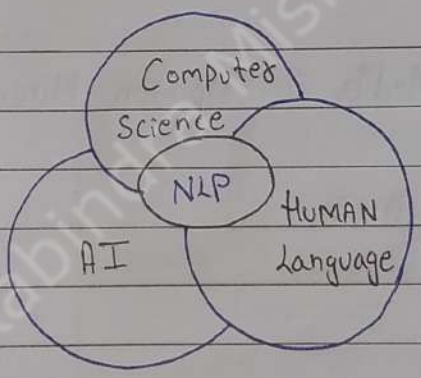
(Natural Language Processing)

NLP

PAGE No.
DATE / /

Introduction : (Wikipedia)

Natural language processing is a subfield of linguistics, computer science and AI concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data.



NLP Diagram.

This field new and it has been just 50 years while 10 years ∵ it has started booming (current Year 2024).

- Natural Language : Language that has naturally evolved in humans.
  It is different than constructed or Program language.

Job Role : NLP Engineer

PAGE No.
DATE / /

Main Objective : Making communication with machines using natural language for easier use.

Real World Applications.

1) Contextual Advertisements
{displaying ad's based on your contextual info such as post, profile }

2) Spam filtering of Emails based on Contents

3) Social Media : Opinion Mining, Removing adult Conte

4) Search Engines.

5) ChatBots.

Common NLP Tasks.

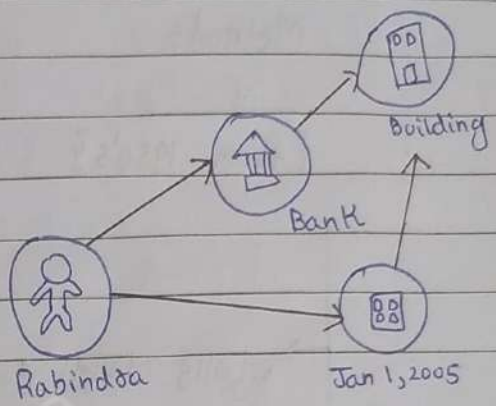
(1) Text / Document Classification	(2) Sentiment Analysis.
(2) Information Retrieval.	(3) Parts of Speech Tagging

⑤ Language Detection and Machine Translation

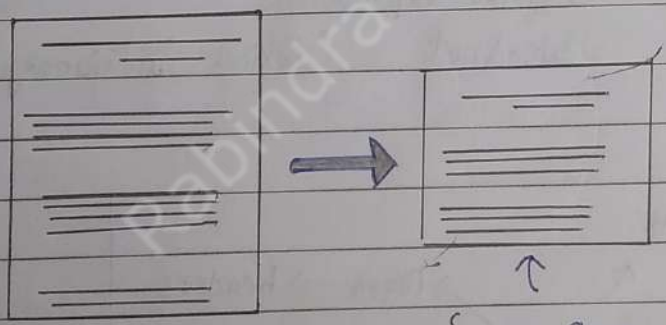
⑥ Conversational Agents (Alexa).

⑦ Knowledge Graph and QA Systems.

(Advance Level)



⑧ Text Summarization



LTA Algorithm

{ only imp Points }

⑨ Text Generation { on Keyboards }

⑩ Spell checking and Grammar Correction

⑪ Text Parsing :

Sentence → verb → "Tom"
→ Noun → Pronoun "ate"

⑫ Speech to Text :

PAGE No.
DATE / /

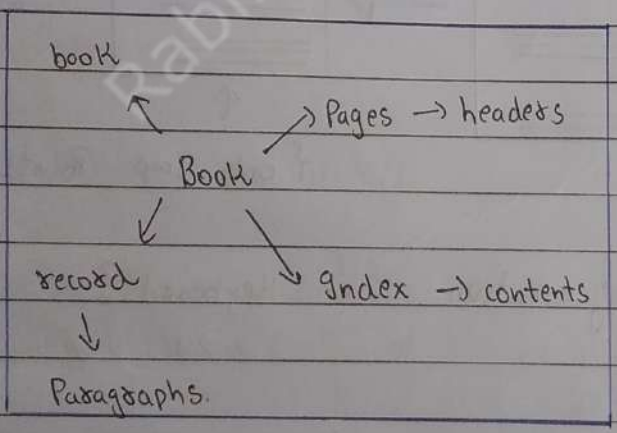
APPROACHES To NLP {started in 1950's}

① Heuristic Methods → {from 1950's}

② Machine Learning Based Models. → {from 2000's}

③ Deep Learning Based Models → {from 2010

- अनुमान लगाना (Mental Shortcut)
- Count (+ve, -ve) words for sentiment Analysis.
- Example & Regular Expressions
  Wordnet (Lexical Dictionary).



Open Mind Common Sense.

Advantages:
① Quick
② Error is not so large
③ Used in current Scenario.

PAGE No.
DATE / /

① Machine Learning Approach

(Big Advantage :) No need to make rules based on open approach and datasets (Most widely used).

Convert text to vector (ie number)

Algorithms Used

① Naive Bayes
② Logistic Regression
③ SVM
④ LDA
⑤ Hidden Markov Models.

② Deep Learning Approach

① Converting numbers to text or vice-versa may loose sequential information of texts. (ie) sequence of words if we use ML model.

② Converts or generate Feature Selection.

i) RNN
ii) LSTM (Long Short Term Memory)
iii) GRU (Text Generation)
iv) CNN (Image)
v) Transformers (Google).
vi) Autoencoders (Set of Two neural networks) (Encoder, Decoder).

NLP is Difficult

PAGE No.
DATE / /

Challenges.

1. **Ambiguity** eg: I never tasted a cake quite like that one!.

2. **Contextual Words.**

eg: I ran to the store because we ran out of milk

3. **Colloquialisms** and **Slang**
eg: Cake Walk , pulling your leg.

4. Synonyms

eg:-> What!! , What?

5. **Spelling Errors.**

6. Diversity of Languages.

NLP Pipeline

A set of steps followed to build an end to end NLP software.

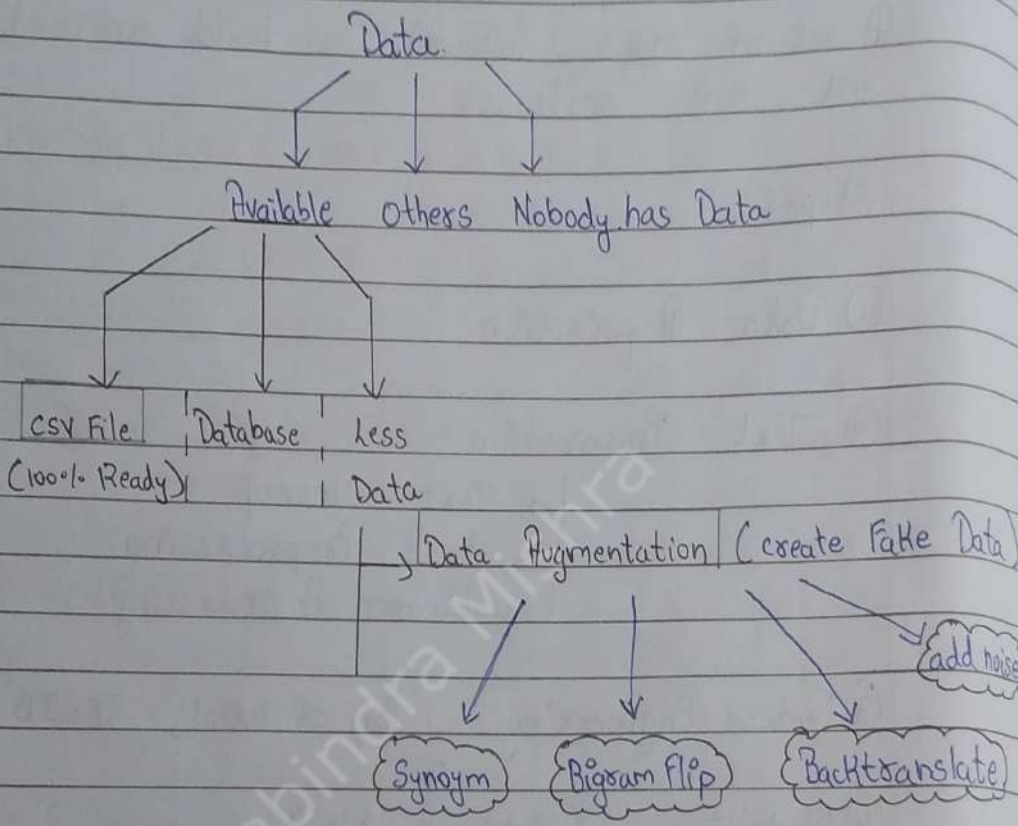
Steps:

1. Data Acquisition
2. Text Preparation
   - Text cleanup
   - Basic Preprocessing
   - Advance Preprocessing
3. Feature Engineering (Bag of Word, TF-ID)
4. Modelling (Model Building)
5. Deployment.
   - Monitoring
   - Model Update

Important:

1. Sometimes Pipeline can also follow the Non-linear trends.
2. Deep Learning Pipelines are slightly Different.

Data Acquisition.



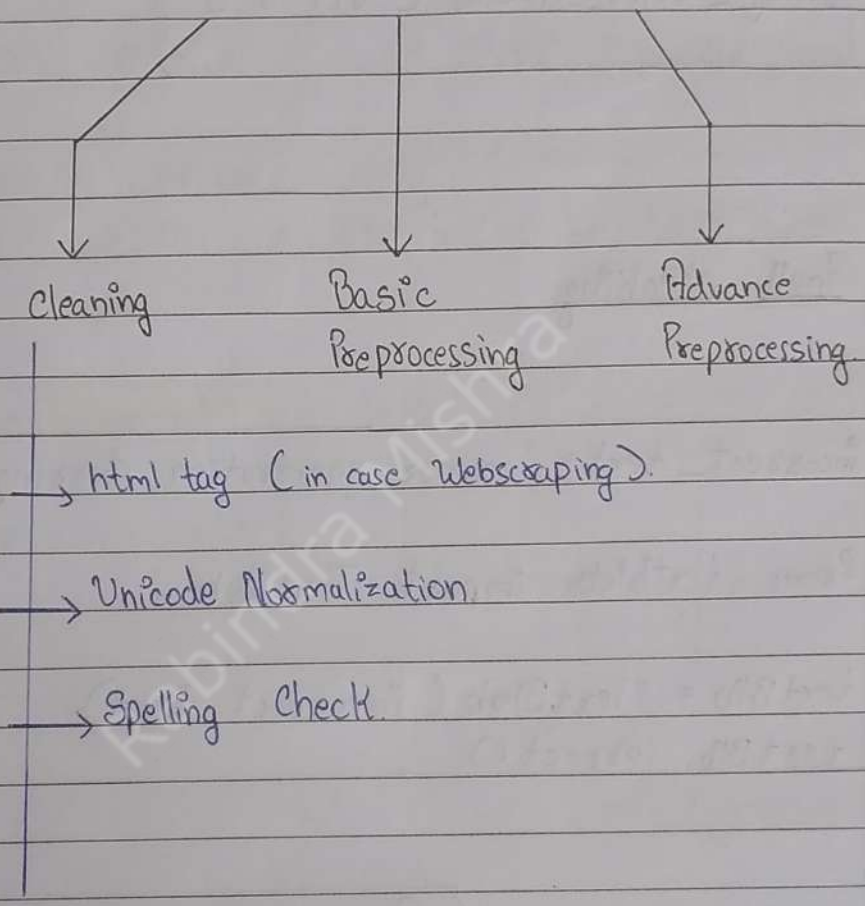
Others :->

- Public Dataset
- Web scraping -> (beautiful Soup)
- API
- Pdf
- Image
- Audio (speech to Text).

API : FastAPI, Flask, DJANGO, Requests, Bottle.

PAGE No.
DATE / /

Text Preparation :



Code:

```
import re
def clean(Data):
    p = re.compile(r'<.*?>')
    return p.sub(' ', data)
```

#gt will remove all this.

While web scraping we also extract emoji's but need to convert them into machine understandable language.

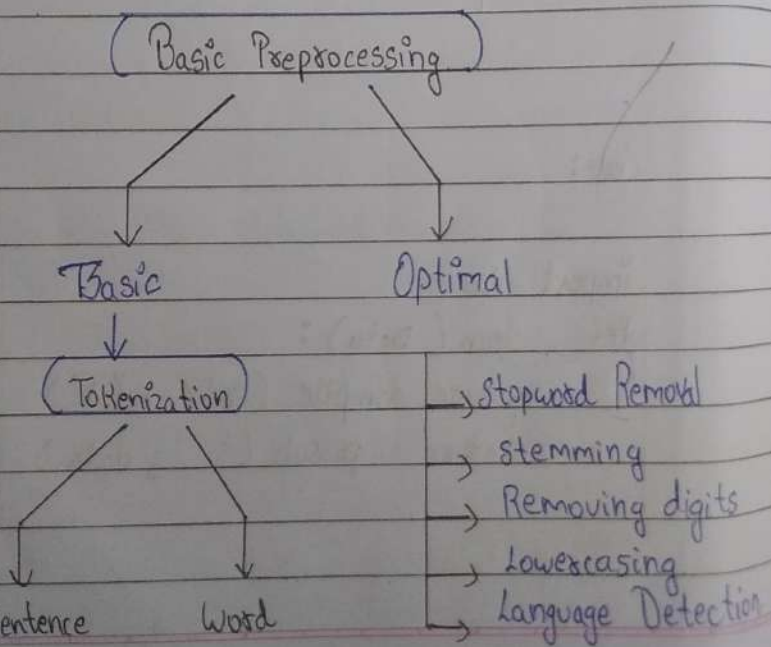
```
emoji_text.encode('utf-8')
```

Spell checking

```
incorrect_text = 'maner ggeneration durringg'

from textblob import TextBlob.

textBlb = TextBlob(incorrect_text)
textBlb.correct()
```



PAGE No
DATE / /

Understand Tokenization using Code

```
dummy = "Lorem Ipsum. . . . . "

from nltk.tokenize import sent_tokenize, word_tokenize

sents = sent_tokenize(dummy).
        ↳ It will break paragraphs into sentences.
```

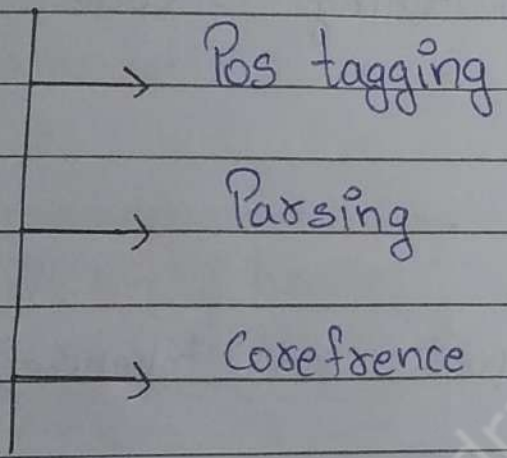
Word Tokenization

```
for sent in sents:
    print(word_tokenize(sent))
```


Stemming :

Different words having same meaning
eg: Dance, Dancing only difference is tense.
Thus in stemming we convert them into root word.

➡ Advance Preprocessing.



In Parsing we create a parse Tree for syntactic combination.

PAGE No.
DATE / /

FEATURE ENGINEERING

Basically converting texts to numbers because machines can process number in a more better way.

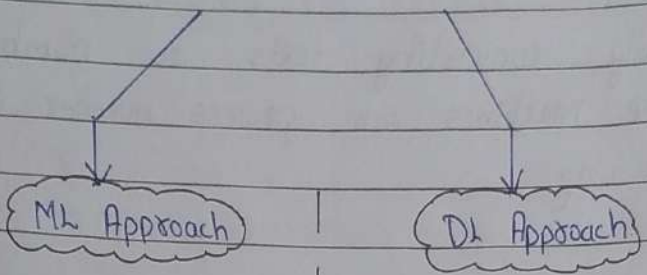
Example of feature engineering

Review_Text	Sentiments
	-1 (negative)
	1 (positive)

No. of +ve words	No. of -ve words	Neutral
5	3	2

Final Outcome : Positive Review (But these are basic)

Feature Engineering

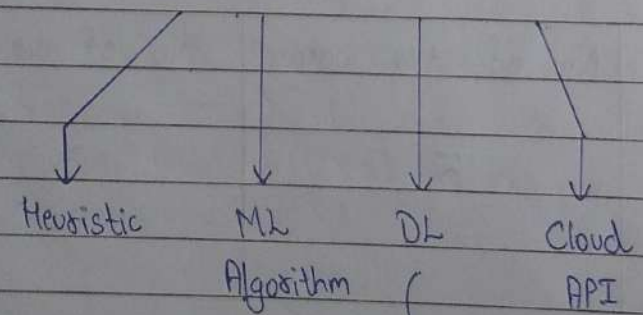


Advantage: Interpretability
Disadvantage: Create Feature on own.
Feature created by own need not be helpful for model.

Advantage: No manual feature creation
Disadvantage: No interpretability is you never know why model is performing good or bad.

MODELLING

Training the Model by applying algorithms.



Approach depends on
① Amount of Data
② Nature of Problem.

BERT: A Transformer which is already trained on 40GB Data on internet.

PAGE No.
DATE / /

If enough data is not present use Heuristic approach.

Deep Learning Requires very large amount of Data.

Evaluation of Model:
- Intrinsic Evaluation (good accuracy)
- Extrinsic Evaluation (Based on users)

Deployment

- Deploy
  - API (On Cloud Service)
  - ChatBot
  - Android App
- Monitor
  - Monitoring as Dashboard
  - Using Graph
  - KPI Cards.
- Update
  - update as per Location / Environment

#Day 3 100 Days of Machine Learning

PAGE No.
DATE 12/7/24

Text Preprocessing

1. Lowercasing
2. Remove HTML Tags
3. Remove URL's
4. Remove Punctuation
5. Chat word treatment
6. Spelling Correction
7. Removing Stopwords
8. Handling Emoji's
9. Tokenization
10. Stemming
11. Lemmatization
12. Assignment.

Lowercasing:

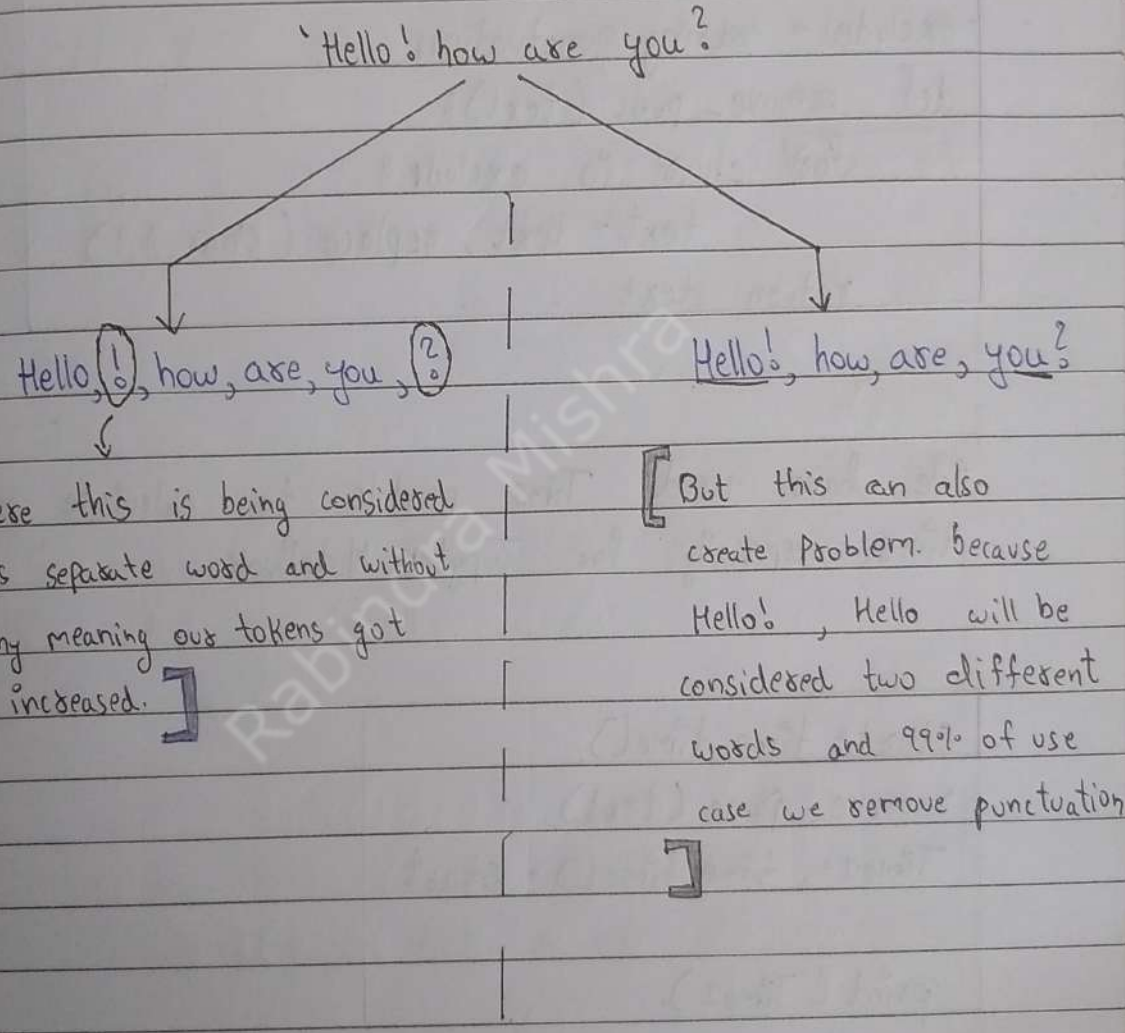
:: Python is a case sensitive language we convert our text either in lower or uppercase Because (Tram, tram) in such cases Python will identify this as two different words.

```
df['review'].str.lower()
```

PAGE No.
DATE / /

Remove Punctuation

It is necessary because whenever we do tokenization our tokens must be appropriate.




```
import string, time
string.punctuation
```

O/P :-> `!"#$%&'()*+,-./:;<=>?@[\]^_'{|}~`

↑ All this include total punctuations.

PAGE No.
DATE / /

Code:

```
import string, ~~re~~, time
string.punctuation

exclude = string.punctuation
def remove_punc(text):
    for char in exclude:
        text = text.replace(char,'')
    return text
```

We have used Time module to calculate execution time especially for Jupyter NoteBook.

```
start= time.time()
remove_punc(text).
Time1= time.time() - start

print(Time1).
```

But the above method is a slower approach.

Method 2:

```
def remove_punc1(text):
    return text.translate(str.maketrans('','',exclude)
```

$

function

PAGE No.
DATE / /

df. sample(5)

o/p: #gt will give any 5 random sample from Datasets.

PAGE No.
DATE / /

Chat Word Treatment

eg:→

① ASAP ② GN ③ lmaO ④ imho.

We need to replace this with Good Night.

upper() : Returns all characters in uppercase.

Code:

```
Chat_words = {'THX': 'Thank You', 'ASAP': 'As soon as Possible',
              'WB': 'Welcome Back' }.

def Chat_conversion (text):
    new_text = []
    for w in text.split():

        if w.upper() in chat_words:
            new_text.append (chat_words
                             [w.upper()])
        else:
            new_text.append(w)

    return " ".join(new_text).
```

Spelling Correction

Eg: "Please read the notebook and also like ntebook."

We can use (textblob) for this. / There are different other libraries.

Removing Stop Words

a the of are my } All these words are used for sentence formation

But donot contribute to the meaning much so for different tasks such as "sentiment analysis" we remove this words.

We use (NLTK.)
↳ It contains list of stopwords.

Code:

```
from nltk.corpus import stopwords
stopwords.words('english').
```

#We need to specify language.

If we want Pos tagging we don't remove stopwords.

Code:

```
def remove_stopwords (text):

    new_text = []

    for word in text.split():

        if word in stopwords.words('english'):
            next_text.append('')

        else:
            new_text.append(word)


    x = new_text[:]
    new_text.clear()
    return " ".join(x).
```

remove_stopwords ('probably my-all time favourite movie, a story of selflessnes and saccifice')

HANDLING EMOJIS


```
U"\U0001F600-\U0001F64F"
```

↳ This is unicode where U denotes unicode string range between U0001F600 - U0001F64F

#Printing all emojis in Python 3.x

```
emojis = [ chr(code) for code in range(0x1F600, 0x1F610)]
print(" ".join(emojis)).
```

Method 2 :→ Replacing Emoji.

```
import emoji
print(emoji.demojize('Python is 🔥'))
```

O/P: Python is : Fire

Tokenization

Process of breaking the text into smaller parts called as tokens.

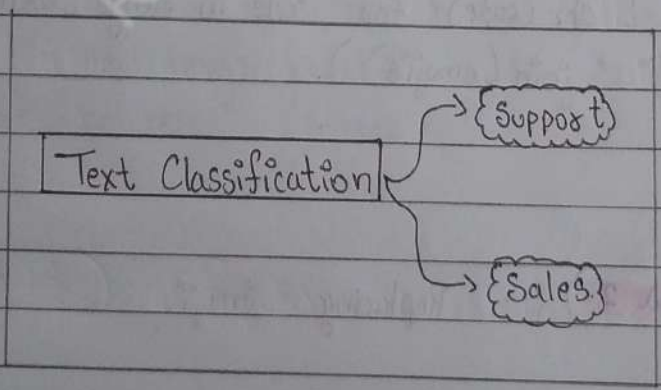
I am an INDIAN

[I , am, an , INDIAN.]

This is called as word tokenization.

Suppose we have Text and we need to do text classification.

and we need to classify the class based on text.



Problem: I am new in new delhi.

In this case it will create 2 tokens for "new" but the second new is for new Delhi.

Problems during Tokenization:

1) Prefix: character(s) at the begining	$("¿
2) Suffix: character(s) at the end	Km),..!"
3) Exception: Special-case rule to split a string into several tokens.	U.S, Let's
4) Infix: character(s) in between.	---/---

eg: ① 12Km ie 12 & Km must be tokenized differently

12 Km

② U.S (X) Here U.S must me tokenized as a single entity.

U S

PAGE NO.
DATE / /

Using Split() Method:

(1) Sent1 = "I am living in Mumbai"
Sent1.split()

O/P:→ ['I', 'am', 'living', 'in', 'Mumbai']

Problem:

(2) Sent3 = "I am going Delhi!"
Sent3.split().

O/p: ['I', 'am', 'going', 'Delhi!']
↳ This is the problem because while analyzing we want ['Delhi', '!']

Method 2) Regular Expression

```
Import re
sent3 = 'I am going to Delhi!'
tokens = re.findall("[\w]+", sent3)
tokens
```

PAGE NO.
DATE / /

Other Examples :

sent5 = " I have a Ph.D in A.I "
sent6 = " help! We're mail us nk@gmail "
sent7 = " A 5Km ride cost $10.50 "

Problem with nltk

① sent6 it breaks email into different tokens
② sent7 5Km does get tokenizes differently.

Difference between math_operations.py

```
def add(a,b):
    return a+b

def subtract(a,b):
    return a-b

Class Calculator:
    def multiply(self,a,b):
        return a*b
```

Collections of functions, class, variables.

Object= math_operations.Calculator().

Method 3 — SPACY

```
import spacy                                  → nlp library
nlp = spacy.load('en_core_web_sm')            → english dictionary

doc1 = nlp(sent5)
doc2 = nlp(sent6)                             → convert sentence to documents
doc3 = nlp(sent7)
doc4 = nlp(sent1)
```

```
for token in doc4:
    print(token)
```

- for doc3 spacy works betters. # gt does not tokenizes 5KM differently.

- # for doc2 also spacy works better.

For some cases spacy will work better while in other cases nlp.tokenize will work better.

No Technique is Fullproof.

nltk: Module | nltk.tokenize : Submodule

word-tokenize & Class of nltk.

PAGE No.
DATE / /

STEMMING

In grammar, inflection is the modification of a word to express different grammatical categories such as tense, case, voice, aspect, person, number, gender and mood.

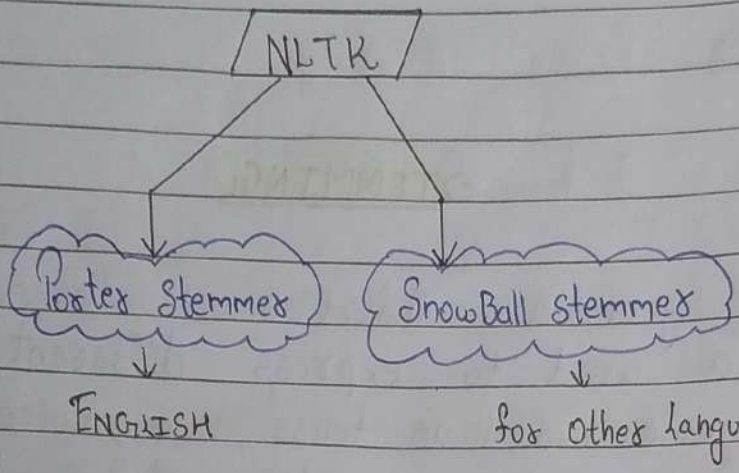
Stemming

"Process of reducing inflection in words to their root forms such as mapping a group of words to the same stem even if the stem itself is not a valid word in the language"

These are mainly used for Information Retrieval System. eg: Google.

Suppose you are Searching: fish, fishing, fishnet.
But whenever you are searching any one of them results must be displayed for all ie include all for wide explorations

- We Use Stemmer (Algorithm) for stemming.



Output after stemming would sometimes not be even a valid english word. This is the biggest problem with stemming.

Example :→

```
from nltk.stem.porter import PorterStemmer

ps = PorterStemmer()

def stem_words(text):
    return " ".join(ps.stem(word) for word in
                    text.split() )

text = "Dance like Dancer"
```

Stem_words(text)

o/p :→ Danc like Dancer.

Thus, we need to apply Lemmatization.

LEMMATIZATION

Difference in lemmatization is that the output of lemmatization will always be an english word.
But lemmatization is slow as compared to stemming.

If speed is your concern use stemming.

In Lemmatization root word is called as ~~st~~ Lemma and lemma always belongs to the language.

↳ Here we use Wordnet Lemmatizer (lexical Dictionary).

Lemmatization में Searching होती है। करके Slow है।

MISHRA
PAGE No. RABINDRA
DATE 17/7/24

CHP : 2

TEXT

REPRESENTATION

- Converting text to Number

Famous Quote in ML "Garbage In Garbage Out"

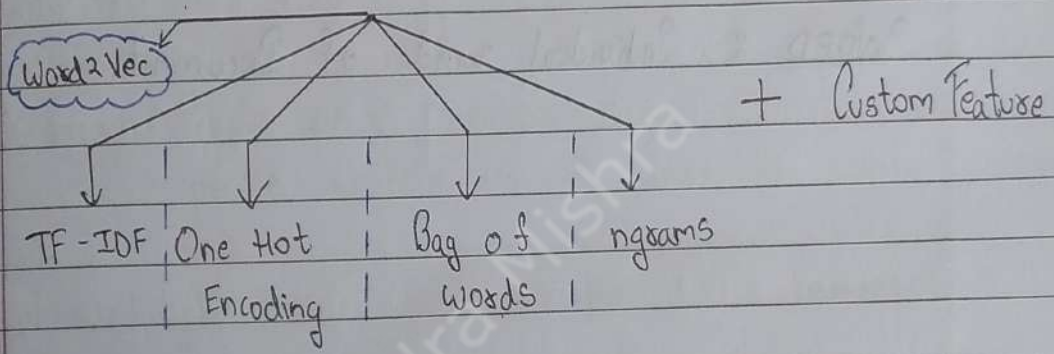
PAGE NO.
DATE / /

What is Feature Extraction from text?

Ans Text vectorization / Text Representation

eg: ① Input :- Audio Clip Output :- Emotions.

② Hello How are you? (sentiment Analysis)



Common Terms

1) Corpus
2) Vocabulary
3) Document
4) Word.

Combining all words of Dataset is called as "Corpus"

① Hello
② Hi
③ Come College

⇒ Corpus: "Hello Hi Come College"

PAGE No.
DATE / /

Vocabulary: Unique Words of Corpus

Document: Inidival Review

Here,	eg:	Hello	1st Document
		Hi	2nd Document

WORD: Individual word of Document.

One Hot Encoding

D1	People watch campusx
D2	Campusx watch Campusx
D3	People write comment
D4	campusx write comment

Corpus: People watch campusx campusx watch campusx people write comment campusx write comment

Vocabulary: People watch campusx write comment.

⇓

V = 5.

T1:

People	watch	campusx	write	comment

People ↱ watch ↱ campusx ↱

$D_1 = [\ [1,0,0,0,0], [0,1,0,0,0], [0,0,1,0,0]\ ]$

Shape of $D_1 = (3,5)$

We are numbering "1" as per above T1 representation.

$D_2 = [\ [0,0,1,0,0], [0,1,0,0,0], [0,0,1,0,0]\ ]$

PAGE No.
DATE / /

Pros
- → Intuitive
- → Easy to Implement

Cons
- → Sparse Array. (causes Overfitting)
- → Variable Document Size (ML won't work)
- → OOV (out of Vocabulary)
- → No capturing of Semantic

Understanding Sparse Arrays.

Array in which only one element is one and rest all are zero's.

eg:

```
[ [1,0,0,0]
  [0,0,1,0]
  [0,1,0,0]
]
```

It becomes difficult for ML model to handle Sparse Array because it contains too many zero's causing overfitting Thus in real world scenarios you won't find using One Hot Encoding eg: V = 50,000.

Understanding Variable Document Size.

PAGE No.
DATE / /

eg: D1 : Hello Rabindra
D2 : Rabindra is Reading.

Vocabulary ⟹ Hello Rabindra is Reading.

For Document D1 :

"Hello Rabindra"

Hello	Rabindra	is	Reading
1	0	0	0
0	1	0	0

D1 : [[1,0,0,0], [0,1,0,0]] (2,4)

For Document D2 :

"Rabindra is Reading"

Hello	Rabindra	is	Reading
0	1	0	0
0	0	1	0
0	0	0	1

D_2 = [[0,1,0,0], [0,0,1,0], [0,0,0,1]] (3,5).

Variable length of D1 and D2.

1. Out of Vocabulary Problem.

		Prediction
D1	Hello Rabindra	0
D2	Rabindra is Reading	1

But suppose you have given Input as "Hi Rabindra" for prediction It would not be able to predict it properly.
Because

Hi is not present in Vocabulary.

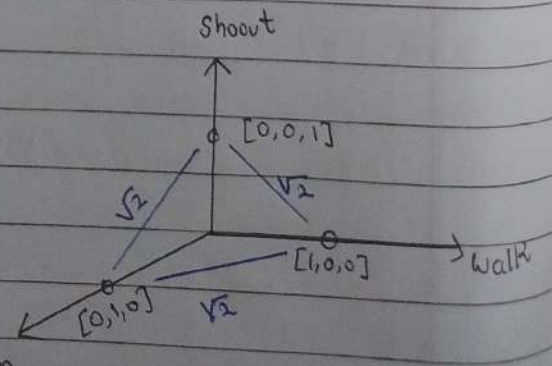
Hello	Rabindra	is	Reading

2. No capturing semantic.

Walk	Run	Shoout

Walk [1, 0, 0] } (vectors)
Run [0, 1, 0] }
Shout [0, 0, 1] }

Plotting Vectors



PAGE No.
DATE / /

Now, although all vectors are equidistant but they differ in meaning thus unable to capture semantic meaning

Because Run and walk are different and has distance of $\sqrt{2}$ but shout also has same distance although shout is completely dis similiar than (Run and walk).

Meaning of words is unable to be represented by numbers.

Bag of Words

#Mostly used Technique in NLP
#Used for Text Classification

D1	People watch	1
D2	Campusx watch	1
D3	People write	1
D4	Campusx write	1

V=4

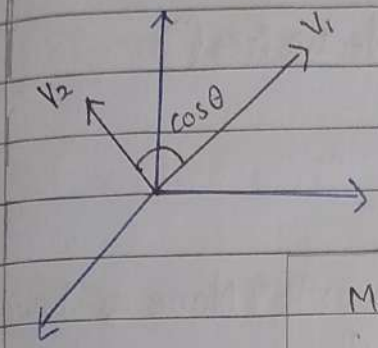
	People	Watch	Campusx	Write
D1	1	1	0	0
D2	0	1	1	0
D3	1	0	0	1
D4	0	0	1	1

Intuition : Same word being repeated same frequency of times (order of words does not matter) (context does not matter)

{ Vdim Plotting Here, V=4. }

Name is given Bag of words due to it is understood that collecting all the words inside a Bag and then assiging it as per Vtable (Vocabulary table).

Intuition : Quick Understanding without Reasoning.



If angle between V_1 and V_2 is small then they will come under one category.

More the angle More is dissimilarity

In this we will use **CountVectorizer()**

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
```

```
bow= cv.fit_tranform(df['text'])
# fitting data and transforming it.

print( cv.vocabulary_ )
        ↳ attribute of cv.

print( bow[0].toarray() )
print( bow[1].toarray() ).
```

#Handles OOV problem.

```
cv.transform(["campusx is Rabindra"]).toarray()
```

"Scikit-learn.org"

PAGE No.
DATE / /

Hyperparameter of CountVectorizer()

CountVectorize (*, stop_words = None, lowercase = True
↓
we can mention language to remove stopwords

preprocessor = None, binary = False, max_features,
n_gram = (1,1)
↳ #unigram

Understand binary = True

If Hyperparameter binary is set to True it means all non zero counts are set to 1

ex:

Vocabulary Table

Rabindra	is	Coding
1	2	0

binary = False

Vocabulary Table

Rabindra	is	Coding
1	1	0

binary = True
(either 1 or 0)

★ If we want to find frequency of words binary = False
★ If we only want to check if word exists or not binary = True.

for sentiment analysis we make binary = True as per Research.

Understanding max_feature

Suppose if, max_feature = 1
In this case it means we only want One feature as an output whose frequency is the highest (eg:) Rabindra would be output

max_feature = 2

↑ Rabindra	↑ is	Coding
2	1	0

In this case here it will only include "Rabindra" and "is" while eliminate "coding"
This is used for eliminating rare occurrence words.

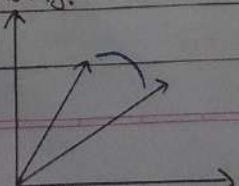
Advantages.	Disadvantages.
→ Simple and Intuitive	→ Sparsity.
→ Fixed Document size	→ OOV ('Hi' is new word) In this we ignore new word. However, Hi is bringing new information.
→ OOV	
→ Some semantic meaning is preserved	→ Ordering is not considered.

Understanding Major flaws of Bag of Words

Eg:→ This is a good movie
This is not a good movie.

} In both the sentence there is only a minor difference in vocabulary.

In this case Bag of Words will ignore the "not" word and both the vectors would be very much close showing similarity.



"But meaning is very much different although vocabulary is very much similiar"

This problem is been handled by

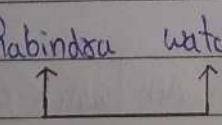
Bag of N grams.

N-grams :

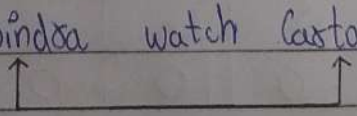
D1	Rabindra watch Cartoon	1
D2	Cartoon watch Cartoon	1
D3	Rabindra write comment	0
D4	Cartoon write comment	0

Bigram:

Take two words at a time



Tri-gram:



Similarly, n-gram

Bag of bi-gram:

Here, One vocabulary is combination of Two words.

(eg:)

D1	Rabindra watch Cartoon	1
D2	Cartoon watch Cartoon	1
D3	Rabindra write Comment	0
D4	Cartoon write Comment	0

Vocabulary:

Rabindra watch	watch Cartoon
Cartoon watch	watch Cartoon
Rabindra write	write Comment
Cartoon write	write Comment

Vocabulary Table:

	People Watch	Watch Campusx	Cartoon watch	watch Cartoon	Rabindra write			
D1	1	1	0	0	0	0	0	0
D2	0	1	1	0	0	0	0	0
D3	0	0	0	0	1	1	0	0
D4	0	0	0	0	0	1	0	1

If that Digram is present in Document increement the count else keep zero.

Trigrams:

people watch campusx | campusx watch campusx | D3 | D4

D1	1	0	0	0
D2	0	1	0	0
D3	0	0	1	0
D4	0	0	0	1

- `CountVectorizer( n_gram_range = (2,2) )`

  ↓

  This means only Bigram

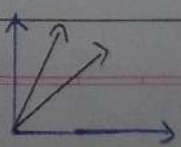
ngram_range = (1, 2) # This means both unigram and Bigrams

* Problem overcomed. *

Bag of Words

	This	movie	is	very	good	not
D1	1	1	1	1	1	0
D2	1	1	1	0	1	1

Now, vectors are very close.



Using Bigram Approach :

This movie is very good
This movie is not good

Using Bigram Vocabulary :->

	This movie	movie is	is very	very good	is not	not good
D1	1	1	1	1	0	0
D2	1	1	0	0	1	1

Different Values

Now, vectors are more far away.

Advantages	Disadvantages
1) Able to capture semantic of the sentence	1) For large Datasets
2) Easy Implement.	Unigram → Bigram, Trigram.
	May increase dimension (slows down Algorithm)
	4) No solution for Out of Vocabulary.

PAGE No.
DATE / /

"TF-IDF"

D1	People watch CampusX	1
D2	CampusX watch CampusX	1
D3	people write comment	0
D4	CampusX write comment	0

People	watch	campusx	write	Comment
1	1	1	0	0

UnliHe other Techniques Tf-IDf assigns different "Weights"

Idea: / Intuition

अगर कोई एक word बहुत बार आ रहा है। In a particular Document but it is not occuring many times in whole corpus.
In such secanario that word becomes important for the Document and is given more importance.

TF (Term Frequency)
IDF (Inverse Document Frequency)

Mishra.
Rabindra
PAGE No.
DATE / /

Q.) How to Calculate TF, IDF?

Ans:

$$TF(t,d) = \frac{\text{(Number of occurrences of term } t \text{ in document } d)}{\text{(Total number of terms in document } d)}$$

$$IDF(t) = \log_e \frac{\text{(Total number of Documents in the Corpus)}}{\text{(Number of documents with term } t \text{ in them)} + 1}$$

eg:-> TF (people, D1) = 1/3

{ एक डॉक्युमेंट में People आया है
Total 3 terms है }

TF (campusx, D2) = 2/3

Note: $0 < TF < 1$ #can consider it as probability

$$IDF(\text{campusx}) = \log\left(\frac{4}{3}\right)$$

	IDF
People	log(4/2)
Watch	log(4/2)
Campusx	log(4/3)
Write	log(4/2)
Comment	log(4/2)

Analyzing Trend behind TF-IDF :-

- If the term is more frequent in Corpus :⇒ IDF ↓
  eg: $\log\left(\frac{4}{4}\right) = \log(1) = 0 + 1$
- If the term is less frequent in Corpus :⇒ IDF ↑

TF is basically probality of frequency of element in a Document.

(Vocabulary Table) FORMULA : TF X IDF

	People	Watch	Campusx	Write	Comment
D1	1/3x0.3	1/3x0.3	1/3 x 0.125	0	0
D2	0	1/3x0.3	2/3x0.125	0	0
D3	1/3x 0.3	0	0	1/3x0.3	1/3x0.3
D4	0	0	1/3x0.125	1/3x0.3	1/3x0.3

PAGE NO.
DATE / /

Code Implementation :

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()

rabindra = tfidf.fit_transform(df['text']).toarray()
rabindra.

print(tfidf.idf_)  # Points IDF Values.
print(tfidf.get_feature_names_out())
```

"+1" is added because terms occurring with all documents may get ignored without "+1"

eg $\log(\frac{4}{4}) = 0$ TF x IDF = 0 (ignored)

BRAINSTORMING

Q.) Why do we use log for calculating IDF?

Ans

Suppose, there is one word which is very rare out of 10,000

$$IDF = \left(\frac{N}{n}\right) = \frac{10,000}{1} \quad \log\left(\frac{10000}{1}\right) = 4.$$

Using log will prevent unnecessary Big value for IDF

In such case Tf x IDF while $0 < Tf < 1$ and

IDF ⇓ 1000

in this case IDf will dominate over Tf and then tf will not have any significance.

Advantages	Disadvantages
→ Information Retrieval	→ Sparsity → OOV → Dimension → Semantic

Custom Features

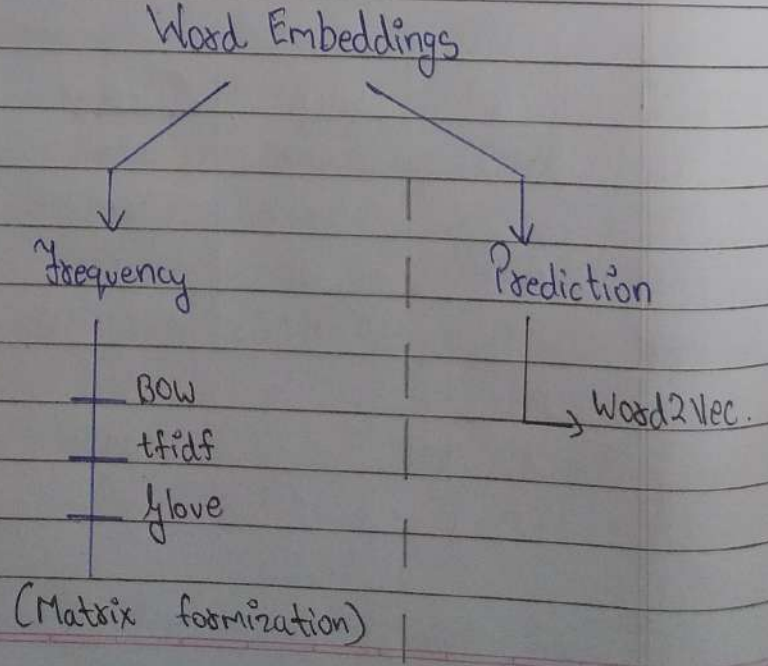
Such as: ① Ratio of +ve/-ve ② Word Count
③ No. of +ve words, -ve words

WORD2Vec (Deep Learning)

Understanding Word Embeddings:

In NLP processing, word embedding is a term used for the representation of words for text analysis, typically in the form of a real-valued vector that encodes the meaning of the word such that the words that are closer in the vector space are expected to be similiar in the meaning

eg: We have a word "King" and we want to convert it to vector ie [24 19]



Q.] What is Word2Vec? (# Complex)

a) Developed by Google engineers in 2013

b) Word2Vec can capture Relationships like → Happy, → Joy

ie it can capture Semantic meaning.

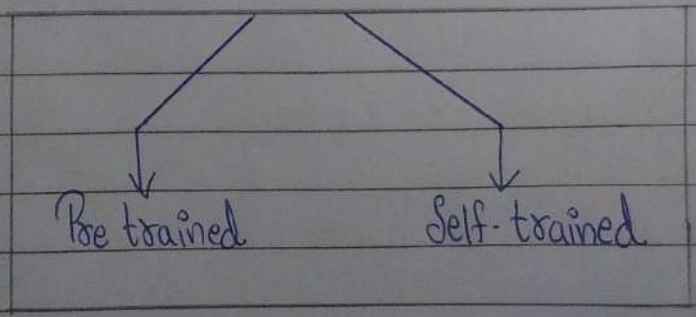
c.) Creates low dimensional vectors usually in range of [100 - 300] as compared to BOW/TF-IDF

d.) In case of BOW/TF-IDF you'll see a very sparse Matrix [0 0 0 2 0 0.]
But In Word2Vec this matrix is more denser ie less ~~non~~ of zero and thus reduces overfitting problem.

e.) It is a deep learning Technique.

DEMO

There are two ways of using Word2Vec.



We will use the pre-trained weights of word2Vec that was trained on Google ~~View~~ News corpus containing 3 billion words. This model consists of 300-dimensional vectors for 3 million words and phrases.

Here, it is trying to say that we will have Vector form of 30 lakh words as a dimension of 300
ie [0 ..... 300]

Thus it makes this word2Vec accurate.

```
import genism
from genism.models import word2Vec, KeyedVectors
```

PAGE NO.
DATE / /

Important Commands

① Pip install
② pip list — Shows all packages install
③ pip show package_name. — search by package name

Pip: prefered Installer program

PyPi: Python Package Index.

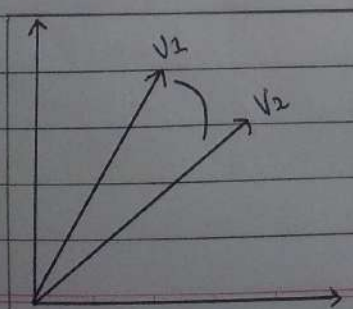
↳ It is a central repository for python programming language.

Pip installs packages from this default repository from PyPi.

Word2Vec

For time being consider 300 Dimensional vector as 2D so we can visualize.

Now, we can use these values to find similarity of two words



We can also perform ① Vector Addition
② Vector Sub
and other vector operations

most_similar() : Function

It returns list of all similar vectors to the input word

eg: model.most_similar('man')

o/p:→ [('woman', 0.76640129...),
('men', 0.8367...),
('teen', 0.7347...),
('girl', 0.7446...),
]

→ vector values

doesnt_match()

Gives oddman out.

eg: model.doesnt_match(['cricket', 'NASA', 'ISRO'])
o/p : Cricket

Vector Operation.

```
vec = model['wicket'] + model['RUN'] - model['Ball']
model.most_similar([vec])
```

```
vec2 = model['INR'] - model['USD']
model.most_similar([vec2])
```

INTUITION of Word2Vec

- It captures semantic meaning
- We need to convert words in such a way that it should capture the meaning

In this we convert words based on different features.

Features.	King	Queen	Man	Woman	Monkey
Gender	1	0	1	0	1
Wealth	1	1	0.3	0.3	0
Power	1	0.7	0.2	0.2	0
Weight	0.8	0.4	0.6	0.5	0.3
Speak	1	1	1	1	0

King [1 1 1 0.8 1]
Monkey [1 0 0 0.3 0]

Understanding Vector Operations

[King] - [Monkey] will give us something close to "Queen"

Problem

1) In real-world instead of 5 we will have lakhs of vocabulary so handcrafting it will be impossible

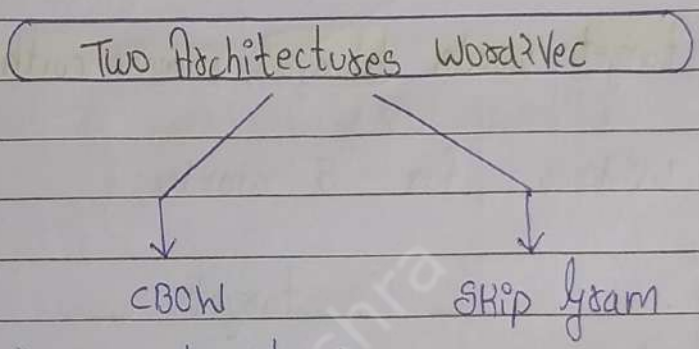
2) Thus we need neural network to create features automatically But disadvantage is you will not know what exactly are the features.

3) It has generally 300 features (300 Dimension)

The underlying assumption of word2vec is that two words sharing similar contexts also share a similiar meaning and consequently a similar vector representation from the model.

(eg:) Football Player took a shot
Hockey Player took a shot.

Here, Football ↔ Hockey is been used for same context.



Both are shallow neural networks.

CBOW Architecture

Here, we try to solve a fake problem and solve the problem as a by product we get our vector.

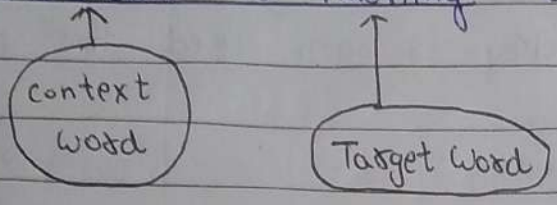
Fake Problem
↳ Solve → Vector → Byproduct

Rabindra is learning Data Science.

To convert all the words into vector we will use a dummy problem.

Assuming a window of 3 words.

Rabindra is learning Data Science



target words always comes with context word

Windows for 5 words

__ __ target __ __

a) Now we will take this window for training our Data

X	Y
~~word~~	
Rabindra, is	learning.

b) Now, we will take another window

Rabindra	is	learning	DataScience

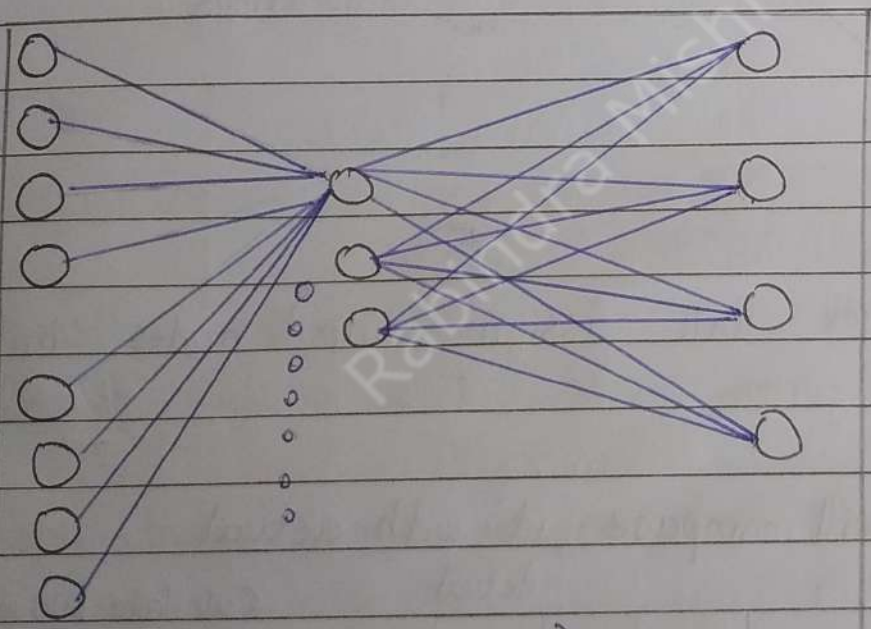
X (Training Data)	Y
Rabindra, is	Learning
is, learning	Datasceince

Then we will convert our Input to "One hot Encoded vectors."

1000
0100
0010
0001

} One Hot Encoded Vectors.

Using this data we will create neural network.

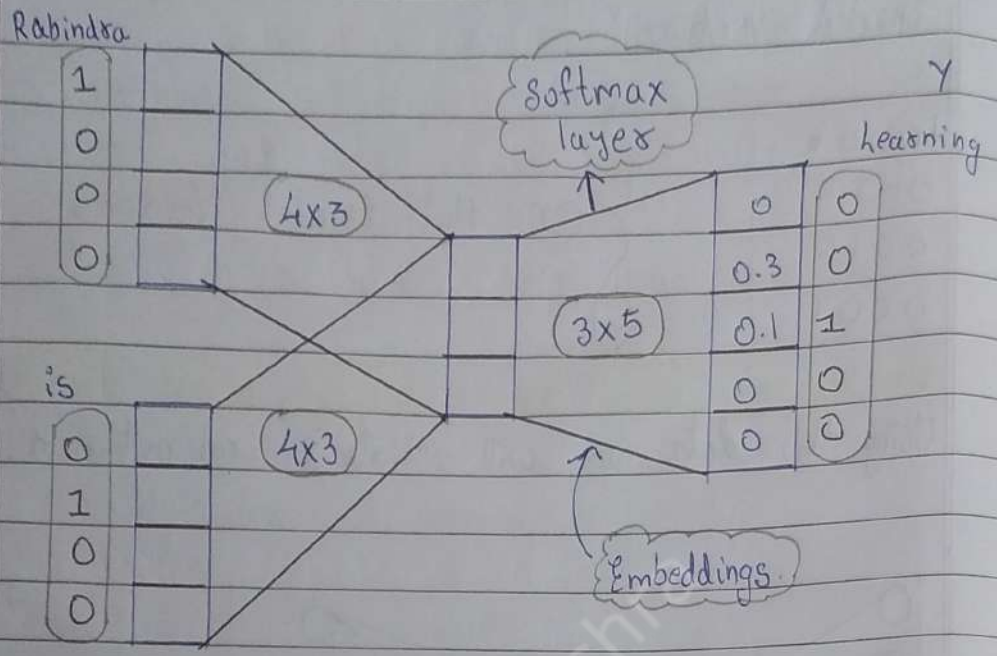


(Hidden layer)

Here 3 nodes are there in hidden layer representing Window Size.

In Output layer we have 4 words represented by indival node

X



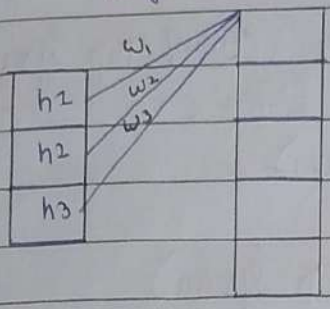
Softmax layer will assign output nodes in terms of probability.

Now, it will compare it with actual

		Actual	Calculate Looss
ie	0	0	
	0.3	0	
	0.1	1	
	0	0	
	0	0	

After calculating loss we will backpropogate for multiple epox. to adjust the weights.

Fully Connected Neural Network



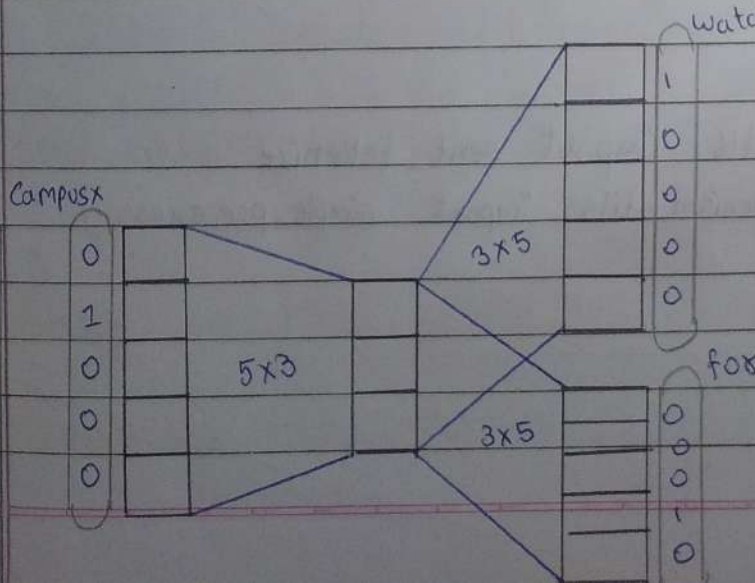
Vector Representation of Rabindra : $[h1*w_1 + h2*w_1 + h3*w_3]$

SKIP-GRAM ARCHITECTURE

- Apposite of CBOW
- In this also we will use dummy Problem

From target word you need to predict | Context word

campusx	→	watch, for
for	→	campusx, data
Data	→	for, Science



From Research it is proven CBOW/SKip Gram

- Small Data (CBOW)
- Large Data (SKip Gram)

Word2Vec Embeddings (Improve)

- → Increase the training Data
- → Increase Dimension of vector
- → Increase window size

"Training your own Model"

```
import genism
import numpy as np
import pandas as pd
import os
```

```
from nltk import sent_tokenize
import genism.utils import simple_preprocess
```

```
story = []
Data = r" C:\user\ Game of throne"
for filename in os.listdir (Data):

        f = open (os.path.join (Data, filename))
        corpus = f.read()
        raw_sent = sent_tokenize (corpus)

        for sent in raw_sent:
            story.append (simple_preprocess (sent))
```

defining our Model structure.

```
model = gensim.models.Word2Vec (
        window = 10
        min_count = 2   # sentence having atleast 2 words
```

```
model.build_vocab (story)   # Vocab build

model.train ( story, total_examples = model.corpus_count,
              epochs = model.epochs )   # training Model
```

- total_examples: tells total sentences in corpus

Accessing our Model

→ model.wv.most_similar ('King')

↳ word2vec

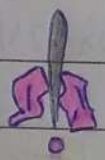
doesnt_match():

→ model.wv.doesnt_match(['jon', 'rikon', 'robb', 'arya'])
O/P :-> 'arya'

Calculating Similarity

1) model.wv.similarity('arya', 'sansa')
0.8534194

2) model.wv.similarity('tywin', 'sansa')
0.2644586

PROBLEM ALERT 

Plotting 100 dimensional vector will be complex thus we will need to reduce it to 2 Dim / 3 Dim.

→ Apply PCA.

Getting Vector Representation of all words

⇒ `model.wv.get_normed_vectors().shape`

(17453, 100)

#Here, we have 17453 words in our corpus.

V.V IMP ② `model.wv.get_normed_vectors()`

(O/P:→)

```
( [  [-0.119, 03176, 0.77, .... 0.138],
     [0.765, 0.117, 0.712, .... 0.551],
     :
     :
     [0.6179, 0.932, 0.003, .... 0.05]
  ])
```

ie 1743 words of 100 Dimension

To see which words are presented for this particular vector.

```
y = model.wv.index_to_key
y
```

for docstring in jupyter Notebook
click (Shift + tab)

PAGE No.
DATE / /

Plotting the graph

↳ Applying PCA

```
from sklearn.decomposition import PCA
pca = PCA(n_components=3)          #3D
X = pca.fit_transform(model.wv.get_normed_vectors())
X[:5]
```

```
import plotly.express as px
```

```
# effect of PCA
X.shape
```

O/P: (17453, 3)

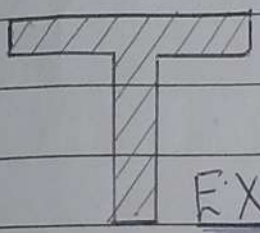
↳ Reduce Dimension

```
fig = px.scatter_3d(X[:100], x=0, y=1, z=2, color=y[:100])
fig.show()
```

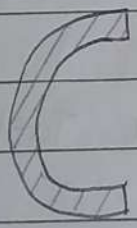
Important Visualization libraries

→ Matplotlib
→ Seaborn
→ Plotly (for interactive Plots, Dashboard)

PAGE No.
DATE / /



EXT

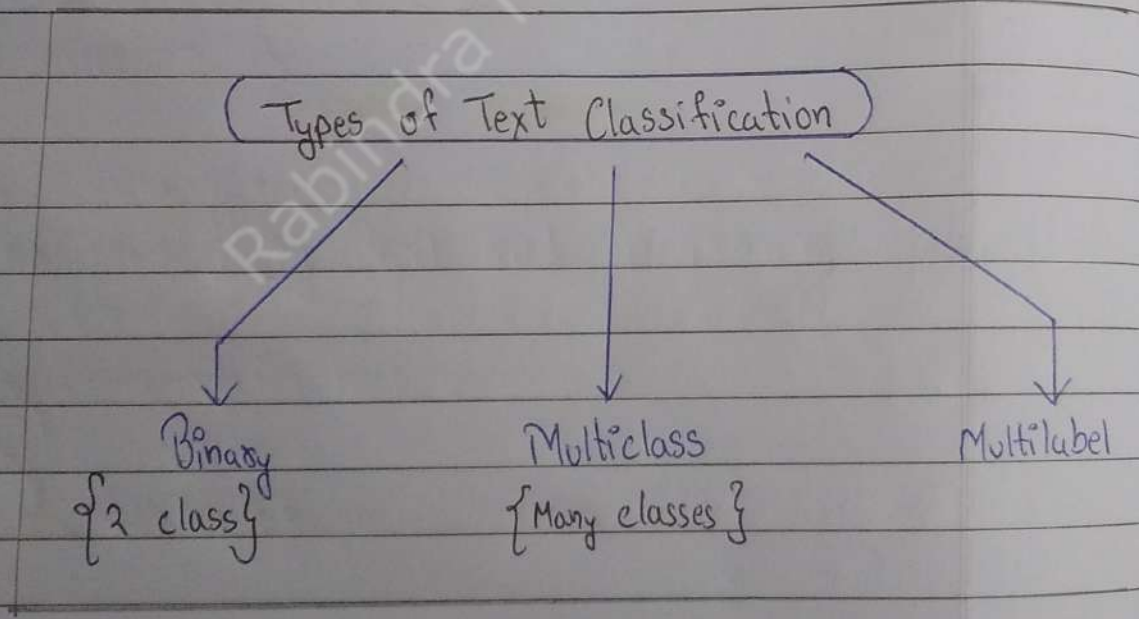


CLASSIFICATION

Classification on Different Data.

- tabular
- Image
- Text

It is a Supervised ML Task

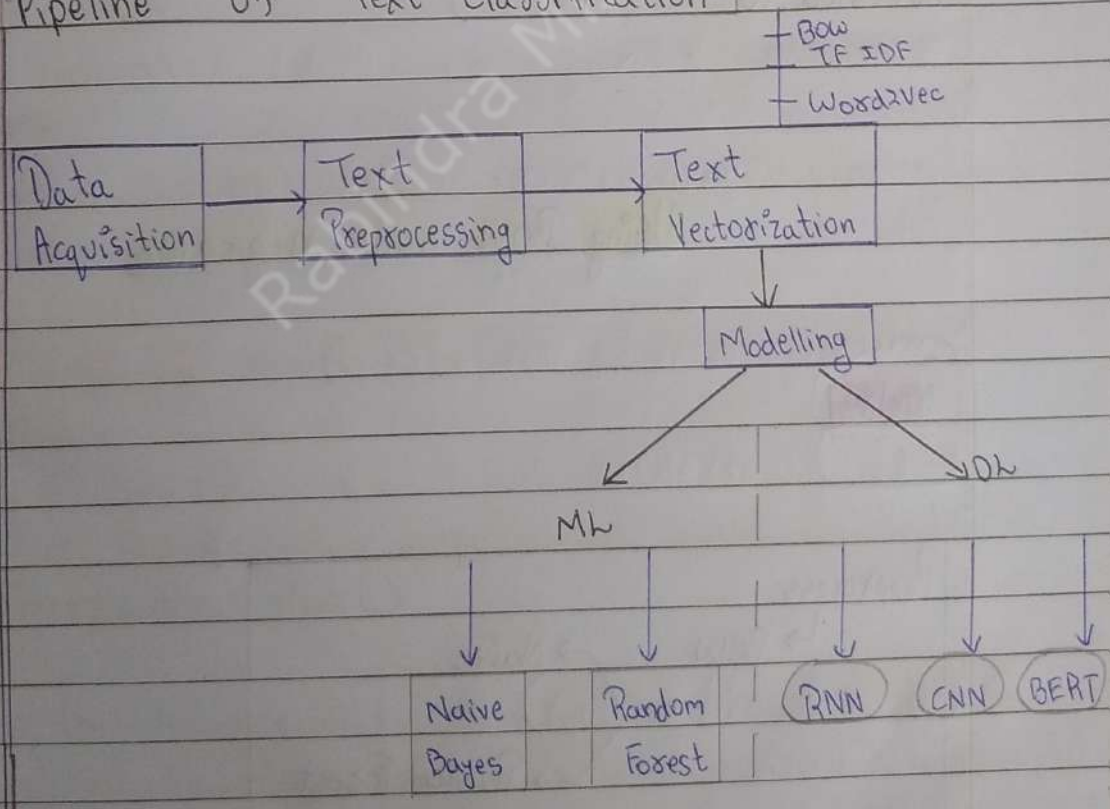


Imp: NLP is mostly used for text classification.

Applications

Email Filtering	Customer Support	Sentiment Analysis for Review
Language Detection	Fake news Detection	

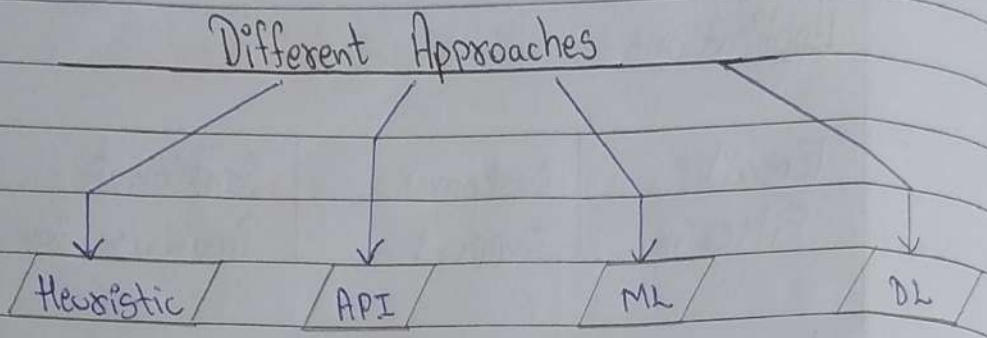
Pipeline of Text Classification



Evaluation → accuracy
→ confusion

BERT: Transformer

Mishra
PAGE No. Rabindra
DATE / /

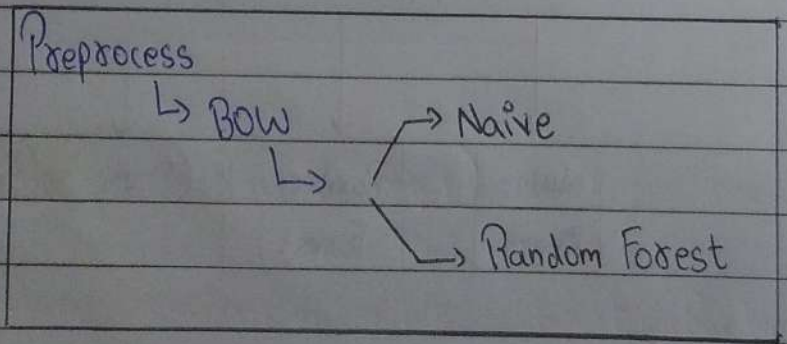


When we don't have enough data we need to use heuristic approach

NLPCloud.io : API

Using BOW and n-grams

Pipeline



```
import numpy as np
import pandas as pd
import os
```

```
df['sentiment'].value_counts()
```

positive : 5028
negative : 4972

```
df.isnull().sum()          # checking is null
```

review 0
sentiment 0

```
df.duplicated().sum()      # checking duplicates
```

17

Fitting our model as BOW

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()

x_train_bow = cv.fit_transform(x_train['review']).toarray()
x_test_bow = cv.transform(x_test['review']).toarray()

print(x_train_bow.shape, x_test_bow.shape)
```

Apply NaiveBayes

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()

# fitting our model
gnb.fit(x_train_bow, y_train)

# Predicting our model
y_pred = gnb.predict(x_test_bow)

#Calculate Accuracy of Model

from sklearn.metrics import accuracy_score,
                            confusion_matrix

accuracy_score(y_test, y_pred)
```

O/P:→ 0.62844266

Apply RandomForest

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()

# fitting Model
rf.fit(x_train_bow, y_train)
```

PAGE No.
DATE / /

```
# Predicting the Model

y_pred = rf.predict(x_test_bow)

# Calculate Accuracy.

accuracy_score(y_test, y_pred)
```

PAGE No.
DATE / /

Using Word2Vec

- Pre trained Word2Vec
  - Vocab
- Train on your own Data → Enough Data

Generally we convert all the words as a vector

example :→ Delhi → [ ] → 300 Dimension

But we havent seen anything getting like eg Document as a vector

ie Document 1 : [ ]

Example → I Love NLP

[ ] [ ] [ ]

Adding all these vectors will give us new vector

[---------]

```
def document_vector(doc):
    # Remove out of vocabulary word

    doc = [word for word in doc.split() if word in
                    model.wv.index_to_key ]

    return np.mean(model.wv[doc], axis=0)
```

```
from tqdm import tqdm
```

tqdm is a popular python library used to display progress bars for loops, making it easy to monitor the progress of lengthy computations.

* Practical Advise *

1) Ensemble Technique
2) Heuristic features
3) Deep Learning

POS TAGGING

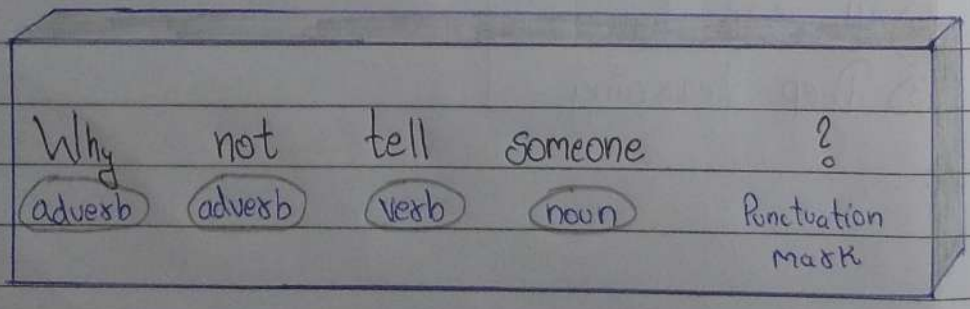
- It is necessary for complex system like chatbots, information retrieval system.

- Here, we will learn about how does POS tagging work internally.

Pos tagging means giving a part of speech to a sentence word by word.

Definition : In simple words, we can say that POS tagging is a task of labelling each word in a sentence with its appropriate part of speech.

In traditional grammar, a part of speech is a category of words that have similar grammatical properties.



Note : It is a preprocessing step.

✯ Major Applications ✯

① Named Entity Recognition
② Question Answering System
③ Word Sense disambiguation
④ Chatbots.

Done using "(Spacy.)"

```
import spacy
nlp = spacy.load('en_core_web_sm')
doc = nlp(u"I will google about rabindra")
```
↳ (This denotes unicode string)

```
doc[0]
```
O/P :-> I

`doc[0].pos_`	# I is pronoun
O/P : 'PRON'	
`doc[0].tag_`	# Present Participle.
O/P : 'PRP'	

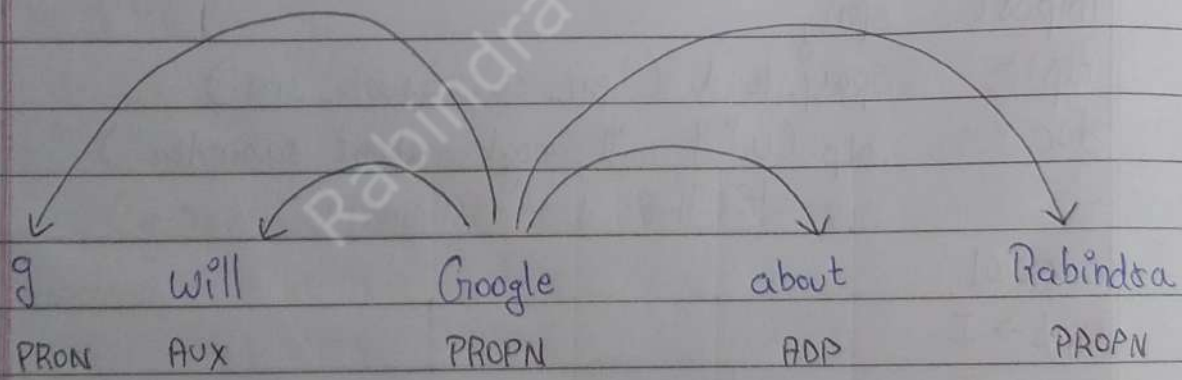
Note: `pos_` gives course great parts of speech
`tag_` gives fine great parts of speech
(This tells us about in more detail)

✯ To understand Spacy we can write explain() function ✯

① spacy.explain('NNP') (or) spac.explain('PRP') (OR)...

Pos Tagging Visualization

```
from spacy import displacy
displacy.render(doc, style='dep', jupyter=True)
```



Working of POS TAGGING ?

Soln:

FAMOUS Algorithm HMM

↓

Hidden Markov Model

Consider a statement

1) Will will google Chess

↳ here Will is name of a person

But for predicting the output for above statement we need to train it on some Data.

eg:	Nitish loves chess
	Can nitish google Campusx
	Will Ankita google Chess
	Ankita loves Will

Using this data we will train our HMM model to predict above statement "Will will google chess" ie we will create pos tagging of above statements and then try to create pos of our statement "Will will google chess"

PAGE No.
DATE / /

Understanding our statement wise POS

(N) Nitish	(V) loves	(N) Chess	
(M) Can	(N) Nitish	(V) google	(N) campusx / chess
(M) Will	(N) Ankita	(V) google	(N) chess
(N) Ankita	(V) Loves	(N) Will	
(N) Will	(V) Love	(N) Google	

N: Noun
V: Verb
M: Model

Rows will contain all unique words while column will contain all unique Pos

	N	M	V
Nitish	2	0	0
loves	0	0	2+1
chess	3	0	0
Google	1	0	2
Will	2	1	0
Ankita	2	0	0
Can	0	1	0

Here, we need to calculate Emission Probability. Divide by total of the column.

Emission Probability

	N	M	V
Nitish	2/10	0	0
loves	0	0	3/5
chess	3/10	0	0
Google	1/10	0	2/5
will	2/10	1/2	0
Ankita	2/10	0	0
Can	0	1/2	0

अगर Given word is Noun (N) probability of it being Nitish is [2/10]

TRANSITION Probability:

(N) → (V) → (N)

ie probability of transition ie. Nitish loves chess

Noun का अगला word Verb होने की संभावना

→ Verb का अगला word Noun होने की संभावना
→ Model का अगला word Verb होने की Probability
→ Noun का अगला word Model होने की Probability

To calculate this we will assign 'Start' and 'End' to each and every sentence.

	(N)	(V)	(N)	
S	Nitish	loves	Chess	E
	(M)	(N)	(V)	(N)
S	Can	Nitish	Google	chess E
	(M)	(N)	(V)	(N)
S	Will	Ankita	Google	chess E
	(N)	(V)	(N)	
S	Ankita	loves	Will	E
	(N)	(V)	(N)	
S	Will	Loves	Google	E

TRANSITION TABLE

END

		N	M	V	E
Start ←	S	3	2	0	0
	N	0	0	5	5
	M	2	0	0	0
	V	5	0	0	0

ie Start Ke baad Noun 3 bar aya hai
Noun Ke baad Verb 5 bar aya hai
⋮

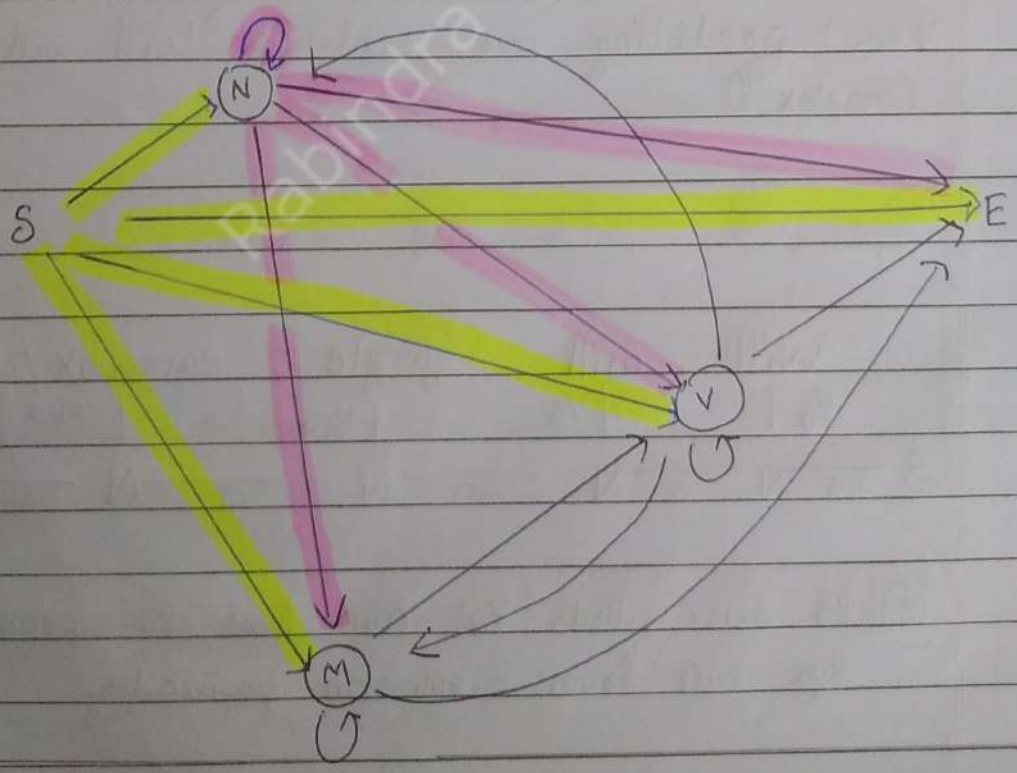
Calculating Probability

Divide it by total of "Row" and not Column.

Transition Probability

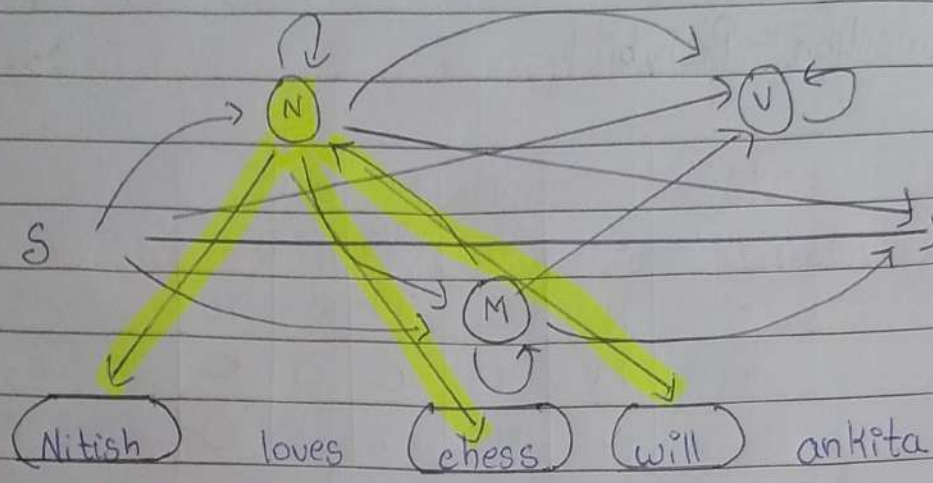
	N	M	V	E
S	3/5	2/5	0	0
N	0	0	5/0	5/0
M	2/2	0	0	0
V	5/5	0	0	0

Using "Emission" Table and "Transition" Table we can create HMM model



Highlighting All vertices from start, and all vertices originating from N (NOUN).

This what is called Transition Diagra.



ie Probability of Noon with respect to "Nitish", "chess" and "will"

Now, predicting our sentence "Will will google Complex."

Will	will	google	campusx/chess	
1/5	1/5	1/10	3/10	
S —3/5→ N	N	N	N —5/10→ E	0

(ie) ye saare words ka noun hone ka probability. Kya hai from emission probability.

Will	Will	google	chess
1/5	1/5	1/10	3/10

S	N	N	N	N	E	= 0
3/5	2	0	0	5/10		

S	N	N	N	M	E	= 0
1/2	1/5	2/5	1/10			

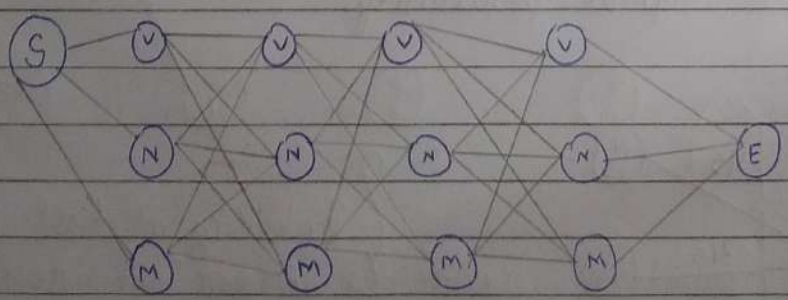
S	M	N	V	N	E	≠ 0
2/5	1	1/2	1	1/2		

In this case, we will use the above (S- M- N- V- N- E) combination for the statement "Will will google chess"

Problem:

For Large Number of words No. of combinations would increase

eg: Will Will google chess



Total combinations is = V^W (V = No. of POS, W = No. of words)

$= 3^4 = 81$

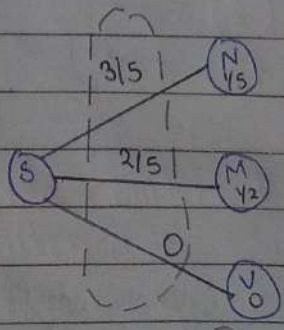
So as soon as number will increase our combinations will grow exponentially.

For optimization we can use Viterbi Algorithm.

Viterbi Algorithm

Will	Will	google	Chess
N 1/5	N 4/5	N 4/10	N 3/10
S — M 1/2	M 1/2	M 0	M 0
V 0	V 0	V 2/5	V 0

Step 1: We will first consider the past with highest probability



Transition Probability

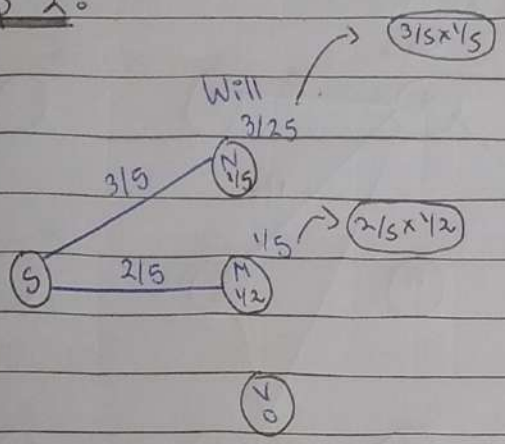
By looking at first stage we can eliminate S → O → V

Mishra Rabindra

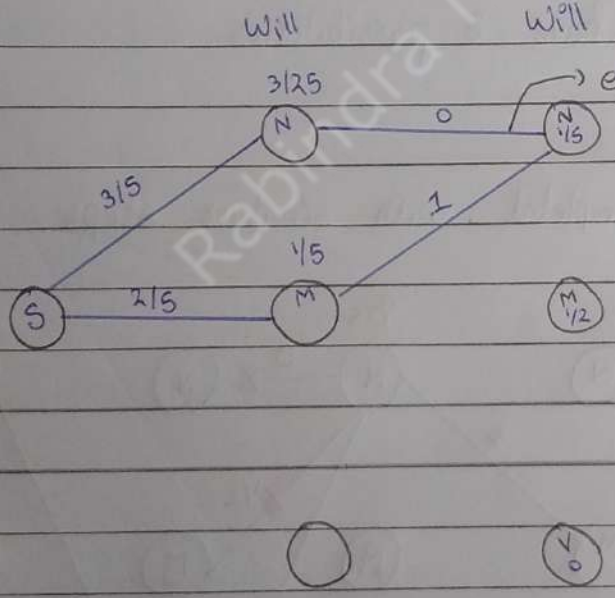
PAGE No.
DATE / /

Step 2:

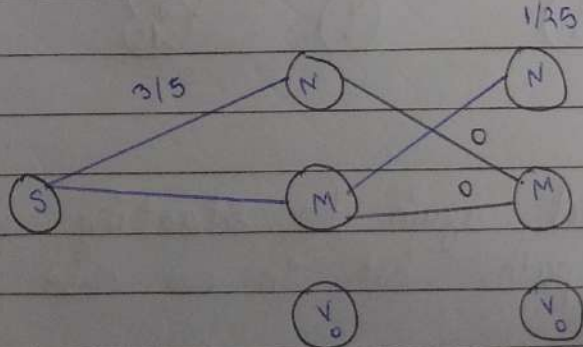
(i)



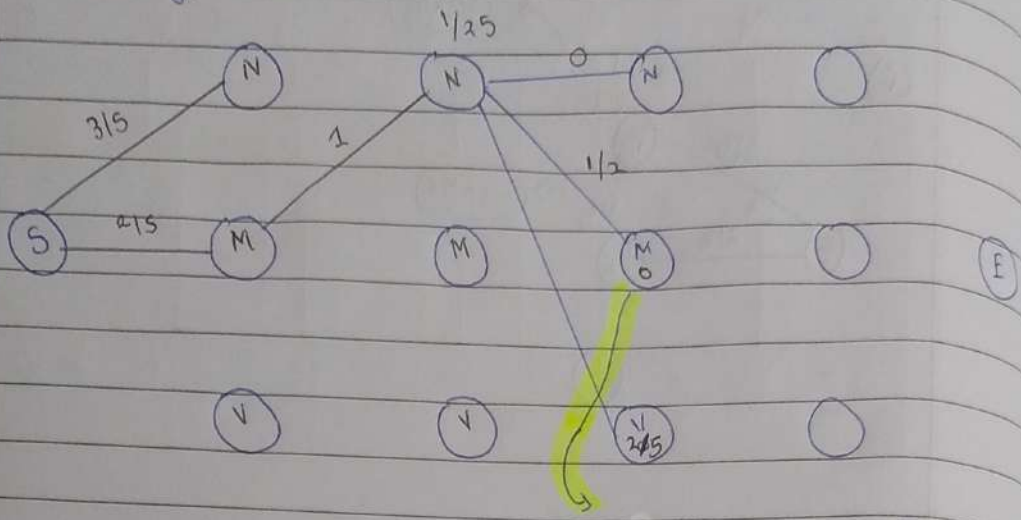
(ii)



(iii)



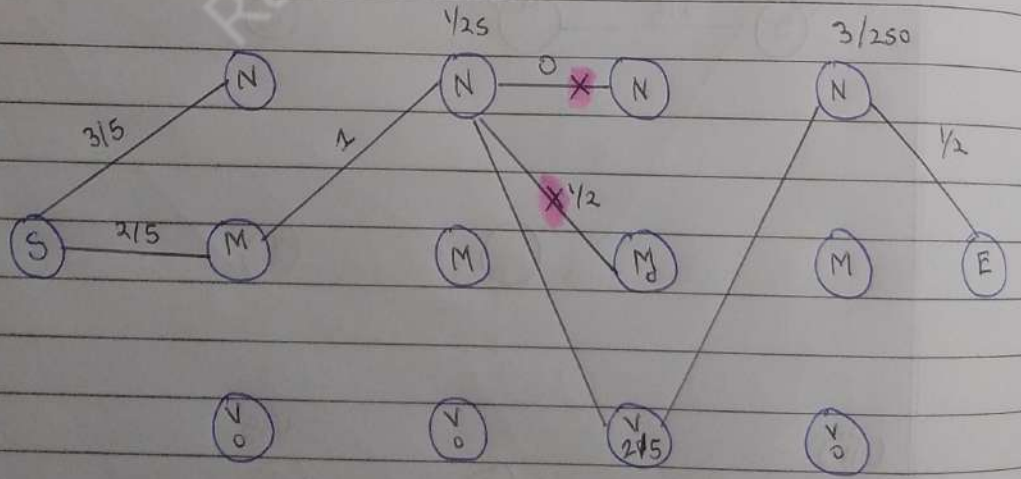
∴ Similarly, for second Layer.



eliminate this one ∴ NFD

Here, we have 3 possibilities.

step 3: Completed with similar steps.



Now, Backtract again to starting position for alternative paths but in our case we don't have any.

So Final Output : M/N/V/N → Pos Tagging.

So In this algorithm we are able to reduce certain possibilities.

(PROJECT): 1) Duplicate Question Pairs

2) Spam Classifier.

Dataset: Quora (A very good project).