International Institute of Information Technology
Bangalore

AI 829

NATURAL LANGUAGE PROCESSING

Mandate 2

Hate and Offensive Speech Detection in Hindi and Marathi Social Media Texts

MT2023158	Manasi Purkar
MT2023051	Sanket Patil

<u>Index</u>

Contents

Introduction

In this mandate, we focused on lexical processing of English, Hindi and Marathi text data.
We utilized the HASOC 2021, 2020, 2019 dataset for english, HASOC 2021 dataset for Hindi and a combination of HASOC and MOLD datasets for Marathi. Our processing steps included removing punctuation marks and special characters, hashtags and URLs, which often clutter text data. We also tackled the challenge of handling emojis, a common element in online communication. Specifically to Indian languages, we removed stop words (common words with little value for analysis) and converting all text to lowercase to maintain consistency. Additionally, we implemented tokenization, breaking text into individual words or tokens.This thorough lexical processing aims to clean and prepare the data for further analysis effectively.

Lexical Processing is the process of identifying and analyzing the structure of words and phrases. used to convert an array of log data into a uniform structure.It can help to improve the accuracy of the results, as well as the efficiency of the process. Additionally, it can allow us to work with a larger variety of data.

Libraries Used

- pandas
- re
- numpy
- stopwordsiso
- string
- emoji
- sklearn
- BertTokenizer
- fasttext
- Nltk
- WordNetLemmatizer

Steps In Lexical Processing:

- **Removing unwanted columns:**
  Removing unwanted columns from dataset which are not required for our task.

- **Label Encoding:**
  Label encoding is a method to convert categorical labels into numerical values, assigning a unique integer to each category. It facilitates the use of categorical data in machine learning algorithms that require numerical input.

- **Removing unwanted patterns:**
  Removing specific patterns, special characters, URLs, numbers, and extra spaces using regular expressions. Regular expressions are the series/sequence of character(s) which can replace a set of patterns in a text dataset.

- **Lowering words:**
  The text is converted into the same case.

- **Handling Emojis**:
  In sentiment analysis, emojis express an emotion. we may lose valuable information if we remove the emojis. In this case, a better approach is to convert emoji to word format so that it preserves the emoji information.

- **Removing Punctuations:**
  In this step, all the punctuations from the text are removed. string library of Python contains some pre-defined list of punctuations such as '!”#$%&'()*+,-./:;?@[\]^_`{|}~'

- **Stopword Removal:**
  Stop words are a set of commonly used words in a language. Examples of stop words in Hindi language are "मैं", "मुझको", "मेरा", "हमने", "हमारा", "अपना", "हम", "आप", "आपका", "तुम्हारा", "इसे" and etc. The idea behind using stop words is that, by removing low information words from text, we can focus on the important words instead.
  In marathi and hindi done using **stopwordsiso** library and in english using **nltk**.

- **Expanding Contractions:**
  Expanding contractions involves transforming contracted words (e.g., "don't" to "do not") in text to their full forms. This process aids in standardizing and understanding language, improving natural language processing tasks by ensuring consistency and clarity in textual data. We did this in English dataset preprocessing.

- **Tokenization:**

  Tokenization refers to dividing the text into a sequence of words or sentences.
  A tokenizer breaks unstructured data and natural language text into chunks of information that can be considered as discrete elements. Tokenization can separate sentences, words, characters, or subwords. When we split the text into sentences, we call it sentence tokenization. For words, we call it word tokenization. Here we did word tokenization.
  Tokenization done using **BertTokenizer**.

- **Lemmatization:**

  Text pre-processing technique used to break a word down to its root meaning to identify similarities. For example, a lemmatization algorithm would reduce the word better to its root word, or lemme, good. We are not performing lemmatization in Marathi and Hindi to prevent the occurrence of slang words.
  In english lemmatization done using **WordNetLemmatizer**.

- **Sentence embeddings:**

  The **FastText** library is used to download a pre-trained FastText model, and the learned embeddings are applied to obtain sentence vectors for the text data. These embeddings capture semantic information about sentence.

Preprocessing Text Data in Hindi Language

Removing unwanted patterns:

```
import re
# remove @USERS
df['text'] = df['text'].str.replace(r'@\w+\s*', '', regex=True)
# More than one spaces
df['text'] = df['text'].str.replace(r'\s+', ' ', regex=True).str.strip()
# Removing Hashtags
df['text'] = df['text'].str.replace(r'#\w+', '', regex=True)
# Remove URLs
df['text'] = df['text'].str.replace(r'https?://\S+|www\.\S+', '', regex=True)
```

Lowering words:

```
df["text"] = df['text'].str.lower()
```

Label Encoding:

```
from sklearn.preprocessing import LabelEncoder
# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Fit and transform the 'task_1' column
df['task_1'] = label_encoder.fit_transform(df['task_1'])
```

Handling Emojis:

```
import emoji

def process_text_with_emojis(text):
    text_without_emojis = emoji.demojize(text)
    return text_without_emojis

df["text"] = df["text"].apply(process_text_with_emojis)
```

Removing Punctuations:

```
import string
df['text'] = df['text'].str.replace(f'[{string.punctuation}]', '', regex=True)
```

Stopword Removal:

```
!pip install stopwordsiso
```

```python
import stopwordsiso
stopwordsiso.has_lang("hi")  # check if there is a stopwords for hindi language
```

```python
from stopwordsiso import stopwords

# Load Hindi stopwords
marathi_stopwords = stopwords("hi")

def remove_stopwords(text):
    words = text.split()
    filtered_words = [word for word in words if word not in marathi_stopwords]
    return ' '.join(filtered_words)

df['text'] = df['text'].apply(remove_stopwords)
```

Tokenization:

```
!pip install transformers
```

```python
from transformers import BertTokenizer

# Load mBERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-multilingual-cased')

def tokenize_long_text(text, max_length=512, stride=100):
    tokens = []
    for i in range(0, len(text), stride):
        chunk = text[i:i+max_length]
        chunk_tokens = tokenizer.encode(chunk, add_special_tokens=True)
        tokens.extend(chunk_tokens)
    return tokens

# Tokenize the 'text' column with sliding window
df['tokenized_text'] = df['text'].apply(lambda x: tokenize_long_text(x, max_length=512, stride=100))

# Convert token IDs to tokens
df['tokenized_text_tokens'] = df['tokenized_text'].apply(lambda x: tokenizer.convert_ids_to_tokens(x))
```

FastText embedding:

```python
import os
import fasttext.util

# Specify the path where you want to save the model
model_path = './'

# Download the pre-trained FastText model for Hindi
fasttext.util.download_model('hi', if_exists='ignore')

# Move the downloaded model to the specified path
os.rename('cc.hi.300.bin', model_path)

# Load the pre-trained model
model = fasttext.load_model(model_path)
```

```python
# Function to get the sentence vector using FastText
def get_sentence_vector(text):
    return model.get_sentence_vector(text)

# Apply the function to the 'tweet' column to get sentence vectors
df['text_vector'] = df['text'].apply(get_sentence_vector)
```

Preprocessing Text Data in Marathi Language

Remove unwanted columns:

```
# Specify the columns to remove
columns_to_remove = ['subtask_b', 'subtask_c']

# Remove columns that are in the specified list
df.drop(columns=columns_to_remove, inplace=True)
```

Label Encoding:

```
from sklearn.preprocessing import LabelEncoder
# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Fit and transform the 'task_1' column
df2['task_1'] = label_encoder.fit_transform(df2['task_1'])
print(df2.head())  #Hate means 0
```

Removing unwanted patterns in text:

```
import re
#  remove "@USER"
df['tweet'] = df['tweet'].str.replace(r'@user', '', flags=re.IGNORECASE)

pattern = r'(\d+)\s'
# remove above pattern which comes with date
df['tweet'] = df['tweet'].apply(lambda x: re.sub(pattern, '', x))

# remove special characters, numbers
df['tweet'] = df['tweet'].str.replace('[\d@?()~|'•©-]', '', regex=True)

# remove more than one spaces
df['tweet'] = df['tweet'].apply(lambda x: re.sub(r'\s+', ' ', x))
```

Handling Emojis:

```
import emoji

# Function to demojize text
def demojize_text(text):
    return emoji.demojize(text)

# Apply the function to the 'text' column
df2['text'] = df2['text'].apply(demojize_text)


print(df2['text'].head())
print(df2.loc[1320, 'text'])
```

Punctuation marks removal:

```python
import string
# Function to remove punctuations from Marathi text
def remove_punctuations(text):
    translator = str.maketrans('', '', string.punctuation)
    return text.translate(translator)

# Apply the function to the 'tweet' column
df['tweet'] = df['tweet'].apply(remove_punctuations)
```

Stopwords Removal:

```python
import stopwordsiso
stopwordsiso.has_lang("mr")  # check if there is a stopwords for Marathi language
from stopwordsiso import stopwords

# Load Marathi stopwords
marathi_stopwords = stopwords("mr")

# Function to remove stopwords
def remove_stopwords(text):
    words = text.split()
    filtered_words = [word for word in words if word not in marathi_stopwords]
    return ' '.join(filtered_words)

# Apply the remove_stopwords function to the 'tweet' column
df['without_stopwords'] = df['tweet'].apply(remove_stopwords)
```

Merging the HASOC and MOLD dataset for marathi after applying above processing on both of the datasets

```python
# Concatenate DataFrames along rows
concatenated_df = pd.concat([df, df2], ignore_index=True)
```

Tokenization:

```python
import pandas as pd
from transformers import BertTokenizer

# Load mBERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-multilingual-cased')

def tokenize_long_text(text, max_length=512, stride=100):
    tokens = []
    for i in range(0, len(text), stride):
        chunk = text[i:i+max_length]
        chunk_tokens = tokenizer.encode(chunk, add_special_tokens=True)
        tokens.extend(chunk_tokens)
    return tokens


# Tokenize the 'tweet' column with sliding window
concatenated_df['tokenized_tweet'] = concatenated_df['tweet'].apply(lambda x: tokenize_long_text(x, max_length=512, stride=100))

# Convert token IDs to tokens
concatenated_df['tokenized_tweet_tokens'] = concatenated_df['tokenized_tweet'].apply(lambda x: tokenizer.convert_ids_to_tokens(x))
```

FastText Embeddings:

```
import os
import fasttext.util

# Specify the path where you want to save the model
model_path = '/content/cc.mr.300.bin.gz.part'

# Download the pre-trained FastText model for Marathi
fasttext.util.download_model('mr', if_exists='ignore')

# Move the downloaded model to the specified path
os.rename('cc.mr.300.bin', model_path)

# Load the pre-trained model
model = fasttext.load_model(model_path)
```

```
# Function to get the sentence vector using FastText
def get_sentence_vector(text):
    return model.get_sentence_vector(text)

# Apply the function to the 'tweet' column to get sentence vectors
concatenated_df['tweet_vector'] = concatenated_df['tweet'].apply(get_sentence_vector)
```

In hindi and marathi dataset we didn’t performed lemmatization because lemmatization will alter hate words in marathi and hindi.

Preprocessing Text Data in English Language

Removing unwanted columns:

```
df = df.drop(['Unnamed: 0','_id','task_2'], axis=1)
df2= df2.drop(['tweet_id','task2','ID'], axis=1)
df3= df3.drop(['text_id','task_2', 'task_3'], axis=1)
```

Label Encoding:

```
from sklearn.preprocessing import LabelEncoder
# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Fit and transform the 'label' column
df['label'] = label_encoder.fit_transform(df['label'])
print(df.head())  #Hate means 0
```

Removing unwanted patterns:

```python
import re
import string


# Remove @usernames
df['text'] = df['text'].apply(lambda x: re.sub(r'@[^ ]+', '', x))

# Remove &amp
df['text'] = df['text'].apply(lambda x: re.sub(r'&amp', '', x))

# Remove URLs from the 'text' column
df['text'] = df['text'].apply(lambda x: re.sub(r'https?://\S+|www\.\S+', '', x))

#remove numbers
df['text'] = df['text'].apply(lambda x: re.sub(r'\d+', '', x))

#remove newline character
df['text'] = df['text'].apply(lambda x: re.sub(r'\n', '', x))
```

Expanding Contractions:

```python
!pip install contractions
import contractions

def expand_contractions(text):
    # Use the contractions library to expand contractions
    expanded_text = contractions.fix(text)
    return expanded_text
df['text'] = df['text'].apply(lambda x: expand_contractions(x))
```

Removing Punctuation marks:

```python
def remove_punc1(text):
    return text.translate(str.maketrans('','',string.punctuation.replace('#','')))

df['text']=df['text'].apply(remove_punc1)
```

```python
def remove_non_english(text):
    # Replace non-English characters with an empty string
    cleaned_text = re.sub(r'[^a-zA-Z\s#]', '', text)
    return cleaned_text

# Assuming df is your DataFrame with a 'text' column
df['text'] = df['text'].apply(remove_non_english)
```

Stopwords Removal:

```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize

def custom_tokenize(text):
    # Tokenize the text using the default word tokenizer
    tokens = word_tokenize(text)

    # Merge '#' and its following text into a single token
    merged_tokens = []
    i = 0
    while i < len(tokens):
        if tokens[i] == '#' and i + 1 < len(tokens):
            merged_tokens.append('#' + tokens[i + 1])
            i += 2
        else:
            merged_tokens.append(tokens[i])
            i += 1

    return merged_tokens
```

```
nltk.download('stopwords')
from nltk.corpus import stopwords
# Get the list of English stop words
stop_words = set(stopwords.words('english'))

# Function to remove stop words from text
def remove_stop_words(text):
    words = custom_tokenize(text)
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered_words)

# Apply the function to the 'review_text' column
df['text'] = df['text'].apply(remove_stop_words)
```

Removing repeated characters in word:

```
import nltk
import subprocess
from nltk.corpus import wordnet
class RepeatReplacer(object):
    def __init__(self):
        self.repeat_regexp = re.compile(r'(\w*)(\w)\2(\w*)')
        self.repl = r'\1\2\3'

    def replace(self, word):
        if wordnet.synsets(word):
            return word

        repl_word = self.repeat_regexp.sub(self.repl, word)

        if repl_word != word:
            return self.replace(repl_word)
        else:
            return repl_word
```

```python
# Instantiate the RepeatReplacer class
replacer = RepeatReplacer()

# Function to apply the RepeatReplacer on the 'text' column
def apply_replacer(text):
    words = text.split()
    replaced_words = [replacer.replace(word) for word in words]
    return ' '.join(replaced_words)

# Apply the function to the 'text' column
df['text'] = df['text'].apply(apply_replacer)
```

Lemmatization:

```python
import pandas as pd
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt')
nltk.download('wordnet')



# Function to perform lemmatization on English text
def perform_lemmatization(text):
    lemmatizer = WordNetLemmatizer()
    words = custom_tokenize(text)
    lemmatized_words = [lemmatizer.lemmatize(word) for word in words]
    lemmatized_text = ' '.join(lemmatized_words)
    return lemmatized_text

# Apply the function to the 'text' column
df['text'] = df['text'].apply(perform_lemmatization)
```

Tokenization:

```python
import pandas as pd
from transformers import BertTokenizer

# Load mBERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-multilingual-cased')

def tokenize_long_text(text, max_length=512, stride=100):
    tokens = []
    for i in range(0, len(text), stride):
        chunk = text[i:i+max_length]
        chunk_tokens = tokenizer.encode(chunk, add_special_tokens=True)
        tokens.extend(chunk_tokens)
    return tokens


# Tokenize the 'text' column with sliding window
df['tokenized_text'] = df['text'].apply(lambda x: tokenize_long_text(x, max_length=512, stride=100))
```

FastText Embedding:

```python
import os
import fasttext.util

# Specify the path where you want to save the model
model_path = '/content/cc.en.300.bin.gz.part'

# Download the pre-trained FastText model for English
fasttext.util.download_model('en', if_exists='ignore')

# Move the downloaded model to the specified path
os.rename('cc.en.300.bin', model_path)

# Load the pre-trained model
model = fasttext.load_model(model_path)
```

```python
# Function to get the sentence vector using FastText
def get_sentence_vector(text):
    return model.get_sentence_vector(text)

# Apply the function to the 'text' column to get sentence vectors
df['text_vector'] = df['text'].apply(get_sentence_vector)
```

In english dataset spelling correction is not performed because sometimes hate words are misspelled and by spelling correction we can get some different word from it and because of this we can lose hate words.

Challenges faced:

- Obtaining a lot of data for languages with fewer resources, such as Hindi and Marathi, is hard.
- Finding suitable libraries for stopwords in Hindi and Marathi was also tricky.
- Cleaning scrapped tweets in Hindi and Marathi involved addressing unwanted patterns and noise inherent in social media language.

Plan for the next mandates:

We intend to fine-tune pre-trained language model such as mBERT on a substantial English dataset for the hatespeech detection task. Furthermore, we plan to apply transfer learning to adapt the model to Hindi and Marathi datasets as part of our mandate 4 to enhance performance across languages.

Notebooks:

1. Preprocessing on hindi dataset -
https://www.kaggle.com/code/sanketp029/preprocess-hindi-hasoc
2. Preprocessing on english dataset -
https://colab.research.google.com/drive/1J7e1TBJXYtiymbR8AMMlGvhN3sBJ1hyW?usp=sharing
3. Preprocessing on marathi dataset-
https://colab.research.google.com/drive/1MphvwYzT3-VhqJloSM8Pl9mifAh0UhCN?usp=sharing

Datasets:

https://drive.google.com/drive/u/2/folders/1IfrRIiTDk6NrsDe9mSJar3poT-MKzj4Y