lect 1/2

Natural language programming (NLP)

#NLP Timeline

① → Data finding

② → Text prep
- ↳ clean up
- ↳ spelling mistakes, etc
- → Html tags, Emoji
- spelling check, fast typing
- → Tokenization
  - sentence
  - word
- lowering
- → stopword (for/are/in/the)
- → stemming (dance, dancing)

③ → Feature Engg.

④ → modeling
- ↳ model Build
- ↳ Evaluation

⑤ → Deployment

⑥ → update model.

Adv Text pre-processing

→ Part of speech procesing (noun, verb, subject, etc)

→ Coreference resolution

ex: A: (my) name is Tanishq and (I'm) 19

(same person Tanishq)

lect3] # Text preprocessing (Basic)

① Lower casing

A → Start of the sentence

a → middle sentence

→ Both are different tokenized, and interpretted different.

```
df['column'] = df['column'].str.lower()
```

② HTML Tag remove

→ use regular expr

```
import re
pattern = re.compile('<.*?>')
filtered_pattern.sub(r'', text)
```

③ Remove urls

→ using regular exp

④ punctuation

→ !'"#$%&\()*+,-./:;<=>?@[\]^_`{|}~

```
import string
```

string.punctuation.

→ too slow

```
for char in string.punctuations:
    text = text.replace(char, "")
```

```
text = text.translate(str.maketrans("", "", string.punctuations)) → Fast
```

⑤ Chat word treatment

① lmao / fyi / gn / ASAP (example)

↓

Have to do it manually (get list of these words and use map).

⑥ Spelling check

```
from textblob import TextBlob
text = TextBlob(text).correct().string
```

use this method

⑦ Stop words

① a / the / of / are / my → sentiment

② pos tagging don't do this removal.

③ from nltk.corpus import stopwords

stopwords.words('english')

⑧ Handelling Emoji's

① use regx

```
pattern = re.compile("[
    ____   # emotions
    ____   # symbols & pictography
    ____   # transport, map symbol
    ____   # flags (ios
]", flags= re.UNICODE)
```

② use (emoji) package

```
import emoji
text= emoji.demojize('python is 🔥')
```

fire symbol

→ python is :fire:

⑨ Tokenization

$20	→	$, 20	✓
Delhi!	→	Delhi, !	✓
New-York	→	New, York	⊗
U.S	→	U, S	⊗
Ph.D	→	Ph, D	⊗
$ 10.5	→	$, 10, 5	⊗

```
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp(text)
for word in doc:
    print(word)
```

⑩ Stemming

(fast) → process of inflected words to root word
(But here root word may be not a valid word)
ex: story → stori

root word → walk, walking, walks } this is called inflection

inflected words

```
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()
text = "".join([ps.stem(word) for word in text.split()])
```

(11) Lemmatization (done)

Inflected word → to → root word (valid root word)

from nltk.stem import WordNetLemmatizer

Lect 9] # Text Representation

Common terms

(1) corpus (C) → count of total words in a dataset.
(2) vocab (V) → count of unique words → ——— .
(3) Document (D) → Individual row (sentence) in dataset.
(4) word (w)
(5) OOV (out of vocab) → word which

① One hot Encoding

D1) people watch yt
D2) yt watch yt
D3) people write comment
D4) yt write comment

V = 5

	people	watch	yt	write	comment
D1)	1	1	1	0	0
D2)	0	0	1	0	0
⋮					

② Bag of words

D1) people watch yt → [1 1 1 0]
D2) → [0 0 2 0]

↑ frequency of words

D1)
D2)
D3)
D4)

from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer()

cv.fit_transform(text)

OOV is ignored

⑧ N-gram

→ similary like Bag of words but contains N-words

Bag of words

D1) people watch yt yt
D2) people comment watch

vocab	people	watch	yt	comment
D1)	1	1	2	0
D2)	1	1	0	1

2 gram (bi-gram)

save doc

vocab	people watch	watch yt	yt yt	people comment	comment watch
D1)	1	1	1	0	0
D2)	0	0	0	1	1

vocab of two combining words

⑨ TF-IDF (gives weitage to words)

TF → term frequency
IDF → Inverse doc frequency

$$TF(t,d) = \frac{\text{Freq of term } (t) \text{ in doc } (d)}{\text{Total no. of terms in doc } (d)}$$

$$IDF(t) = \ln\left(\frac{\text{Total no. of docs in corpus}}{\text{Total no. of docs which has term } (t) \text{ in them}}\right)$$

```
from sklearn.feature_extraction.text
import TfidfVectorizer.
```

D1) People watch yt
D2) yt watch yt
D3) people write comment
D4) yt write comment

note

finding

TF (of term t) → gives prob of term t in that doc
- high TF → term freq ↑ ⇒ more imp
- low TF → term freq ↓ ⇒ less imp

IDF (of term t) → tell the value of that term in corpus.
- high IDF → term (t) freq ↓ ⇒ more valuable.
- low IDF → term (t) freq ↑ ⇒ less valuable.

TF(people, P_1)* IDF(people)

	People	watch	yt	write	comment
P1)	$\frac{1}{3} \times 0.3$	$\frac{1}{3} \times 0.3$	$\frac{1}{3} \times 0.125$	0	0
P2)	0	$\frac{1}{3} \times 0.3$	$\frac{2}{3} * 0.125$	0	0

lect 5] # word2vec

→ this can distinguish b/w Happy and Joy (semantic meaning preserve)

→ low dimn vector (unlike Bag of word / etc)

→ Dense vec (unlike 2 which has sparse vector).

How to use it

use pre-trained model

→ google new corpus trained on 3billion words

300 dimn vector

train model in your dataset then use it

```
import gensim
from gensim.models import Word2Vec, KeyedVectors
# download google model (1.5GB)
model = KeyedVectors.load_word2vec_format('GoogleNews - - - ', binary=True)
model['cricket'] = [- - - - -]
                   (i.e 300)
```

Here

man/women will have similar vector.

we use deep learning to create these features

cy

these are features

	money	power	Alive	Gender
king	1	1	1	1
man	0	0	1	1
queen	1	0	1	0
women	0	0	1	0
Boy	0	0	1	1
girl	0	0	1	0
Bottle	0	0	0	0

types of word2vec (training model)

custom

for

① **CBOW** (for small data)

context (feature)

⇒ watch (campusx) for data science

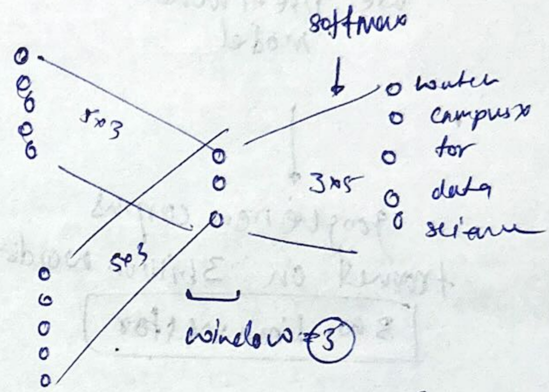
target

window=3

feature		output
watch	for	campusx
campux	data	for
for	science	data

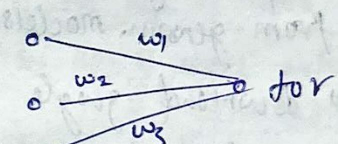
ohe

watch	100000
campusx	010000
for	001000
data	000100
science	000 →



vector representation of "for" is it's weights

⇒ [w1, w2, w3]



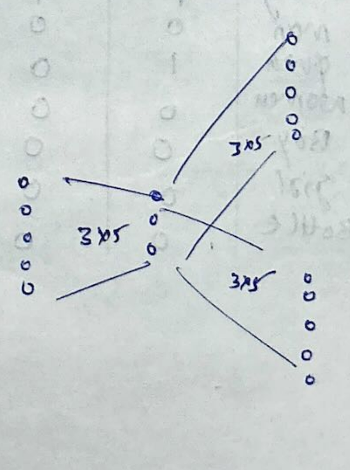
no. of windows == no. of N-vector

② **skip-gram** (for large data)

→ this is reverse of CBOW

→

Input	output
campusx	watch, for
for	campusx, data
data	for, science



lect 6]

Text Classification

email content → spam/not spam

sentence → (+ve)/(-ve) sentiment

type

- **Binary**: spam/not-spam
- **multiclass**: action/murder/fighting horror.
- **label**: sports/cricket/dhoni

(Binary, multiclass) — any option is answer

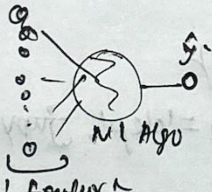
(label) — can have multiple answer

modelling

- **ML**
  - → Naive Bayes
  - → Random Forest
  - → SVM
- **DL**
  - → RNN
  - → LSTM/GRU

methods:

- Bow/N-gram → each row is converted to vocab(V) features.



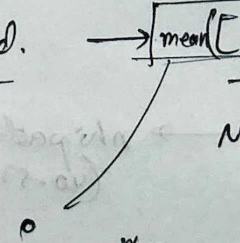
- word2vec → train model/use pre-trained model.
  - (← return N-dimn vector for each word.)

D1) Hello ross, i am good.

Hello → N-dimn, ross → N-dimn, i →, am →, good →

→ mean([- - - -]) → Add, axis=0

→ N-dimn



lect 7] # part of speech tagging (POS) → uses Hidden markov models Algorithm.

optimization: viterbi Algo

Application of POS

1. Named Entity recognition
2. question / Ans system
3. Hi (my) name is Tanya, (I'm) good.
   refer same person and means same.
4. I (ran) to shop, but shop (ran) out of milk
   same word different meaning.

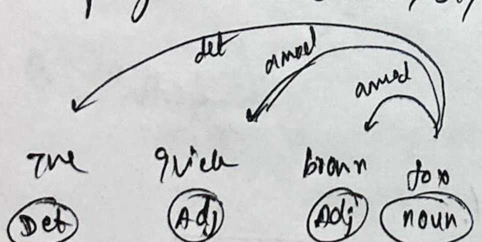
code

```
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp(text)  → I will google about facebook.
```

- doc.text (will return text)
- doc[0] = 'I'
- doc[-1] = 'facebook'
- doc[0].pos_ } → Noun, verb, ~~etc~~
- doc[0].tag_ } code
- doc[0].pos_ → coarse grain pos
- doc[0].tag_ → fine grain pos

If you don't understand what this code is
spacy.explain(code) → 'VB' / 'PRP'
- VB → verb, base form
- PRP → pronoun person

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



How it works? → nhi padha
(40-50 min ka video hai)