In [1]:
```
!pip install vaderSentiment
!pip install textstat
```

```
Requirement already satisfied: vaderSentiment in /usr/local/lib/python3.7/dis
t-packages (3.3.2)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-pack
ages (from vaderSentiment) (2.23.0)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
packages (from requests->vaderSentiment) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /us
r/local/lib/python3.7/dist-packages (from requests->vaderSentiment) (1.24.3)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/
dist-packages (from requests->vaderSentiment) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.
7/dist-packages (from requests->vaderSentiment) (2020.12.5)
Requirement already satisfied: textstat in /usr/local/lib/python3.7/dist-pack
ages (0.7.0)
Requirement already satisfied: pyphen in /usr/local/lib/python3.7/dist-packag
es (from textstat) (0.10.0)
```

In [2]:
```
import pandas as pd
import numpy as np
import pickle
import sys
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
from nltk.stem.porter import *
import string
import re
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer as VS
from textstat.textstat import *
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import classification_report
from sklearn.svm import LinearSVC
import matplotlib.pyplot as plt
import seaborn
%matplotlib inline
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

In [3]:
```
import io
df = pd.read_csv('/content/labeled_data.csv')
```

In [4]:
```
df
```

Out[4]:

	Unnamed: 0	count	hate_speech	offensive_language	neither	class	tweet
0	0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...

	Unnamed: 0	count	hate_speech	offensive_language	neither	class	tweet
1	1	3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2	2	3	0	3	0	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3	3	3	0	2	1	1	!!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4	4	6	0	6	0	1	!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...
...	...	...	...	...	...	...	...
19966	20407	3	0	3	0	1	RT @sangelina_xo: what's with guys and fat bit...
19967	20408	3	0	3	0	1	RT @santos_brina: I could never mess around wi...
19968	20409	3	2	1	0	0	RT @saramariewelch: Been my main nigguh since ...
19969	20410	3	0	3	0	1	RT @saramariewelch: Can't mean something to so...
19970	20411	3	0	3	0	1	RT @saraschaefer1: Can you get eye cancer from...

19971 rows × 7 columns

In [5]:
```
df.describe()
```

Out[5]:

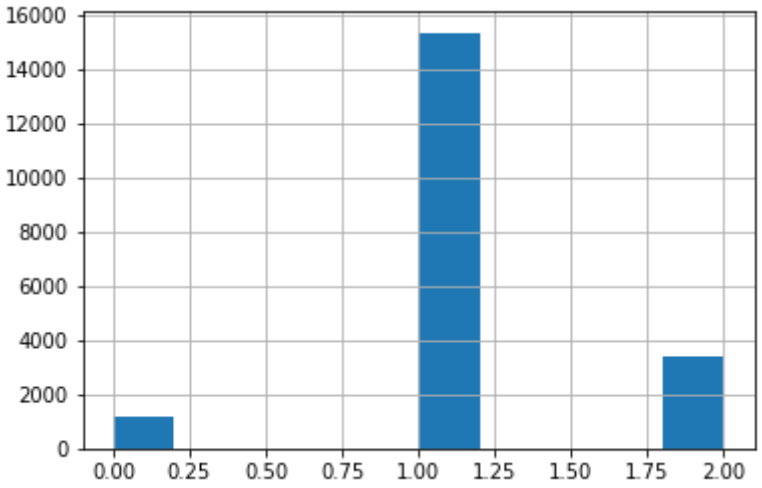
	Unnamed: 0	count	hate_speech	offensive_language	neither	class
count	19971.000000	19971.000000	19971.000000	19971.000000	19971.000000	19971.000000
mean	10230.831856	3.238796	0.285314	2.397176	0.556307	1.110160
std	5892.278820	0.876505	0.641819	1.395421	1.121989	0.467344
min	0.000000	3.000000	0.000000	0.000000	0.000000	0.000000
25%	5139.500000	3.000000	0.000000	2.000000	0.000000	1.000000
50%	10257.000000	3.000000	0.000000	3.000000	0.000000	1.000000
75%	15333.500000	3.000000	0.000000	3.000000	0.000000	1.000000
max	20411.000000	9.000000	7.000000	9.000000	9.000000	2.000000

In [6]:
```
df.columns
```

Out[6]:
```
Index(['Unnamed: 0', 'count', 'hate_speech', 'offensive_language', 'neither',
       'class', 'tweet'],
      dtype='object')
```

In [7]:
```
df['class'].hist()
```

Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f40422660d0>
```



In [8]:
```
tweets=df.tweet
```

In [9]:
```
stopwords=stopwords = nltk.corpus.stopwords.words("english")

other_exclusions = ["#ff", "ff", "rt"]
stopwords.extend(other_exclusions)

stemmer = PorterStemmer()


def preprocess(text_string):
    """
    Accepts a text string and replaces:
    1) urls with URLHERE
    2) lots of whitespace with one instance
    3) mentions with MENTIONHERE

    This allows us to get standardized counts of urls and mentions
    Without caring about specific people mentioned
    """
    space_pattern = '\s+'
    giant_url_regex = ('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|'
        '[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+')
    mention_regex = '@[\w\-]+'
    parsed_text = re.sub(space_pattern, ' ', text_string)
    parsed_text = re.sub(giant_url_regex, '', parsed_text)
    parsed_text = re.sub(mention_regex, '', parsed_text)
    return parsed_text

def tokenize(tweet):
    """Removes punctuation & excess whitespace, sets to lowercase,
    and stems tweets. Returns a list of stemmed tokens."""
    tweet = " ".join(re.split("[^a-zA-Z]*", tweet.lower())).strip()
    tokens = [stemmer.stem(t) for t in tweet.split()]
    return tokens

def basic_tokenize(tweet):
    """Same as tokenize but without the stemming"""
```

```
    tweet = " ".join(re.split("[^a-zA-Z.,!?]*", tweet.lower())).strip()
    return tweet.split()

vectorizer = TfidfVectorizer(
    tokenizer=tokenize,
    preprocessor=preprocess,
    ngram_range=(1, 3),
    stop_words=stopwords,
    use_idf=True,
    smooth_idf=False,
    norm=None,
    decode_error='replace',
    max_features=10000,
    min_df=5,
    max_df=0.75
    )
```

In [10]:

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [11]:

```
#Construct tfidf matrix and get relevant scores
tfidf = vectorizer.fit_transform(tweets).toarray()
vocab = {v:i for i, v in enumerate(vectorizer.get_feature_names())}
idf_vals = vectorizer.idf_
idf_dict = {i:idf_vals[i] for i in vocab.values()} #keys are indices; values
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/feature_extraction/text.py:38
5: UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['b', 'c', 'e', 'f', 'g', 'h',
'j', 'l', 'n', 'p', 'r', 'u', 'v', 'w'] not in stop_words.
  'stop_words.' % sorted(inconsistent))
```

In [12]:

```
#Get POS tags for tweets and save as a string
tweet_tags = []
for t in tweets:
    tokens = basic_tokenize(preprocess(t))
    tags = nltk.pos_tag(tokens)
    tag_list = [x[1] for x in tags]
    tag_str = " ".join(tag_list)
    tweet_tags.append(tag_str)
```

In [13]:

```
#We can use the TFIDF vectorizer to get a token matrix for the POS tags
pos_vectorizer = TfidfVectorizer(
    tokenizer=None,
    lowercase=False,
    preprocessor=None,
    ngram_range=(1, 3),
    stop_words=None,
    use_idf=False,
    smooth_idf=False,
    norm=None,
    decode_error='replace',
    max_features=5000,
    min_df=5,
    max_df=0.75,
    )
```

In [14]:

```
#Construct POS TF matrix and get vocab dict
pos = pos_vectorizer.fit_transform(pd.Series(tweet_tags)).toarray()
```

```
pos_vocab = {v:i for i, v in enumerate(pos_vectorizer.get_feature_names())}
```

In [15]:

```
#Now get other features
sentiment_analyzer = VS()

def count_twitter_objs(text_string):
    """
    Accepts a text string and replaces:
    1) urls with URLHERE
    2) lots of whitespace with one instance
    3) mentions with MENTIONHERE
    4) hashtags with HASHTAGHERE

    This allows us to get standardized counts of urls and mentions
    Without caring about specific people mentioned.

    Returns counts of urls, mentions, and hashtags.
    """
    space_pattern = '\s+'
    giant_url_regex = ('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|'
        '[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+')
    mention_regex = '@[\w\-]+'
    hashtag_regex = '#[\w\-]+'
    parsed_text = re.sub(space_pattern, ' ', text_string)
    parsed_text = re.sub(giant_url_regex, 'URLHERE', parsed_text)
    parsed_text = re.sub(mention_regex, 'MENTIONHERE', parsed_text)
    parsed_text = re.sub(hashtag_regex, 'HASHTAGHERE', parsed_text)
    return(parsed_text.count('URLHERE'),parsed_text.count('MENTIONHERE'),pars

def other_features(tweet):
    """This function takes a string and returns a list of features.
    These include Sentiment scores, Text and Readability scores,
    as well as Twitter specific features"""
    sentiment = sentiment_analyzer.polarity_scores(tweet)

    words = preprocess(tweet) #Get text only

    syllables = textstat.syllable_count(words)
    num_chars = sum(len(w) for w in words)
    num_chars_total = len(tweet)
    num_terms = len(tweet.split())
    num_words = len(words.split())
    avg_syl = round(float((syllables+0.001))/float(num_words+0.001),4)
    num_unique_terms = len(set(words.split()))

    ###Modified FK grade, where avg words per sentence is just num words/1
    FKRA = round(float(0.39 * float(num_words)/1.0) + float(11.8 * avg_syl) -
    ##Modified FRE score, where sentence fixed to 1
    FRE = round(206.835 - 1.015*(float(num_words)/1.0) - (84.6*float(avg_syl)

    twitter_objs = count_twitter_objs(tweet)
    retweet = 0
    if "rt" in words:
        retweet = 1
    features = [FKRA, FRE,syllables, avg_syl, num_chars, num_chars_total, num
                num_unique_terms, sentiment['neg'], sentiment['pos'], sentime
                twitter_objs[2], twitter_objs[1],
                twitter_objs[0], retweet]
    #features = pandas.DataFrame(features)
    return features

def get_feature_array(tweets):
    feats=[]
    for t in tweets:
```

```
        feats.append(other_features(t))
    return np.array(feats)
```

In [16]:
```
other_features_names = ["FKRA", "FRE","num_syllables", "avg_syl_per_word", "r
                        "num_terms", "num_words", "num_unique_words", "vader
                        "vader compound", "num_hashtags", "num_mentions", "nu
```

In [17]:
```
feats = get_feature_array(tweets)
```

In [18]:
```
#Now join them all up
M = np.concatenate([tfidf,pos,feats],axis=1)
```

In [19]:
```
M.shape
```

Out[19]: (19971, 3853)

In [20]:
```
#Finally get a list of variable names
variables = ['']*len(vocab)
for k,v in vocab.items():
    variables[v] = k

pos_variables = ['']*len(pos_vocab)
for k,v in pos_vocab.items():
    pos_variables[v] = k

feature_names = variables+pos_variables+other_features_names
```

In [21]:
```
X = pd.DataFrame(M)
y = df['class'].astype(int)
```

In [22]:
```
from sklearn.model_selection import train_test_split
```

In [23]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, te
```

In [24]:
```
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.pipeline import Pipeline
```

In [25]:
```
pipe = Pipeline(
        [('select', SelectFromModel(LogisticRegression(class_weight='balanced
                                                       penalty="l1", C=0.01, solve
        ('model', LogisticRegression(class_weight='balanced',penalty='l2', sc
```

In [26]:
```
param_grid = [{}] # Optionally add parameters here
```

In [27]:
```
grid_search = GridSearchCV(pipe,
                           param_grid,
                           cv=StratifiedKFold(n_splits=5,
```

```
                                                random_state=42).split(X_train,
                            verbose=2)
```

In [28]:
```
model = grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV]  ................................................................
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
rs.
[CV] ................................................. , total=   7.4s
[CV]  ................................................................
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    7.4s remaining:    0.
0s
[CV] ................................................. , total=   6.0s
[CV]  ................................................................
[CV] ................................................. , total=   5.9s
[CV]  ................................................................
[CV] ................................................. , total=   6.8s
[CV]  ................................................................
[CV] ................................................. , total=   4.8s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   30.8s finished
```

In [29]:
```
y_preds = model.predict(X_test)
print(X_test)
```

```
           0         1     2        3     4     ...    3848  3849  3850  3851
3852
15930  3.875943  0.000000   0.0  0.00000   0.0  ... -0.2598   0.0   1.0   0.0
0.0
3213   1.291981  0.000000   0.0  0.00000   0.0  ... -0.8658   5.0   1.0   0.0
0.0
18924  1.291981  0.000000   0.0  0.00000   0.0  ...  0.3612   0.0   2.0   0.0
0.0
9564   2.583962  3.918247   0.0  4.68943   0.0  ... -0.8074   0.0   0.0   0.0
0.0
16570  5.167923  0.000000   0.0  0.00000   0.0  ...  0.0129  10.0   1.0   0.0
0.0
...         ...       ...   ...      ...   ...  ...     ...   ...   ...   ...
...
14044  1.291981  0.000000   0.0  0.00000   0.0  ...  0.0754   2.0   2.0   1.0
0.0
11073  1.291981  0.000000   0.0  0.00000   0.0  ...  0.1531   0.0   0.0   0.0
1.0
8623   2.583962  0.000000   0.0  0.00000   0.0  ...  0.0577   0.0   0.0   0.0
1.0
13108  2.583962  0.000000   0.0  0.00000   0.0  ...  0.7500   0.0   0.0   0.0
0.0
11178  1.291981  0.000000   0.0  0.00000   0.0  ... -0.5423   0.0   0.0   0.0
0.0

[1998 rows x 3853 columns]
```

In [30]:
```
report = classification_report( y_test, y_preds )
```

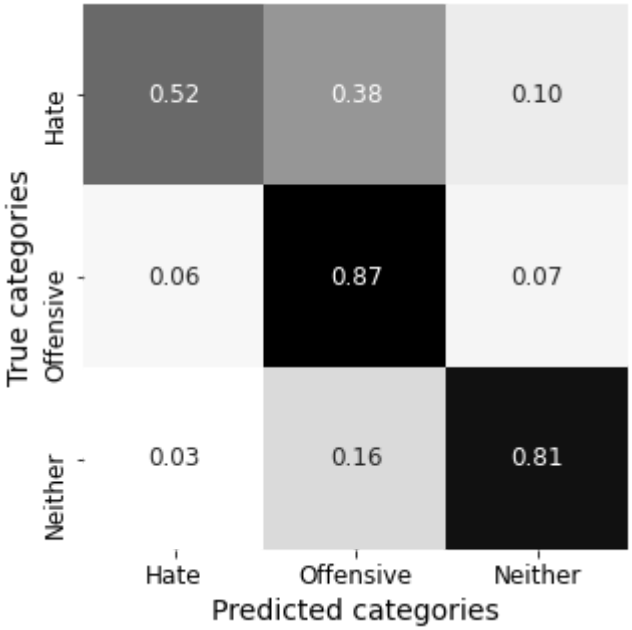
In [31]:
```
print(report)
```

```
              precision    recall  f1-score   support

           0       0.38      0.52      0.44       122
           1       0.93      0.87      0.90      1522
           2       0.71      0.81      0.76       354

    accuracy                           0.84      1998
```

```
   macro avg       0.67      0.73      0.70      1998
weighted avg       0.86      0.84      0.85      1998
```

In [32]:

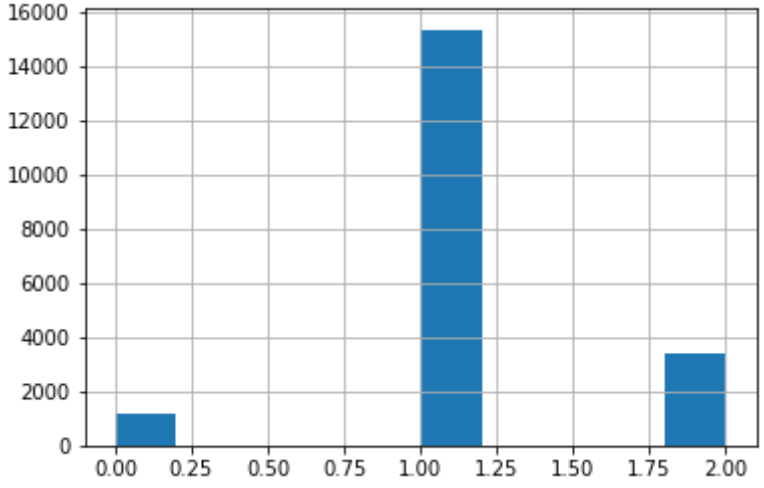
```
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test,y_preds)
matrix_proportions = np.zeros((3,3))
for i in range(0,3):
    matrix_proportions[i,:] = confusion_matrix[i,:]/float(confusion_matrix[i,
names=['Hate','Offensive','Neither']
confusion_df = pd.DataFrame(matrix_proportions, index=names,columns=names)
plt.figure(figsize=(5,5))
seaborn.heatmap(confusion_df,annot=True,annot_kws={"size": 12},cmap='gist_gra
plt.ylabel(r'True categories',fontsize=14)
plt.xlabel(r'Predicted categories',fontsize=14)
plt.tick_params(labelsize=12)

#Uncomment line below if you want to save the output
#plt.savefig('confusion.pdf')
```



In [33]:

```
#True distribution
y.hist()
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x7f403ee117d0>



In [34]:
```
pd.Series(y_preds).hist()
```

Out[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f403eaa1cd0>
```