**EXPERIMENT - 1**

**Aim**: Perform pre-processing of text on any dataset

**Theory**:

We performed a series of steps under each component based on the general outline above.

1. Remove HTML tags
2. Remove extra whitespaces
3. Convert accented characters to ASCII characters
4. Expand contractions
5. Remove special characters
6. Lowercase all texts
7. Convert number words to numeric form
8. Remove numbers
9. Remove stopwords
10. Lemmatization
11. Tokenisation
12. Stemming
13. Normalisation
14. POS Tagging

CODE:

| from indicnlp.normalize.indic\_normalize import IndicNormalizerFactory  from idatasets import load\_devdas  from nltk.stem import PorterStemmer  from nltk.corpus import stopwords  from nltk.tokenize import word\_tokenize  from nltk.stem import WordNetLemmatizer  from nltk.stem import PorterStemmer  import spacy  import pandas as pd  import stanfordnlp  import requests  from bs4 import BeautifulSoup  import unicodedata  import contractions  from word2number import w2n  import re  import stanza  from inltk.inltk import setup  from inltk.inltk import tokenize  URL = "http://www.values.com/inspirational-quotes"  r = requests.get(URL)  text = r.content |
| --- |

**Remove HTML Tags**

If the reviews or texts are web-scraped, chances are they will contain some HTML tags. Since these tags are not useful for our NLP tasks, it is better to remove them.

| def remove\_html(text):  soup = BeautifulSoup(text, "lxml")  text = soup.get\_text()  return str(text) |
| --- |

| output1 = remove\_html(text)  output1 |
| --- |

| \n\n\n\n\n\nInspirational Quotes - Motivational Quotes - | The Foundation for a Better Life\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n window.dataLayer = window.dataLayer || [];\n function gtag(){dataLayer.push(arguments);}\n gtag('js', new Date());\n gtag('config', 'UA-1179606-29');\n \n\n window.fbAsyncInit = function() {\n FB.init({\n appId : '483774921971842',\n autoLogAppEvents :... |
| --- |

**Removing extra whitespaces and tabs**

Extra whitespaces and tabs do not add any information to text processing.

| def remove\_whitespace(text):  text = " ".join(text.split())  return text |
| --- |

| output2 = remove\_whitespace(output1)  output2 |
| --- |

| Inspirational Quotes - Motivational Quotes - | The Foundation for a Better Life window.dataLayer = window.dataLayer || []; function gtag(){dataLayer.push(arguments);} gtag('js', new Date()); gtag('config', 'UA-1179606-29'); window.fbAsyncInit = function() { FB.init({ appId : '483774921971842', autoLogAppEvents : true, xfbml : true, version : 'v6.0' }); }; (function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm.start': new Date().getTime()... |
| --- |

**Convert Accented Characters**

Words with accent marks like “latté” and “café” can be converted and standardized to just “latte” and “cafe”, else our NLP model will treat “latté” and “latte” as different words even though they are referring to same thing.

| def accented\_to\_ascii(text):  try:  text = unicode(text, "utf-8")  except (TypeError, NameError): # unicode is a default on python 3  pass  text = unicodedata.normalize("NFD", text)  text = text.encode("ascii", "ignore")  text = text.decode("utf-8")  return str(text) |
| --- |

| output3 = accented\_to\_ascii(output2)  output3 |
| --- |

**Expand Contractions**

Contractions are shortened words, e.g., don’t and can’t. Expanding such words to “do not” and “can not” helps to standardize text.

We use the contractions module to expand the contractions.

| def expand\_contractions(text):  expanded\_words = []  for word in text.split():  expanded\_words.append(contractions.fix(word))  expanded\_text = " ".join(expanded\_words)  return expanded\_text |
| --- |

| output4 = expand\_contractions(output3)  output4 |
| --- |

| Inspirational Quotes - Motivational Quotes - | The Foundation for a Better Life window.dataLayer = window.dataLayer || []; function gtag(){dataLayer.push(arguments);} gtag('js', new Date()); gtag('config', 'UA-1179606-29'); window.fbAsyncInit = function() { FB.init({ appId : '483774921971842', autoLogAppEvents : true, xfbml : true, version : 'v6.0' }); }; (function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm.start': new Date().getTime(),... |
| --- |

**Removing Special Characters**

Special characters, as you know, are non-alphanumeric characters. These characters are most often found in comments, references, currency numbers etc. These characters add no value to text-understanding and induce noise into algorithms.

| def remove\_special(text):  text = text.split()  text = " ".join(x for x in text if not x.isalnum())  text = text.split()  special\_char\_list = ["$", "@", "#", "&", "%"]  text = " ".join([k for k in text if k not in special\_char\_list])  text = " ".join(text.split())  return text |
| --- |

| output5 = remove\_special(output4)  output5 |
| --- |

| - - | window.dataLayer = window.dataLayer || []; gtag(){dataLayer.push(arguments);} gtag('js', Date()); gtag('config', 'UA-1179606-29'); window.fbAsyncInit = function() { FB.init({ : '483774921971842', : true, : true, : 'v6.0' }); }; (function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm.start': Date().getTime(),event:'gtm.js'});var f=d.getElementsByTagName(s)[0], j=d.createElement(s),dl=l!='dataLayer'?'&l='+l:'';j.async=true;j.src= … |
| --- |

**Lowercase**

Changing case to lower can be achieved by using lower function.

| def text\_to\_lowercase(text):  text = text.lower()  return text |
| --- |

| output6 = text\_to\_lowercase(output5)  output6 |
| --- |

**Treatment for Numbers**

There are two steps in our treatment of numbers.

One of the steps involve the conversion of number words to numeric form, e.g., seven to 7, to standardize text. To do this, we use the word2number module. The other step is to remove numbers.

| def number\_word\_to\_numeric(text):  text = text.split()  output = ""  for i in text:  try:  res = w2n.word\_to\_num(i)  except:  res = i  output += str(res) + " "  output = output.rstrip()  return output  def remove\_number(text):  res = " ".join([i for i in text if not i.isdigit()])  return res |
| --- |

**Remove stopwords**

Stopwords are very common words. Words like “we” and “are” probably do not help at all in NLP tasks such as sentiment analysis or text classifications. Hence, we can remove stopwords to save computing time and efforts in processing large volumes of text.

| def remove\_stop\_words(text):  stop = open("stopwords.txt")  stop\_words = []  for x in stop:  stop\_words.append(x)  stop\_words = list(set(stop\_words))  word\_tokens = word\_tokenize(text)  filtered\_sentence = []  for w in word\_tokens:  if w not in stop\_words:  filtered\_sentence.append(w)  filtered\_sentence = " ".join(filtered\_sentence)  return filtered\_sentence |
| --- |

**Lemmatization:**

Like stemming, lemmatization also converts a word to its root form. The only difference is that lemmatization ensures that the root word belongs to the language. We will get valid words if we use lemmatization. In NLTK, we use the WordNetLemmatizer to get the lemmas of words. We also need to provide a context for the lemmatization.

| def lemmatization(text):  nlp = stanza.Pipeline(lang="en", processors="tokenize, pos, lemma")  doc = nlp(text)  parsed\_text = {"word": [], "lemma": []}  for sent in doc.sentences:  for wrd in sent.words:  parsed\_text["word"].append(wrd.text)  parsed\_text["lemma"].append(wrd.lemma)  return pd.DataFrame(parsed\_text) |
| --- |

| INFO:stanza:Checking for updates to resources.json in case models have been updated. Note: this behavior can be turned off with download\_method=None or download\_method=DownloadMethod.REUSE\_RESOURCES  Downloading https://raw.githubusercontent.com/stanfordnlp/stanza-resources/main/resources\_1.4.1.json:  193k/? [00:00<00:00, 5.55MB/s]  INFO:stanza:Loading these models for language: en (English):  ========================  | Processor | Package |  ------------------------  | tokenize | combined |  | pos | combined |  | lemma | combined |  ========================  INFO:stanza:Use device: cpu  INFO:stanza:Loading: tokenize  INFO:stanza:Loading: pos  INFO:stanza:Loading: lemma  INFO:stanza:Done loading processors!  word  lemma  0  Inspirational  inspirational  1  Quotes  quote  2  -  -  3  Motivational  Motivational  4  Quotes  quote  ...  ...  ...  915  token  token  916  ;  ;  917  }  }  918  No  no  919  thanks  thanks  920 rows × 2 columns |
| --- |

**Tokenization**

In natural language processing, tokenization is the text preprocessing task of breaking up text into smaller components of text (known as tokens).

| def tokenization(text):  tokenized\_text = tokenize(text, "en")  return tokenized\_text |
| --- |

| [‘This’, ‘is’, ‘a’, ‘sample’, ‘sentence’, ‘for’, ‘tokenization’] |
| --- |

**Stemming:**

Stemming is the process of getting the root form of a word. Stem or root is the part to which inflectional affixes (-ed, -ize, -de, -s, etc.) are added. The stem of a word is created by removing the prefix or suffix of a word. So, stemming a word may not result in actual words.

| def stemming(text):  ps = PorterStemmer()  text = text.split()  output = ""  for i in text:  res = ps.stem(i)  output += str(res) + " "  return output |
| --- |



**Normalization**

Usually, text normalisation starts with tokenizing the text, which our longer corpus is now to be split into chunks of words, which the tokenizer class from NLTK can do. Post that, we need to lower case each word of our corpus, converting numbers to the words and last with contraction replacement.

| def text\_normalization(text):  factory = IndicNormalizerFactory()  normalizer = factory.get\_normalizer("en")  text = normalizer.normalize(text)  return text |
| --- |

**Part-of-Speech Tagging**

In natural language processing, part-of-speech tagging is the process of assigning a part of speech to every word in a string. Using the part of speech can improve the results of lemmatization.

| def pos\_tagging(text):  nlp = stanza.Pipeline(lang="en", processors="tokenize, pos, lemma")  doc = nlp(text)  parsed\_text = {"word": [], "upos": [], "xpos": []}  for sent in doc.sentences:  for wrd in sent.words:  parsed\_text["word"].append(wrd.text)  parsed\_text["upos"].append(wrd.upos)  parsed\_text["xpos"].append(wrd.xpos)  return pd.DataFrame(parsed\_text) |
| --- |

[ ('My', PRP$'), ('name', 'NN'), ('is', 'VBZ'), (‘Junaid’, 'NNP'), (‘.’, ‘.’ ) ]

**Conclusion:**

We have successfully performed text pre-processing tasks on a given piece of English text.