NLP

DELTA Page No.
Date / /

Regular Expression

A RegEx, or regular Expression, is a sequence of characters that forms a search pattern.

RegEx can be used to check if a string contains the specified search pattern.

→ Python has a build-in package called re, which can be used to work with Regular Expressions.

Demo :-

```
Import re

txt = "The rain in spain"
x = re.search ("^The.*spain$", txt)
```

There is many future function in "re".

1) find all — (Return a list containing all matches)
2) search — Return a Match object if there is a match anywhere in the string.
3) split — Returns a list where the string at each match
4) sub — Replaces one or many matches a string.

Word to vector

DELTA Page No.
Date / /

Processing natural language text and extract useful information from the given word, a sentence using machine learning and deep learning techniques requires the string/ text needs to be converted into a set of real number called vector.

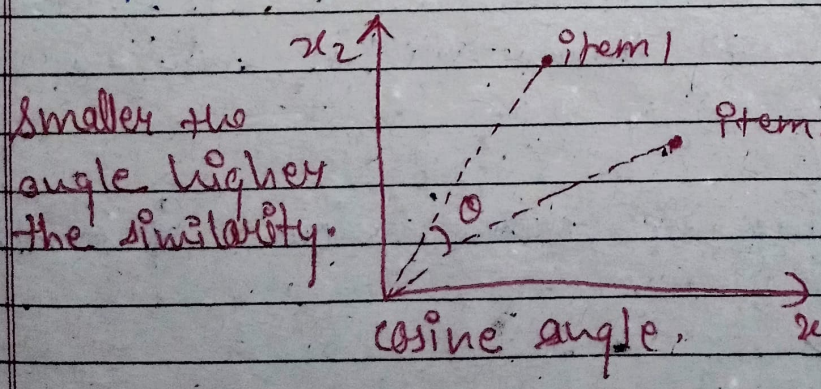
→ Now word embedding or word vectorization is a methodology in NLP to map ~~into~~ words or phrases from vocabulary to a corresponding vector of real numbers which used to find word predictions, word similarities/ semantics.

Cosine similarity approach is used to find the similarity between the documents.

→ Cosine Similarity:

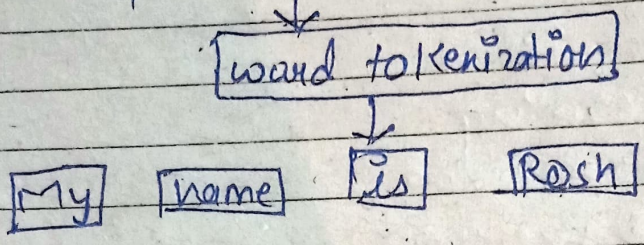
It measures the cosine of the angle b/w two vectors (item1, item2) projected in an N-dimensional vector space.

It predicts the document similarity.



1) tokenization:- segmenting the text into a list of tokens.

Ex:- My name is Rosh

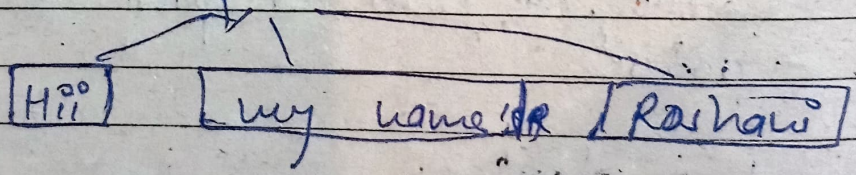


2) Lowercase:- convert all text into lower case

3) stop word removal:- it is used to apart the sentences to extract the meaningful or explaination.
sentence stopers like " , or . (full stop)"

it is kind off problem in it.. ~~(the)~~ for example

Hii, my name's Roshani.



for that we use:-

4) Stemming or lemmatization:-

Stemming used to stripping the suffixes from words to get their stem,

→ I [love eating] pizza.
→ I love eat pizza.

DELTA Page No. Date / /

→ eating and eat is similar. but not eat and ate.
lemmatization involve reducing words to their base form based on their part of speech.

example :- eat and ate (both are kinda similar)

3) Removing digit/punctuation:-
Removing digits and punctuations from the text.

4) Feature Engineering :-

Feature Engineering represent the text in the numeric vector in such a way that the ML algorithm can understand the text attribute.
some of the methods are.
1) TF-IDF vectorizer
2) one hot Encoding
3) word * Embedding.

5) Model Building.
build the model for identify or predict the result.

6) Evaluate
Test the model, it's corrections, faults.
If it is able for correction. Then we must move to the text processing step.

Spacy & NLTK

Spacy is object oriented. we need to create an object of that string then work on it.

It is user friendly. it is a kind off in-built feature which provide best algorithm and give best result.

It provide most efficient NLP algorithm for a given task. Hence if you care about the end result, go with spacy.

→ code :-

```
import spacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("I am Rosh. This is my code.")

for sentence in doc.sents:
    print(sentence)
```

output ⇒ I am Rosh.
This is my code.

```
for sentence in doc.sents:
    for word in sentence:
        print(word)
```

output ⇒ I
am
Rosh
.
This
is
my
code

DELTA Page No.
Date / /

NLTK is a string processing library. we directly pass the string and perform text preprocessing.

NLTK is also user friendly but probably less user friendly compared to spacy.

provides access to many algorithms. if you care about specific algo and customizations go with NLTK

~~NLTK~~ NLTK use - by mostly researchers.

Code :-

```
import nltk
nltk.download("punkt")
from nltk.tokenize import sent_tokenize
sent_tokenize("I am Rosh. This is my python")
```

output ⇒ I am Rosh.
This is my python.

NLP Language processing pipeline in spacy:

1) NLP pipeline is a series of steps that transform a raw text data into a desired output.
like :- label, summary, or a response

DELTA Page No
Date / /

1) This pipeline perform a specific function, such as tokenization, lemmatization, sentiment analysis, named entity recognition.

→ Design a Pipeline.

To design a pipeline, define goal (output)
your data
your tasks
your model.

we can either import blank pipeline (with no functions) or we can import pretrained pipeline (with all function).

→ blank pipeline:

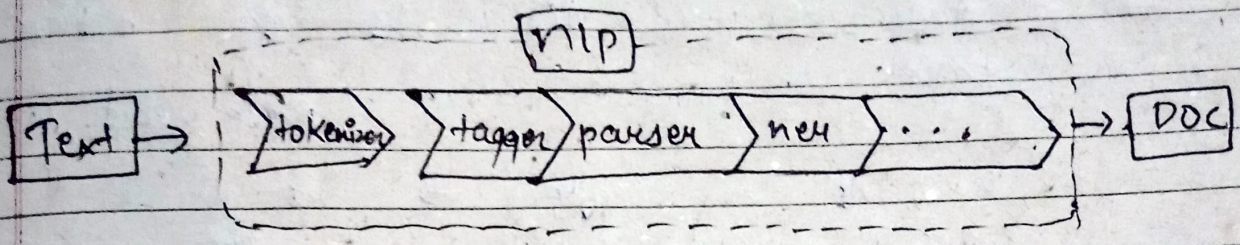
```
nlp = spacy.blank("en")
nlp.pipe_names
```

output ⇒ []

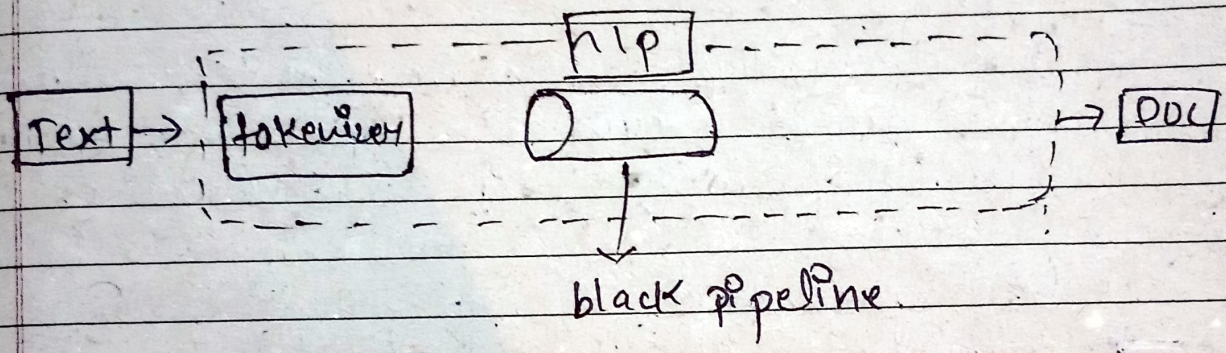
→ pretrained pipeline.

```
nlp = spacy.load("en_core_web_sm")
nlp.pipe_names
```

output ⇒ ['tok2vec', 'tagger', 'parser', 'attribute_ruler', 'lemmatizer', 'ner']



dig:- nlp pipeline with functions.



⇒ add component in blank pipeline.

```
nlp = spacy.blank("en")

nlp.add_pipe("ner", source = source_nlp)
nlp.pipe_names.
```

output ⇒ ['ner']

we can add functions individually in pipeline.

stemming and lemmatization function in pipeline

stemming and lemmatization in nlp is play a crucial role.

→ It reduce words to their base or root forms.
(Base word).

example ⇒ ate → eat
talking → talk
adjustable → adjust

→ This technique normalize the text, allowing for more accurate analysis, information retrieval, and lang. understanding.

Let's understand in deep :-

⇒ stemming (stem)

1) It involve reducing words to their base or root form, know as stem.

2) It remove prefixes, suffixes, and inflections from words.

3) help in reducing vocabulary size, normalizing text, and improving text analysis task.

⇒ Lemmatization (lemma)

1) It is a technique that involves reducing words to their base or dictionary form, know as lemma.

2) Lemmatization takes into account the word's part of speech (pos) and context to deter-

DELTA Page No.
Date / /

wined its canonical form.

ex:- lemma of [ORG]

tesla | ORG | like : company, organization, industrial group.

⇒ stemming and lemmating in nltk

import nltk

from nltk.stem import PorterStemmer
stemmer = PorterStemmer()

stemmer.stem("helping")

output ⇒ helpi

⇒ In spacy

import spacy

nlp = spacy.load("en_core_web_sm")

doc = nlp("helping helps helped")

for token in doc:
print(token, "||", token.lemma_)

output ⇒ helping || help
helps || help
helped || helped.

DELTA Page No.
Date / /

modification in lemmatization dictionary :-

→ attribute_ruler :-

It is used to add an addition word as according to your preference.

→ code

```
att = nlp.get_pipe("attribute_ruler")
att.add([["text": "helping"], ["text": "helped"]],
        {"LEMMA": "help"}])
doc = nlp("helping helps helped")
for token in Doc:
    print(token, "||", token.lemma_)
```

output :-

helping || help
helps || help
helped || help

Part of Speech (POS).

It is a process of converting a sentence to forms - list of words, list of tuples (where each tuple is having a form (word, tag))

It signifies whether the word is a noun, adjective, verb, and so on.

POS	Tag
Noun	n
Verb	v
Adjective	a
adverb	r

⇒ Some of the function in POS

1) token.pos -
provide the type of token it is (noun, verb, (pronoun).

2) spacy.explain (explained-function-provided)
explain the function.

3) token.tag -
it provide the tag given to token (VBP, NN, CC)

4) doc.count_by (spacy.attrs.POS):
count the no. of different token in sentence.

NER (named Entity Recognition)

It is one of the most popular data preprocessing.
It identify the key information in the text and classified it in a predefined categories.

Some of the categories are:-

1) Person
2) organization
3) place/ location
4) date / time
5) expression
6) money, percent, weight, etc)
7) E-mail address

Code :-

```
nlp = spacy.load("en_core_web_sm")

doc = nlp("Twitter Inc is worth of $48 million")

for ent in doc.ents:
    print(ent.text, ent.label_, spacy.explain(ent.label_))
```

→ output

Twitter Inc ORG
$48 million MONEY

DELTA Page No.
Date / /

⇒ Feature Engineering section

Text representation :-

Text representation is a crucial aspect of NLP that involves converting raw text data into machine-readable form using one-hot encoding, Bag of words, TF-IDF, word Embedding.

→ Limitation of one-hot-encoding / Label coding.

1) No fixed length representation.
2) out of vocabulary, if new word is added
3) consume too much memory & compute resources
4) Similar words do not have similar representation

⇒ due to this limitation we use Bag of words, TF-IDF, word Embedding techniques.

1) Bag of words (BoW)

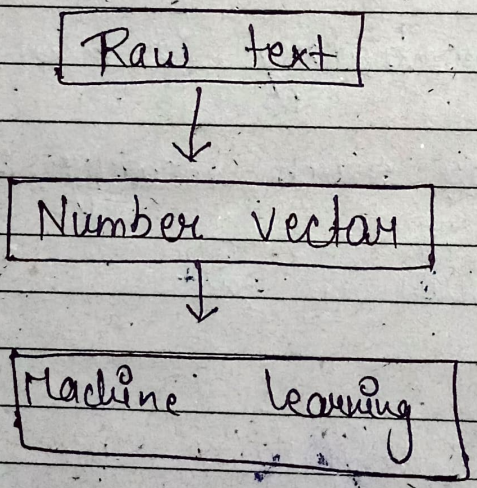
Each word in a text considered as a feature, and the number of times that word appear in text represent the importance of the word.
Disregarding grammer and word order but

DELTA Page No.
Date / /

Keeping track of the frequency of each word.

ex:- The cat in the hat
The dog in the house

	dog	cat	bird	in	house	hat	the
The cat in the hat	0	1	0	1	0	1	1
The dog in the house	1	0	0	1	1	0	1



1) CountVectorizer() , (count the no. of words appearing

2) Pipeline

→ Code

```
from sklearn.pipeline import pipeline
pipe_model = pipeline([
```

DELTA Page No.
Date / /

(1) ('vectorizer', CountVectorizer()),
('nb', MultinomialNB())
])

(2) — pipe_model.fit(x_train, y_train)

1) train model using pipeline for text training. (NLP)

2) train dataset using pipe_model function. by method "fit".

Stop words.

It is a set of commonly used words in a language.

Ex:- "a", "the", "is", "are", etc.

Commonly used in Text mining and NLP to eliminate words that are so widely used that they carry very little useful information.

Wikipedia is great example for this, it provide keyword related to person or organization using this feature.

→ Limitations

DELTA Page No.
Date / /

chatbot, AI can't use this feature cuz it's remove the meaning of the sentence.

ex:- I am using laptop.

after using feature.

output → laptop { This is useless and meaning less }

We can't use stop word feature in chat bot, QA system, language translation etc

⇒ Code

DELTA Page No.
Date / /

⇒ N-GRAMS in BOW

It is a set of words, so that word vectorization has more meaningfull.

Useful in many text analytics applications where sequence of words are relevant, such as in sentiment analysis, text classification, and text generation.

ex:- Hello, this is my python code.

1-gram. 'Hello', 'this', 'is', 'my', 'python', 'code'

2-gram 'Hello, this', 'this is', 'is my', 'my python', 'python code'.

⇒ TF-IDF (Term frequency - Inverse Document freq)

→ determines the importance of word in a document
→ identify key terms and concepts in text.
→ Improve text classification and clustering.

1) term frequency (TF):
how frequently a term appears within a document.

$$TF(t,d) = \frac{\text{(Number of times term 't' appears in doc)}}{\text{(total terms in document d)}}$$

2) Inverse Document Frequency (IDF):

Measure how rare a term is across the entire collection of document

$$IDF(t) = \log\left(\frac{\text{Total number of doc}}{\text{No. of doc. with term } t}\right)$$

3) TF-IDF

→ combine TF and IDF to assign a weight to each term in a document.

→ $TF\text{-}IDF(t,d) = TF(t,d) * IDF(t)$

→ Higher TF-IDF scores indicate more important terms.

Fast Text Embedding :-

Fasttext is a word embedding that provides embedding to the character n-grams. It is the extension of the word2vec model.

word = cat
1-gram = "c", "a", "t"
2-gram = "ca", "at"
3-gram = "cat"

→ It's advantages :-

1) Better handling of rare and OOV words :
It can generate vector representations for

words it has never seen before by simply summing the vectors (n-gram).

- OOV → out of vocabulary.

→ Fasttext is often a first choice when you want to train custom word embeddings for your domain.

→ It is a technique as well as a library.

→ fasttext is pre-trained model on wikipedia.