# Natural Language Processing: Introduction

Problems in hand:

We have text message as input and we want to predict if it is spam
Given facebook post we want to classify if it is political, funny, adult, meme, etc.
Classify comment into positive, negative and neutral category.

Some points to remember:

1. Machine/computer can't understand language.
2. Each character or symbol is binary number in machine language.
3. Machine only understands numbers not words.
4. Machine learning models take vector/matrics as input.

**Now we can either teach machine a language or convert our language into numbers**

it is not possible to teach machine a language, so let's see how we can convert language into numbers so that machine can process it.

Get code in sem 1 - sub 3 at https://github.com/mayurkagathara/Python_Practice/tree/main/UOH_AAIC
or scan QR code

## INDEX

We will see this in next modules.

# 1. Text Pre-processing

## Stop word removal

In language we use so many common words like article (a, an, the), tense specific words (will, shall, did), pronoun specific words(am, is, are, has, had, etc.

This words have no importance in the meaning of a sentence and in most of the cases don't hold the major essence of a sentence.

So we can remove them to reduce the unwanted dimensionality.

HOW TO GET IT? from nltk.corpus import stopwords

## Stemming

In language we have so many forms of a same word or verb.

EG- have, has, had, having, go, going, man, men, mouse, mice, smart, smarter.

Stemming is used to remove suffix part of the word at the end and gives the root of the word. Sometimes the root of a word has no meaning at all.

We have below stemmers in NLTK library: (most used in order)

1. Snowball stemmer
2. Porter stemmer
3. Lancaster stemmer

## Lemmatization

nltk.download('wordnet')

Lemmatization is better than stemming. It does morphological analysis of a word and bring the contect to the word. It will link all the words with the similar meaning to the one word.

It used wordnet data to perform lemmatization.

```
word          snow_ball     porter        lemma
-----------------------------------------------------------
cared         care          care          cared
university    univers       univers       university
mice          mice          mice          mouse
easily        easili        easili        easily
singing       sing          sing          singing
language      languag       languag       language
corpora       corpora       corpora       corpus
singer        singer        singer        singer
sportingly    sport         sportingli    sportingly
rocks         rock          rock          rock
```

## Tokenization nltk.download('punkt')

After performing cleaning and other transformation process on raw text, we can now divide whole corpus into smaller parts (in words, sentences). They are called tokens and this process is called tokenization.

Corpus: 'Hi Machine learning aspirants. Hope you are doing great. NLP is good.'

sentence tokenization:

'Hi Machine learning aspirants.'
'Hope you are doing great.'
'NLP is good.'

word tokenization:

'Hi', 'Machine', 'learning', 'aspirants', '.', 'Hope', 'you', 'are', 'doing', 'great', '.', 'NLP', 'is', 'good', '.'

NOTE: We won't get any stopwords, punctuation marks in real. For example we have not perform cleaning on the data, thus we are getting full stop also.

---

Now we have completed cleaning, transforming (stemming, lemmatizing) and tokenizing corpus(raw data) into sentences or into words.

Still the data is in specific language and not in numbers.

So our next task is to convert these words into numbers and documents into vectors(of numbers).

We will cover Bag of words, Binary bag of words, TF-IDF, word2vec.

### Terminology ALERT:

Whole dataset or raw_data is called corpus or corpora.
Each data point (review, message, post..) is called document.
In each document we have multiple sentences/words.

## 2. Bag Of Words (BOW)

There are two flavours of BOW - Simple bow and binary bow.

1. Perform cleaning and transformation on corpus.
2. Count the words in the corpus and create word dictionary.
3. Vectorize each sentence using word dictionary (bow).

Example

**Corpus:** "You cannot believe in god until you believe in yourself. The greatest sin is to think that you are weak. Believe in yourself and the world will be at your feet."

| Sentences after cleaning and removing stop words. | |
|---|---|
| 0 | 'believe god believe' |
| 1 | greatest sin think weak |
| 2 | believe world feet |

**BOW(bag of words):** 'believe': 3, 'god': 1, 'greatest': 1, 'sin': 1, 'think': 1, 'weak': 1, 'world': 1, 'feet': 1

| Simple Bag Of Words | | 'believe' | 'feet' | 'god' | 'greatest' | 'sin' | 'think' | 'weak' | 'world' |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 'believe god believe' | 3 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | greatest sin think weak | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 2 | believe world feet | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |

| Binary Bag Of Words | | 'believe' | 'feet' | 'god' | 'greatest' | 'sin' | 'think' | 'weak' | 'world' |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 'believe god believe' | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | greatest sin think weak | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 2 | believe world feet | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |

This is the simplest technique. In simple bag of words we use the count of word in the corpus to vectorize the sentences.

In case of BInary bag of words, we use 1 if count is greater than 0, else 0.

## 3. TF-IDF (Term Frequence Inverse Document Frequency)

1. Perform cleaning and transformation on corpus.
2. Calculate Term Frequency (TF) for each word in sentence.
3. Calculate Inverse Document Frequency (IDF) for all unique words in the corpus.
4. Calculate TFIDF = TF x IDF.

*TF is calculated document (sentecnce) wise for each word in the document.*

$$TF = \frac{\text{number of words in sentence}}{\text{total number of words in sentence}}$$

*IDF is calculated for all unique words in the corpus.*

$$IDF = ln\left(\frac{\text{total number of sentences}}{\text{number of sentences containing word}}\right)$$

*we take natural log because word frequency is distributed exponentially.*

**Example**

Corpus = [ 'this is the first document mostly',
'this document is the second document',
'and this is the third',
'is this document' ]

| IDF | | |
|---|---|---|
| N=4 | Ni | ln (N / Ni) |
| this | 4 | 0 |
| is | 4 | 0 |
| the | 3 | 0.2876821 |
| first | 1 | 1.3862944 |
| document | 3 | 0.2876821 |
| mostly | 1 | 1.3862944 |
| second | 1 | 1.3862944 |
| and | 1 | 1.3862944 |
| third | 1 | 1.3862944 |

**Ni** = number of sentences containing word
**N** = total number of sentences = 4

*We take the nine unique words and then calculate IDF for each word and create the dictionary as shown on left.*

*Now we will calculate term frequency for each word in all sentences.*

*Simultaneously we can calculate TFIDF value by multiplying TF and IDF value.*

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **this is the first document mostly** | **this** | **is** | **the** | **first** | **document** | **mostly** | second | and | third |
| **TF** | 0.1666667 | 0.1666667 | 0.1666667 | 0.1666667 | 0.1666667 | 0.1666667 | 0 | 0 | 0 |
| **IDF** | 0 | 0 | 0.2876821 | 1.3862944 | 0.2876821 | 1.3862944 | 1.3862944 | 1.3862944 | 1.3862944 |
| **TFIDF** | **0** | **0** | **0.048** | **0.231** | **0.048** | **0.231** | **0** | **0** | **0** |
| **this document is the second document** | **this** | **is** | **the** | first | **document** | mostly | **second** | and | third |
| **TF** | 0.1666667 | 0.1666667 | 0.1666667 | 0 | 0.3333333 | 0 | 0.1666667 | 0 | 0 |
| **IDF** | 0 | 0 | 0.2876821 | 1.3862944 | 0.2876821 | 1.3862944 | 1.3862944 | 1.3862944 | 1.3862944 |
| **TFIDF** | **0** | **0** | **0.048** | **0** | **0.096** | **0** | **0.231** | **0** | **0** |
| **and this is the third** | **this** | **is** | **the** | first | document | mostly | second | **and** | **third** |
| **TF** | 0.2 | 0.2 | 0.2 | 0 | 0 | 0 | 0 | 0.2 | 0.2 |
| **IDF** | 0 | 0 | 0.2876821 | 1.3862944 | 0.2876821 | 1.3862944 | 1.3862944 | 1.3862944 | 1.3862944 |
| **TFIDF** | **0** | **0** | **0.058** | **0** | **0** | **0** | **0** | **0.277** | **0.277** |
| **is this document** | **this** | **is** | the | first | **document** | mostly | second | and | third |
| **TF** | 0.3333333 | 0.3333333 | 0 | 0 | 0.3333333 | 0 | 0 | 0 | 0 |
| **IDF** | 0 | 0 | 0.2876821 | 1.3862944 | 0.2876821 | 1.3862944 | 1.3862944 | 1.3862944 | 1.3862944 |
| **TFIDF** | **0** | **0** | **0** | **0** | **0.096** | **0** | **0** | **0** | **0** |

| TF-IDF | this | is | the | first | document | mostly | second | and | third |
|---|---|---|---|---|---|---|---|---|---|
| this is the first document mostly | 0 | 0 | 0.048 | 0.231 | 0.048 | 0.231 | 0 | 0 | 0 |
| this document is the second document | 0 | 0 | 0.048 | 0 | 0.096 | 0 | 0.231 | 0 | 0 |
| and this is the third | 0 | 0 | 0.058 | 0 | 0 | 0 | 0 | 0.277 | 0.277 |
| is this document | 0 | 0 | 0 | 0 | 0.096 | 0 | 0 | 0 | 0 |

## 4. n-grams

(uni-gram, bi-grams, tri-grams,...,n-grams)

In any langurage single word doesn't always convey the meaning but the pair of two, three or n consecutive words have the better meaning than single word

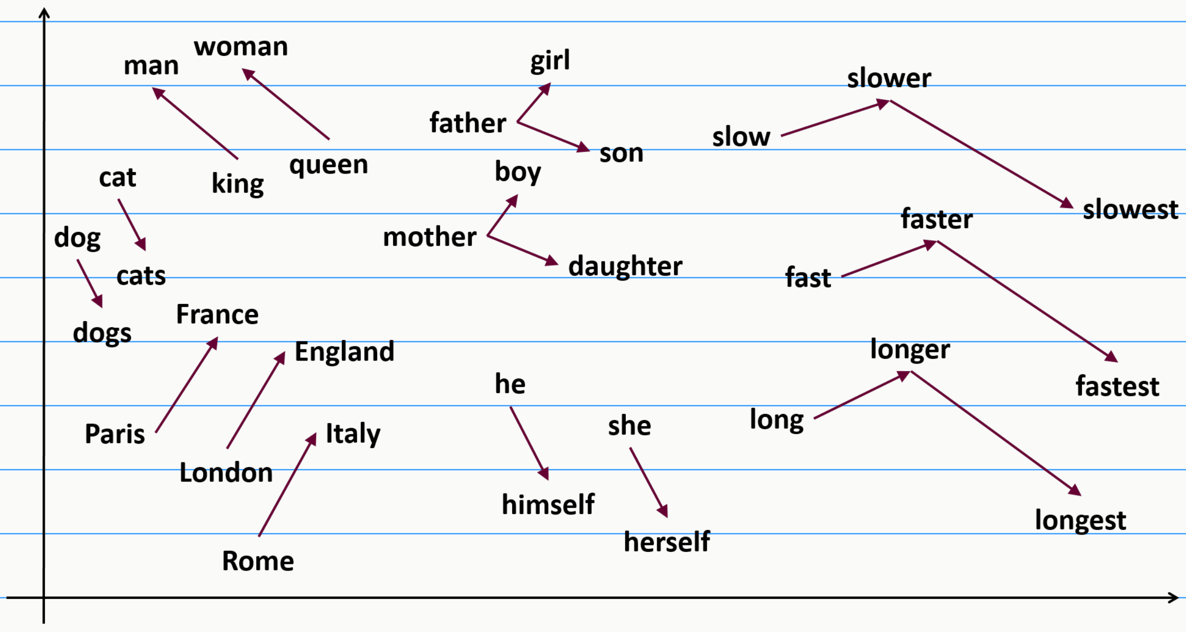
Sentences:
1. She has not passed the big data examination with good marks

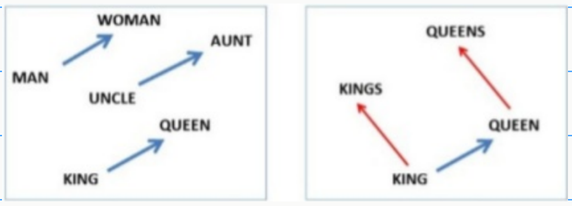
| | |
|---|---|
| uni-gram | She, has, not, passed, the, big, data, examination, with, good, marks |
| bi-grams | 'She has', 'has not', 'not passed', 'passed the', 'the big', 'big data', 'data examination', 'examination with', 'with good', 'good marks' |
| tri-grams | 'She has not', 'has not passed', 'not passed the', 'passed the big', 'the big data', 'big data examination', 'data examination with', 'examination with good', 'with good marks' |

## 5. word2vec

word2vec is powerful technique used neural network model inside to learn words inside a corpus and then vectorize each word such that synonym or like words are grouped together in n-dimension space.

The cosine similarity between 2 vectors represent the level of semantic similarity between the words represented by those vectors.





As we can observe, the semantic meaning between two words are preserved in terms of cosine similarities between vectors in n-dim space.

*Finally we have converted our words into vector or number. Now we need to convert our document(sentence) into vector. There are two methods to achieve this. Once documents are vectorized we can use any techniques in ML/DL to predict the dependent variable.*

## 6. Sentence vectorization

### a. Average word2vec

*First we get vectors for all words using word2vec.*

Sentence_Vector = 1/n (word_1_vector + word_2_vector + ... + word_n_vector)

### b. TF-IDF weighted word2vec

*First we calculate TF-IDF get vectors for all words using word2vec.*

*Each words get same weight in average word2vec. But in TF-IDF weighted word2vec, given word get weight based on TF-IDF value.*

Let t_i be the TF-IDF of word_i and
wi_vec be the word2vec of word_i

Sentence_Vector = t_1*w1_vec + t_2*w2_vec + ... + t_n*wn_vec

*Notice that we don't need to take average in case TF-IDF, because TF-IDF value itself is weight of given word.*

Follow me:

Linkedin: https://www.linkedin.com/in/mayur-kagathara/

Github: https://github.com/mayurkagathara