Help Twitter Combat Hate Speech Using NLP and Machine Learning

1 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

Analysis to be done: Clean up tweets and build a classification model by using NLP techniques, cleanup specific for tweets data, regularization and hyperparameter tuning using stratified k-fold and cross validation to get the best model.

Content:

id: identifier number of the tweet

Label: 0 (non-hate) /1 (hate)

Tweet: the text in the tweet

[1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
import nltk
import re
import string
import warnings
```

```
warnings.filterwarnings('ignore')
```

1.0.1 1. Load the tweets file using read_csv function from Pandas package.

[3]:
```
data = pd.read_csv('TwitterHate.csv', encoding="utf-8")
data.head(3)
```

[3]:
```
   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
```

[4]:
```
data.shape
```

[4]:
```
(31962, 3)
```

[5]:
```
data.isnull().sum()
```

[5]:
```
id       0
label    0
tweet    0
dtype: int64
```

1.0.2 2. Get the tweets into a list for easy text cleanup and manipulation.

[6]:
```
data = data[['tweet', 'label']]
```

[7]:
```
data.shape
```

[7]:
```
(31962, 2)
```

[8]:
```
data.isnull().sum()
```

[8]:
```
tweet    0
label    0
dtype: int64
```

[9]:
```
data.head(3)
```

[9]:
```
                                               tweet  label
0   @user when a father is dysfunctional and is s…      0
1  @user @user thanks for #lyft credit i can't us…      0
2                                bihday your majesty      0
```

1.0.3 3. 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.

[10]:
```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
```

[11]:
```
from nltk.tokenize import TweetTokenizer
```

[12]:
```
def preprocess_tweet_text(tweet):
    # Normalize the casing.
    tweet.lower()

    tweet = re.sub('[^A-Za-z0-9]+', ' ', tweet)

    # Remove user @ references and '#' from tweet
    tweet = re.sub(r'\@\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))

    #Using TweetTokenizer from NLTK, tokenize the tweets into individual terms.
    tk = TweetTokenizer()
    tweet_tokens = tk.tokenize(tweet)

    # Remove stopwords
    filtered_words = [w for w in tweet_tokens if not w in stop_words]

    # Remove redundant terms like 'amp', 'rt', etc.
    filtered_words_final = [w for w in filtered_words if not w in ('amp', 'rt')]

    return " ".join(filtered_words_final)
```

[13]:
```
data.tweet = data['tweet'].apply(preprocess_tweet_text)
```

[14]:
```
data.head(3)
```

[14]:

	tweet	label
0	user father dysfunctional selfish drags kids d…	0
1	user user thanks lyft credit use cause offer w…	0
2	bihday majesty	0

1.0.4 4. Extra cleanup by removing terms with a length of 1.

[15]:
```
data['length']= data['tweet'].apply(len)
data.head(3)
```

[15]:

	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

[16]:
```
data.shape
```

[16]: (31962, 3)

[17]:
```
len(data[data['length'] == 0])
```

[17]: 11

[18]:
```
len(data[data['length'] == 1])
```

[18]: 2

[19]:
```
len(data[data['length']>1])
```

[19]: 31949

[20]:
```
data[data['length'] == 0]
```

[20]:

	tweet	label	length
3351		0	0
7222		0	0
10461		0	0
13038		0	0
15434		0	0
16250		0	0
20261		0	0
22709		0	0

```
25629            1       0
29803            1       0
31781            0       0
```

[21]:
```
data = data[data['length']>1]
```

[22]:
```
data.shape
```

[22]:
```
(31949, 3)
```

[23]:
```
data.head(3)
```

[23]:
```
                                                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
```

1.0.5 5. Check out 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.

[24]:
```
def token_text(text):
    # tokenize the text into a list of words
    tokens = nltk.tokenize.word_tokenize(text)
    return tokens
```

[25]:
```
# Final list with tokenized words
tokenized_large = []

# Iterating over each string in data
for x in data['tweet']:
    # Calliing preprocess text function
    token = token_text(x)

    tokenized_large.append(token)

flattened_tokeninized_final = [i for j in tokenized_large for i in j]
```

[26]:
```
type(flattened_tokeninized_final)
```

[26]:
```
list
```

[27]:
```
# Use the counter and find the 10 most common terms.
from collections import Counter
```

```
most_common_words= [word for word, word_count in␣
 ↪Counter(flattened_tokeninized_final).most_common(10)]
print(most_common_words)
```

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

1.0.6 6. Data formatting for predictive modeling:

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.

[28]:
```
data.head()
```

[28]:

	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

[29]:
```
data.shape
```

[29]:
```
(31949, 3)
```

[30]:
```
X = data['tweet']
Y = data['label']
```

[31]:
```
# Splitting data into train and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y ,test_size=0.20,␣
 ↪random_state=10)
```

1.0.7 7. 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.

[32]:
```
# 1. Import TF-IDF  vectorizer from sklearn.
from sklearn.feature_extraction.text import TfidfVectorizer
```

[33]:
```
# 2. Instantiate with a maximum of 5000 terms in your vocabulary.
tfidf_vect = TfidfVectorizer(max_features =5000)
```

[34]:
```
# 3. Fit and apply on the train set.
X_train_tfidf = tfidf_vect.fit_transform(X_train)
```

[35]:
```
# 4. Apply on the test set.
X_test_tfidf = tfidf_vect.transform(X_test)
```

1.0.8 8. 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.

[36]:
```
# Logistic Regression
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
```

[37]:
```
lr.fit(X_train_tfidf, y_train)
```

[37]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

[38]:
```
# Make predictions for the train and the test set.
predictions_train = lr.predict(X_train_tfidf)
predictions_test = lr.predict(X_test_tfidf)
```

1.0.9 9. Model evaluation: Accuracy, recall, and f_1 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.

[39]:
```
from sklearn.metrics import classification_report, recall_score,␣
↪accuracy_score, f1_score
```

[40]:
```
print(classification_report(y_test, predictions_test))
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      5965
           1       0.93      0.35      0.51       425

    accuracy                           0.95      6390
   macro avg       0.94      0.67      0.74      6390
weighted avg       0.95      0.95      0.95      6390
```

[41]:
```
print("Recall : ", recall_score(y_test, predictions_test))
```

```
Recall :  0.34823529411764703
```

Recall is 34%. ie Sensitivity or true positive rate is 34%, which is very low.

[42]:
```
print("f1_score : ",f1_score(y_test, predictions_test))
```

```
f1_score :  0.5068493150684932
```

[43]:
```
print(accuracy_score(y_test, predictions_test))
```

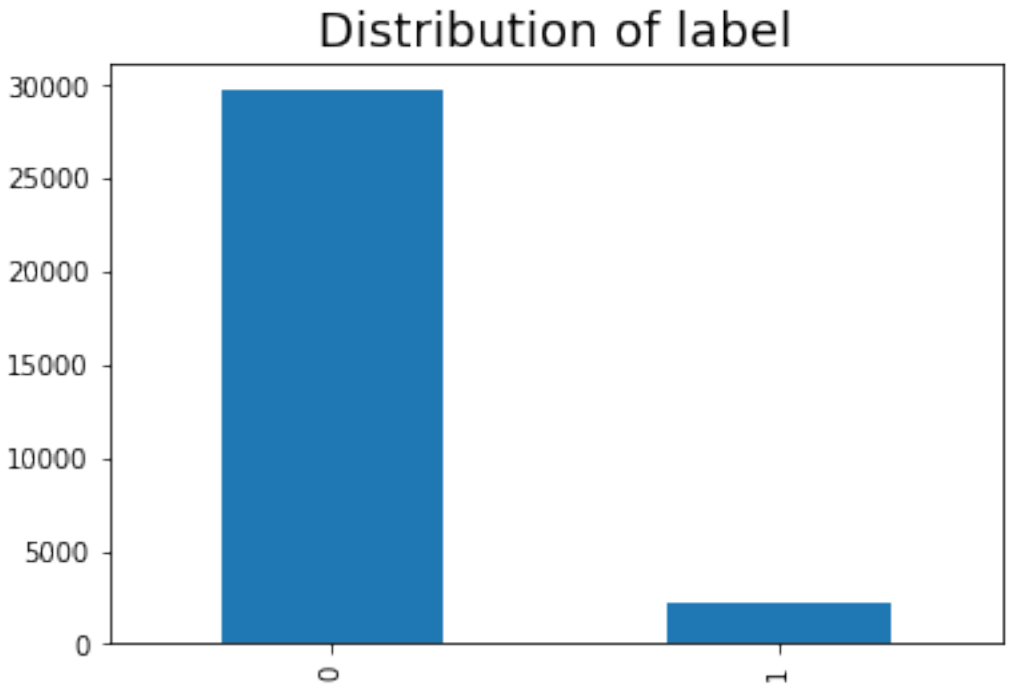
```
0.9549295774647887
```

1.0.10 10. 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.

[44]:
```
data['label'].value_counts().plot(kind='bar')
plt.title("Distribution of label", size=18)
```

[44]:
```
Text(0.5, 1.0, 'Distribution of label')
```



[45]:
```
# distribution
data['label'].value_counts()/data.shape[0]
```

[45]:
```
0    0.929888
1    0.070112
Name: label, dtype: float64
```

[46]:
```
# define class weights
w = {0:1, 1:92} # lable distribution % is 1:0==7:92

lr_2 = LogisticRegression(random_state=11, class_weight=w)
```

1.0.11 11. 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.

[47]:
```
lr_2.fit(X_train_tfidf, y_train)
```

[47]:
```
LogisticRegression(C=1.0, class_weight={0: 1, 1: 92}, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=11, solver='lbfgs', tol=0.0001, verbose=0,
```

```
                   warm_start=False)
```

[48]:
```
predictions_2 = lr_2.predict(X_test_tfidf)
```

[49]:
```
print("Accuracy : ", accuracy_score(y_test, predictions_2))
print("  Recall : ", recall_score(y_test, predictions_2))
print("f1_score : ", f1_score(y_test, predictions_2))
```

```
Accuracy :  0.8539906103286385
  Recall :  0.8870588235294118
f1_score :  0.4469472436277415
```

1.0.12 12. 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.

[50]:
```
from sklearn.model_selection import GridSearchCV, StratifiedKFold
```

[51]:
```
C_val = np.arange(0.5, 20.0, 0.5)
penalty_val = ["l1", "l2"]
```

[52]:
```
hyperparam_grid = {"penalty": penalty_val, "C": C_val }
```

[53]:
```
lr_3 = LogisticRegression(random_state=11, class_weight=w)
```

1.0.13 13. 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.

[54]:
```
grid = GridSearchCV(lr_3, hyperparam_grid, scoring="recall", cv=4)
```

[55]:
```
grid.fit(X_train_tfidf, y_train)
```

[55]:
```
GridSearchCV(cv=4, error_score=nan,
             estimator=LogisticRegression(C=1.0, class_weight={0: 1, 1: 92},
                                          dual=False, fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='auto',
```

```
                                          n_jobs=None, penalty='l2',
                                          random_state=11, solver='lbfgs',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': array([ 0.5,  1. ,  1.5,  2. ,  2.5,  3. ,  3.5,
       4. ,  4.5,  5. ,  5.5,
              6. ,  6.5,  7. ,  7.5,  8. ,  8.5,  9. ,  9.5, 10. , 10.5, 11. ,
             11.5, 12. , 12.5, 13. , 13.5, 14. , 14.5, 15. , 15.5, 16. , 16.5,
             17. , 17.5, 18. , 18.5, 19. , 19.5]),
                         'penalty': ['l1', 'l2']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='recall', verbose=0)
```

1.0.14 14. What are the best parameters?

[56]:
```
print(f'Best recall: {grid.best_score_} with param: {grid.best_params_}')
```

```
Best recall: 0.9019215995176552 with param: {'C': 0.5, 'penalty': 'l2'}
```

1.0.15 15. 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 f_1 score?

[57]:
```
lr_4 = LogisticRegression(random_state=11, class_weight=w, C=0.5, penalty='l2')
lr_4.fit(X_train_tfidf, y_train)
```

[57]:
```
LogisticRegression(C=0.5, class_weight={0: 1, 1: 92}, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=11, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

[58]:
```
predictions = lr_4.predict(X_test_tfidf)
```

[59]:
```
print("Accuracy : ", accuracy_score(y_test, predictions))
print("  Recall : ", recall_score(y_test, predictions))
print("f1_score : ", f1_score(y_test, predictions))
```

```
Accuracy :  0.8195618153364632
  Recall :  0.9129411764705883
f1_score :  0.4022809745982374
```

[ ]:

[ ]: