Page No.
Date

Unit-6 (Honor)

Text processing - It deal with Text data.

Before prediction we need to perform text preprocessing.

Some steps are.

* Removing punctuations like . , ! $ ( ) * % @
* Removing URL
* Removing Stop word
* lower Casing
* Tokenization
* Stemming
* Lemmatization.

Removing punctuation

```
Import String
def remove_punctuation(text):
    final = "".join([i for i in text if i not
                     in String.punctuation])
    return final.
```

Lowering case

```
d['a'] = data['b'].apply(lamba x: x.lower())
```

Stop word removal

→ I, am, not, is, a

Page No.
Date

```
Import nltk
Stopword = nltk.corpus.stopwords.word('english')
```

Stemming

Converting to root base

But meaning may loss.

```
from nltk.stem.porter import PorterStemmer
port_st = PorterStemmer().
new = port_st.stem(old).
```

lemmitation

to root base but have meaning

```
from nltk.stem import WordNetLemmatizer
w_L = WordNetLemmatizer()
new = W_L.lemmatize(old)
```

Tokenization

- Splitting a string
- text to list of tokens
- text to Sentence
  - → text to word
- text to RE

Types.

① Word Tokenization

```
from nltk.tokenize import word_tokenize

text = "hello everyone. welcome"
word_tokenize(text)
```

② Character Tokenization.

③ Sentence Tokenization.

④ Subword Tokenization eg lower → low-er
Smartest → smart-test

Feature extraction.

→ The process of Converting textual into number is called Vectorization.

→ This feature extraction

→ Easy for Computation

disadvantage
① Sparse matrix
② Can't relate with similar matrix
③ Not relationship wise

two main Method

① bag-of-word with TF-IDF

bag of word.

- It is representation model of document data.

- Count the no. of word in document.

```
from sklearn.feature_extraction.text import
        CountVectorizer
```

```
Sent = ["Twis is a nemagoodisboy"]

vt = CountVectorizer()
vt.fit(Sent)
v = vt.transform(Sent)
Ans = v.toarray()
Ans
```

Tf-IDF

Tf - Measure the frequency

$$Tf = \frac{\text{Count of t in d}}{\text{No. of word in d}}$$

IDf - measure the importance of a word

$$IDf = \log\left(\frac{\text{Total no. of document in corpus}}{\text{no: of document containing term t}}\right)$$

TFIDF = Tf X IDf

Here eg. "the" can occur more time But it has less significance so, TFIDF will detect it. it has less TFIDF value.

② Word Embedding

→ representing numeric input that represent a word in lower-dimensional space.

It allow word with similar meaning to have similar representation.

→ It have feature such as Age, Sports, food, etc. and many

Man →

(a) Word2vec

→ each word is represented as a vector of 32 or more dimension.

→ Semantic info stored

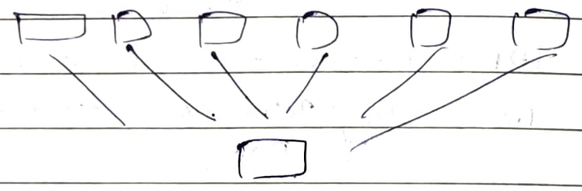
King - Man + Woman = Queen

Two algorithm

(a) continuous Bag of word [CBOW]

* predict the target word from the context

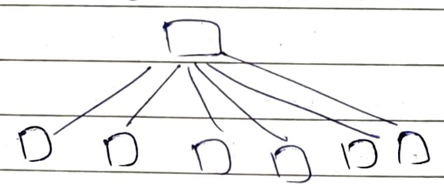
neural network is used



(b) Skip Gram

- predict the context word from target

neural network is used.



(b) Glove

derive the relationship b/w the word from Global Statistics.

eg 1) I play cricket
I love Cricket
I love football

Page No.
Date

	I	play	cricket	football	love
I	0	1	0	0	2
play	1	0	1	0	0
cricket	0	1	0	0	1
football	0	0	0	0	1
love	2	0	1	1	0

Topic Modeling

→ discovering abstract "topics" in documents.

→ unsupervised classification pattern.

Latent Dirichlet Allocation.

Aim - find topic.

first step document word matrix

	w_1	w_2	w_3	w_4	w_5	w_6	w_7
D_1	0	1	1	1	0	0	1
D_2		0	0	0	1	1	1
D_3		1	1	0	0	1	0
D_4		0	0	1	1	0	0

Document topic matrix

	K_1	K_2	K_3	...
D_1				
D_2				
D_3				

word topic matrix

	w_1	w_2	w_3	...
K_1				
K_2				
...				

Step - 3 Decrement Count by 1 from document topic matrix

and same with word topic matrix

Step 4 Calculate

$$P(t_k|d_i) = \frac{n_{ik} + \alpha}{N_{i-1} + k\alpha} =$$

n_{ik} = no. of word in ith doc and kth topic
α = hyper parameter (0.1)
N_i = total no. of word in ith document
k = total no. of topic

	T1	T2	T3
DOC1	2-1	1-1	3-1
DOC2	2-1	3-1	1-1
DOC3	1-1		3-1

$$P(t_k|d_i) = \frac{1+0.1}{6-1+3*0.1} = 0.188$$

Step 5 Calculate $P(w_j|T_k)$

	T_1	T_2	
good	12-1	2-1	20-1
Travel	13-1	10-1	11-1
home	11-1	14-1	25-1
tree	16-1	24-1	

M_{jk} = no. of time word k come in topic K
β = hyper parameter
V = to. no of word.

$$P(w_j|t_k) = \frac{M_{jk} + \beta}{\sum_{j \in V} M_{jk} + V\beta} = \frac{11+0.1}{47+4\times0.1} = 0.025$$

Step 6 Calculate $P(w_j|t_k, d_i)$

$$P(w_j|t_k, d_i) = P(t_k|d_i) \times \beta P(w_j|t_k)$$

Step 7 resampling

for a given word w_j and document i find the topic $P(w_j|t_k, d_i)$ is maximum

Step 8 Repeat all step till n iteration

Text Similarity.

* Used to discover most similar text.

1. Jaccard Similarity.
   - ratio of common word to total unique words
   - 1 represent high similarity
   - 0 represent low similarity

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

2. Cosine Similarity.
   - measure cosine of angle b/w two vectors.
   - Vector can be bag of word, TF-IDF, or DOC2Vec

$$\text{similarity}(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|}$$