HateClassification

June 10, 2023

[1]:
```
import numpy as np
import pandas as pd
import string
import re
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to /home/sit/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /home/sit/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /home/sit/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

[1]:
```
True
```

[2]:
```
train_data = pd.read_excel('constraint_Hindi_Train.xlsx')
valid_data = pd.read_excel('Constraint_Hindi_Valid.xlsx')
train_data.head()
```

[2]:
```
   Unique ID                                              Post  \
0          1           ...
1          2           ...
2          3       9    ,  ...
3          4  @prabhav218      ...
4          5  #unlock4guidelines - -4   ...

             Labels Set
0        hate,offensive
1           non-hostile
2           non-hostile
3  defamation,offensive
4           non-hostile
```

[3]:
```
train_data.rename(columns={'Unique ID':'ID','Post':'Text','Labels Set':
 ↪'Labels'},inplace=True)
```

```
valid_data.rename(columns={'Unique ID':'ID','Post':'Text','Labels Set':
↪'Labels'},inplace=True)
train_data.head()
```

[3]:
```
   ID                                                 Text               Labels
0   1           ...         hate,offensive
1   2           ...            non-hostile
2   3       9     ,  ...            non-hostile
3   4  @prabhav218       ...  defamation,offensive
4   5  #unlock4guidelines - -4    ...            non-hostile
```

[4]:
```
print(train_data.Labels.nunique())
print(train_data.Labels.unique())
print(train_data.Labels.value_counts())
```

```
16
['hate,offensive' 'non-hostile' 'defamation,offensive' 'fake' 'hate'
 'offensive' 'fake,hate' 'defamation' 'defamation,hate'
 'defamation,hate,offensive' 'defamation,fake,offensive' 'fake,offensive'
 'defamation,fake' 'defamation,fake,hate' 'fake,hate,offensive'
 'defamation,fake,hate,offensive']
non-hostile                          3050
fake                                 1009
hate                                  478
offensive                             405
defamation                            305
hate,offensive                        163
defamation,offensive                   81
defamation,hate                        74
defamation,fake                        34
defamation,hate,offensive              28
fake,offensive                         28
fake,hate                              27
defamation,fake,offensive              24
defamation,fake,hate                    9
defamation,fake,hate,offensive          9
fake,hate,offensive                     4
Name: Labels, dtype: int64
```

[5]:
```
train_sample = train_data.loc[train_data['Labels'].
↪isin(['non-hostile','fake','hate','offensive','defamation'])]
valid_sample = valid_data.loc[valid_data['Labels'].
↪isin(['non-hostile','fake','hate','offensive','defamation'])]
train_sample.head()
```

[5]:
```
   ID                                                 Text        Labels
1   2           ...  non-hostile
2   3       9     ,  ...  non-hostile
```

```
4   5  #unlock4guidelines - -4   ...  non-hostile
5   6    UN          ...          fake
6   7    #Corona     \n#ZeeJankar...  non-hostile
```

[6]:
```python
print(len(train_sample))
print(len(valid_sample))
```

```
5247
747
```

[7]:
```python
import re
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import emoji
#from cleantext import clean

def preprocess_tweet(tweet):
    # Convert to lowercase
    tweet = tweet.lower()

    # Remove URLs
    tweet = re.sub(r"http\S+|www\S+|https\S+", "", tweet, flags=re.MULTILINE)

    # Remove usernames and hashtags symbols
    tweet = re.sub(r"\@\w+|\#+", "", tweet)

    #Clean tweet
    #tweet = clean(tweet,no_emoji=True)

    # Remove punctuation
    tweet = tweet.translate(str.maketrans("", "", string.punctuation))

    #Encoding UTF-8
    #tweet = tweet.encode(encoding='UTF-8',errors='strict')
    # Tokenize the tweet
    #tokens = word_tokenize(tweet)

    # Remove stopwords
    #stop_words = set(stopwords.words("english"))
    #tokens = [token for token in tokens if token not in stop_words]

    # Lemmatize tokens
    #lemmatizer = WordNetLemmatizer()
    #tokens = [lemmatizer.lemmatize(token) for token in tokens]
    # Join tokens back into a single string
    processed_tweet = "".join(str(tweet))

    return processed_tweet

# Example usage
tweet = "Great article on #AI! @username Check it out: https://www.example.com/
↪article"
processed_tweet = preprocess_tweet(tweet)
print(processed_tweet)
```

```
great article on ai  check it out
```

[8]:
```
train_sample['processed_text'] = train_sample['Text'].apply(lambda x :␣
↪preprocess_tweet(x))
train_sample.head()
```

```
/tmp/ipykernel_5136/21834224.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_sample['processed_text'] = train_sample['Text'].apply(lambda x :
preprocess_tweet(x))
```

[8]:
```
   ID                                               Text       Labels  \
1   2           ...  non-hostile
2   3       9    ,  ...  non-hostile
4   5  #unlock4guidelines - -4   ...  non-hostile
5   6    UN         ...          fake
6   7    #Corona     \n#ZeeJankar...  non-hostile

                                      processed_text
1           ...
2       9       ...
4  unlock4guidelines  4    ...
5    un           ...
6    corona     \nzeejankario...
```

[9]:
```
valid_sample['processed_text'] = valid_sample['Text'].apply(lambda x :␣
↪preprocess_tweet(x))
valid_sample.head()
```

```
/tmp/ipykernel_5136/3194795115.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
```

```
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  valid_sample['processed_text'] = valid_sample['Text'].apply(lambda x :
preprocess_tweet(x))
```

[9]:
```
   ID                                              Text      Labels  \
0   1          ...  non-hostile
1   2     rss       ...   defamation
2   3       /   10 ...  non-hostile
3   4     PM  -  ...  non-hostile
4   5  :   Toilet ,   ...  non-hostile

                                            processed_text
0          ...
1     rss       ...
2          10 ...
3     pm    ...
4     toilet     ...
```

[10]:
```python
#Unique Word Count
from collections import Counter
frames = [train_sample,valid_sample]
words_df = pd.concat(frames,axis=0)

results = Counter()
words_df['processed_text'].str.split().apply(results.update)
print(len(results))
```

```
20916
```

[11]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
```

[12]:
```python
from sklearn.preprocessing import LabelEncoder
lbl_enc = LabelEncoder()
```

[13]:
```python
X_train_vec = tfidf.fit_transform(train_sample['processed_text'])
X_valid_vec = tfidf.transform(valid_sample['processed_text'])
```

[14]:
```python
lbl_enc.fit(train_sample['Labels'])
train_sample['Enc_Labels'] = lbl_enc.transform(train_sample['Labels'])
valid_sample['Enc_Labels'] = lbl_enc.transform(valid_sample['Labels'])
```

```
/tmp/ipykernel_5136/2798184267.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_sample['Enc_Labels'] = lbl_enc.transform(train_sample['Labels'])
```

```
/tmp/ipykernel_5136/2798184267.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  valid_sample['Enc_Labels'] = lbl_enc.transform(valid_sample['Labels'])
```

[15]:
```
lbl_enc.classes_
```

[15]:
```
array(['defamation', 'fake', 'hate', 'non-hostile', 'offensive'],
      dtype=object)
```

[16]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(X_train_vec,train_sample['Enc_Labels'])
```

[16]:
```
RandomForestClassifier()
```

[17]:
```
y_pred = rfc.predict(X_valid_vec)
```

[18]:
```
from sklearn.metrics import classification_report
print(classification_report(valid_sample['Enc_Labels'],y_pred))
```

```
              precision    recall  f1-score   support

           0       0.33      0.02      0.04        43
           1       0.52      0.33      0.40       144
           2       0.22      0.06      0.09        68
           3       0.68      0.95      0.79       435
           4       0.35      0.19      0.25        57

    accuracy                           0.64       747
   macro avg       0.42      0.31      0.32       747
weighted avg       0.56      0.64      0.57       747
```

[41]:
```
import matplotlib.pyplot as plt
import os
import re
import shutil
import string
import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras import layers
from tensorflow.keras import losses
```

[42]:
```
train_text = train_sample['processed_text'].values
train_label = train_sample['Enc_Labels'].values
valid_text = valid_sample['processed_text'].values
valid_label = valid_sample['Enc_Labels'].values
```

[43]:
```
# Define the maximum number of words to keep in the vocabulary
max_words = 10000

# Create a tokenizer and fit it on the training text data
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(train_text)

# Convert the text data to sequences
train_sequences = tokenizer.texts_to_sequences(train_text)
valid_sequences = tokenizer.texts_to_sequences(valid_text)

# Pad the sequences to have the same length
max_sequence_length = max([len(sequence) for sequence in train_sequences])
train_data = pad_sequences(train_sequences, maxlen=max_sequence_length)
valid_data = pad_sequences(valid_sequences, maxlen=max_sequence_length)
```

[44]:
```
# Convert the labels to one-hot encoded vectors
num_classes = len(np.unique(train_label))
train_label = tf.keras.utils.to_categorical(train_label, num_classes=num_classes)
valid_label = tf.keras.utils.to_categorical(valid_label, num_classes=num_classes)
```

[45]:
```
# Define the model architecture
model = tf.keras.models.Sequential([
    tf.keras.layers.Embedding(input_dim=max_words, output_dim=100,␣
 ↪input_length=max_sequence_length),
    tf.keras.layers.Conv1D(128, 5, activation='relu'),
    tf.keras.layers.GlobalMaxPooling1D(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
```

[55]:
```
model.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy','categorical_accuracy'])
```

[56]:
```
# Train the model
model.fit(train_data, train_label, epochs=10, batch_size=32,␣
 ↪validation_data=(valid_data, valid_label))
```

```
Epoch 1/10
164/164 [==============================] - 6s 30ms/step - loss: 0.0146 -
accuracy: 0.9975 - categorical_accuracy: 0.9975 - val_loss: 1.5407 -
val_accuracy: 0.7122 - val_categorical_accuracy: 0.7122
Epoch 2/10
```

```
164/164 [==============================] - 5s 29ms/step - loss: 0.0134 -
accuracy: 0.9966 - categorical_accuracy: 0.9966 - val_loss: 1.5798 -
val_accuracy: 0.7082 - val_categorical_accuracy: 0.7082
Epoch 3/10
164/164 [==============================] - 5s 29ms/step - loss: 0.0089 -
accuracy: 0.9977 - categorical_accuracy: 0.9977 - val_loss: 1.5667 -
val_accuracy: 0.7149 - val_categorical_accuracy: 0.7149
Epoch 4/10
164/164 [==============================] - 5s 29ms/step - loss: 0.0077 -
accuracy: 0.9975 - categorical_accuracy: 0.9975 - val_loss: 1.7107 -
val_accuracy: 0.7149 - val_categorical_accuracy: 0.7149
Epoch 5/10
164/164 [==============================] - 5s 29ms/step - loss: 0.0070 -
accuracy: 0.9981 - categorical_accuracy: 0.9981 - val_loss: 1.7269 -
val_accuracy: 0.7055 - val_categorical_accuracy: 0.7055
Epoch 6/10
164/164 [==============================] - 5s 29ms/step - loss: 0.0087 -
accuracy: 0.9979 - categorical_accuracy: 0.9979 - val_loss: 1.8557 -
val_accuracy: 0.7162 - val_categorical_accuracy: 0.7162
Epoch 7/10
164/164 [==============================] - 5s 29ms/step - loss: 0.0059 -
accuracy: 0.9977 - categorical_accuracy: 0.9977 - val_loss: 2.1018 -
val_accuracy: 0.7216 - val_categorical_accuracy: 0.7216
Epoch 8/10
164/164 [==============================] - 5s 29ms/step - loss: 0.0083 -
accuracy: 0.9970 - categorical_accuracy: 0.9970 - val_loss: 1.9744 -
val_accuracy: 0.6908 - val_categorical_accuracy: 0.6908
Epoch 9/10
164/164 [==============================] - 5s 29ms/step - loss: 0.0046 -
accuracy: 0.9985 - categorical_accuracy: 0.9985 - val_loss: 1.9756 -
val_accuracy: 0.7149 - val_categorical_accuracy: 0.7149
Epoch 10/10
164/164 [==============================] - 5s 29ms/step - loss: 0.0036 -
accuracy: 0.9985 - categorical_accuracy: 0.9985 - val_loss: 1.9839 -
val_accuracy: 0.7175 - val_categorical_accuracy: 0.7175
```

```
[56]: <keras.callbacks.History at 0x7f6aa8559b50>
```

```
[48]: # Evaluate the model on the test set
loss, accuracy = model.evaluate(valid_data, valid_label)
print(f'Test loss: {loss:.4f}')
print(f'Test accuracy: {accuracy:.4f}')
```

```
24/24 [==============================] - 0s 11ms/step - loss: 1.1967 - accuracy:
0.7202
Test loss: 1.1967
Test accuracy: 0.7202
```

[57]:
```
pred = model.predict(valid_data)
```

```
24/24 [==============================] - 0s 11ms/step
```

[60]:
```
print(classification_report(np.argmax(valid_label,axis=1),np.
 ↪argmax(pred,axis=1)))
```

```
              precision    recall  f1-score   support

           0       0.44      0.33      0.37        43
           1       0.52      0.63      0.57       144
           2       0.43      0.38      0.40        68
           3       0.88      0.88      0.88       435
           4       0.52      0.40      0.46        57

    accuracy                           0.72       747
   macro avg       0.56      0.52      0.54       747
weighted avg       0.72      0.72      0.71       747
```

[ ]:
```

```