Natural Language Processing (NLP)

What is NLP?

- deals with the interactions between computers and human languages
- how to program computers to process and analyze large amounts of natural language data
- computers can read text, hear speech, interpret it, measure sentiment and determine which parts are important

What is NLP?

- ML Algorithm study millions of text examples written by humans
- Algorithms gain understanding of the context
- This helps in differentiating between meaning of various texts
- App: Optical Character Recognition (OCR), Speech Recognition, Machine Translation, and Chatbots

NLP around us

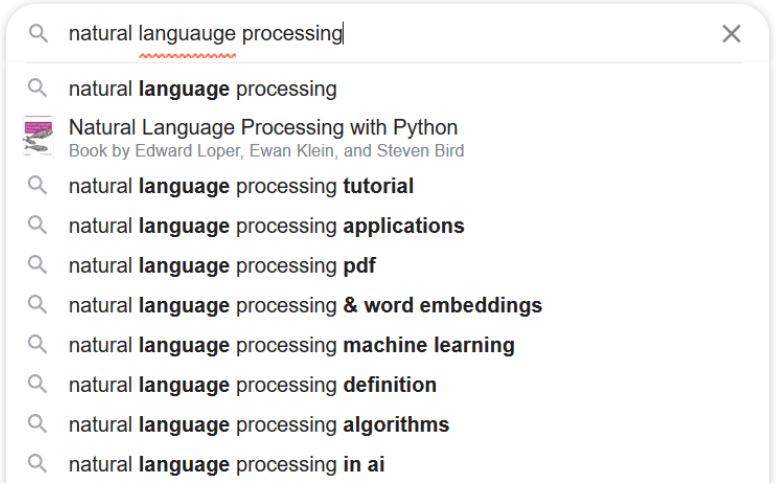
Spam Filter

Spam

Inbox

Auto-Complete

Google



Topics NLP

- Spam Filter
- Sentiment Analysis
- Auto Summarizing Articles
- Article Classification

Good to know:

- Python – Regular Exp. , Graph, library(nlargest, FreqDist)
- ML – Classification(Supervised Learning)
  - Clustering (UnSupervised Learning)

Text Preprocessing

Computer clean/process/vectorize text and develops understanding by building model

Stopwords, punctuations, etc

E.g: tokenization, stopwords/punctuation – removal, stemming, countvectorizer, etc

Library – nltk

nltk.download()

Machine Learning – Model Building Steps

1. Import Dataset
2. Data Analysis
3. Clean Text
4. Tokenize
5. Vectorize – Text to numeric form
6. Machine Learning Algorithms (Supervised, UnSupervised)

Text Preprocessing

Structure vs UnStructured Data

- Can be displayed in rows, columns, DB
- Requires less storage
- Useful for analysis
- Easier to manage

- Can’t be displayed in rows, columns, DB efficiently
- Requires much more storage
- Can’t be used for analysis until cleaned
- Very difficult to manage

Regular Expressions

- Reg-Ex for searching a pattern in a text
- sub(), split(), findall

- [a-z] -> b
- [a-z]+ -> natural
- [0-9]+ -> 819823
- [a-z0-9]+ -> year2020
- r'\W' matches any non-word character (equal to [^a-zA-Z0-9_])
- r'\s' matches single space

Link:

https://regex101.com/r/jE4cE4/62

Reg-Ex -> Application

- Email format checking
- Passwords meet criteria
- Searching for log files with specific format
- Cleaning the texts

Text Preprocessing/Cleaning

Computer clean/process text and develops understanding by building model

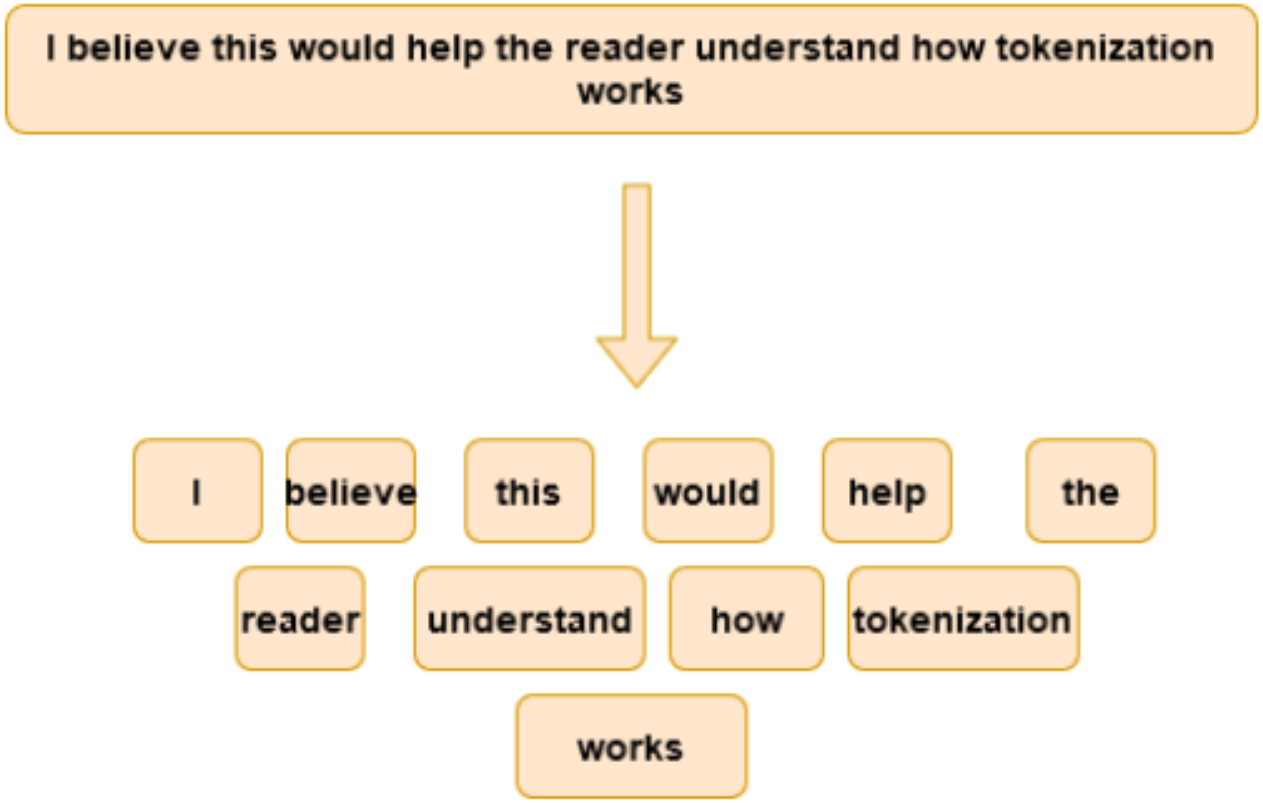
- Tokenization
- Stopword Removal
- Remove Punctuation
- N- Grams
- Stemming
- Lemmatizing
- Word Sense Disambiguation

Tokenization

- Task of breaking a text into pieces called as token

Types:

- Word Tokenization
- Sentence Tokenization



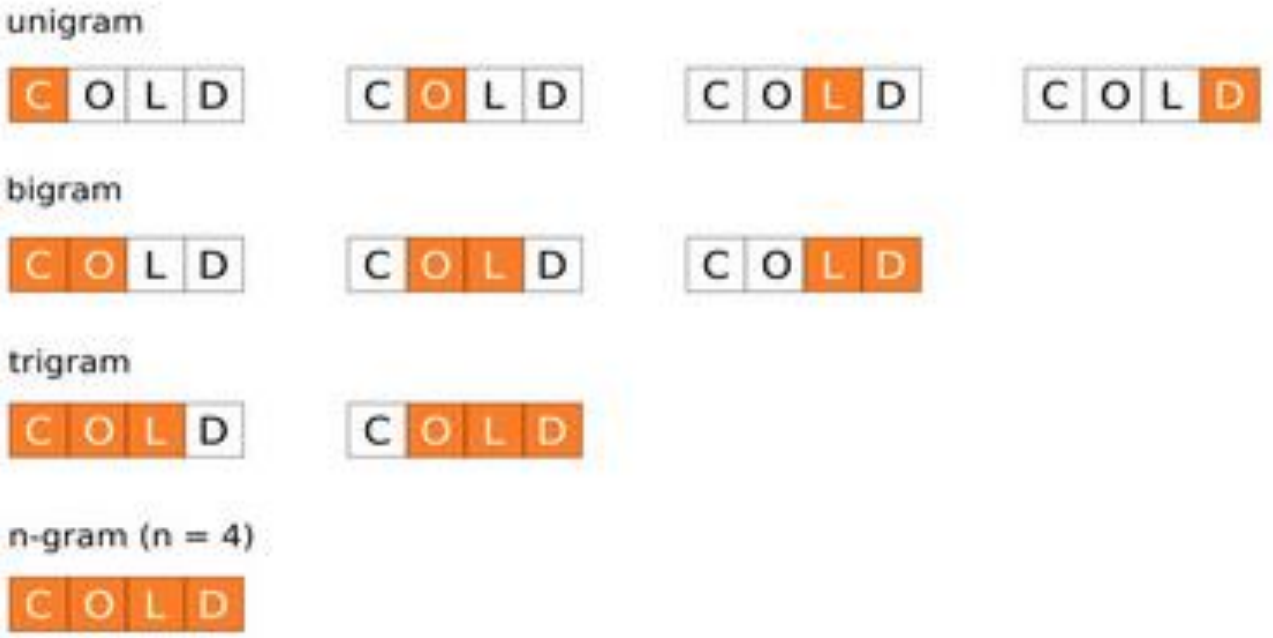
Stop Word Removal

- Stopwords are the English words which does not add much meaning to a sentence.
- They can safely be ignored without sacrificing the meaning of the sentence.
- A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore.

Text with Stop Words	After Removing Stop Words
I believe this would help the reader understand how stop words works	'I', 'believe', 'would', 'help', 'reader', 'understand', 'stop', 'words', 'works'
as well as realize its importance	'well', 'realize', 'importance'

N-Grams

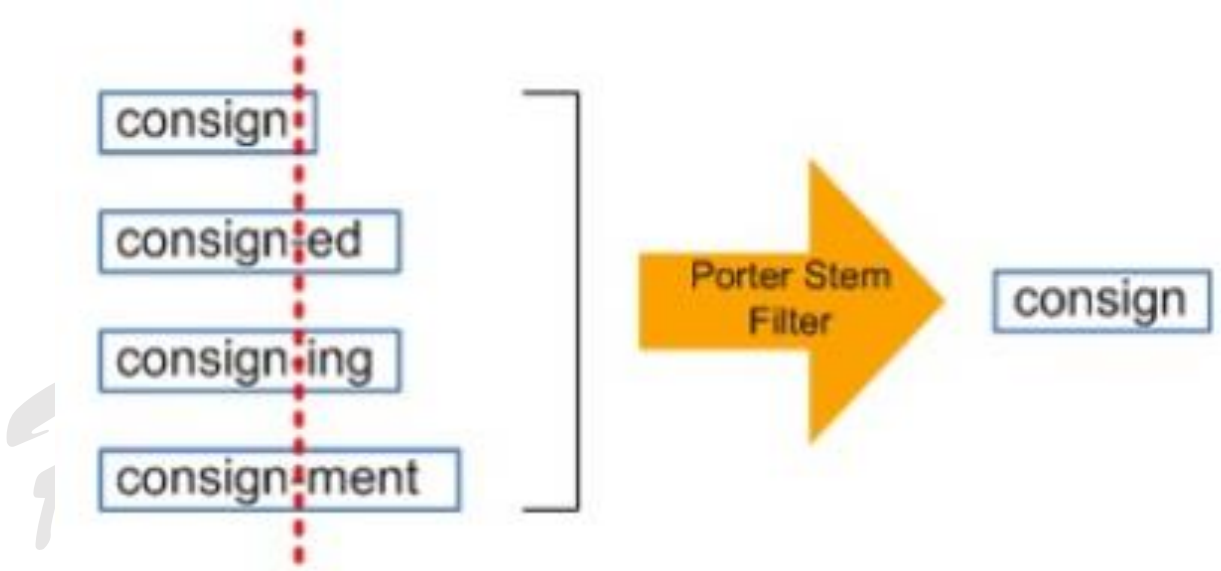
- An n-gram is a contiguous sequence of n items from a given sample of text or speech.



- While typing we get suggestion

Stemming

- Stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form



- E.g: Search Engine

Word Sense Disambiguation

- WSD is identifying which sense of a word (i.e. meaning) is used in a sentence, when the word has multiple meanings.

Vectorizing

- Process of converting words into numeric form

Why?

- ML algorithm understand numbers
- So text need to be converted into numbers

E.g:

- Count Vectorizer
- TF-IDF(TfidfVectorizer)
- HashingVectorizer

CountVectorizer

- Provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary.
- The same vectorizer can be used on documents that contain words not included in the vocabulary. These words are ignored and no count is given in the resulting vector.
- Issue: Appearance of “the”
- Each column represents one word, count refers to frequency of the word
- Sequence of words are not maintained

CountVectorizer

Text:

- 0 This is the first document from heaven
- 1 but the second document is from mars
- 2 And this is the third one from nowhere
- 3 Is this the first document from nowhere?

- Vocabulary:
- {'this': 13, 'is': 6, 'the': 11, 'first': 3, 'document': 2, 'from': 4, 'heaven': 5, 'but': 1, 'second': 10, 'mars': 7, 'and': 0, 'third': 12, 'one': 9, 'nowhere': 8}

Array:

- [[0 0 1 1 1 1 1 0 0 0 0 1 0 1]
- [0 1 1 0 1 0 1 1 0 0 1 1 0 0]
- [1 0 0 0 1 0 1 0 1 1 0 1 1 1]
- [0 0 1 1 1 0 1 0 1 0 0 1 0 1]]

TF-IDF(Tfidf Vectorizer)

- TF-IDF are word frequency scores that try to highlight words that are more interesting, e.g. frequent in a document but not across documents.
- The importance is in scale of 0 & 1

Term Frequency: This summarizes how often a given word appears within a document.
Inverse Document Frequency: This downscales words that appear a lot across documents.

Adv:

- Feature vector much more tractable in size
- Frequency and relevance captured

DisAdv:

- Context still not captured

TF-IDF(Tfidf Vectorizer)

Text:

- 0 This is the first document from heaven
- 1 but the second document is from mars
- 2 And this is the third one from nowhere
- 3 Is this the first document from nowhere?

- {'this': 13, 'is': 6, 'the': 11, 'first': 3, 'document': 2, 'from': 4, 'heaven': 5, 'but': 1, 'second': 10, 'mars': 7, 'and': 0, 'third': 12, 'one': 9, 'nowhere': 8}

- [1.91629073 1.91629073 1.22314355 1.51082562 1. 1.91629073
- 1. 1.91629073 1.51082562 1.91629073 1.91629073 1. 1.91629073 1.22314355]

Hashing Vectorizer

- Issue with Counts and frequencies – vocabulary can become very large
- Work around is to use a one way hash of words to convert them to integers
- No vocabulary is required and you can choose an arbitrary-long fixed length vector
- Downside - no way to convert the encoding back to a word

Hashing Vectorizer

- Step 1:

	The food was good	The ambience was good, service was exellent
The	1	1
food	1	0
was	1	2
good	1	1
ambience	0	1
service	0	1
excellent	0	1

- Step2:

	The food was good	The ambience was good, service was exellent
0	1	1
1	1	0
2	1	2
3	1	1
4	0	1
5	0	1
6	0	1

-

Hashing Vectorizer

- Step 3:

	The food was good	The ambience was good, service was exellent
0	1	1
1	1	0
2	1	2
3	1	1
4	0	2
5	0	1

Spam Filter: CountVectorizer

- Article Classification(E.g: Spam Classification) using bag-of-words representation (BOW)

- corpus = [
- 'i earn 20 lakh rupees per month just chitchating on the net!',
- 'are you free for a meeting anytime tomorrow?',
-]

Python Code

```
import pandas as pd
corpus = [
    'i earn twenty lakh rupees per month just chitchating on the net!',
    'are you free for a meeting anytime tomorrow?',
]
df = pd.DataFrame({'Text':corpus})
from sklearn.feature_extraction.text import CountVectorizer
count_v = CountVectorizer()
X = count_v.fit_transform(df.Text).toarray()
print(X)
print(count_v.vocabulary_)
```

```
new_txt = ['io etrn are you free ruppee for a monnth meeting chitcchting anytime tomorrow neet']
df_new = pd.DataFrame({'new_txt':new_txt})
y = count_v.transform(df_new.new_txt).toarray()
print(y)
```

Spam Filter: CountVectorizer

- Vocabulary:

{'anytime': 0, 'are': 1, 'chitchating': 2, 'earn': 3, 'for': 4, 'free': 5, 'just': 6, 'lakh': 7, 'meeting': 8, 'month': 9, 'net': 10, 'on': 11, 'per': 12, 'rupees': 13, 'the': 14, 'tomorrow': 15, 'twenty': 16, 'you': 17 }

- 'i earn 20 lakh rupees per month just chitchating on the net!',
- [0 0 1 1 0 0 1 1 0 1 1 1 1 1 0 1 0]

- 'are you free for a meeting anytime tomorrow?',
- [1 1 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 1]

Spam Filter: CountVectorizer

- **New Mail**:
- io etrn are you free ruppee for a monnth meeting chitcchting anytime tomorrow neet
- *With Existing Count Vector*:
- [1 1 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 1]

- *With NEW Count Vector* :
- [1 1 1 0 0 1 1 1 1 0 0 1 1 0 1 0 0 0 0 1 0 1 0 1]

Spam Filter: CountVectorizer

F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	Result
0	0	1	1	0	0	1	1	0	1	1	1	1	1	1	0	1	0	Spam
1	1	0	0	1	1	0	0	1	0	0	0	0	0	0	1	0	1	Not Spam
1	1	0	0	1	1	0	0	1	0	0	0	0	0	0	1	0	1	Not Spam

F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F20	F21	F22	F23	F24	F25
1	1	1	0	0	1	1	1	1	0	0	1	1	0	1	0	0	0	0	1	0	1	0	1

- *With NEW Count Vector* :
- [1 1 1 0 0 1 1 1 1 0 0 1 1 0 1 0 0 0 0 1 0 1 0 1]

Hashing

- Apply Hash Function
- “Rishi Bansal” → 23
- “Rashi Bansal” → 72
- Output number depend on Hash function

Features:

- Same value for same string
- Collison: Possibility of same value for different string