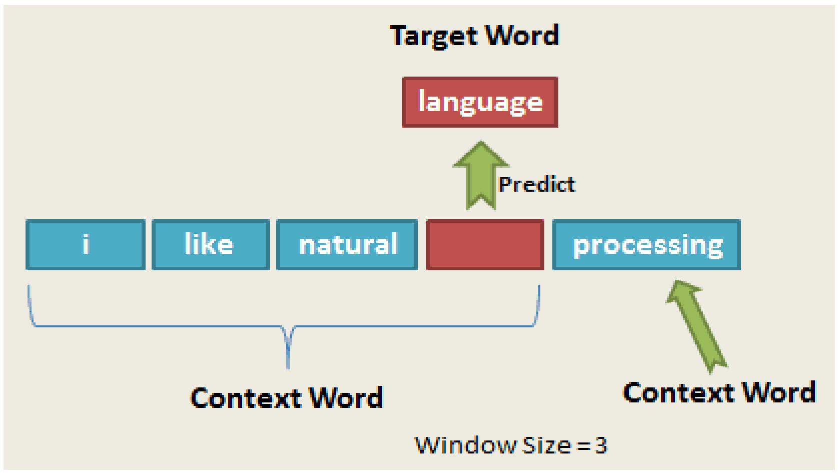
It allows words with similar meaning to have a similar representation. They can also approximate meaning. A word vector with 50 values can represent 50 unique features.

Word2vec is a common technique used in Natural Language Processing. In it, similar words have similar word embeddings; this means that they are close to each other in terms of cosine distance. **There are two main algorithms to obtain a Word2Vec implementation: Continous Bag of Words and Skip-Gram**. These algorithms use **neural network models** in order to obtain the word vectors.These models work using context. This means that the embedding is learned by looking at nearby words; if a group of words is always found close to the same words, they will end up having similar embeddings.

1) CBOW

CBOW or Continous bag of words is to use embeddings in order to train a neural network where the context is represented by multiple words for a given target words.

It attempts to guess the output (target word) from its neighboring words (context words). You can think of it like fill in the blank task, where you need to guess word in place of blank by observing nearby words.



CBOW is a variant of the word2vec model predicts the center word(target word) from (bag of) context words. So given all the words in the context window (excluding the middle one), CBOW would tell us the most likely the word at the center.

Better method for small amount of data

Eg: watch campus for data science

Window size=3

watch campus for data science

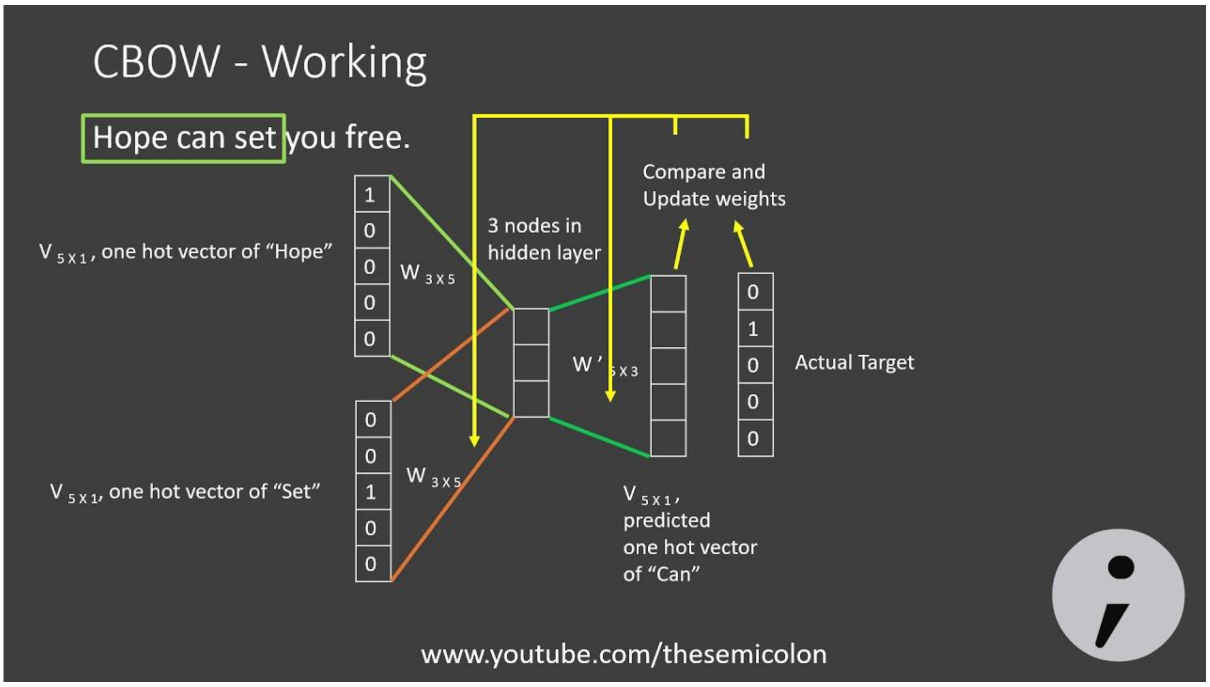
target

context word   context word

X   Y

Input		output
Watch	For	campusx
campusx	Data	for
for	Science	data

Train the neural mnetwork with this dataset

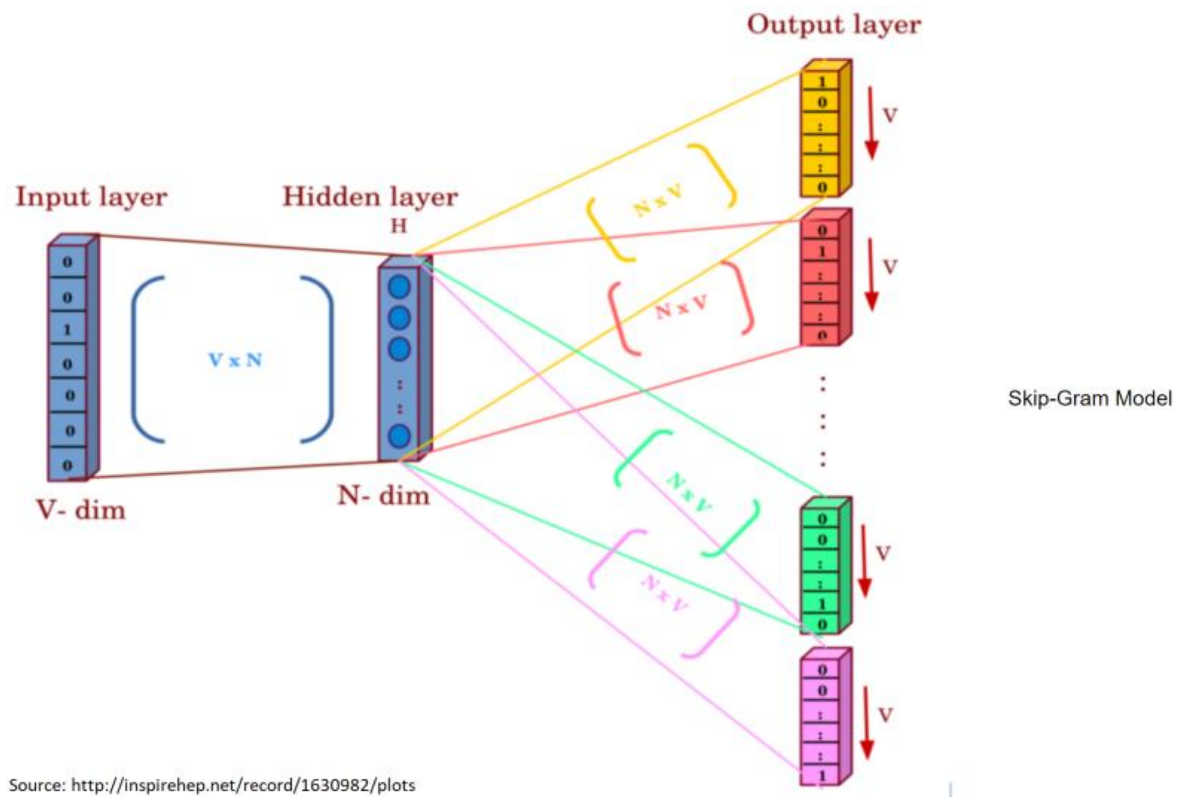

CBOW - Working
Hope can set you free.
V 5 x 1, one hot vector of "Hope"
V 5 x 1, one hot vector of "Set"
W 3 x 5
W 3 x 5
3 nodes in hidden layer
W ' 5 x 3
Compare and Update weights
Actual Target
V 5 x 1, predicted one hot vector of "Can"
www.youtube.com/thesemicolon

2.Skip-gram

Skip-gram is used to predict the context word for a given target word. It's reverse of CBOW algorithm._ Skip-gram is one of the unsupervised learning techniques used to find the most related words for a given word. Here, target word is input while context words are output. As there is more than one context word to be predicted which makes this problem difficult.

X Y

input	Output	
campusx	Watch	For
for	Campus	Data
data	For	science



1.The word w(t) is passed to the hidden layer from |v| neurons.

2. Hidden layer performs the dot product between weight vector W[|v|, N] and the input vector w(t). In this, we can conclude that the (t)th row of W[|v|, N] will be the output(H[1, N]).

3. Remember there is no activation function used at the hidden layer so the H[1,k]will be passed directly to the output layer.

4. Output layer will apply dot product between H[1, N] and W'[N, |v|] and will give us the vector U.

5. Now, to find the probability of each vector we'll use the softmax function. As each iteration gives output vector U which is of one hot encoding type.

6. The word with the highest probability is the result and if the predicted word for a given context position is wrong then we'll use backpropagation to modify our weight vectors W and W'.

This steps will be executed for each word w(t) present in vocabulary. And each word w(t) will be passed k times. So, we can see that forward propagation will be processed |v|*k times in each epoch.