NLP CHEAT SHEET

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.download('punkt')

paragraph = " write paragraph here to convert into tokens."

sentences = nltk.sent _tokenize(paragraph)

words = nltk.word_tokenize(paragraph)

# Spacy

from spacy.lang.en

import English

nlp = English()

sbd = nlp.create_pipe('sentencizer')

nlp.add_pipe(sbd)

doc = nlp(paragraph)

[sent for sent in doc.sents]

nlp = English()

doc = nlp(paragraph)

[word for word in doc]
```

 # Keras

from keras.preproces sing.text

import text_to_word_sequence

text_to_word_sequence(paragraph)

 # genis

from gensim.summarization.textcleaner

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 preprocessing like tokenization, stemming/ lemmatization, stopwords

from sklearn.feature_extraction.text

import CountVectorizer

cv = CountVectorizer(max_features = 1500)

X = cv.fit _transform(counters).toarray()

Term Frequency -Inverse

Document Frequency (TFIDF):

Term frequency –in -verse 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 _extraction.text

```
import TfidfVectorizer

cv = TfidfVectorizer()

X = cv.fit_transform(counters).toarray()
```

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”, “Analytics”, “Vidhya”. 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 and 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_tokenize

from nltk import word_tokenize

stem = Porter Stemmer()

sentence = sent_tokenize(paragraph)[1]

words = word_tokenize(sentence)

[stem.s te m(word) for word in words]

# Spacy
```

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'warningwarned')

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 = Word2Vec(sentences, min_count=1)

```
words = model.wv.vocab

vector = model.wv[ 'freedom']

similar = model.wv.most_similar['freedom']
```

Stopwords

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.corpus

import stopwords

from nltk.tokenize

import word_tokenize

stopwords = set(stopwords.words('english'))

word_tokens = word_tokenize(paragraph)

[word for word in word_t okens if word not in stopwords]

# Spacy

from spacy.lang.en

import English

from spacy.lang.en.stop _words

import STOP_WORDS

nlp = English()

my_doc = nlp(paragraph)

# Create list of word tokens

token_list = [token.text for token in my_doc]

# Create list of word tokens after removing stopwords

filtered_ sentence =[]

for word in token_ list:
```

```
lexeme = nlp.vocab[word]

if lexeme.is _stop == False:

filtered_ sentence.append (word)

# Gensim

from gensim.parsing.preprocessing

import remove_stopwords

remove_stopwords(paragraph)
```

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 relationship between the words.

```
# NLTK

from nltk.tokenize

import word_tokenize

from nltk import pos_tag

nltk.download('averaged_ perceptr n_tagger')

word_tokens = word_tokenize('Are you afraid of something?')

pos_tag(word_tokens)

# Spacy

nlp = spacy.load("en_core_web_sm")

doc = nlp("Coronavirus: Delhi resident tests positive for corona virus, 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_tokenize(text)

word_pos = pos_tag(word_to kens)

chunkP arser = nltk.RegexpParser(grammar)

tree = chunkParser.parse(word_pos)

Named Entity Recognization: It is used to extract inform ation from unstructured text. It is used to classy the entities which is present in the text into categories like a person, organi zation, 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 _web_sm")

doc = nlp("Co ron avirus: Delhi resident tests positive for corona virus, total 31 people infected in India")

for ent in doc.ents:

print(ent.text, ent.start _char, ent.end_char, ent.label_)