Timothy Compton
Project: Natural Language Processing with Python (Jupyter Notebooks)

Detection and Analysis of Toxic Comments: Wikipedia

Business Scenario:

Wikipedia, a reference website, relies upon millions of community contributors to maintain and update information found on the website. Wikipedia works hard to help community interaction during these tasks to be productive and respectful. In order to achieve this on such a large scale, a predictive model must be made that will flag toxic comments for review and cleanup.

Objectives:

Create a model which uses Natural Language Processing and Machine Learning to identify toxic comments among thousands of conversations regarding page updates. After identification, analyze which words are found most frequently in the toxic comments.

Summary of Data used for Analysis:

The data provided for analysis contains 3 columns and 5000 rows. The columns are id (comment identification number), comment_text (full comment), and toxic (binary identifier, toxic=1, not toxic=0).

Solutions Executive Summary:

The machine learning model used for toxic comment classification employed the Support Vector Machines (SVM) algorithm. The algorithm was most effective in classifying comment toxicity after a number of algorithms were used to organize and optimize the training data presented to the SVM algorithm. These included TfidfVectorizer, GridSearchCV, and StratifiedKFold. The primary metric used to measure success of the model was Recall, because recall focuses on how well the model can identify what it is looking for among the test data. The optimized machine learning model developed received a recall score of 64%.

The words most commonly found in the comments identified as toxic were mostly obscenities. These can be found in the “Solutions Detailed Summary”. Certain words such as “go”, “hey”, “get”, and “think” were also some of the more common words used in toxic comments.

Solutions Detailed Summary:

In [2]:
```
import pandas as pd
import numpy as np
import re
import nltk
import string
import itertools
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
from sklearn import svm
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold

stop_words=stopwords.words('english')
punc = string.punctuation
punc = list(punc)
punc.extend(['``',"''"])

remove_list=punc+stop_words
```

In [3]:
```
df = pd.read_csv("train.csv")
```

In [4]:
```
df.shape
```

Out[4]:
```
(5000, 3)
```

We first analyze the provided “train” data provided, and preprocess/clean the data to prepare for the machine learning algorithms.

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

Out[5]:

	id	comment_text	toxic
0	e617e2489abe9bca	"\r\n\r\n A barnstar for you! \r\n\r\n The De...	0
1	9250cf637294e09d	"\r\n\r\nThis seems unbalanced. whatever I ha...	0
2	ce1aa4592d5240ca	Marya Dzmitruk was born in Minsk, Belarus in M...	0
3	48105766ff7f075b	"\r\n\r\nTalkback\r\n\r\n Dear Celestia... "	0
4	0543d4f82e5470b6	New Categories \r\n\r\nI honestly think that w...	0

In [6]:
```
text = df["comment_text"].tolist()
text
```

```
 "I really really struggle to cope with users like logan and I don't think he was right",
 'Its not correct to blame STiki for this, actually',
 '"\r\n\r\nGreetings & question\r\nHi, Oda Mari. Long time no see. I hope all is well with you and that yo
u\'re having fun editing. I\'ve stopped editing here and moved over to a Japanese film-specific wiki. (There
are several reasons for my departure. The main one is that I believe, Wikipedia\'s ""notability"" criteria m
akes all non-English, non-mainstream film articles easy targets for deletion, unless they have been extensiv
ely written about in English.) Unfortunately, so far, the editors only consist of a native English-speaker
(myself), and (I believe) a native Czech-speaker. So, with no native Japanese-speaker, we sometimes have to
wrestle with the Japanese language a bit ;-) A question has come up at: \r\n\r\nhttp://eiga.wikia.com/wiki/T
alk:Humanity_and_Paper_Balloons#Name_of_the_seller_of_goldfish\r\n\r\nWould you be kind enough to offer inpu
t? Best regards, and take care. (I hope you might consider looking in on Eigapedia from time to time also. I
am ""Tokkan Kozo"", and I do not limit my work there to ""Pink"" I work on all the colors of the rainbow ;-)
"',
 '"\r\n\r\n Blocked for another 6 months \r\n\r\n If you wish to contribute to this encyclopedia, please cre
ate an account  and log inDue to persistent vandalism (), editing by anonymous users from your school, libra
ry or institution\'s IP address  currently disabled. You continue to have full access to read the encycloped
ia. If you are logged in but still unable to edit, please follow these instructions. To prevent abuse, accou
nt creation via this IP address is probably also disabled.If accounts need to be created at school for class
projects, please have your teacher or network administrator contact us (with reference to this IP address) a
```

In [7]:
```
len(text)
```

Out[7]: 5000

In [8]:
```
def preprocess(clean):
    clean = re.sub('\d*\.\d*\.\d*\.\d*', '', clean)
    clean = re.sub(r'(https?://\S+)', '', clean)
    clean = re.sub(r'\n', '', clean)
    clean = re.sub(r'\r','',clean)
    word_tokens = word_tokenize(clean.lower())
    words_list=[]
    for word in word_tokens:
        if word not in remove_list:
            words_list.append(word)

    #filtered_text = ' '.join(words_list)


    return words_list #filtered_text
```

In [9]:
```
new=[]
```

In [10]:
```
for i in text:
    x = preprocess(i)
    new.append(x)
new
```

Out[10]:
```
[['barnstar',
  'defender',
  'wiki',
  'barnstar',
  'like',
  'edit',
  'kayastha',
  'page',
  'lets',
  'form',
  'solidarity',
  'group',
  'malign',
  'article',
  'subject',
  'matter',
  'propose',
  'folloing',
  'name',
```

In [11]:
```
len(new)
```

Out[11]: 5000

In [12]:
```
count1 = 0
for listElem in new:
    count1 += len(listElem)

count1
```

Out[12]: 178019

In [13]:
```
#4. Using a Counter, Find the Top Terms in the Data. Remove Contextual Stop Words.
```

In [14]:
```
words = list(itertools.chain.from_iterable(new))
words
```

Out[14]:
```
['barnstar',
 'defender',
 'wiki',
 'barnstar',
 'like',
 'edit',
 'kayastha',
 'page',
 'lets',
 'form',
 'solidarity',
 'group',
 'malign',
 'article',
 'subject',
 'matter',
 'propose',
 'folloing',
 'name',
```

In [15]:
```
len(words)
```

Out[15]: 178019

In [16]:
```
freqTable={}

for word in words:
    if word in freqTable:
        freqTable[word] += 1

    else:
        freqTable[word] = 1
```

In [17]:
```
{k: v for k, v in sorted(freqTable.items(),reverse=True, key=lambda item: item[1])}
```

Out[17]:
```
{"'s": 1988,
 "n't": 1795,
 'article': 1673,
 'page': 1450,
 'wikipedia': 1388,
 'talk': 1167,
 'would': 1004,
 'please': 1003,
 'ass': 985,
 'fuck': 902,
 'one': 853,
 'like': 832,
 'also': 640,
 'think': 628,
 'see': 626,
 'know': 594,
 'edit': 557,
 'people': 548,
 'use': 539,
```

In [18]:
```
#Top Meaningless symbols and Contextual Stop Words to Remove:
# "'s", "n't", "article", "page", "wikipedia", "edit", "'m", "articles", "'re", "'ve", "editing", "edits", "page
```

In [19]:
```
#Update Frequency Table:
```

In [20]:
```
remove_list2 = (["'s", "n't", "article", "page", "wikipedia", "edit", "'m", "articles", "'re", "'ve", "editing",
```

In [21]:
```
words2=[]
for word in words:
    if word not in remove_list2:
        words2.append(word)
```

In [22]:
```
len(words2)
```

Out[22]:
```
166129
```

In [23]:
```
freqTable2={}

for word in words2:
    if word in freqTable2:
        freqTable2[word] += 1

    else:
        freqTable2[word] = 1
```

In [24]:
```
{k: v for k, v in sorted(freqTable2.items(),reverse=True, key=lambda item: item[1])}
```

Out[24]:
```
{'talk': 1167,
 'would': 1004,
 'please': 1003,
 'ass': 985,
 'fuck': 902,
 'one': 853,
 'like': 832,
 'also': 640,
 'think': 628,
 'see': 626,
 'know': 594,
 'people': 548,
 'use': 539,
 'name': 532,
 'may': 530,
 'time': 470,
 'user': 413,
 'even': 402,
 'could': 392,
```

In [25]:
```
#Next, remove all determined words/symbols from the actual list of reviews:
```

In [26]:
```
def postprocess(cleaner):
    words_list2=[]
    for word in cleaner:
        if word not in remove_list2:
            words_list2.append(word)

    return words_list2
```

In [27]:
```
newer=[]
```

In [28]:
```
for i in new:
    x = postprocess(i)
    newer.append(x)
newer
```

Out[28]:
```
[['barnstar',
  'defender',
  'wiki',
  'barnstar',
  'like',
  'kayastha',
  'lets',
  'form',
  'solidarity',
  'group',
  'malign',
  'subject',
  'matter',
  'propose',
  'folloing',
  'name',
  'group.united',
  'intellectuals',
  'front',
```

In [29]:
```
len(newer)
```

Out[29]:
```
5000
```

In [30]:
```
count2 = 0
for listElem in newer:
    count2 += len(listElem)

count2
```

Out[30]:
```
166129
```

In [31]:
```
#5. Separate into Train and Test Sets
```

In [32]:
```
def condense(hello):
    filtered_text = ' '.join(hello)

    return filtered_text
```

In [33]:
```
comment_text2 = []
```

In [34]:
```
for i in newer:
    x = condense(i)
    comment_text2.append(x)
comment_text2
```

Out[34]:
```
['barnstar defender wiki barnstar like kayastha lets form solidarity group malign subject matter propose fol
loing name group.united intellectuals front kayastha ethinicty racist castist abuse uifkearca',
 "seems unbalanced whatever said mathsci said far extreme unpleasant things mention others much greater freq
uency happy reign 'd like ruth told trying get mathsci pay attention stop uncivil would expect issue request
mathsci intentionally unbalanced whatever reason please let know voluntarily close account move things like
lot contribute way point contributing project editors administrative leave aggressively rude good editor rea
lly deserve people riding ass every time try certain things happily leave hands drama-prone think best ludwi
gs2",
 'marya dzmitruk born minsk belarus march 19 1992. mother olga nikolaevna moroz born baranovichi belarus fat
her born brest belarus second child family parents divorced 1998 soon father remarried two children marya ag
e 4 began gymnastics quit two years later denied medal competition age incorrectly marked turned 6 years old
got admitted music school 4 minsk class violin public school 66 piano classes main course age 11 marya starr
ed belarusfilm movie called " dunechka " soon started play theatre featured television shows 2005 mother dec
ided move united states september 2005 marya went first american school ingrid b. lacy middle school graduat
ed spring 2006 traveled back belarus 2 months august 2006 went oceana high school graduate 2010. marya dzmit
ruk member isar international society astrological research also member non-profit government organization d
eals human rights abuse throughout world also known helsinki committee marya holds two diplomas music school
s four scholarships lisa spector ' music school several awards yli youth leadership institute marya close re
lationship mother personal life " happy could possibly get " source says currently dating alex k. odessa ukr
```

In [35]:
```
len(comment_text2)
```

Out[35]:
```
5000
```

In [36]:
```
df = df.assign(comment_text2 = comment_text2)
```

In [37]:
```
df = df[['id', 'comment_text', 'comment_text2', 'toxic']]
```

In [38]:
```
df
```

Out[38]:

	id	comment_text	comment_text2	toxic
0	e617e2489abe9bca	"\r\n\r\n A barnstar for you! \r\n\r\n The De...	barnstar defender wiki barnstar like kayastha ...	0
1	9250cf637294e09d	"\r\n\r\nThis seems unbalanced. whatever I ha...	seems unbalanced whatever said mathsci said fa...	0
2	ce1aa4592d5240ca	Marya Dzmitruk was born in Minsk, Belarus in M...	marya dzmitruk born minsk belarus march 19 199...	0
3	48105766ff7f075b	"\r\n\r\nTalkback\r\n\r\n Dear Celestia... "	talkback dear celestia	0
4	0543d4f82e5470b6	New Categories \r\n\r\nI honestly think that w...	new categories honestly think need add categor...	0
...	...	...	...	...
4995	60229df7b48ba6ff	"\r\n\r\n Dildo, if you read my response corre...	dildo read response correctly never said going...	0
4996	36a645227572ec5c	CALM DOWN, CALM DOWN, DON'T GET A BIG DICK	calm calm get big dick	1
4997	6d47fa39945ed6f5	In my opinion Dougweller is using his privileg...	opinion dougweller using privileges poorly per...	0
4998	de2e4c0d38db6e30	The style section has been expanded too. I did...	style section expanded remember placed tag	0
4999	4cda24210a33ac35	ANY ONE THAT IS NOT AGREEMENT WITH YOU OR IS A...	one agreement repulican joe hazelton.it wack m...	0

5000 rows × 4 columns

In [ ]:
```
```

In [39]:
```
#5. Separate into Train (70%) and Test (30%) sets.
```

In [40]:
```
X_train, X_test, y_train, y_test = train_test_split(df['comment_text2'], df['toxic'], test_size=0.3, random_stat
```

```
random_state=101)
```

In [42]:
```
#6. Use TF-IDF values for the terms as feature to get into a vector space model.
```

In [43]:
```
tfidf = TfidfVectorizer(max_features = 4000) #analyzer = 'word', ngram_range = (1,2),
```

In [44]:
```
X_train_fit = tfidf.fit_transform(X_train)
X_train_fit
```

Out[44]:
```
<3500x4000 sparse matrix of type '<class 'numpy.float64'>'
        with 72927 stored elements in Compressed Sparse Row format>
```

In [45]:
```
X_train_fit_df = pd.DataFrame(X_train_fit.todense(),columns=tfidf.get_feature_names())
X_train_fit_df.head()
```

Out[45]:

	00	000	0000z	01	02	03	04	0422	05	06	...	zach	zero	zheng	maj	jул	jули	jун	jуни	сеп	сто
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 4000 columns

In [46]:
```
X_test_fit = tfidf.transform(X_test)
X_test_fit
```

Out[46]:
```
<1500x4000 sparse matrix of type '<class 'numpy.float64'>'
        with 31453 stored elements in Compressed Sparse Row format>
```

In [47]:
```
X_test_fit_df = pd.DataFrame(X_test_fit.todense(),columns=tfidf.get_feature_names())
X_test_fit_df
```

Out[47]:

	00	000	0000z	01	02	03	04	0422	05	06	...	zach	zero	zheng	maj	jул	jули	jун	jуни	сеп	сто
0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.433677	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1495	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1496	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1497	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1498	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1499	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

We now review our first model, clf (SVM algorithm). This is a very simple model, which does not optimize any parameters (example, “C”) and receives the train data in a very unbalanced manner. This results in a Recall value of only 41% on the Test set.

```
In [48]: #7. Model Building: Support Vector Machine (SVC)

In [49]: clf = svm.SVC(C=1, kernel='linear')

In [50]: clf.fit(X_train_fit, y_train)
Out[50]: SVC(C=1, kernel='linear')

In [51]: predict_train = clf.predict(X_train_fit)

In [52]: predict_test = clf.predict(X_test_fit)

In [53]: print(clf, '\n', classification_report(y_train, predict_train))
         print(clf, '\n', classification_report(y_test, predict_test))

         SVC(C=1, kernel='linear')
                       precision    recall  f1-score   support

                    0       0.97      1.00      0.98      3188
                    1       1.00      0.63      0.77       312

             accuracy                           0.97      3500
            macro avg       0.98      0.82      0.88      3500
         weighted avg       0.97      0.97      0.96      3500

         SVC(C=1, kernel='linear')
                       precision    recall  f1-score   support

                    0       0.95      1.00      0.97      1375
                    1       0.94      0.41      0.57       125

             accuracy                           0.95      1500
            macro avg       0.95      0.70      0.77      1500
         weighted avg       0.95      0.95      0.94      1500


In [54]: print(len(predict_train), ",", len(predict_test))
         3500 , 1500
```

```
In [55]: #8. Model Evaluation: Accuracy, Recall, and f1_score of the Train set

In [56]: train_accuracy = clf.score(X_train_fit, y_train)

         train_recall = recall_score(y_train, predict_train)

         train_precision = precision_score(y_train, predict_train)
         train_f1score = (2*train_precision*train_recall)/(train_precision+train_recall)


         print("Train Accuracy: ", train_accuracy, "\nTrain Recall: ", train_recall, "\nTrain f1_score", train_f1score)

         Train Accuracy:  0.9671428571428572
         Train Recall:  0.6314102564102564
         Train f1_score 0.7740667976424361

In [57]: #Recall value and f1_score aren't very good...

In [58]: test_accuracy = clf.score(X_test_fit_df, y_test)

         test_recall = recall_score(y_test, predict_test)

         test_precision = precision_score(y_test, predict_test)
         test_f1score = (2*test_precision*test_recall)/(test_precision+test_recall)


         print("Test Accuracy: ", test_accuracy, "\nTest Recall: ", test_recall, "\nTest f1_score", test_f1score)

         Test Accuracy:  0.9486666666666667
         Test Recall:  0.408
         Test f1_score 0.5698324022346368
```

We now attempt a quick fix to clf, presenting clf2 (SVM algorithm). In this case, a simple human guess-and-check of parameter "C" results in use of C=100 (as opposed to clf's C=1). This allows for significant improvement of the Recall score to 58% on the Test set. Still not a very desirable result, however.

```
In [60]: #9. Adjust SVC parameter(s)

In [61]: # Increasing the C value in the svm.SVC() algorithm increases the value for train_accuracy,
         # and GREATLY increases the values of test_recall and test_f1score.
         # After a few guess and checks(C=1,10,100,1000,10000) it looks like C=100 is a good value to move forward with.

In [62]: clf2 = svm.SVC(C=100, kernel='linear')

In [ ]:

In [63]: #10. Train again with the adjustment, and evaluate on Test Set

In [64]: clf2.fit(X_train_fit, y_train)
Out[64]: SVC(C=100, kernel='linear')

In [70]: predict2_train = clf2.predict(X_train_fit)
         predict2_test = clf2.predict(X_test_fit)
```

In [71]:
```
print(clf, '\n', classification_report(y_train, predict2_train))
print(clf, '\n', classification_report(y_test, predict2_test))
```

```
SVC(C=1, kernel='linear')
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3188
           1       1.00      0.98      0.99       312

    accuracy                           1.00      3500
   macro avg       1.00      0.99      1.00      3500
weighted avg       1.00      1.00      1.00      3500

SVC(C=1, kernel='linear')
              precision    recall  f1-score   support

           0       0.96      0.98      0.97      1375
           1       0.71      0.58      0.64       125

    accuracy                           0.95      1500
   macro avg       0.84      0.78      0.81      1500
weighted avg       0.94      0.95      0.94      1500
```

In [72]:
```
train_accuracy2 = clf2.score(X_train_fit, y_train)

train_recall2 = recall_score(y_train, predict2_train)

train_precision2 = precision_score(y_train, predict2_train)
train_f1score2 = (2*train_precision2*train_recall2)/(train_precision2+train_recall2)


print("Train Accuracy2: ", train_accuracy2, "\nTrain Recall2: ", train_recall2, "\nTrain f1_score2", train_f1sco
```

```
Train Accuracy2:  0.9985714285714286
Train Recall2:  0.9839743589743589
Train f1_score2 0.9919224555735056
```

In [73]:
```
test_accuracy2 = clf2.score(X_test_fit, y_test)

test_recall2 = recall_score(y_test, predict2_test)

test_precision2 = precision_score(y_test, predict2_test)
test_f1score2 = (2*test_precision2*test_recall2)/(test_precision2+test_recall2)


print("Test Accuracy2: ", test_accuracy2, "\nTest Recall2: ", test_recall2, "\nTest f1_score2", test_f1score2)
```

```
Test Accuracy2:  0.9453333333333334
Test Recall2:  0.584
Test f1_score2 0.6403508771929824
```

In [74]:
```
# Recall and f1_score are poor, even with the C value update.
# I did verify these values w/ C=1,10,100,1000,10000 and the results do not improve much with differing C values
```

We now turn to a more powerful Machine Learning approach. We look to optimize our “C” parameter by using algorithm GridSearchCV, and in parallel pass a smarter distribution of training data to our new clf3 (SVM) algorithm (folds) using the StratifiedKFold algorithm. This results in the choice of C=50 as the optimal parameter for the clf3 algorithm. Parameter of C=50 is applied in the clf4 (SVM) algorithm, while using StratifiedKFold, to achieve a more acceptable Recall value of 64%.

```
In [75]: #11. Hyperparameter Tuning
```

```
In [76]: EST = svm.SVC(kernel='linear', class_weight='balanced')
         gridvalues = {'C':[1,10,20,30,40,50,60,70,80,90,100,110,120,130,140,150]}
```

```
In [77]: clf3 = GridSearchCV(estimator = EST, param_grid = gridvalues, scoring = 'recall')
         clf3
```

```
Out[77]: GridSearchCV(estimator=SVC(class_weight='balanced', kernel='linear'),
                      param_grid={'C': [1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110,
                                        120, 130, 140, 150]},
                      scoring='recall')
```

```
In [78]: #12. Find the parameters with the best recall in Cross Validation
```

```
In [79]: skf = StratifiedKFold(n_splits=5)
         skf
```

```
Out[79]: StratifiedKFold(n_splits=5, random_state=None, shuffle=False)
```

```
In [80]: y_all = y_train.append(y_test)
         y_all
```

```
Out[80]: 2654    0
         2468    0
         290     0
         1463    1
         4508    0
                ..
         3412    1
         4020    0
         4635    0
         1700    0
         790     1
         Name: toxic, Length: 5000, dtype: int64
```

```
In [81]: y_all_zero = y_all.reset_index(drop=True)
         y_all_zero
```

```
Out[81]: 0       0
         1       0
         2       0
         3       1
         4       0
                ..
         4995    1
         4996    0
         4997    0
         4998    0
         4999    1
         Name: toxic, Length: 5000, dtype: int64
```

In [82]:
```
df5 = X_train_fit_df
df5
```

Out[82]:

	00	000	0000z	01	02	03	04	0422	05	06	...	zach	zero	zheng	маj	јул	јули	јун	јуни	гел	тим
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3495	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3496	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3497	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3498	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3499	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

3500 rows × 4000 columns

In [83]:
```
df5 = df5.append(X_test_fit_df)
df5
```

Out[83]:

	00	000	0000z	01	02	03	04	0422	05	06	...	zach	zero	zheng	маj	јул	јули	јун	јуни	гел	тим
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1495	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1496	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1497	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1498	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1499	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5000 rows × 4000 columns

In [84]:
```
df5 = df5.reset_index(drop=True)
df5
```

Out[84]:

	00	000	0000z	01	02	03	04	0422	05	06	...	zach	zero	zheng	maj	jул	jули	jун	juни	сша	тим
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5000 rows × 4000 columns

In [85]:
```
df5 = df5.assign(toxic = y_all_zero)
df5
```

Out[85]:

	00	000	0000z	01	02	03	04	0422	05	06	...	zero	zheng	maj	jул	jули	jун	juни	сша	тим	toxic
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
4996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1

5000 rows × 4001 columns

In [86]:
```
target = df5.loc[:,'toxic']
```

```
In [240]: def train_model(train5, test5, fold_no):
              #X5 = [:4000]
              y5 = ['toxic']
              X_train5 = train5.iloc[:,:4000] #<- train5 is full data. figure out how to pull all column names except for
              y_train5 = train5[y5]
              X_test5 = test5.iloc[:,:4000]
              y_test5 = test5[y5]
              clf3.fit(X_train5,y_train5)
              predictions = clf3.predict(X_test5)
              print('Fold',str(fold_no),'Recall:',recall_score(y_test5,predictions),'f1_score',f1_score(y_test5,prediction
```

```
In [241]: fold_no = 1
```

```
In [242]: for train_index, test_index in skf.split(df5, target):
              train5 = df5.loc[train_index,:]
              test5 = df5.loc[test_index,:]
              train_model(train5,test5,fold_no) #<- pulling to definition above
              fold_no += 1
```

```
C:\Users\Tim\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vec
tor y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  return f(*args, **kwargs)
C:\Users\Tim\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vec
tor y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  return f(*args, **kwargs)
C:\Users\Tim\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vec
tor y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  return f(*args, **kwargs)
C:\Users\Tim\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vec
tor y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  return f(*args, **kwargs)
C:\Users\Tim\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vec
tor y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
```

```
Fold 1 Recall: 0.6206896551724138 f1_score 0.4615384615384615 Best Estimator SVC(C=60, class_weight='balance
d', kernel='linear')

Fold 2 Recall: 0.6666666666666666 f1_score 0.45849802371541504 Best Estimator SVC(C=100, class_weight='balan
ced', kernel='linear')

Fold 3 Recall: 0.6781609195402298 f1_score 0.4627450980392156 Best Estimator SVC(C=50, class_weight='balance
d', kernel='linear')

Fold 4 Recall: 0.6590909090909091 f1_score 0.44961240310077516 Best Estimator SVC(C=50, class_weight='balanc
ed', kernel='linear')

Fold 5 Recall: 0.6477272727272727 f1_score 0.4634146341463415 Best Estimator SVC(C=70, class_weight='balance
d', kernel='linear')
```

In [87]:
```
#13. Best Value of Parameter C: 50
```

In [88]:
```
# The best value of Parameter C is 50, which, when paired w/ StratifiedKfold tuning, yields a top Recall value o
# See results above in for loop.
```

In [ ]:
```

```

In [89]:
```
#14. Predict and Evaluate Using the Best Estimator.
```

In [90]:
```
# As shown above in the for loop, The best estimator was C = 50.
```

In [91]:
```
# Now we can run a new program w/ C=50 on all values, and use StratifiedKFold w/ 5 folds once again.
```

In [92]:
```
clf4 = svm.SVC(C=50, kernel='linear', class_weight='balanced')
```

In [ ]:
```

```

In [94]:
```
target = df5.loc[:,'toxic']
X_test5_all = []
predict5_all = []
```

In [95]:
```
def train_model(train5, test5, fold_no, X_test_all, predict_all):
    #X5 = [:4000]
    y5 = ['toxic']
    X_train5 = train5.iloc[:,:4000]
    y_train5 = train5[y5]
    X_test5 = test5.iloc[:,:4000]
    y_test5 = test5[y5]
    clf4.fit(X_train5,y_train5)
    predictions = clf4.predict(X_test5)
    X_test5_all = X_test_all.append(X_test5)
    predict5_all = predict_all.append(predictions)


    print('Fold',str(fold_no),'Recall:',recall_score(y_test5,predictions),'f1_score',f1_score(y_test5,prediction
    return X_test5_all,#predict5_all
```

In [96]:
```
fold_no = 1
```

In [97]:
```
for train_index, test_index in skf.split(df5, target):
    train5 = df5.loc[train_index,:]
    test5 = df5.loc[test_index,:]
    X_test_all = X_test5_all
    predict_all = predict5_all
    train_model(train5, test5, fold_no, X_test_all, predict_all)
    fold_no += 1
```

```
C:\Users\Tim\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vec
tor y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  return f(*args, **kwargs)

Fold 1 Recall: 0.6091954022988506 f1_score 0.4549356223175966

C:\Users\Tim\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vec
tor y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  return f(*args, **kwargs)

Fold 2 Recall: 0.632183908045977 f1_score 0.4867256637168141

C:\Users\Tim\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vec
tor y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  return f(*args, **kwargs)

Fold 3 Recall: 0.6781609195402298 f1_score 0.4627450980392156
```

```
C:\Users\Tim\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vec
tor y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  return f(*args, **kwargs)

Fold 4 Recall: 0.6590909090909091 f1_score 0.44961240310077516

C:\Users\Tim\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vec
tor y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  return f(*args, **kwargs)

Fold 5 Recall: 0.625 f1_score 0.5092592592592593
```

Last we look into which words are most prominent in the comments flagged as toxic by the clf4 model. We do this in two different ways. We first analyze the total number of occurrences of each word across the comments that were marked as toxic (line 120, below). This can be misleading because some users may post a comment that would repeat a series of words a large number of times, skewing the results. Because of this, we next analyze the total value of the words across all comments as determined by the TF-IDF Vectorizer that was employed to preprocess the data for the ML classification algorithms (line 132, below). The TF-IDF Vectorizer specifically takes into account the number of times that a word is used in a single comment, related to the number of comments that it is used in. This results in a more realistic presentation of the common words across comments classified as toxic.

```
In [98]: #15. Most Prominent Terms in the Toxic Comments

In [99]: X_test5_all
```

Out[99]:

	00	000	0000z	01	02	03	04	0422	05	06	...	zach	zero	\
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
1001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	

	zheng	мај	јул	јули	јун	јуни	ѓоп	ѕшо
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

In [101]:
```
X_test5_all_2 = pd.concat(X_test5_all,join = 'inner')
X_test5_all_2
```

Out[101]:

	00	000	0000z	01	02	03	04	0422	05	06	...	zach	zero	zheng	maj	jyл	jyли	jyн	jyни	кол	тыс
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5000 rows × 4000 columns

In [102]:
```
a = X_test5_all_2.index
print(*a, sep = "\n")
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
```

```
In [103]: predict5_all
```

```
Out[103]: [array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
                  0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
                  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
                  0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                  0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
                  0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                  0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
                  0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
                  0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
                  0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0,
                  0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
                  0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1,
                  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                  0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
                  0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                  0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
                  0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
                  0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                  0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
```

```
In [104]: predict5_all_2 = []
```

```
In [105]: for i in predict5_all:
              listed = i.tolist()
              predict5_all_2 += listed
```

```
In [106]: len(predict5_all_2)
```

```
Out[106]: 5000
```

```
In [107]: predict5_all_2
```

```
Out[107]: [0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           1,
```

In [108]:
```
X_test5_all_2['toxic_pred'] = predict5_all_2
X_test5_all_2
```

Out[108]:

	00	000	0000z	01	02	03	04	0422	05	06	...	zero	zheng	maj	jул	jули	jун	juни	сел	тыс	toxic_pred
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

In [109]:
```
HALO = X_test5_all_2.sort_index()
HALO
```

Out[109]:

	00	000	0000z	01	02	03	04	0422	05	06	...	zero	zheng	maj	jул	jули	jун	juни	сел	тыс	toxic_pred
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

In [111]:
```
y_all
```

Out[111]:
```
2654    0
2468    0
290     0
1463    1
4508    0
       ..
3412    1
4020    0
4635    0
1700    0
790     1
Name: toxic, Length: 5000, dtype: int64
```

In [112]:
```
HALO2 = HALO.set_index(y_all.index)
HALO2
```

Out[112]:

	00	000	0000z	01	02	03	04	0422	05	06	...	zero	zheng	maj	jул	jули	jун	juни	сша	тим	toxic_pred
2654	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
2468	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
290	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
1463	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4508	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3412	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4020	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4635	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
1700	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
790	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

5000 rows × 4001 columns

In [113]:
```
HALO2 = HALO2.sort_index()
HALO2
```

Out[113]:

	00	000	0000z	01	02	03	04	0422	05	06	...	zero	zheng	maj	jул	jули	jун	juни	сша	тим	toxic_pred
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
4997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

5000 rows × 4001 columns

In [114]:
```
df3 = df
```

In [115]:
```
df3['toxic_pred'] = HALO2['toxic_pred']
df3
```

Out[115]:

	id	comment_text	comment_text2	toxic	toxic_pred
0	e617e2489abe9bca	"\r\n\r\n A barnstar for you! \r\n\r\n The De...	barnstar defender wiki barnstar like kayastha ...	0	0
1	9250cf637294e09d	"\r\n\r\nThis seems unbalanced. whatever I ha...	seems unbalanced whatever said mathsci said fa...	0	0
2	ce1aa4592d5240ca	Marya Dzmitruk was born in Minsk, Belarus in M...	marya dzmitruk born minsk belarus march 19 199...	0	0
3	48105766ff7f075b	"\r\n\r\nTalkback\r\n\r\n Dear Celestia... "	talkback dear celestia	0	1
4	0543d4f82e5470b6	New Categories \r\n\r\nI honestly think that w...	new categories honestly think need add categor...	0	0
...	...	...	...	...	...
4995	60229df7b48ba6ff	"\r\n\r\n Dildo, if you read my response corre...	dildo read response correctly never said going...	0	0
4996	36a645227572ec5c	CALM DOWN, CALM DOWN, DON'T GET A BIG DICK	calm calm get big dick	1	1
4997	6d47fa39945ed6f5	In my opinion Dougweller is using his privileg...	opinion dougweller using privileges poorly per...	0	0
4998	de2e4c0d38db6e30	The style section has been expanded too. I did...	style section expanded remember placed tag	0	0
4999	4cda24210a33ac35	ANY ONE THAT IS NOT AGREEMENT WITH YOU OR IS A...	one agreement repulican joe hazelton.it wack m...	0	0

5000 rows × 5 columns

In [116]:
```
df_filtered = df[df['toxic_pred'] == 1]

toxic_pred_comments = df_filtered["comment_text2"].tolist()

def process (token):
    words_list3=[]
    word_tokens = word_tokenize(token)
    for word in word_tokens:
        words_list3.append(word)
    return words_list3

toxic = []

for i in toxic_pred_comments:
    x = process(i)
    toxic.append(x)

words_toxic = list(itertools.chain.from_iterable(toxic))
words_toxic
```

Out[116]:
```
['talkback',
 'dear',
 'celestia',
 'loser',
 'ca',
 'block',
 'forever',
 'admin',
 'ego',
 'hippie',
 'freak',
 'seen',
 'source',
 'gwern',
 'used',
 'ran',
 'across',
 'sites',
 'like',
```

In [117]:
```
freqTable3={}

for word in words_toxic:
    if word in freqTable3:
        freqTable3[word] += 1

    else:
        freqTable3[word] = 1
```

In [118]:
```
toxic_list_all = {k: v for k, v in sorted(freqTable3.items(),reverse=True, key=lambda item: item[1])}
toxic_list_all
```

Out[118]:
```
{'ass': 976,
 'fuck': 888,
 'suck': 370,
 'mexicans': 356,
 'fucking': 272,
 'gay': 229,
 'nigger': 187,
 'die': 162,
 'must': 162,
 'jim': 157,
 'wales': 156,
 'cuntbag': 126,
 '====': 125,
 'shit': 115,
 'bastard': 114,
 'pro-assad.hanibal911you': 106,
 'eat': 99,
 'admins': 99,
 'hate': 99,
```

In [119]:
```
toxic_top15 = ['ass','fuck','suck','mexicans','fucking','gay','nigger','die','must','jim','wales','cuntbag','===
```

In [120]:
```
toxic_top15
```

Out[120]:
```
['ass',
 'fuck',
 'suck',
 'mexicans',
 'fucking',
 'gay',
 'nigger',
 'die',
 'must',
 'jim',
 'wales',
 'cuntbag',
 '===',
 'shit',
 'bastard']
```

In [121]:
```
X_test5_all_2_filtered = X_test5_all_2[X_test5_all_2['toxic_pred'] == 1]
```

In [128]:
```
X_test5_all_2_filtered = X_test5_all_2_filtered.drop(['toxic_pred'], axis=1)
```

In [129]:
```
toxic_df = X_test5_all_2_filtered.transpose()
toxic_df
```

Out[129]:

	18	26	32	38	40	56	61	68	71	72	...	4949	4951	4962	4970	4971	4980	4983	4985	4989	4991
00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0000z	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
01	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
02	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
јули	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
јун	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
јуни	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ѕа	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ѕи	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

4000 rows × 751 columns

In [130]:
```
toxic_df['sum'] = toxic_df.sum(axis=1)
toxic_df
```

Out[130]:

	18	26	32	38	40	56	61	68	71	72	...	4951	4962	4970	4971	4980	4983	4985	4989	4991	sum
00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.606199
000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
0000z	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
01	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.374334
02	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
јули	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
јун	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
јуни	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
ѕа	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
ѕи	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000

4000 rows × 752 columns

In [131]:
```
toxic_df_top15 = toxic_df.nlargest(15,'sum')
toxic_df_top15
```

Out[131]:

	18	26	32	38	40	56	61	68	71	72	...	4951	4962	4970	4971	4980	4983	4985	4989	4991	sum
fuck	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	...	0.220921	0.000000	0.0	0.0	0.000000	0.0	0.000000	0.177293	0.0	25.506751
fucking	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	...	0.000000	0.000000	0.0	0.0	0.000000	0.0	0.598752	0.000000	0.0	17.282863
go	0.000000	0.0	0.0	0.0	0.444011	0.0	0.0	0.000000	0.0	0.0	...	0.167931	0.000000	0.0	0.0	0.000000	0.0	0.000000	0.000000	0.0	10.721418
gay	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	...	0.000000	0.000000	0.0	0.0	0.000000	0.0	0.000000	0.000000	0.0	9.074074
like	0.150836	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	...	0.248865	0.000000	0.0	0.0	0.000000	0.0	0.000000	0.000000	0.0	9.042856
ass	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	...	0.000000	0.000000	0.0	0.0	0.000000	0.0	0.000000	0.000000	0.0	8.865383
hey	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.822947	0.0	0.0	...	0.218347	0.000000	0.0	0.0	0.000000	0.0	0.000000	0.175228	0.0	8.210776
get	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	...	0.000000	0.000000	0.0	0.0	0.000000	0.0	0.142400	0.000000	0.0	7.928830
shit	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	...	0.000000	0.000000	0.0	0.0	0.000000	0.0	0.000000	0.000000	0.0	7.857516
people	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	...	0.000000	0.000000	0.0	0.0	0.000000	0.0	0.000000	0.000000	0.0	7.669021
suck	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	...	0.000000	0.000000	0.0	0.0	0.000000	0.0	0.000000	0.000000	0.0	7.664086
asshole	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	...	0.000000	0.000000	0.0	0.0	0.707107	0.0	0.000000	0.000000	0.0	6.816681
think	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	...	0.000000	0.000000	0.0	0.0	0.000000	0.0	0.000000	0.000000	0.0	6.547413
hell	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	...	0.000000	0.000000	0.0	0.0	0.000000	0.0	0.000000	0.000000	0.0	6.197224
dick	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	...	0.000000	0.382607	0.0	0.0	0.000000	0.0	0.000000	0.000000	0.0	6.073617

15 rows × 752 columns

In [132]:
```
toxic_df_top15.index
```

Out[132]:
```
Index(['fuck', 'fucking', 'go', 'gay', 'like', 'ass', 'hey', 'get', 'shit',
       'people', 'suck', 'asshole', 'think', 'hell', 'dick'],
      dtype='object')
```

Conclusions:

This toxicity detector developed for Wikipedia comments has been optimized using the above procedure. Further optimization may be achieved in the future with increased analysis of stopwords and additional stopword removal. A recall score of 64% is far from perfect, but will certainly help Wikipedia to achieve its business objective of promoting productive and respectful community interaction on its website.