Assessment

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 Scenario:

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

Tasks

1. Get the tweets into a list for easy text clean-up and manipulation.

Dataset consists of total of 31962 tweets, among which 29720 tweets are non hate a nd 2242 is hate. Here dataset is imbalanced.

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

[2]: data = pd.read_csv('TwitterHate.csv')

[3]: data.head()

[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
3  4   0      #model i love u take with u all the time in ...
4  5   0      factsguide: society now #motivation

[4]: tweets = data['tweet']
     tweets

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

2. 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.

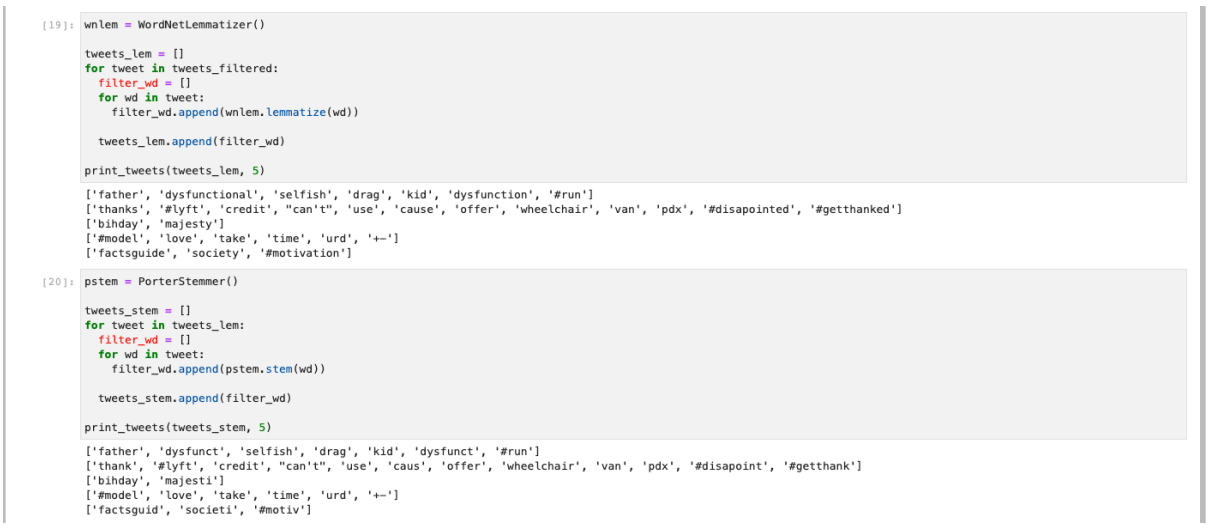
```
[8]: tweets_clean = [re.sub('@([a-z]+[a-z0-9-_]+)','',tweet).strip() for tweet in tweets]
     print_tweets(tweets_clean, 5)

     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    ±!!! ð     ð     ð     ð     ð    ¦ð    ¦ð    ¦
     factsguide: society now    #motivation

[9]: url_re = 'www\.([a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),])+([a-z0-9])'
     tweets_clean = [re.sub(url_re,'',tweet).strip() for tweet in tweets]

     for tweet in tweets_clean:
       if(re.search(url_re, tweet)):
         print(tweet)

[10]: tknzr = TweetTokenizer( strip_handles=True, reduce_len=True)

[11]: tweets_tokenized = [tknzr.tokenize(tweet) for tweet in tweets_clean]
      print_tweets(tweets_tokenized, 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']
```

5. Remove stop words.
6. Remove redundant terms like 'amp', 'rt', etc.
7. Remove '#' symbols from the tweet while retaining the term.

3. Extra clean-up by removing terms with a length of 1.

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

[16]: tweets_filter_sw = []
      for tweet in tweets_filtered:
        filter_wd = []
        for wd in tweet:
          if wd not in stop_words:
            filter_wd.append(wd)

        tweets_filter_sw.append(filter_wd)

      print_tweets(tweets_filter_sw, 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', 'urd', '', '', '+–', '!', '!', '!', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '|', '', '', '|', '', '', '|']
      ['factsguide', ':', 'society', '#motivation']

[17]: tweets_filtered = []
      for tweet in tweets_filter_sw:
        filter_wd = []
        for wd in tweet:
          if len(wd) > 1:
            filter_wd.append(wd)

        tweets_filtered.append(filter_wd)

      print_tweets(tweets_filtered, 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', 'urd', '+–']
      ['factsguide', 'society', '#motivation']
```

4. 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



```
[19]: wnlem = WordNetLemmatizer()

      tweets_lem = []
      for tweet in tweets_filtered:
        filter_wd = []
        for wd in tweet:
          filter_wd.append(wnlem.lemmatize(wd))

        tweets_lem.append(filter_wd)

      print_tweets(tweets_lem, 5)

      ['father', 'dysfunctional', 'selfish', 'drag', 'kid', 'dysfunction', '#run']
      ['thanks', '#lyft', 'credit', "can't", 'use', 'cause', 'offer', 'wheelchair', 'van', 'pdx', '#disapointed', '#getthanked']
      ['bihday', 'majesty']
      ['#model', 'love', 'take', 'time', 'urd', '+–']
      ['factsguide', 'society', '#motivation']

[20]: pstem = PorterStemmer()

      tweets_stem = []
      for tweet in tweets_lem:
        filter_wd = []
        for wd in tweet:
          filter_wd.append(pstem.stem(wd))

        tweets_stem.append(filter_wd)

      print_tweets(tweets_stem, 5)

      ['father', 'dysfunct', 'selfish', 'drag', 'kid', 'dysfunct', '#run']
      ['thank', '#lyft', 'credit', "can't", 'use', 'caus', 'offer', 'wheelchair', 'van', 'pdx', '#disapoint', '#getthank']
      ['bihday', 'majesti']
      ['#model', 'love', 'take', 'time', 'urd', '+–']
      ['factsguid', 'societi', '#motiv']
```

```
[24]: words_cloud = WordCloud().generate(all_words_non_hate)
print('Word Cloud: Non hate words')
plt.figure(figsize= (12,8))
plt.imshow(words_cloud, interpolation = 'bilinear',cmap='viridis')
plt.axis('off')
```

```
Word Cloud: Non hate words
[24]: (-0.5, 399.5, 199.5, -0.5)
```

5. Data formatting for predictive modelling:
    1. Perform train_test_split using sklearn.
6. We'll use TF-IDF values for the terms as a feature to get into a vector space model.
    1. Instantiate with a maximum of 5000 terms in your vocabulary.
    2. Fit and apply on the train set.
    3. Apply on the test set.
7. 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.

8. 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.

```
[29]: logReg = LogisticRegression(random_state=99)

      logReg.fit(x_train_vect, y_train)

[29]: LogisticRegression(random_state=99)

[30]: lr_pred_train = logReg.predict(x_train_vect)
      lr_pred_test = logReg.predict(x_test_vect)

[31]: print('Test set')
      print('Accuracy score: ', accuracy_score(y_test, lr_pred_test))
      print('Recall Score: ', recall_score(y_test, lr_pred_test))
      print('F1 score: ', f1_score(y_test, lr_pred_test))
      print('Confusion Matrix : \n', confusion_matrix(y_test, lr_pred_test))

      Test set
      Accuracy score:  0.9524455104807592
      Recall Score:  0.3302325581395349
      F1 score:  0.48299319727891155
      Confusion Matrix :
       [[8920   24]
       [ 432  213]]
```

9. 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.
10. 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.

```
[32]: logReg = LogisticRegression(class_weight='balanced', random_state=99)

      logReg.fit(x_train_vect, y_train)

      lr_pred_train = logReg.predict(x_train_vect)
      lr_pred_test = logReg.predict(x_test_vect)

[33]: print('Test set')
      print('Accuracy score: ', accuracy_score(y_test, lr_pred_test))
      print('Recall Score: ', recall_score(y_test, lr_pred_test))
      print('F1 score: ', f1_score(y_test, lr_pred_test))
      print('Confusion Matrix : \n', confusion_matrix(y_test, lr_pred_test))

      Test set
      Accuracy score:  0.9296068411721764
      Recall Score:  0.7891472868217054
      F1 score:  0.6012994683992912
      Confusion Matrix :
       [[8405  539]
       [ 136  509]]
```

11. 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.
12. 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.

```
[38]: GridSearchCV(cv=StratifiedKFold(n_splits=4, random_state=99, shuffle=True),
                   estimator=LogisticRegression(class_weight='balanced',
                                                random_state=99),
                   param_grid={'C': [100, 10, 1.0, 0.1, 0.01],
                               'penalty': ['l1', 'l2'],
                               'solver': ['liblinear', 'newton-cg', 'sag']},
                   scoring='recall', verbose=1)

[39]: grid_search.best_estimator_

[39]: LogisticRegression(class_weight='balanced', random_state=99, solver='liblinear')

[40]: grid_search.best_score_

[40]: 0.7802067669172932

[41]: grid_search.best_params_

[41]: {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}

[42]: gs_y_pred = grid_search.best_estimator_.predict(x_test_vect)
```

Results

```
[42]: gs_y_pred = grid_search.best_estimator_.predict(x_test_vect)

[43]: print('Test set')
      print('Accuracy score: ', accuracy_score(y_test, gs_y_pred))
      print('Recall Score: ', recall_score(y_test, gs_y_pred))
      print('F1 score: ', f1_score(y_test, gs_y_pred))
      print('Confusion Matrix : \n', confusion_matrix(y_test, gs_y_pred))

      Test set
      Accuracy score:  0.92950255501095
      Recall Score:  0.7891472868217054
      F1 score:  0.6009445100354192
      Confusion Matrix :
       [[8404  540]
       [ 136  509]]
```