Vasquez-Ed_Assignment5

August 20, 2021

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pprint
import sklearn.naive_bayes as nb
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from scipy.stats import norm
import re
from nltk.corpus import stopwords
import string
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
```

WARNING: Offensive Language in the Data

0.1 Multinomial Naive Bayes vs Offensive Tweets

I'll use Multinomial Naive Bayes to help classify this dataset's tweets into either hate speech, simply offensive language, or neither. ### Attributes For simplicity's sake, I'm going to drop all columns except "class" and "tweet." This dataset originally had people manually read through the tweet text, and then vote on whether the tweet was offensive, hate speech, or neither. Hopefully Naive Bayes and NLTK can do just as well! - Levels for **class**: 1. 0 = Hate Speech 2. 1 = Offensive Language 3. 2 = Neither

So, our classifier needs to discern between 3 distinct classes.

[6]:
```
hate = pd.read_csv("hate_speech.csv")
hate.drop(columns = ['Unnamed: 0', 'count', 'hate_speech', 'neither',␣
 ↪'offensive_language'], inplace = True)
```

[7]:
```
hate
```

```
[7]:        class                                              tweet
     0          2  !!! RT @mayasolovely: As a woman you shouldn't...
     1          1  !!!!! RT @mleew17: boy dats cold...tyga dwn ba...
     2          1  !!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
     3          1  !!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
     4          1  !!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...
     ...      ...                                                ...
     24778      1  you's a muthaf***in lie &#8220;@LifeAsKing: @2...
     24779      2  you've gone and broke the wrong heart baby, an...
     24780      1  young buck wanna eat!!.. dat nigguh like I ain...
     24781      1              youu got wild bitches tellin you lies
     24782      2  ~~Ruffled | Ntac Eileen Dahlia - Beautiful col...

     [24783 rows x 2 columns]
```

0.1.1 Sample of uncleaned tweets

```
[8]: hate.iloc[215:220]['tweet']
```

```
[8]: 215    "@ProdsJewel_: &#8220;@sreadthepeace: &#8220;@...
     216    "@QUAN1TO: 61% of welfare/government aid is cl...
     217    "@QUAN1TO: These bitches don't care they just ...
     218    "@Queen_Kaaat: It took a while for you to find...
     219    "@RTNBA: Drakes new shoes that will be release...
     Name: tweet, dtype: object
```

0.1.2 Replace @user from tweets, and http links

```
[9]: hate['tweet'] = hate['tweet'].replace('@\w*', '', regex = True)
     hate.iloc[215:220]['tweet']
```

```
[9]: 215    ": &#8220;: &#8220;:  Prince, THIS is art. htt...
     216    ": 61% of welfare/government aid is claimed by...
     217    ": These bitches don't care they just play tha...
     218    ": It took a while for you to find me, but I w...
     219    ": Drakes new shoes that will be released by N...
     Name: tweet, dtype: object
```

0.1.3 Cleaning the text

I ran the text through the same cleaning function I made before, which essentially removes punctuation and tokenizes words.

```
[14]: cleaned_strings = []
      for i in range(len(hate.values)):
          cleaned_strings.append(' '.join(clean(hate.iloc[i].tweet)))
```

```
[15]: hate['tweet'] = cleaned_strings
```

0.1.4 Cleaned samples (well, cleaner)

```
[16]: hate.iloc[215:220]['tweet']
```

```
[16]: 215                    prince art http nobody takin bitch
      216          aid claimed white people black slander trash
      217                                 bitches care play role
      218     took find hiding lime tree dis bitch da wrong …
      219     drakes new shoes released yes glitter shoes ht…
      Name: tweet, dtype: object
```

0.2 Simple Model

0.2.1 tfid transforming, train/test split

```
[22]: X = tfid.fit_transform(hate.tweet)
      y = hate['class']
      X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42,␣
       ↪test_size = .3)
```

0.2.2 Untuned model

```
[23]: mnb.fit(X_train, y_train)
      print("Accuracy:", mnb.score(X_test, y_test))
```

```
Accuracy: 0.7994620040349697
```

0.2.3 Tuning alpha and ngram ranges

For the same reasons as before, I could not run this through a pipeline. This for loop accomplishes the same thing, more or less.

```
[24]: ngram_ranges = [(1,1), (1,2), (2, 2), (2, 3), (3,3)]
      alpha = [.4, .6, .8, 1.0]
      scores = []
      for i in ngram_ranges:
          tfid = TfidfVectorizer(ngram_range = i)
          X = tfid.fit_transform(hate.tweet)
          for j in alpha:
              score = cross_val_score(nb.MultinomialNB(alpha = j), X, y, cv = 10)
              scores.append({"Accuracy" : np.mean(score),
                             "ngram_range" : i,
                             "nb_alpha" : j
                            })
      scores
```

[24]: [{'Accuracy': 0.8375508264579855, 'ngram_range': (1, 1), 'nb_alpha': 0.4},
 {'Accuracy': 0.826777424310943, 'ngram_range': (1, 1), 'nb_alpha': 0.6},
 {'Accuracy': 0.8168512518879328, 'ngram_range': (1, 1), 'nb_alpha': 0.8},
 {'Accuracy': 0.8088618650090948, 'ngram_range': (1, 1), 'nb_alpha': 1.0},
 {'Accuracy': 0.8224604840466212, 'ngram_range': (1, 2), 'nb_alpha': 0.4},
 {'Accuracy': 0.8010745630528074, 'ngram_range': (1, 2), 'nb_alpha': 0.6},
 {'Accuracy': 0.7891708755483104, 'ngram_range': (1, 2), 'nb_alpha': 0.8},
 {'Accuracy': 0.7835619201290844, 'ngram_range': (1, 2), 'nb_alpha': 1.0},
 {'Accuracy': 0.8135433362602601, 'ngram_range': (2, 2), 'nb_alpha': 0.4},
 {'Accuracy': 0.7907446765908694, 'ngram_range': (2, 2), 'nb_alpha': 0.6},
 {'Accuracy': 0.7812214693823598, 'ngram_range': (2, 2), 'nb_alpha': 0.8},
 {'Accuracy': 0.77775115652677, 'ngram_range': (2, 2), 'nb_alpha': 1.0},
 {'Accuracy': 0.8066836487023685, 'ngram_range': (2, 3), 'nb_alpha': 0.4},
 {'Accuracy': 0.782512865943172, 'ngram_range': (2, 3), 'nb_alpha': 0.6},
 {'Accuracy': 0.7781142712587185, 'ngram_range': (2, 3), 'nb_alpha': 0.8},
 {'Accuracy': 0.7766212455815289, 'ngram_range': (2, 3), 'nb_alpha': 1.0},
 {'Accuracy': 0.7790021165685219, 'ngram_range': (3, 3), 'nb_alpha': 0.4},
 {'Accuracy': 0.777065103121263, 'ngram_range': (3, 3), 'nb_alpha': 0.6},
 {'Accuracy': 0.7758545144834039, 'ngram_range': (3, 3), 'nb_alpha': 0.8},
 {'Accuracy': 0.775047428260178, 'ngram_range': (3, 3), 'nb_alpha': 1.0}]

0.3 Final Model

According to the cv search, the best parameters look to be ngram_range set to (1,1) and the smoothing parameter (alpha) for the MultinomialNB set to .4. Let's see how it looks for our test set now:

[25]:
```
tfid = TfidfVectorizer(ngram_range = (1,1))
X = tfid.fit_transform(hate.tweet)
y = hate['class']
# split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42,␣
↪test_size = .3)

# fit
mnb = nb.MultinomialNB(alpha = .4)
mnb.fit(X_train, y_train)
# score
print("Accuracy:", mnb.score(X_test, y_test))
```

Accuracy: 0.8314727639542704

[36]:
```
```

```
  (0, 17520)	0.2193415827357477
  (0, 16712)	0.291102930020111
  (0, 401)	0.29279230041783566
  (0, 10335)	0.2599423583435001
```

```
(0, 455)      0.23567602704432328
(0, 8047)     0.335009226866536
(0, 3130)     0.48977264047592894
(0, 3397)     0.41387288873118766
(0, 19033)    0.34724102010901836
(0, 14403)    0.1197175347666778
```

0.4 Results

Tuning the model yielding roughly a 6% increase in accuracy, however, surely more work could be done on the text cleaning. Regardless, fairly promising results for minimal effort. Sure beats manual classification! I sure as hell wouldn't want to read even a fraction of the tweets in this dataset.