Article on the subject = Medium Post

Comprehensive Text Preprocessing NLP (Natural Language Processing)

Text preprocessing plays a crucial role in Natural Language Processing (NLP) by addressing words that may be difficult for AI models to understand or process. By applying techniques such as cleaning, standardization, and transformation, we ensure that the text is in a format that the models can comprehend. This preprocessing step aims to enhance the model's ability to produce more accurate and meaningful results, as it removes obstacles and facilitates better analysis of the text data.

Some techniques for NLP:

1 - Text Extraction and Cleanup:

- Removing HTML tags
- Removing HTML tags
- Removing URLs
- Removing all irrelevant characters
- Removing punctuation
- Removing duplicate text
- Removing numbers or replace
- Managing whitespace

2 - Spelling and Grammar Correction:

- Correcting Spelling And Grammar Errors

3 - Text Formatting:

- Lowercasing
- StopWords Removal

4 - Sentence and Word Tokenization:

- Sentence Tokenization
- Word Tokenization

5 - Word Processing:

- Stemming
- Lemmatization

6 - Special NLP Processes:

- Tok2vec
- Tagger
- Parser
- Attribute_ruler
- Ner

1- Text Extraction and Cleanup:

It is the process of extracting relevant text from various sources and cleaning it by removing unnecessary elements.

a) Removing HTML tags: It is the process of eliminating HTML tags from text data.

```
import pandas as pd
from bs4 import BeautifulSoup

# Function to remove HTML tags from a given text
def remove_html_tags(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

# Sample dataframe
data = {'Text': ['<p>This is some <b>HTML</b> text.</p>',
                 'Another <a href="example.com">example</a>']}
df = pd.DataFrame(data)


# Apply the function to the 'Text' column of the dataframe
df['Text'] = df['Text'].apply(remove_html_tags)

df.head()
```

```
                      Text
0  This is some HTML text.
1          Another example
```

b) Removing URLs: It refers to the process of eliminating URLs (web addresses) from text data

```
import pandas as pd
import re

# Function to remove URLs from a given text
def remove_urls(text):
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    return url_pattern.sub('', text)

# Sample dataframe
data = {'Text': ['Visit our website at www.example.com', 'Check out
```

```
this link: https://example.com']}
df = pd.DataFrame(data)


# Apply the function to the 'Text' column of the dataframe
df['Text'] = df['Text'].apply(remove_urls)

df.head()
```

```
                    Text
0  Visit our website at
1  Check out this link:
```

c) Removing All Irrelevant Characters: This step involves removing any characters from the text that are deemed irrelevant or unnecessary for further analysis. This may include special symbols, emojis, or any non-alphanumeric characters that do not contribute to the overall meaning of the text.

Regex removes any non-alphanumeric characters from the text, keeping only letters (both lowercase and uppercase) and digits. Whitespace characters are preserved and not removed.

```
import pandas as pd
import re

# Function to remove irrelevant characters
def remove_irrelevant_chars(text):
    cleaned_text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    return cleaned_text

# Sample dataframe
data = {'Text': ['This! is? some text... with irrelevant
characters!']}
df = pd.DataFrame(data)

# Apply the function to the 'Text' column of the dataframe
df['Text'] = df['Text'].apply(remove_irrelevant_chars)

df.head()
```

```
                                          Text
0  This is some text with irrelevant characters
```

d) Removing punctuation This step focuses on eliminating punctuation marks such as periods, commas, question marks, and exclamation marks from the text.

```
import pandas as pd
import string

# Function to remove punctuation
def remove_punctuation(text):
```

```
    cleaned_text = text.translate(str.maketrans("", "",
string.punctuation))
    return cleaned_text

# Sample dataframe
data = {'Text': ['This, is "some" text... with, punctuation!']}
df = pd.DataFrame(data)

# Apply the function to the 'Text' column of the dataframe
df['Text'] = df['Text'].apply(remove_punctuation)

df.head()
```

```
                                   Text
0  This is some text with punctuation
```

e) Removing Duplicate Text: Duplicate text removal aims to identify and eliminate repeated instances of text within a given dataset. By removing duplicates, we can avoid biased analysis or redundant information in the subsequent processing steps, ensuring cleaner and more accurate results.

```
import pandas as pd

# Remove duplicate text
def remove_duplicate_text(df):
    df = df.drop_duplicates()
    return df

# Sample dataframe
data = {'Text': ['This is some duplicate text.','This is some
duplicate text.','This is some duplicate text.']}
df = pd.DataFrame(data)

df = remove_duplicate_text(df)

df.head()
```

```
                            Text
0  This is some duplicate text.
```

f) Removing Numbers or Replace With Text: In this step, numbers are either removed or replaced with a designated placeholder in the text. Removing numbers can be beneficial when numeric values are not relevant to the analysis. Alternatively, replacing numbers with a placeholder (e.g., 'NUM') allows retaining the presence of numeric information while abstracting the specific values.

— Removing Numbers or Deleting: You can completely remove or delete numbers from the text. This can be done using regular expressions to match and remove numeric patterns.

```
import re

text = "I have 5 apples and 3 oranges."
text_without_numbers = re.sub(r'\d+', '', text)
print(text_without_numbers)
# Output: "I have apples and oranges."
```

```
I have  apples and  oranges.
```

— **Converting Numbers to Words:** You can convert numbers to words to make numerical expressions more readable. This can be achieved using libraries like num2words.

```
!pip install num2words
```

```
Collecting num2words
  Downloading num2words-0.5.12-py3-none-any.whl (125 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/125.2 kB ? eta -:--:--
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 122.9/125.2 kB 4.1 MB/s eta
0:00:01 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 125.2/125.2 kB 3.1
MB/s eta 0:00:00
 num2words)
  Downloading docopt-0.6.2.tar.gz (25 kB)
  Preparing metadata (setup.py) ... e=docopt-0.6.2-py2.py3-none-
any.whl size=13707
sha256=de563b9cba0a62a79fa910ba6e14f2c9562c0f594f14f83c5063c52e4a72523
c
  Stored in directory:
/root/.cache/pip/wheels/fc/ab/d4/5da2067ac95b36618c629a5f93f8094257005
06f72c9732fac
Successfully built docopt
Installing collected packages: docopt, num2words
Successfully installed docopt-0.6.2 num2words-0.5.12
```

```
from num2words import num2words

text = "I have 5 apples and 3 oranges."
words = []
for word in text.split():
    if word.isdigit():
        words.append(num2words(int(word)))
    else:
        words.append(word)
text_with_words = ' '.join(words)
print(text_with_words)
# Output: "I have five apples and three oranges."
```

```
I have five apples and three oranges.
```

— Replacing Numbers with a Symbol or Placeholder: You can replace numbers with a specific symbol or placeholder to indicate the presence of numerical values. This can be useful in cases where you want to retain the existence of numbers.

import re

text = "I have 5 apples and 3 oranges." text_with_symbols = re.sub(r'\d+', '#NUMBER#', text) print(text_with_symbols)

Output: "I have #NUMBER# apples and #NUMBER# oranges."

g) Managing Whitespace Managing Whitespace refers to the process of handling spaces, tabs, and other whitespace characters in text data. It involves tasks such as removing excessive whitespace, normalizing whitespace, or preserving specific whitespace patterns.

```
import pandas as pd

# Create an example DataFrame with whitespace in text
df = pd.DataFrame({"text": ["   Hello     World  "]})

# Remove excessive whitespace
df["text"] = df["text"].str.strip()

# Normalize whitespace
df["text"] = df["text"].str.replace(r'\s+', ' ', regex=True)

# Print the results
df.head()
```

```
          text
0  Hello World
```

2 — Spelling and Grammar Correction:

Spelling and Grammar Correction automatically identifies and corrects errors in writing, such as spelling mistakes, punctuation errors, and grammar issues, improving the accuracy and readability of the text.

```
!pip install pyspellchecker
```

```
Collecting pyspellchecker
  Downloading pyspellchecker-0.7.2-py3-none-any.whl (3.4 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.4/3.4 MB 30.4 MB/s eta
0:00:00
```

```
from spellchecker import SpellChecker

# Create an instance of the SpellChecker
spell = SpellChecker()

# Text with spelling errors
text = "Ths sentnce has speling errrs."

# Split the text into individual words
words = text.split()

# Check each word for spelling errors and correct them
corrected_words = []
for word in words:
    corrected_word = spell.correction(word)
    corrected_words.append(corrected_word)

# Join the corrected words back into a sentence
corrected_text = ' '.join(corrected_words)

# Print the corrected text
print(corrected_text)
```

```
the sentence has spelling errors
```

3 — Text Formatting:

Text formatting refers to the process of manipulating the structure or appearance of a text to achieve a desired format or style. It involves applying specific transformations or modifications to the text to meet certain requirements or conventions.

a) Lowercasing Lowercasing is a text formatting technique that involves converting all characters in a text to lowercase.Because even if it is the same word, the presence of uppercase or lowercase letters can confuse the model and lead it to perceive them as different words. (Example : House, hoUse)

Lowercasing is often used to ensure consistency, remove case sensitivity, or facilitate text processing tasks such as text matching, comparison, or analysis.

```
text = "This is an EXAMPLE text."

lowercased_text = text.lower()

print(lowercased_text) # this is an example text.
```

```
this is an example text.
```

b) StopWords Removal StopWords Removal is a process of eliminating common words, known as stopwords, from a text. These stopwords are usually words that do not carry significant

meaning or contribute to the overall context of the text. By removing stopwords, we can focus on the more meaningful and informative words in the text for analysis or processing tasks.

```
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
nltk.download('punkt')
nltk.download('stopwords')

# Create an example DataFrame
df = pd.DataFrame({"text": ["This is an example sentence.", "Another
example sentence."]})

# Define the stopwords
stopwords_list = set(stopwords.words("english"))

# Apply stopwords removal to each text in the DataFrame
df["text_without_stopwords"] = df["text"].apply(lambda x: '
'.join([word for word in word_tokenize(x) if word.lower() not in
stopwords_list]))

# Print the results
df.head()
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

	text	text_without_stopwords
0	This is an example sentence.	example sentence .
1	Another example sentence.	Another example sentence .

4 — Sentence and Word Tokenization:

These tokenization techniques are essential preprocessing steps in natural language processing tasks, enabling effective analysis, modeling, and understanding of textual data.

a) Sentence Tokenization Sentence tokenization is the process of breaking down a text or a document into individual sentences. It involves splitting the given text based on specific punctuation marks or patterns that indicate the end of a sentence, such as periods, question marks, and exclamation points. The goal of sentence tokenization is to segment the text into meaningful units to facilitate further analysis or processing.

```
import pandas as pd
import nltk
nltk.download('punkt')
```

```
# Sample text
text = "Hello! How are you? I hope you're doing well. Have a great
day!"

# Tokenize sentences
sentences = nltk.sent_tokenize(text)

# Create a dataframe
df = pd.DataFrame({'Sentences': sentences})

# Print the dataframe
df.head()
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
                   Sentences
0                     Hello!
1                How are you?
2  I hope you're doing well.
3           Have a great day!
```

b) Word Tokenization Word tokenization is the process of breaking down a sentence or a text document into individual words or tokens. It involves splitting the given text based on specific delimiters such as spaces or punctuation marks. The goal of word tokenization is to segment the text into meaningful units, allowing for further analysis, processing, or language-related tasks like counting word frequencies or building language models.

```
import pandas as pd
import nltk
nltk.download('punkt')

# Sample text
text = "I love to eat pizza."

# Tokenize words
words = nltk.word_tokenize(text)

# Create a dataframe
df = pd.DataFrame({'Words': words})

# Print the dataframe
df.head()
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
   Words
0      I
```

```
1   love
2     to
3    eat
4  pizza
```

5 — Word Processing:

Steamming and lemmatization are two different methods used in the preprocessing of text data in the field of natural language processing (NLP). Both are used to derive word roots or normalize words, but they offer different approaches and results.

a) Stemming Stemming attempts to find the root of a word without considering its current meaning, which means that the derived root may not have any connection to the current meaning of the word.

```
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
import nltk

nltk.download('punkt')

# Create an example sentence for text analysis
text = "computers information"

# Tokenize the words
words = word_tokenize(text)

# Create an instance of PorterStemmer class
stemmer = PorterStemmer()

# Apply stemming to each word
stemmed_words = [stemmer.stem(word) for word in words]

# Print the stemmed words
print("Stemmed Words:", stemmed_words)
```

```
Stemmed Words: ['comput', 'inform']
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

b) Lemmatization Lemmatization, on the other hand, tries to find the root of a word by considering its current meaning, which means that the derived root should be connected to the current meaning of the word.

```
import spacy

# Load the Spacy model
```

```
nlp = spacy.load("en_core_web_sm")

# Create an example sentence for text analysis
text = "I am running in the park with my dogs"

# Process the document and extract the lemma for each word
doc = nlp(text)

# Get the lemma for each word
lemmatized_words = [token.lemma_ for token in doc]

# Print the lemmatized words
print("Lemmatized Words:", lemmatized_words)
```

```
Lemmatized Words: ['I', 'be', 'run', 'in', 'the', 'park', 'with',
'my', 'dog']
```

6 — Special NLP Processes:

They employ specialized algorithms and models to extract insights from text data, enabling targeted analysis and understanding of language patterns beyond basic NLP tasks.

a) Tok2vec : Tok2vec is a technique to represent words or text phrases as vectors. These vectors are trained to capture linguistic features and contextual information within the text.Artificial intelligence models cannot directly work with text data; we need to convert them into numerical values.

```
import spacy

nlp = spacy.load("en_core_web_sm")

# Create a tok2vec vector
text = "I will review this book."
doc = nlp(text)

# Get the tok2vec vector for the first token
tok2vec_vector = doc[0].vector

# Print the tok2vec vector
print(tok2vec_vector)
# Output: [-0.8694229, 0.8168279, -0.09419681, ..., -0.4794904, -
0.9357786, 0.3212685...]
```

```
[-0.8475625  -0.1087786   0.723475   -0.05477826 -0.16502021 -
0.51059765
  1.53197    -0.20134415  0.35173658 -0.31501323  1.9730195
1.4457998
 -0.8263972   0.50297165 -1.2208867  -0.79898393  0.46120888
```

```
0.00700635
 -0.40814275 -0.69746506 -1.5664808   0.566766    0.61037576 -
1.2097255
 -0.8960694  -0.3761478  -0.07286161  0.40165257 -0.23793092
0.4727332
  0.2570689  -0.2880685   0.38371062  0.3358915  -0.30448458
1.0774269
 -0.8815656  -0.13856497 -0.6470118   1.6094114  -1.3961741
0.10297833
  0.4162639   0.6213889  -0.96504104 -0.89392287  0.43724385
2.9410152
  0.2404128  -0.02080229 -1.3451581  -0.8809747   1.4717823  -
1.5098917
 -0.08040679 -0.7624888   0.62514013 -1.1576056  -0.11995807  0.723379
  1.1804109   1.421203   -0.26907903 -0.54537135 -0.1878475
0.9960146
 -0.7397172  -0.2469005  -0.9839954  -1.1825488  -0.8943811
0.68959475
  0.6131857  -0.8966919  -0.8691123  -0.14561123 -0.14888728
0.11907951
  0.24081329 -0.08970817 -0.5955648  -0.8678413   0.4366747  -
0.0689545
  1.0917003   0.4287699  -0.14495087  0.11007045  1.3244289
0.38318714
  1.1105771  -0.6539103   0.8245801  -0.66518074 -0.7729554
0.92884946]
```

b) Tagger : A Tagger is a trained model that identifies the linguistic features(verb,noun) or labels of words or word groups in a text.

```python
import spacy
import pandas as pd

# Load the Spacy model
nlp = spacy.load("en_core_web_sm")

# Create an example dataframe
df = pd.DataFrame({"text": ["This is an example sentence."]})

# Apply Tagger to determine the tags and retrieve the tag values and
their numerical representations
tags = []
tag_values = []
tag_numerical_values = []
for text in df["text"]:
    doc = nlp(text)
    sentence_tags = [token.tag_ for token in doc]
    tags.append(sentence_tags)
    sentence_tag_values = [token.pos_ for token in doc]
```

```
    tag_values.append(sentence_tag_values)
    sentence_tag_numerical_values = [token.pos for token in doc]
    tag_numerical_values.append(sentence_tag_numerical_values)

# Add the generated tags, tag values, and numerical values to the
dataframe
df["tags"] = tags
df["tag_values"] = tag_values
df["tag_numerical_values"] = tag_numerical_values

# Print the results
df.head()
```

```
                           text                         tags  \
0  This is an example sentence.  [DT, VBZ, DT, NN, NN, .]

                            tag_values      tag_numerical_values
0  [PRON, AUX, DET, NOUN, NOUN, PUNCT]  [95, 87, 90, 92, 92, 97]
```

c) Parser: Parser focuses on syntactic analysis. It aims to analyze the grammatical structure of sentences and determine the relationships between words. The parser identifies the subject, verb, object, and other syntactic components, creating a parse tree or dependency tree that represents the hierarchical structure of the sentence.

```
import spacy
import pandas as pd

# Load the Spacy model
nlp = spacy.load("en_core_web_sm")

# Create an example dataframe
df = pd.DataFrame({"text": ["Jhon hit the ball"]})

# Apply Parser to get the dependency relationships
dependencies = []
for text in df["text"]:
    doc = nlp(text)
    sentence_dependencies = [(token.text, token.dep_, token.head.text)
for token in doc]
    dependencies.append(sentence_dependencies)

# Add the generated dependencies to the dataframe
df["dependencies"] = dependencies

# Print the results
df.head()
```

```
                text
dependencies
```

```
0  Jhon hit the ball  [(Jhon, nsubj, hit), (hit, ROOT, hit), (the,
d...
```

d) Attribute_ruler : AttributeRuler is a component in SpaCy that allows you to add custom attributes to tokens, such as binary flags or linguistic annotations. It provides a way to define rules based on patterns or linguistic criteria, which can then be applied to annotate tokens with the desired attributes. The AttributeRuler is useful for creating custom features or enriching the token representation with additional information.

e) NER (named entity recognition) "Ner" (named entity recognition) is a natural language processing (NLP) technique used to recognize named entities (e.g., persons, locations, organizations) in text. It involves identifying specific words or phrases in the text and assigning relevant labels to them based on their entity type.

Without NER, it is more difficult to identify meaningful entities in texts and obtain information about them.

```
import spacy
import pandas as pd

# Load the Spacy model
nlp = spacy.load("en_core_web_sm")

# Create an example dataframe
df = pd.DataFrame({"text": ["John Smith is a software engineer.",
"Microsoft is headquartered in Redmond."]})

# Apply NER to extract named entities
entities = []
for text in df["text"]:
    doc = nlp(text)
    sentence_entities = [(entity.text, entity.label_) for entity in
doc.ents]
    entities.append(sentence_entities)

# Add the extracted entities to the dataframe
df["entities"] = entities

# Print the results
df.head()
```

```
                                    text
entities
0      John Smith is a software engineer.              [(John Smith,
PERSON)]
1  Microsoft is headquartered in Redmond.  [(Microsoft, ORG),
(Redmond, GPE)]
```

Thank you for listening, and we have reached the end of today's topic. I hope you have enjoyed it. If you have anything you would like to add to the discussion, please feel free to leave a comment.