NLP [Natural Language Processing]

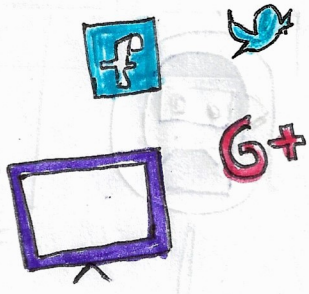
Text Mining :-

→ Understand

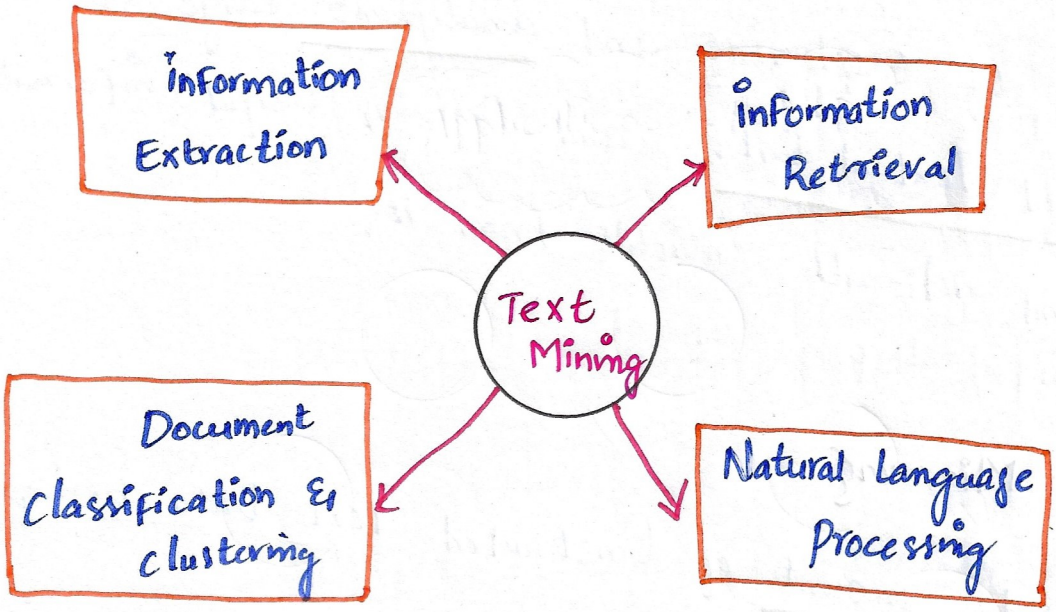
The Process of Exploring and analyzing large Collection of "Unstructured Textual Data" and Deriving Useful information, patterns, and Actionable insights from it.

Need Of Text Mining

- Unprecedented amount of Unstruted text Data being Generated Every Day
- Text mining transforms Unstruted Data into Structured Data To Further be used for Analysis
- Identify bussiness insights to feed bussiness processes and reduce Risks.
- Make informed Decisions, automate processes, Market research Using Sentiment analysis, etc.

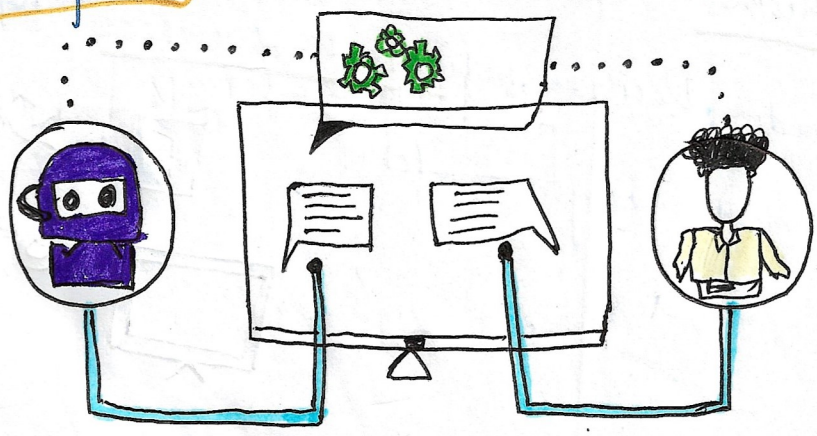


Areas of Text Mining



* NLP [Natural Language processing]

it is the Field of Study that Deals with the Understanding, interpreting and Manipulating human language Using Computers.



NLP in Text Mining

- Since most of the significant information is Written down in Natural Language such as English, French, German etc... and it is not Conveniently tagged.

- So, after identification and Extraction of the Content needed for text analysis, we Can Use different NLP techniques to Extract meaningful information from it.

- NLP helps Computers to Communicate with humans in their Own language and perform other Language related risks & Tasks

- NLP Makes it Feasible For Computers to read text, hear speech, interpret etc. it measures Sentiment and determine which parts are important.

Applications of NLP

* Sentimental analysis 
* Chat Bot 
* Speech Recognition 

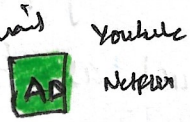
* Machine translation 

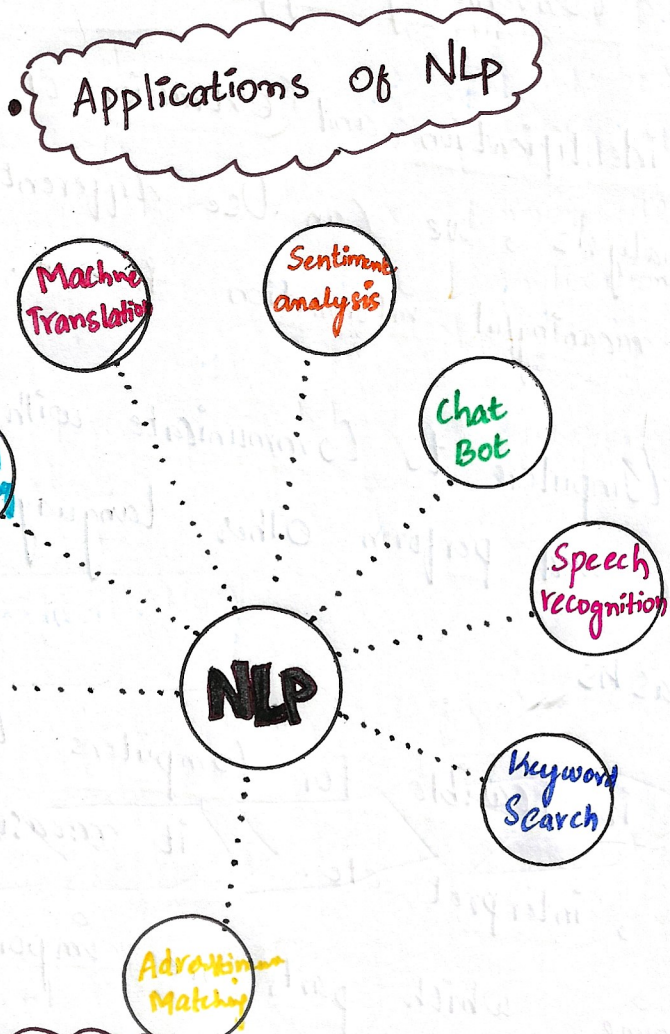
* Spelling checking 

* Information Extraction

* Keyword Search

* Advertisment Matching





* Natural language Tool Kit

- Python [NLTK corpora]
- Spacy
- Hugging Face
- BERT
- GPT3

NLTK Corpora

A Corpus is a huge collection of written texts. A Compilation of Corpuses is called a Corpora. it is a body of written (or) Spoken Texts used for Linguistic analysis and the development of NLP tools.

* Corpus :- Multiple Sentences (↳ Paragraph) in a Document / File is called Corpus.

* Corpora :- Multiple "Corpus" is called as Corpora

Ex :- Combination of different Gmails, Ms-word Documents

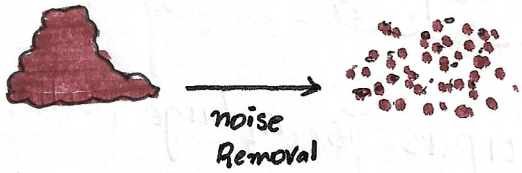
CODE

```
# import nltk
# nltk.download()
# Paragraph = "He is a good boy. She is good girl. boy & girl are good."
```

Steps

* Read Raw Text
* tokenize
* Clean text
  - stop words
  - stemming
  - Lemmatization
* Vectorize
* Build Machine Learing model

Text Cleaning

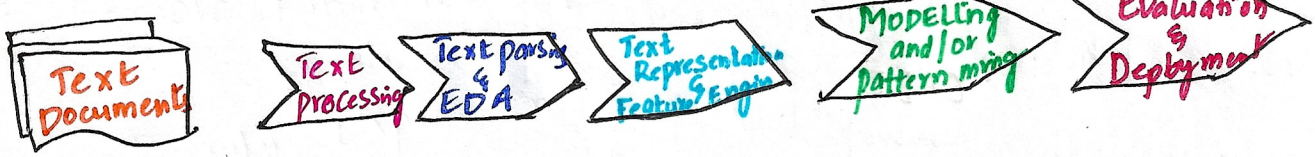


- Different methods in Text Cleaning

* Filter out Punctuation
* Split into words [Tokenization]
* Filter out Stop words [is, the, are]
* Stemming words] Base word
* Lemmatization
* Normalizing Case

Noise Removal

NLT Task



- Text Document
- Text processing
- Text Parsing & EDA
- Text Representation & Feature Engineering
- MODELLing & Pattern mining
- Evaluation & Deployment

⇒ **Tokenization** :- Breaking into Parts

- Sentence Tokenization :- Breaking into Sentences
  - str.split(".")
    - ↳ Dot
- Word Tokenization :- Breaking into Words
  - str.split("-")
    - ↳ Space.

CODE

```
# Sentence tokenization
# Sentences = nltk.Sent_tokenize(Paragraph)
# Sentences
```

Out

```
["he is a good boy."
 "she is a good girl."
 "boy & girl are good."]
```

```
# Word tokenization
# words = nltk.word_tokenize(paragraph)
# words
```

Out

```
['he',    "she",   "boy",
 "is",    "is",    "&",
 "a",     "a",     "girl",
 "good",  "good",  "are",
 "boy",   "girl",  "good",
          ".",     "."
```

Text Cleaning

- Remove Punctuation [. , ! " " &]
- Remove stop words [is, the, are, so, and]
- Stemming } Base words
- Lemmatization }

* history, historical, histo
* Beautiful, Beautifully, Beauty

⇒ Vectorization [Words → Numeric]
{Converting words to vectors}

* Text cleaning + Vectorization } Text Preprocessing

Remove Punctuation [. , ! " " &]

- Regular Expression [Regex]

#Website : regex101.com

```
import re
```

Extraction of words, sentences, numbers, Punctations

* \d ⇒ matches a digit [0-9]
* \d{3} ⇒ matches Exactly Three times [777 000 999]
* \(\d{3}\)-\d{3}-\d{4} | \d{10}

* We have phone number 8886669990. Using regex how we extract data.

Pattern : \d \d \d \d \d \d \d \d \d \d

8 8 8 6 6 6 9 9 9 0

(or)

\d{10}

8886669990

* Suppose we have USA number (888)-666-9990

Pattern : \(\d{3}\) - \d{3} - \d{4}

(888) - 666 - 9990

∴ ( ) are special character. So, we Use \ before ( ) & End

code

```
# import re
# text = " Jack is very Far away, we can Contact him by 7989900099. and he had another USA number (799)-899-0877".
# pattern = "\d{10} | \(\d{3}\) - \d{3} - \d{4}"   (or)
# re.findall (Pattern, text)
```

Out: ['7989900099", "(799)-899-0877"]

If we want to Extract headline of a paragraph

```
# text = '''Note 1 - Overview
The project is all about ..... .....
..... ..... ..... ..... ..... .....
..... ..... ..... ..... ..... .....
..... ..... ..... ..... ..... .....
Note 2 - Memory
memory is all about ..... .....
..... ..... ..... ..... ..... .....
..... ..... ..... ..... ..... .....'''
```

```
# Pattern = "Note \d -([^\n]+)"
```

- Note → Starting word
- \d → digit
- (→ capturing within
- [^\n]+ → including one (or) more
- [^ → (Starting of string)
- \n → (newline)
-) → Extract

```
# re.findall(pattern, text)
```

Out: ["Overview", "Memory"]

∴ If we don't mention ( ) in pattern.

```
# pattern = "Note \d -[^\n]+"
```

Out: ["Note 1 - Overview", "Note 2 - Memory"]

If we want Exact any Series of code in text

FY2022 Q1

Pattern: FY\d{4} Q\d → digits

- FY → Starting Letters
- \d{4} → Upto 4 digits
- Q → "Q" mention in Question

But, any Non-linear Conter like Q5, Q6 ... From text we don't those Q5, Q6, to Extact

#Text = """ The year consist FY2021 Q1 was $21 billions. FY2022 Q5 $ 2 billion, FY2023 Q2 $ 6billions"

Pattern = " FY \d{4} Q[1-4] "
↳ digit
↳ upto 1 to 4

re.findall ["pattern, text]

Out ["FY2021 Q1, FY2023 Q2"] ∴ Because, we have Taken range [1-4], So, It didn't get FY2022 Q5

If we want Ignore Capital and Small Letters.

Ex = fy2021 Q3, FY 2022 Q5, fy 2026 Q6

Pattern = FY \d{4} Q[1-4]

re.findall [Pattern, Ex, flags = re.IGNORECASE]
↳ it ignore case in the text.

Out : fy2021 Q3, FY2022 Q5, fy2026 Q6

IF we want only year and Q values.

Ex = fy2021 Q3, Fy 2022 Q5

Pattern = Fy (\d{4} Q\d)
↳ Here we Use Brackets to Exact information only inside Brackets.

re.findall [Pattern, Ex, flags = re.IGNORECASE)

Out : ['2021 Q3', 2022 Q5']

Extract only $ 4.85 billions, $ 3 billion from text

Text = "Jack hay $ 4.85 billions of cash in Dreams, but in hand he has $ 3 billion in hand"

Pattern = "\$[0-9\.]+"

- range
- dot also special character so, mention / before it
- include all after the dot
- starting
- dollar is a special character so, we mention slash before it

re.findall (pattern, text)

Out ["$4.85", "$3"]

pattern = "\$([0-9\.]+)" → Here we mentioned Bracket, so, it Extract with in the Brackets

Out : [4.85, 3]

Extract Email's from the text using regex

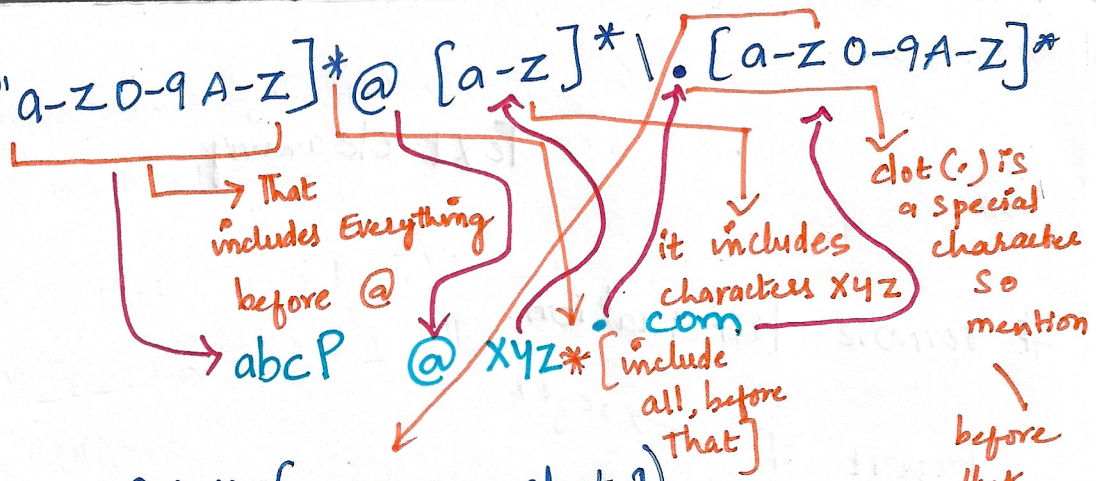
chat 1 = regarding flight ticket Contact abcP@xyz.com

chat 2 = Sorry I lost Contacting you my mail address sad_29@xyz.in

chat 3 = Your saidlygoand mail of mine is dog@que.com

Pattern = "[a-zA-Z0-9]*@[a-z]*\.[a-zA-Z0-9]*"

matches =



matches = re.findall [Pattern, chat 1)

matches

[Same for chat 2, chat 3]

Out: abcP@xyz.com
abc_29@mad.in
Badboy@gmail.com

Email Extraction : Regex

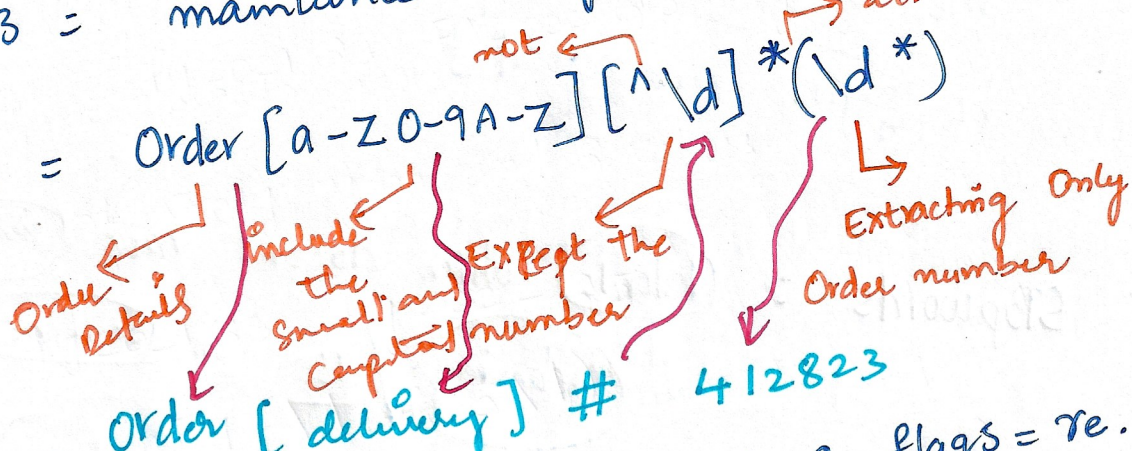
Extracting Order Issue Details from the Database.

Issue 1 = Problem with Order delivery # 412823

Issue 2 = MyOrder 283282 charged 80$ for product

Issue 3 = maintance charges details Order 200823

pattern = Order [a-zA-Z0-9] [^\d] *(\d*)



match = re.findall [pattern, Issue 1, flags = re.IGNORECASE)

Out: order delivery # 412823

Same format for Issue 2, Issue 3.

Text cleaning

remove Punctuation.

import re → regex

corpus = []

Len(list) → range(3) → [0, 1, 2,] (3 → i = 0)

for i in range (len(sentences)):

rp = re.sub ("[^a-zA-Z]", ' ', sentences[i])

^ → Except

[^a-zA-Z] → remove everything other than a-z, A-Z

' ' → remove space also.

re.sub → Substract

corpus.append (rp)

print (corpus)

Out: ['He is a good boy'
'She is a good girl'
boy & girl are good']

. are removed by "Regex"

Stopwords : Words which doesn't have any meaning / use in Sentence.

They are just used for framing Sentences.

Ex: For Example

∴ Example is Enough, But, we are using "For" → Stopwords

Examples of Stopwords

```
From nltk.corpus import stopwords

# stopwords.words("English")
```

Out :

i	you	him	It's	which	is	each	own	will
me	you're	his	itself	who	are	few	same	just
my	you've	himself	they	whom	was	more	so	don
myself	you'll	she	them	this	were	most	than	don't
we	you'd	she's	their	that	why	other	too	now
our	yourself	her	theirs	that'll	how	some	very	d
ours	yourselves	herself	themselves	these	all	such	s	m
ourselves	he	it	what	those	any	no	t	could
					both	nor	can	couldn
						only		

179 elements .....

```
# remove stopwords
# corpus = []
# for i in range [len(sentences)):
    rp = re.sub("[^a-zA-Z]," " ", sentences[i])
    rp = rp.lower()
    rp = rp.split()
    rp = [word for word in rp if not word
          in set(stopwords.words("English"))]
    rp = " ".join(rp)
    corpus.append(rp)
    Print(corpus)
```

Out : [good boy , good girl , boy girl good]

Stemming.
Lemmatization

→ Base Word : In order to process Data, we are going work on "WORDS". that's the Reason we bring it on Base word.

Application :- No human interaction. we use stemming
Ex:- Classification
Gmail

* Stemming :- bring To Base word, which has no meaning (or) not a dictionary word
* Lemmatization :- brings to Same Base word, But, it has meaning

Application :- human intraction, we use Lemmatization.
Ex:- ChatBoT
Alexa
Siri
Google assistant

#Stemming code

```
from nltk.stem import PorterStemmer

# ps = porterstemmer()

# ps.stem("history")
```

Out: "histori"

Lemmatization

```
from nltk.stem import WordNetLemmatizer

# wnl = WordNetLemmatizer()

# wnl.lemmatize("history")
```

Out: "history"

Backend, it opens "Dictionary".

→ only one should be used while writing a code (or) Building a model.

Stemming	Lemmatization
• Converting words to their word stem or root	• reducing words into their Lemma (or) dictionary
• Chopping off the end of a word, to leave only the base	• Remove inflectional endings and return to dictionary form of a word
• beautiful and beautifully words to beauti	• beautiful and beautifully words to beautiful.

Code

- We are working on Sentiment analysis and Text classification. So, we use "Stemming" for this model
- Lemmatization is used for the chatbot, Alexa, which has interactions.

Text Cleaning [stopwords, stemming, lemmatization]

```
# corpus = []
# for i in range(len(sentences)):
        rp = re.sub("[^a-z A-Z]", " ", sentences[i])
        rp = lower()
        rp = split()
        rp = [ps.stem(words) for word in rp
              if not word in set(stopwords.
                 words("english")
        rp = " ".join(rp)
        corpus.append(rp)

# corpus
```

Out: ["good boy", "good girl", "boy girl good"]

Vectorization

Convert tokens into to a format that a machine learning understands

"Converting the Text to Numeric Representation"

⇒ Methods of Vectorization *

1. Count Vectorization [Bag of words]
2. N-grams Count Vectorization
3. Term frequency - inverse document frequency [TF-IDF]

* **Bag of Words [BOW] / Count Vectorization :-**

Counting the number of "words" in the Sentence is known as "Bow" / Count Vectorizer.

Ex:- Sentence 1 = good boy
Sentence 2 = good girl
Sentence 3 = boy girl good

	boy	girl	good
Sentence 1	1	0	1
Sent 2	0	1	1
Sent 3	1	1	1

This is How Bow [Bag of words] works. and also Count Vectorizer works.

* it Just Counts the words in the Sentence and seperate them.

Ex: 2

Sentence 1 = Data good Furtur good

Sentence 2 = Data best present

Sentence 3 = Data poor past

	best	Data	furtur	good	present	poor	past
Sentence 1	0	1	1	1	0	0	0
Sentence 2	1	1	0	0	1	0	0
Sentence 3	0	1	0	0	0	1	1

code # bow [bag of words]

```
from sklearn.feature_extraction.text import
                                    CountVectorizer
# cv = CountVectorizer() binary=True, ngram_range=(3,3))
# bow = cv.fit_transform(corpus).toarray()
# bow
```

out: array([[1, 0, 1],
[0, 1, 1],
[1, 1, 1]].

	Boy	girl	good
Sent 1	1	0	1
Sent 2	0	1	1
Sent 3	1	1	1

* Problem with [Bag of words] : it is going to give Only Same weightage Count / Frequency of all words.

we can't identify which word is important in the Sentence

Advantage

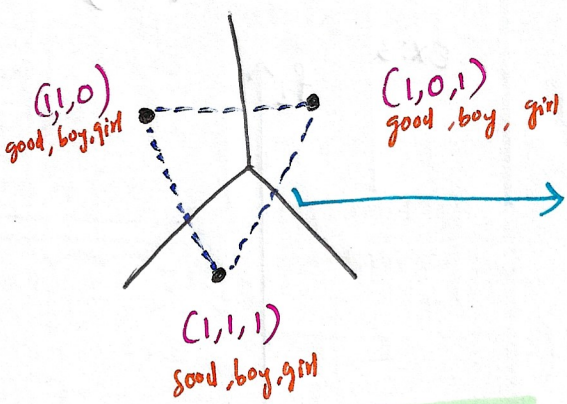
* Simple & Intuitive

DisAdvantage

* Sparsity * Semantic meaning lost (not able to capture)
* OOV [Out of vocublary]
* Ordering of words are completely changed.

If we want to plot in 3 [three] dimension

Ex :-



Distance Between them can be Calculated by.

* Cosine similarity
* Euclidian distance.

* So, what is Cosine Similarity & Cosine Distance ?

→ Tells How similar Sent 1, Sent 2 Sent 3 are similar to Each other.

∴ it is Widely Used in "Recommendation Systems".

We have Two points : P_1 & P_2

Similarity = Distance

Similarity Decreases ↓ ⇒ Distance increases ↑

Similarity Increases ↑ ⇒ Distance Decreases ↓

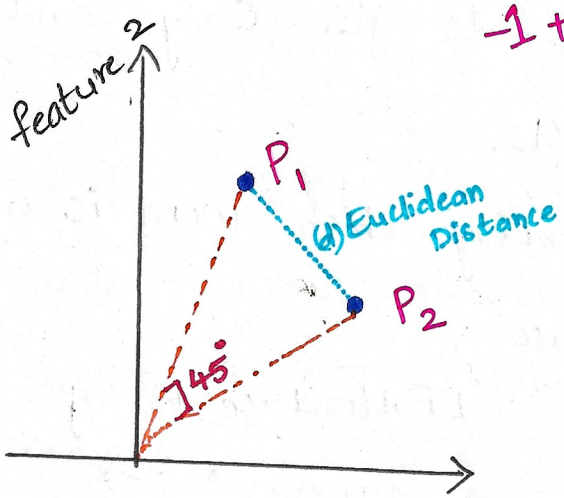
Formula.

$$1 - \text{Cosine-Similarity} = \text{Cosine Distance}$$

∴ According to Geometric Term.

it Ranges b/w
-1 to +1

Ex:-



$\therefore$ Cos-sin = Cos θ

Angle b/w P_1 & P_2

$\therefore \cos 45^\circ = 0.707$

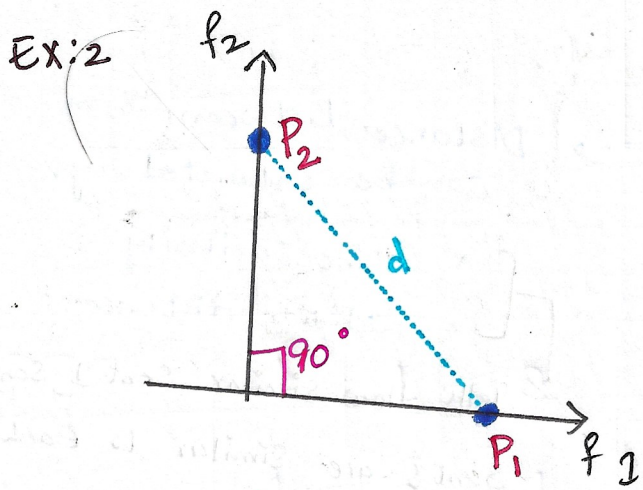
$\Rightarrow 0.70 \Rightarrow 70\%$ Similar.

Based on Distance & angle b/w them

But, In Cos-sine Similarity to find the Similarity we, have to find "angle" between the Data Points.

**** "Cosine Similarity" Formula $[\cos(\theta)] = \frac{A \cdot B}{|A| \times |B|}$ ← Dot product

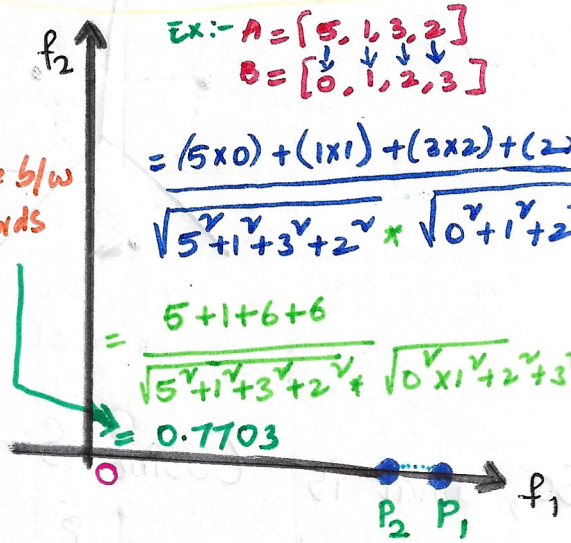
Ex:2



$\therefore \cos 90^\circ = 0$

These Data points are not similar To Each other.

Ex: 3



Ex:- A = [5, 1, 3, 2]
B = [0, 1, 2, 3]

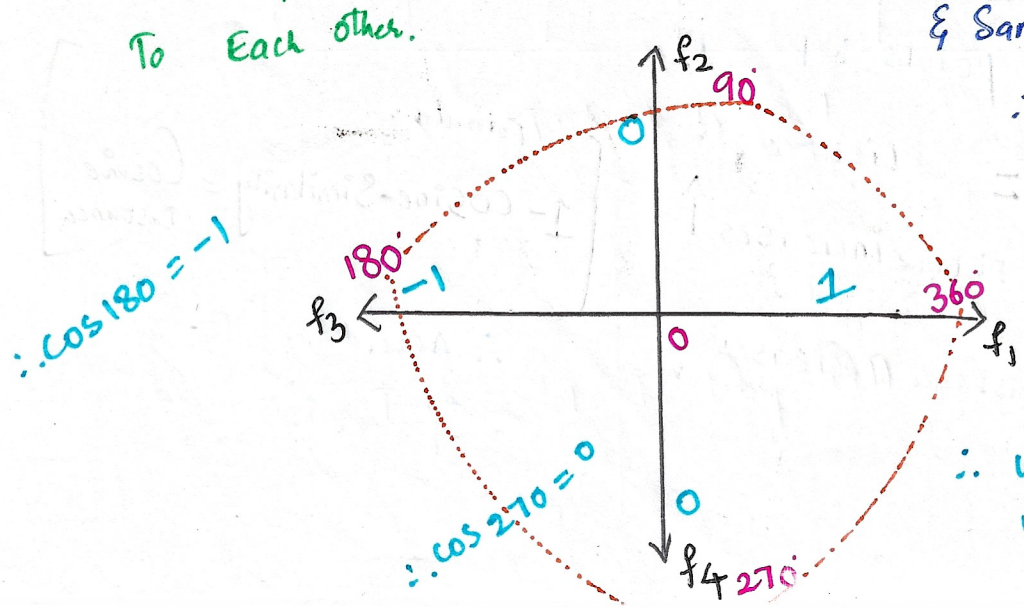
$$= \frac{(5\times0)+(1\times1)+(3\times2)+(2\times3)}{\sqrt{5^2+1^2+3^2+2^2} * \sqrt{0^2+1^2+2^2+3^2}}$$

$$= \frac{5+1+6+6}{\sqrt{5^2+1^2+3^2+2^2} * \sqrt{0^2\times1^2+2^2+3^2}}$$

$= 0.7703$

P_1 & P_2 are on the same plane & Same vector

$\therefore \cos 0^\circ = 1$



$\therefore \cos 180 = -1$

$\therefore \cos 360^\circ = 1$

$\therefore \cos 270 = 0$

$\therefore$ when angle is 0 = 1
when angle is 90 = 0

∴ So, we Want to Calculate ∴ Cosine Distance

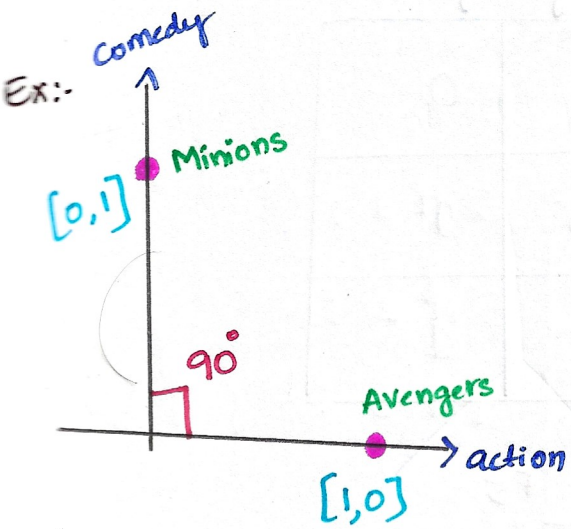
Cos-distance = 1 - Cos-similarity

→ cos 45 = 1 - 707 = 0.3

→ cos 90 = 1 - 0 = 1

→ cos 0 = 1 - 1 = 0

Ex:-

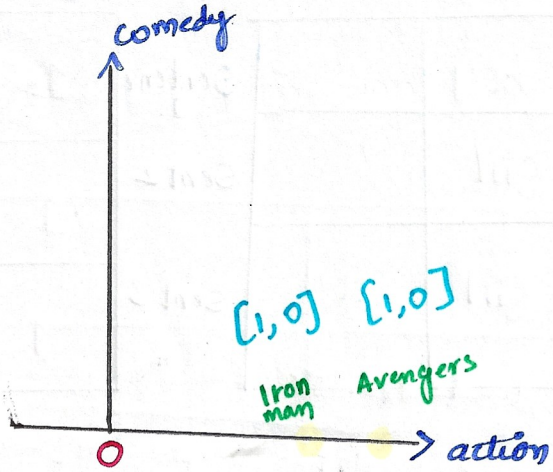


∴ cos 90° = 0

Distance = 1 - cos-sim

= 1 - 0

Distance = 1



∴ cos 0° = 1

Distance = 1 - cos-sim

= 1 - 1

= 0

∴ Distance = 0

* In Bag of words [Bow]/Count vectorize. we can't find The "Semantic meaning" (or) "information.

↳ Make words Meaningful.

So, we Use Method Called as "N-grams".

→ Combination of "n" words/Features.

N-grams

* Unigrams = it uses only single word (or) single features
* Bigrams = Combination of two words/features
* trigrams = Combination of three words/features

Ex:- Bi-grams.

D_1	Good	boy	
D_2	Good	girl	
D_3	Boy	girl	good

Feature 1	F_2	F_3
Good	boy	girl
1	1	0
1	0	1
1	1	1

Sentence 1
Sent 2
Sent 3

Bi-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

Ex:- Jack Eats Food = Bi grams

∴ Jack Eats
Eats Food

Trigrams

I am not Feeling well = Trigrams

∴ I am not
am not Feeling
not Feeling well

Ex:- The food is good
The food is not good → opposite

Food	good	is	not	The
1	1	1	0	1
1	1	1	1	1

But, by The Bag of words, it gives Every individual option has Equal importance.

So, we use method Called "TD-IDF"

TF & IDF

TF-IDF:- [Term Frequency & Inverse Doument Frequency]

∴ which are words are rarely present in the Sentences we should try to give more weightage

EX:-

Cat	Eat	Food
Dog	Eat	Food
PIG	Eat	Food

These are rare / Unique. So, we Calculate

These Common words Captured by The

TF :- Term Frequency

IDF :- inverse Doumment Frequency.

When we combine both TF & IDF we can get the most rare words in The sentences

Formula :

$$\therefore \text{Term Frequency} = \frac{\text{No. of repetation of Words in Sentence}}{\text{No. of words in Sentence}}$$

*

$$\text{Inverse Doument Frequency} = \log_e \left[\frac{\text{No. of Sentences}}{\text{No. of Sentences containing word}} \right]$$

⇓

TF-IDF

⇒ Term Frequency :- For Every Sentences

Sent 1 : Good boy

Sent 2 : Good girl

Sent 3 : boy girl good

	Sentence 1	Sent2	Sent 3
Unique words	Formula = No. of repetation of words in Sentences / No. of words in Sentences		
good	$\frac{1}{2}\left[\frac{1\text{-good}}{2\text{-good boy}}\right]$	$\frac{1}{2}\left[\frac{1\text{-good}}{2\text{-good girl}}\right]$	$\frac{1}{3}\left[\frac{1\text{-good}}{3\text{-good girl boy}}\right]$
boy	$\frac{1}{2}\left[\frac{1\text{-boy}}{2\text{-good boy}}\right]$	$\frac{0}{2}=0$	$\frac{1}{3}$
girl	$\frac{0}{2}=0$	$\frac{1}{2}\left[\frac{1\text{-good}}{2\text{-good girl}}\right]$	$\frac{1}{3}$

3 Sentences

⇒ Inverse Doument Frequency : For Every words

Words	IDF
good	$\log_e\left[\frac{3}{3}\right]=0$
boy	$\log_e\left[\frac{3}{2}\right]$
girl	$\log_e\left[\frac{3}{2}\right]$

↳ $\log_e \left[\frac{\text{No. of Sentences}}{\text{No. of Sentences containing word}} \right]$

TF : Term Frequency X IDF : Inverse Document Frequency

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

*

Words	IDF
good	$\log_e[3/3] = 0$
boy	$\log_e[3/2]$
girl	$\log_e[3/2]$

TF-IDF (Calculation)

	Feature 1	Feature 2	Feature 3
	good	boy	girl
Sent 1	$\frac{1}{2} \times 0 = 0$	$\frac{1}{2} \times \log_e[\frac{3}{2}] = 0.2027$	$0 \times \log_e \frac{3}{2} = 0$
Sent 2	$\frac{1}{2} \times 0 = 0$	$0 \times \log_e[\frac{3}{2}] = 0$	$\frac{1}{2} \times \log_e \frac{3}{2} = 0.2027$
Sent 3	$\frac{1}{3} \times 0 = 0$	$\frac{1}{3} \times \log_e[\frac{3}{2}] = 0.1351$	$\frac{1}{3} \times \log_e \frac{3}{2} = 0.1351$

importances of word Captured Using TF-IDF

code

1. ngram_range : tuple (min_n, max_n) ⇒ Ex [1,1] means "unigram"
   [1,2] → unigram & Bigram
   [2,2] ⇒
   "ngram"
2. max_features : If not None, build a vocabulary that Consider, top max_features ordered by term frequency across corpus

```
from sklearn.feature_extraction.text import TfidfVectorizer

# cv = TfidfVectorizer(ngram_range=(3,3),
                       max_features=10)

# x = cv.fit_transform(corpus)
```

Another Example

```
# corpus[0]
```

Out : 'narendra damodardas modi Gujarathi n End

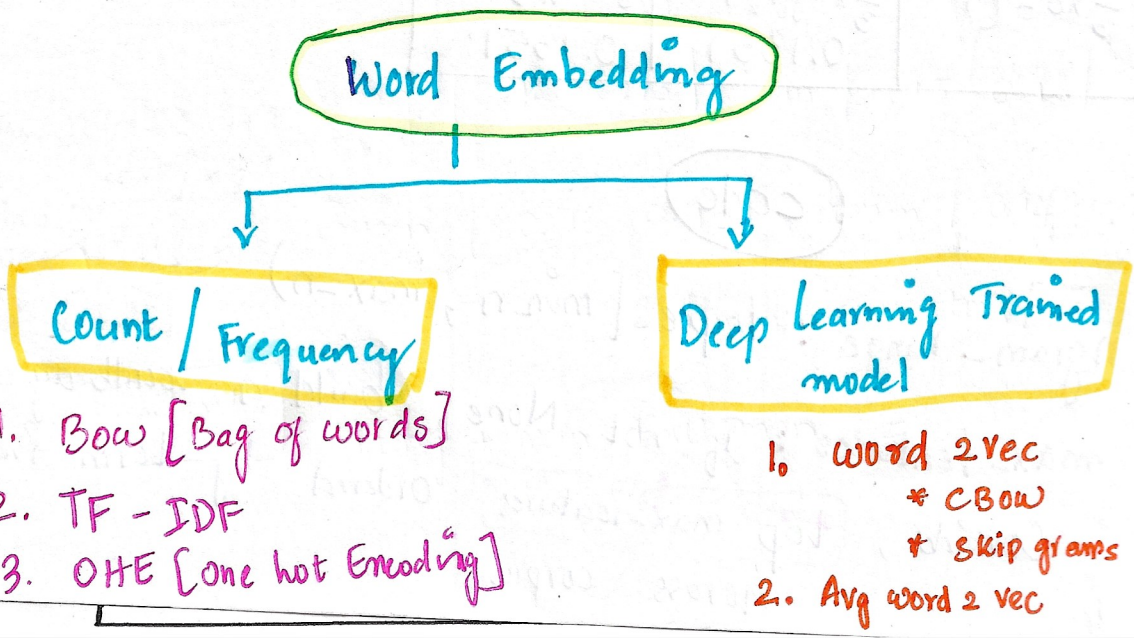
```
# x[0].toarray()
```

Out : array([[0. , 0., 0., 0., 0.70710678
0 , 0.70710678, 0. 0.]])

*Word Embedding

Word Embedding

it is a Techniques which Converts words into Vectors



Word2vec

Feature representation

Vocabulary

f_1	f_2	f_3	f_4	f_5	f_6 ...
Boy	Girl	King	Queen	apple	Mango

* Problem with :- BOW, TFIDF [Semantic Meaning]
  [Missing]
  Similar words [Ex :- good, honest] Synonyms
  * They are creating Sparse matrix
    [0, 1] (or) [0.21, 0]
    BOW TDIDF
  * Because, These create "Huge dimensions" and They have more vectors. it leads to "overfitting".

* Word2vec :
  * Limited Dimensions are created
  * Sparsity is Reduced
  * Semantic remained constant

* Word2vec :- it is divided into two types
  1. CBOW [Continous Bag of words]
  2. SkipGram

Feature representations → created by model

	f_1	f_2	f_3	f_4	f_5	f_6	...
	Boy	Girl	King	Queen	apple	Mango.	
Gender	-1	1	-0.92	+0.93	0	0.1	
Royal	0.01	0.02	0.95	0.96	-0.02	0.01	
age	0.03	0.02	0.7	0.6	0.95	0.96	
food							
⋮							

Relation b/w

300 dimensions [features] → vectors

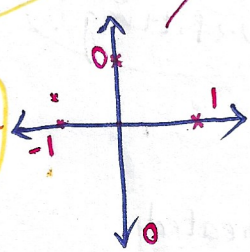
↳ These dimensions are created by "word2vec" algorithm.

"Trained Google done with 3 billion words"

* cosine similarity

∴ Value toward 0 : Similar

Value towards 1 : Not similar



[-1, +1 ; -0.92, +0.93] ∴ They are just opposite, so, we +,-

Ex:- King [0.96, 0.95] Man [0.95, 0.98]

Queen [-0.96, 0.95] women [-0.94, -0.96]

2 dimension vectors

Matched ⟸

By cosine similarity

King - man + women = Queen

* CBOW :- [Continous Bag of words]

Corpus :-

Mr Prudhvi Vardhan is From Warangal city

Context — Context

* Window size = 5

↳ two words before & two words after the input word.

in addition to input word itself.

Training Data

Independent Feature	O/P
Mr, Prudhvi, is, From	Vardhan
Prudhvi, Vardhan, From, Warangal	is
Vardhan, is, warangal city	From

∴ Here Semantic meaning being captured.

By [Text → vectors]

Given To → Fully Connected Layer "ANN" [Artificial Neural Network]

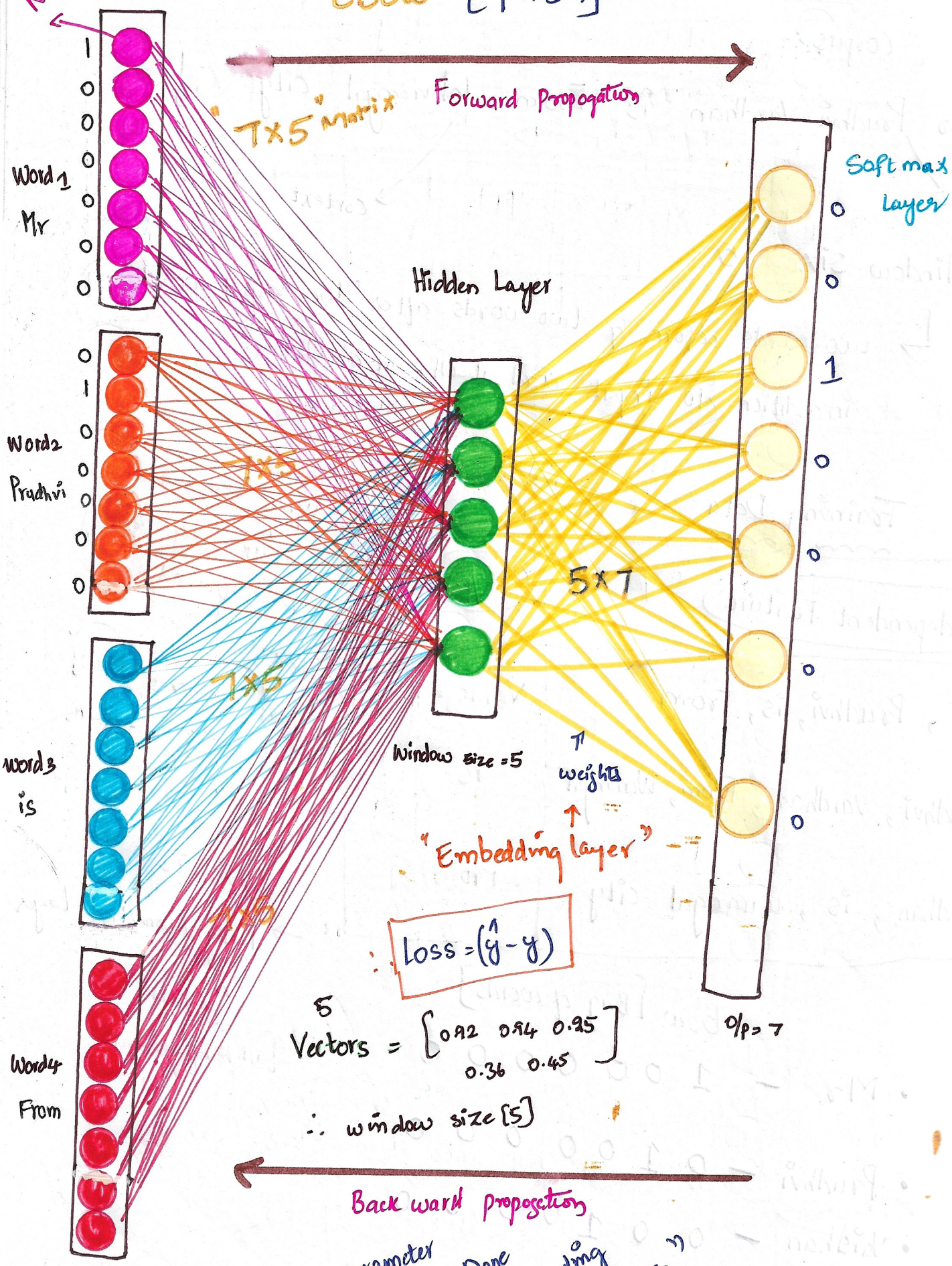
∴ Bow [Bag of words]

- Mr - 1 0 0 0 0 0 0
- Prudhvi - 0 1 0 0 0 0 0
- Vardhan - 0 0 1 0 0 0 0
- is - 0 0 0 1 0 0 0

Here input is = 4 , window size = 5 , O/P = 7 → vectors

Neurons ↓ layer

"CBOW" [Genism]



∴ Hyperparameter Tuning is Done By Extending "window size"

Skip Gram

In the CBow model, the Distributed representations of Context [or Surrounding words] are combined to predict the Word in middle.

* But..., In SKIPGram model, the distributed representation of input Word is Used to "Predict the Context".

Ex:- it Exact Opposite of "CBow"

corpus : Mr Prudhvi Vardhan is From Warangal city

Window size = 5

context

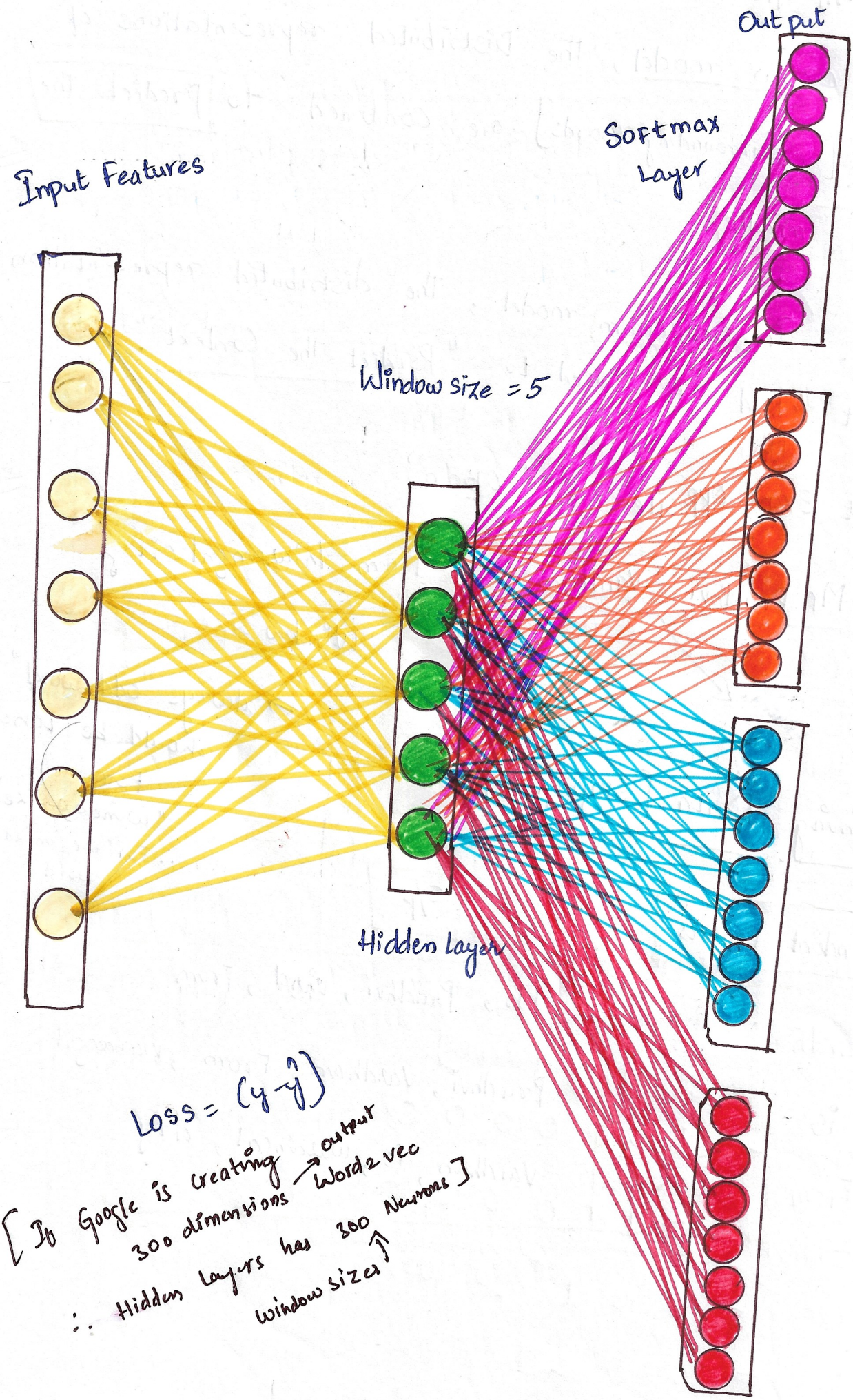
↳ always "odd..word" should be taken For "window size"
∴ For the middle word

Training Data [skip Gram]

Independent Features	O/P
Vardhan	Mr, Prudhvi, is, From
is	Prudhvi, Vardhan, From, Warangal
From	Vardhan, is, Warangal, city

∴ BIGGER DATA = SKIPGram , Short/small Data = CBow

Here input features = 7 , Window size = 5 , O/p = 4



Loss = (y − ŷ)

[If Google is creating 300 dimensions → output Word2vec

∴ Hidden Layers has 300 Neurons]

window size

Code

* CBOW / Skipgram	Based on Transformer Architecture	Based on LSTM
→ Word2vec	→ BERT	→ ELMo
→ Glove	→ GPT	
→ fastText		

→ Gensim → topic modelling, similarity retreival with large corpora.
in NLP, IR [information retreival]

```
# ! pip install genism

import genism
from genism.models import word2vec, KeyedVectors

# import genism.downloader as api
# wv = api.load("word2vec-google-news-300")
# vec = wv["king"]
# vec
```

Google Pretrained model has trained with billion of words with 300 dimensions.

out: array([1.2759 2.97851 8.609 1.396 ...])

[300 vectors].

Similarity b/w words.

wv.similarity ("great", "good")

Out: 0.729

wv.similarity ("jesus", "christ")

Out: 0.637

Semantic meaning can be done :- option by "most-similar".

wv.most_similar ("good")

Out: [("great": 0.729
"bad": 0.719
"terrific": 0.683
"decent": 0.688
"nice": 0.683
"Excellent: 0.644)]
⋮

wv.most_similar ("jesus christ")

Out: [("jesus christ": 0.639
"christ": 0.637
"cuzz_u": 0.634
"jesse": 0.632
"lmfao": 0.621)]
⋮

vec = wv["king"] - wv["man"] + wv["women"]

vec

Out: 300 vectors .. ..
0.236 0.834 0.638 0.843 - ..

wv.most_similar([vec])

Out: [("king": 0.647
queen : 0.535
women :
king

# wv.most_similar(positive = ["king", "woman"],
negative = ["man"], topn = 5)
↳ Top 5 numbers

Out: [("queen": 0.7118
'monarch': 0.6182
'princess : 0.590
crown_prince : 0.549
'prince" : 0.537)]

Project :- Spam Classification

```
import Pandas as Pd
```

Seperate by "TaB"

```
# df = Pd.read_csv("Spam_Collection", sep="/t", names=
                   ["Label", "messages"])
# df.head(5)
```

Out:

Label	messages
ham	Go until point, crazy, Available Only
ham	Ok jar, joking
spam	Free Entry, Coupons Available
ham	You don't Know Brother
ham	Want Prizes Will Get Frend

Text cleaning :

```
import re
import nltk
From nltk.stem import PorterStemmer
From nltk.corpus import stopwords

# ps = PorterStemmer()

# corpus = []
    for in range (len (df)):
        rp = re.sub ("[^a-zA-Z0-9]", " ",
                     df["messages"][i])
```

```
rp = rp.lower()
rp = rp.split()
rp = [ps.stem(word) For word in rp if not
        word in set(stopwords.words("english"))]
rp = " ".join(rp)
corpus.append(rp)
```

```
# corpus
```

Out: [go joury point . . . . .
- - - - - - - - - - ——
- - - - - -]

Vectorization

```
from sklearn.feature_extraction.text import CountVectorizer

# cv = CountVectorizer()

# X = cv.fit(corpus).toarray()

# X.shape()
```

Out: (5572, 7248)

```
# y = pd.get_dummies(df["Label"], drop_first = "True")
```

→ Here "ham" column got deleted, So, we have only "spam" column

```
# y.shape()
```

Out: (5572, 1)

Train Test Split

```
from Sklearn.model_selection import train_test_split

# x_train, x_test, y_train, y_test = train_test_split (x, y,
                                     test_size = 0.3, random_state = 29)

# x_train.shape
```

Out: (3900, 7248)

```
# x_train
```

Out:
```
array ([[[0, 0, 0 ..., 0, 0, 0],
        [0, 0, 0 ... 0, 0, 0],
        [0.0, 0 ... 0 ., 0.0].
```

```
# x_test
```

Out:
```
array ([[[0, 0, 0 ... 0, 0, 0],
        [0, 0, 0 ... 0, 0, 0],
        [0, 0, 0, ... 0, 0.0],
```

```
# y_train.shape
```

Out: (3900, 1)

y_train

Out :

	spam
4480	0
3887	0
⋮	
1653	0
2607	1
2732	1

3900 rows x 1 column

y_test

Out :

	spam
534	0
2279	1
⋮	
4159	0
1504	1
2946	1

1672 rows x 1 column.

Modelling

```
from sklearn.naive_bayes import MultinomialNB

# model = MultinomialNB()

# model.fit(x_train, y_train)
```

Out : MultinomialNB()

Prediction

```
# ypred_train = model.predict(x_train)

# ypred_test  = model.predict(x_test)
```

Evaluation

```
from sklearn.metrics import accuracy_score

# print("    Accuracy:", accuracy_score(y_train, ypred_train)
# print("test Accuracy:", accuracy_score(y_test, ypred_test)
```

Out: Train Accuracy : 0.9928

Test Accuracy : 0.9856