Natural Language Processing

16/11/22

1) Basic NLP
   - i) Text pre-processing
   - ii) text — vector
     - → techniques — BOW
       - TF, IDF

Python — library — NLTK
↓
Natural Language Tool Kit.

Coding

```
import nltk
from nltk.tokenize import sent_tokenize,
                          word_tokenize

sent = " biryani is
```

whenever you are using paragraph & spliting by sent_tokenization always the sentence should have space before it starts or after ending with symbols.

stem → is an attribute

Parts of speech also matters

POS tagging → then using this we can use lemmatization correctly.

Sklearn – Library – scikit

used for all ML algorithm

CountVectorizer – is a class → initialize the class, returns object.
↓ same as BOW

whenever you want to train the data, use fit → this learning is called fit

fit → transforming
↓ fit & transforming

fit: learning from the data

transforming: whatever learnt from data transform it to matrix vectors

unique words will automatically be saved in class not to the assigned variable

import nltk
from nltk.tokenize import sent_tokenize, word_tokenize

for a sentence → use word_tokenize
for a paragraph → use sent_tokenize

after ending the sentence with any character like, sentence ending symbols, ex- ., ?, ! the next sentence should start with space.

from nltk.corpus import stopwords
then assign it to a variable

stp = stopwords.words("english")

⇒ stemming & lemmatization :-

from nltk.stem import PorterStemmer, LancasterStemmer, SnowballStemmer, WordNetLemmatizer

⇒ ps = PorterStemmer()
ps.stem("give the word here")

Same for others as well.

Text pre-processing

```
tf = [ ]
for sent in data["text"]:
    t = [ ]
    ts = sent.lower( )  —— (I st step)   *
    ts = re.sub("[^a-z A-Z]", " ", ts) : (II nd)
    for word in word_tokenize(ts): III rd
        if word in stp:  (stopwords)
            pass
        else:
            stems = PorterStemmer( ).stem(word)
            t.append(stems)  (Stemming)
    tf.append(" ".join(t))
```

⇒ Import sklearn
from ~~nltk~~ sklearn.feature_extraction.text
import CountVectorizer

```
cv = CounterVectorizer( )
CounterVectorizer(binary = True)
```

↓

This will only '0's & '1's

'0's ↓ no word

'1's ↓ word yes.

→ cv.fit_transform(data["review"])

this will give sparse-vector matrix which contains more zeros.

17/11/22

to correct spelling.

import TextBlob

st → pip install textblob

from textblob import TextBlob

pd.DataFrame(Nwo-cv.toarray(), columns = sorted(cv.vocabulary_))

18/11/22

→ when you have reviews in different languages use Translator.

from translate import Translator
translator = Translator(from_lang="hindi", to_lang="en")

(A) I.am happy am

I will be happy

I	am	here	will	be
1	2	1	0	0

	1			

translation = translator.translate("अलविदा")

⇒ output → translation
Bye-Bye

18/11/22

whenever your categorical column is in text formate - we have two techniques

- → label encoding → simply mapping → we whenever your column already have an order
- → One hot encoding

ex:- Hair colour

"brown"	1
"black"	2
"red"	3
"white"	4

this is called mapping technique

this is not correct as this column has no order.

this is leading to → machine thinks "white" is more important and follows. by giving this

to get rid of this we One hot encoding

whenever our column is categorical & is having no order or nominal use One hot encoding

How does it work :-

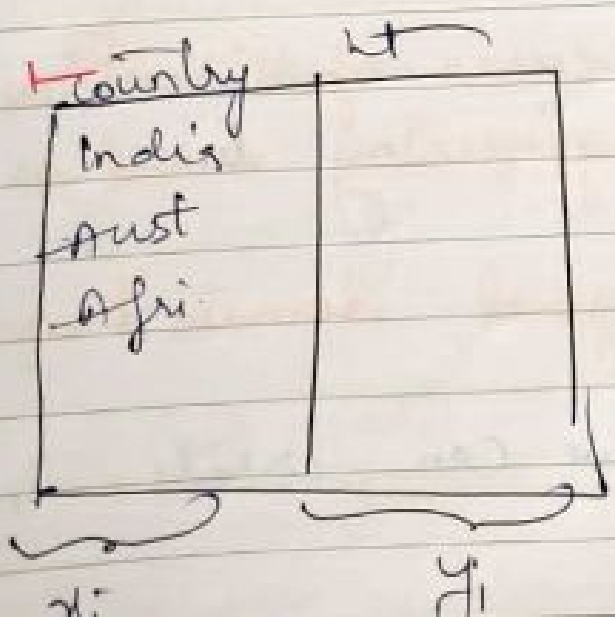
1) count Unique categories — and the length with this we know dimension
2) vector of d-dim → from length.
3)

"black"	1	0	0	0
"brown"	0	1	0	0
"red"	0	0	1	0
"white"	0	0	0	1

when dimensions ↑ses. curse of dimensionality ↑ses

PS → predict ↓t.

Hack



take country and do average of persons from that country & replace it wherever that country is there

ex1 —

ht	Country
10	Ind (10)
16	Ameri (16)
14	Aust (14)
13	UK (13)
11	Ind (10)
16	Ameri (16)
9	Ind (10)

⟹

ht	Country
10	10
16	16
14	14
13	13
11	10
16	16
9	10

will consider all the ht from country and do average & replace it at that particular column.

Ind → $\frac{10+11+9}{3} = \frac{30}{3} = 10$

Ameri → $\frac{16+16}{2} = \frac{32}{2} = 16$

= to get rid of curse of dimentionality do feature engineering based on PS.

= also having good domain knowledge.

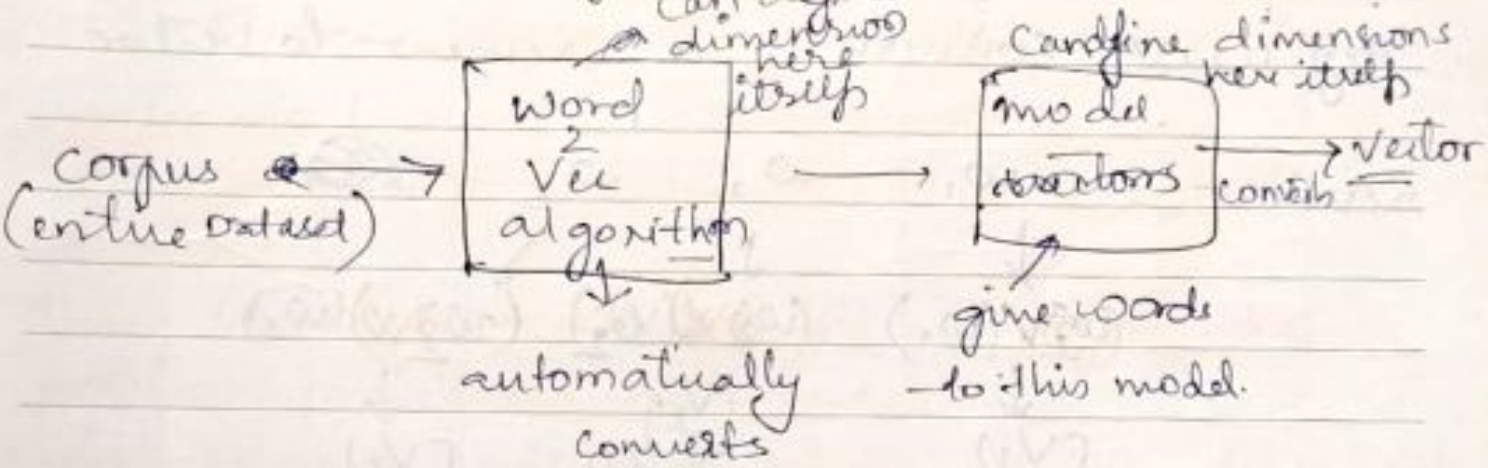
↓

using this we can create new features.

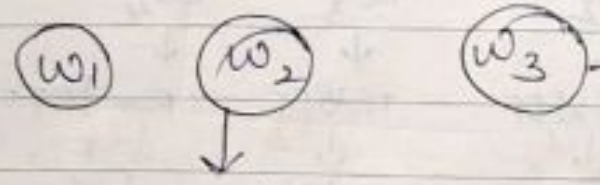
State of art of technique :- Word 2 Vec

Deep learning Technique:

- Word 2 Vec → considers semantic meaning
  - It says give me a word I will create a vector.
  - → also learning relationship betn vectors.



- → It works best for large Dataset
- → the vectors max^m dimension would be 500
- → this will be in dense matrix, where there are less zeros.
- → as dimentionality increase the vector will be strong.
- →



based on neighbouring words, it will convert to vector considering semantic meaning.

If $V_1 - V_2 \parallel V_3 - V_4$

There is a Semantic relation & it is parallel to each other.

Average W2V :-

W2V converts word to vectors.
Avg W2V converts a review to vector

review = w_1 w_2 w_3

$(w_2V)(w_1)$ $(w_2V)(w_2)$ $(w_2V)(w_3)$

$(V_1)_{20}$ $(V_2)_{20}$ $(V_3)_{20}$

then add vectors & take avg.

$$\boxed{\frac{1}{n} V_1 + V_2 + V_3}$$

TF IDF - W2V :-

$r_1 = w_1$ w_2 w_3 w_4

I → TF×IDF TF×IDF TF×IDF TF×IDF

II → w_2V w_2V w_2V w_2V

t_1 t_2 t_3 t_4

V_1 V_2 V_3 V_4

III → $t_1 \times V_1 + t_2 \times V_2 + t_3 \times V_3 + t_4 \times V_4$ / $t_1 + t_2 + t_3 + t_4$

Avg W_2V } convert a sentence to vector
TFIDF W_2V }

W_2V → Convert word to vector

One hot encoder | LabelEncoder
import sklearn,
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

[LabelEncoder]
→ this class object → has to be assigned to a variable.
→ this variable must be fit_transform the particular | mentioned column.
→ this column will be given labels based on alphabetical order.

→ One hot :—
array should always be 2d.
→ here, considering the rank it gives output array object.
based on ranks — 0001 → this means the
1000

row 1 with number 1 means the object is present in that place.

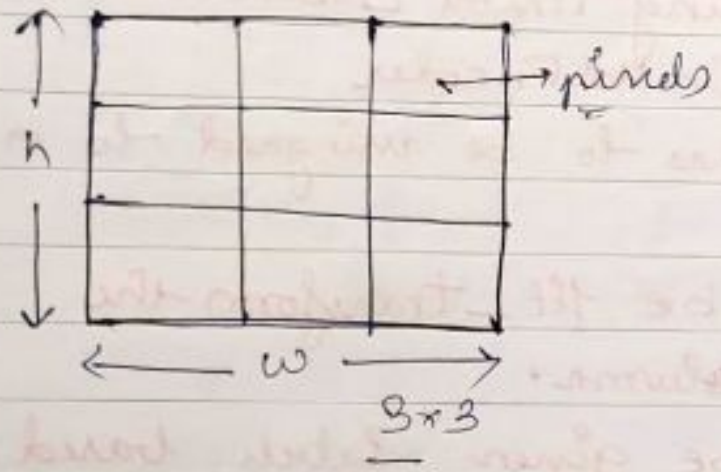
19/11/22

Image :—

2-dimensional representation of a visible spectrum

- representation of visible light.

more the pixels more the clarity of image



if someone say your image has 2 level — it means Black, white i.e, 0, 1

If your image is Gray scale image there values range from 0 – 255

0 → Black

1 → white

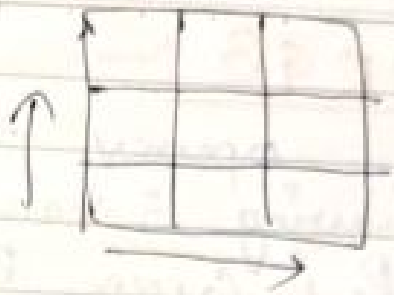
remaining 244 – grey.

256 → in total → 2^8 → 8 bit value.

starts with black ends with white

Color Image :-

we have three values



Red (0-255) Green (0-255) Blue (0-255)

Black & white and Gray scale image → stored in machine with (length, ht) 2-dimensional array

Colour image → 3-dimensional array (R, G, B)

21/11/22

Colour space :- * It is a method by using which we can store the image in a machine

* We can also define as storing the colour.

1) Black and White :- In Image Stores 0, 1 in each pixel.

0 → black, 1 - white. 2^1 → 1 bit

→ stored in black & white space.

2) Gray scale image :- (0 → 255)

between 0 to 255 it is levels of gray scale

→ 2^8 → 8 bit (each pixel stored in) stored in Gray scale space

→ 3) RGB :-

opencv says that each pixel is having 3 values belongs-to Three Colours Red, Green, Blue.

Red → 0 but shades of red 255
↓ high intense Red.

the image is stored in RGB colour space.

4) HSV :- Hue, Saturation, Value.

Hue → stands for colour
Saturation → states Intensity of Colour
Value → brightness of colour

opencv range Hue → 0 → 180 (or 275)
Saturation → 0 – 255
Value → 0 – 255

Colour space used here is HSV.

open CV → Computer Vision

python library which contains lot of tools & methods, we can apply them on images & videos.

Data + Algorithm

Image → Vector → ML model.

How to convert image to vectors :-

Copy image path & read

cv2.imread(path) → convert image to array.

→ to see image :-

imshow("any msg." img array)

this converts into 2-d image.

we can give waitkey → if waitkey = 0 & time it will wait till you close.

cv2.imshow( )
cv2.waitkey(0)
cv2.destroyAllWindows

if waitkey(10) → 10 milli secs.
your image automatically gone

1) Bag of words :- It is a technique by which we can convert text to vectors

1) Sentence → group of words/tokens.
2) Documents → can be a single sentence, paragraph, files
collection of sentences, paragraphs, files

3) Corpus :- Collection of all the documents

4) Tokens :- breaking a huge thing into sub-groups is called Tokens

ex:- Today is ML class

breaking it to sub words.

today
is
ML
class

} tokens

→ tokens → can be words, sentence, ~~paragraph etc.~~

5) tokenization :- process by using which we can get the tokens.

There are two types of tokenization
1) Sentence tokenization
2) word tokenization

→ Vocabulary → set of all the words.

Bag of words (BOW) :-

Let, us say, we have ~~five~~ 3 documents.

d_1 :- w_1 w_2 w_3 w_4 w_5 w_6
d_2 :- w_1 w_2 w_4 w_8 w_5 w_6
d_3 :- w_1 w_9 w_3 w_7 w_5 w_6

① create a dictionary :-
↳ not python dictionary
→ In this we have all the unique words from corpus.

~~the length of~~.

② no. of unique words indicates → no of dimension

ex:- 9 dimensions i.e 9 unique words.
each word is called dimension.

③ V_1 =

w_1	w_4	w_3	w_7	w_4	w_8	w_9	w_5	w_2
1	1	1	0	1	0	0	1	1

No sequential order.

How many no. of times a word is occuring

$V_2 =$

w_1	w_6	w_3	w_7	w_4	w_8	w_9	w_5	w_2
1	1			1			1	1

$T_1 \longrightarrow V_1 \longrightarrow P_1$

$T_2 \longrightarrow V_2 \longrightarrow P_2$

$T_3 \longrightarrow V_3 \longrightarrow P_3$

features :

words	-	-	-
-	-	-	
-		-	

Technique bag of words

ex1- d_1 :- Biryani is very good & less
d_2 :- " " " bad & cheap
d_3 " " " good " affordable.

unique words = biryani is very good, bad, less cheap & affordable

$V_1 =$

1	1	1	1	0	1	0	1	0

$\Rightarrow P_1$

bir is very good bad less chp and affor

$V_2 =$

1	1	1	0	1	0	1	1	0

$\Rightarrow P_2$

$V_3 =$

1	1	1	1	0	0	0	1	1

$\Rightarrow P_3$

$$P_1 - P_3 = \sqrt{(1-1)^2 + (1-1)^2 + (1-1)^2 + (1-1)^2 + (1) + (0) + (1-1)^2 + (1)^2}$$

$$= \sqrt{1+1} = \sqrt{2}$$

$$\boxed{V_1 - V_3 = \sqrt{2}}$$

$$S(r_1, r_3) > S(r_1, r_2)$$

$$\therefore d(V_1 - V_3) < d(V_1 - V_2)$$

$V_1 - V_2$:-

$= (1-1)^2 + (1-1)^2 + (1-1)^2 + (1)^2 + (1)^2 + (1)^2 + (1-1)^2 + (0)$

$\boxed{V_1 - V_2 = \sqrt{4}}$

$S(V_1 - V_3) > S(V_1 - V_2)$ ✓ Hence proved

Binary bag of words :-

① creating a dictionary → set of all unique words.

② the number is 1 or 0, if the word occurred give 1, if does'nt occur give '0'

disadvantage :-

bag of words.

→ does'nt consider semantic meaning.

→ No sequential order.

Sparse :- used for n-dim

In a vector, if more no. of elements are '0' it is sparse vectors.

as your corpus size ↑ dimensionality ↑

before converting text to vector we do pre-processing on text called Text pre-processing.

this is mainly done to check whether the dimensionality can be decreased.

Text pre-processing:

1) Convert all your text to lower case letters. (second step)
2) Remove all the stopwords (**) (first step)

↓

those words which will not give meaning to a ~~text~~ sentence.

ex:- 1) this is python class.
python class.

2) This Biryani is very good
Biryani very good

→ very } remove
→ not } them without changing the meaning

3) Remove all unnecessary symbols :-

using Reg X

4) Stemming & lemmatization :-

process by using which we will get the root of the word.

this will try to remove suffixes of the word and give the root word.

stemming
ex :- tasty, tastier, tastfull.

- give root word ⟹ tast

stemming → give root word may not be actual english word.

lemmatization → give root word which have actual english word.

lemmatization :- tasty, tastier, tastfull

→ give root word → tasty

15/11/22

Types of stemming ~~lemmatization~~ algorithms :-

1) Porter stemmer
2) Lancaster stemmer
3) Snowball stemmer

root word
↓
called as stem

= Porter, Lancaster are english stemmer i.e., work only on english
= Snowball stemmer → for other languages i.e., non-english.

Porter stemmer :- (oldest)

has set of five rules. $c[cv]^m v$

c ⇒ consonant
v = vowels.

It will take a word & apply rules.
→ after stemming - the last letter it is not checking for the word.

Lancaster stemmer :- Iterative algorithm.
(new stemmer)

we have 120 rules.
becoz of high stemming the root word will not be the correct english word.

Snowball Stemmer :- use on non-english words.

Lemmatization :-

= algorithm → work not lemmatization

after stemming the last letter it checking with the Database & confirms if it is a root word present in the corpus (i.e., contains all english words) ~~and~~ then gives output, if it is not present in corpus, ~~it~~ ~~then~~ then it redirects to last letter & stops there.

text → Pre-processing → pre-processed text → BOW → TF & IDF → vector

1) lower/upper
2) Remove stopwords
3) Remove symbols
4) stemming & lemmatization

2) TF - IDF :-

Term Frequency — Inverse Document Frequency

steps :-

1) Create a Dictionary (this step → to know the dimensions)
↓
we have set of unique words.

2) Term Frequency (TF) :-

ex :-

r_1 — w_1 w_2 w_4 w_3 w_5

r_2 — w_5 w_6 w_2 w_1 w_2

⋮

r_n

$$\boxed{TF(w_i, r_i) = \frac{\text{No. of times } w_i \text{ occurs in } r_i}{\text{Total No. of Words in } r_i}}$$

Term Frequency of w_i with respect to r_i is equal to No. of times w_i occurs in r_i divided by total No. of words in r_i

$$\rightarrow TF(w_1, r_1) = \frac{1}{5}$$

$$TF(w_2, r_2) = \frac{2}{5}$$

Always Term Frequency value will be between 0 & 1.

$$\boxed{0 < TF < 1}$$

3) Inverse Document Frequency :-

ex:-

Corpus = D

$r_1 = w_1 \quad w_2 \quad w_5 \quad w_6$

$r_2 = w_3 \quad w_1 \quad w_2 \quad w_5$

$r_7 = w_3$

$r_{10} = w_3$

$r_n =$

$$\boxed{IDF(w_i, D) = \log\left[\frac{N}{n}\right]}$$

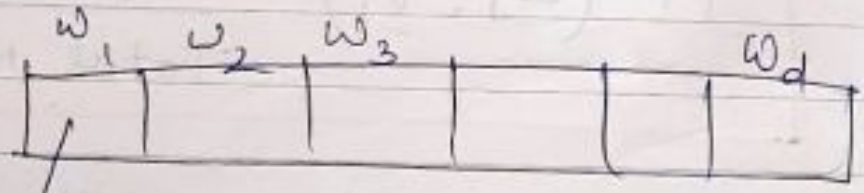
∴ N = no. of documents

n = Total no. of document where w_i occurs.

IDF of w_i with respect to Documents (entire Corpus) will be equal to log of $\frac{N}{n}$

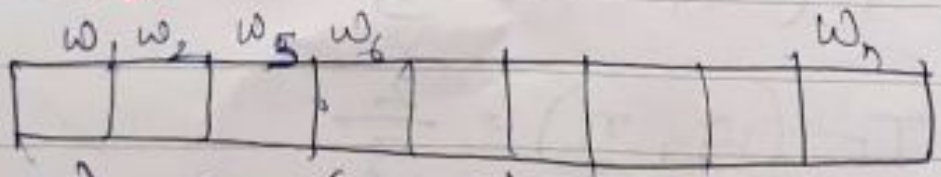
$$IDF(w_3, D) = \frac{100}{3}$$

4) $r_i \rightarrow V \rightarrow$



$$TF(w_1, r_1) \times IDF(w_1, D)$$

ex:- d = 12



$$TF(w_1, r_1) \times IDF(w_1, D) = \frac{100}{2}$$

$$= \frac{1}{4} \times \frac{100}{2} = \boxed{w_1 \text{ value}}$$ → keep it in w_1 vector.

Ex:-

$r_1 = w_1 \quad w_2 \quad w_3 \quad w_4$

$r_2 = w_1 \quad w_3 \quad w_4 \quad w_7$

$r_3 = w_2 \quad w_3 \quad w_6 \quad w_5$

$r_4 = w_1 \quad w_2$

from the above we got $d = 7$

Vectors for r_1:-

w_1	w_3	w_4	w_5	w_6	w_7	w_2

$w_1 \Rightarrow TF(w_1, r_1) \times IDF(w_1, D)$

$\frac{1}{4} \times \frac{4}{3}$

$w_2 \Rightarrow TF(w_2, r_1) \times IDF(w_2, D)$

$= \frac{1}{4} \times \frac{4}{3}$

$w_3 \Rightarrow TF(w_3, r_1) \times IDF(w_3, D)$

$\frac{1}{4} \times \frac{4}{3}$

$=$

Vectors of $r_4 \rightarrow$

w_1	w_3	w_4	w_5	w_6	w_7	w_2
2/3	0	0	0	0	0	2/3

$w_1 \rightarrow TF(w_1, r_1) \times IDF(w_1, D) = \frac{1}{2} \times \frac{4}{3}$

$w_2 = \frac{1}{2} \times \frac{4}{3}$

$\cancel{w_3 \Rightarrow TF(w_3, r_1) \times IDF(w_3, D)} =$

This value will be higher only when

$TF(w_i, r_i) \times IDF(w_i, R)$

more — less

The word which is given more importance is words with more occurance in a single review will have more value & word which is occurring rarely in entire corpus.

$\log \frac{N}{n}$ → less so N will be more.

$\frac{N}{n} = 1$

$\log(1) = 0 \Longrightarrow n \leq N$

when $n = N \Longrightarrow \log(1) = \underline{0}$

then $\log\left(\frac{N}{n}\right) \geq 0$

if IDF = 0, then, that particular word is there in all the documents so the word is given less imp.

use log to lessen the gap between TF & IDF. (as T.F will always be bet

Disadvantages of TF & IDF :-

- ✓ No sequential order
- ✓ No semantic meaning consideration.

→ To give some sequential concept to the ~~Bag of wd~~ TF & IDF so that the meaning is not changing.

for this we use n- grams.

1-gram → 1 word = 1 dimension
2-grams → 2 consequetive word = 1 dim
3 → grams → 3 consequetive word = 1 dim

$n \uparrow$ $d \uparrow$ → curse of dimentionality

$\boxed{\text{she}}\ \boxed{\text{is}}\ \boxed{\text{a}}\ \boxed{\text{good}}\ \boxed{\text{girl}}$ — 5

she is ~~is~~ a good good girl →