### 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.

```python
# 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)
```

### Tokenization (cont)

```python
[word for word in doc]
# Keras
from keras.preprocessing.text import Tokenizer
sentence = sent_tokenize(paragraph)

# gensim
from gensim.summarization import TextCleaner
split_sentences = split_sentences(paragraph)
from gensim import tokenization
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.

```python
# counters
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features = 1500)
X = cv.fit_transform(counters).toarray()
```

### Bag Of Words & TF-IDF (cont)

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.
- \( T.F = \frac{No \ of \ rep \ of \ words \ in \ sentence}{No \ of \ words \ in \ sentence} \)
- \( I.D.F = \frac{No \ of \ sentences}{No \ of \ sentences \ containing \ words} \)
- \( X = cv.fit_transform(counters).toarray() \)
- N-gram Language Model: An N-gram is a sequence of N tokens (or words). A 1-gram (or unigram) is one word sequence. The unigrams would simply be: “I”, “love”, “reading”, “blogs”, “about”, “data”, “science”, “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.

```python
# NLTK
from nltk.stem import PorterStemmer
from nltk import sent_tokenize
stem = PorterStemmer()
sentence = sent_tokenize(paragraph)[1]
words = word_tokenize(sentence)
[stem.stem(word) for word in words]
```

---

### 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”.

---

**Cheatography**

By sree017

cheatography.com/sree017/

Published 26th September, 2020.

Last updated 26th September, 2020.

Page 1 of 3.

Sponsored by CrosswordCheats.com

Learn to solve cryptic crosswords!

http://crosswordcheats.com
**Cheatography**

### Stemming & Lemmatization (cont)

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

### 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 represented as a vector of 32 or more dimension instead of a single number.

Here the semantic information and relation between words is also preserved.

Steps:
1. Tokenization of the sentences
2. Create Histograms
3. Take most frequent words
4. Create a matrix with all the unique words. It also represents the occurrence 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')

### 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.corpus import stopwords

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

word_tokens = word_tokenize(paragraph)

[word for word in word_tokens if word not in stopwords]

# Spacy

from spacy.lang.en import 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 = []

### Stop Words (cont)

for word in token_list:
    lexeme = nlp.vocab[word]
    if lexeme.is_stop == False:
        filtered_sentence.append(word)

# Gensim

gensim.parsing.preprocessing import remove_stopwords

remove_stopwords(paragraph)

### Tokenization

<table>
<thead>
<tr>
<th>NLTK</th>
<th>Spacy</th>
<th>Keras</th>
<th>Tensorflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>dfdf</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

The pos(parts of speech) explain 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
Parts of Speech (POS) Tagging, Chunking & NER (cont)

```python
from nltk.tokenize import word_tokenize
from nltk import pos_tag
nltk.download('averaged_perceptron_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 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_tokenize(text)
```

```python
word_pos = pos_tag(word_tokens)
chunkParser = nltk.RegexpParser(grammar)
tree = chunkParser.parse(word_pos)

Named Entity Recognition:
It is used to extract information from unstructured text. It is used to classify 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_web_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_)
```