nlp-twitter-combat-hate-speech

March 28, 2023

[1]:
```
pip install nltk
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: nltk in /usr/local/lib/python3.9/dist-packages
(3.8.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.9/dist-packages
(from nltk) (1.1.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages
(from nltk) (4.65.0)
Requirement already satisfied: click in /usr/local/lib/python3.9/dist-packages
(from nltk) (8.1.3)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.9/dist-
packages (from nltk) (2022.10.31)
```

[2]:
```
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
nltk.download('punkt')
nltk.download('stopwords')
import re
import warnings
warnings.filterwarnings('ignore')

import numpy as np,pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[3]:
```
df = pd.read_csv('/TwitterHate.csv')
```

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31962 entries, 0 to 31961
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   id      31962 non-null  int64
 1   label   31962 non-null  int64
 2   tweet   31962 non-null  object
dtypes: int64(2), object(1)
memory usage: 749.2+ KB
```

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

[5]:
```
   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
```

[8]:
```
df.isnull().sum()
```

[8]:
```
id       0
label    0
tweet    0
dtype: int64
```

[9]:
```
def text_preprocessing(text):
  text=re.sub('@',' ',text)
  text=re.sub('URLs',' ',text)
  text=re.sub('amp',' ',text)
  text=re.sub('#',' ',text)
  text=re.sub('rt',' ',text)
  text=re.sub('[^a-zA-z]',' ',text)
  text=text.lower()
  tweet_tokens=word_tokenize(text)
  text=[word for word in tweet_tokens if not word in stopwords.words('english')]
  return ' '.join(text)
```

[10]:
```
df['tweet']=df['tweet'].apply(text_preprocessing)
```

[11]:
```
df.duplicated().sum()
```

[11]: 0

[12]:
```
corpus=df['tweet'].apply(lambda x:word_tokenize(x))
corpus
```

[12]:
```
0        [user, father, dysfunctional, selfish, drags, …
1        [user, user, thanks, lyft, credit, use, cause,…
2                                       [bihday, majesty]
3                        [model, love, u, take, u, time, ur]
4                          [factsguide, society, motivation]
                                    …
31957                                [ate, user, isz, youuu]
31958    [see, nina, turner, airwaves, trying, wrap, ma…
31959    [listening, sad, songs, monday, morning, otw, …
31960    [user, sikh, temple, vandalised, calgary, wso,…
31961                                  [thank, user, follow]
Name: tweet, Length: 31962, dtype: object
```

[14]:
```
corpus=corpus.apply(lambda x:' '.join(x))
corpus
```

[14]:
```
0        u s e r   f a t h e r   d y s f u n c t i o n …
1        u s e r   u s e r   t h a n k s   l y f t   c …
2                              b i h d a y   m a j e s t y
3        m o d e l   l o v e   u   t a k e   u   t i m …
4        f a c t s g u i d e   s o c i e t y   m o t i …
                                    …
31957                  a t e   u s e r   i s z   y o u u u
31958    s e e   n i n a   t u r n e r   a i r w a v e …
31959    l i s t e n i n g   s a d   s o n g s   m o n …
31960    u s e r   s i k h   t e m p l e   v a n d a l …
31961                    t h a n k   u s e r   f o l l o w
Name: tweet, Length: 31962, dtype: object
```

[15]:
```
from collections import Counter
counter=Counter(corpus)
most_common_word=[]
for i in range(0,len(counter.most_common(10))):
  count=counter.most_common(10)[i][0]
  most_common_word.append(count)
  print(count)
  common_word=' '.join(most_common_word)
```

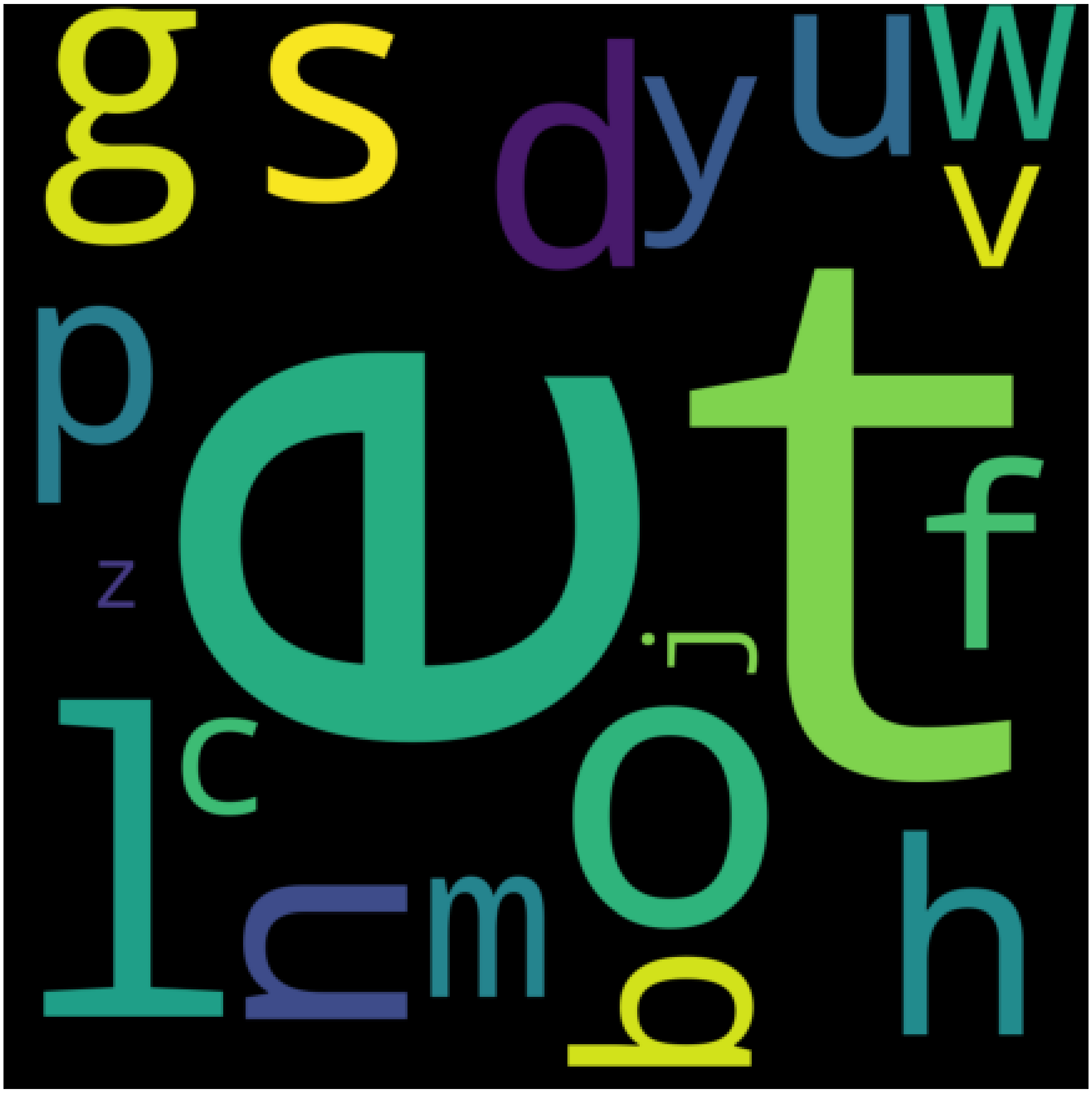
```
m o d e l   l o v e   u   t a k e   u   t i m e   u r
f i n a l l y   f o u n d   w a y   d e l e t e   o l d   t w e e t s   m i g h
t   f i n d   u s e f u l   w e l l   d e l e t e t w e e t s
a w w   y e a h   g o o d   b i n g   b o n g   b i n g   b o n g
g r a t e f u l   a f f i r m a t i o n s
u s e r   m i g h t   l i b t a r d   l i b t a r d   s j w   l i b e r a l   p
o l i t i c s
l o v e   i n s t a g o o d   p h o t o o f t h e d a y   t o p   t a g s   t b
t   c u t e   b e a u t i f u l   f o l l o w m e   f o l l o w
```

```
m i g h t   l i b t a r d   l i b t a r d   s j w   l i b e r a l   p o l i t i
c s
h a p p y   w o r k   c o n f e r e n c e   r i g h t   m i n d s e t   l e a d
s   c u l t u r e   d e v e l o p m e n t   o r g a n i z a t i o n s   w o r k
m i n d s e t
l i g h t t h e r a p y   h e l p   d e p r e s s i o n   a l t w a y s t o h e
a l   h e a l t h y   h a p p y
l o v e r   s t o p   a n g r y   v i s i t   u s   g t   g t   g t   l o v e r
f r i e n d   a s t r o l o g e r   l o v e
```

[18]:
```
from wordcloud import WordCloud
plt.figure(figsize=(10,8))
cloud=WordCloud(height=1000,width=1000,background_color='black')
img=cloud.generate(common_word)
plt.axis('off')
plt.title('Most common word',size=10)
plt.imshow(img)
```

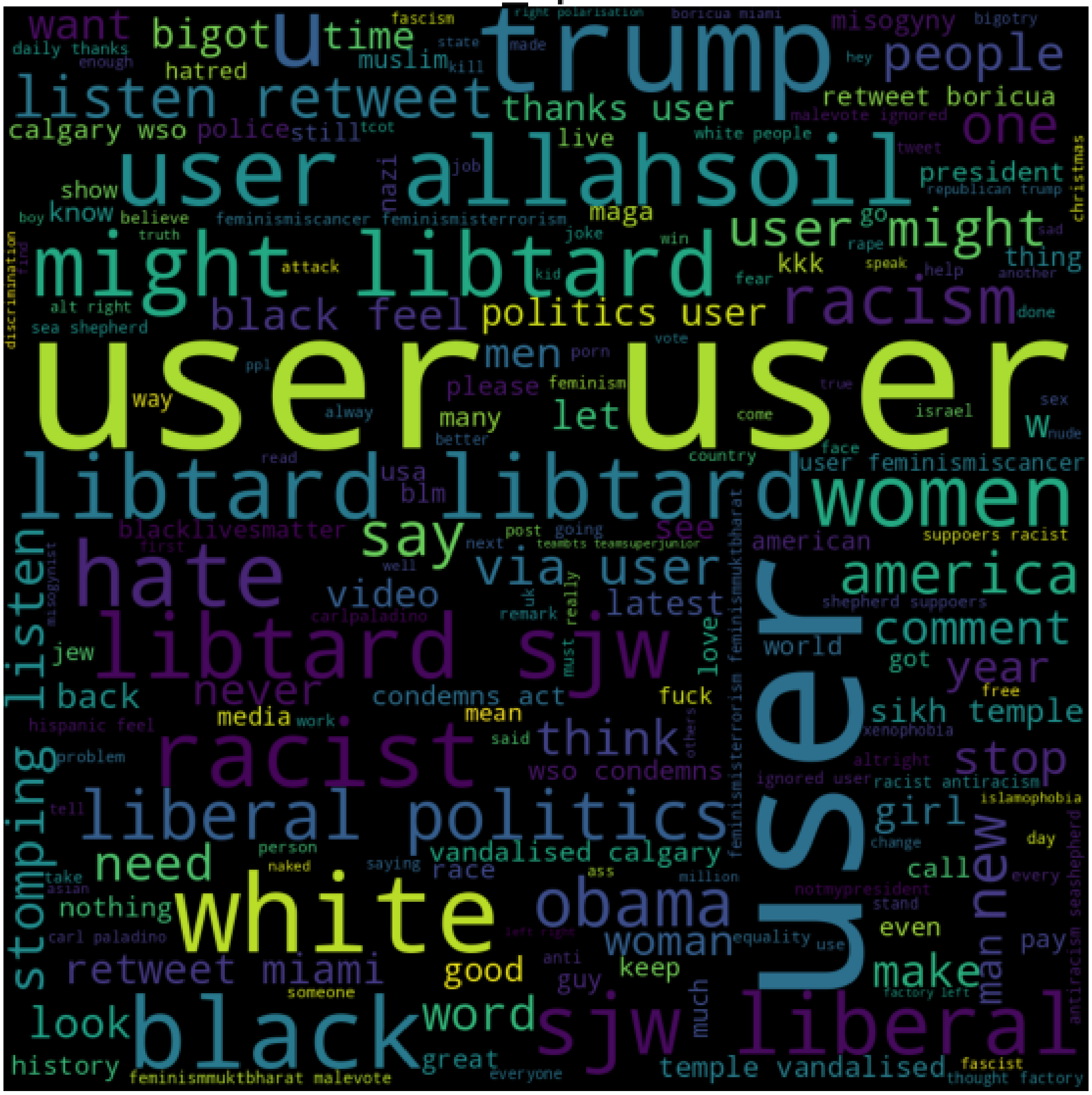
[18]:
```
<matplotlib.image.AxesImage at 0x7f5f8837e0d0>
```

Most common word



[19]:
```
plt.figure(figsize=(20,10))
Hate_speech=cloud.generate(df[df['label']==1]['tweet'].str.cat(sep=' '))
plt.imshow(Hate_speech)
plt.title('Hate_speech',size=25)
plt.axis('off')
plt.show()
```

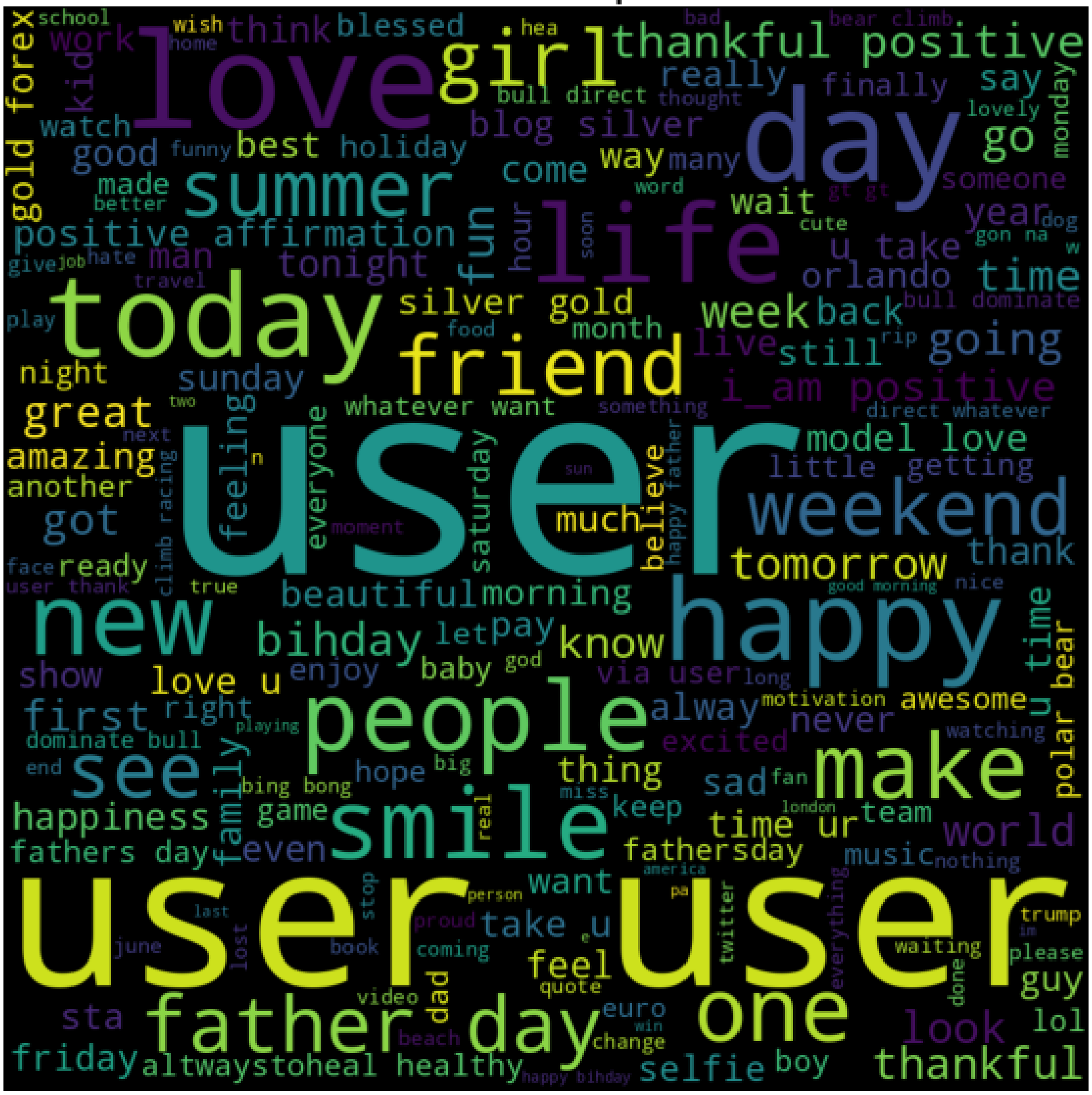
Hate_speech




```python
plt.figure(figsize=(15,10))
Non_hate_speech=cloud.generate(df[df['label']==0]['tweet'].str.cat(sep=' '))
plt.imshow(Non_hate_speech)
plt.title('Non-Hate-Speech',size=25)
plt.axis('off')
plt.imshow(Non_hate_speech)
```

[20]: <matplotlib.image.AxesImage at 0x7f5f87baeeb0>

Non-Hate-Speech




```
[21]: df.drop('id',inplace=True,axis=1)
      df.drop_duplicates(inplace=True)
```

#Data formatting for predictive modeling: # Join the tokens back to form strings. This will be required for the vectorizers. # Assign x and y. # Perform train_test_split using sklearn.

```
[22]: x=df['tweet']
      y=df['label']
```

```
[23]: x
```

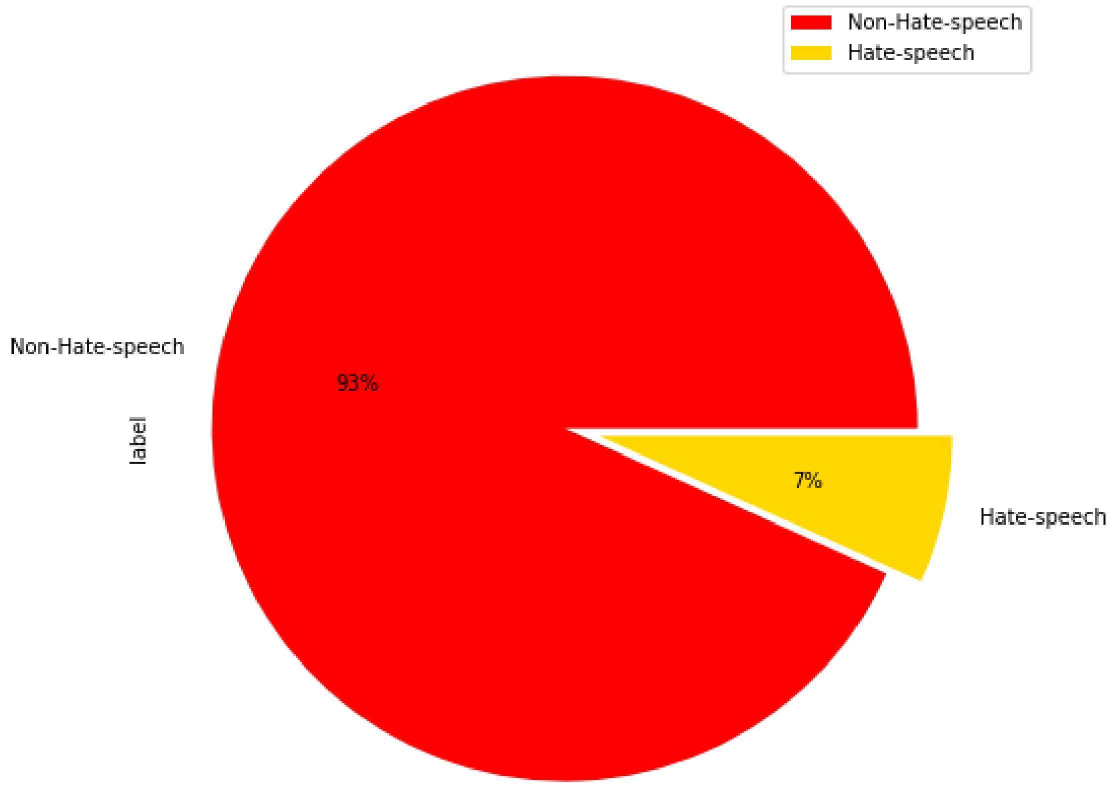
[23]:
```
0        user father dysfunctional selfish drags kids d…
1        user user thanks lyft credit use cause offer w…
2                                       bihday majesty
3                                model love u take u time ur
4                              factsguide society motivation
                                           …
31956    fishing tomorrow user carnt wait first time years
31957                                   ate user isz youuu
31958    see nina turner airwaves trying wrap mantle ge…
31959      listening sad songs monday morning otw work sad
31961                                    thank user follow
Name: tweet, Length: 29193, dtype: object
```

[24]:
```
y
```

[24]:
```
0        0
1        0
2        0
3        0
4        0
        ..
31956    0
31957    0
31958    0
31959    0
31961    0
Name: label, Length: 29193, dtype: int64
```

[26]:
```
#Our Data is imbalanced
y.value_counts().plot(kind='pie',figsize=(10,8),autopct='%1.
 ↪0f%%',colors=['red','gold'],explode=[0.
 ↪1,0],labels=['Non-Hate-speech','Hate-speech'])
plt.legend()
```

[26]:
```
<matplotlib.legend.Legend at 0x7f5f8835ffd0>
```



[27]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
↪2,random_state=25)
```

We'll use TF-IDF values for the terms as a feature to get into a vector space model. Import TF-IDF vectorizer from sklearn. Instantiate with a maximum of 5000 terms in your vocabulary. Fit and apply on the train set. Apply on the test set.

[28]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vector =TfidfVectorizer(max_features=5000)
x_train_vector =vector.fit_transform(x_train).toarray()
```

[29]:
```
x_test_vector=vector.transform(x_test).toarray()
```

Model building: Ordinary Logistic Regression Instantiate Logistic Regression from sklearn with default parameters. Fit into the train data. Make predictions for the train and the test set

[31]:
```
from sklearn.linear_model import LogisticRegression
lc=LogisticRegression(class_weight='balanced')
```

```
lc.fit(x_train_vector,y_train)
```

[31]:
```
LogisticRegression(class_weight='balanced')
```

[32]:
```
y_pred=lc.predict(x_test_vector)
```

Model evaluation: Accuracy, recall, and f_1 score. Report the accuracy on the train set. Report the recall on the train set: decent, high, or low. Get the f1 score on the train set.

[33]:
```
from sklearn.metrics import accuracy_score,classification_report
from sklearn.metrics import confusion_matrix,recall_score,f1_score
print("accuracy",accuracy_score(y_test,y_pred))
print("recall",recall_score(y_test,y_pred))
print("f1_score",f1_score(y_test,y_pred))
```

```
accuracy 0.9128275389621511
recall 0.8163771712158809
f1_score 0.5638389031705228
```

[34]:
```
x_test_vector
```

[34]:
```
array([[0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       …,
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.]])
```

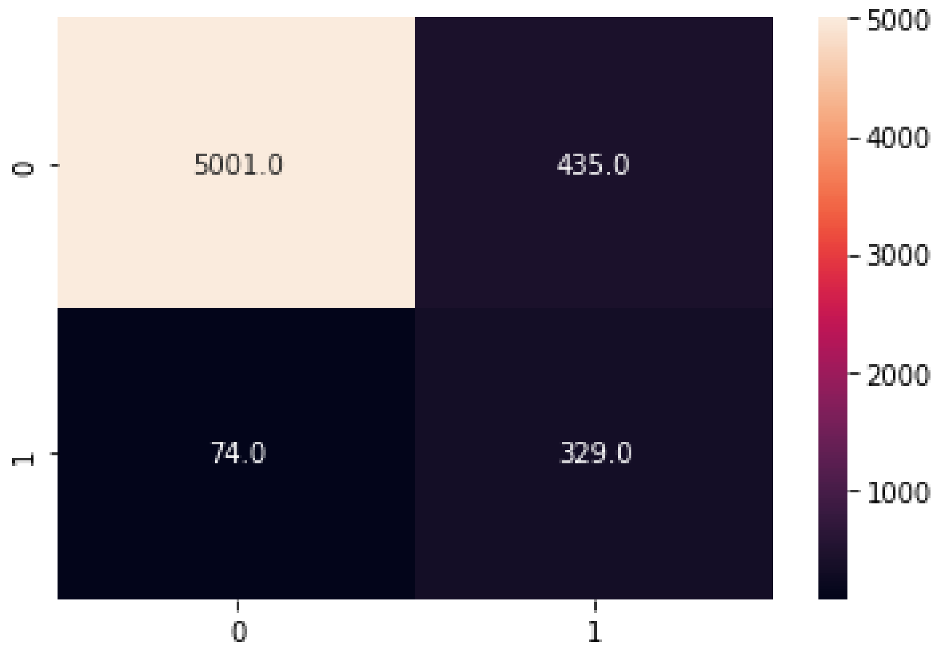
[35]:
```
print(classification_report(y_test,y_pred))
sns.heatmap((confusion_matrix(y_test,y_pred)),annot=True,fmt='0.1f')
```

```
              precision    recall  f1-score   support

           0       0.99      0.92      0.95      5436
           1       0.43      0.82      0.56       403

    accuracy                           0.91      5839
   macro avg       0.71      0.87      0.76      5839
weighted avg       0.95      0.91      0.92      5839
```

[35]:
```
<Axes: >
```



Looks like you need to adjust the class imbalance, as the model seems to focus on the 0s. Adjust the appropriate class in the LogisticRegression model.

[36]:
```
df.label.value_counts()
```

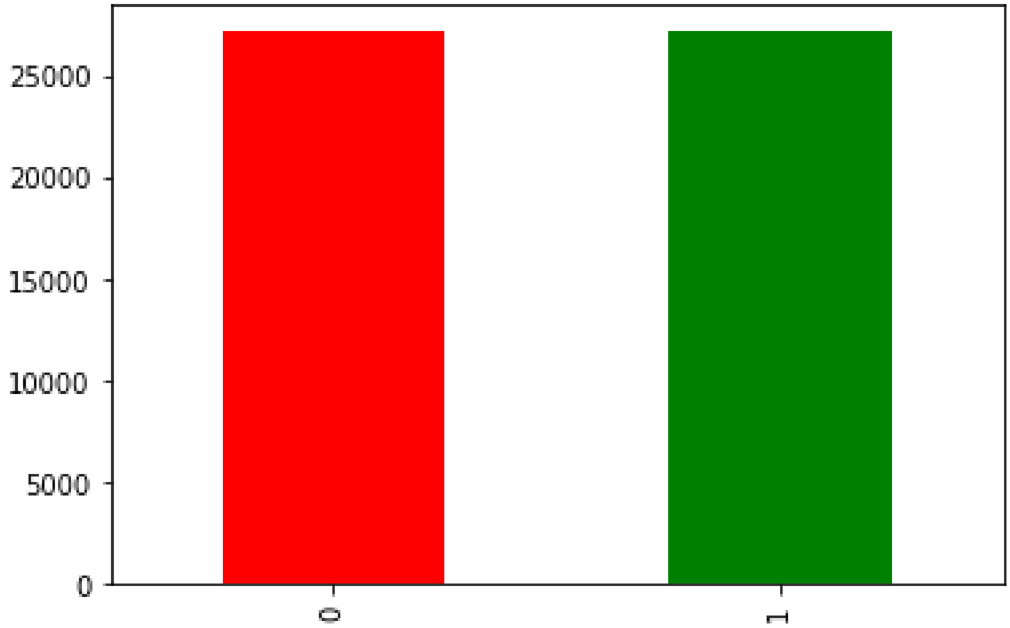
[36]:
```
0    27222
1     1971
Name: label, dtype: int64
```

[37]:
```
label1=df[df['label']==1].sample(25230,replace=True)
```

[38]:
```
normalised_data=pd.concat([df,label1])
```

[39]:
```
normalised_data['label'].value_counts().plot(kind='bar',color=['red','green'])
```

[39]:
```
<Axes: >
```



[40]:
```
x_vector1=vector.fit_transform(normalised_data['tweet']).toarray()
```

[41]:
```
x_vector1
```

[41]:
```
array([[0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       …,
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.]])
```

Train again with the adjustment and evaluate. 1. Train the model on the train set. 1. Evaluate the predictions on the train set: accuracy, recall, and f_1 score.italicized text

[43]:
```
x_train1,x_test1,y_train1,y_test1=train_test_split(x_vector1,normalised_data['label'],test_siz
↪2,random_state=100)
```

[44]:
```
lc.fit(x_train1,y_train1)
```

[44]:
```
LogisticRegression(class_weight='balanced')
```

[45]:
```
y_pred1=lc.predict(x_test1)
```

[46]:
```
print("accuracy",accuracy_score(y_test1,y_pred1))
```

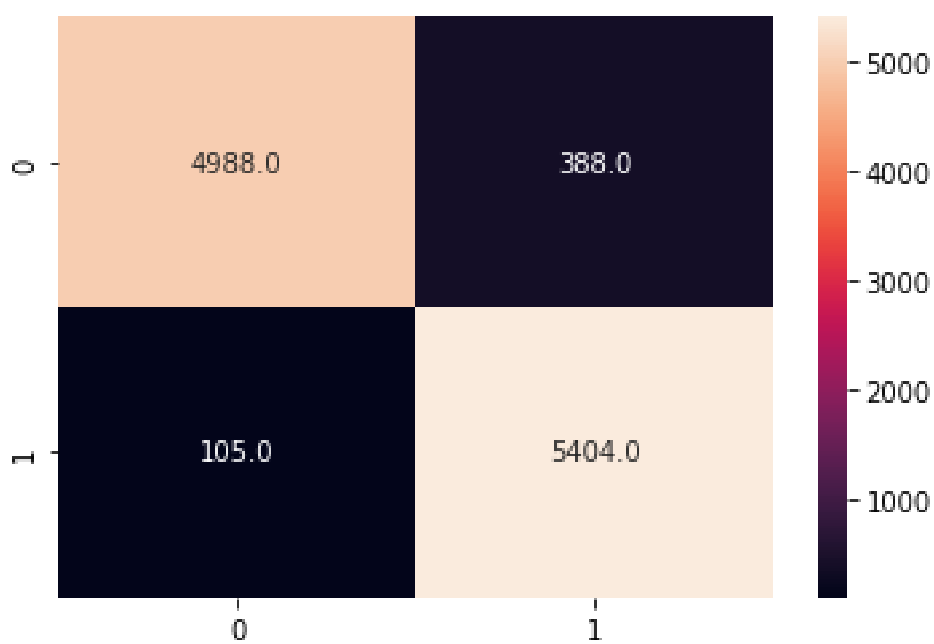
```python
LogisticRegression(class_weight='balanced')
print("recall",recall_score(y_test1,y_pred1))
print("f1_score",f1_score(y_test1,y_pred1))
print(classification_report(y_test1,y_pred1))
sns.heatmap((confusion_matrix(y_test1,y_pred1)),annot=True,fmt='0.1f')
```

```
accuracy 0.9547083141938447
recall 0.9809402795425667
f1_score 0.9563755419874348
              precision    recall  f1-score   support

           0       0.98      0.93      0.95      5376
           1       0.93      0.98      0.96      5509

    accuracy                           0.95     10885
   macro avg       0.96      0.95      0.95     10885
weighted avg       0.96      0.95      0.95     10885
```

[46]: <Axes: >



Regularization and Hyperparameter tuning: A. Import GridSearch and StratifiedKFold because of class imbalance. B. Provide the parameter grid to choose for 'C' and 'penalty' parameters. C. Use a balanced class weight while instantiating the logistic regression. 13 .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.

[47]:
```
from sklearn.model_selection import␣
  ↪RandomizedSearchCV,GridSearchCV,StratifiedKFold
lc=LogisticRegression(class_weight='balanced')
cv=StratifiedKFold(n_splits=4)
parameters = {
'penalty' : ['l1','l2'],
'C' : np.logspace(-3,3,7)
}
grid_cv=GridSearchCV(lc,param_grid=parameters,cv=cv,scoring='recall',n_jobs=-1)
```

[48]:
```
grid_search=grid_cv.fit(x_train1,y_train1)
```

[49]:
```
grid_search.best_params_
```

[49]:
```
{'C': 1000.0, 'penalty': 'l2'}
```

[50]:
```
acuracy=grid_search.best_score_
acuracy
```

[50]:
```
0.997925502489397
```

Predict and evaluate using the best estimator. Use the best estimator from the grid search to make predictions on the test set. What is the recall on the test set for the toxic comments? What is the f_1 score?

[51]:
```
lc=LogisticRegression(penalty='l2',C=100)
lc.fit(x_train1,y_train1)
```

[51]:
```
LogisticRegression(C=100)
```

[52]:
```
y_pred2=lc.predict(x_test1)
```

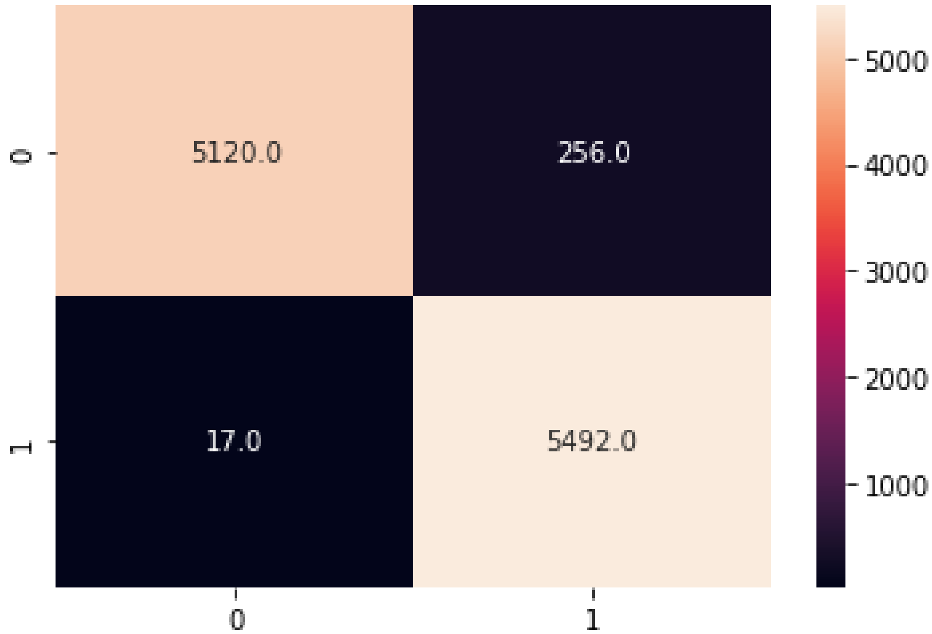
[53]:
```
print("recall",recall_score(y_test1,y_pred2))
print("f1_score",f1_score(y_test1,y_pred2))
print(classification_report(y_test1,y_pred2))
sns.heatmap((confusion_matrix(y_test1,y_pred2)),annot=True,fmt='0.1f')
```

```
recall 0.9969141404973679
f1_score 0.9757484232033401
              precision    recall  f1-score   support

           0       1.00      0.95      0.97      5376
           1       0.96      1.00      0.98      5509

    accuracy                           0.97     10885
   macro avg       0.98      0.97      0.97     10885
weighted avg       0.98      0.97      0.97     10885
```

[53]: <Axes: >



[54]:
```python
pred=y_pred2=lc.predict(x_train1)
```

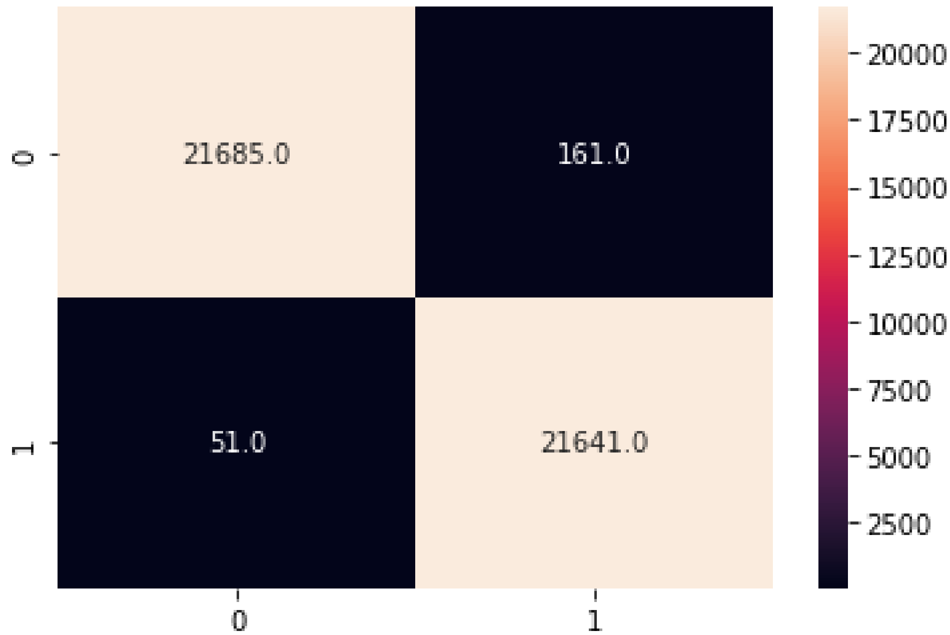
[55]:
```python
print("recall",recall_score(y_train1,pred))
print("f1_score",f1_score(y_train1,pred))
print(classification_report(y_train1,pred))
sns.heatmap((confusion_matrix(y_train1,pred)),annot=True,fmt='0.1f')
```

```
recall 0.9976489028213166
f1_score 0.9951257644732607
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     21846
           1       0.99      1.00      1.00     21692

    accuracy                           1.00     43538
   macro avg       1.00      1.00      1.00     43538
weighted avg       1.00      1.00      1.00     43538
```

[55]: <Axes: >


21685.0
161.0
51.0
21641.0
0
1
0
1
20000
17500
15000
12500
10000
7500
5000
2500

[ ]: