NATURAL LANGUAGE PROCESSING

- Allows computers to understand language naturally, as a person does

Text preprocessing

(1) Data cleaning

a. Lowercasing
- The method lower() converts all uppercase characters into lower case

b. Punctuation Removal
- Removing punctuation is a crucial step

import string
string.punctuation → returns all punctuations

Then we can remove using ~~punct~~ lambda function

c. Stop words removal
- Words that frequently occur in sentences and carry no significant meaning. Data must be in lowercase
- We use the NLTK library

(d) Spelling correction
- Improves model accuracy
- We use textblob library

(2) Tokenization
- Splitting text into [illegible] word (tokenizing).

- Sentence tokenizing splits a paragraph into meaningful sentences
- Word tokenizing splits a sentence into unit meaningful words
- We use NLTK

(f) Stemming
- Converts words into their root word using some set of rules irrespective of meaning, i.e

fish, fishes, fishing ⇒ fish
- We use NLTK

g. Lemmatization
- Converts words into their root word using vocabulary mapping. Done with help of part of speech and its meaning, hence it doesn't generate meaningless root words
- We use NLTK: chatbot

EDA
- We explore & understand the data

(1) Word frequency in Data
- Counting unique words in our data gives an idea about our data's most frequent & least frequent
- We drop the least frequent
- We use NLTK's FreqDist()

TEXT PREPROCESSING

III Word Embedding I

a. Bag of words

- Simplest form of text representation in numbers. Like the term itself, we can represent a sentence as a bag of word vectors

Eg

S1: He is a good boy.
S2: She is a good girl
S3: Boys and girls are good

↓ stop words

S1: good boy
S2: good girl
S3: Boy girl good

Then we derive vectors; create a histogram

Words	Frequency
good	3
boy	2
girl	2

Features

	good	boy	girl	output
S1	1	1	0	
S2	1	0	1	
S3	1	1	1	

i) Binary BoW

Con - Ignores order of the words
- [illegible] semantic meaning

b. Term Frequency - Inverse Document Frequency (TF-IDF)

TF-IDF: is a numerical statistic that is intended to reflect how important a word is to a document in a collection

TF: Measure of how frequently a term t appears in a document d

$$tf_{t,d} = \frac{n_{t,d}}{\text{Number of terms in the document}}$$

n = No. of times the term t appears in a document

IDF: Measure of how important a term is

$$idf_t = \log \frac{\text{No. of documents}}{\text{No. of documents with term 't'}}$$

$$(tf\_idf)_{t,d} = tf_{t,d} * idf_t$$

c. N-grams

Contains sequence (N) of words or symbols, or tokens in a document

Implementing pre-trained word ...

pip install --upgrade gensim -i

```
import gensim
from gensim.models import word2vec
from gensim.models.word2vec import Word2Vec
# gensim is a library, it has API through which
  we can download model

import gensim.downloader as api
print(list(gensim.downloader.info()['models'].keys()))
#prints list of models available in gensim API

                                        dim
wv = api.load('glove-twitter-50') # loads a pretrained
  model
wv = api.load( ..... )
wv.save('  path/        ') # save model in the
  local machine

wv['apple'] # ~~vector~~ returns vector rep of the word

from gensim.models import KeyedVectors
wv = KeyedVectors.load('  path  ') # load the
  saved model from the local machine

wv.similarity("apple", "mango") #

    # Embedding matrix
~~Embedding_dim~~
embedding_matrix = np.zeros((input_dim, 100))
for word, i in word_index.items():
    if word in wv:
        embedding_matrix[i] = wv[~~model~~ word]

model = Sequential()
model.add(Embedding(input_dim, 100, weights
  = [embedding_matrix], input_length = max_len,
```