Tokenization

Tokenization breaks the raw text into words, sentences called tokens. These tokens help in understanding the context or developing the model for the NLP. ... If the text is split into words using some separation technique it is called word tokenization and same separation done for sentences is called sentence tokenization.

```
# NLTK
import nltk
nltk.d own loa d(' -
punkt')
paragraph = " write
paragaraph here to
convert into tokens."
sentences = nltk.s ent -
_to ken ize (pa rag -
raph)
words = nltk.w ord _to -
ken ize (pa rag raph)
# Spacy
from spacy.l ang.en
import English
nlp = English()
sbd = nlp.cr eat e_p -
ipe ('s ent enc izer')
nlp.ad d_p ipe (sbd)
doc = nlp(pa rag raph)
[sent for sent in
doc.sents]
nlp = English()
doc = nlp(pa rag raph)
```

Tokenization (cont)

```
> [word for word in doc]
# Keras
from keras.preprocessing.text
import text_to_word_sequence
text_to_word_sequence(parag-
raph)
# genis
from gensim.summarization.tex-
tcleaner import split_sentences
split_sentences(paragraph)
from gensim.utils import
tokenize
list(tokenize(paragraph))
```

Bag Of Words & TF-IDF

Bag of Words model is used to preprocess the text by converting it into a bag of words, which keeps a count of the total occurrences of most frequently used words

```
# counters = List of
stences after pre
processing like tokeni -
zation, stemmi ng/ lem -
mat iza tion, stopwords
from sklear n.f eat -
ure _ex tra cti on.text
import CountV ect orizer
cv = CountV ect ori -
zer (ma x_f eatures =
1500)
```

Bag Of Words & TF-IDF (cont)

```
> X = cv.fit_transform(counter-
s).toarray()
```

Term Frequency-Inverse Document Frequency (TF-IDF):

Term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

T.F = No of rep of words in setence/No of words in sentence

IDF = No of sentences / No of sentences containing words

```
from sklearn.feature_extracti-
on.text import TfidfVectorizer
cv = TfidfVectorizer()
X = cv.fit_transform(counters).t-
oarray()
```

N-gram Language Model: An N-gram is a sequence of N tokens (or words).

A 1-gram (or unigram) is a one-word sequence.the unigrams would simply be: "I", "love", "reading", "blogs", "about", "data", "science", "on", "Analy-tics", "Vidhya".

Bag Of Words & TF-IDF (cont)

> A 2-gram (or bigram) is a two-word sequence of words, like "I love", "love reading", or "Analytics Vidhya".
And a 3-gram (or trigram) is a three-word sequence of words like "I love reading", "about data science" or "on Analytics Vidhya".

Stemming & Lemmatization

From Stemming we will process of getting the root form of a word. We would create the stem words by removing the prefix of suffix of a word. So, stemming a word may not result in actual words.

```
paragraph = " "
# NLTK
from nltk.stem import
Porter Stemmer
from nltk import
sent_t okenize
from nltk import
word_t okenize
stem = Porter Ste mmer()
sentence = sent_t oke -
niz e(p ara gra ph)[1]
words = word_t oke niz -
e(s ent ence)
[stem.s te m(word) for
word in words]
# Spacy
```

By sree017
cheatography.com/sree017/

Published 26th September, 2020.
Last updated 26th September, 2020.
Page 1 of 3.

Stemming & Lemmatization (cont)

> No Stemming in spacy
Keras
No Stemming in Keras
Lemmatization:
As stemming, lemmatization do the same but the only difference is that lemmatization ensures that root word belongs to the language
# NLTK
from nltk.stem import WordNetLemmatizer
lemma = WordNetLemmatizer()
sentence = sent_tokenize(paragraph)[1]
words = word_tokenize(sentence)
[lemma.lemmatize(word) for word in words]
# Spcay
import spacy as spac
sp = spac.load('en_core_web_sm')
ch = sp(u'warning warned')
for x in ch:
    print(ch.lemma_)
# Keras
No lemmatization or stemming

Word2Vec

```
In BOW and TF-IDF
approach semantic
information not stored.
TF-IDF gives importance
to uncommon words. There
is definitely chance of
overfitting.
  In W2v each word is
basically repres ented
as a vector of 32 or
more dimension instead
of a single number. Here
the semantic inform -
ation and relation
between words is also
preserved.
Steps:
1. Tokeni zation of the
sentences
2. Create Histograms
3. Take most frequent
words
4. Create a matrix with
all the unique words. It
also represents the
occurence relation
between the words
from gensim.models
import Word2Vec
model = Word2V ec( sen -
tences, min_co unt=1)
words = model.w v.v ocab
vector = model.w -
v[ 'fr eedom']
similar = model.w v.m -
os t_s imi lar ['f ree -
dom']
```

Stop Words

```
Stopwords are the most
common words in any
natural language. For
the purpose of analyzing
text data and building
NLP models, these
stopwords might not add
much value to the
meaning of the document.
# NLTK
from nltk.c orpus import
stopwords
from nltk.t okenize
import word_t okenize
stopwords = set(st opw -
ord s.w ord s(' eng -
lish'))
word_t okens = word_t -
oke niz e(p ara graph)
[word for word in
word_t okens if word not
in stopwords]
# Spacy
from spacy.l ang.en
import English
from spacy.l an g.e -
n.s top _words import
STOP_WORDS
nlp = English()
my_doc = nlp(pa rag -
raph)
# Create list of word
tokens
token_list = [token.text
for token in my_doc]
# Create list of word
tokens after removing
stopwords
filter ed_ sen tence =[]
```

Stop Words (cont)

```
> for word in token_list:
    lexeme = nlp.vocab[word]
    if lexeme.is_stop == False:
        filtered_sentence.append-
(word)
# Gensim
from gensim.parsing.prepro-
cessing import remove_st-
opwords
remove_stopwords(paragraph)
```

Tokenization

NLTK	Spacy	Keras	Tensorlf
dfdfd			

Parts of Speech (POS) Tagging, Chunking & NER

```
The pos(parts of speech)
explain you how a word
is used in a sentence.
In the sentence, a word
have different contexts
and semantic meanings.
The basic natural
language processing(NLP)
models like bag-of-
words(bow) fails to
identify these relation
between the words. For
that we use pos tagging
to mark a word to its
pos tag based on its
context in the data. Pos
is also used to extract
rlationship between the
words
# NLTK
```

Parts of Speech (POS) Tagging, Chunking & NER (cont)

```
> from nltk.tokenize import
word_tokenize
from nltk import pos_tag
nltk.download('averaged_per-
ceptron_tagger')
word_tokens = word_tokeniz-
e('Are you afraid of something?')
pos_tag(word_tokens)
# Spacy
nlp = spacy.load("en_core_we-
b_sm")
doc = nlp("Coronavirus: Delhi
resident tests positive for
coronavirus, total 31 people
infected in India")
[token.pos_ for token in doc]
```

Chunking:

Chunking is the process of extracting phrases from the Unstructured text and give them more structure to it. We also called them shallow parsing.We can do it on top of pos tagging. It groups words into chunks mainly for noun phrases. chunking we do by using regular expression.

```
# NLTK
word_tokens = word_tokeniz-
e(text)
```

Parts of Speech (POS) Tagging, Chunking & NER (cont)

```
> word_pos = pos_tag(word_to-
kens)
chunkParser = nltk.RegexpPars-
er(grammar)
tree = chunkParser.parse(wor-
d_pos)
```

Named Entity Recognization:

It is used to extract information from unstructured text. It is used to classy the entities which is present in the text into categories like a person, organization, event, places, etc. This will give you a detail knowledge about the text and the relationship between the different entities.

```
# Spacy
import spacy
nlp = spacy.load("en_core_we-
b_sm")
doc = nlp("Coronavirus: Delhi
resident tests positive for
coronavirus, total 31 people
infected in India")
for ent in doc.ents:
    print(ent.text, ent.start_char,
ent.end_char, ent.label_)
```