# NLP_Assement-Project_02

August 20, 2022

# 1 Project Name:- Help Twitter Combat Hate Speech Using NLP and Machine Learning

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
# Import Require library

import pandas as pd
import re
from nltk.tokenize import TweetTokenizer
from nltk.corpus import stopwords
from string import punctuation
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import␣
↪accuracy_score,classification_report,recall_score,f1_score
```

[3]:
```
# Set column width 150 for showing most data
pd.set_option('display.max_colwidth',150)
```

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

[4]:
```
data = pd.read_csv('TwitterHate.csv',usecols=['label','tweet'])
data.head(10)
```

[4]:
```
   label  \
0      0
1      0
2      0
3      0
4      0
5      0
```

```
6      0
7      0
8      0
9      0

                                                                 tweet
0                                            @user when a father is
dysfunctional and is so selfish he drags his kids into his dysfunction.   #run
1                        @user @user thanks for #lyft credit i can't use cause
they don't offer wheelchair vans in pdx.    #disapointed #getthanked
2
bihday your majesty
3                                                            #model   i love u
take with u all the time in urð ±!!! ð  ð  ð  ð
ð ¦ð ¦ð ¦
4
factsguide: society now    #motivation
5                              [2/2] huge fan fare and big talking before they
leave. chaos and pay disputes when they get there. #allshowandnogo
6                                                                        @user
camping tomorrow @user @user @user @user @user @user @user dannyâ¦
7  the next school year is the year for exams.ð ¯ can't think about that ð
#school #exams   #hate #imagine #actorslife #revolutionschool #girl
8                                                           we won!!! love the
land!!! #allin #cavs #champions #cleveland #clevelandcavaliers  â¦
9
@user @user welcome here !  i'm   it's so #gr8 !
```

[5]:
```python
# Check the shape of tha dataset
data.shape
```

[5]: (31962, 2)

[6]:
```python
# Check the information of dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31962 entries, 0 to 31961
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   label   31962 non-null  int64
 1   tweet   31962 non-null  object
dtypes: int64(1), object(1)
memory usage: 499.5+ KB
```

[7]:
```python
# Check the missing value in dataset
data.isna().sum().any()
```

[7]:
```
False
```

[8]:
```python
data['label'].value_counts()
```

[8]:
```
0    29720
1     2242
Name: label, dtype: int64
```

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

[9]:
```python
tweet = data['tweet'].values
```

[10]:
```python
tweet[:5]
```

[10]:
```
array([' @user when a father is dysfunctional and is so selfish he drags his
kids into his dysfunction.   #run',
       "@user @user thanks for #lyft credit i can't use cause they don't offer
wheelchair vans in pdx.    #disapointed #getthanked",
       '  bihday your majesty',
       '#model   i love u take with u all the time in urð\x9f\x93±!!! ð\x9f\x98\
x99ð\x9f\x98\x8eð\x9f\x91\x84ð\x9f\x91\x85ð\x9f\x92¦ð\x9f\x92¦ð\x9f\x92¦  ',
       ' factsguide: society now    #motivation'], dtype=object)
```

# 4 3. To cleanup:

## 4.1 i. Normalize the casing.

[11]:
```python
tweet_lower = [term.lower() for term in tweet]
```

[12]:
```python
tweet_lower[:5]
```

[12]:
```
[' @user when a father is dysfunctional and is so selfish he drags his kids into
his dysfunction.   #run',
 "@user @user thanks for #lyft credit i can't use cause they don't offer
wheelchair vans in pdx.    #disapointed #getthanked",
 '  bihday your majesty',
 '#model   i love u take with u all the time in urð\x9f\x93±!!! ð\x9f\x98\x99ð\x
9f\x98\x8eð\x9f\x91\x84ð\x9f\x91\x85ð\x9f\x92¦ð\x9f\x92¦ð\x9f\x92¦  ',
 ' factsguide: society now    #motivation']
```

## 4.2 ii. Using regular expressions, remove user handles. These begin with '@'.

[13]:
```
tweet_nonuser = [re.sub('@\w+','',term) for term in tweet_lower]
```

[14]:
```
tweet_nonuser[:5]
```

[14]:
```
['  when a father is dysfunctional and is so selfish he drags his kids into his
dysfunction.   #run',
 "  thanks for #lyft credit i can't use cause they don't offer wheelchair vans
in pdx.    #disapointed #getthanked",
 '  bihday your majesty',
 '#model   i love u take with u all the time in urð\x9f\x93±!!! ð\x9f\x98\x99ð\x
9f\x98\x8eð\x9f\x91\x84ð\x9f\x91\x85ð\x9f\x92¦ð\x9f\x92¦ð\x9f\x92¦  ',
 ' factsguide: society now    #motivation']
```

## 4.3 iii. Using regular expressions, remove URLs.

[15]:
```
tweet_nonurl = [re.sub('\w+://\S+','',term) for term in tweet_nonuser]
```

[16]:
```
tweet_nonurl[:5]
```

[16]:
```
['  when a father is dysfunctional and is so selfish he drags his kids into his
dysfunction.   #run',
 "  thanks for #lyft credit i can't use cause they don't offer wheelchair vans
in pdx.    #disapointed #getthanked",
 '  bihday your majesty',
 '#model   i love u take with u all the time in urð\x9f\x93±!!! ð\x9f\x98\x99ð\x
9f\x98\x8eð\x9f\x91\x84ð\x9f\x91\x85ð\x9f\x92¦ð\x9f\x92¦ð\x9f\x92¦  ',
 ' factsguide: society now    #motivation']
```

## 4.4 iv. Using TweetTokenizer from NLTK, tokenize the tweets into individual terms.

[17]:
```
tweet_token = [TweetTokenizer().tokenize(sent) for sent in tweet_nonurl]
```

[18]:
```
print(tweet_token[:5])
```

```
[['when', 'a', 'father', 'is', 'dysfunctional', 'and', 'is', 'so', 'selfish',
'he', 'drags', 'his', 'kids', 'into', 'his', 'dysfunction', '.', '#run'],
['thanks', 'for', '#lyft', 'credit', 'i', "can't", 'use', 'cause', 'they',
"don't", 'offer', 'wheelchair', 'vans', 'in', 'pdx', '.', '#disapointed',
'#getthanked'], ['bihday', 'your', 'majesty'], ['#model', 'i', 'love', 'u',
'take', 'with', 'u', 'all', 'the', 'time', 'in', 'urð', '\x9f', '\x93', '±',
'!', '!', '!', 'ð', '\x9f', '\x98', '\x99', 'ð', '\x9f', '\x98', '\x8e', 'ð',
'\x9f', '\x91', '\x84', 'ð', '\x9f', '\x91', 'ð', '\x9f', '\x92', '¦', 'ð',
'\x9f', '\x92', '¦', 'ð', '\x9f', '\x92', '¦'], ['factsguide', ':', 'society',
'now', '#motivation']]
```

## 4.5 v. Remove stop words.

[19]:
```
stop_words = stopwords.words('english')
```

[20]:
```
def remove_stopwords(input_word):
    remove = [term for term in input_word if term not in stop_words]
    return remove

tweet_nonstopword = [remove_stopwords(sent) for sent in tweet_token]
```

[21]:
```
print(tweet_nonstopword[:5])
```

```
[['father', 'dysfunctional', 'selfish', 'drags', 'kids', 'dysfunction', '.',
'#run'], ['thanks', '#lyft', 'credit', "can't", 'use', 'cause', 'offer',
'wheelchair', 'vans', 'pdx', '.', '#disapointed', '#getthanked'], ['bihday',
'majesty'], ['#model', 'love', 'u', 'take', 'u', 'time', 'urð', '\x9f', '\x93',
'±', '!', '!', '!', 'ð', '\x9f', '\x98', '\x99', 'ð', '\x9f', '\x98', '\x8e',
'ð', '\x9f', '\x91', '\x84', 'ð', '\x9f', '\x91', 'ð', '\x9f', '\x92', '¦', 'ð',
'\x9f', '\x92', '¦', 'ð', '\x9f', '\x92', '¦'], ['factsguide', ':', 'society',
'#motivation']]
```

[22]:
```
punct = list(punctuation)
punct.extend(['...','``',"''",".."])
```

[23]:
```
def remove_punctuation(input_word):
    remove = [term for term in input_word if term not in punct]
    return remove

tweet_nonpunctuation = [remove_punctuation(sent) for sent in tweet_nonstopword]
```

[24]:
```
print(tweet_nonpunctuation[:5])
```

```
[['father', 'dysfunctional', 'selfish', 'drags', 'kids', 'dysfunction', '#run'],
['thanks', '#lyft', 'credit', "can't", 'use', 'cause', 'offer', 'wheelchair',
'vans', 'pdx', '#disapointed', '#getthanked'], ['bihday', 'majesty'], ['#model',
'love', 'u', 'take', 'u', 'time', 'urð', '\x9f', '\x93', '±', 'ð', '\x9f',
'\x98', '\x99', 'ð', '\x9f', '\x98', '\x8e', 'ð', '\x9f', '\x91', '\x84', 'ð',
'\x9f', '\x91', 'ð', '\x9f', '\x92', '¦', 'ð', '\x9f', '\x92', '¦', 'ð', '\x9f',
'\x92', '¦'], ['factsguide', 'society', '#motivation']]
```

## 4.6 vi. Remove redundant terms like 'amp', 'rt', etc.

[25]:
```
context = ['amp','rt']
```

[26]:
```
def stop_context(sent):
    return [term for term in sent if term not in context]

tweet_noncontext = [stop_context(tweet) for tweet in tweet_nonpunctuation]
```

[27]:
```
print(tweet_noncontext[:5])
```

```
[['father', 'dysfunctional', 'selfish', 'drags', 'kids', 'dysfunction', '#run'],
['thanks', '#lyft', 'credit', "can't", 'use', 'cause', 'offer', 'wheelchair',
'vans', 'pdx', '#disapointed', '#getthanked'], ['bihday', 'majesty'], ['#model',
'love', 'u', 'take', 'u', 'time', 'urð', '\x9f', '\x93', '±', 'ð', '\x9f',
'\x98', '\x99', 'ð', '\x9f', '\x98', '\x8e', 'ð', '\x9f', '\x91', '\x84', 'ð',
'\x9f', '\x91', 'ð', '\x9f', '\x92', '¦', 'ð', '\x9f', '\x92', '¦', 'ð', '\x9f',
'\x92', '¦'], ['factsguide', 'society', '#motivation']]
```

### 4.7 vii. Remove '#' symbols from the tweet while retaining the term.

[28]:
```
def remove_symbol(sent):
    return [re.sub('#','',term) for term in sent]
```

[29]:
```
tweet_nonsymbol = [remove_symbol(tweet) for tweet in tweet_noncontext]
```

[30]:
```
print(tweet_nonsymbol[:5])
```

```
[['father', 'dysfunctional', 'selfish', 'drags', 'kids', 'dysfunction', 'run'],
['thanks', 'lyft', 'credit', "can't", 'use', 'cause', 'offer', 'wheelchair',
'vans', 'pdx', 'disapointed', 'getthanked'], ['bihday', 'majesty'], ['model',
'love', 'u', 'take', 'u', 'time', 'urð', '\x9f', '\x93', '±', 'ð', '\x9f',
'\x98', '\x99', 'ð', '\x9f', '\x98', '\x8e', 'ð', '\x9f', '\x91', '\x84', 'ð',
'\x9f', '\x91', 'ð', '\x9f', '\x92', '¦', 'ð', '\x9f', '\x92', '¦', 'ð', '\x9f',
'\x92', '¦'], ['factsguide', 'society', 'motivation']]
```

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

[31]:
```
def clean_tweet(sent):
    return [term for term in sent if len(term)>1]
```

[32]:
```
tweet_clean = [clean_tweet(sent) for sent in tweet_nonsymbol]
```

[33]:
```
print(tweet_clean[:5])
```

```
[['father', 'dysfunctional', 'selfish', 'drags', 'kids', 'dysfunction', 'run'],
['thanks', 'lyft', 'credit', "can't", 'use', 'cause', 'offer', 'wheelchair',
'vans', 'pdx', 'disapointed', 'getthanked'], ['bihday', 'majesty'], ['model',
'love', 'take', 'time', 'urð'], ['factsguide', 'society', 'motivation']]
```

# 6 5. Check out the top terms in the tweets:

## 6.1 i. First, get all the tokenized terms into one large list.

[34]:
```
tweet_list = []
for token in tweet_clean:
    tweet_list.extend(token)
```

## 6.2 ii. Use the counter and find the 10 most common terms.

[35]:
```
most_common_10 = Counter(tweet_list)
most_common_10.most_common(10)
```

[35]:
```
[('love', 2748),
 ('day', 2276),
 ('happy', 1684),
 ('time', 1131),
 ('life', 1118),
 ('like', 1047),
 ("i'm", 1018),
 ('today', 1013),
 ('new', 994),
 ('thankful', 946)]
```

# 7 6. Data formatting for predictive modeling:

## 7.1 i. Join the tokens back to form strings. This will be required for the vectorizers.

[36]:
```
tweet_clean = [' '.join(term) for term in tweet_clean]
```

[37]:
```
tweet_clean[:5]
```

[37]:
```
['father dysfunctional selfish drags kids dysfunction run',
 "thanks lyft credit can't use cause offer wheelchair vans pdx disapointed
getthanked",
 'bihday majesty',
 'model love take time urð',
 'factsguide society motivation']
```

## 7.2 ii. Assign x and y.

[38]:
```
len(tweet_clean)
```

[38]:
```
31962
```

[39]:
```
len(data['label'])
```

[39]: 31962

[40]:
```
x = tweet_clean
y = data['label']
```

### 7.3 iii. Perform train_test_split using sklearn.

[41]:
```
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.2,random_state=42)
```

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

### 8.1 i. Import TF-IDF vectorizer from sklearn.

[42]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

### 8.2 ii. Instantiate with a maximum of 5000 terms in your vocabulary.

[43]:
```
vector = TfidfVectorizer(max_features=5000)
```

### 8.3 iii. Fit and apply on the train set.

[44]:
```
xtrain_tfidf = vector.fit_transform(xtrain)
```

### 8.4 iv. Apply on the test set.

[45]:
```
xtest_tfidf = vector.transform(xtest)
```

[46]:
```
xtrain_tfidf.shape,xtest_tfidf.shape
```

[46]: ((25569, 5000), (6393, 5000))

## 9 8. Model building: Ordinary Logistic Regression

### 9.1 i. Instantiate Logistic Regression from sklearn with default parameters.

[47]:
```
model = LogisticRegression()
```

### 9.2 ii. Fit into the train data.

[48]:
```
model.fit(xtrain_tfidf,ytrain)
```

[48]:
```
LogisticRegression()
```

### 9.3 iii. Make predictions for the train and the test set.

[49]:
```
y_train_pred = model.predict(xtrain_tfidf)
y_test_pred = model.predict(xtest_tfidf)
```

## 10 9. Model evaluation: Accuracy, recall, and f_1 score.

### 10.1 i. Report the accuracy on the train set.

[50]:
```
print('Accuracy Score on Train Dataset:',accuracy_score(ytrain,y_train_pred))
```

```
Accuracy Score on Train Dataset: 0.9563142868317103
```

### 10.2 ii. Report the recall on the train set: decent, high, or low.

[51]:
```
print('Recall Score on Train dataset:',␣
 ↪recall_score(ytrain,y_train_pred,average='weighted'))
```

```
Recall Score on Train dataset: 0.9563142868317103
```

### 10.3 iii. Get the f1 score on the train set.

[52]:
```
print('F1 score on Train dataset:',␣
 ↪f1_score(ytrain,y_train_pred,average='weighted'))
```

```
F1 score on Train dataset: 0.9477746245284707
```

[53]:
```
print('Classification Report on Train Dataset:')
print(classification_report(ytest,y_test_pred))
```

```
Classification Report on Train Dataset:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97      5937
           1       0.91      0.34      0.50       456

    accuracy                           0.95      6393
   macro avg       0.93      0.67      0.73      6393
weighted avg       0.95      0.95      0.94      6393
```

# 11 10. Looks like you need to adjust the class imbalance, as the model seems to focus on the 0s.

## 11.1 i. Looks like you need to adjust the class imbalance, as the model seems to focus on the 0s.

[54]:
```
model_imbalance = LogisticRegression(class_weight='balanced')
```

# 12 11. Train again with the adjustment and evaluate.

## 12.1 i. Train the model on the train set.

[55]:
```
model_imbalance.fit(xtrain_tfidf,ytrain)
```

[55]:
```
LogisticRegression(class_weight='balanced')
```

[56]:
```
y_train_pred = model_imbalance.predict(xtrain_tfidf)
y_test_pred = model_imbalance.predict(xtest_tfidf)
```

## 12.2 ii. Evaluate the predictions on the train set: accuracy, recall, and f_1 score.

[57]:
```
print('Accuracy on Class Imbalance Train Dataset:
↪',accuracy_score(ytest,y_test_pred))
```

```
Accuracy on Class Imbalance Train Dataset: 0.923197246988894
```

[58]:
```
print('Recall and F1 Score on Class Imbalace Train Dataset:')
print(classification_report(ytest,y_test_pred))
```

```
Recall and F1 Score on Class Imbalace Train Dataset:
              precision    recall  f1-score   support

           0       0.98      0.93      0.96      5937
           1       0.48      0.80      0.60       456

    accuracy                           0.92      6393
   macro avg       0.73      0.87      0.78      6393
weighted avg       0.95      0.92      0.93      6393
```

# 13 12. Regularization and Hyperparameter tuning:

## 13.1 i. Import GridSearch and StratifiedKFold because of class imbalance

[59]:
```
from sklearn.model_selection import GridSearchCV,StratifiedKFold
```

## 13.2 ii. Provide the parameter grid to choose for 'C' and 'penalty' parameters.

[60]:
```
params = {'C': [0.01,0.1,1,10,100],
              'penalty': ["l1","l2"]}
```

## 13.3 iii. Use a balanced class weight while instantiating the logistic regression.

[61]:
```
model_gridsearch = LogisticRegression(class_weight='balanced')
```

# 14 13. Find the parameters with the best recall in cross validation.

## 14.1 i. Choose 'recall' as the metric for scoring.

## 14.2 ii. Choose stratified 4 fold cross validation scheme.

[62]:
```
grid_search =␣
 ↪GridSearchCV(estimator=model_gridsearch,param_grid=params,cv=StratifiedKFold(4),n_jobs=-1,v
                           scoring='recall')
```

## 14.3 iii. Fit into the train set.

[63]:
```
%%time
grid_search.fit(xtrain_tfidf,ytrain)
```

```
Fitting 4 folds for each of 10 candidates, totalling 40 fits
Wall time: 9.53 s
```

[63]:
```
GridSearchCV(cv=StratifiedKFold(n_splits=4, random_state=None, shuffle=False),
             estimator=LogisticRegression(class_weight='balanced'), n_jobs=-1,
             param_grid={'C': [0.01, 0.1, 1, 10, 100], 'penalty': ['l1', 'l2']},
             scoring='recall', verbose=1)
```

# 15 14. What are the best parameters?

[64]:
```
grid_search.best_estimator_
```

[64]:
```
LogisticRegression(C=1, class_weight='balanced')
```

[65]:
```
grid_search.best_params_
```

[65]: {'C': 1, 'penalty': 'l2'}

# 16 15. Predict and evaluate using the best estimator.

## 16.1 i. Use the best estimator from the grid search to make predictions on the test set.

[66]:
```
y_test_grid_pred = grid_search.best_estimator_.predict(xtest_tfidf)
```

## 16.2 ii. What is the recall on the test set for the toxic comments?

[67]:
```
print('Recall Score on Test Dataset:',␣
↪recall_score(ytest,y_test_grid_pred,average='weighted'))
```

```
Recall Score on Test Dataset: 0.923197246988894
```

## 16.3 iii. What is the f_1 score?

[68]:
```
print('F1 score:', f1_score(ytest,y_test_grid_pred,average='weighted'))
```

```
F1 score: 0.9318895024646877
```

[69]:
```
print('Classification Report on GridSearchCV:')
print(classification_report(ytest,y_test_grid_pred))
```

```
Classification Report on GridSearchCV:
              precision    recall  f1-score   support

           0       0.98      0.93      0.96      5937
           1       0.48      0.80      0.60       456

    accuracy                           0.92      6393
   macro avg       0.73      0.87      0.78      6393
weighted avg       0.95      0.92      0.93      6393
```

[ ]:
```

```