

Natural Language Processing (NLP)

Contents

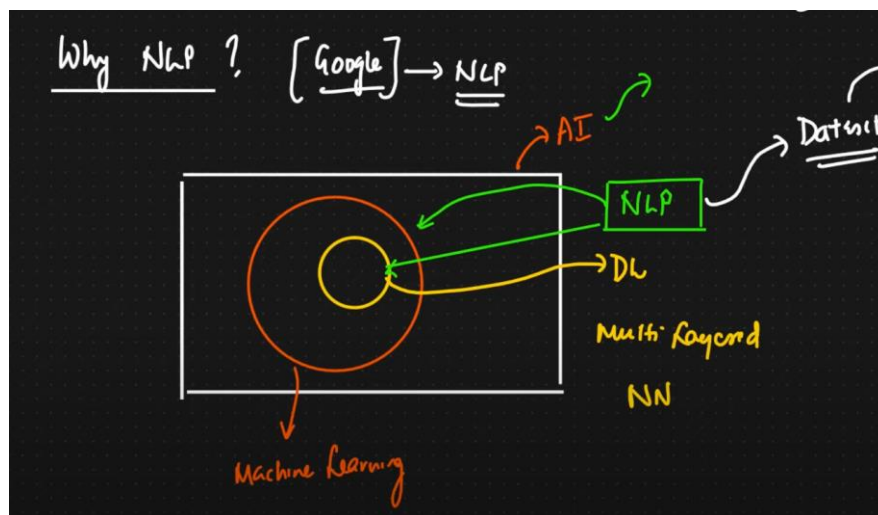
Why NLP?.....	3
Current limitation of NLP	4
Roadmap of NLP	4
Basic Terminology used in NLP.....	4
Preprocessing of text data	5
Tokenization	5
Stop words.....	6
Stemming	6
Lemmatization	7
Lowering the case of Words.....	7
Word Embedding.....	7
One Hot Encoding.....	8
Bag of Words (BOW)	9
Capturing semantic Information using Ngrams.....	13
TF-IDF (Term Frequency - Inverse Document Frequency).....	14
Word2Vec	17
Feature Representation	18
CBOW (Continuous Bag of Words) Architecture	19
Skip Gram.....	22
Word to Cloud.....	23
Average Word2vec	24
Spam Classification using Different Techniques from scratch	24

Why NLP?

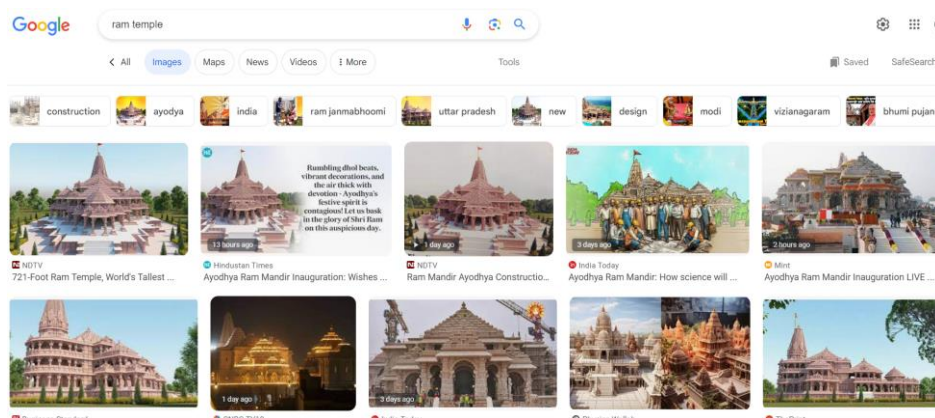
- Content recommendation like news ads based on profile and interest.
- Sentiment analysis
- Prediction of next text for Generative AI
- Text summarization, language translation
- Spam classification
- Chatbot

NLP can be used both machine learning and deep learning. At the end of the day we are creating a AI application.

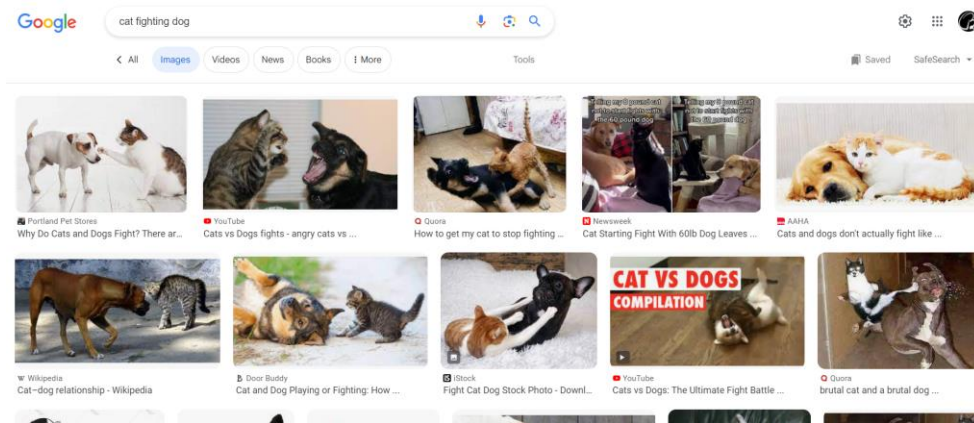
We will see how words can be converted to vector how machine is able to understand text and give output.



Example: Say we are searching in google as ram temple



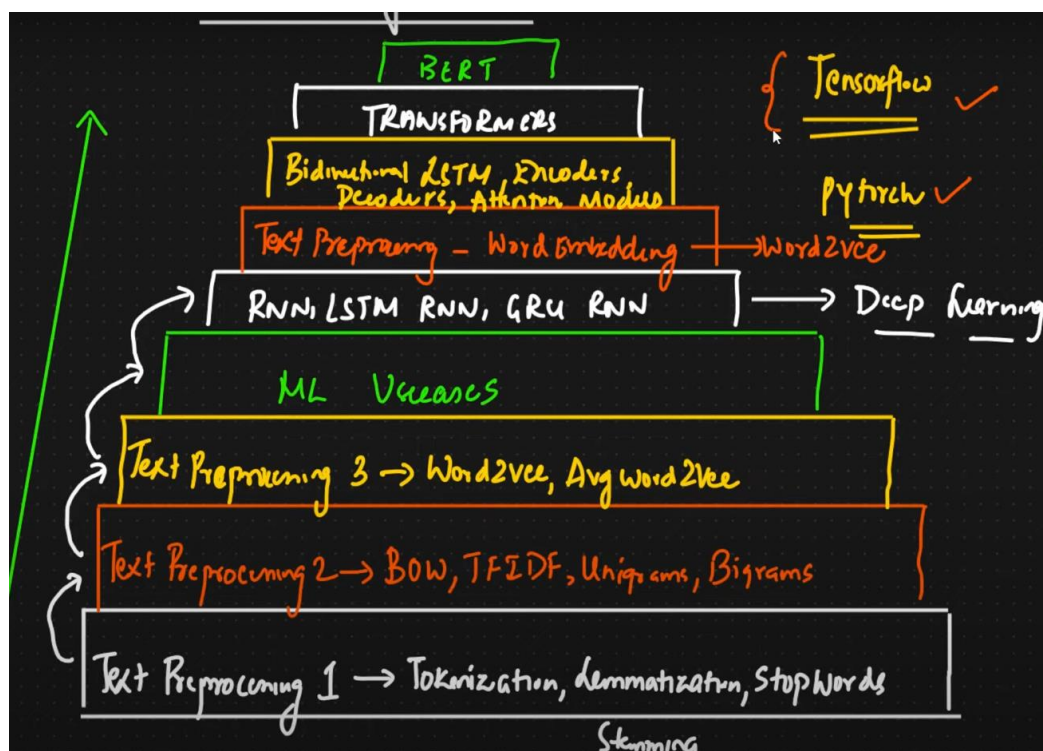
Now how all the images are associated with these images mean in the background text to image conversion is happening that might be using NLP.



Current limitation of NLP

NLP not able to understand sarcasm in text. Machine is not able to understand sarcasm properly, so research is going on.

Roadmap of NLP



Basic Terminology used in NLP

1. CORPUS
2. Documents
3. Vocabulary
4. Words

	<u>Text</u>	<u>o/p</u>
D1	The food is good	1
D2	The food is bad	0
D3	Pizza is amazing	1
D4	Burger is bad	0

CORPUS: It's like a **paragraph** like entire data set if we club together. Say my CORPUS is [D1,D2,D3,D4]

Document: Each **sentence** in the paragraph can be called as Document in NLP like D1 D2 etc. Though it is just a sentence but for NLP it is document. e.g. The food is good => D1

Vocabulary: It is the collection of all **unique words** in the CORPUS.

Say in my CORPUS we have 10K unique words in this case my vocabulary will be

The, food, is, good, bad, Pizza, Amazing, Burger

Words: All words in the CORPUS.

Preprocessing of text data

We need to do preprocessing of raw text before using it for NLP we will be using **NLTK** (**Natural Language Toolkit**) libraries for doing pre-processing. Here is the way we can do,

Code Link: <https://github.com/gourabb8273/ML-Model-HandsOn-Hub/tree/master/NLP>

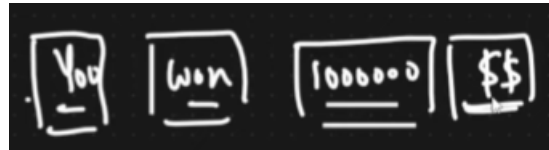
Tokenization

Say we are using spam classifier. Our data set will be like

<u>Mail Spam Classifier</u>			
<u>Dataset</u>			<u>Spam/Ham</u>
<u>f1</u>	<u>Email body</u>	<u>f2</u> <u>Email Subject</u>	<u>o/p</u>
1)	You won 100000 \$\$	Billionaire	Spam
2)	Hey KRISH, HOW ARE YOU	Hello	HAM
3)	Credit Cards worth	Wither	Spam

The first step here would be tokenization. It's about converting sentence into word so that those should be understandable by machine learning algorithm.

Sentence: You won 1000000 \$\$ after tokenization it will be



Stop words

Next step is stop words. Say we have a sentence like

“Hey buddy I want to go to your house” here we can see some words like **to** are not making any sense in prediction so we can remove those words.

Example of stop words: to, he she, is for of etc.

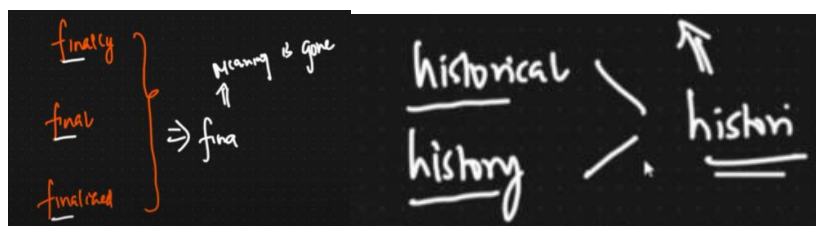
To remove all these words which are insignificant in output prediction using stop words. We can create our own custom stop words list as well. However we may need this in text summarization but not needed for spam classification or sentiment prediction.

While removing stop words you have to be very careful as it may delete the entire sentence so imbalance will create in target and input feature better to have custom stop word

Stemming

Here if we have say many similar types of words in different form we can convert those in base or root word thus we can reduce the number of words. It's a process of reducing words to the base word.

But base word may or may not have any meaning.



Advantage: It's very fast can be applied on large dataset

Disadvantages: It is removing the meaning of the word

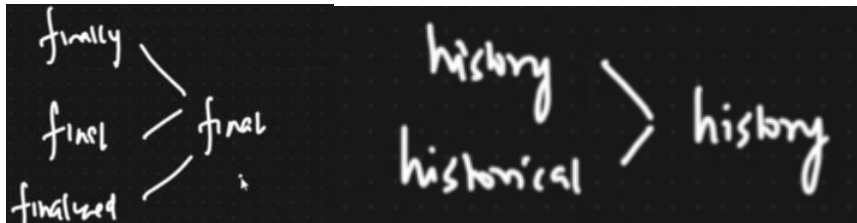
Use cases:

- Spam classification
- Review classification

To overcome it we have lemmatization.

Lemmatization

It has entire dictionary of words it will work the same way like stemming but it will give meaningful word.



Advantage: We will be able to get meaningful word.

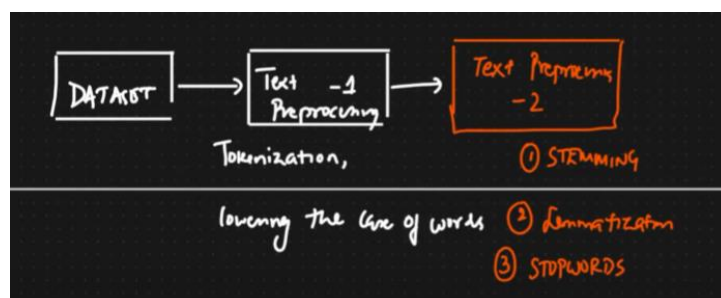
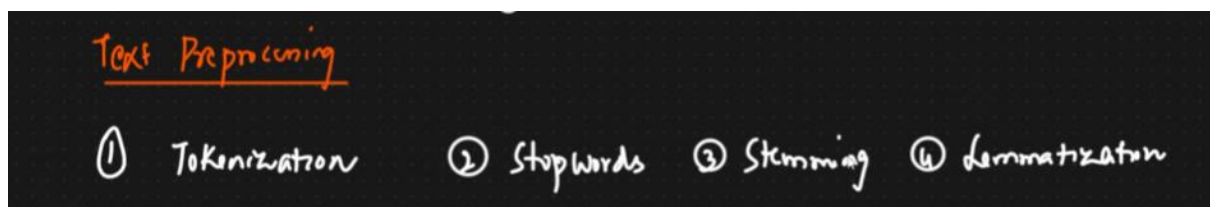
Disadvantages: It is slow as it has to do lots of comparison.

Use cases:

- Text summarization
- Language translation
- Chatbot

Lowering the case of Words

We will be now lowering the case of all word.

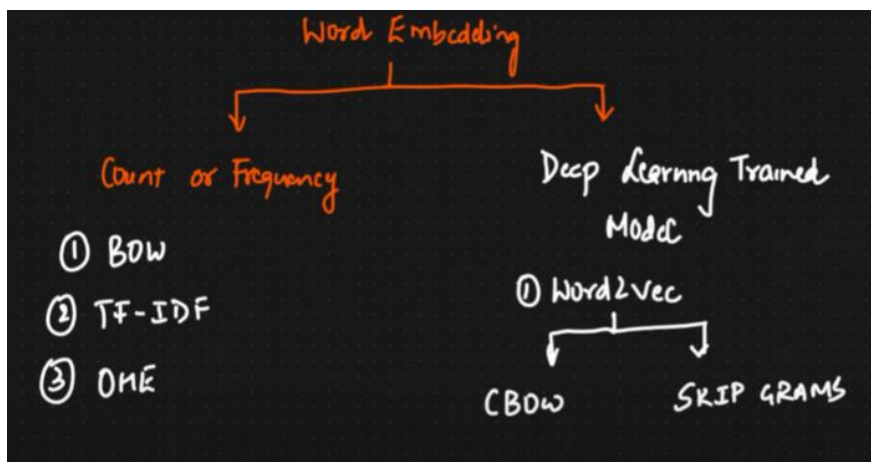
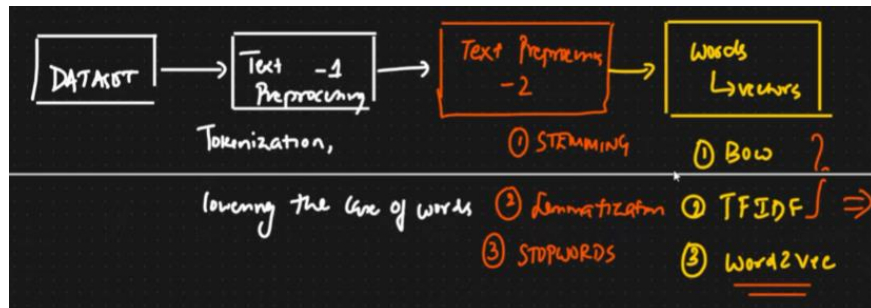


Word Embedding

In the next step we will be converting words to vector using different techniques, Word embedding is a technique to convert **words into vector**.

1. OHE (One Hot Encoding)
2. Bag of words (BOW)

3. TF-IDF (Term Frequency Inverse Document Frequency)
4. WordtoVec
 - a. CBOW (Continuous Bag of Words)
 - b. Skip Gram



One Hot Encoding

Say my CORPUS is



Now each word will be represented as one hot encoding. After doing this we will get as follows

For first document D1 – **A man eat food** we will get.

$$D1 \rightarrow \begin{bmatrix} [1 & 0 & 0 & 0] \\ [0 & 1 & 0 & 0] \\ [0 & 0 & 1 & 0] \\ [0 & 0 & 0 & 1] \end{bmatrix}$$

Here we have 4 words

For the second document D2 – **Cat Eat Food** we will get.

$$D2 - \begin{bmatrix} [1 & 0 & 0] \\ [0 & 1 & 0] \\ [0 & 0 & 1] \end{bmatrix}$$

Here we have only 3 words so if vocabulary size is decreases then we can't train the model. Input feature should be fixed constant this called out of vocabulary.

Advantage:

- Simple to implement.
- Intuitive

Disadvantage

- For large set of vocabular size of sparse (many zero values) matrix will be large difficult to compute
- OOV (Out of vocabulary), my sentence is not fixed size. Any new extra words if we get can't be handled.
- Between the word semantic meaning is not captured like how two words related as each row only one column is 1 rest zero.

Bag of Words (BOW)

It is a technique of converting **text to vector**. Say my CORPUS is as follows,

D1 → He is a good boy
 D2 → She is a good girl
 D3 → Boys and girls are good

Now we will use stop words and remove unnecessary words also will lower all case.

Bag of Words

Stop words

D1 → He is a good boy D1 → good boy
 D2 → She is a good girl D2 → good girl
 D3 → Boy and girl are good D3 → Boy girl good

Now my vocabulary unique words will be my feature. Let's see that with frequency like no of time words are repeating,

Order of the feature will be based on frequency e.g. if **good** is repeating more time than other then first feature f1 will be **good**.

<u>Vocabulary</u>	<u>Frequency</u>		f ₁	f ₂	f ₃
good	3		→ good	boy	girl
boy	2	Doc 1	1	1	0
girl	2	Doc 2	1	0	1
		Doc 3	1	1	1

We will just increase the count in the feature based on the document. Like first doc was good boy so f1 and f2 became 1 and f3 is 0.

Say in Document 1 we have one more good like – **good boy good** then count of **f1** will be increased by 2.

Vocabulary	Frequency		f_1	f_2	f_3
good	3		→ good	boy	girl
boy	2	Doc 1	2	1	0
girl	2	Doc 2	1	0	1
		Doc 3	1	1	1

Binary Bag of Words: In Bag of Word there is an option called **Binary Bag of Words**. Like if any count more than 1 like f_1 then also we will make it 1.

	f_1	f_2	f_3	Max Defined
	→ good	boy	girl	O/p
Doc 1	1	1	0	0
Doc 2	1	0	1	1
Doc 3	1	1	1	0

Classification

Advantage:

- Simple and Intuitive

Disadvantage

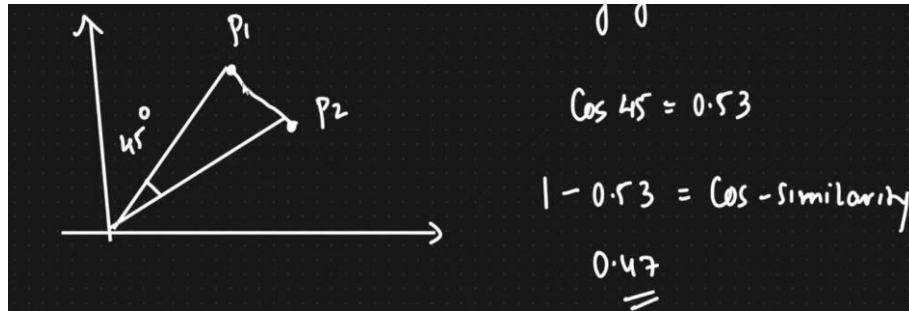
- Sparsity is still there though improved but not fixed.
- OOV (Out of vocabulary), if we get any new word like say **cat** then we won't be able to handle in feature so entire word will get rejected so meaning will get changed.
- Ordering of the word completely changed as we are doing ordering based on max frequency so won't be able to make semantic relationship.
- Semantic meaning not captured well like good and better are similar word so how similar words are.

If we take those features of 3 dimensions and plot it we will be able to calculate distance between them to find the similarity

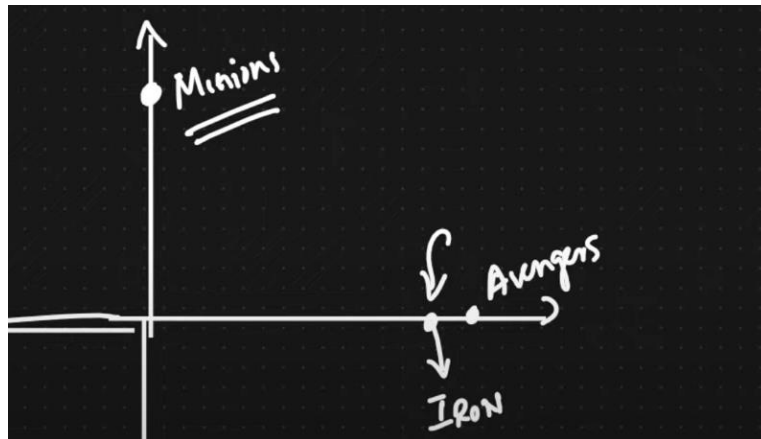
- Euclidian Distance
- Cosine Similarity

Cosine Similarity

Say we have two point and we need to find out the similarity we will find the angle between them and subtract from 1. So P1 and P2 is 47% similar.

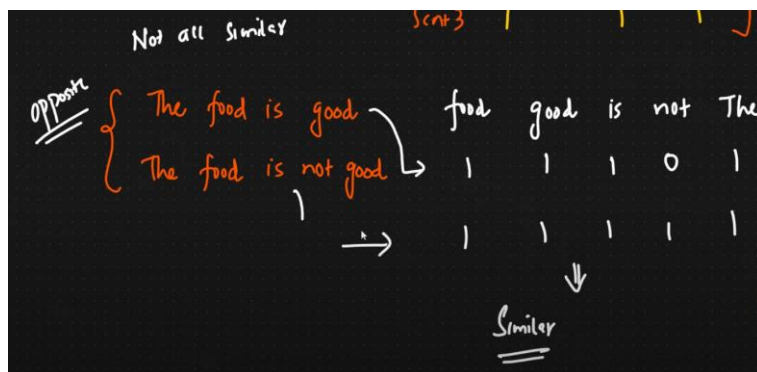


Say another example,



If the cosine similarity is close to 0 value then words are very similar that happens when angle will be less between them

This is heavily used in [recommendation engine](#) here avenger and Iron man are similar, but Minions is different as they are making 90-degree angle between them.



In the above example both the sentence or **documents are different like opposite** but the vector **looks similar** if we find cosine similarity that's the main problem **semantic meaning is missing** like which words should I focus more that will be **fixed using TF-IDF**.

Capturing semantic Information using Ngrams

- Bigrams (combination of two words)
- Trigram (Combination of three words)
- Ngrams (Combination of n words)

Apart from single feature we will be using combinations of features to understand semantic meaning of words.

D1 → good boy good
 D2 → good girl
 D3 → Boy girl good

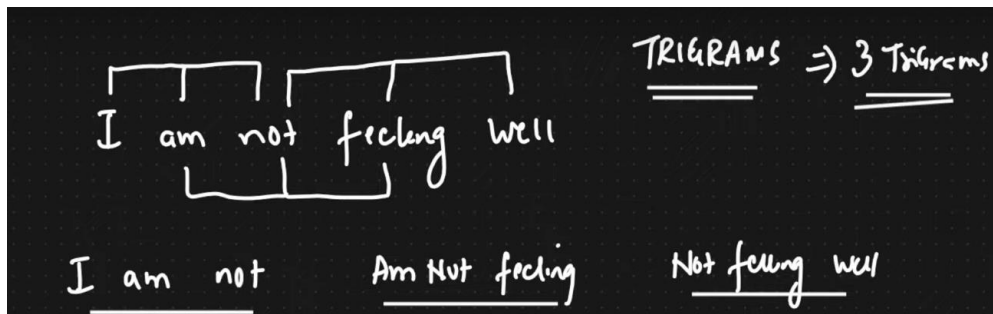
Ngrams ⇒ Bigrams, Trigrams, — N grams

	f ₁ good	f ₂ boy	f ₃ girl	f ₄ good boy	f ₅ good girl
Sent 1	1	1	0	1	0
Sent 2	1	0	1	0	1
Sent 3	1	1	1	0	0

We are adding two more features f₄ and f₅ using **Bigrams**.

How many **Bigrams** will be possible for word **I love food**? **I love and love food** only two bigram I food can't be possible.

How many **Trigram** will be possible for word **I am not feeling well**?



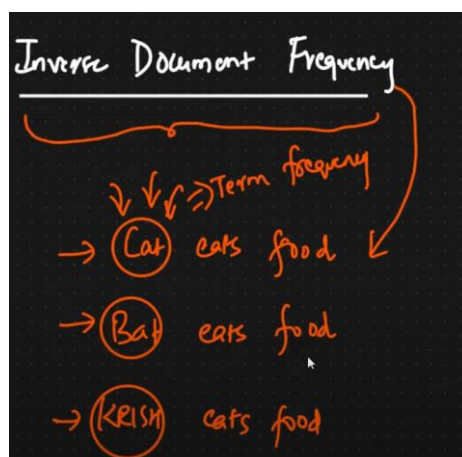
If we say combination of **Ngrams is (1,3)** then we will include single feature (unigram), Bigram and trigram 1 to 3. Say doc is I am feeling well my feature will be.

- f1 – I,
- f2 – am,
- f3- feeling,
- f4- well,
- f5-I am,
- f6-am feeling,
- f7-feeling well,
- f8-I am feeling,
- f9- am feeling well

TF-IDF (Term Frequency - Inverse Document Frequency)

It has two important concept **Term Frequency** and Inverse **Document Frequency**. Here we are trying to solve the issue i.e. semantic meaning of the words to know which word we need to focus more on.

Whichever words are **rarely present** in the sentence we will give **more weightage**.



Here rare words are Cat, Bat and Krish rarely present in the sentences so we need to give more weightage.

- These **rare** words will be captured by **Term Frequency** which will have more weightage.
- **Common** words will get captured by **Inverse Document Frequency**.

When we combine both semantic meaning will be there.

Term Frequency:

No of repetition of word in sentence / No of words in sentence

$$\text{Term Frequency} = \frac{\text{No. of rep of words in sentence}}{\text{No. of words in sentence}}$$

Inverse Document Frequency:

$$\text{IDF} = \log_c \left(\frac{\text{No. of sentences}}{\text{No. of sentences containing the word}} \right)$$

By multiplying both we will get TF-IDF

Example

Find Term Frequency

Here is my sentence.

		Term Frequency		
		Sent 1	Sent 2	Sent 3
Sent 1 :	good boy			
Sent 2 :	good girl	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{3}$
	boy	$\frac{1}{2}$	0	$\frac{1}{3}$
Sent 3 :	boy girl good	0	$\frac{1}{2}$	$\frac{1}{3}$

Here for word good in sentence 1 only one time is present and total word in sentence 1 is 2 hence TF is $\frac{1}{2}$.

For boy in sentence-3 only one time is present total word in sentence is 3 so TF is $\frac{1}{3}$.

Find Inverse Document Frequency

This gets computed for every sentence.

		Inverse Document Frequency	
		Words	IDF
Sent 1 :	good boy	good	$\log_c(3/3) = 0$
Sent 2 :	good girl	boy	$\log_c(3/2)$
Sent 3 :	boy girl good	girl	$\log_c(3/2)$

Here no of sentence is fixed that is 3 and good is appearing in all 3 sentences so good is common word and $\log(3/3) = 0$ so no weightage.

Similarly for word boy it is appearing only in sentences so $\log(3/2)$

If we multiply both then we will get TF-IDF

Term Frequency				Inverse Document Frequency	
	Sent 1	Sent 2	Sent 3	Words	IDF
↓ good	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{3}$	good	$\log_c(3/3) = 0$
boy	$\frac{1}{2}$	0	$\frac{1}{3}$	boy	$\log_c(3/2)$
girl	0	$\frac{1}{2}$	$\frac{1}{3}$	girl	$\log_c(3/2)$

	f ₁	f ₂	↓ f ₃
	good	boy	girl
Sent 1	0	$\frac{1}{2} \times \log_c(3/2)$	0
Sent 2	0	0	$\frac{1}{2} \times \log_c(3/2)$
Sent 3	0	$\frac{1}{3} \log_c(3/2)$	$\frac{1}{3} \log_c(3/2)$

Now we will multiply TF and IDF in feature space like for good in sentence 1 we had $\frac{1}{2}$ and from IDF it is 0 so it will be $\frac{1}{2} \times 0 = 0$

- ✓ Thus, we are removing or reducing weightage for common words and increasing weightage for rare word so some semantic meaning is captured.

Advantage

- Intuitive

- Word important is getting captured like semantic meaning.

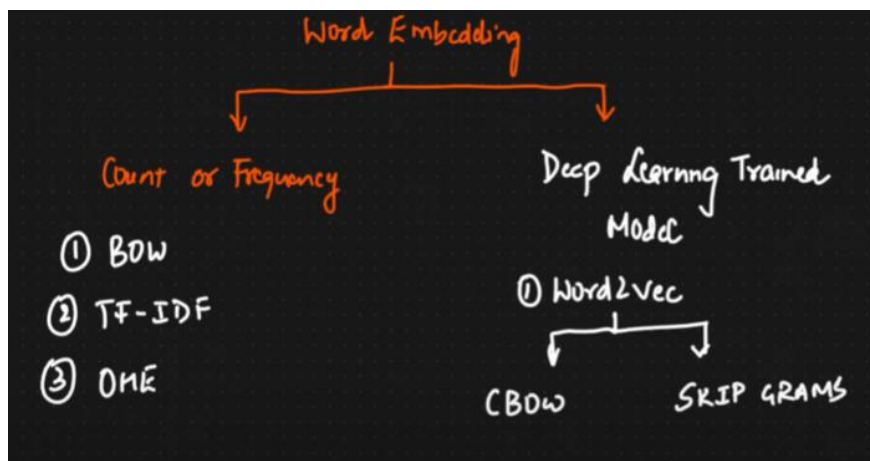
Disadvantage

- Sparsity is still there for large data set though better than BOW.
- Out of Vocabulary is still there.
- Semantic meaning not captured well like good and better are similar word so how similar words are.

Word2Vec

https://github.com/gourabb8273/ML-Model-HandsOn-Hub/blob/master/NLP/Practicle/text_preprocessing_Word2Vec.ipynb

Now we have seen few technique for creating word embedding based on count of frequency now it's time to see another popular technique using Deep Learning Trained Model. Those are CBOW and Skip Gram.



The main two problem that is **sparsity** and **semantic meaning** of words like **how similar two words** will be fixed in word2Vec technique. We will have embedding layer that convert text to vector.

Say we have some features like words in sentences.



- Every word we will create a vector but of **limited dimension** like 100 or 300 dimensions within that many dimensions we will represent the entire word.
- Sparsity will be reduced we won't found zero value all will be non-zero value.

- Semantic meaning will be captured like if two words say good and better are similar then in vector also difference in value will be very less.

Feature Representation

Every word like Boy Girl will be represented as some no of features.

- Say we have 300 dimensions of word or feature like Gender, Royal Age etc and every word of my input document will be represented as weightage of those feature in vector format or relate with those 300 dimensions of feature. Those 300 dimensions of feature will be created by word2vec algorithm.
- Google has done this with more than **3 billion** no of dimension of feature.

Handwritten table illustrating semantic features (Gender, Royal, Age, Food) related to words (Boy, Girl, King, Queen, Apples, Mango) across 300 dimensions. The table shows weightage values for each feature across the words.

Semantic (related words)	f_1	f_2	f_3	f_4	f_5	f_6
Gender	-1	1	-0.92	+0.93	0	0.1
Royal	0.01	0.02	0.75	0.96	-0.02	0.01
Age	0.03	0.02	0.7	0.6	0.95	0.76
Food						
↓						
300 dimension						

- In the row we have feature or vocabulary from our corpus.
- In the column we have 300 dimension of feature those are created by word2vec algo
- We are having weightage or how my vocabulary is related to those features.
Royal is related to King so 0.95 say. Thus, we are identifying semantic meaning of the words.

Handwritten table illustrating feature representation (Gender, Royal, Age, Food) for words (Boy, Girl, King, Queen, Apples, Mango) across 300 dimensions. The table shows weightage values for each feature across the words.

Feature Representation	Boy	Girl	King	Queen	Apples	Mango
Gender	-1	1	-0.92	+0.93	0	0.1
Royal	0.01	0.02	0.75	0.96	-0.02	0.01
Age	0.03	0.02	0.7	0.6	0.95	0.76
Food						
↓						
300 dimension						

- Here each of my words like **Boy represented as vector of having limited dimensions like here 300** through we are representing based of 300 features. Ideally words will be very high dimensions like billion.

Say assuming we have only 2 feature dimensions and we have words like King, Queen, Man, Women so we will get **King - Man + Women = Queen** using vector addition and subtraction or **cosine similarity** we can also use Euclidian distance or Manhattan distance.

$$\begin{array}{ll} \text{King} [0.96 & 0.95] & \text{Man} [0.95 & 0.98] \\ \text{Queen} [-0.96, & 0.95] & \text{Women} [-0.94 & -0.96] \end{array}$$

$$\text{KING} - \text{Man} + \text{Women} = \underline{\underline{\text{Queen}}}$$

CBOW (Continuous Bag of Words) Architecture

Here we predict the target word from context.



Say we have a Corpus collection of sentences.

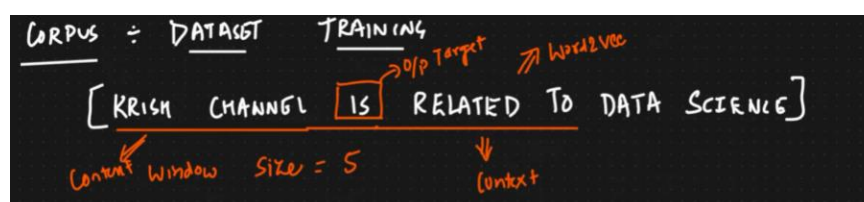
[KRISH CHANNEL IS RELATED TO DATA SCIENCE]

- We need **window size** for creating our training data.
- With respect to training data we will have independent feature and output feature.

Example:

Assume window size is 5 (choose odd number).

So according to CBOW **centre word** will be the **output** or target and before and after words will be independent feature or context.



Independent feature	O/p
KRISH, CHANNEL, Related, To	IS

Now I will **slide my window** by one step. So next centre word or target is **Related**. Similarly, we will get the independent feature and target.

<u>Independent feature</u>	<u>O/p</u>
→ KRISH, CHANNEL, Related, To	IS
→ CHANNEL, IS, To, DATA	Related
→ IS, Related, DATA, SCIENCE	To

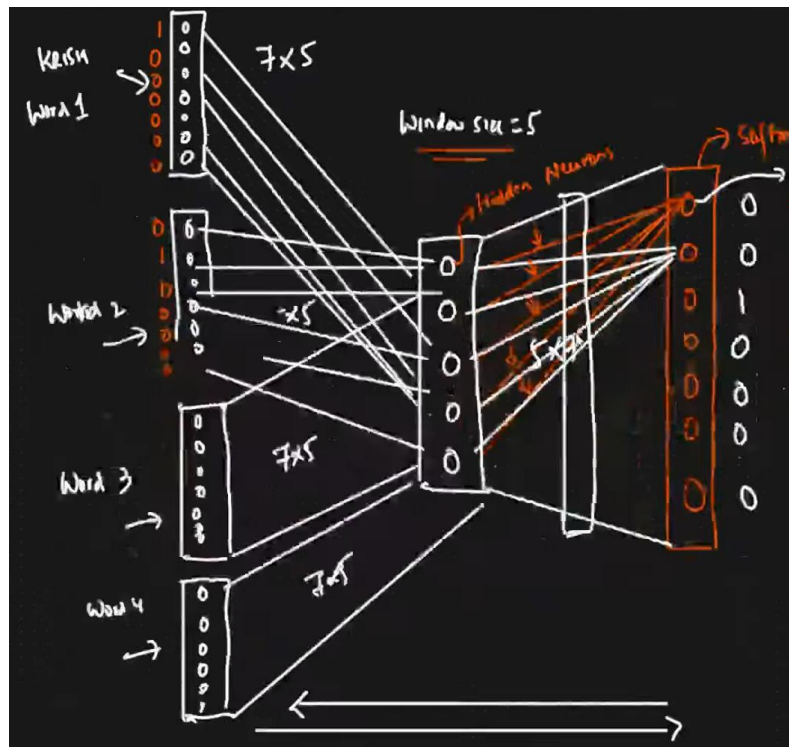
Now I can't slide because I won't get 5 words. In real google take entire dictionary 13 million words to do this with millions of words.

Now we also need to represent the words as One Hot Encoding like OHE. Then both OHE and our feature data will be fed to Deep Learning Neural network i.e. fully connected layer to learn each word representation as vector of feature.

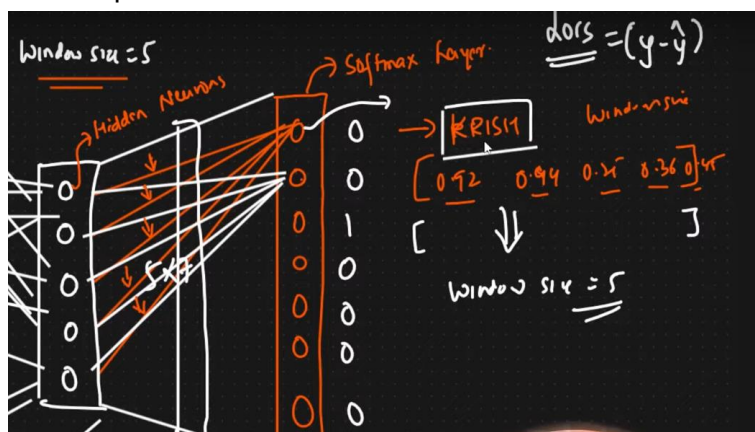
<u>Independent feature</u>	<u>O/p</u>		Box OHE
→ KRISH, CHANNEL, Related, To	IS	KRISH	1 0 0 0 0 0 0
→ CHANNEL, IS, To, DATA	Related	Channel	0 1 0 0 0 0 0
→ IS, Related, DATA, SCIENCE	To	IS	0 0 1 0 0 0 0
		Related	0 0 0 1 0 0 0

↓

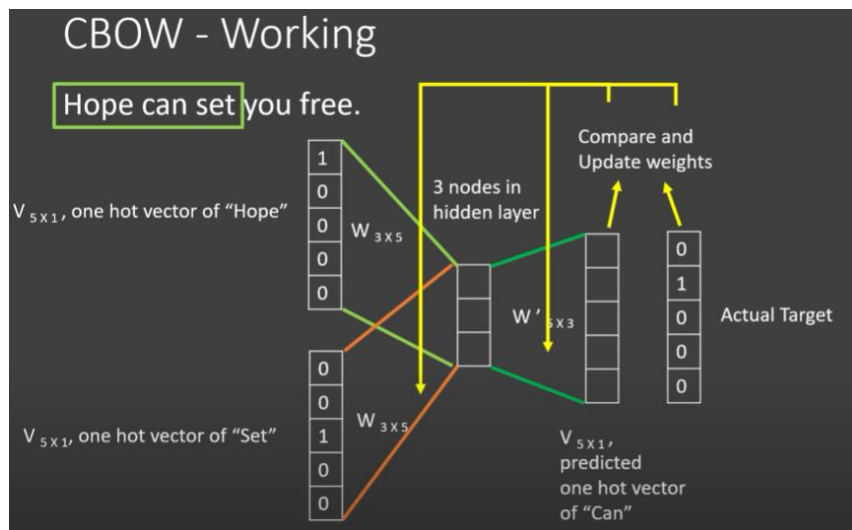
Here input size is 4 words due to 5 window size and each vector is represented by 7 vector using OHE.



- In first sample 4 words like Krish, Channel, Related, to each words will be represented as 7 vector as input feature i.e. 7*5 matrices in ANN.
- **Window size is 5 so hidden layer will be having 5 neurons so each word will be having 5 vector weightages** with 5*7 dimension. Word vector dimension not always same as window size it could be more but not dependent on vocabulary length
- Out put layer is having 7 nodes as IS is having 7 vectors in OHE. Here we will use **SoftMax** layer.
- In the out put we have 7 vectors so the weightage of hidden neuron connection will be the actual vector representation of each words of vocabulary. Like first index in output layer is Krish in our vocabulary which is currently 0. Now after training it will have weightage like the how all hidden neurons are connected to first output index.



- Thus, we are getting semantic meaning and as those weightages are not zero hence it is not a sparse matrix.
- We can increase the hidden layer thus we can do hyper parameter tuning.



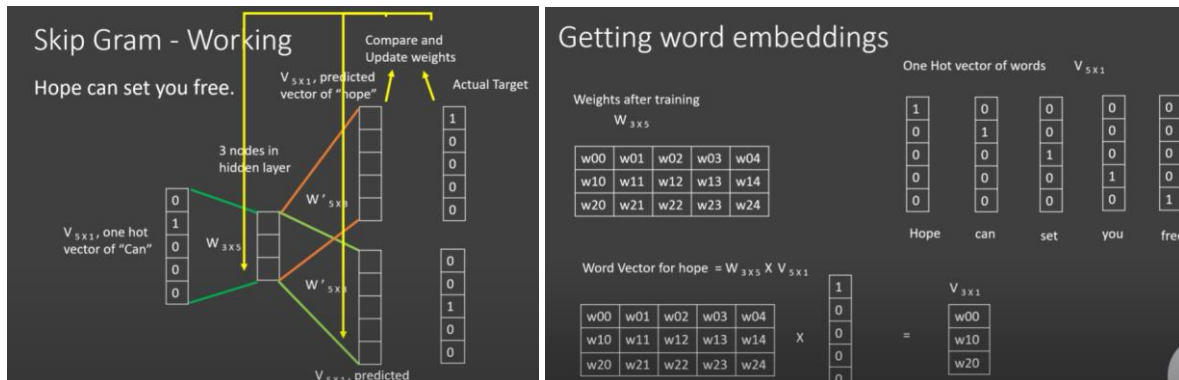
Skip Gram

Here we are doing opposite like predicting the context words from target word.



- Consider window size as same as 5.
- Here input will be the centre word and output will be the context words.
- Just reverse the ANN and we will get Skip Gram
- If google is creating a word to vec as 300 dimensions like every word will have 300 values related to other feature word then window size is 300.

<u>I/p</u>	<u>O/p</u>
Is	KERN, CHANNEL, Related, To
Related	Channel, Is, To, Data.
To	Is, Related, Data, Science



- For large data use Skip Gram other wise use CBOW

More window size means very good model more semantic information can be captured.

Gensim is a Python library for natural language processing, offering tools for document similarity, topic modeling, and word embedding. It enables efficient analysis of large text datasets for extracting meaningful patterns and insights.

```
from gensim.models import Word2Vec, KeyedVectors
```

We will be using pre trained model made using googole news information more 3 billion data by Google for word embedding

```
import gensim.downloader as api
wv = api.load('word2vec-google-news-300') # this eill be of size in 1662MB
```

```
wv.most_similar("king")
```

```
[('kings', 0.7138045430183411),
 ('queen', 0.6510956883430481),
 ('monarch', 0.6413194537162781),
 ('crown prince', 0.6204220056533813),
 ('prince', 0.6159993410110474),
 ('sultan', 0.5864824056625366),
 ('ruler', 0.5797567367553711),
 ('princes', 0.5646552443504333),
 ('Prince Paras', 0.5432944297790527),
 ('throne', 0.5422105193138123)]
```

```
wv.most_similar("man")
```

```
[('woman', 0.7664012908935547),
 ('boy', 0.6824871301651001),
 ('teenager', 0.6586930155754089),
 ('teenage girl', 0.6147903203964233),
 ('girl', 0.5921714305877686),
 ('suspected purse snatcher', 0.571636438369751),
 ('robber', 0.5585119128227234),
 ('Robbery suspect', 0.5584409832954407),
 ('teen_ager', 0.5549196600914001),
 ('men', 0.5489763021469116)]
```

```
wv.similarity("boy", "man")
```

```
0.68248713
```

```
vector = wv["king"] - wv["man"] + wv["woman"]
```

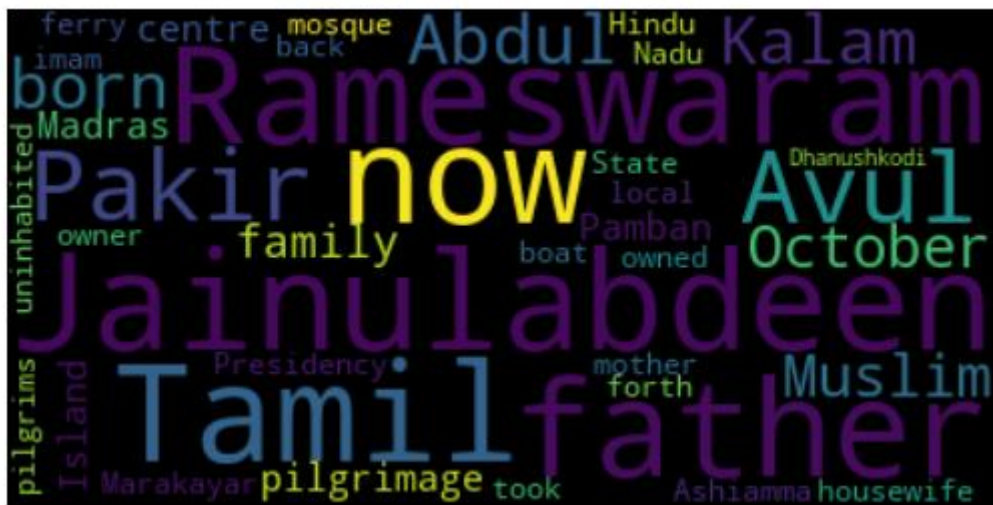
```
wv.most_similar(vector)
```

```
[('king', 0.8449392318725586),
 ('queen', 0.7300517559051514),
 ('monarch', 0.645466148853302),
 ('princess', 0.6156251430511475),
 ('crown prince', 0.5818676352500916),
 ('prince', 0.5777117609977722),
 ('kings', 0.5613663792610168),
 ('sultan', 0.5376775860786438),
 ('Queen Consort', 0.5344247817993164),
 ('queens', 0.5209087109863112)]
```

Word to Cloud

If I convert those 300 dimensions to 2D then it will look as word to Cloud.

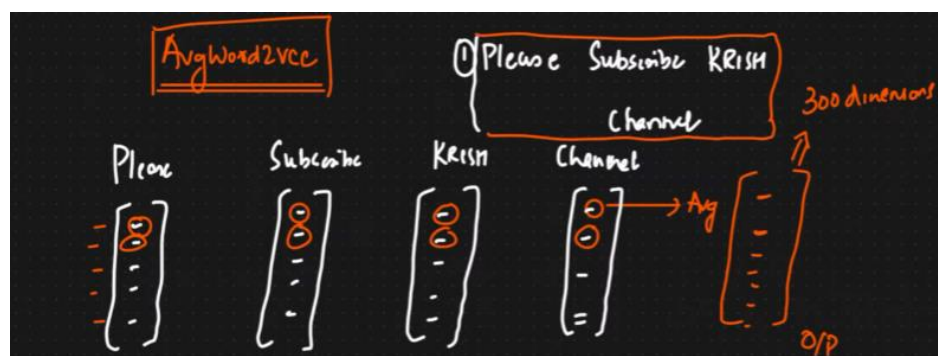
wordcloud is a Python library that creates visual representations of word frequency in a text corpus, generating "word clouds" where the size of each word reflects its frequency or importance in the given text.



Average Word2vec

While training a model using word2Vec from scratch not by using any pretrained model there will be one problem. Here say for spam filtering input would be my message of any no of words.

Now if we use 300 as window size then each word of the message will be of 300 dimensions, but we need 300 as a total dimension for the input feature as **input dimension should be fixed i.e. 300**. Hence, we will be taking the average of vector for all words.



Spam Classification using Different Techniques from scratch

<https://github.com/gourabb8273/ML-Model-HandsOn-Hub/tree/master/NLP/Practicle/Spam-Classifer>