atomcamp

NATURAL LANGUAGE PROCESSING (NLP) MANUAL FOR STUDENTS

Table of Contents

Introduction to NLP :

Natural Language Processing (NLP) is a field at the intersection of computer science, artificial intelligence, and linguistics. It involves the interaction between computers and human language, aiming to read, decipher, understand, and make sense of human languages in a valuable way. This manual focuses on the initial steps in the NLP pipeline: text preprocessing and feature extraction.

Overview of Text Preprocessing

Text preprocessing is a critical step in the NLP pipeline. It transforms raw text into a format that can be effectively used by machine learning models. The main goals are to clean the text, standardize it,and reduce its complexity while preserving its meaning.

Text Preprocessing Steps:

1. Segmentation:

Segmentation breaks a large piece of text into linguistically meaningful sentence units. This is obvious in languages like English, where the end of a sentence is marked by a period, but it is still not trivial. A period can be used to mark an abbreviation as well as to terminate a sentence, and in this case, the period should be part of the abbreviation token itself. The process becomes even more complex in languages, such as ancient Chinese, that don't have a delimiter that marks the end of a sentence.

2. Tokenization

.Tokenization is the process of splitting text into smaller units called tokens (words, phrases, symbols, or other meaningful elements).This allows the computer to work on your text token by token rather than working on the entire text in the following stages.There are various tokenisation algorithms such as the whitespace tokenisation, Regular expression tokenisation (also called Regex), and the statistical tokenisation

```
import nltk
from nltk.tokenize import word_tokenize


def clean_tokenization(review_text):
    return word_tokenize(review_text)

dataset['CleanReview'] = dataset['CleanReview'].apply(clean_tokenization)
```

3. Lowercasing

Lowercasing ALL your text data, although commonly overlooked, is one of the simplest and most effective form of text preprocessing. It is applicable to most text mining and NLP problems and can help in cases where your dataset is not very large and significantly helps with consistency of expected output. While lowercasing should be standard practice, in rare cases preserving the capitalization is important. For example, in predicting the programming language of a source code file. The word System in Java is quite different from system in python. Lowercasing the two makes them identical, causing the classifier to lose important predictive features. While lowercasing is generally helpful, it may not be applicable for all tasks.

```
def clean_lowercase(review_text):
    return str(review_text).lower()
dataset['CleanReview'] = dataset['CleanReview'].apply(clean_lowercase)
```

4. Removing all irrelevant characters (Numbers and Punctuation)

```
def clean_non_alphanumeric(review_text):
    return re.sub('[^a-zA-Z]',' ',review_text)

dataset['CleanReview'] = dataset['CleanReview'].apply(clean_non_alphanumeric)
```

5. URLs removal

```
# Importing the library 're' to remove non url, Numbers and punctuation
import re

def clean_url(review_text):
    return re.sub(r'http\S+', '', review_text)

dataset['CleanReview'] = dataset['Reviews'].apply(clean_url)
```

6. Stop word removal

```
# importing the libraries required to remove the stopwords and punctuation
from nltk.corpus import stopwords
# let's look at the stopwords in english
stopwords.words('english')
```

```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
 'his',
```

```
stop_words = set(stopwords.words('english'))
def clean_stopwords(token):
    return [item for item in token if item not in stop_words]

dataset['CleanReview'] = dataset['CleanReview'].apply(clean_stopwords)
```

7. Accent removal

This process is about removing language specific character symbols from text. Some characters are written with specific accents or symbols to either imply a different pronunciation or to signify that words containing such accented texts have a different meaning. An example of this would be the difference in both meaning and pronunciation between the words résumé and resume. The former refers to a document that highlights your professional skills and achievements, whereas the latter means ‘to take on something again, or to continue a previous task or action’.

8. Stemming and Lemmatization

Stemming and lemmatization are both text normalization techniques used in Natural Language Processing (NLP) to reduce words to their base or root forms. While they share the goal of simplifying words, they operate differently in terms of the linguistic knowledge they apply.

Stemming: : Reducing to Root Forms

Stemming involves cutting off prefixes or suffixes of words to obtain their root or base form, known as the stem. The purpose is to treat words with similar meanings as if they were the same. Stemming is a rule-based method that doesn't always result in a valid word, but it's computationally less intensive.

```
from  nltk.stem import PorterStemmer

stemmer = PorterStemmer()
def clean_stem(token):
    return [stemmer.stem(i) for i in token]

dataset['CleanReview'] = dataset['CleanReview'].apply(clean_stem)
```

	original_word	stemmed_words
0	connect	connect
1	connected	connect
2	connection	connect
3	connections	connect
4	connects	connect

	original_word	stemmed_word
0	trouble	troubl
1	troubled	troubl
2	troubles	troubl
3	troublemsome	troublemsom

Lemmatization: Transforming to Dictionary Form

Lemmatization, on the other hand, involves reducing words to their base or dictionary forms, known as lemmas. It takes into account the context of the word in a sentence and applies morphological analysis. Lemmatization results in valid words and is more linguistically informed compared to stemming.

```
from nltk.stem import WordNetLemmatizer
lemma=WordNetLemmatizer()

def clean_lemmatization(token):
    return [lemma.lemmatize(word=w,pos='v') for w in token]

dataset['CleanReview'] = dataset['CleanReview'].apply(clean_lemmatization)
```

	original_word	lemmatized_word
0	trouble	trouble
1	troubling	trouble
2	troubled	trouble
3	troubles	trouble

	original_word	lemmatized_word
0	goose	goose
1	geese	goose

When to Use Stemming vs. Lemmatization:

Stemming:
- Pros: Simple and computationally less expensive.
- Cons: May not always result in valid words.

Lemmatization:
- Pros: Produces valid words; considers linguistic context.
- Cons: More computationally intensive than stemming.

9. Part-of-Speech Tagging:

Part-of-speech tagging (POS tagging) is a natural language processing task where the goal is to assign a grammatical category (such as noun, verb, adjective, etc.) to each word in a given text. This provides a deeper understanding of the structure and function of each word in a sentence. The Penn Treebank POS Tag Set is a widely used standard for representing these part-of-speech tags in English text.