

NLP Project 2 - Help Twitter Combat Hate Speech Using NLP and Machine Learning

DESCRIPTION

Using NLP and ML, make a model to identify hate speech (racist or sexist tweets) in Twitter.

Problem Statement:

Twitter is the biggest platform where anybody and everybody can have their views heard. Some of these voices spread hate and negativity. Twitter is wary of its platform being used as a medium to spread hate.

You are a data scientist at Twitter, and you will help Twitter in identifying the tweets with hate speech and removing them from the platform. You will use NLP techniques, perform specific cleanup for tweets data, and make a robust model.

Domain: Social Media

```
# Importing required libraries
import re
import nltk
import string
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

Loading the tweets file using read_csv function from pandas package

```
data = pd.read_csv('/content/TwitterHate.csv')
data
```

	id	label	tweet
0	1	0	@user when a father is dysfunctional and is s...
1	2	0	@user @user thanks for #lyft credit i can't us...
2	3	0	bihday your majesty
3	4	0	#model i love u take with u all the time in ...
4	5	0	factsguide: society now #motivation
...
31957	31958	0	ate @user isz that youuu?ð□□□ð□□□ð□□□ð□□□ð...
31958	31959	0	to see nina turner on the airwaves trying to...
31959	31960	0	listening to sad songs on a monday morning otw...
31960	31961	1	@user #sikh #temple vandalised in in #calgary,...
31961	31962	0	thank you @user for you follow
31962 rows × 3 columns			

```
data.label.value_counts()
```

```
0    29720
1     2242
Name: label, dtype: int64
```

On looking at the value counts of the labels we can see that the data is imbalanced

▼ Get the tweets into a list for easy text cleanup and manipulation

```
# Removing unnecessary items from the columns
data.drop('id', axis=1, inplace=True)
```

```
data.sample(5)
```

	label	tweet
1398	0	today was my last day of jsms middle school, a...
4684	0	@user @user she made a song that fucking compe...
7683	0	@user i am thankful for strength. #thankful #...
17556	0	the first session of @user kicks off in just t...
2140	0	cannot wait to move into my own place ð□□□ð□□□...

▼ To cleanup

1. Normalize the casing
2. Using regular expressions, remove user handles. These begin with '@'.
3. Using regular expressions, remove URLs.
4. Using TweetTokenizer from NLTK, tokenize the tweets into individual terms.
5. Remove stop words.
6. Remove redundant terms like 'amp', 'rt', etc.
7. Remove '#' symbols from the tweet while retaining the term.

```
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Unzipping corpora/stopwords.zip.

def tweets_cleanup(tweet):
    # Normalize casing
    tweet.lower()
    tweet = re.sub('[^A-Za-z0-9]+', ' ', tweet)

    # Remove user handles that begins with @
    tweet = re.sub('@\w+\#', ' ', tweet)

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

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

    # Tokenizing the tweets
    tokenizer = nltk.TweetTokenizer()
```

```

tweet_tokens = tokenizer.tokenize(tweet)

# Remove stop words
cleaned_tweets = [w for w in tweet_tokens if w not in stop_words]



# Remove redundant terms like 'amp', 'rt', etc
final_tweets = [w for w in cleaned_tweets if w not in ("amp","rt")]

return " ".join(final_tweets)

data.tweet = data.tweet.apply(tweets_cleanup)

data = data[['tweet', 'label']]
data.sample(5)

```



	tweet	label		
13630	would reason call user name haters sistahsgeti...	0		
16942	ready coldplay coldplay headfullofdreamstour 2...	0		
10023	user people get user florida know	0		
31649	11 months kobe enjoy thank love couple instago...	0		
18079	anti islamist rally organiser neilerikson show...	1		

▼ Extra cleanup by removing items with a length of 1

```

data['length'] = data['tweet'].apply(len)
data.head()

```

	tweet	label	length		
0	user father dysfunctional selfish drags kids d...	0	60		
1	user user thanks lyft credit use cause offer w...	0	87		
2	bihday majesty	0	14		
3	model love u take u time ur	0	27		
4	factsguide society motivation	0	29		

```

data[(data['length'] == 0) | (data['length'] == 1)]

```

	tweet	label	length
3351		0	0
4411	1	0	1

```
data = data[data['length']>1]
```

▼ Checkout the top terms in the tweets

1. First, get all the tokenized terms into one large list
2. Use the counter and find the 10 most common terms

```

nltk.download('punkt')

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt.zip.
True

def tokenize(text):
    tokens = nltk.tokenize.word_tokenize(text)
    return tokens

tokenized_list = []
for tweet in data['tweet']:
    token = tokenize(tweet)

    tokenized_list.append(token)

tokenized_list_large = [i for j in tokenized_list for i in j]

from collections import Counter
top_10_common_words = [word for word, word_count in Counter(tokenized_list_large).most_common(10)]
print(top_10_common_words)

['user', 'love', 'day', 'happy', 'u', 'life', 'time', 'like', 'today', 'new']

```

▼ Data formatting for predictive modelling

1. Join the tokens back to form strings. This will be required for the vectorizers
2. Assign x and y
3. perform train_test_split using sklearn

```

from sklearn.model_selection import train_test_split

# Assigning X and y
X = data['tweet']
y = data['label']

# Splitting the data in to train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(X_train.shape, X_test.shape)

(25559,) (6390,)

```

▼ We'll use TF-IDF values for the terms as a feature to get into a vector space model

1. Import TF-IDF vectorizer from sklearn
2. Instantiate with a maximum of 5000 terms in your vocabulary
3. Fit and apply on the train set
4. Apply on the test set

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vect = TfidfVectorizer(max_features=5000)

X_train_tfidf = tfidf_vect.fit_transform(X_train)

X_test_tfidf = tfidf_vect.transform(X_test)
```

▼ Model building: Ordinary Logistic Regression

1. Instantiate logistic regression from sklearn with default parameters
2. Fit into the train data
3. Make predictions for the train and the test set

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()

lr.fit(X_train_tfidf, y_train)

train_preds = lr.predict(X_train_tfidf)
test_preds = lr.predict(X_test_tfidf)
```

▼ Model evaluation: Accuracy, recall and f1 score

1. Report the accuracy on the train set.
2. Report the recall on the train set: decent, high, or low.
3. Get the f1 score on the train set.

```
from sklearn.metrics import accuracy_score, recall_score, f1_score, classification_report

print(classification_report(y_test, test_preds))
```

	precision	recall	f1-score	support
0	0.95	1.00	0.97	5934
1	0.89	0.35	0.50	456
accuracy			0.95	6390
macro avg	0.92	0.67	0.74	6390
weighted avg	0.95	0.95	0.94	6390

```
print("Accuracy score on test set", accuracy_score(y_test, test_preds))
print("\nRecall on test set", recall_score(y_test, test_preds))
print("\nf1 score on test set", f1_score(y_test, test_preds))
```

Accuracy score on test set 0.950547730829421

Recall on test set 0.3508771929824561

f1 score on test set 0.5031446540880503

- ▼ Looks like you need to adjust the class imbalance, as the model seems to focus on the 0s.

1. Adjust the appropriate class in the LogisticRegression model.

```
# First lets look at the value counts of the label values
data['label'].value_counts()
```

```
0    29709
1     2240
Name: label, dtype: int64
```

```
# Class distribution
data['label'].value_counts()/ data.shape[0]
```

```
0    0.929888
1    0.070112
Name: label, dtype: float64
```

```
# Defining class weights to tackle imbalance
w = {0:1, 1:92}
lr2 = LogisticRegression(random_state=42, class_weight=w)
```

- ▼ Train again with the adjustment and evaluate.

1. Train the model on the train set.
2. Evaluate the predictions on the train set: accuracy, recall, and f_1 score.

```
lr2.fit(X_train_tfidf, y_train)
test_preds2 = lr2.predict(X_test_tfidf)

print("Accuracy score on test set", accuracy_score(y_test, test_preds2))
print("\nRecall on test set", recall_score(y_test, test_preds2))
print("\nf1 score on test set", f1_score(y_test, test_preds2))
```

```
Accuracy score on test set 0.8549295774647887
```

```
Recall on test set 0.8530701754385965
```

```
f1 score on test set 0.4563049853372434
```

- ▼ Regularization and Hyperparameter tuning:

1. Import GridSearch and StratifiedKFold because of class imbalance.
2. Provide the parameter grid to choose for 'C' and 'penalty' parameters.
3. Use a balanced class weight while instantiating the logistic regression.

```
from sklearn.model_selection import GridSearchCV, StratifiedKFold
```

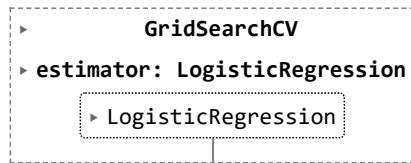
```
# Setting C and penalty parameters
C = np.arange(0.5, 20.0, 0.5)
penalty_values = ["l1", "l2"]
```

```
param_grid = {"C": C, "penalty": penalty_values}
```

- ▼ Find the parameters with the best recall in cross validation.

1. Choose 'recall' as the metric for scoring.
2. Choose stratified 4 fold cross validation scheme.
3. Fit into the train set.

```
grid = GridSearchCV(lr2, param_grid, scoring='recall', cv=4)
grid.fit(X_train_tfidf, y_train)
```



▼ What are the best parameters

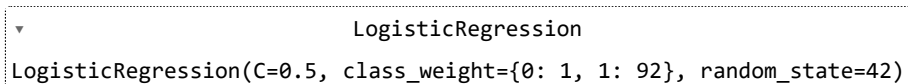
```
print("Best params", grid.best_params_)
print("Best score", grid.best_score_)

Best params {'C': 0.5, 'penalty': 'l2'}
Best score 0.9159192825112107
```

▼ Predict and evaluate using the best estimator.

1. Use the best estimator from the grid search to make predictions on the test set.
2. What is the recall on the test set for the toxic comments?
3. What is the f₁ score?

```
import random
lr3 = LogisticRegression(random_state=42, class_weight=w, C=0.5, penalty='l2')
lr3.fit(X_train_tfidf, y_train)
```



```
test_preds3 = lr3.predict(X_test_tfidf)

print("f1 score on test set", f1_score(y_test, test_preds3))

f1 score on test set 0.4084934277047522
```

✓ 0s completed at 11:49 AM

