# case_study_1_1

March 17, 2020

Description

The Conversation AI team, a research initiative founded by Jigsaw and Google (both part of Alphabet), builds technology to protect voices in conversation. A main area of focus is machine learning models that can identify toxicity in online conversations, where toxicity is defined as anything rude, disrespectful or otherwise likely to make someone leave a discussion.

In year 2018, in the Toxic Comment Classification Challenge, you built multi-headed models to recognize toxicity and several subtypes of toxicity. This year's competition is a related challenge: building toxicity models that operate fairly across a diverse range of conversations.

Here's the background: When the Conversation AI team first built toxicity models, they found that the models incorrectly learned to associate the names of frequently attacked identities with toxicity. Models predicted a high likelihood of toxicity for comments containing those identities (e.g. "gay"), even when those comments were not actually toxic (such as "I am a gay woman"). This happens because training data was pulled from available sources where unfortunately, certain identities are overwhelmingly referred to in offensive ways. Training a model from data with these imbalances risks simply mirroring those biases back to users.

In this competition, you're challenged to build a model that recognizes toxicity and minimizes this type of unintended bias with respect to mentions of identities. You'll be using a dataset labeled for identity mentions and optimizing a metric designed to measure unintended bias. Develop strategies to reduce unintended bias in machine learning models, and you'll help the Conversation AI team, and the entire industry, build models that work well for a wide range of conversations.

Disclaimer: The dataset for this competition contains text that may be considered profane, vulgar, or offensive.

Disclaimer: The dataset for this competition contains text that may be considered profane, vulgar, or offensive.

Evaluation Metric

Competition Evaluation

This competition will use a newly developed metric that combines several submetrics to balance overall performance with various aspects of unintended bias.

First, we'll define each submetric.

Overall AUC

This is the ROC-AUC for the full evaluation set.

Bias AUCs

To measure unintended bias, we again calculate the ROC-AUC, this time on three specific subsets of the test set for each identity, each capturing a different aspect of unintended bias. You can learn more about these metrics in Conversation AI's recent paper Nuanced Metrics for Measuring Unintended Bias with Real Data in Text Classification.

Subgroup AUC: Here, we restrict the data set to only the examples that mention the specific identity subgroup. A low value in this metric means the model does a poor job of distinguishing between toxic and non-toxic comments that mention the identity.

BPSN (Background Positive, Subgroup Negative) AUC: Here, we restrict the test set to the non-toxic examples that mention the identity and the toxic examples that do not. A low value in this metric means that the model confuses non-toxic examples that mention the identity with toxic examples that do not, likely meaning that the model predicts higher toxicity scores than it should for non-toxic examples mentioning the identity.

BNSP (Background Negative, Subgroup Positive) AUC: Here, we restrict the test set to the toxic examples that mention the identity and the non-toxic examples that do not. A low value here means that the model confuses toxic examples that mention the identity with non-toxic examples that do not, likely meaning that the model predicts lower toxicity scores than it should for toxic examples mentioning the identity.

Generalized Mean of Bias AUCs

To combine the per-identity Bias AUCs into one overall measure, we calculate their generalized mean as defined below:

Mp(ms)=(1N s=1Nmps)1pMp(ms)=(1N s=1Nmsp)1p

where:

MpMp

= the pp

th power-mean function msms

= the bias metric mm

calulated for subgroup ss

NN

= number of identity subgroups

For this competition, we use a pp

value of -5 to encourage competitors to improve the model for the identity subgroups with the lowest model performance.

Final Metric

We combine the overall AUC with the generalized mean of the Bias AUCs to calculate the final model score:

score=w0AUCoverall+ a=1AwaMp(ms,a)score=w0AUCoverall+ a=1AwaMp(ms,a)

where:

A = number of submetrics (3) ms,ams,a

= bias metric for identity subgroup ss

using submetric aa

wawa

= a weighting for the relative importance of each submetric; all four ww

values set to 0.25

While the leaderboard will be determined by this single number, we highly recommend looking at the individual submetric results, as shown in this kernel, to guide you as you develop your models.

Submission File

Data Overview

Background

At the end of 2017 the Civil Comments platform shut down and chose make their ~2m public comments from their platform available in a lasting open archive so that researchers could understand and improve civility in online conversations for years to come. Jigsaw sponsored this effort and extended annotation of this data by human raters for various toxic conversational attributes.

In the data supplied for this competition, the text of the individual comment is found in the comment_text column. Each comment in Train has a toxicity label (target), and models should predict the target toxicity for the Test data. This attribute (and all others) are fractional values which represent the fraction of human raters who believed the attribute applied to the given comment. For evaluation, test set examples with target >= 0.5 will be considered to be in the positive class (toxic).

The data also has several additional toxicity subtype attributes. Models do not need to predict these attributes for the competition, they are included as an additional avenue for research. Subtype attributes are:

severe_toxicity

obscene

threat

insult

identity_attack

sexual_explicit

Additionally, a subset of comments have been labelled with a variety of identity attributes, representing the identities that are mentioned in the comment. The columns corresponding to identity attributes are listed below. Only identities with more than 500 examples in the test set (combined public and private) will be included in the evaluation calculation. These identities are shown in bold.

male

female

transgender

other_gender

heterosexual

homosexual_gay_or_lesbian

bisexual

other_sexual_orientation

christian

jewish

muslim

hindu

buddhist

atheist

other_religion

black

white

asian

latino

other_race_or_ethnicity

physical_disability

intellectual_or_learning_disability

psychiatric_or_mental_illness

other_disability

Note that the data contains different comments that can have the exact same text. Different comments that have the same text may have been labeled with different targets or subgroups.

Examples

Here are a few examples of comments and their associated toxicity and identity labels. Label values range from 0.0 - 1.0 represented the fraction of raters who believed the label fit the comment.

Comment: i'm a white woman in my late 60's and believe me, they are not too crazy about me either!!

Toxicity Labels: All 0.0

Identity Mention Labels: female: 1.0, white: 1.0 (all others 0.0)

Comment: Why would you assume that the nurses in this story were women?

Toxicity Labels: All 0.0

Identity Mention Labels: female: 0.8 (all others 0.0)

Comment: Continue to stand strong LGBT community. Yes, indeed, you'll overcome and you have.

Toxicity Labels: All 0.0

Identity Mention Labels: homosexual_gay_or_lesbian: 0.8, bisexual: 0.6, transgender: 0.3 (all others 0.0)

In addition to the labels described above, the dataset also provides metadata from Jigsaw's annotation: toxicity_annotator_count and identity_annotator_count, and metadata from Civil Comments: created_date, publication_id, parent_id, article_id, rating, funny, wow, sad, likes, disagree. Civil Comments' label rating is the civility rating Civil Comments users gave the comment.

Labelling Schema

To obtain the toxicity labels, each comment was shown to up to 10 annotators*. Annotators were asked to: "Rate the toxicity of this comment"

Very Toxic (a very hateful, aggressive, or disrespectful comment that is very likely to make you leave a discussion or give up on sharing your perspective)

Toxic (a rude, disrespectful, or unreasonable comment that is somewhat likely to make you leave a discussion or give up on sharing your perspective)

Hard to Say

Not Toxic

These ratings were then aggregated with the target value representing the fraction of annotations that annotations fell within the former two categories.

To collect the identity labels, annotators were asked to indicate all identities that were mentioned in the comment. An example question that was asked as part of this annotation effort was: "What genders are mentioned in the comment?"

Male

Female

Transgender

Other gender

No gender mentioned

Again, these were aggregated into fractional values representing the fraction of raters who said the identity was mentioned in the comment.

The distributions of labels and subgroup between Train and Test can be assumed to be similar, but not exact.

*Note: Some comments were seen by many more than 10 annotators (up to thousands), due to sampling and strategies used to enforce rater accuracy.

File descriptions

train.csv - the training set, which includes toxicity labels and subgroups test.csv - the test set, which does not include toxicity labels or subgroups

sample_submission.csv - a sample submission file in the correct format

Usage

This dataset is released under CC0, as is the underlying comment text.

[2]:
```
import pandas as pd
```

[27]:
```
train_data = pd.read_csv('train/train.csv')
print(train_data.shape)
```

```
(1804874, 45)
```

[3]:
```
print(train_data.columns)
```

```
Index(['id', 'target', 'comment_text', 'severe_toxicity', 'obscene',
       'identity_attack', 'insult', 'threat', 'asian', 'atheist', 'bisexual',
       'black', 'buddhist', 'christian', 'female', 'heterosexual', 'hindu',
       'homosexual_gay_or_lesbian', 'intellectual_or_learning_disability',
       'jewish', 'latino', 'male', 'muslim', 'other_disability',
       'other_gender', 'other_race_or_ethnicity', 'other_religion',
       'other_sexual_orientation', 'physical_disability',
       'psychiatric_or_mental_illness', 'transgender', 'white', 'created_date',
       'publication_id', 'parent_id', 'article_id', 'rating', 'funny', 'wow',
       'sad', 'likes', 'disagree', 'sexual_explicit',
       'identity_annotator_count', 'toxicity_annotator_count'],
      dtype='object')
```

**There are more than 1 million records and 45 columns and among 45 columns following columns are important including target and comment text :**

**identity attributes, representing the identities that are mentioned in the comment**
male

female

transgender

other_gender

heterosexual

homosexual_gay_or_lesbian

bisexual

other_sexual_orientation

christian

jewish

muslim

hindu

buddhist

atheist

other_religion

black

white

asian

latino

other_race_or_ethnicity

physical_disability

intellectual_or_learning_disability

psychiatric_or_mental_illness

other_disability

import plotly.graph_objects as go import warnings

warnings.filterwarnings('ignore')We need to consider itentities mentioned in bold

[8]:
```
train_data.dtypes
```

[8]:
```
id                                      int64
target                                float64
comment_text                           object
severe_toxicity                       float64
obscene                               float64
identity_attack                       float64
insult                                float64
threat                                float64
asian                                 float64
atheist                               float64
bisexual                              float64
black                                 float64
buddhist                              float64
christian                             float64
female                                float64
heterosexual                          float64
hindu                                 float64
homosexual_gay_or_lesbian             float64
intellectual_or_learning_disability   float64
jewish                                float64
latino                                float64
```

```
male                                          float64
muslim                                        float64
other_disability                              float64
other_gender                                  float64
other_race_or_ethnicity                       float64
other_religion                                float64
other_sexual_orientation                      float64
physical_disability                           float64
psychiatric_or_mental_illness                 float64
transgender                                   float64
white                                         float64
created_date                                   object
publication_id                                  int64
parent_id                                     float64
article_id                                      int64
rating                                         object
funny                                           int64
wow                                             int64
sad                                             int64
likes                                           int64
disagree                                        int64
sexual_explicit                               float64
identity_annotator_count                        int64
toxicity_annotator_count                        int64
dtype: object
```

[5]:
```
train_data.comment_text.describe()
```

[5]:
```
count        1804874
unique       1780823
top       Well said.
freq             184
Name: comment_text, dtype: object
```

[39]:
```
def printCommentText(index):
    print(train_data_after_EDA.comment_text.values[index])
    print('#'*100)
```

[43]:
```
printCommentText(2000)
printCommentText(20000)
printCommentText(200000)
printCommentText(206353)
printCommentText(22342)
printCommentText(1)
```

```
I equally love men. leafy i love you. hugs and kisses.
################################################################################
```

```
####################
I agree with you Mr. Elrey.  People should be required to take a class in order
to publicly carry firearms.  These are serious tools and have to be treated
seriously. There are many people I would be comfortable around who carry weapons
but God help us if it becomes too "cool" to be seen with a weapon and people
start carrying as a fashion statement.
################################################################################
####################
Goodbye Norma Jean…
################################################################################
####################
Oh, you mean when Trump lied about his income tax returns and support for the
Iraq war and Holt didn't docile accept his lies?  Or are you upset at the other
lies he told, like the birther lies long after he knew Obama was born an
American citizen ?  Analysis after analysis has shown conclusively that Trump
lies constantly.  Ignore them at your peril.
################################################################################
####################
Great season ladies your helping to make this a BASKETBALL TOWN! It was good to
see all the community support. Our coach is also a keeper.
################################################################################
####################
Thank you!! This would make my life a lot less anxiety-inducing. Keep it up, and
don't let anyone get in your way!
################################################################################
####################
```

Some capital letters are there

Punchuations are there

Unwanted spaces are there

Stop words are there

[1]:
```python
import plotly.graph_objects as go

import re
import nltk
nltk.download('punkt')
nltk.download('wordnet')
from nltk.stem.wordnet import WordNetLemmatizer
from nltk import word_tokenize
from nltk.stem import PorterStemmer

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
import pickle
```

```
from sklearn.metrics import␣
↪roc_auc_score,roc_curve,auc,confusion_matrix,classification_report
%matplotlib inline

import pandas as pd
import numpy as np
import scipy
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import pickle
from tqdm import tqdm
import seaborn as sns
# import logging
# logger = logging.getLogger("distributed.worker")
# logger1 = logging.getLogger("distributed.utils_perf")
# logger.setLevel(logging.ERROR)
# logger1.setLevel(logging.ERROR)
import seaborn as sns
import time
import gc
import itertools

from tqdm import tqdm
from nltk import FreqDist
from nltk.corpus import stopwords
from wordcloud import WordCloud
from multiprocessing import Pool

plt.style.use('ggplot')
tqdm.pandas()

from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from xgboost import XGBClassifier
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import StackingClassifier, RandomForestClassifier

from sklearn import metrics
import joblib

import warnings
warnings.filterwarnings('ignore')
```

```
[nltk_data] Downloading package punkt to /home/user/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
```

```
[nltk_data] Downloading package wordnet to /home/user/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
/home/user/anaconda3/lib/python3.7/site-packages/tqdm/std.py:658: FutureWarning:

The Panel class is removed from pandas. Accessing it from the top-level
namespace will also be removed in the next version
```

# 1 Exploratory Data Analysis

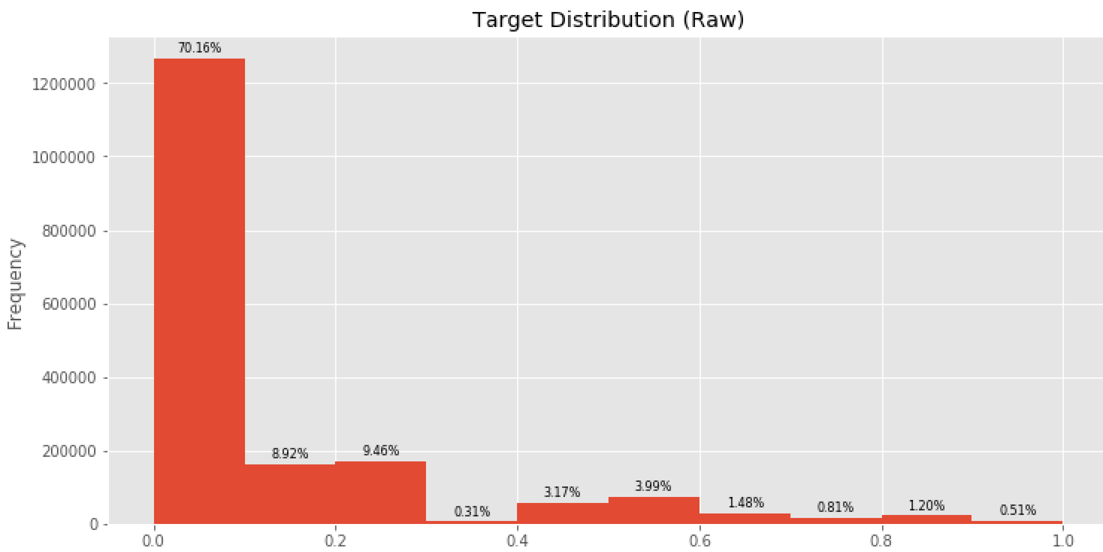
[22]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',␣
↪"you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',␣
↪'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its',␣
↪'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this',␣
↪'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',␣
↪'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',␣
↪'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'into', 'through', 'during',␣
↪'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on',␣
↪'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how',␣
↪'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so',␣
↪'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should',␣
↪"should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',␣
↪"didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't",␣
↪'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',␣
↪"shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

[23]:
```
toxic_subtypes = ['severe_toxicity', 'obscene', 'identity_attack', 'insult',␣
↪'threat', 'sexual_explicit']
```

```
identities = ['asian', 'atheist', 'bisexual',
       'black', 'buddhist', 'christian', 'female', 'heterosexual', 'hindu',
       'homosexual_gay_or_lesbian', 'intellectual_or_learning_disability',
       'jewish', 'latino', 'male', 'muslim', 'other_disability',
       'other_gender', 'other_race_or_ethnicity', 'other_religion',
       'other_sexual_orientation', 'physical_disability',
       'psychiatric_or_mental_illness', 'transgender', 'white']

selected_identities = [
    'male', 'female', 'homosexual_gay_or_lesbian', 'christian', 'jewish',
    'muslim', 'black', 'white', 'psychiatric_or_mental_illness']
```

## 1.1 Target Distribution

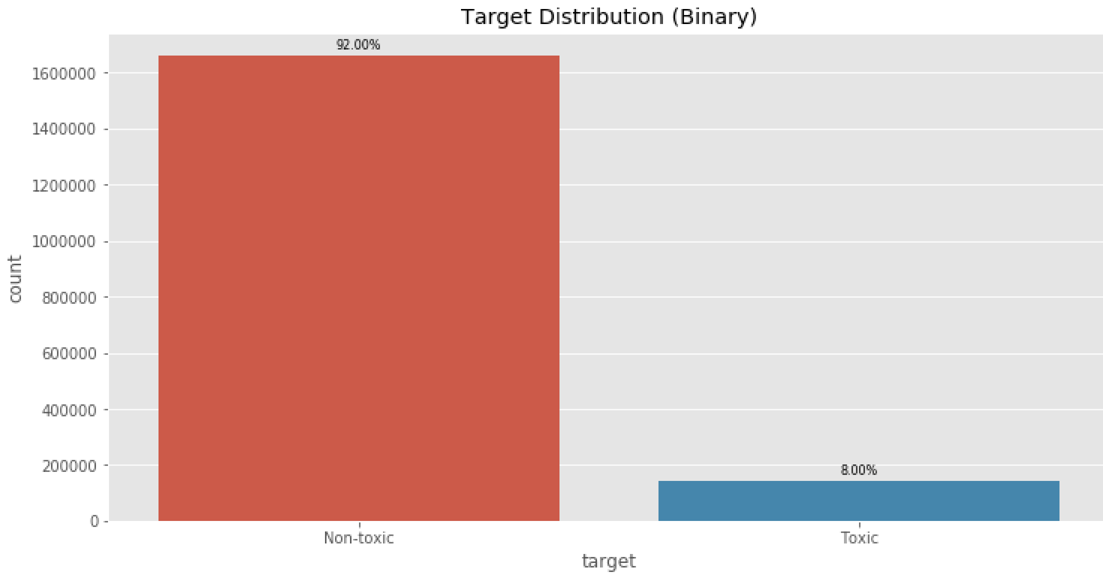
[28]:
```
plt.figure(figsize=(12,6))

plot = train_data.target.plot(kind='hist', bins=10)

ax = plot.axes

for p in ax.patches:
    ax.annotate(f'{p.get_height() * 100 / train.shape[0]:.2f}%',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center',
                va='center',
                fontsize=8,
                color='black',
                xytext=(0,7),
                textcoords='offset points')
plt.title('Target Distribution (Raw)')
plt.show()
```




[29]:
```
def convert_to_bool(df, col_name):
    df[col_name] = np.where(df[col_name] >= 0.5, True, False)

def convert_dataframe_to_bool(df):
    bool_df = df.copy()
    for col in ['target'] + selected_identities:
        convert_to_bool(bool_df, col)
    return bool_df

train_data = convert_dataframe_to_bool(train_data)
```

[30]:
```
plt.figure(figsize=(12,6))
plot = sns.countplot(x='target', data=pd.DataFrame(train_data['target'].
 ↪map({True:'Toxic', False:'Non-toxic'}), columns=['target']))

ax = plot.axes

for p in ax.patches:
    ax.annotate(f'{p.get_height() * 100 / train.shape[0]:.2f}%',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center',
                va='center',
                fontsize=8,
                color='black',
                xytext=(0,7),
                textcoords='offset points')

plt.title('Target Distribution (Binary)')
```

```
plt.show()
```




### 1.1.1 key Takeaways

**Before Binarization**

- Around 70% of data is having target value < 0.1 i.e non-toxic
- But there are 30 % of data having target value > 0.1
- Of all the 10 bins the most interesting bins to notice are 0.1 to 0.5 as annotators seems to be confused if those comments are toxic or not and hence our model may also be confused for those comments. #### After Binarization
- It is a highly imbalanced dataset having only 8% toxic data

## 1.2 Comment Length

[31]:
```
def decontracted(phrase):
    phrase = re.sub(r"won\'t", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
```

```
    phrase = phrase.replace('\\r', ' ')
    phrase = phrase.replace('\\n', ' ')
    phrase = phrase.replace('\\"', ' ')
    phrase = re.sub('[^A-Za-z0-9]+', ' ', phrase)
    return phrase
```

[15]:
```
def cleanComments(text):
    sent = decontracted(text)
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords).
 ↪lower().strip()
    return sent
```

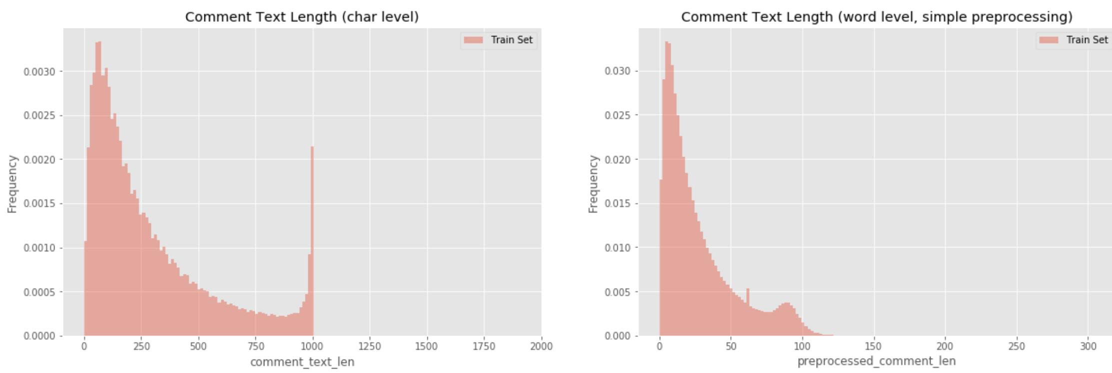
[16]:
```
def preprocessing(titles_array, return_len = False):

    processed_array = []

    for title in tqdm(titles_array):

        # remove other non-alphabets symbols with space (i.e. keep only␣
 ↪alphabets and whitespaces).
        processed = cleanComments(title)

        words = processed.split()

        if return_len:
            processed_array.append(len([word for word in words if word not in␣
 ↪stopwords]))
        else:
            processed_array.append(' '.join([word for word in words if word not␣
 ↪in stopwords]))

    return processed_array
```

[17]:
```
train_data['comment_text_len'] = train_data['comment_text'].progress_apply(len)

train_data['preprocessed_comment_len'] =␣
 ↪preprocessing(train_data['comment_text'], return_len=True)
```

```
100%|          | 1804874/1804874 [00:01<00:00, 1215870.78it/s]
100%|          | 1804874/1804874 [04:04<00:00, 7392.51it/s]
```

[18]:
```
plt.figure(figsize=(20,6))
plt.subplot(121)
sns.distplot(train_data['comment_text_len'], kde=False, bins=150, label='Train␣
 ↪Set', norm_hist=True)
plt.legend()
plt.ylabel('Frequency')
```

```
plt.title('Comment Text Length (char level)')

plt.subplot(122)
sns.distplot(train_data['preprocessed_comment_len'], kde=False, bins=150,␣
↪label='Train Set', norm_hist=True)
plt.legend()
plt.ylabel('Frequency')
plt.title('Comment Text Length (word level, simple preprocessing)')
plt.show()
```



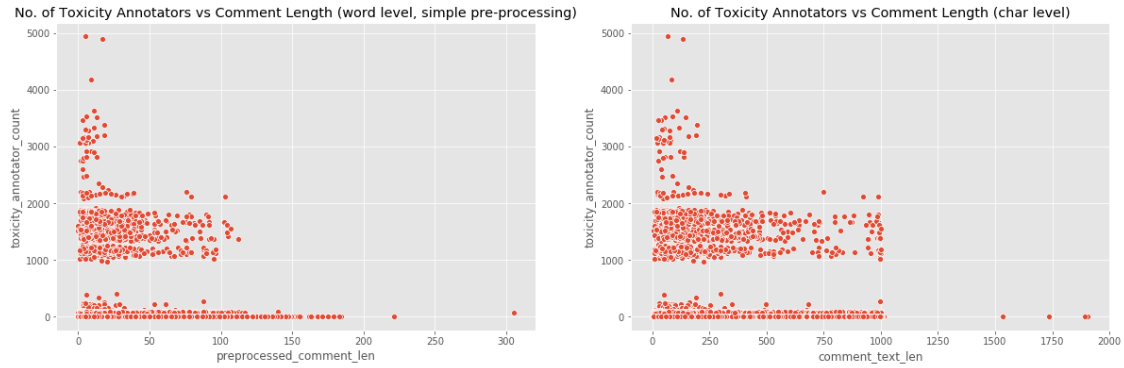

### 1.2.1 Key Takeaways

- Majority of comments have charecter lenght < 1000 but there are few comments with charecter length > 1000. This may be due to some special charecters or stopwords that we removed while cleaning comments.
- The maximum word length of comment text is around 130 after cleaning the comment text. That is a reasonable length.

## 1.3 No. of Toxicity Annotators vs Comment Length

[19]:
```
plt.figure(figsize=(20,6))
plt.subplot(121)
sns.scatterplot(x='preprocessed_comment_len',␣
↪y='toxicity_annotator_count',data=train_data)
plt.title('No. of Toxicity Annotators vs Comment Length (word level, simple␣
↪pre-processing)')

plt.subplot(122)
sns.scatterplot(x='comment_text_len',␣
↪y='toxicity_annotator_count',data=train_data)
plt.title('No. of Toxicity Annotators vs Comment Length (char level)')
```

```
plt.show()
```




### 1.3.1 Key Takeaways

- As we can see in both word level and charecter level, as length increases no of annotators for that comment decreases.

## 1.4 Identity Distribution

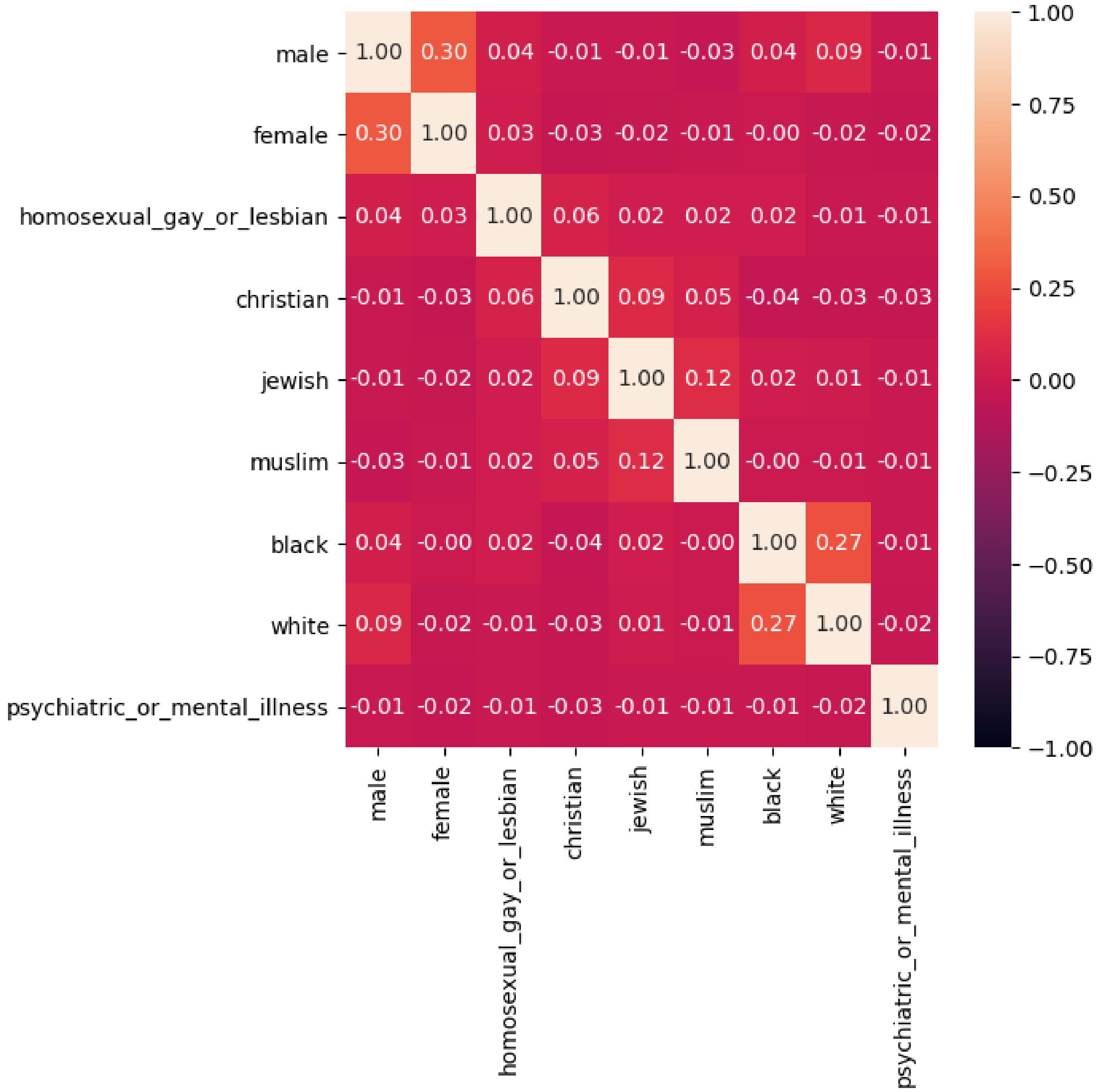
[35]:
```
for identity in selected_identities:
    counts = train_data[identity].sum()
    percentage = train_data[identity].sum() / train_data[identity].count() * 100
    print(f'{identity:<30}: {percentage:.2f}% , {counts}')
```

```
male                           : 2.46% , 44484
female                         : 2.96% , 53429
homosexual_gay_or_lesbian      : 0.61% , 10997
christian                      : 2.24% , 40423
jewish                         : 0.42% , 7651
muslim                         : 1.16% , 21006
black                          : 0.83% , 14901
white                          : 1.39% , 25082
psychiatric_or_mental_illness : 0.27% , 4889
```

[36]:
```
train['non_zero_selected_identity_counts'] = np.
 ↪count_nonzero(train_data[selected_identities], axis=1)
train.loc[train['identity_annotator_count'] == 0,␣
 ↪'non_zero_selected_identity_counts'] = np.NaN
selected_identity_corr = train_data.
 ↪loc[~train['non_zero_selected_identity_counts'].isna(), selected_identities].
 ↪corr()
```

[37]:

```
plt.style.use('default')

plt.figure(figsize=(6,6))
sns.heatmap(selected_identity_corr,
            vmin=-1, vmax=1, annot=True, fmt='.2f')
plt.show()
```

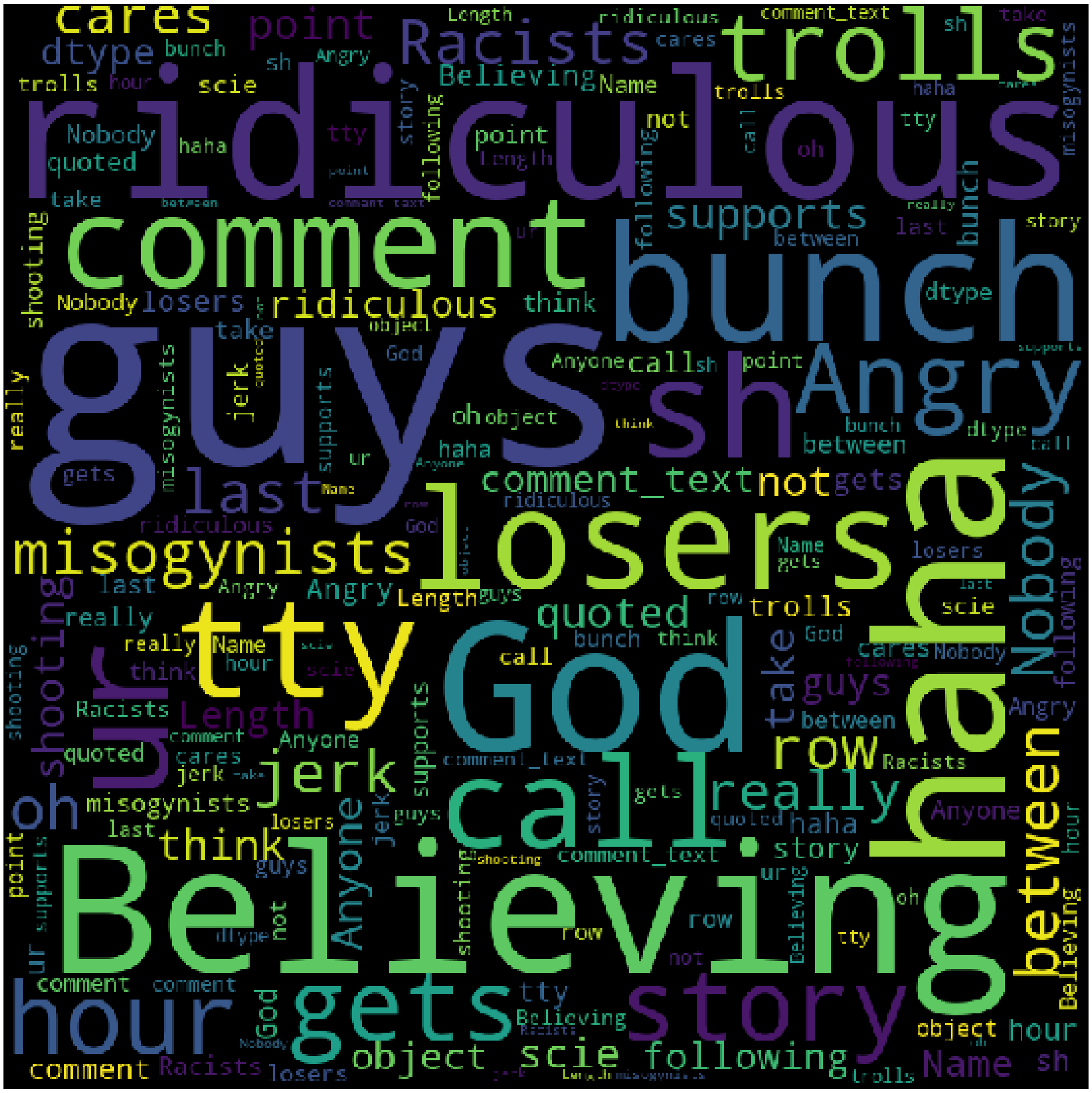


## 1.5 Word Cloud

[42]:
```
from wordcloud import WordCloud
def plot_word_cloud(comment,title):
    wordcloud = WordCloud(width = 800, height = 800,
                background_color ='black',
                stopwords = stopwords,
              min_font_size = 10,random_state=101,repeat=True).
 ↪generate(str(comment))

    # plot the WordCloud image
    plt.figure(figsize = (8, 8), facecolor = None)
    plt.title(title)
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.tight_layout(pad = 0)
    plt.show()
```

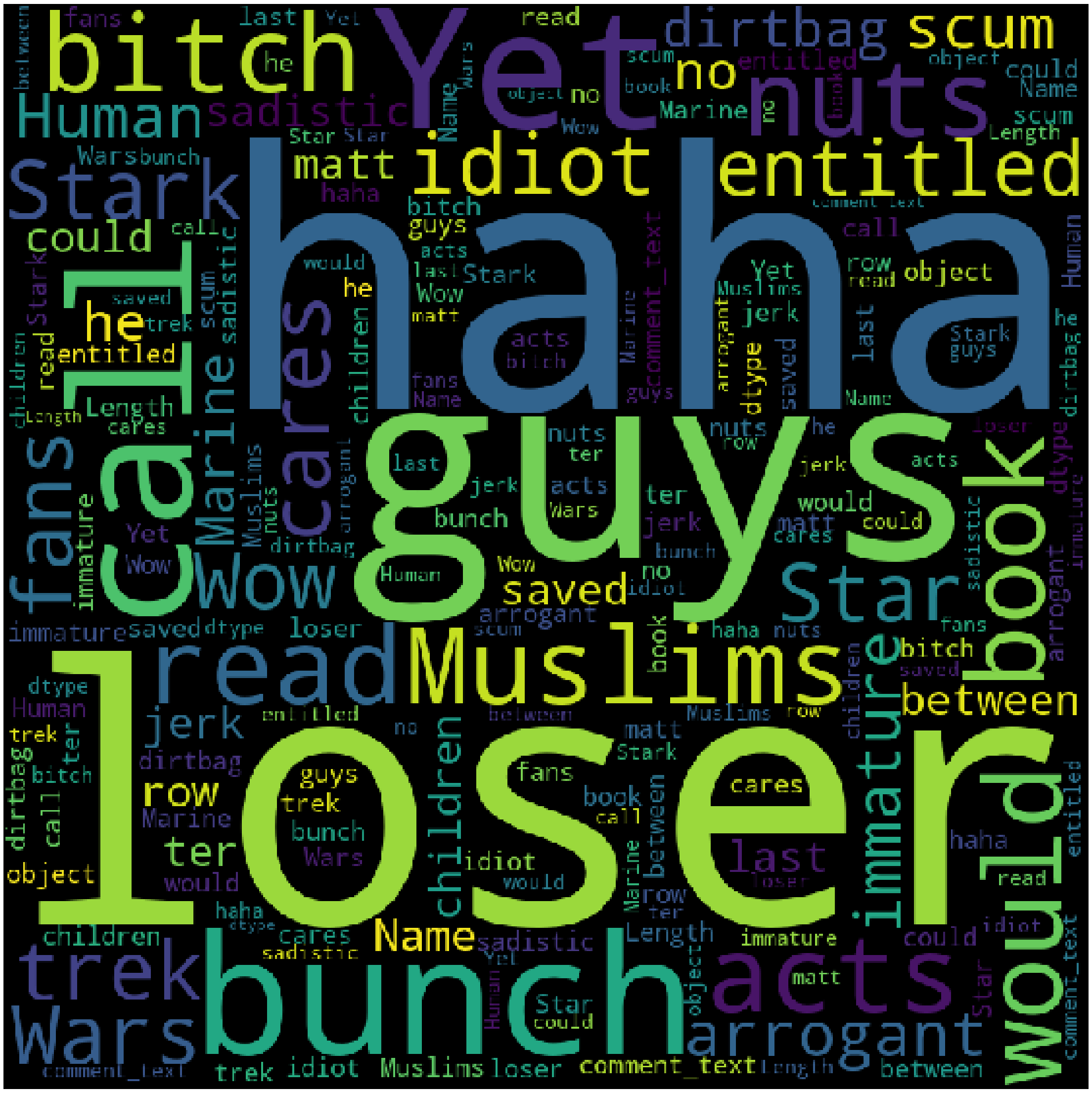
[43]:
```
plot_word_cloud(train_data.loc[train_data['target'] >= 0.5]['comment_text'],␣
 ↪'target value >= 0.5')
```

target value >= 0.5



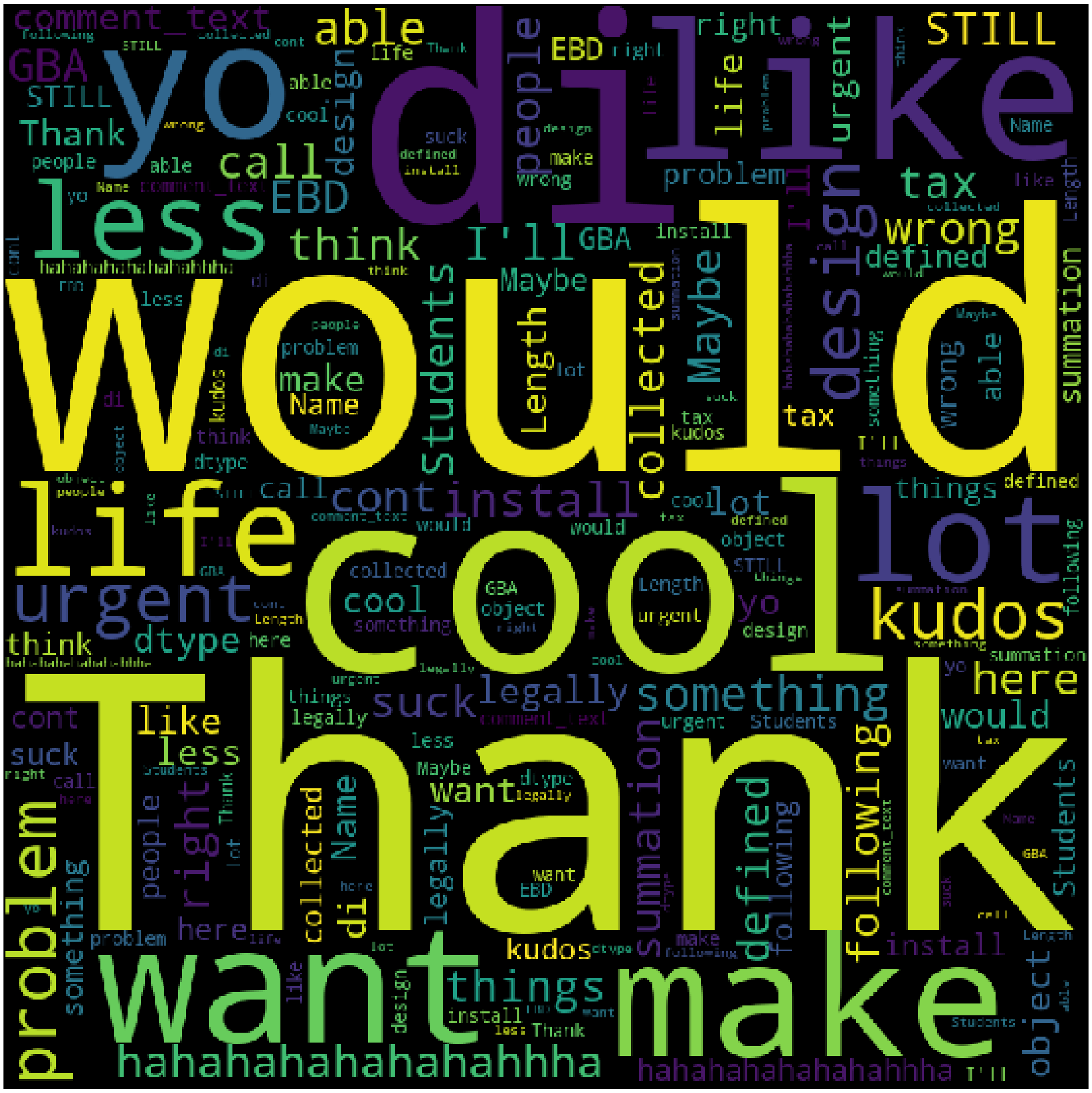

```
[44]: plot_word_cloud(train_data.loc[train_data['target'] >= 0.8]['comment_text'],
      'target value >= 0.8')
```

target value >= 0.8



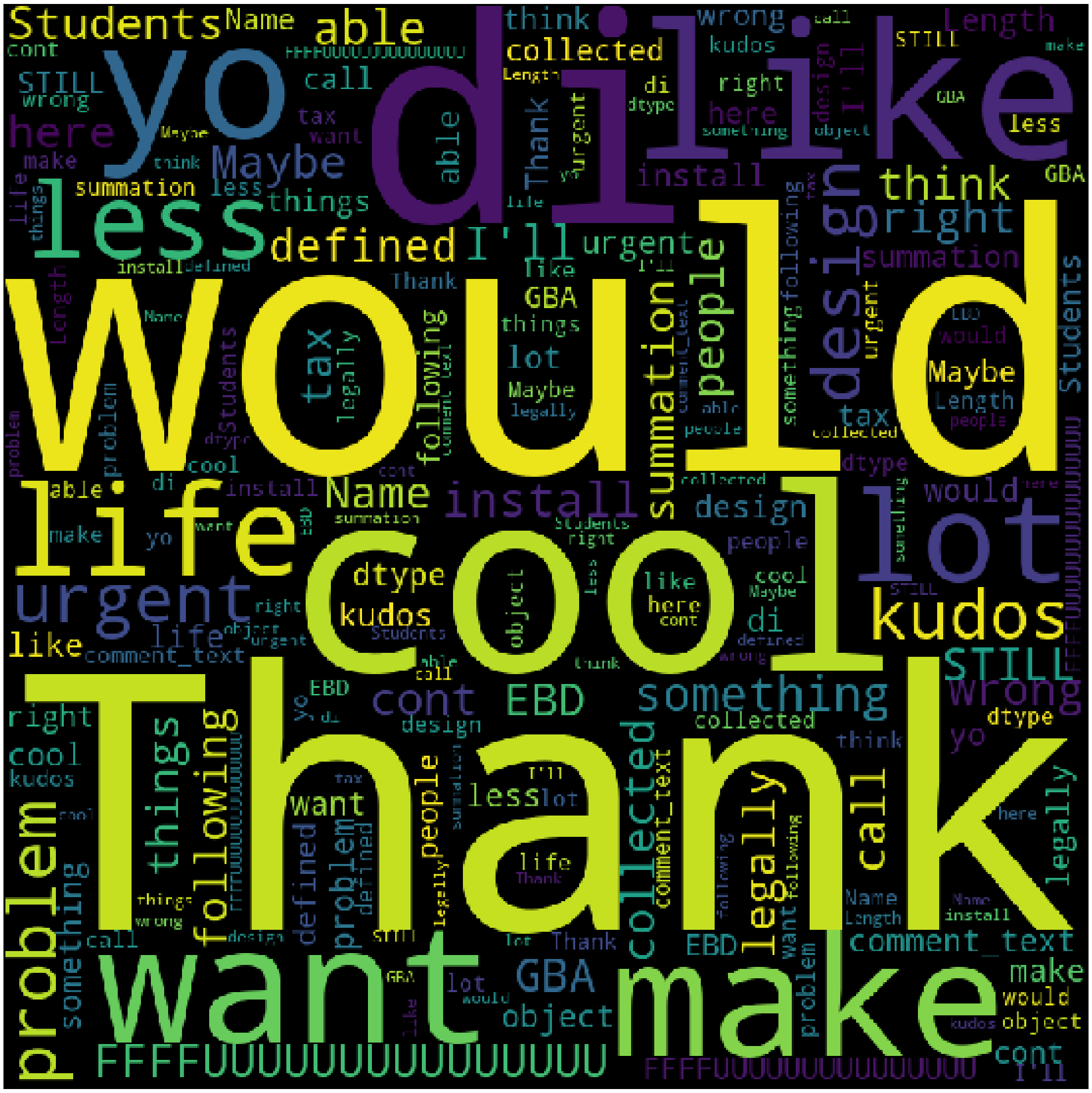

```
[46]: plot_word_cloud(train_data.loc[train_data['target'] < 0.5]['comment_text'],
       'target value < 0.5')
```

target value < 0.5




[47]:
```
plot_word_cloud(train_data.loc[train_data['target'] < 0.3]['comment_text'], 'target value < 0.3')
```

target value < 0.3




```
[15]: train_data_after_EDA =␣
       ↪train_data_filtered_bool[['id','comment_text','male','female','homosexual_gay_or_lesbian','
      print(train_data_after_EDA.shape)
```

(1804874, 12)

```
[16]: train_data_after_EDA.to_csv('train_data_after_EDA.csv')
```

# 2 Data Cleaning

[4]:
```
train_data = pd.read_csv('train_data_after_EDA.csv')
print(train_data.shape)
```

```
(1804874, 13)
```

[9]:
```
test_data = pd.read_csv('test/test.csv')
```

[10]:
```
def decontracted(phrase):
    # specific
    phrase = re.sub(r"won\'t", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    phrase = phrase.replace('\\r', ' ')
    phrase = phrase.replace('\\n', ' ')
    phrase = phrase.replace('\\"', ' ')
    phrase = re.sub('[^A-Za-z0-9]+', ' ', phrase)
    return phrase
```

[11]:
```
def cleanComments(df, column):
    cleaned_comments = []
    lmtzr = WordNetLemmatizer()
    ps = PorterStemmer()
    for sentence in tqdm(df[column]):
        sent = decontracted(sentence)
        sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords).
 ↪lower().strip()
#         https://stackoverflow.com/questions/50685343/
 ↪how-to-lemmatize-a-list-of-sentences
#         https://www.geeksforgeeks.org/python-stemming-words-with-nltk/
        sent = ' '.join(list(set(ps.stem(word) for word in␣
 ↪word_tokenize(sent))))
#     https://www.geeksforgeeks.org/python-lemmatization-with-nltk/
        sent = ' '.join(list(set(lmtzr.lemmatize(word) for word in␣
 ↪word_tokenize(sent))))
        sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
```

```
        cleaned_comments.append(sent)
    return cleaned_comments
```

[14]:
```
cleaned_comments = cleanComments(train_data, 'comment_text')
cleaned_comments_test = cleanComments(test_data, 'comment_text')
```

```
100%|      | 1804874/1804874 [30:16<00:00, 993.74it/s]
100%|      | 97320/97320 [01:37<00:00, 995.22it/s]
```

[15]:
```
train_data['comment_text'] = cleaned_comments
test_data['comment_text'] = cleaned_comments_test
```

[8]:
```
train_data.comment_text.values[20383]
```

[8]: 'forward littl ok peopl look pressur nicer'

[9]:
```
train_data.comment_text.values[20000]
```

[9]: 'comfort around god seen take statement firearm peopl mr treat cool requir class order mani serious publicli carri help becom would tool weapon start elrey seriou agre fashion u'

[10]:
```
train_data.shape
```

[10]: (1804874, 12)

[16]:
```
train_data.to_csv('train_data_cleaned.csv', index_label=False)
test_data.to_csv('test_data_cleaned.csv', index_label=False)
```

# 3 Train test split (80% - 20%)

**using stratified sampling to avoid bias while splitting data**

[6]:
```
train_data, validation_data = train_test_split(train_dataset, test_size=0.2,␣
 ↪stratify=train_data.target.values)
print(train_data.shape)
print(validation_data.shape)
```

```
(1443899, 12)
(360975, 12)
```

**Checking if test data is having approx same proportion of toxic comments compared to train data**

[19]:
```
neg_train = train_data[train_data['target'] == True]
neg_train.shape
```

[19]: (106438, 13)

[8]:
```
neg_validation = validation_data[validation_data['target'] == True]
neg_validation.shape
```

[8]: (21288, 12)

[9]:
```
train_data.to_csv('train_data_splited.csv', index_label=False)
validation_data.to_csv('validation_data_splitted.csv', index_label=False)
```

[10]:
```
train_data.head()
```

[10]:
```
              id                                        comment_text   male  \
153963    430435  could year trailer toilet vandal pee rememb ap…  False
1000606  5341284  counter nation proven support sanction practic…  False
1674253  6174396                                             tell   False
1030975  5377493    build pay mexico make wall state sanctuari citi   False
272931    576586  might outid believ holi pope involv question s…  False

         female  homosexual_gay_or_lesbian  christian  jewish  muslim  black  \
153963    False                      False      False   False   False  False
1000606   False                      False      False   False   False  False
1674253   False                      False      False   False   False  False
1030975   False                      False      False   False   False  False
272931    False                      False      False   False   False  False

         white  target  psychiatric_or_mental_illness
153963   False   False                          False
1000606  False   False                          False
1674253  False   False                          False
1030975  False   False                          False
272931   False   False                          False
```

[20]:
```
y_train = train_data['target']
y_validation = validation_data['target']
```

## 4 Defining Evaluation Metric

[26]:
```
SUBGROUP_AUC = 'subgroup_auc'
BPSN_AUC = 'bpsn_auc'  # stands for background positive, subgroup negative
BNSP_AUC = 'bnsp_auc'  # stands for background negative, subgroup positive
identity_columns = [
    'male', 'female', 'homosexual_gay_or_lesbian', 'christian', 'jewish',
    'muslim', 'black', 'white', 'psychiatric_or_mental_illness']
def compute_auc(y_true, y_pred):
```

```
    try:
        return metrics.roc_auc_score(y_true, y_pred)
    except ValueError:
        return np.nan

def compute_subgroup_auc(df, subgroup, label, model_name):
    subgroup_examples = df[df[subgroup]]
    return compute_auc(subgroup_examples[label], subgroup_examples[model_name])

def compute_bpsn_auc(df, subgroup, label, model_name):
    """Computes the AUC of the within-subgroup negative examples and the␣
↪background positive examples."""
    subgroup_negative_examples = df[df[subgroup] & ~df[label]]
    non_subgroup_positive_examples = df[~df[subgroup] & df[label]]
    examples = subgroup_negative_examples.append(non_subgroup_positive_examples)
    return compute_auc(examples[label], examples[model_name])

def compute_bnsp_auc(df, subgroup, label, model_name):
    """Computes the AUC of the within-subgroup positive examples and the␣
↪background negative examples."""
    subgroup_positive_examples = df[df[subgroup] & df[label]]
    non_subgroup_negative_examples = df[~df[subgroup] & ~df[label]]
    examples = subgroup_positive_examples.append(non_subgroup_negative_examples)
    return compute_auc(examples[label], examples[model_name])

def compute_bias_metrics_for_model(dataset,
                                   subgroups,
                                   model,
                                   label_col,
                                   include_asegs=False):
    """Computes per-subgroup metrics for all subgroups and one model."""
    records = []
    for subgroup in subgroups:
        record = {
            'subgroup': subgroup,
            'subgroup_size': len(dataset[dataset[subgroup]])
        }
        record[SUBGROUP_AUC] = compute_subgroup_auc(dataset, subgroup,␣
↪label_col, model)
        record[BPSN_AUC] = compute_bpsn_auc(dataset, subgroup, label_col, model)
        record[BNSP_AUC] = compute_bnsp_auc(dataset, subgroup, label_col, model)
        records.append(record)
    return pd.DataFrame(records).sort_values('subgroup_auc', ascending=True)


# bias_metrics_df
```

[27]:
```python
def calculate_overall_auc(df, model_name):
    true_labels = df['target']
    predicted_labels = df[model_name]
    return metrics.roc_auc_score(true_labels, predicted_labels)

def power_mean(series, p):
    total = sum(np.power(series, p))
    return np.power(total / len(series), 1 / p)

def get_final_metric(bias_df, overall_auc, POWER=-5, OVERALL_MODEL_WEIGHT=0.25):
    bias_score = np.average([
        power_mean(bias_df[SUBGROUP_AUC], POWER),
        power_mean(bias_df[BPSN_AUC], POWER),
        power_mean(bias_df[BNSP_AUC], POWER)
    ])
    return (OVERALL_MODEL_WEIGHT * overall_auc) + ((1 - OVERALL_MODEL_WEIGHT) *␣
↪bias_score)
def get_metric_value(validate_df, identity_columns, MODEL_NAME):
    bias_metrics_df = compute_bias_metrics_for_model(validate_df,␣
↪identity_columns, MODEL_NAME, 'target')
    return get_final_metric(bias_metrics_df, calculate_overall_auc(validate_df,␣
↪MODEL_NAME))
```

# 5 Machine Learning Models

## 5.1 Vectorizing Comment Text

[21]:
```python
def vectorizeData(train, validation, vectorizing_method, dim, n_gram_range):
    if vectorizing_method == 'bow':
        bow_vectorizer = CountVectorizer(ngram_range=n_gram_range, min_df=3,␣
↪max_df=0.9, max_features=dim)
        train_data_bow = bow_vectorizer.fit_transform(train)
        validation_data_bow = bow_vectorizer.transform(validation)
        return train_data_bow, validation_data_bow
    if vectorizing_method == 'tfidf':
        tfidf_vectorizer = TfidfVectorizer(ngram_range=n_gram_range, min_df=3,␣
↪max_df=0.9, max_features=dim)
        train_data_tfidf = tfidf_vectorizer.fit_transform(train)
        validation_data_tfidf = tfidf_vectorizer.transform(validation)
        return train_data_tfidf, validation_data_tfidf
    if vectorizing_method == 'w2v':
        with open('glove_vectors', 'rb') as f:
            model = pickle.load(f)
            glove_words = set(model.keys())
```

```
        train_data_avg_w2v = []# the avg-w2v for each sentence/review is stored␣
↪in this list
        validation_data_avg_w2v = []
        for i in range(1,3):
            if i == 1:
                data = train
            if i == 2:
                data = validation
            for sentence in tqdm(data): # for each review/sentence
                vector = np.zeros(300) # as word vectors are of zero length
                cnt_words =0; # num of words with a valid vector in the␣
↪sentence/review
                for word in sentence.split(): # for each word in a review/
↪sentence
                    if word in glove_words:
                        vector += model[word]
                        cnt_words += 1
                if cnt_words != 0:
                    vector /= cnt_words
                    if i == 1:
                        train_data_avg_w2v.append(vector)
                    if i == 2:
                        validation_data_avg_w2v.append(vector)
                else:
                    if i == 1:
                        train_data_avg_w2v.append(vector)
                    if i == 2:
                        validation_data_avg_w2v.append(vector)
        return train_data_avg_w2v, validation_data_avg_w2v
```

[22]:
```
train_data['comment_text'] = train_data.comment_text.fillna('')
test_data['comment_text'] = test_data.comment_text.fillna('')
validation_data['comment_text'] = validation_data.comment_text.fillna('')
```

**Considering 25000, 15000, 10000 dimentions**

**25000 top words in bow and tfidf**

[14]:
```
train_comment_bow_25000, validation_comment_bow_25000 =␣
↪vectorizeData(train_data['comment_text'], validation_data['comment_text'],␣
↪'bow', 25000, (1,1))
print(f'train_bow : {train_comment_bow_25000.shape}')
print(f'validation_bow : {validation_comment_bow_25000.shape}')
```

```
train_comment_tfidf_25000, validation_comment_tfidf_25000 =␣
↪vectorizeData(train_data['comment_text'], validation_data['comment_text'],␣
↪'tfidf', 25000, (1,1))
print(f'train_tfidf : {train_comment_tfidf_25000.shape}')
print(f'validation_tfidf : {validation_comment_tfidf_25000.shape}')
```

```
train_bow : (1443899, 25000)
validation_bow : (360975, 25000)
train_tfidf : (1443899, 25000)
validation_tfidf : (360975, 25000)
```

**15000 top words in bow and tfidf**

[23]:
```
train_comment_bow_15000, validation_comment_bow_15000 =␣
↪vectorizeData(train_data['comment_text'], validation_data['comment_text'],␣
↪'bow', 15000, (1,1))
print(f'train_bow : {train_comment_bow_15000.shape}')
print(f'validation_bow : {validation_comment_bow_15000.shape}')

train_comment_tfidf_15000, validation_comment_tfidf_15000 =␣
↪vectorizeData(train_data['comment_text'], validation_data['comment_text'],␣
↪'tfidf', 15000, (1,1))
print(f'train_tfidf : {train_comment_tfidf_15000.shape}')
print(f'validation_tfidf : {validation_comment_tfidf_15000.shape}')

train_comment_tfidf_15000, test_comment_tfidf_15000 =␣
↪vectorizeData(train_data['comment_text'], test_data['comment_text'],␣
↪'tfidf', 15000, (1,1))
print(f'train_tfidf : {train_comment_tfidf_15000.shape}')
print(f'validation_tfidf : {test_comment_tfidf_15000.shape}')
```

```
train_tfidf : (1804874, 15000)
validation_tfidf : (97320, 15000)
```

**10000 top words in bow and tfidf**

[16]:
```
train_comment_bow_10000, validation_comment_bow_10000 =␣
↪vectorizeData(train_data['comment_text'], validation_data['comment_text'],␣
↪'bow', 10000, (1,1))
print(f'train_bow : {train_comment_bow_10000.shape}')
print(f'validation_bow : {validation_comment_bow_10000.shape}')

train_comment_tfidf_10000, validation_comment_tfidf_10000 =␣
↪vectorizeData(train_data['comment_text'], validation_data['comment_text'],␣
↪'tfidf', 10000, (1,1))
print(f'train_tfidf : {train_comment_tfidf_10000.shape}')
print(f'validation_tfidf : {validation_comment_tfidf_10000.shape}')
```

```
train_bow : (1443899, 10000)
validation_bow : (360975, 10000)
train_tfidf : (1443899, 10000)
validation_tfidf : (360975, 10000)
```

**W2V representation**

```
[ ]: train_comment_w2v, validation_comment_w2v =␣
      ↪vectorizeData(train_data['comment_text'], validation_data['comment_text'],␣
      ↪'w2v', None, None)
     print(f'train_w2v : {len(train_comment_w2v), len(train_comment_w2v[0])}')
     print(f'validation_w2v : {len(validation_comment_w2v),␣
      ↪len(validation_comment_w2v[0])}')
     train_comment_w2v = np.asarray(train_comment_w2v).
      ↪reshape(len(train_comment_w2v), len(train_comment_w2v[0]))
     validation_comment_w2v = np.asarray(validation_comment_w2v).
      ↪reshape(len(validation_comment_w2v), len(validation_comment_w2v[0]))
```

```
[16]: scipy.sparse.save_npz('train_comment_bow_25000.npz',train_comment_bow_25000)
      scipy.sparse.save_npz('validation_comment_bow_25000.
       ↪npz',validation_comment_bow_25000)
      scipy.sparse.save_npz('train_comment_tfidf_25000.npz',train_comment_tfidf_25000)
      scipy.sparse.save_npz('validation_comment_tfidf_25000.
       ↪npz',validation_comment_tfidf_25000)

      scipy.sparse.save_npz('train_comment_bow_15000.npz',train_comment_bow_15000)
      scipy.sparse.save_npz('validation_comment_bow_15000.
       ↪npz',validation_comment_bow_15000)
      scipy.sparse.save_npz('train_comment_tfidf_15000.npz',train_comment_tfidf_15000)
      scipy.sparse.save_npz('validation_comment_tfidf_15000.
       ↪npz',validation_comment_tfidf_15000)

      scipy.sparse.save_npz('train_comment_bow_10000.npz',train_comment_bow_10000)
      scipy.sparse.save_npz('validation_comment_bow_10000.
       ↪npz',validation_comment_bow_10000)
      scipy.sparse.save_npz('train_comment_tfidf_10000.npz',train_comment_tfidf_10000)
      scipy.sparse.save_npz('validation_comment_tfidf_10000.
       ↪npz',validation_comment_tfidf_10000)


      np.save('train_comment_w2v',train_comment_w2v)
      np.save('validation_comment_w2v',validation_comment_w2v)
```

```
[24]: #https://gist.github.com/shaypal5/94c53d765083101efc0240d776a23823
      def plot_confusion_matrix(confusion_matrix, class_names, figsize = (6,4),␣
       ↪fontsize=14):
          df_cm = pd.DataFrame(
```

```
        confusion_matrix,index=class_names, columns=class_names
    )
    fig = plt.figure(figsize=figsize)
    heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0,␣
↪ha='right', fontsize=fontsize)
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45,␣
↪ha='right', fontsize=fontsize)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
```

[25]:
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr

# (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
# print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold",␣
↪np.round(t,3))
def predict_with_best_t(proba, tpr, fpr, threshould):
    t = threshould[np.argmax(tpr*(1-fpr))]
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

Models we are going to try

Naive Bayes

Logistic Regression (SGD with 'log' loss)

SVM (SGD with 'hinge' loss)

XG-Boost

TabdomForestClassifier

Stacking above based on confusion matrix

### 5.1.1 Naive Bayes

**Considering BOW featues**

**25000 features**

[90]:
```
alpha = [1e-09, 1e-07, 1e-05, 1e-03, 1, 10]
train_auc_list = []
```

```
validation_auc_list = []
names = []
for param in tqdm(alpha):
    MODEL_NAME = f'NB-BOW_25k_{param}'
    clf = MultinomialNB(alpha=param)
    clf.fit(train_comment_bow_25000, y_train)
    predicted_train = clf.predict_proba(train_comment_bow_25000)[:,1]
    predicted_validation = clf.predict_proba(validation_comment_bow_25000)[:,1]
    train_data[MODEL_NAME] = predicted_train
    validation_data[MODEL_NAME] = predicted_validation

    train_auc_list.append(get_metric_value(train_data, identity_columns,␣
↪MODEL_NAME))
    validation_auc_list.append(get_metric_value(validation_data,␣
↪identity_columns, MODEL_NAME))
    names.append(MODEL_NAME)
```

```
100%|          | 6/6 [01:22<00:00, 13.68s/it]
```

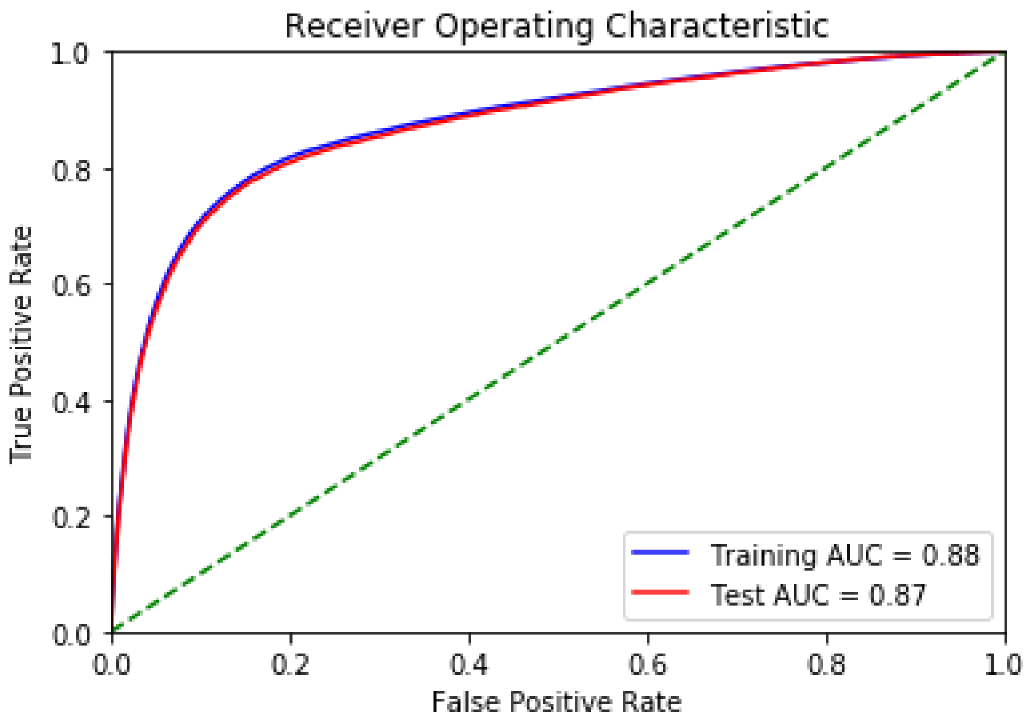
[91]:
```
pd.DataFrame({'name':names, 'train-score':train_auc_list, 'test-score':
↪validation_auc_list}).sort_values(by=['test-score'])
```

[91]:

| | name | train-score | test-score |
|---|---|---|---|
| 0 | NB-BOW_25k_1e-09 | 0.858525 | 0.819695 |
| 1 | NB-BOW_25k_1e-07 | 0.858495 | 0.820695 |
| 2 | NB-BOW_25k_1e-05 | 0.858352 | 0.822888 |
| 3 | NB-BOW_25k_0.001 | 0.857813 | 0.826576 |
| 5 | NB-BOW_25k_10 | 0.842939 | 0.832772 |
| 4 | NB-BOW_25k_1 | 0.853244 | 0.836316 |

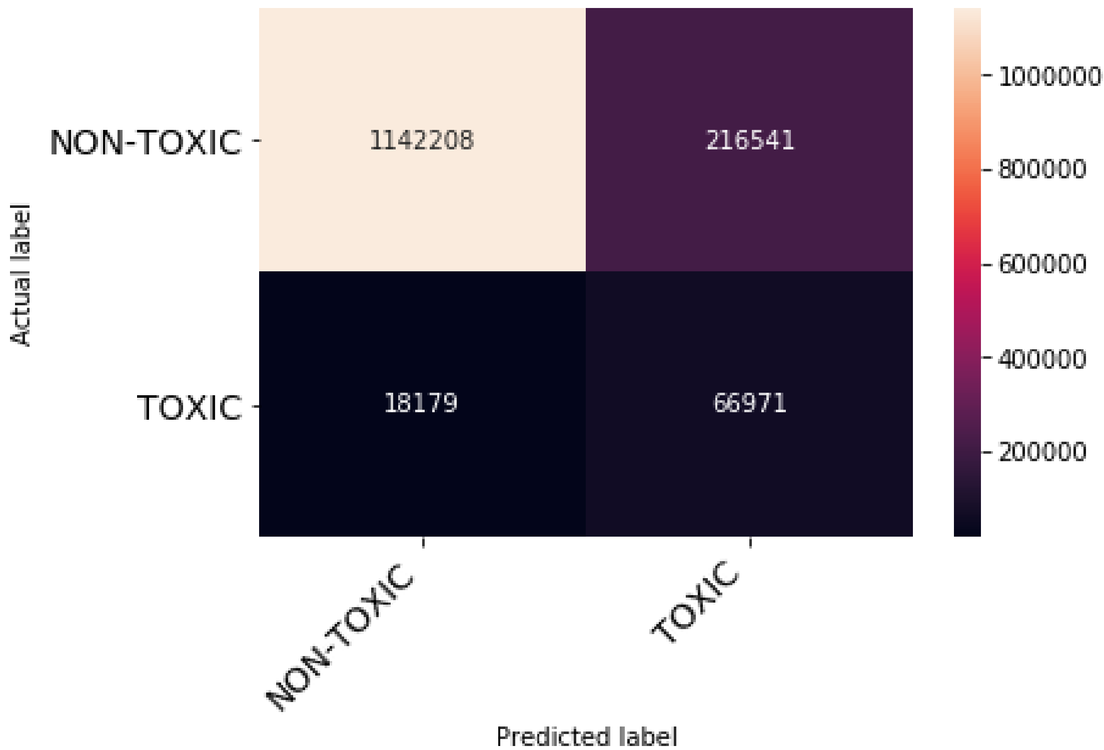
[92]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
↪predicted_validation)

roc_auc_train = auc(fpr_train, tpr_train)
roc_auc_test = auc(fpr_test, tpr_test)


plt.title('Receiver Operating Characteristic')

plt.plot(fpr_train, tpr_train, 'b', label = 'Training AUC = %0.2f' %␣
↪roc_auc_train)
plt.plot(fpr_test, tpr_test, 'r', label = 'Test AUC = %0.2f' % roc_auc_test)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
```

```
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



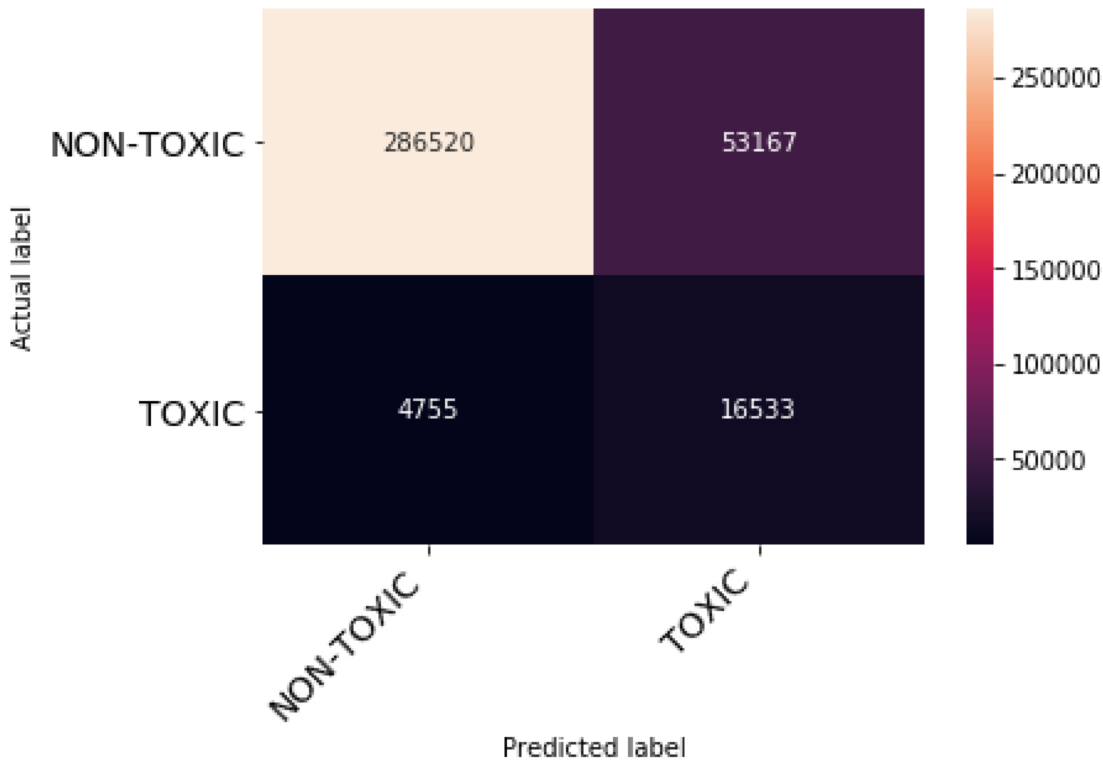

[94]:
```
pred_train =␣
↪predict_with_best_t(predicted_train,tpr_train,fpr_train,threshold_train)
cm = confusion_matrix(y_train, pred_train)
print("\tTRAIN DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	TRAIN DATA CONFUSION MATRIX
```




[96]:
```
pred_test =␣
↪predict_with_best_t(predicted_validation,tpr_test,fpr_test,threshold_test)
cm = confusion_matrix(y_validation, pred_test)
print("\ttest DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
        test DATA CONFUSION MATRIX
```




**15000 features**

[97]:
```
alpha = [1e-09, 1e-07, 1e-05, 1e-03, 1, 10]
train_auc_list = []
validation_auc_list = []
names = []
for param in tqdm(alpha):
    MODEL_NAME = f'NB-BOW_15k_{param}'
    clf = MultinomialNB(alpha=param)
    clf.fit(train_comment_bow_15000, y_train)
    predicted_train = clf.predict_proba(train_comment_bow_15000)[:,1]
    predicted_validation = clf.predict_proba(validation_comment_bow_15000)[:,1]

    train_data[MODEL_NAME] = predicted_train
    validation_data[MODEL_NAME] = predicted_validation

    train_auc_list.append(get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME))
    validation_auc_list.append(get_metric_value(validation_data,␣
 ↪identity_columns, MODEL_NAME))
    names.append(MODEL_NAME)
```

```
100%|          | 6/6 [01:10<00:00, 11.81s/it]
```

[98]:

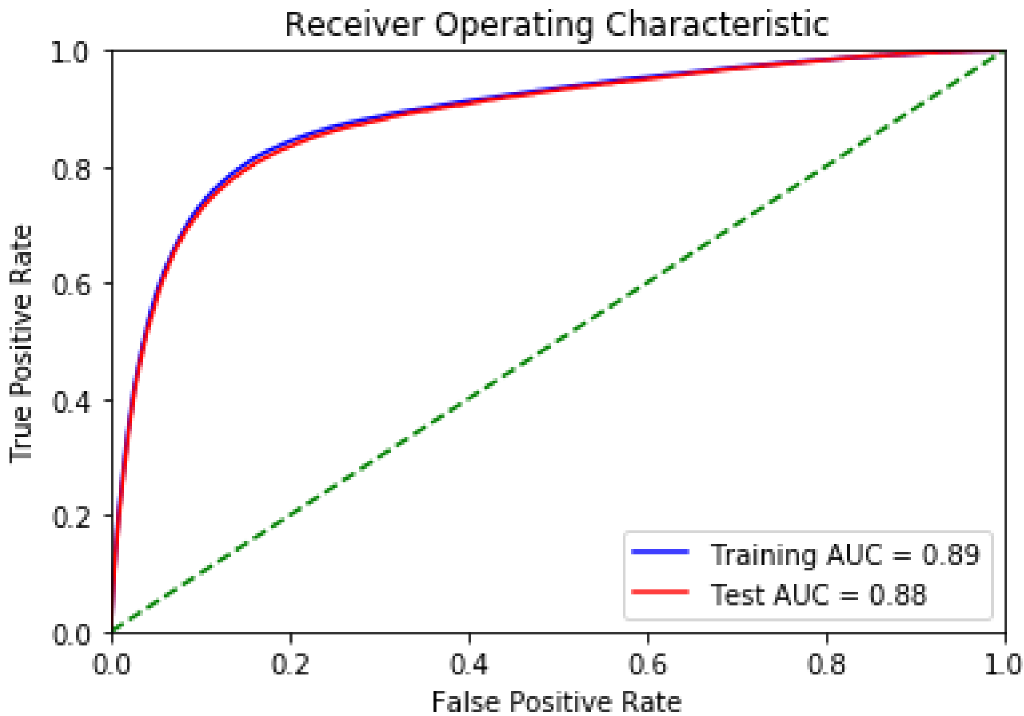
```python
pd.DataFrame({'name':names, 'train-score':train_auc_list, 'test-score':
 ↪validation_auc_list}).sort_values(by=['test-score'])
```

[98]:

| | name | train-score | test-score |
|---|---|---|---|
| 0 | NB-BOW_15k_1e-09 | 0.849862 | 0.831703 |
| 1 | NB-BOW_15k_1e-07 | 0.849859 | 0.831952 |
| 2 | NB-BOW_15k_1e-05 | 0.849841 | 0.832404 |
| 3 | NB-BOW_15k_0.001 | 0.849772 | 0.832974 |
| 4 | NB-BOW_15k_1 | 0.849333 | 0.835220 |
| 5 | NB-BOW_15k_10 | 0.846040 | 0.836102 |

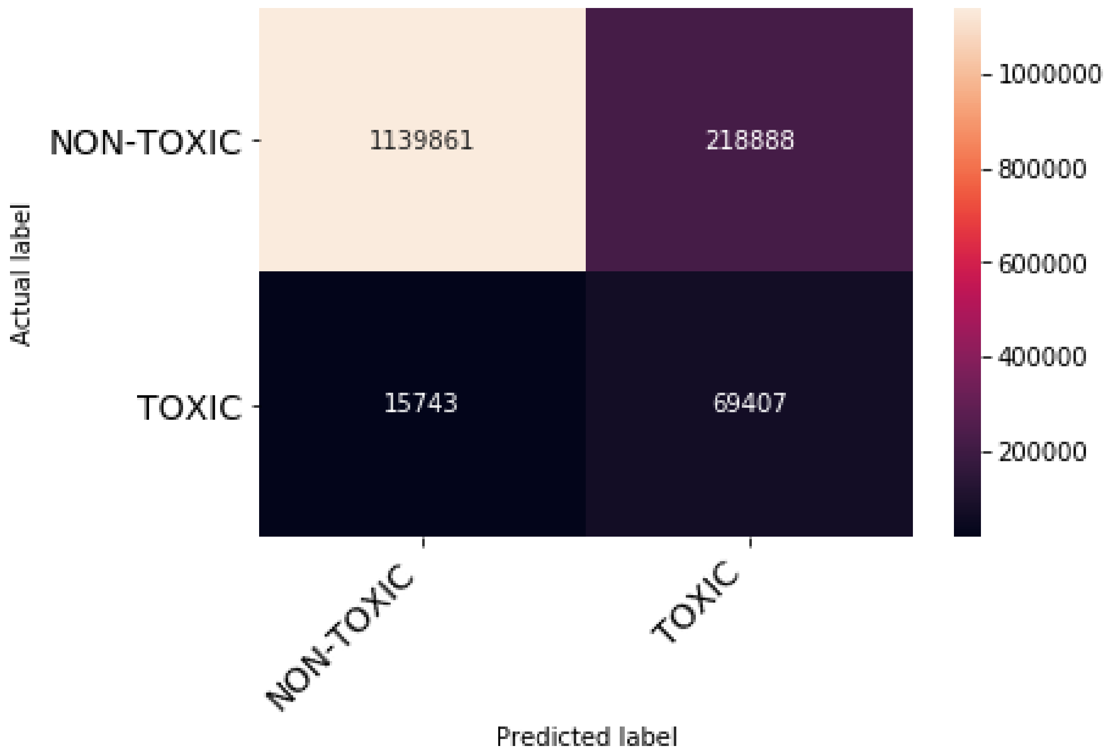
[99]:

```python
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
 ↪predicted_validation)

roc_auc_train = auc(fpr_train, tpr_train)
roc_auc_test = auc(fpr_test, tpr_test)


plt.title('Receiver Operating Characteristic')

plt.plot(fpr_train, tpr_train, 'b', label = 'Training AUC = %0.2f' %␣
 ↪roc_auc_train)
plt.plot(fpr_test, tpr_test, 'r', label = 'Test AUC = %0.2f' % roc_auc_test)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



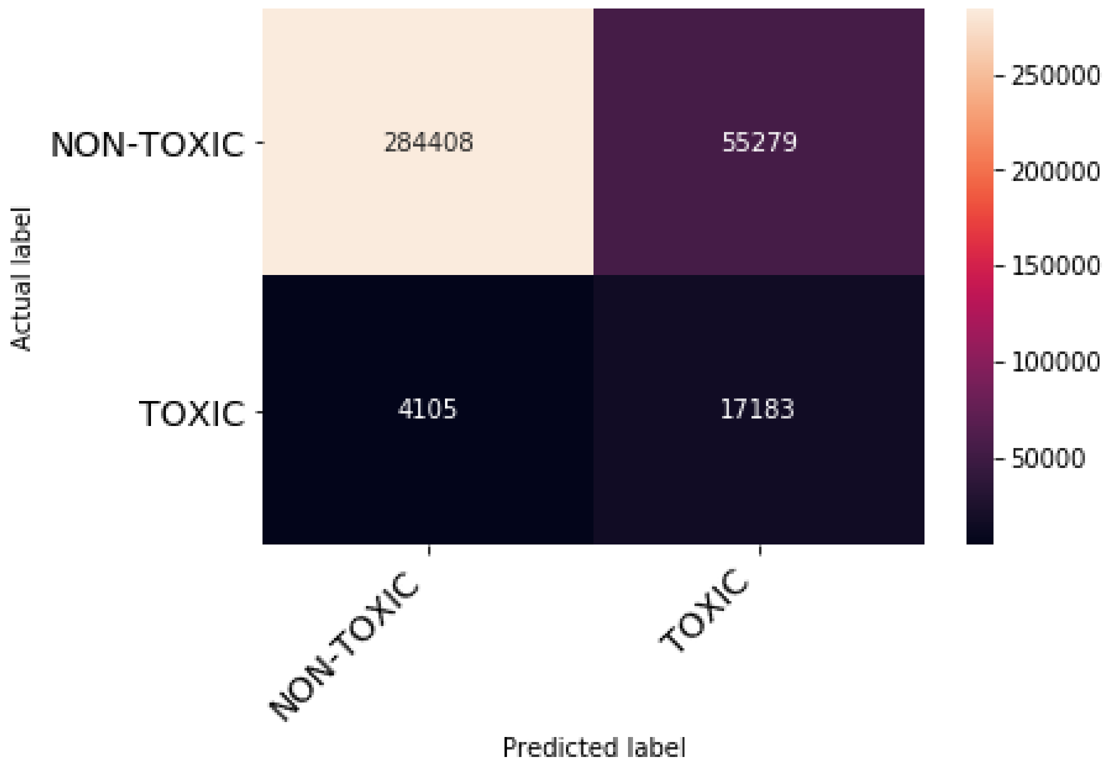

```
[100]: pred_train =␣
        ↪predict_with_best_t(predicted_train,tpr_train,fpr_train,threshold_train)
       cm = confusion_matrix(y_train, pred_train)
       print("\tTRAIN DATA CONFUSION MATRIX")
       plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
        TRAIN DATA CONFUSION MATRIX
```




```
[101]: pred_test =␣
        ↪predict_with_best_t(predicted_validation,tpr_test,fpr_test,threshold_test)
       cm = confusion_matrix(y_validation, pred_test)
       print("\ttest DATA CONFUSION MATRIX")
       plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
        test DATA CONFUSION MATRIX
```




**10000 features**

[102]:
```
alpha = [1e-09, 1e-07, 1e-05, 1e-03, 1, 10]
train_auc_list = []
validation_auc_list = []
names = []
for param in tqdm(alpha):
    MODEL_NAME = f'NB-BOW_10k_{param}'
    clf = MultinomialNB(alpha=param)
    clf.fit(train_comment_bow_10000, y_train)
    predicted_train = clf.predict_proba(train_comment_bow_10000)[:,1]
    predicted_validation = clf.predict_proba(validation_comment_bow_10000)[:,1]

    train_data[MODEL_NAME] = predicted_train
    validation_data[MODEL_NAME] = predicted_validation

    train_auc_list.append(get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME))
    validation_auc_list.append(get_metric_value(validation_data,␣
 ↪identity_columns, MODEL_NAME))
    names.append(MODEL_NAME)
```

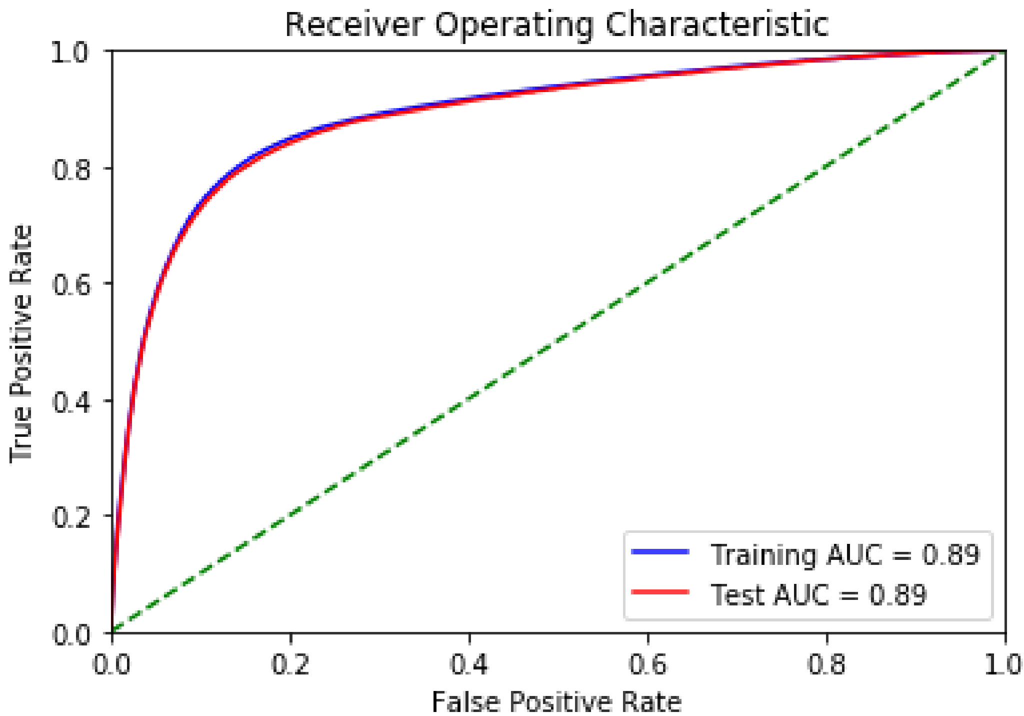
```
100%|          | 6/6 [01:10<00:00, 11.75s/it]
```

[103]:
```
pd.DataFrame({'name':names, 'train-score':train_auc_list, 'test-score':
 ↪validation_auc_list}).sort_values(by=['test-score'])
```

[103]:

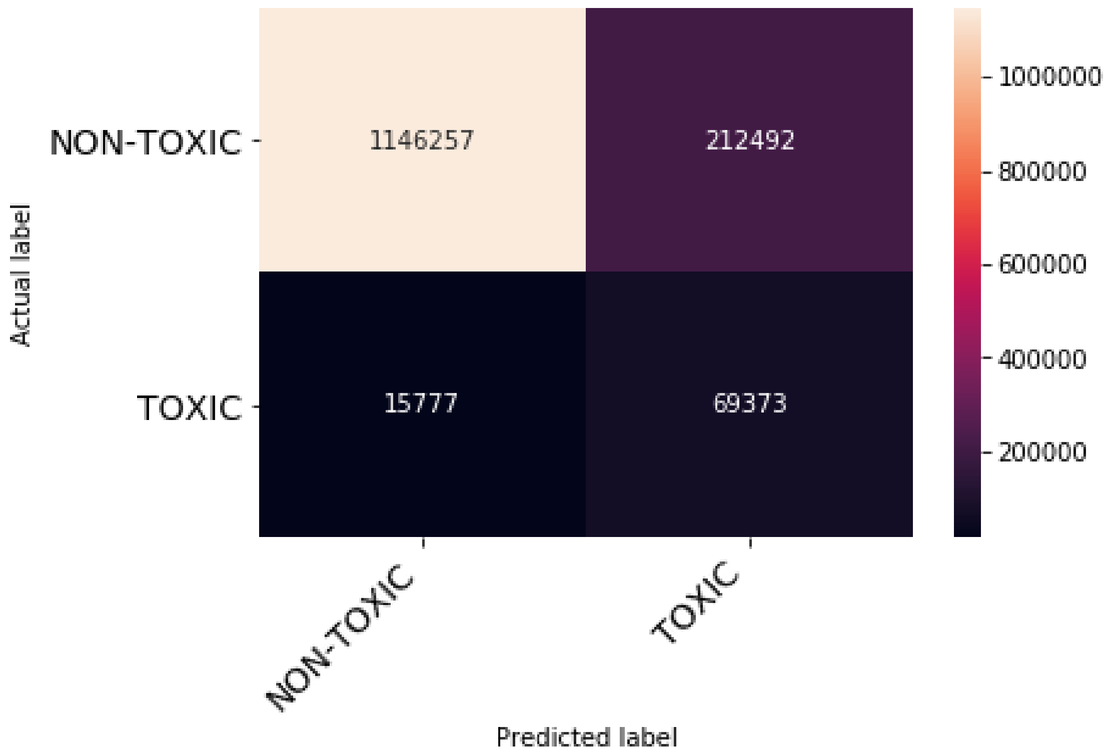
| | name | train-score | test-score |
|---|---|---|---|
| 0 | NB-BOW_10k_1e-09 | 0.844828 | 0.832513 |
| 1 | NB-BOW_10k_1e-07 | 0.844828 | 0.832518 |
| 2 | NB-BOW_10k_1e-05 | 0.844827 | 0.832531 |
| 3 | NB-BOW_10k_0.001 | 0.844824 | 0.832544 |
| 4 | NB-BOW_10k_1 | 0.844998 | 0.833089 |
| 5 | NB-BOW_10k_10 | 0.844491 | 0.834385 |

[104]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
 ↪predicted_validation)

roc_auc_train = auc(fpr_train, tpr_train)
roc_auc_test = auc(fpr_test, tpr_test)


plt.title('Receiver Operating Characteristic')

plt.plot(fpr_train, tpr_train, 'b', label = 'Training AUC = %0.2f' %␣
 ↪roc_auc_train)
plt.plot(fpr_test, tpr_test, 'r', label = 'Test AUC = %0.2f' % roc_auc_test)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



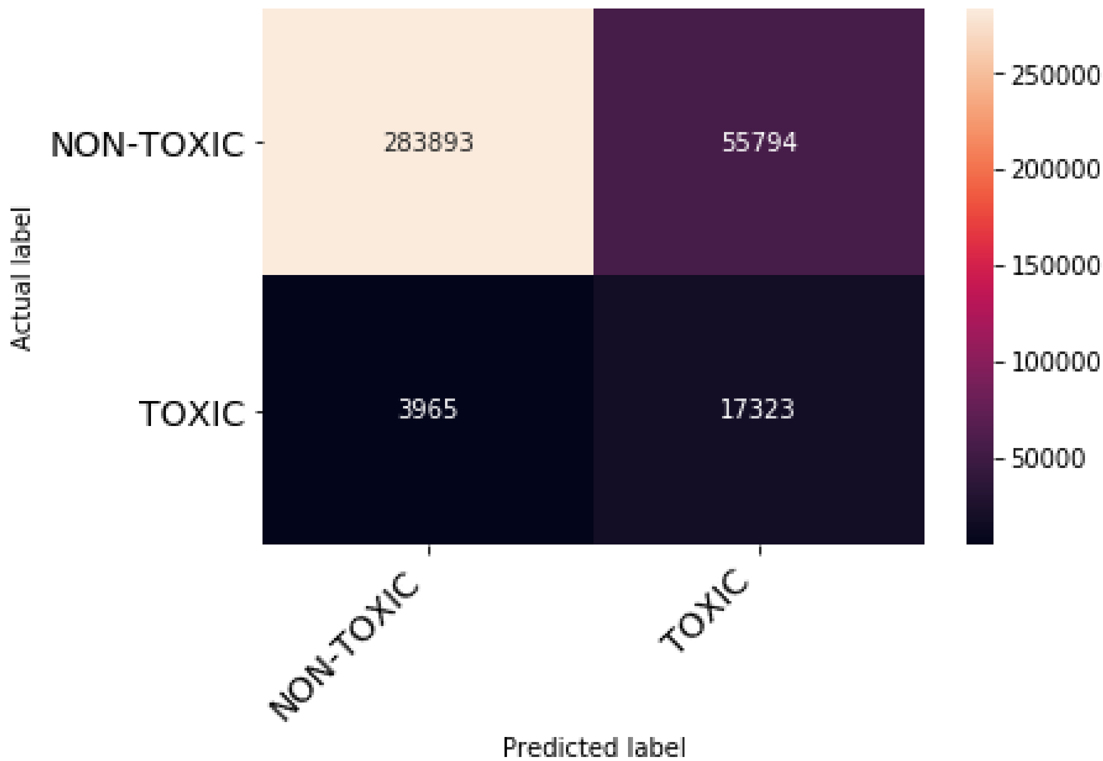

[105]:
```
pred_train =␣
 ↪predict_with_best_t(predicted_train,tpr_train,fpr_train,threshold_train)
cm = confusion_matrix(y_train, pred_train)
print("\tTRAIN DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	TRAIN DATA CONFUSION MATRIX
```




[106]:
```
pred_test =␣
 ↪predict_with_best_t(predicted_validation,tpr_test,fpr_test,threshold_test)
cm = confusion_matrix(y_validation, pred_test)
print("\ttest DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	test DATA CONFUSION MATRIX
```




**Considering TFIDF**

**25000 features**

[107]:
```
alpha = [1e-09, 1e-07, 1e-05, 1e-03, 1, 10]
train_auc_list = []
validation_auc_list = []
names = []
for param in tqdm(alpha):
    MODEL_NAME = f'NB-tfidf_25k_{param}'
    clf = MultinomialNB(alpha=param)
    clf.fit(train_comment_tfidf_25000, y_train)
    predicted_train = clf.predict_proba(train_comment_tfidf_25000)[:,1]
    predicted_validation = clf.predict_proba(validation_comment_tfidf_25000)[:
 ↪,1]

    train_data[MODEL_NAME] = predicted_train
    validation_data[MODEL_NAME] = predicted_validation

    train_auc_list.append(get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME))
```

```
    validation_auc_list.append(get_metric_value(validation_data,␣
↪identity_columns, MODEL_NAME))
    names.append(MODEL_NAME)
```

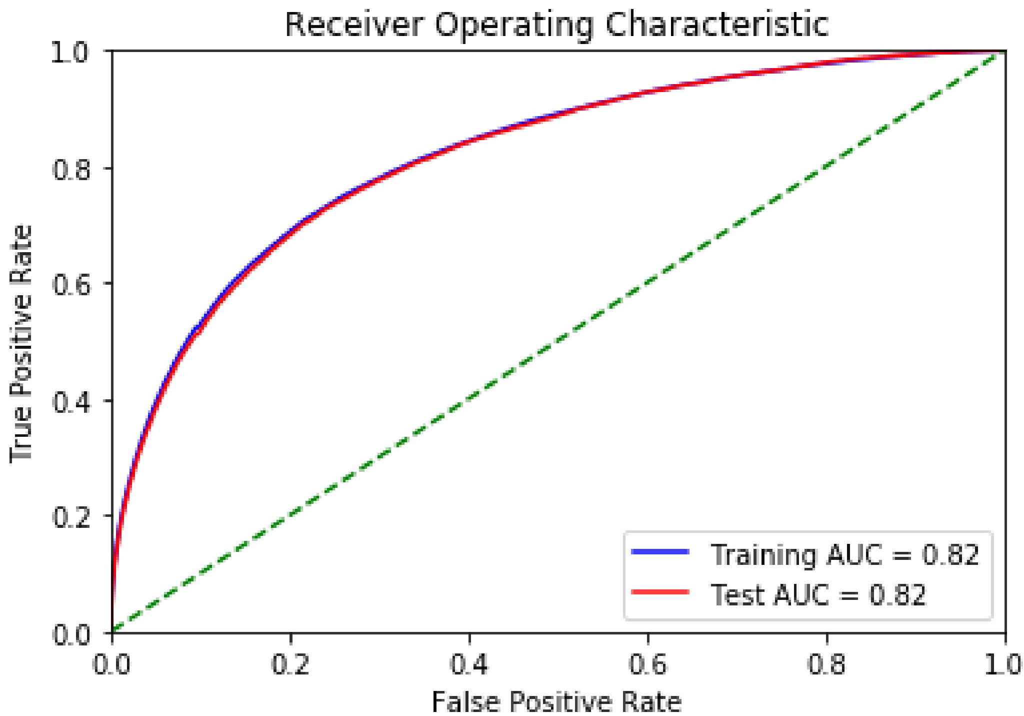
```
100%|      | 6/6 [01:09<00:00, 11.65s/it]
```

[108]:
```
pd.DataFrame({'name':names, 'train-score':train_auc_list, 'test-score':
 ↪validation_auc_list}).sort_values(by=['test-score'])
```

[108]:

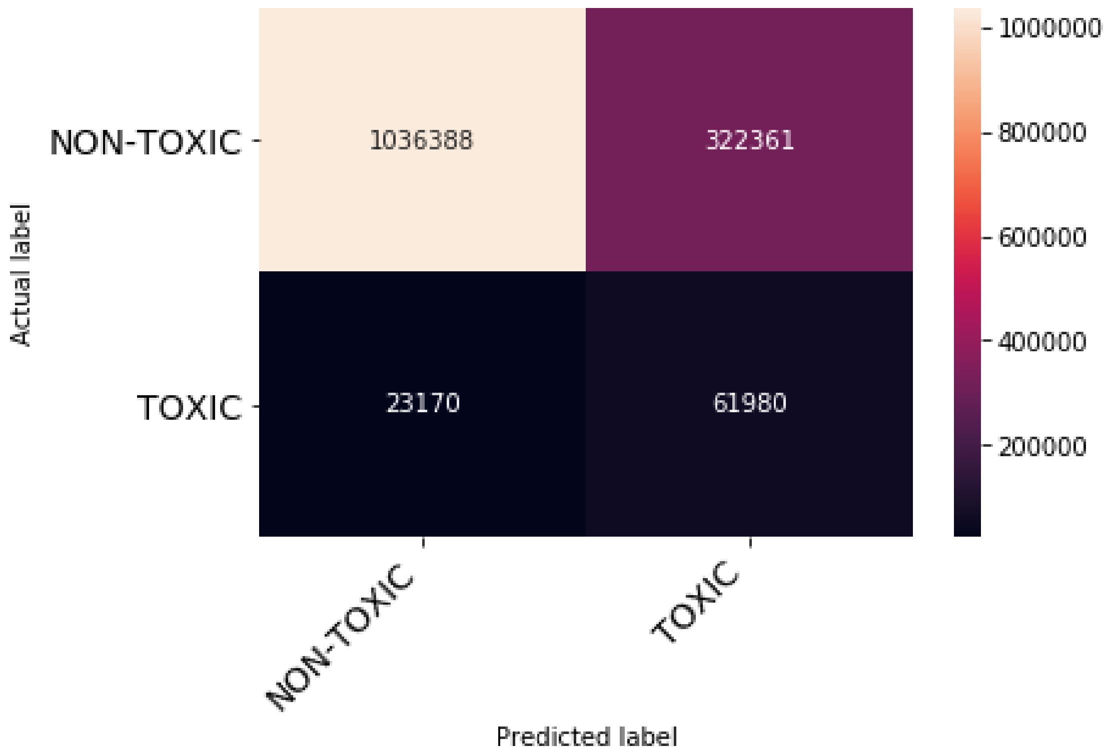
| | name | train-score | test-score |
|---|---|---|---|
| 5 | NB-tfidf_25k_10 | 0.801803 | 0.794206 |
| 0 | NB-tfidf_25k_1e-09 | 0.882603 | 0.839547 |
| 1 | NB-tfidf_25k_1e-07 | 0.882601 | 0.839878 |
| 2 | NB-tfidf_25k_1e-05 | 0.882580 | 0.841083 |
| 3 | NB-tfidf_25k_0.001 | 0.882403 | 0.844974 |
| 4 | NB-tfidf_25k_1 | 0.876217 | 0.860478 |

[109]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
 ↪predicted_validation)

roc_auc_train = auc(fpr_train, tpr_train)
roc_auc_test = auc(fpr_test, tpr_test)


plt.title('Receiver Operating Characteristic')

plt.plot(fpr_train, tpr_train, 'b', label = 'Training AUC = %0.2f' %␣
 ↪roc_auc_train)
plt.plot(fpr_test, tpr_test, 'r', label = 'Test AUC = %0.2f' % roc_auc_test)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



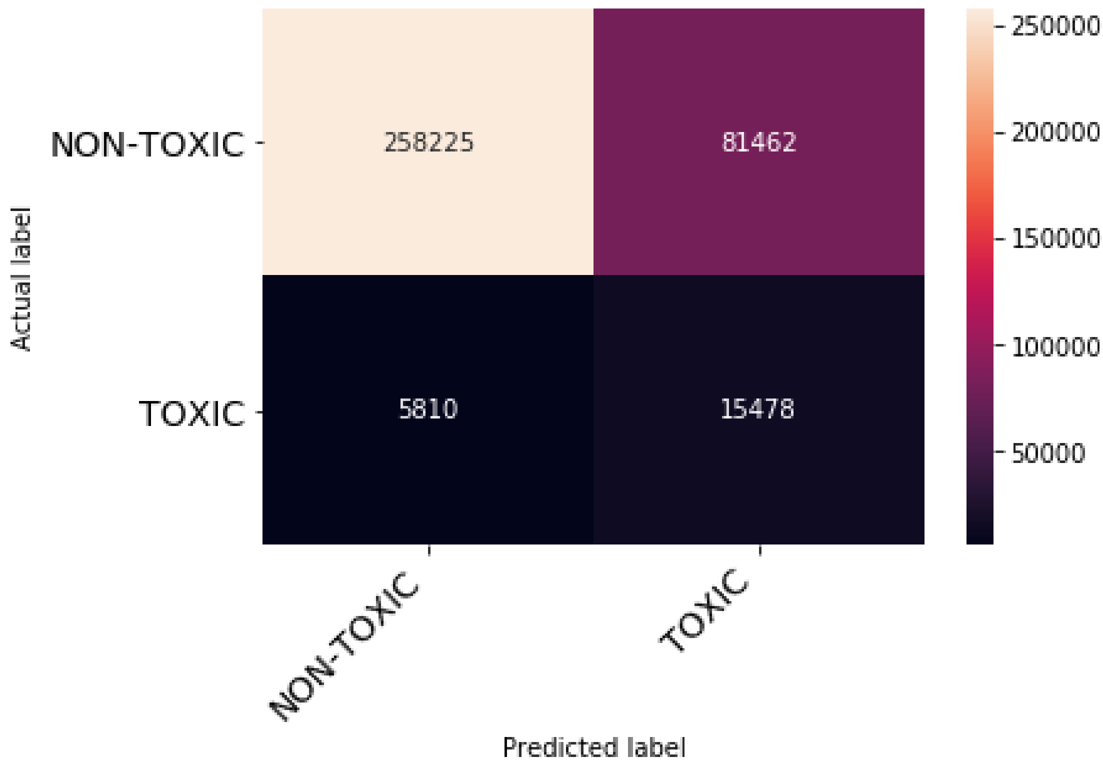

[110]:
```
pred_train =␣
↪predict_with_best_t(predicted_train,tpr_train,fpr_train,threshold_train)
cm = confusion_matrix(y_train, pred_train)
print("\tTRAIN DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	TRAIN DATA CONFUSION MATRIX
```




[111]:
```
pred_test =␣
 ↪predict_with_best_t(predicted_validation,tpr_test,fpr_test,threshold_test)
cm = confusion_matrix(y_validation, pred_test)
print("\ttest DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
        test DATA CONFUSION MATRIX
```




**15000 features**

[112]:
```
alpha = [1e-09, 1e-07, 1e-05, 1e-03, 1, 10]
train_auc_list = []
validation_auc_list = []
names = []##### 25000 features
for param in tqdm(alpha):
    MODEL_NAME = f'NB-tfidf_15k_{param}'
    clf = MultinomialNB(alpha=param)
    clf.fit(train_comment_tfidf_15000, y_train)
    predicted_train = clf.predict_proba(train_comment_tfidf_15000)[:,1]
    predicted_validation = clf.predict_proba(validation_comment_tfidf_15000)[:
↪,1]

    train_data[MODEL_NAME] = predicted_train
    validation_data[MODEL_NAME] = predicted_validation

    train_auc_list.append(get_metric_value(train_data, identity_columns,␣
↪MODEL_NAME))
    validation_auc_list.append(get_metric_value(validation_data,␣
↪identity_columns, MODEL_NAME))
    names.append(MODEL_NAME)
```

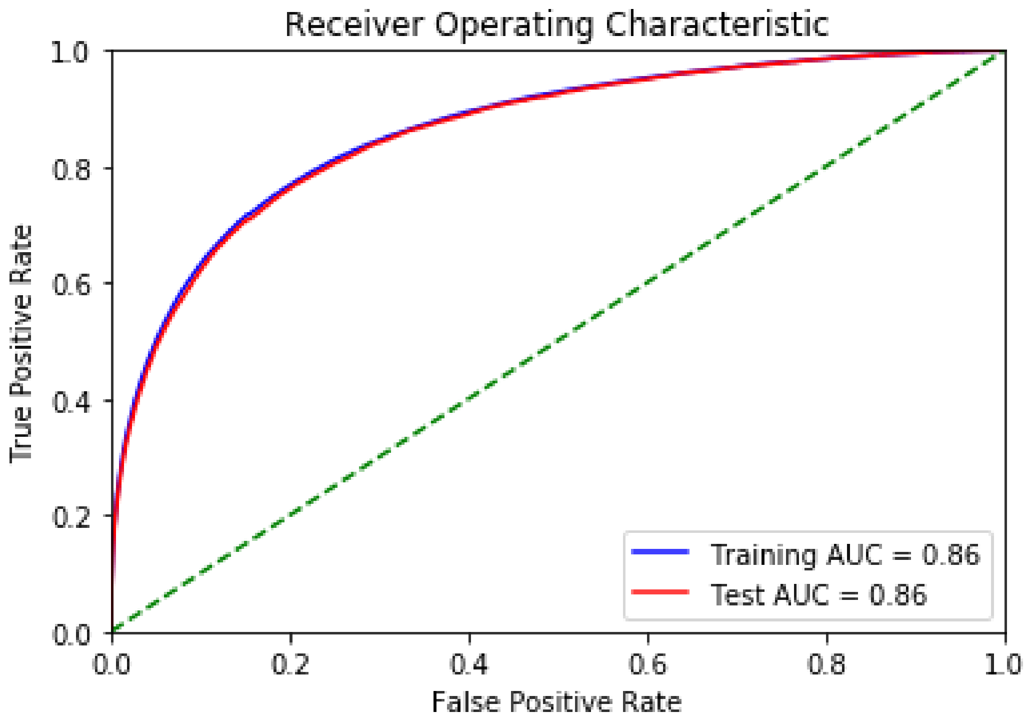
```
100%|          | 6/6 [01:09<00:00, 11.64s/it]
```

[113]:
```
pd.DataFrame({'name':names, 'train-score':train_auc_list, 'test-score':
 ↪validation_auc_list}).sort_values(by=['test-score'])
```

[113]:

| | name | train-score | test-score |
|---|---|---|---|
| 5 | NB-tfidf_15k_10 | 0.836687 | 0.828308 |
| 0 | NB-tfidf_15k_1e-09 | 0.876000 | 0.855023 |
| 1 | NB-tfidf_15k_1e-07 | 0.876000 | 0.855120 |
| 2 | NB-tfidf_15k_1e-05 | 0.875997 | 0.855451 |
| 3 | NB-tfidf_15k_0.001 | 0.875981 | 0.856204 |
| 4 | NB-tfidf_15k_1 | 0.876676 | 0.862418 |

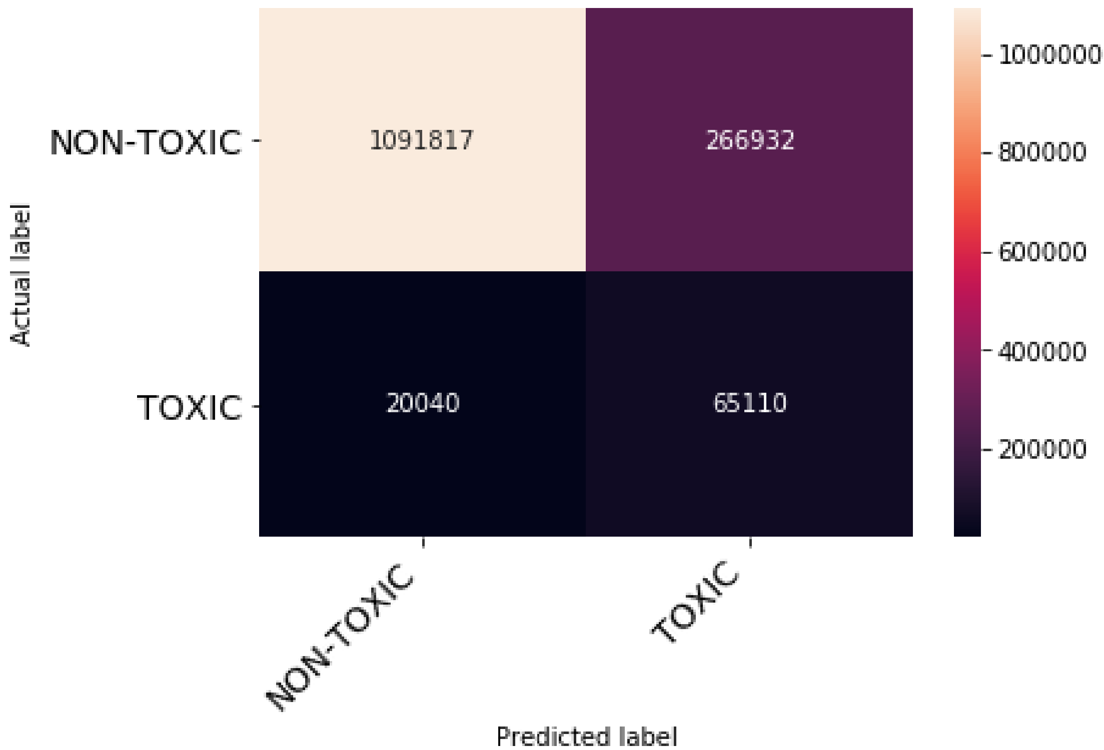
[114]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
 ↪predicted_validation)

roc_auc_train = auc(fpr_train, tpr_train)
roc_auc_test = auc(fpr_test, tpr_test)


plt.title('Receiver Operating Characteristic')

plt.plot(fpr_train, tpr_train, 'b', label = 'Training AUC = %0.2f' %␣
 ↪roc_auc_train)
plt.plot(fpr_test, tpr_test, 'r', label = 'Test AUC = %0.2f' % roc_auc_test)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```




[115]:

```
pred_train = 
 ↪predict_with_best_t(predicted_train,tpr_train,fpr_train,threshold_train)
cm = confusion_matrix(y_train, pred_train)
print("\tTRAIN DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

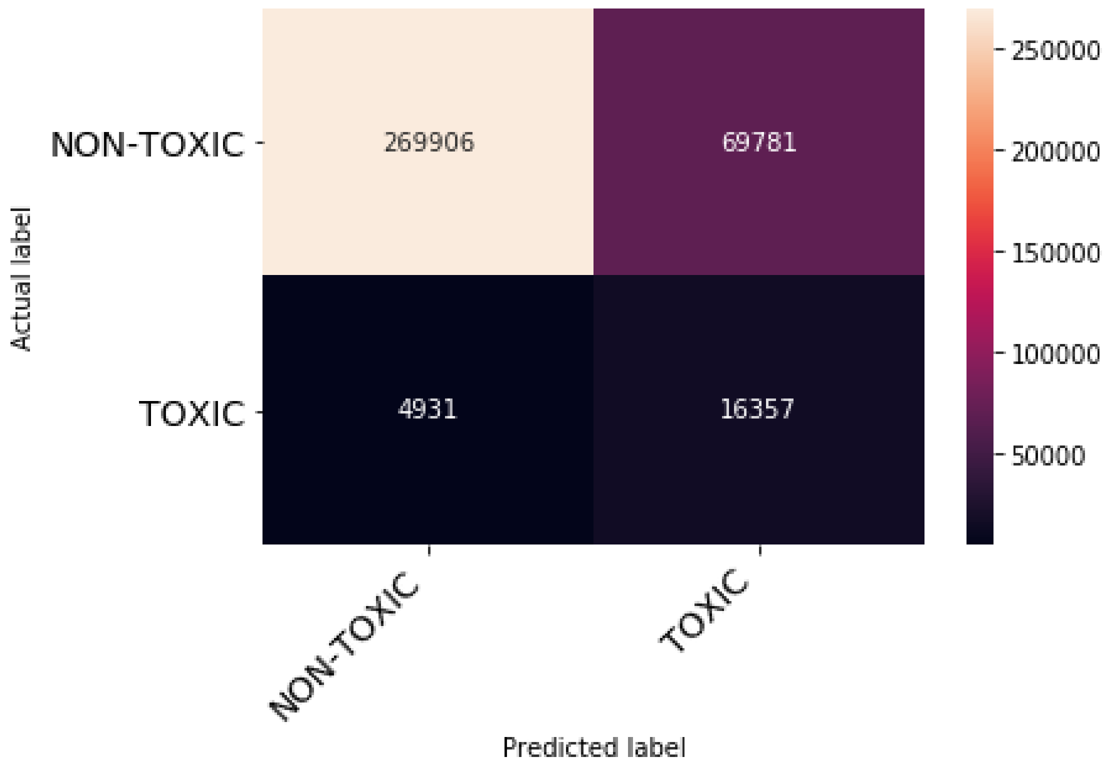
```
	TRAIN DATA CONFUSION MATRIX
```




[116]:

```
pred_test =␣
↪predict_with_best_t(predicted_validation,tpr_test,fpr_test,threshold_test)
cm = confusion_matrix(y_validation, pred_test)
print("\ttest DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	test DATA CONFUSION MATRIX
```




**10000 features**

[117]:
```
alpha = [1e-09, 1e-07, 1e-05, 1e-03, 1, 10]
train_auc_list = []
validation_auc_list = []
names = []##### 25000 features
for param in tqdm(alpha):
    MODEL_NAME = f'NB-tfidf_10k_{param}'
    clf = MultinomialNB(alpha=param)
    clf.fit(train_comment_tfidf_10000, y_train)
    predicted_train = clf.predict_proba(train_comment_tfidf_10000)[:,1]
    predicted_validation = clf.predict_proba(validation_comment_tfidf_10000)[:
 ↪,1]

    train_data[MODEL_NAME] = predicted_train
    validation_data[MODEL_NAME] = predicted_validation

    train_auc_list.append(get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME))
    validation_auc_list.append(get_metric_value(validation_data,␣
 ↪identity_columns, MODEL_NAME))
    names.append(MODEL_NAME)
```

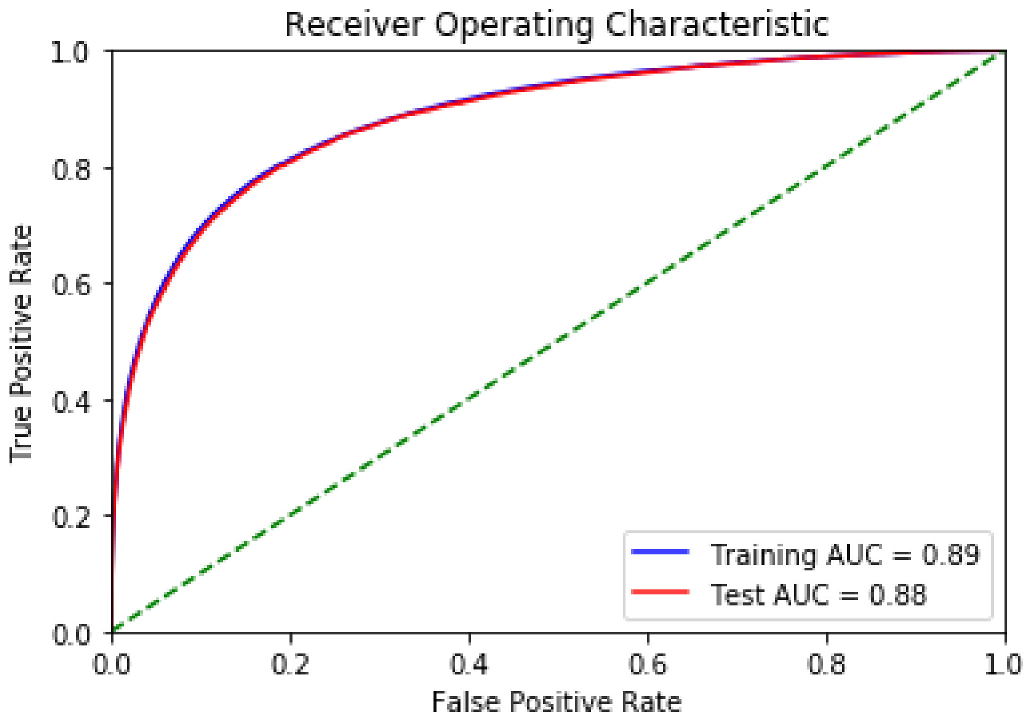
```
100%|          | 6/6 [01:09<00:00, 11.59s/it]
```

[118]:
```
pd.DataFrame({'name':names, 'train-score':train_auc_list, 'test-score':
 ↪validation_auc_list}).sort_values(by=['test-score'])
```

[118]:

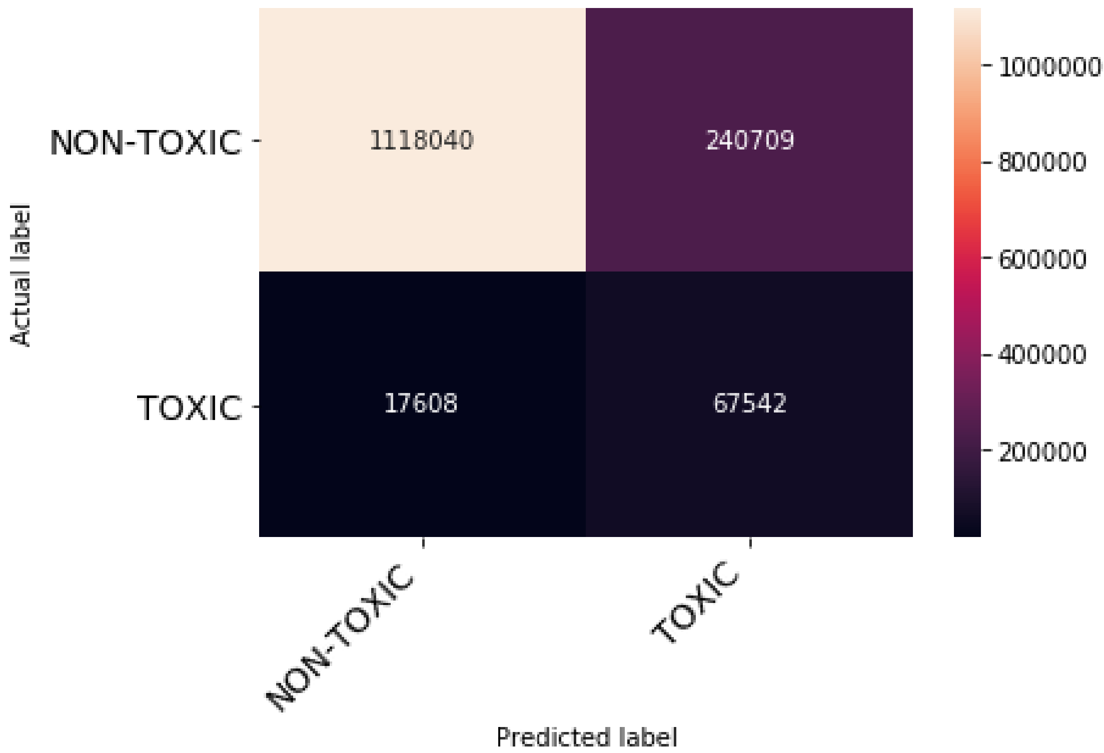
| | name | train-score | test-score |
|---|---|---|---|
| 5 | NB-tfidf_10k_10 | 0.855698 | 0.846789 |
| 0 | NB-tfidf_10k_1e-09 | 0.871553 | 0.857473 |
| 1 | NB-tfidf_10k_1e-07 | 0.871553 | 0.857475 |
| 2 | NB-tfidf_10k_1e-05 | 0.871553 | 0.857481 |
| 3 | NB-tfidf_10k_0.001 | 0.871555 | 0.857500 |
| 4 | NB-tfidf_10k_1 | 0.873399 | 0.860715 |

[119]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
 ↪predicted_validation)

roc_auc_train = auc(fpr_train, tpr_train)
roc_auc_test = auc(fpr_test, tpr_test)


plt.title('Receiver Operating Characteristic')

plt.plot(fpr_train, tpr_train, 'b', label = 'Training AUC = %0.2f' %␣
 ↪roc_auc_train)
plt.plot(fpr_test, tpr_test, 'r', label = 'Test AUC = %0.2f' % roc_auc_test)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



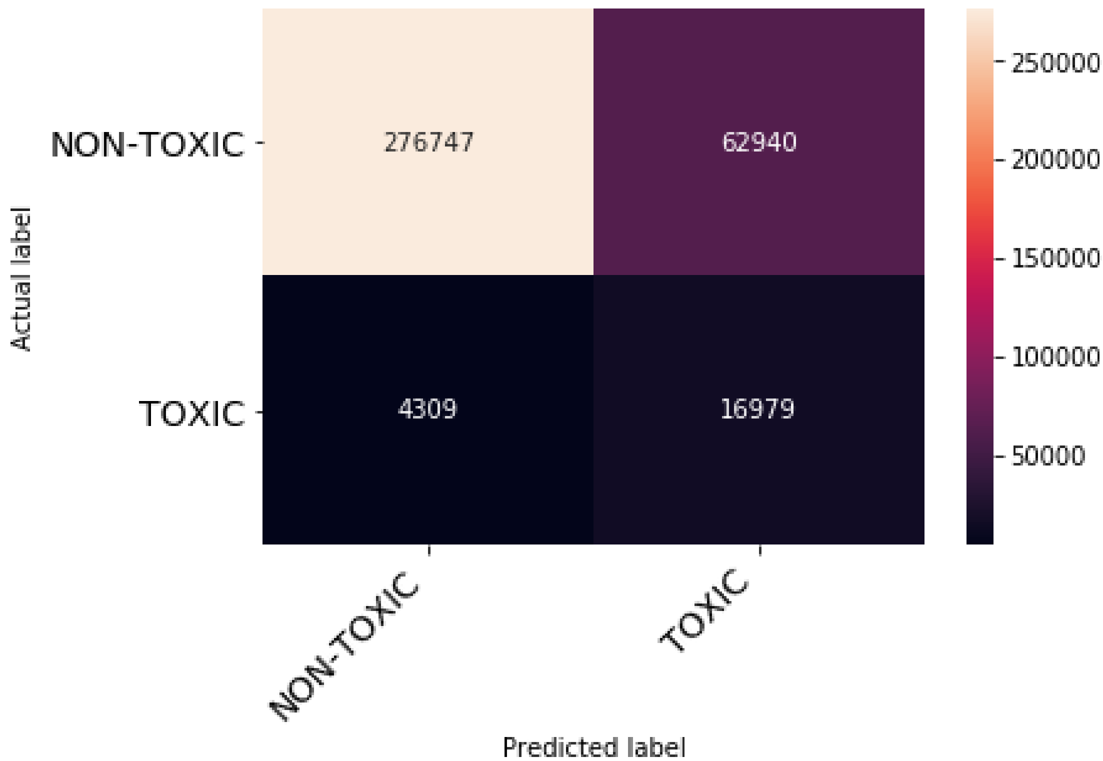

[120]:
```
pred_train =␣
↪predict_with_best_t(predicted_train,tpr_train,fpr_train,threshold_train)
cm = confusion_matrix(y_train, pred_train)
print("\tTRAIN DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	TRAIN DATA CONFUSION MATRIX
```




[121]:
```
pred_test =␣
↪predict_with_best_t(predicted_validation,tpr_test,fpr_test,threshold_test)
cm = confusion_matrix(y_validation, pred_test)
print("\ttest DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	test DATA CONFUSION MATRIX
```




**Considering W2V**

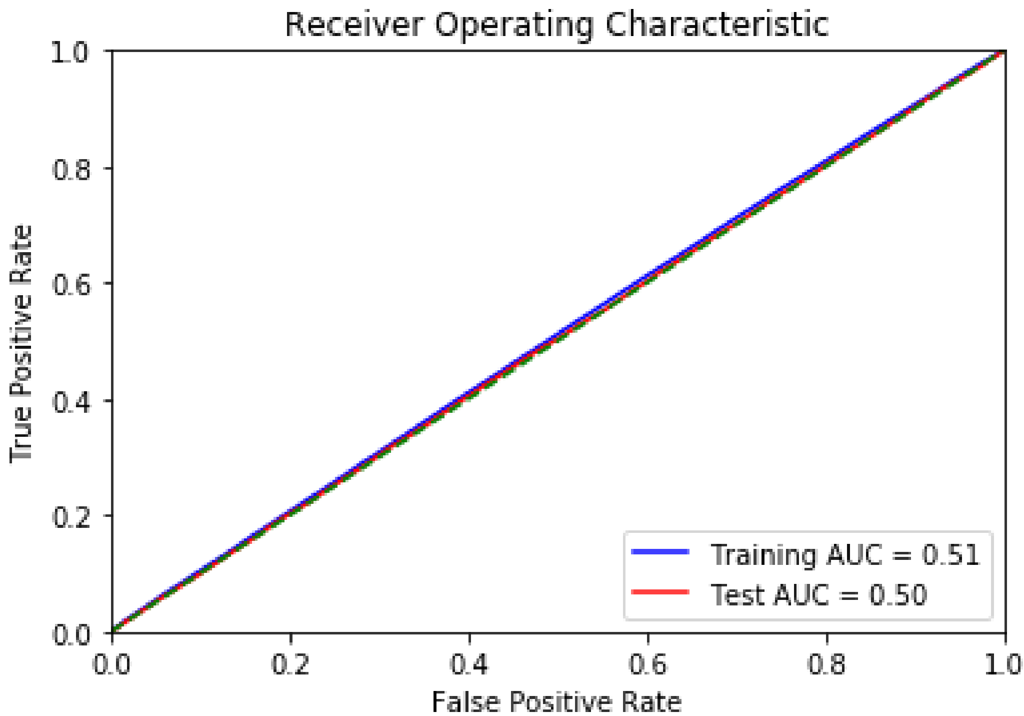
[16]:
```
alpha = [1e-09, 1e-07, 1e-05, 1e-03, 1, 10]
train_auc_list = []
validation_auc_list = []
names = []
for param in tqdm(alpha):
    MODEL_NAME = f'NB-tfidf_10k_{param}'
    clf = GaussianNB(var_smoothing=param)
    clf.fit(train_comment_w2v, y_train)
    predicted_train = clf.predict_proba(train_comment_w2v)[:,1]
    predicted_validation = clf.predict_proba(validation_comment_w2v)[:,1]

    train_data[MODEL_NAME] = predicted_train
    validation_data[MODEL_NAME] = predicted_validation

    train_auc_list.append(get_metric_value(train_data, identity_columns,␣
↪MODEL_NAME))
    validation_auc_list.append(get_metric_value(validation_data,␣
↪identity_columns, MODEL_NAME))
    names.append(MODEL_NAME)
```

```
100%|          | 6/6 [01:38<00:00, 16.45s/it]
```

[17]:
```
pd.DataFrame({'name':names, 'train-score':train_auc_list, 'test-score':
↪validation_auc_list}).sort_values(by=['test-score'])
```

```
[0.5031867576902547, 0.5031867535041921, 0.5031869985568977, 0.5032009538245688,
0.5105167104410043, 0.5122834718987822]
[0.4989776159532118, 0.4989776045293055, 0.4989766111799462, 0.4990092008393191,
0.5001355187837999, 0.49766714248000105]
```

[18]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
↪predicted_validation)

roc_auc_train = auc(fpr_train, tpr_train)
roc_auc_test = auc(fpr_test, tpr_test)


plt.title('Receiver Operating Characteristic')

plt.plot(fpr_train, tpr_train, 'b', label = 'Training AUC = %0.2f' %␣
↪roc_auc_train)
plt.plot(fpr_test, tpr_test, 'r', label = 'Test AUC = %0.2f' % roc_auc_test)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```




[ ]:
```
pred_train =␣
↪predict_with_best_t(predicted_train,tpr_train,fpr_train,threshold_train)
cm = confusion_matrix(y_train, pred_train)
print("\tTRAIN DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

[ ]:
```
pred_test =␣
↪predict_with_best_t(predicted_validation,tpr_test,fpr_test,threshold_test)
cm = confusion_matrix(y_validation, pred_test)
print("\ttest DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

### 5.1.2 Logistic Regression

**Considering BOW**

**25000 features**

[122]:
```
alpha = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10]
train_auc_list = []
validation_auc_list = []
names = []
for param in tqdm(alpha):
```

```
    MODEL_NAME = f'LR-BOW_25k_{param}'
    clf = SGDClassifier(alpha=param, class_weight='balanced', loss='log',␣
 ↪penalty='l2')
    clf.fit(train_comment_bow_25000, y_train)
#     clf = CalibratedClassifierCV(clf, method="sigmoid")
#     clf.fit(train_comment_bow_25000, y_train)
    predicted_train = clf.predict_proba(train_comment_bow_25000)[:,1]
    predicted_validation = clf.predict_proba(validation_comment_bow_25000)[:,1]

    train_data[MODEL_NAME] = predicted_train
    validation_data[MODEL_NAME] = predicted_validation

    train_auc_list.append(get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME))
    validation_auc_list.append(get_metric_value(validation_data,␣
 ↪identity_columns, MODEL_NAME))
    names.append(MODEL_NAME)
```

```
100%|          | 7/7 [01:56<00:00, 16.65s/it]
```

[123]:

```
pd.DataFrame({'name':names, 'train-score':train_auc_list, 'test-score':
 ↪validation_auc_list}).sort_values(by=['test-score'])
```

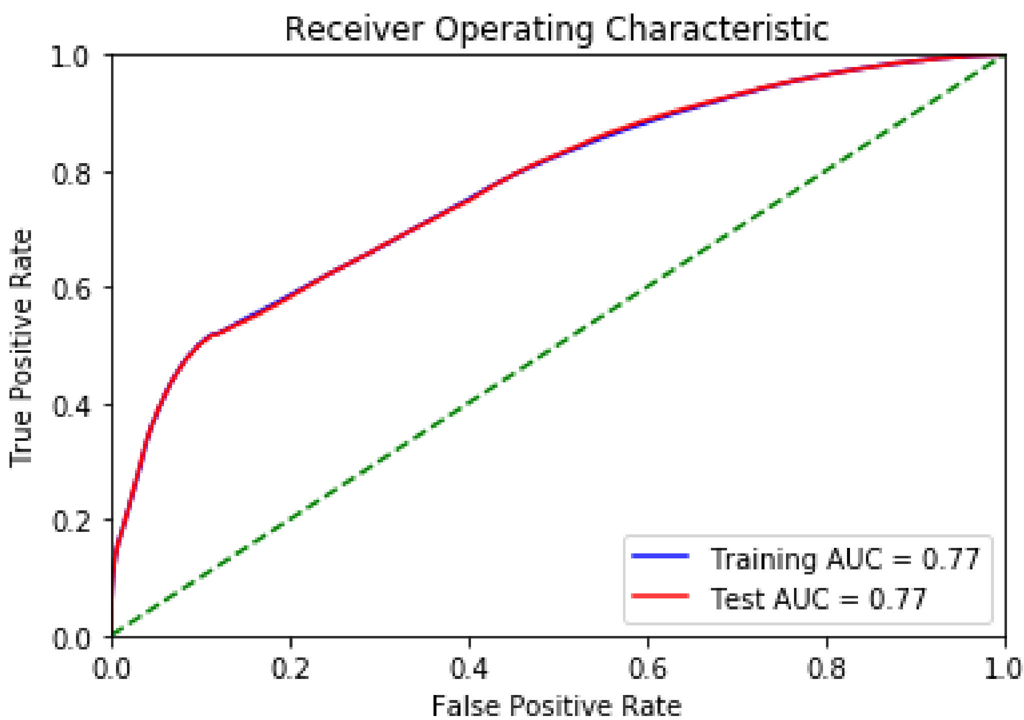
[123]:

| | name | train-score | test-score |
|---|---|---|---|
| 6 | LR-BOW_25k_10 | 0.751828 | 0.747758 |
| 5 | LR-BOW_25k_1 | 0.761209 | 0.756367 |
| 4 | LR-BOW_25k_0.1 | 0.779744 | 0.774906 |
| 3 | LR-BOW_25k_0.01 | 0.827500 | 0.822857 |
| 2 | LR-BOW_25k_0.001 | 0.874368 | 0.867314 |
| 1 | LR-BOW_25k_0.0001 | 0.898346 | 0.886196 |
| 0 | LR-BOW_25k_1e-05 | 0.913540 | 0.890871 |

[124]:

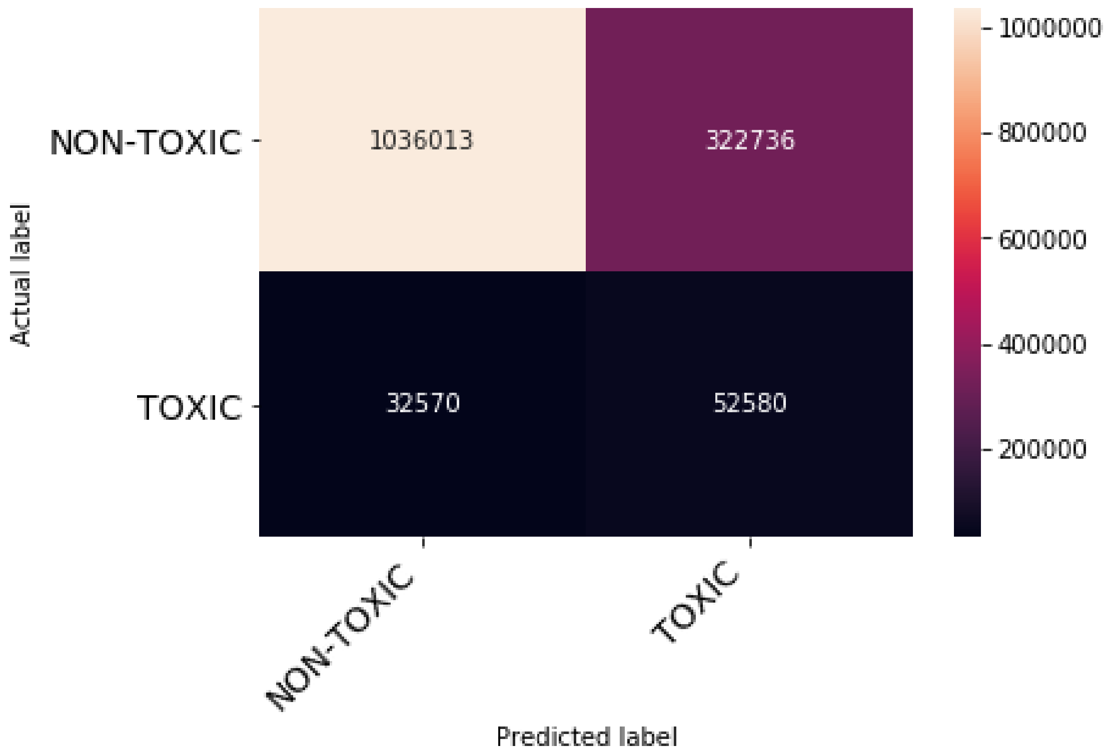
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
 ↪predicted_validation)

roc_auc_train = auc(fpr_train, tpr_train)
roc_auc_test = auc(fpr_test, tpr_test)


plt.title('Receiver Operating Characteristic')

plt.plot(fpr_train, tpr_train, 'b', label = 'Training AUC = %0.2f' %␣
 ↪roc_auc_train)
plt.plot(fpr_test, tpr_test, 'r', label = 'Test AUC = %0.2f' % roc_auc_test)
```

```
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



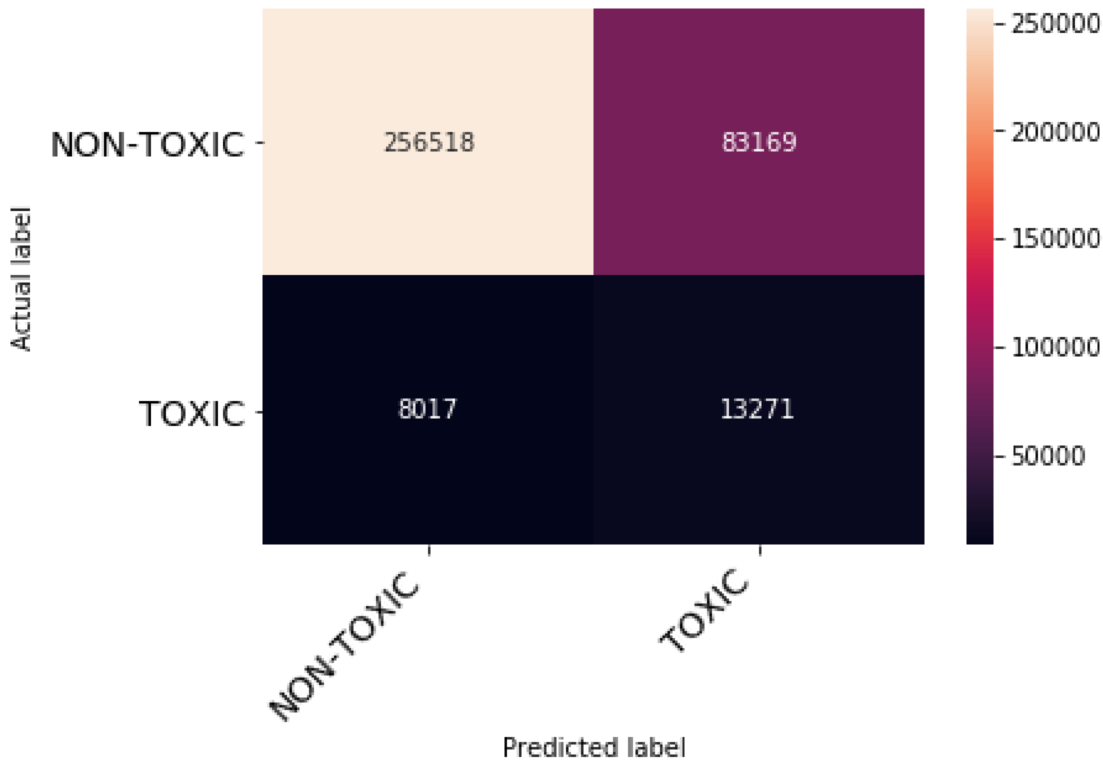

[125]:
```
pred_train =␣
 ↪predict_with_best_t(predicted_train,tpr_train,fpr_train,threshold_train)
cm = confusion_matrix(y_train, pred_train)
print("\tTRAIN DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	TRAIN DATA CONFUSION MATRIX
```




[126]:
```
pred_test =␣
↪predict_with_best_t(predicted_validation,tpr_test,fpr_test,threshold_test)
cm = confusion_matrix(y_validation, pred_test)
print("\ttest DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
        test DATA CONFUSION MATRIX
```




**15000 features**

[127]:
```
alpha = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10]
train_auc_list = []
validation_auc_list = []
names = []
for param in tqdm(alpha):
    MODEL_NAME = f'LR-BOW_15k_{param}'
    clf = SGDClassifier(alpha=param, class_weight='balanced', loss='log',␣
↪penalty='l2')
    clf.fit(train_comment_bow_15000, y_train)
#     clf = CalibratedClassifierCV(clf, method="sigmoid")
#     clf.fit(train_comment_bow_15000, y_train)
    predicted_train = clf.predict_proba(train_comment_bow_15000)[:,1]
    predicted_validation = clf.predict_proba(validation_comment_bow_15000)[:,1]

    train_data[MODEL_NAME] = predicted_train
    validation_data[MODEL_NAME] = predicted_validation

    train_auc_list.append(get_metric_value(train_data, identity_columns,␣
↪MODEL_NAME))
```

```
    validation_auc_list.append(get_metric_value(validation_data,␣
↪identity_columns, MODEL_NAME))
    names.append(MODEL_NAME)
```

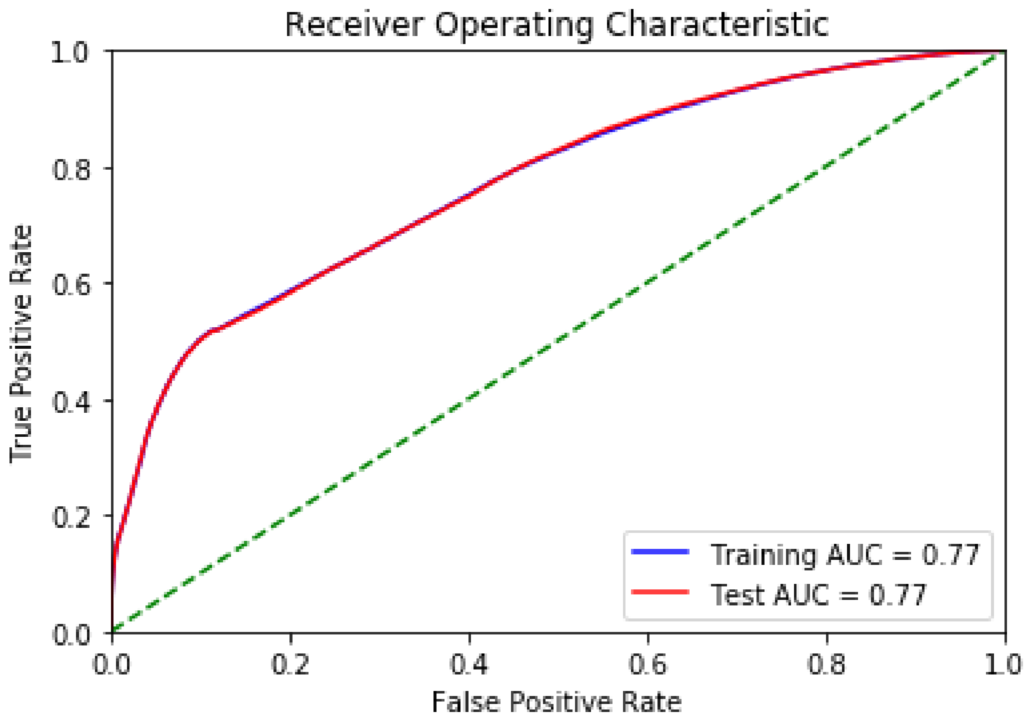
```
100%|          | 7/7 [01:54<00:00, 16.39s/it]
```

[128]:
```
pd.DataFrame({'name':names, 'train-score':train_auc_list, 'test-score':
 ↪validation_auc_list}).sort_values(by=['test-score'])
```

[128]:

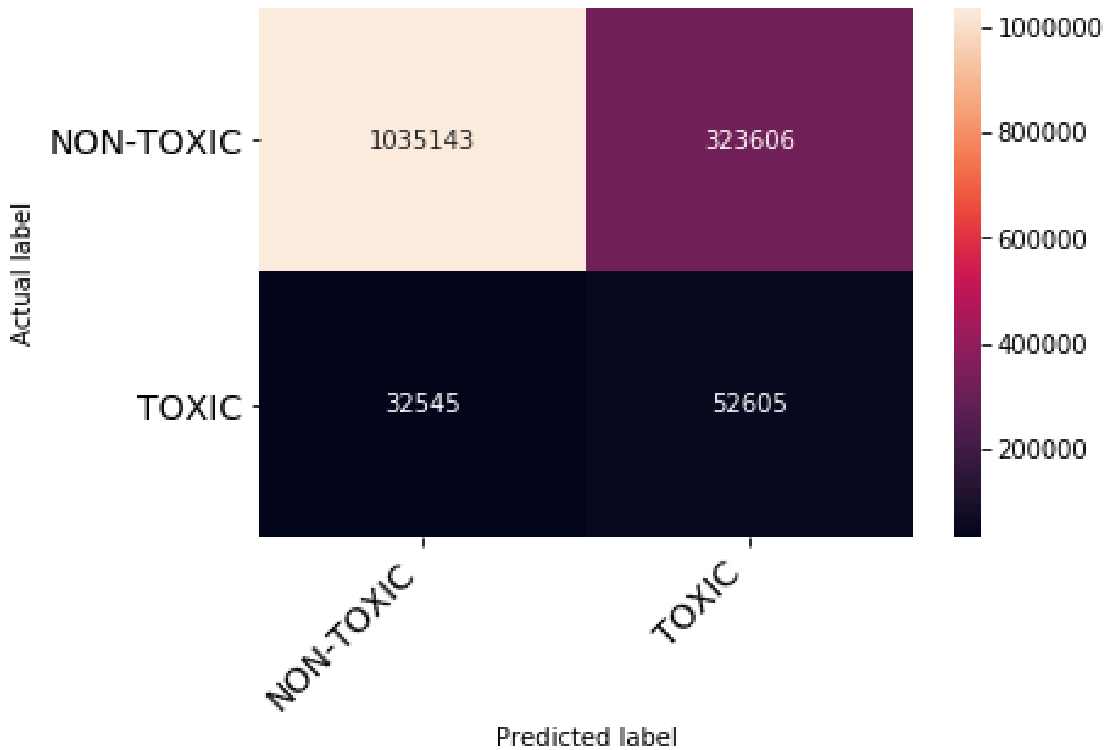
| | name | train-score | test-score |
|---|---|---|---|
| 6 | LR-BOW_15k_10 | 0.751741 | 0.747689 |
| 5 | LR-BOW_15k_1 | 0.755531 | 0.751251 |
| 4 | LR-BOW_15k_0.1 | 0.779571 | 0.774765 |
| 3 | LR-BOW_15k_0.01 | 0.827396 | 0.822562 |
| 2 | LR-BOW_15k_0.001 | 0.873701 | 0.866733 |
| 1 | LR-BOW_15k_0.0001 | 0.898889 | 0.887605 |
| 0 | LR-BOW_15k_1e-05 | 0.908690 | 0.888824 |

[129]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
 ↪predicted_validation)

roc_auc_train = auc(fpr_train, tpr_train)
roc_auc_test = auc(fpr_test, tpr_test)


plt.title('Receiver Operating Characteristic')

plt.plot(fpr_train, tpr_train, 'b', label = 'Training AUC = %0.2f' %␣
 ↪roc_auc_train)
plt.plot(fpr_test, tpr_test, 'r', label = 'Test AUC = %0.2f' % roc_auc_test)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



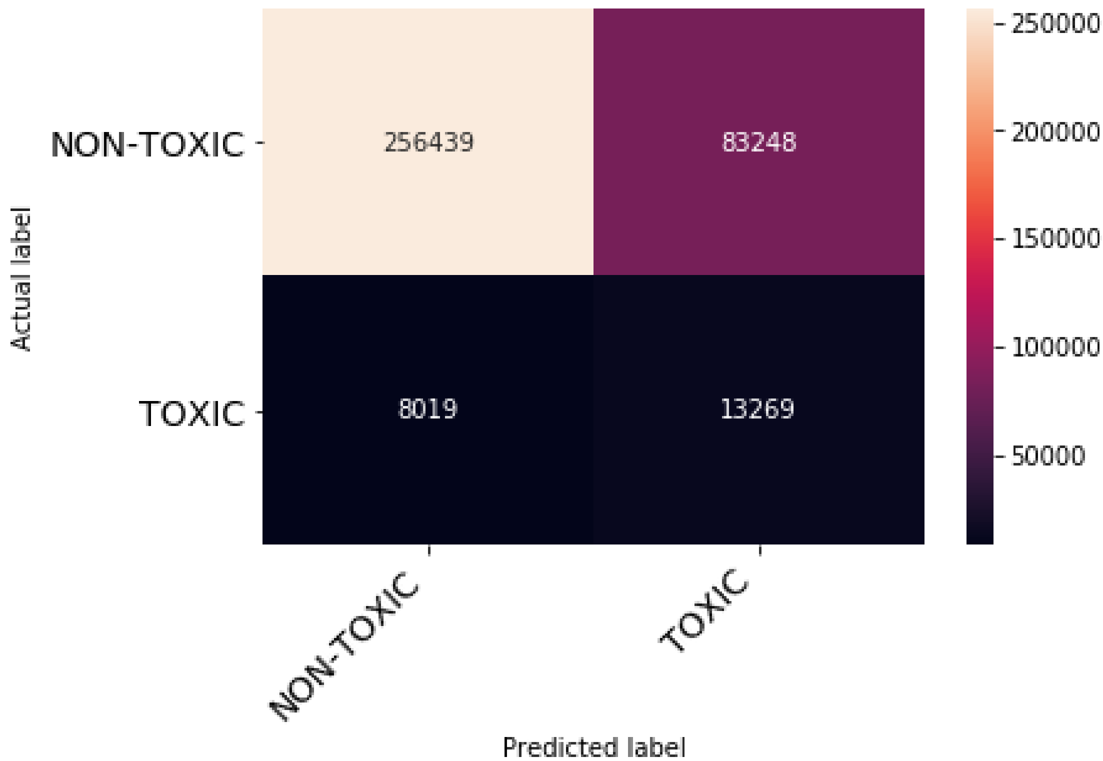

[130]:
```
pred_train = predict_with_best_t(predicted_train,tpr_train,fpr_train,threshold_train)
cm = confusion_matrix(y_train, pred_train)
print("\tTRAIN DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	TRAIN DATA CONFUSION MATRIX
```




[131]:
```
pred_test =␣
 ↪predict_with_best_t(predicted_validation,tpr_test,fpr_test,threshold_test)
cm = confusion_matrix(y_validation, pred_test)
print("\ttest DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
        test DATA CONFUSION MATRIX
```




**10000 features**

[132]:
```
alpha = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10]
train_auc_list = []
validation_auc_list = []
names = []
for param in tqdm(alpha):
    MODEL_NAME = f'LR-BOW_10k_{param}'
    clf = SGDClassifier(alpha=param, class_weight='balanced', loss='log',␣
 ↪penalty='l2')
    clf.fit(train_comment_bow_10000, y_train)
#     clf = CalibratedClassifierCV(clf, method="sigmoid")
#     clf.fit(train_comment_bow_10000, y_train)
    predicted_train = clf.predict_proba(train_comment_bow_10000)[:,1]
    predicted_validation = clf.predict_proba(validation_comment_bow_10000)[:,1]

    train_data[MODEL_NAME] = predicted_train
    validation_data[MODEL_NAME] = predicted_validation

    train_auc_list.append(get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME))
```

```
    validation_auc_list.append(get_metric_value(validation_data,␣
↪identity_columns, MODEL_NAME))
    names.append(MODEL_NAME)
```

```
100%|          | 7/7 [01:54<00:00, 16.29s/it]
```

[133]:

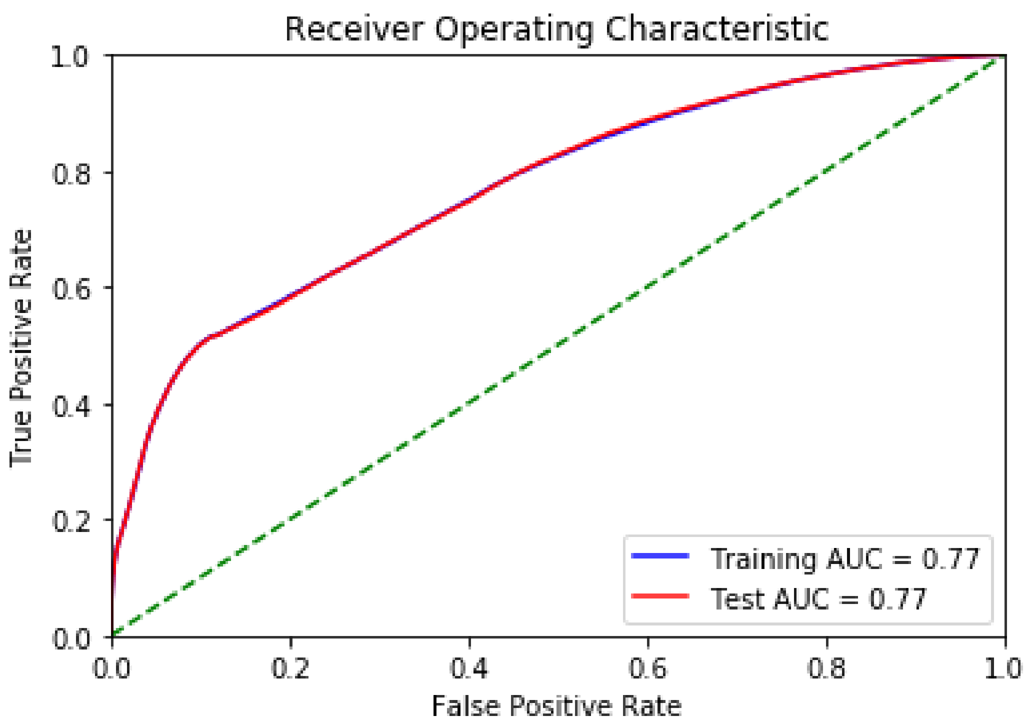
```
pd.DataFrame({'name':names, 'train-score':train_auc_list, 'test-score':
↪validation_auc_list}).sort_values(by=['test-score'])
```

[133]:

| | name | train-score | test-score |
|---|---|---|---|
| 6 | LR-BOW_10k_10 | 0.751267 | 0.747289 |
| 5 | LR-BOW_10k_1 | 0.760469 | 0.755752 |
| 4 | LR-BOW_10k_0.1 | 0.781014 | 0.776068 |
| 3 | LR-BOW_10k_0.01 | 0.826869 | 0.822427 |
| 2 | LR-BOW_10k_0.001 | 0.872461 | 0.865625 |
| 0 | LR-BOW_10k_1e-05 | 0.900373 | 0.882911 |
| 1 | LR-BOW_10k_0.0001 | 0.896571 | 0.885959 |

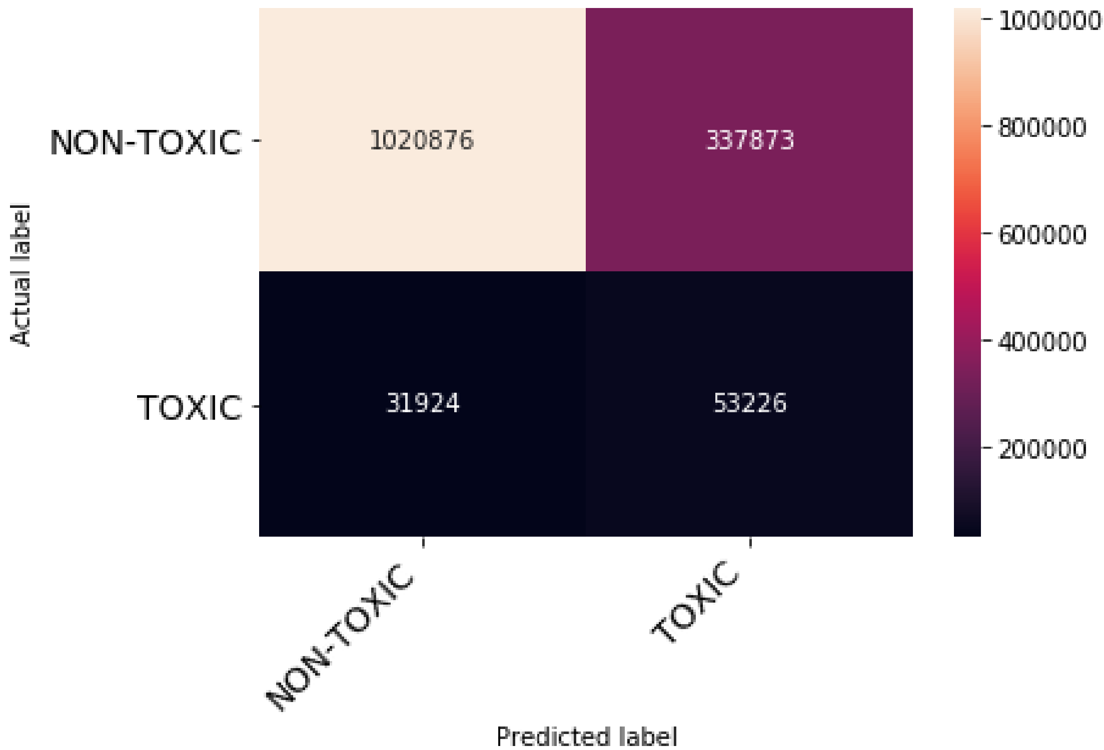
[134]:

```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
↪predicted_validation)

roc_auc_train = auc(fpr_train, tpr_train)
roc_auc_test = auc(fpr_test, tpr_test)


plt.title('Receiver Operating Characteristic')

plt.plot(fpr_train, tpr_train, 'b', label = 'Training AUC = %0.2f' %␣
↪roc_auc_train)
plt.plot(fpr_test, tpr_test, 'r', label = 'Test AUC = %0.2f' % roc_auc_test)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



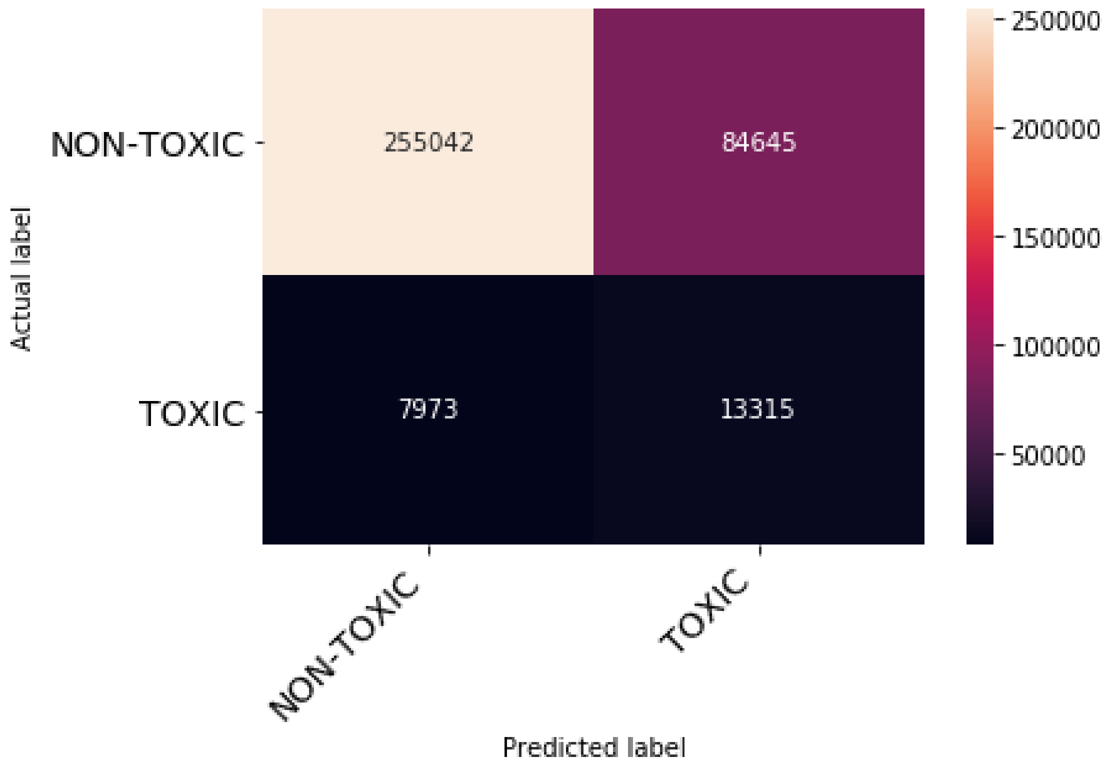

[135]:
```
pred_train =␣
↪predict_with_best_t(predicted_train,tpr_train,fpr_train,threshold_train)
cm = confusion_matrix(y_train, pred_train)
print("\tTRAIN DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	TRAIN DATA CONFUSION MATRIX
```




[136]:
```
pred_test =␣
↪predict_with_best_t(predicted_validation,tpr_test,fpr_test,threshold_test)
cm = confusion_matrix(y_validation, pred_test)
print("\ttest DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	test DATA CONFUSION MATRIX
```




**Considering TFIDF**

**25000 features**

[137]:
```
alpha = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10]
train_auc_list = []
validation_auc_list = []
names = []
for param in tqdm(alpha):
    MODEL_NAME = f'LR-tfidf_25k_{param}'
    clf = SGDClassifier(alpha=param, class_weight='balanced', loss='log',␣
 ↪penalty='l2')
    clf.fit(train_comment_tfidf_25000, y_train)
#     clf = CalibratedClassifierCV(clf, method="sigmoid")
#     clf.fit(train_comment_tfidf_25000, y_train)
    predicted_train = clf.predict_proba(train_comment_tfidf_25000)[:,1]
    predicted_validation = clf.predict_proba(validation_comment_tfidf_25000)[:
 ↪,1]


    train_data[MODEL_NAME] = predicted_train
```

```
    validation_data[MODEL_NAME] = predicted_validation

    train_auc_list.append(get_metric_value(train_data, identity_columns,␣
↪MODEL_NAME))
    validation_auc_list.append(get_metric_value(validation_data,␣
↪identity_columns, MODEL_NAME))
    names.append(MODEL_NAME)
```

```
100%|      | 7/7 [01:43<00:00, 14.73s/it]
```

[138]:

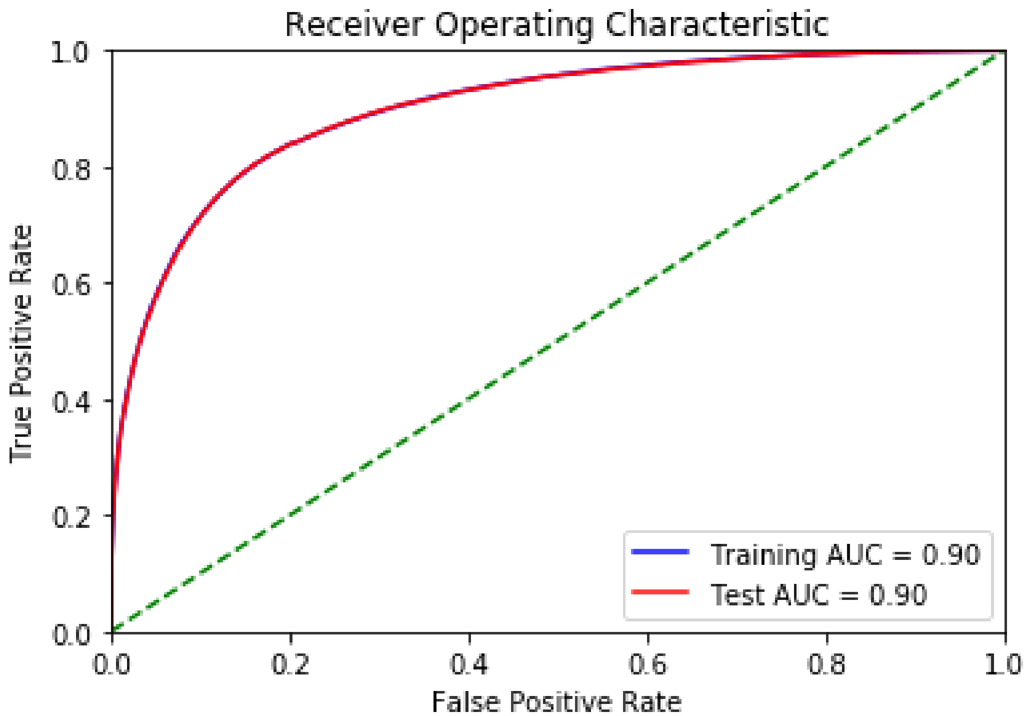
```
pd.DataFrame({'name':names, 'train-score':train_auc_list, 'test-score':
↪validation_auc_list}).sort_values(by=['test-score'])
```

[138]:

| | name | train-score | test-score |
|---|---|---|---|
| 5 | LR-tfidf_25k_1 | 0.849398 | 0.844226 |
| 6 | LR-tfidf_25k_10 | 0.849540 | 0.844428 |
| 4 | LR-tfidf_25k_0.1 | 0.849667 | 0.844975 |
| 3 | LR-tfidf_25k_0.01 | 0.851261 | 0.846079 |
| 2 | LR-tfidf_25k_0.001 | 0.862609 | 0.857095 |
| 1 | LR-tfidf_25k_0.0001 | 0.890625 | 0.883832 |
| 0 | LR-tfidf_25k_1e-05 | 0.913394 | 0.902973 |

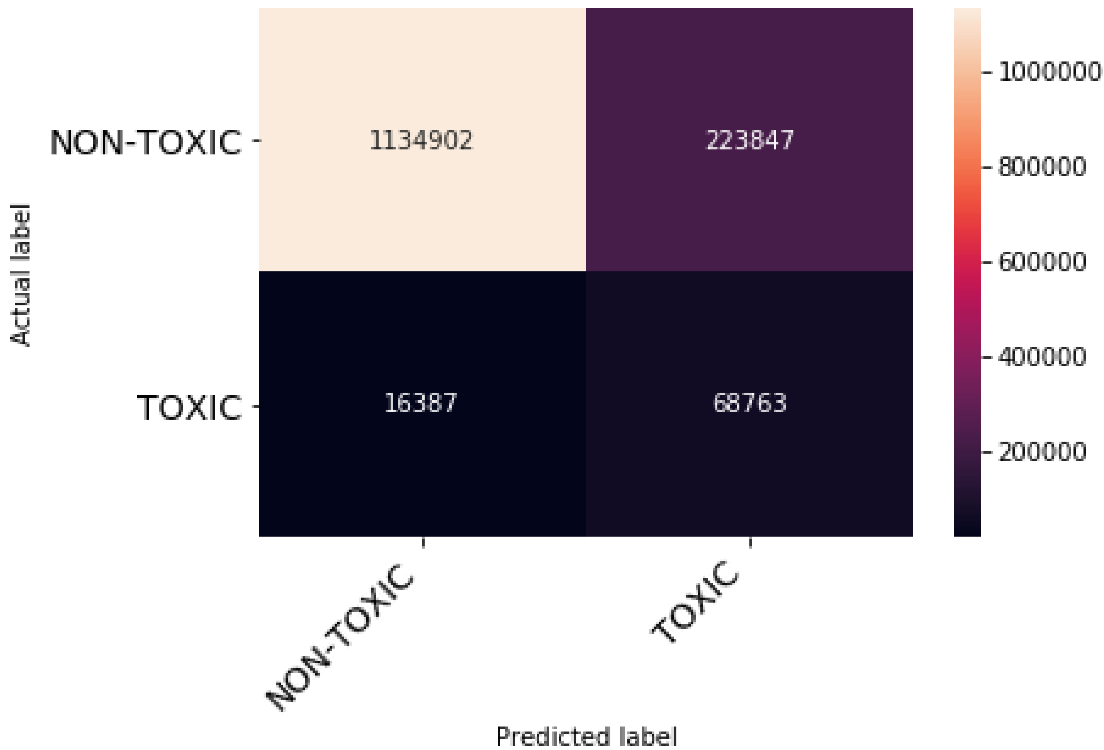
[139]:

```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
↪predicted_validation)

roc_auc_train = auc(fpr_train, tpr_train)
roc_auc_test = auc(fpr_test, tpr_test)


plt.title('Receiver Operating Characteristic')

plt.plot(fpr_train, tpr_train, 'b', label = 'Training AUC = %0.2f' %␣
↪roc_auc_train)
plt.plot(fpr_test, tpr_test, 'r', label = 'Test AUC = %0.2f' % roc_auc_test)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



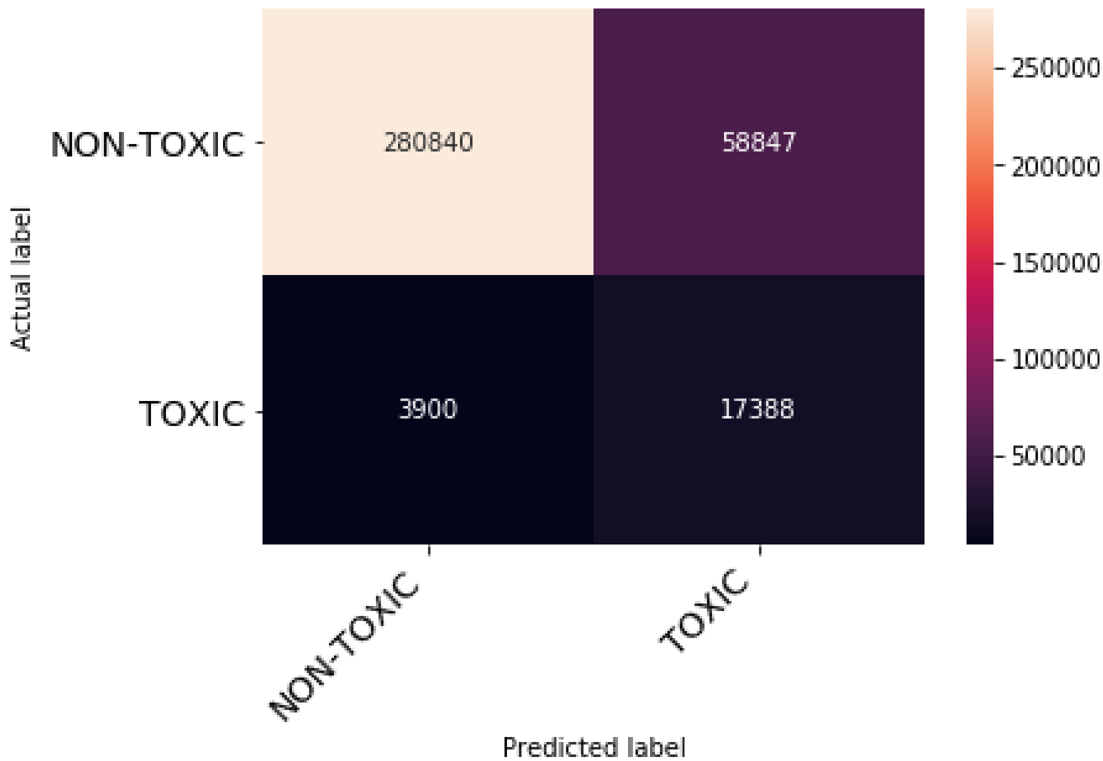

[140]:
```
pred_train = predict_with_best_t(predicted_train,tpr_train,fpr_train,threshold_train)
cm = confusion_matrix(y_train, pred_train)
print("\tTRAIN DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	TRAIN DATA CONFUSION MATRIX
```




[141]:
```
pred_test =␣
↪predict_with_best_t(predicted_validation,tpr_test,fpr_test,threshold_test)
cm = confusion_matrix(y_validation, pred_test)
print("\ttest DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	test DATA CONFUSION MATRIX
```




**15000 features**

[142]:
```
alpha = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10]
train_auc_list = []
validation_auc_list = []
names = []
for param in tqdm(alpha):
    MODEL_NAME = f'LR-tfidf_15k_{param}'
    clf = SGDClassifier(alpha=param, class_weight='balanced', loss='log',␣
 ↪penalty='l2')
    clf.fit(train_comment_tfidf_15000, y_train)
#     clf = CalibratedClassifierCV(clf, method="sigmoid")
#     clf.fit(train_comment_tfidf_15000, y_train)
    predicted_train = clf.predict_proba(train_comment_tfidf_15000)[:,1]
    predicted_validation = clf.predict_proba(validation_comment_tfidf_15000)[:
 ↪,1]

    train_data[MODEL_NAME] = predicted_train
    validation_data[MODEL_NAME] = predicted_validation

    train_auc_list.append(get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME))
```

```
    validation_auc_list.append(get_metric_value(validation_data,␣
↪identity_columns, MODEL_NAME))
    names.append(MODEL_NAME)
```

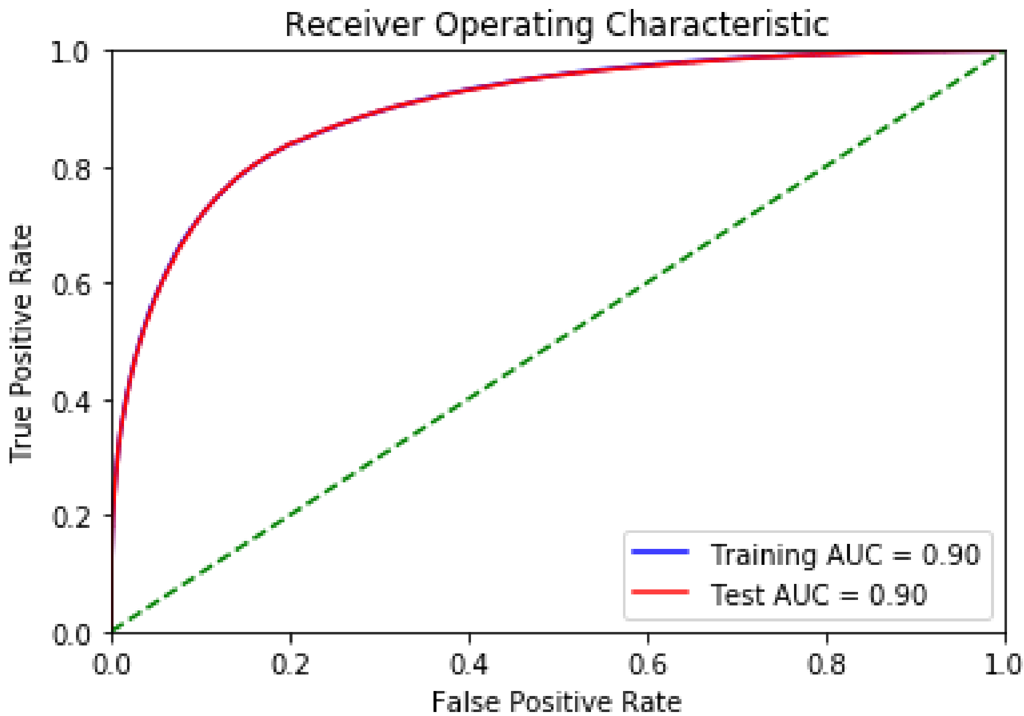
100%|          | 7/7 [01:45<00:00, 15.01s/it]

[143]:
```
pd.DataFrame({'name':names, 'train-score':train_auc_list, 'test-score':
↪validation_auc_list}).sort_values(by=['test-score'])
```

[143]:

| | name | train-score | test-score |
|---|---|---|---|
| 4 | LR-tfidf_15k_0.1 | 0.846915 | 0.841272 |
| 5 | LR-tfidf_15k_1 | 0.849392 | 0.844140 |
| 6 | LR-tfidf_15k_10 | 0.849631 | 0.844461 |
| 3 | LR-tfidf_15k_0.01 | 0.851762 | 0.846732 |
| 2 | LR-tfidf_15k_0.001 | 0.863051 | 0.857573 |
| 1 | LR-tfidf_15k_0.0001 | 0.890845 | 0.884218 |
| 0 | LR-tfidf_15k_1e-05 | 0.912073 | 0.902164 |

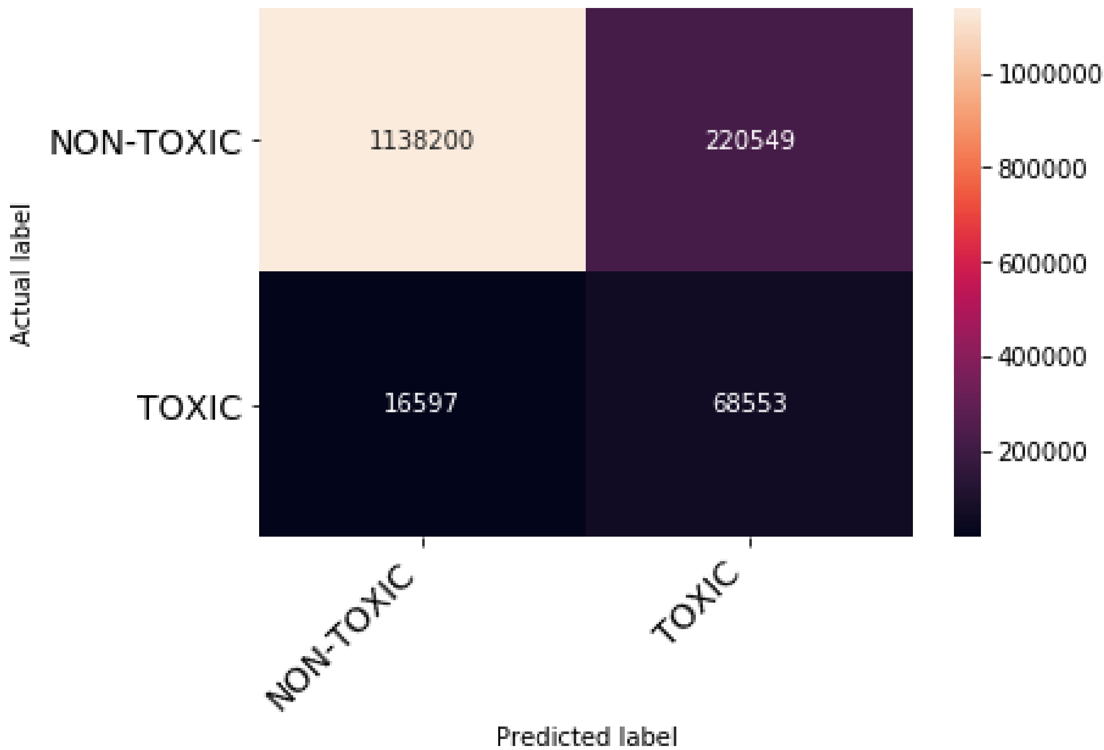
[144]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
↪predicted_validation)

roc_auc_train = auc(fpr_train, tpr_train)
roc_auc_test = auc(fpr_test, tpr_test)



plt.title('Receiver Operating Characteristic')

plt.plot(fpr_train, tpr_train, 'b', label = 'Training AUC = %0.2f' %␣
↪roc_auc_train)
plt.plot(fpr_test, tpr_test, 'r', label = 'Test AUC = %0.2f' % roc_auc_test)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```




[145]:

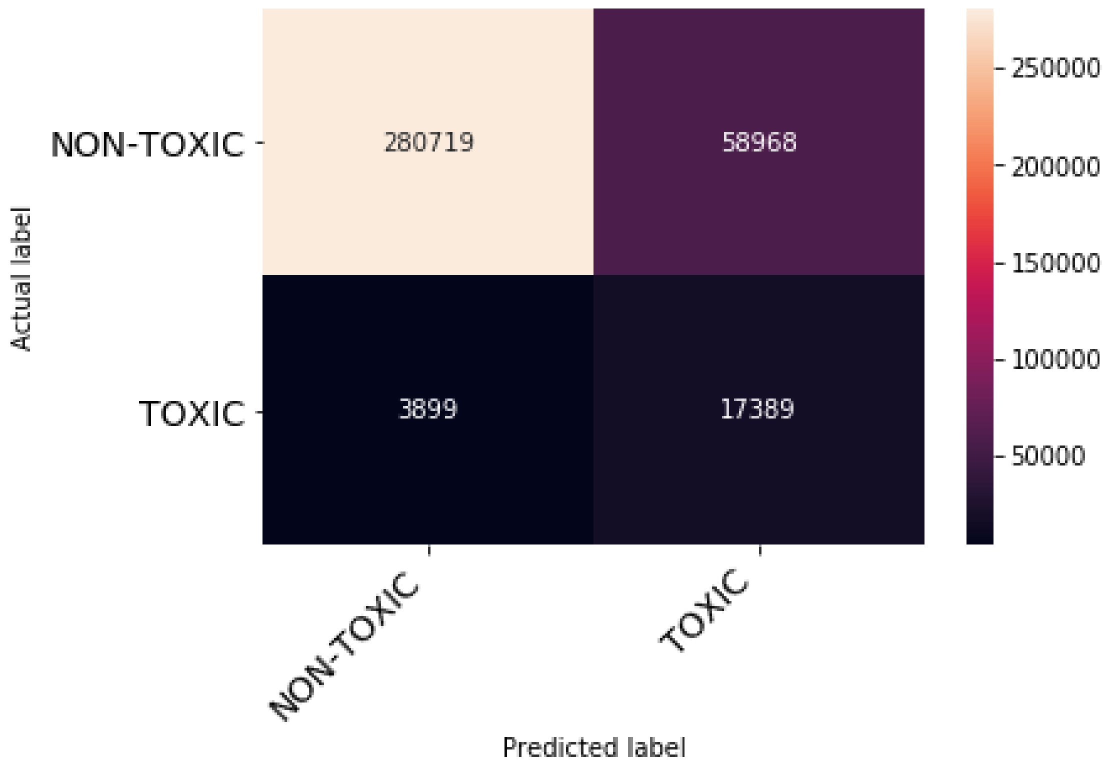
```
pred_train =␣
↪predict_with_best_t(predicted_train,tpr_train,fpr_train,threshold_train)
cm = confusion_matrix(y_train, pred_train)
print("\tTRAIN DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	TRAIN DATA CONFUSION MATRIX
```




[146]:
```
pred_test =␣
↪predict_with_best_t(predicted_validation,tpr_test,fpr_test,threshold_test)
cm = confusion_matrix(y_validation, pred_test)
print("\ttest DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	test DATA CONFUSION MATRIX
```




**10000 features**

[147]:
```
alpha = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10]
train_auc_list = []
validation_auc_list = []
names = []
for param in tqdm(alpha):
    MODEL_NAME = f'LR-tfidf_10k_{param}'
    clf = SGDClassifier(alpha=param, class_weight='balanced', loss='log',␣
 ↪penalty='l2')
    clf.fit(train_comment_tfidf_10000, y_train)
#     clf = CalibratedClassifierCV(clf, method="sigmoid")
#     clf.fit(train_comment_tfidf_10000, y_train)
    predicted_train = clf.predict_proba(train_comment_tfidf_10000)[:,1]
    predicted_validation = clf.predict_proba(validation_comment_tfidf_10000)[:
 ↪,1]

    train_data[MODEL_NAME] = predicted_train
    validation_data[MODEL_NAME] = predicted_validation

    train_auc_list.append(get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME))
```

```
    validation_auc_list.append(get_metric_value(validation_data,␣
↪identity_columns, MODEL_NAME))
    names.append(MODEL_NAME)
```

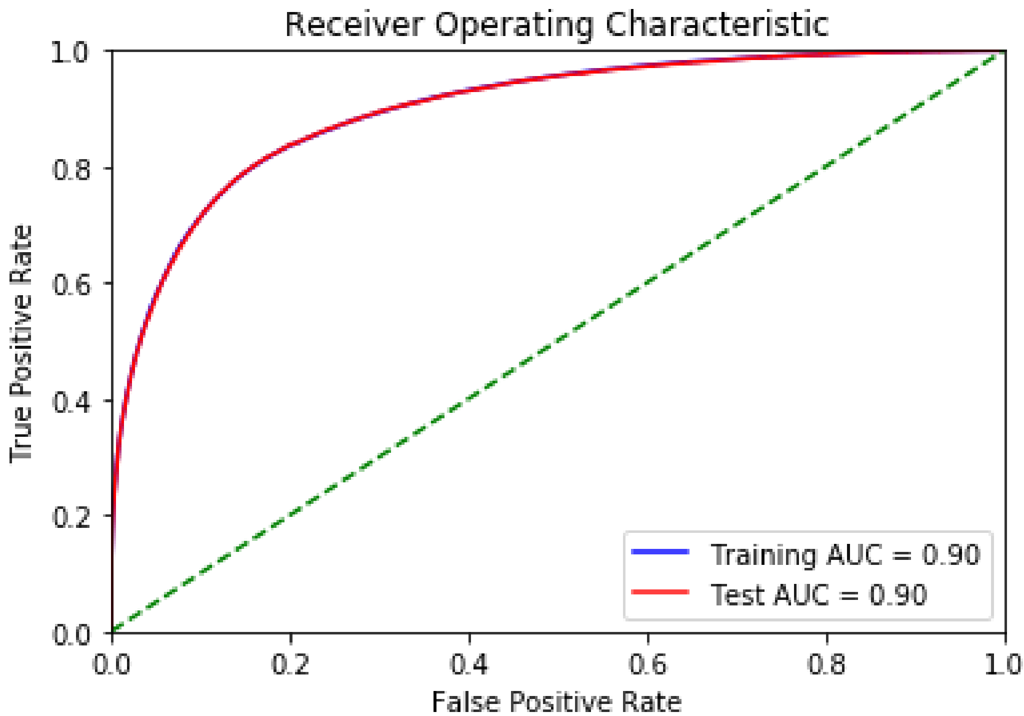
```
100%|     | 7/7 [01:42<00:00, 14.67s/it]
```

[148]:
```
pd.DataFrame({'name':names, 'train-score':train_auc_list, 'test-score':
↪validation_auc_list}).sort_values(by=['test-score'])
```

[148]:

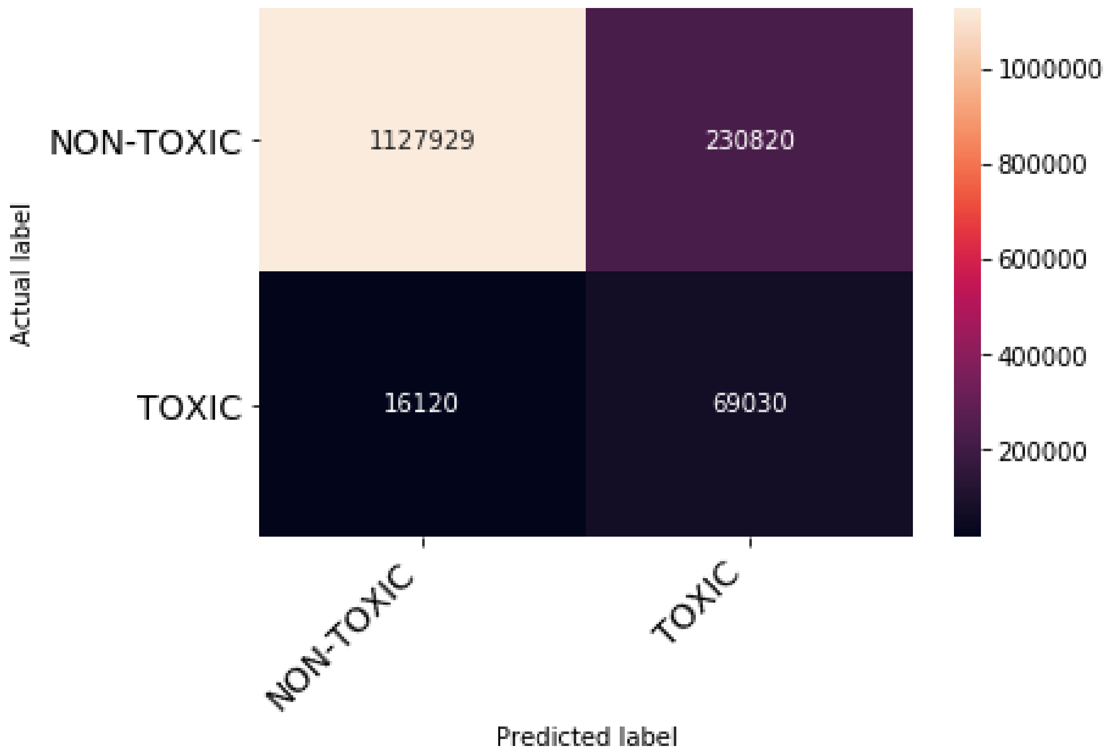
| | name | train-score | test-score |
|---|---|---|---|
| 6 | LR-tfidf_10k_10 | 0.849295 | 0.844280 |
| 4 | LR-tfidf_10k_0.1 | 0.848905 | 0.844328 |
| 5 | LR-tfidf_10k_1 | 0.849367 | 0.844368 |
| 3 | LR-tfidf_10k_0.01 | 0.850548 | 0.845328 |
| 2 | LR-tfidf_10k_0.001 | 0.862988 | 0.857688 |
| 1 | LR-tfidf_10k_0.0001 | 0.890238 | 0.883962 |
| 0 | LR-tfidf_10k_1e-05 | 0.909149 | 0.900014 |

[149]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
↪predicted_validation)

roc_auc_train = auc(fpr_train, tpr_train)
roc_auc_test = auc(fpr_test, tpr_test)


plt.title('Receiver Operating Characteristic')

plt.plot(fpr_train, tpr_train, 'b', label = 'Training AUC = %0.2f' %␣
↪roc_auc_train)
plt.plot(fpr_test, tpr_test, 'r', label = 'Test AUC = %0.2f' % roc_auc_test)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



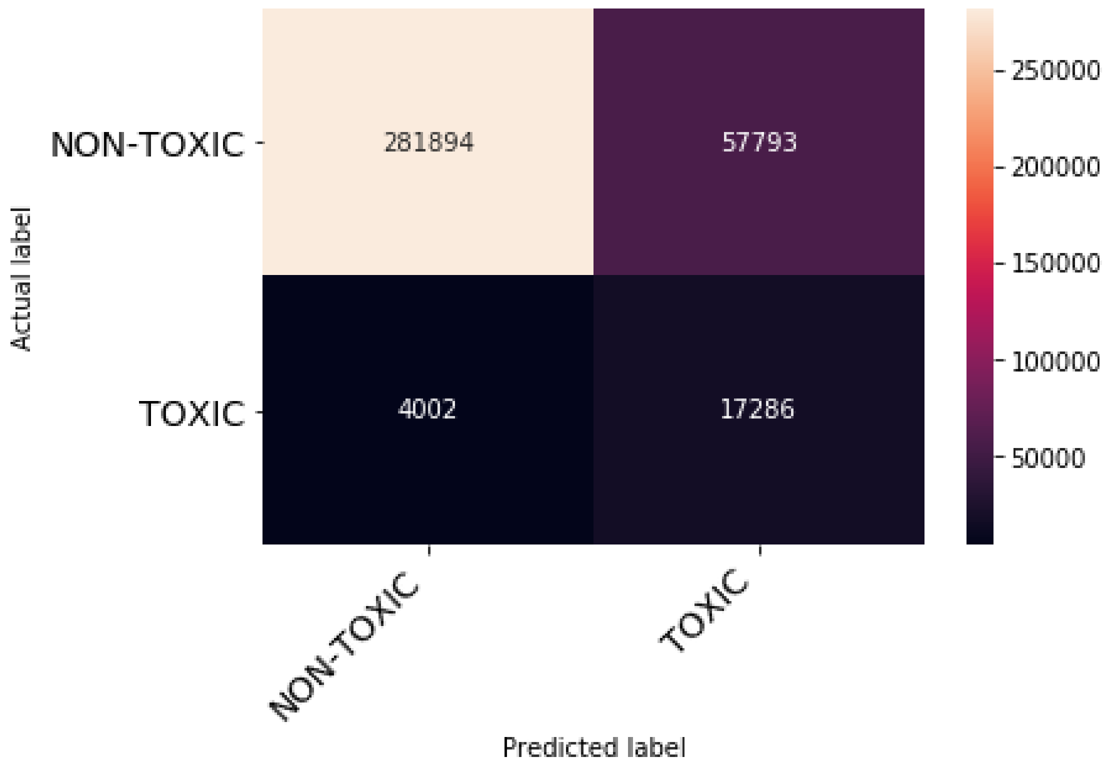

[150]:
```
pred_train =␣
↪predict_with_best_t(predicted_train,tpr_train,fpr_train,threshold_train)
cm = confusion_matrix(y_train, pred_train)
print("\tTRAIN DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	TRAIN DATA CONFUSION MATRIX
```




[151]:
```
pred_test =␣
↪predict_with_best_t(predicted_validation,tpr_test,fpr_test,threshold_test)
cm = confusion_matrix(y_validation, pred_test)
print("\ttest DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
        test DATA CONFUSION MATRIX
```




[152]:
```
import gc
gc.collect()
```

[152]: 9982

### 5.1.3 SVM

**Considering TFIDF**

**25000 features**

[153]:
```
alpha = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10]
train_auc_list = []
validation_auc_list = []
names = []
for param in tqdm(alpha):
    MODEL_NAME = f'svm-tfidf_25k_{param}'
    clf = SGDClassifier(alpha=param, class_weight='balanced', loss='hinge',␣
 ↪penalty='l2')
    clf.fit(train_comment_tfidf_25000, y_train)
    clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```
    clf.fit(train_comment_tfidf_25000, y_train)
    predicted_train = clf.predict_proba(train_comment_tfidf_25000)[:,1]
    predicted_validation = clf.predict_proba(validation_comment_tfidf_25000)[:
↪,1]

    train_data[MODEL_NAME] = predicted_train
    validation_data[MODEL_NAME] = predicted_validation

    train_auc_list.append(get_metric_value(train_data, identity_columns,␣
↪MODEL_NAME))
    validation_auc_list.append(get_metric_value(validation_data,␣
↪identity_columns, MODEL_NAME))
    names.append(MODEL_NAME)
```

```
100%|      | 7/7 [04:07<00:00, 35.38s/it]
```

[154]:

```
pd.DataFrame({'name':names, 'train-score':train_auc_list, 'test-score':
↪validation_auc_list}).sort_values(by=['test-score'])
```

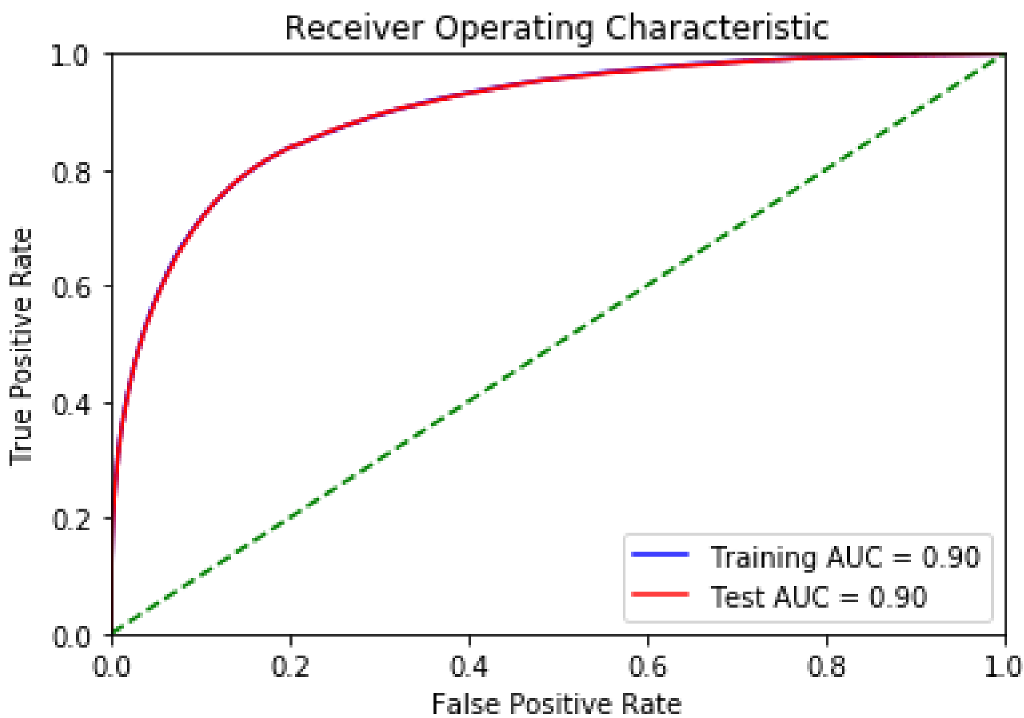
[154]:

| | name | train-score | test-score |
|---|---|---|---|
| 3 | svm-tfidf_25k_0.01 | 0.849373 | 0.844324 |
| 4 | svm-tfidf_25k_0.1 | 0.849521 | 0.844395 |
| 5 | svm-tfidf_25k_1 | 0.849521 | 0.844395 |
| 6 | svm-tfidf_25k_10 | 0.849521 | 0.844395 |
| 2 | svm-tfidf_25k_0.001 | 0.860354 | 0.854579 |
| 1 | svm-tfidf_25k_0.0001 | 0.894229 | 0.886785 |
| 0 | svm-tfidf_25k_1e-05 | 0.916734 | 0.903651 |

[155]:

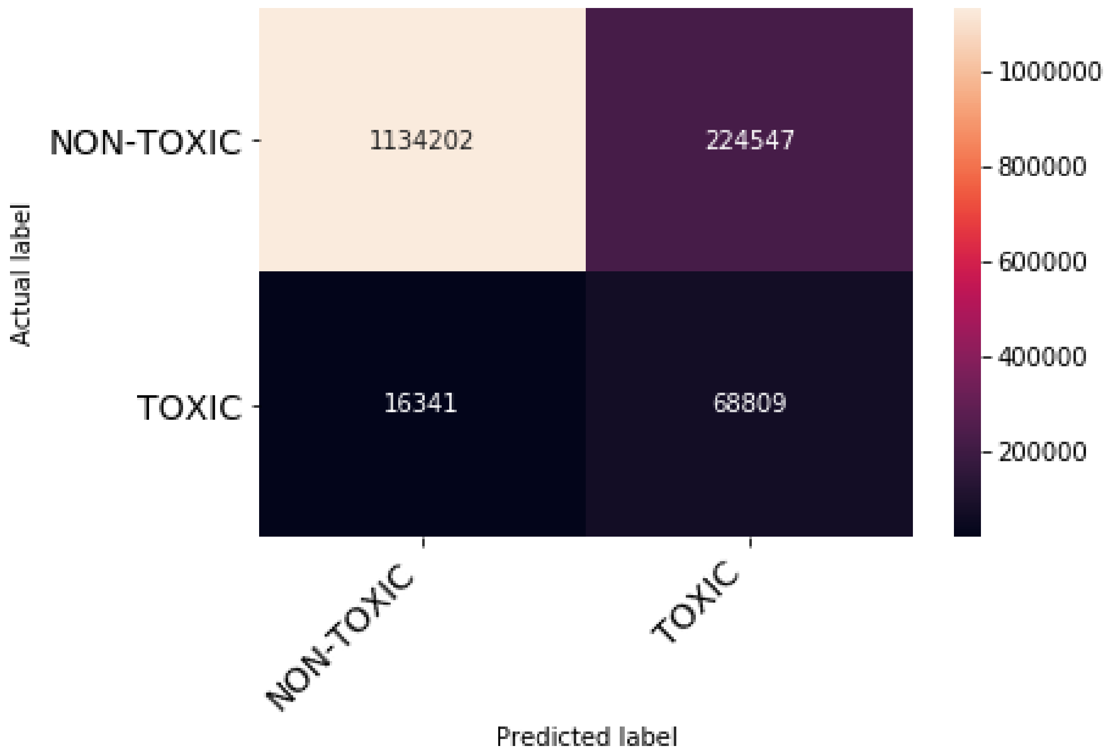
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
↪predicted_validation)

roc_auc_train = auc(fpr_train, tpr_train)
roc_auc_test = auc(fpr_test, tpr_test)


plt.title('Receiver Operating Characteristic')

plt.plot(fpr_train, tpr_train, 'b', label = 'Training AUC = %0.2f' %␣
↪roc_auc_train)
plt.plot(fpr_test, tpr_test, 'r', label = 'Test AUC = %0.2f' % roc_auc_test)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
```

```
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



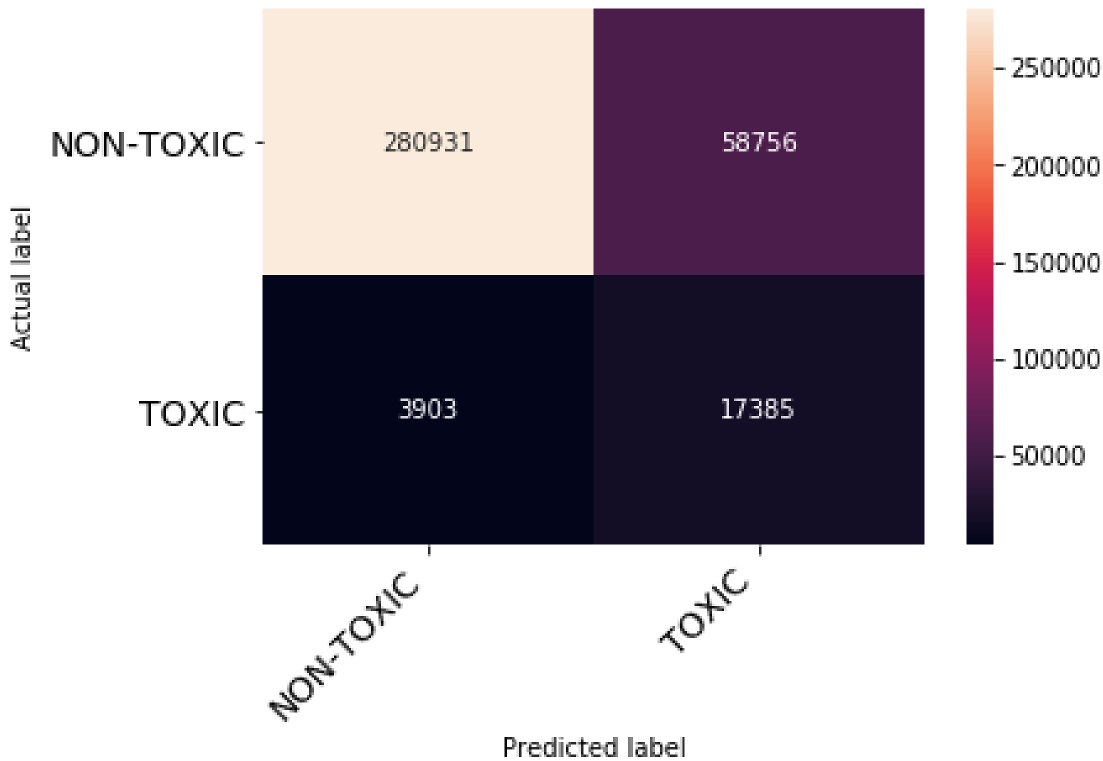

[156]:
```
pred_train =␣
 ↪predict_with_best_t(predicted_train,tpr_train,fpr_train,threshold_train)
cm = confusion_matrix(y_train, pred_train)
print("\tTRAIN DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
        TRAIN DATA CONFUSION MATRIX
```




[157]:
```
pred_test =␣
 ↪predict_with_best_t(predicted_validation,tpr_test,fpr_test,threshold_test)
cm = confusion_matrix(y_validation, pred_test)
print("\ttest DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
        test DATA CONFUSION MATRIX
```




**15000 features**

[158]:
```
alpha = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10]
train_auc_list = []
validation_auc_list = []
names = []
for param in tqdm(alpha):
    MODEL_NAME = f'svm-tfidf_15k_{param}'
    clf = SGDClassifier(alpha=param, class_weight='balanced', loss='hinge',␣
 ↪penalty='l2')
    clf.fit(train_comment_tfidf_15000, y_train)
    clf = CalibratedClassifierCV(clf, method="sigmoid")
    clf.fit(train_comment_tfidf_15000, y_train)
    predicted_train = clf.predict_proba(train_comment_tfidf_15000)[:,1]
    predicted_validation = clf.predict_proba(validation_comment_tfidf_15000)[:
 ↪,1]
    train_data[MODEL_NAME] = predicted_train
    validation_data[MODEL_NAME] = predicted_validation

    train_auc_list.append(get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME))
```

```
    validation_auc_list.append(get_metric_value(validation_data,␣
↪identity_columns, MODEL_NAME))
    names.append(MODEL_NAME)
```

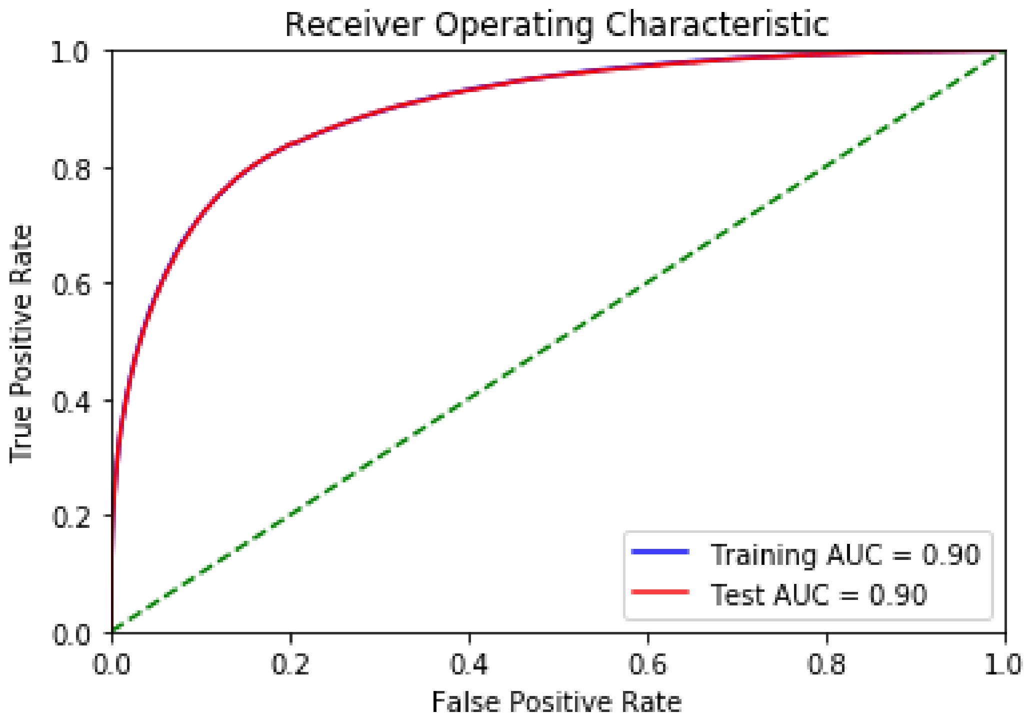
```
100%|          | 7/7 [04:03<00:00, 34.73s/it]
```

[159]:
```
pd.DataFrame({'name':names, 'train-score':train_auc_list, 'test-score':
↪validation_auc_list}).sort_values(by=['test-score'])
```

[159]:

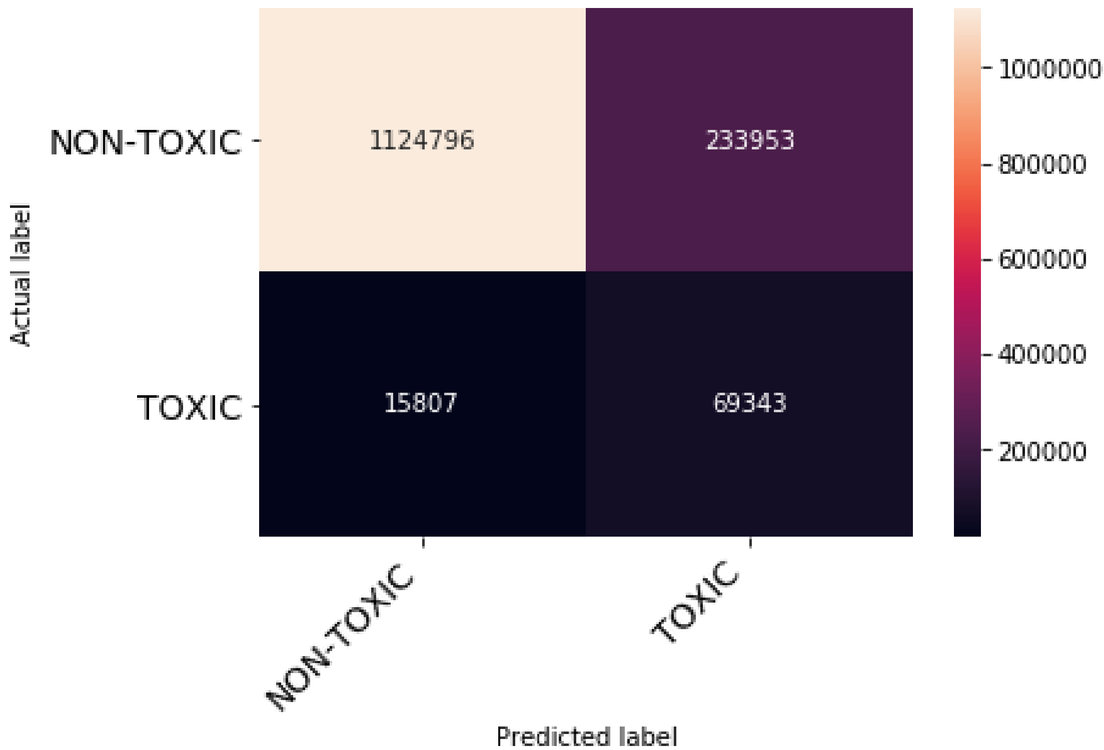
| | name | train-score | test-score |
|---|---|---|---|
| 3 | svm-tfidf_15k_0.01 | 0.849661 | 0.844536 |
| 4 | svm-tfidf_15k_0.1 | 0.849689 | 0.844552 |
| 5 | svm-tfidf_15k_1 | 0.849689 | 0.844552 |
| 6 | svm-tfidf_15k_10 | 0.849689 | 0.844552 |
| 2 | svm-tfidf_15k_0.001 | 0.861176 | 0.855522 |
| 1 | svm-tfidf_15k_0.0001 | 0.894382 | 0.887301 |
| 0 | svm-tfidf_15k_1e-05 | 0.914568 | 0.902665 |

[160]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
↪predicted_validation)

roc_auc_train = auc(fpr_train, tpr_train)
roc_auc_test = auc(fpr_test, tpr_test)


plt.title('Receiver Operating Characteristic')

plt.plot(fpr_train, tpr_train, 'b', label = 'Training AUC = %0.2f' %␣
↪roc_auc_train)
plt.plot(fpr_test, tpr_test, 'r', label = 'Test AUC = %0.2f' % roc_auc_test)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



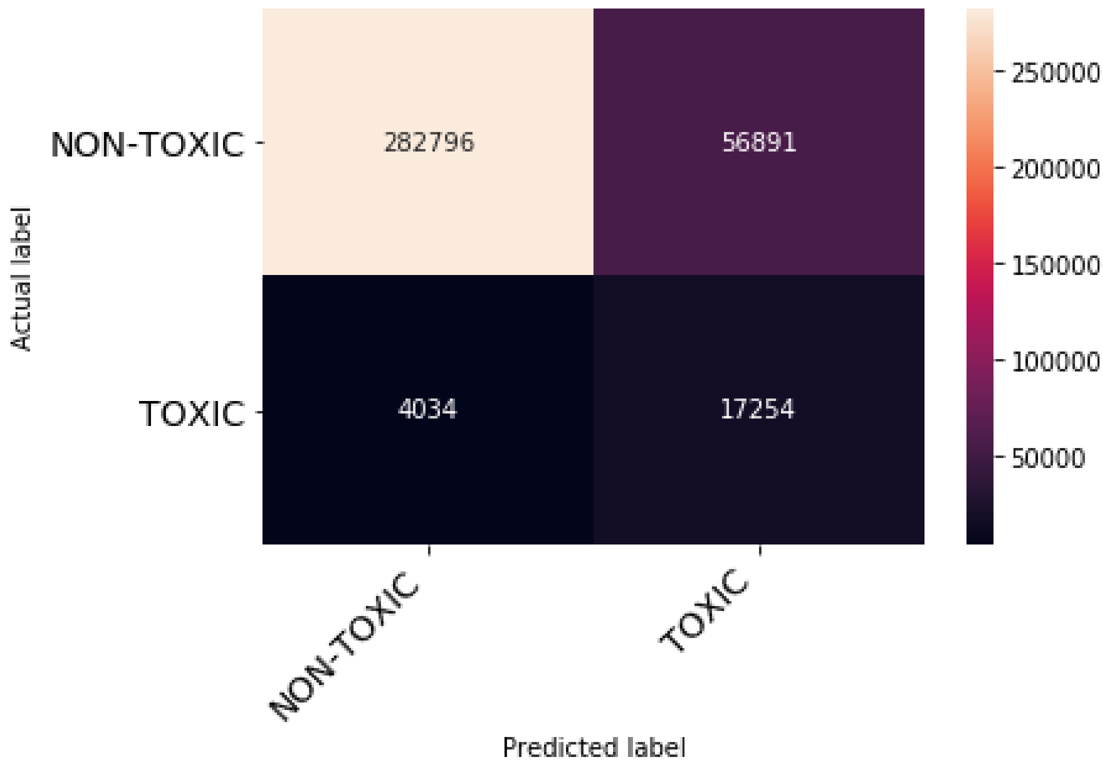

[161]:
```
pred_train =␣
↪predict_with_best_t(predicted_train,tpr_train,fpr_train,threshold_train)
cm = confusion_matrix(y_train, pred_train)
print("\tTRAIN DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	TRAIN DATA CONFUSION MATRIX
```




```
[162]: pred_test =␣
        ↪predict_with_best_t(predicted_validation,tpr_test,fpr_test,threshold_test)
       cm = confusion_matrix(y_validation, pred_test)
       print("\ttest DATA CONFUSION MATRIX")
       plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
        test DATA CONFUSION MATRIX
```




**10000 features**

[163]:
```
alpha = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10]
train_auc_list = []
validation_auc_list = []
names = []
for param in tqdm(alpha):
    MODEL_NAME = f'svm-tfidf_10k_{param}'
    clf = SGDClassifier(alpha=param, class_weight='balanced', loss='hinge',␣
 ↪penalty='l2')
    clf.fit(train_comment_tfidf_10000, y_train)
    clf = CalibratedClassifierCV(clf, method="sigmoid")
    clf.fit(train_comment_tfidf_10000, y_train)
    predicted_train = clf.predict_proba(train_comment_tfidf_10000)[:,1]
    predicted_validation = clf.predict_proba(validation_comment_tfidf_10000)[:
 ↪,1]
    train_data[MODEL_NAME] = predicted_train
    validation_data[MODEL_NAME] = predicted_validation

    train_auc_list.append(get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME))
```

```
    validation_auc_list.append(get_metric_value(validation_data,␣
↪identity_columns, MODEL_NAME))
    names.append(MODEL_NAME)
```

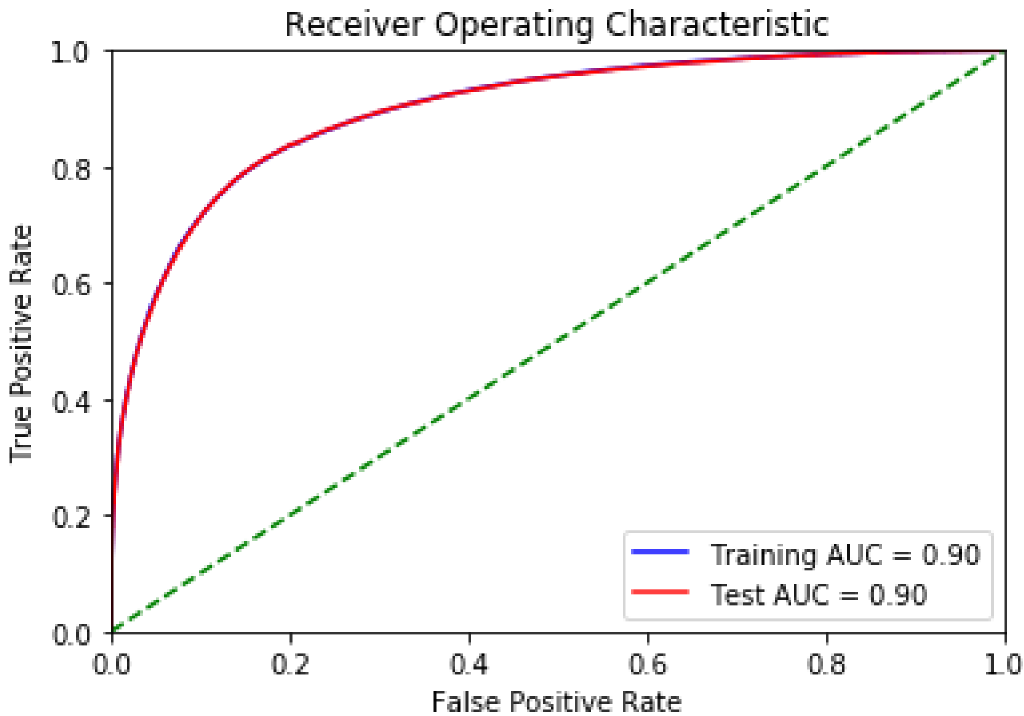
```
100%|          | 7/7 [04:04<00:00, 34.93s/it]
```

[164]:
```
pd.DataFrame({'name':names, 'train-score':train_auc_list, 'test-score':
 ↪validation_auc_list}).sort_values(by=['test-score'])
```

[164]:

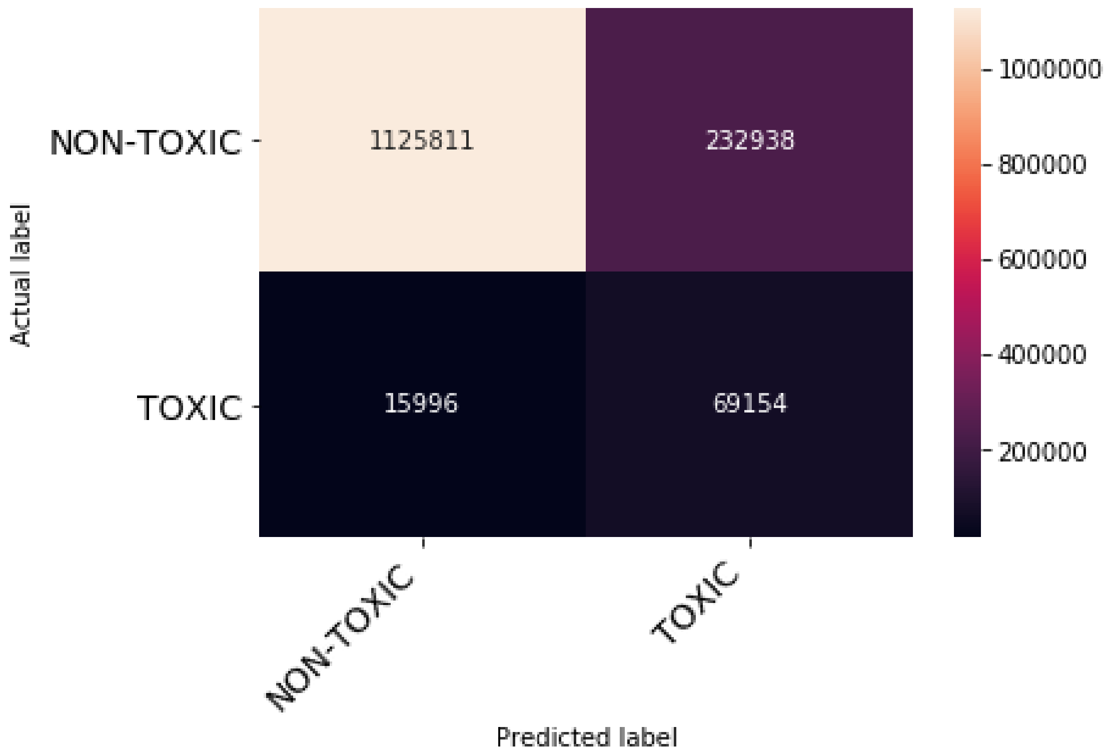
| | name | train-score | test-score |
|---|---|---|---|
| 3 | svm-tfidf_10k_0.01 | 0.849197 | 0.844213 |
| 6 | svm-tfidf_10k_10 | 0.849305 | 0.844292 |
| 4 | svm-tfidf_10k_0.1 | 0.849305 | 0.844292 |
| 5 | svm-tfidf_10k_1 | 0.849305 | 0.844292 |
| 2 | svm-tfidf_10k_0.001 | 0.861578 | 0.856153 |
| 1 | svm-tfidf_10k_0.0001 | 0.892689 | 0.886083 |
| 0 | svm-tfidf_10k_1e-05 | 0.910408 | 0.899807 |

[165]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
 ↪predicted_validation)

roc_auc_train = auc(fpr_train, tpr_train)
roc_auc_test = auc(fpr_test, tpr_test)


plt.title('Receiver Operating Characteristic')

plt.plot(fpr_train, tpr_train, 'b', label = 'Training AUC = %0.2f' %␣
 ↪roc_auc_train)
plt.plot(fpr_test, tpr_test, 'r', label = 'Test AUC = %0.2f' % roc_auc_test)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



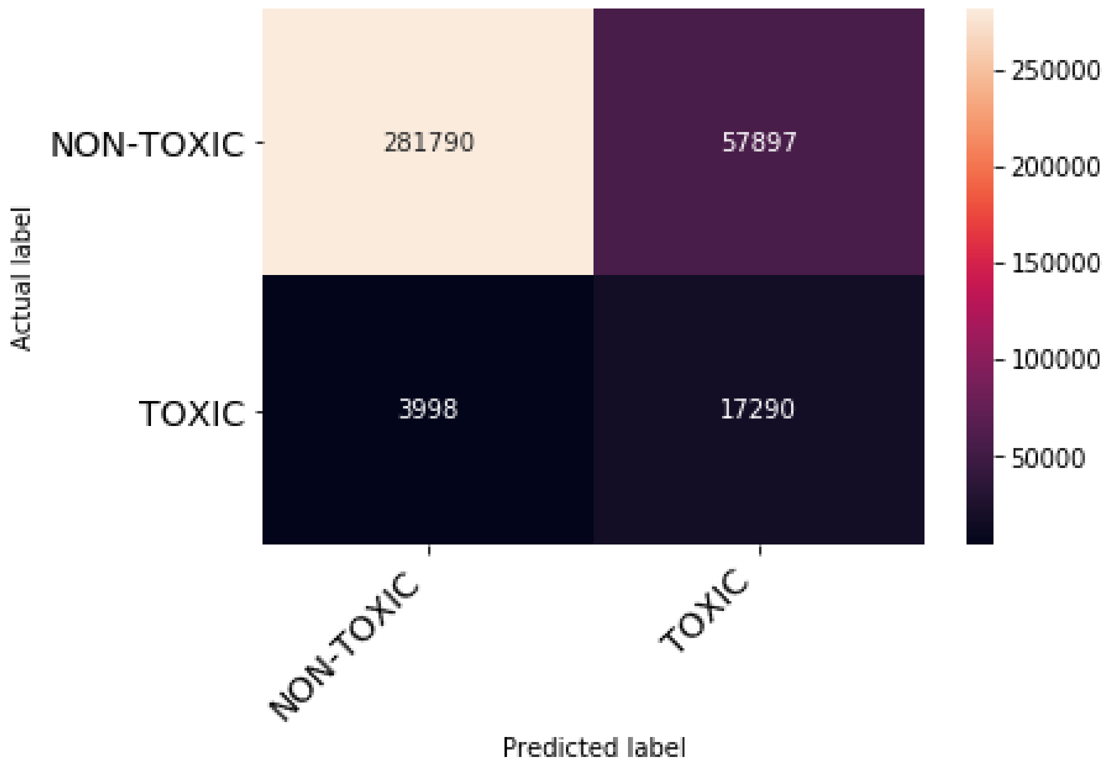

[166]:
```
pred_train =␣
↪predict_with_best_t(predicted_train,tpr_train,fpr_train,threshold_train)
cm = confusion_matrix(y_train, pred_train)
print("\tTRAIN DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	TRAIN DATA CONFUSION MATRIX
```




[167]:
```
pred_test =␣
↪predict_with_best_t(predicted_validation,tpr_test,fpr_test,threshold_test)
cm = confusion_matrix(y_validation, pred_test)
print("\ttest DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	test DATA CONFUSION MATRIX
```

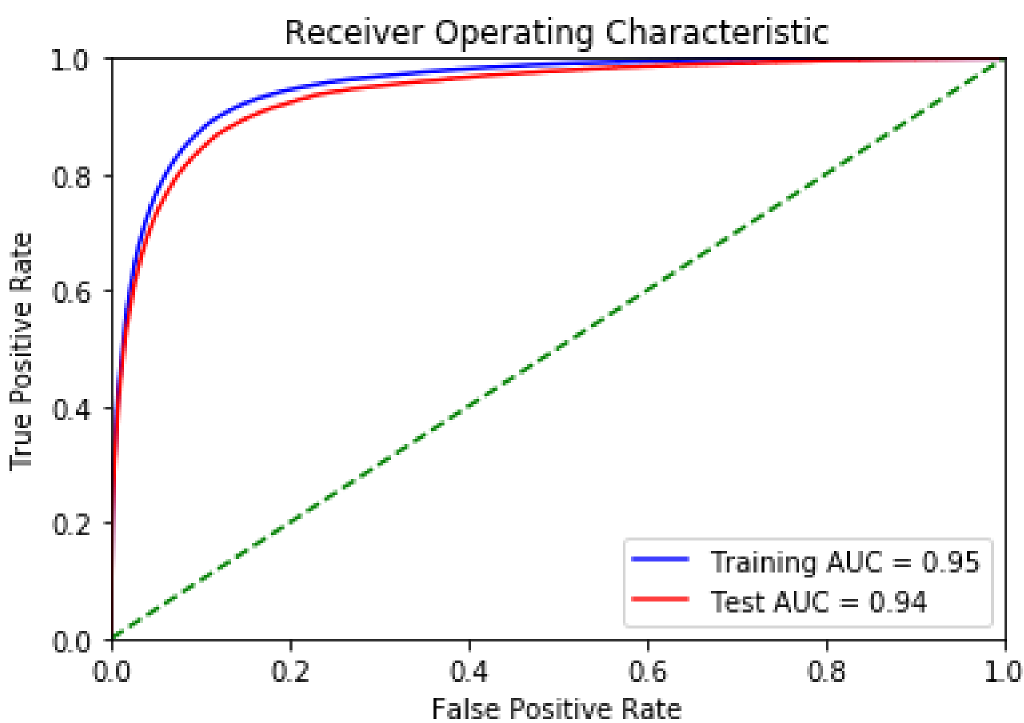



### 5.1.4 XG-Boost

[168]:
```
# train_auc_list = []
# validation_auc_list = []
MODEL_NAME = f'xgb_10k'
clf = XGBClassifier(scale_pos_weight=99,n_estimators=2000, n_jobs=-1)
clf.fit(train_comment_tfidf_10000, y_train)
# clf = CalibratedClassifierCV(clf, method="sigmoid")
# clf.fit(train_comment_tfidf_10000, y_train)
predicted_train = clf.predict_proba(train_comment_tfidf_10000)[:,1]
predicted_validation = clf.predict_proba(validation_comment_tfidf_10000)[:,1]
train_data[MODEL_NAME] = predicted_train
validation_data[MODEL_NAME] = predicted_validation

print(get_metric_value(train_data, identity_columns, MODEL_NAME))
print(get_metric_value(validation_data, identity_columns, MODEL_NAME))
```

```
0.9105878029913056
0.8873125814205747
```

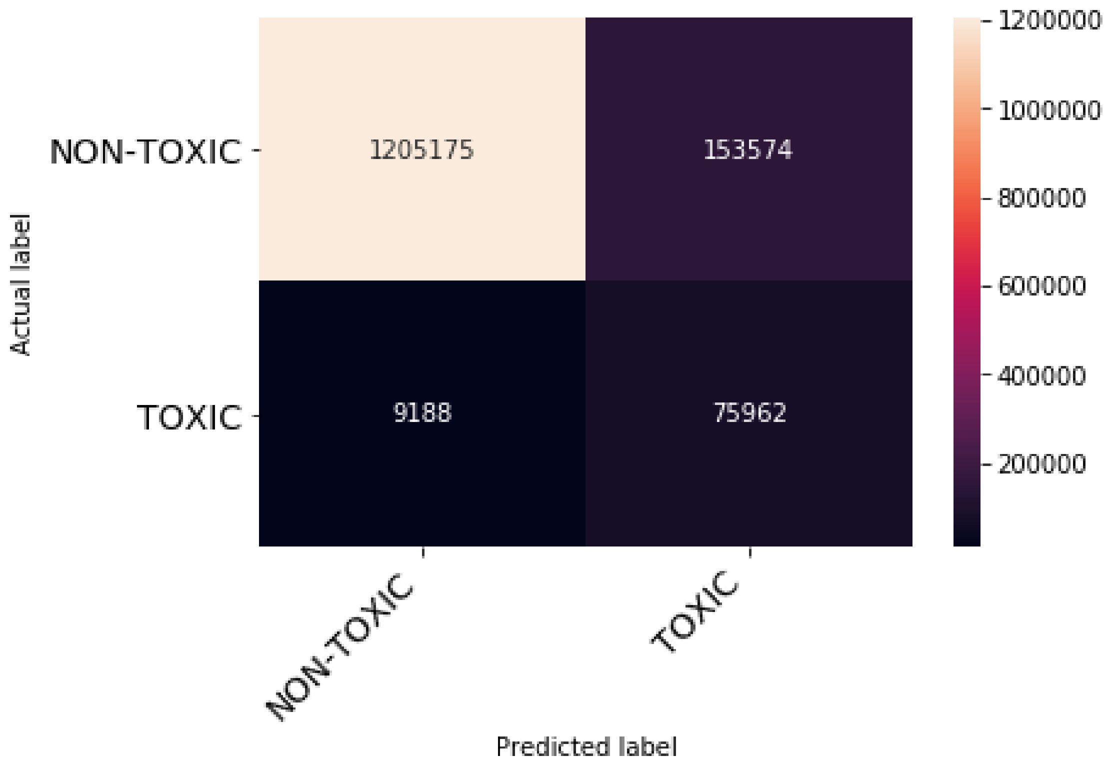
[169]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
↪predicted_validation)

roc_auc_train = auc(fpr_train, tpr_train)
roc_auc_test = auc(fpr_test, tpr_test)


plt.title('Receiver Operating Characteristic')

plt.plot(fpr_train, tpr_train, 'b', label = 'Training AUC = %0.2f' %␣
↪roc_auc_train)
plt.plot(fpr_test, tpr_test, 'r', label = 'Test AUC = %0.2f' % roc_auc_test)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



[170]:

```
pred_train = predict_with_best_t(predicted_train,tpr_train,fpr_train,threshold_train)
cm = confusion_matrix(y_train, pred_train)
print("\tTRAIN DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

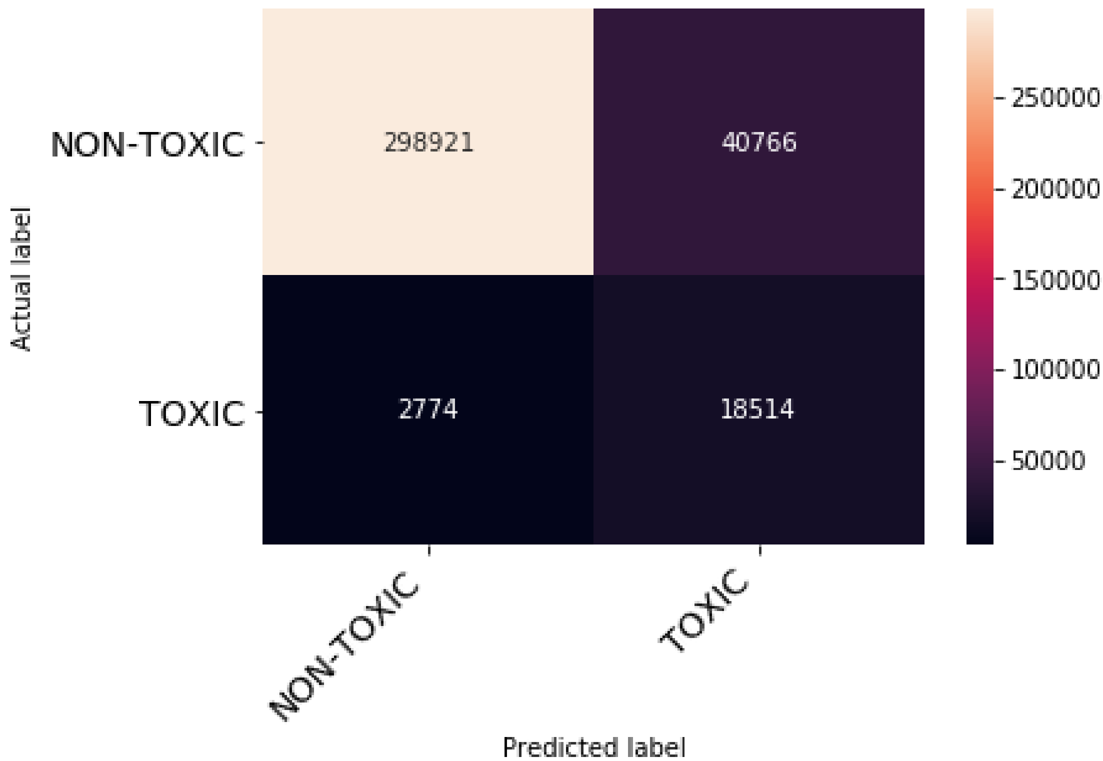
```
	TRAIN DATA CONFUSION MATRIX
```




[171]:

```
pred_test = predict_with_best_t(predicted_validation,tpr_test,fpr_test,threshold_test)
cm = confusion_matrix(y_validation, pred_test)
print("\ttest DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	test DATA CONFUSION MATRIX
```

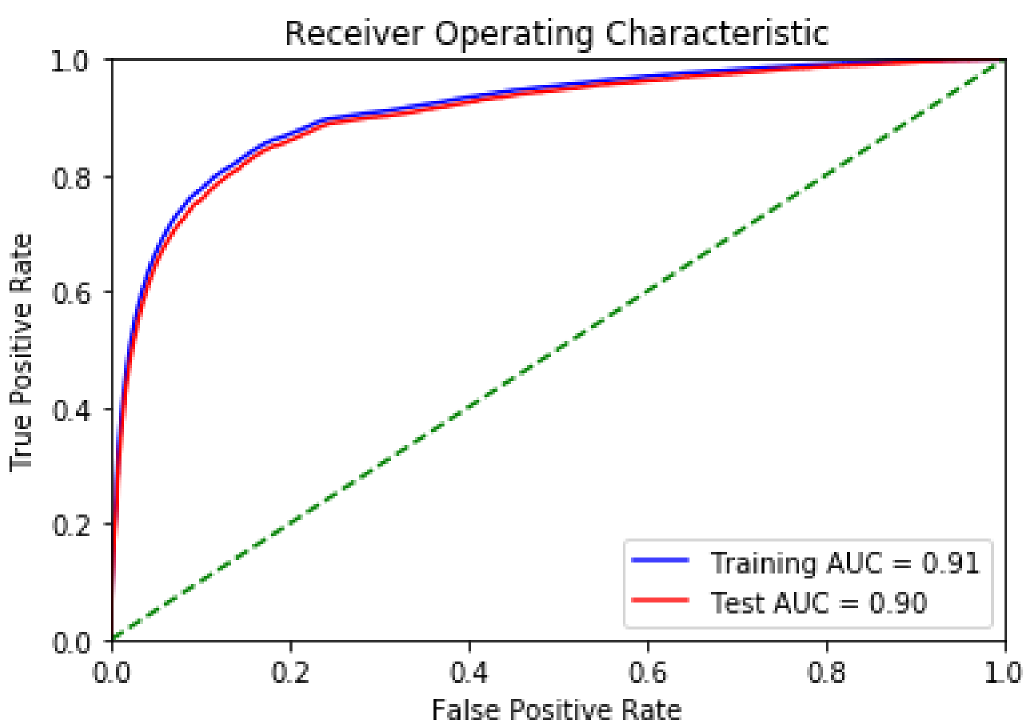



### 5.1.5 RandomForest Classifier

[28]:
```
n_estimators = 2000
max_depth= 12
n_jobs = -1
class_weight = 'balanced'

MODEL_NAME = f'RF-tfidf_10k'
clf = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth,␣
 ↪class_weight=class_weight, n_jobs=n_jobs)
clf.fit(train_comment_tfidf_10000, y_train)
clf = CalibratedClassifierCV(clf, method="sigmoid")
clf.fit(train_comment_tfidf_10000, y_train)
predicted_train = clf.predict_proba(train_comment_tfidf_10000)[:,1]
predicted_validation = clf.predict_proba(validation_comment_tfidf_10000)[:,1]

train_data[MODEL_NAME] = predicted_train
validation_data[MODEL_NAME] = predicted_validation

print(get_metric_value(train_data, identity_columns, MODEL_NAME))
print(get_metric_value(validation_data, identity_columns, MODEL_NAME))
```

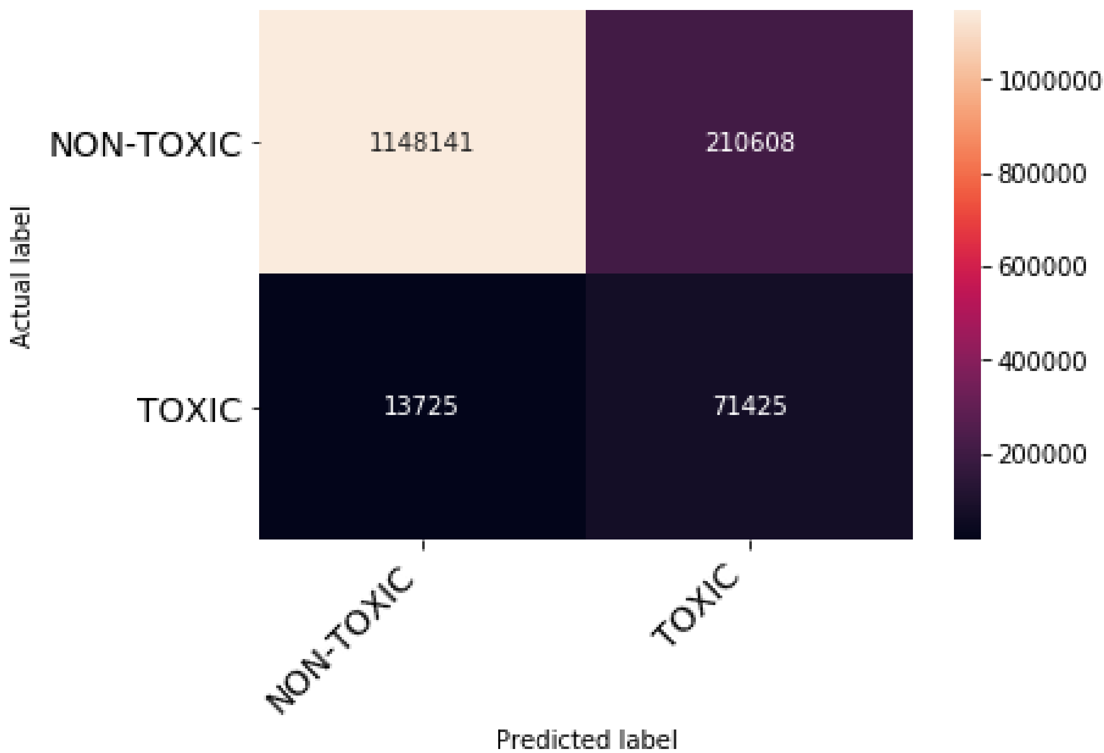
```
0.8551017671999488
0.8390758559337348
```

[29]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
 ↪predicted_validation)

roc_auc_train = auc(fpr_train, tpr_train)
roc_auc_test = auc(fpr_test, tpr_test)


plt.title('Receiver Operating Characteristic')

plt.plot(fpr_train, tpr_train, 'b', label = 'Training AUC = %0.2f' %␣
 ↪roc_auc_train)
plt.plot(fpr_test, tpr_test, 'r', label = 'Test AUC = %0.2f' % roc_auc_test)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



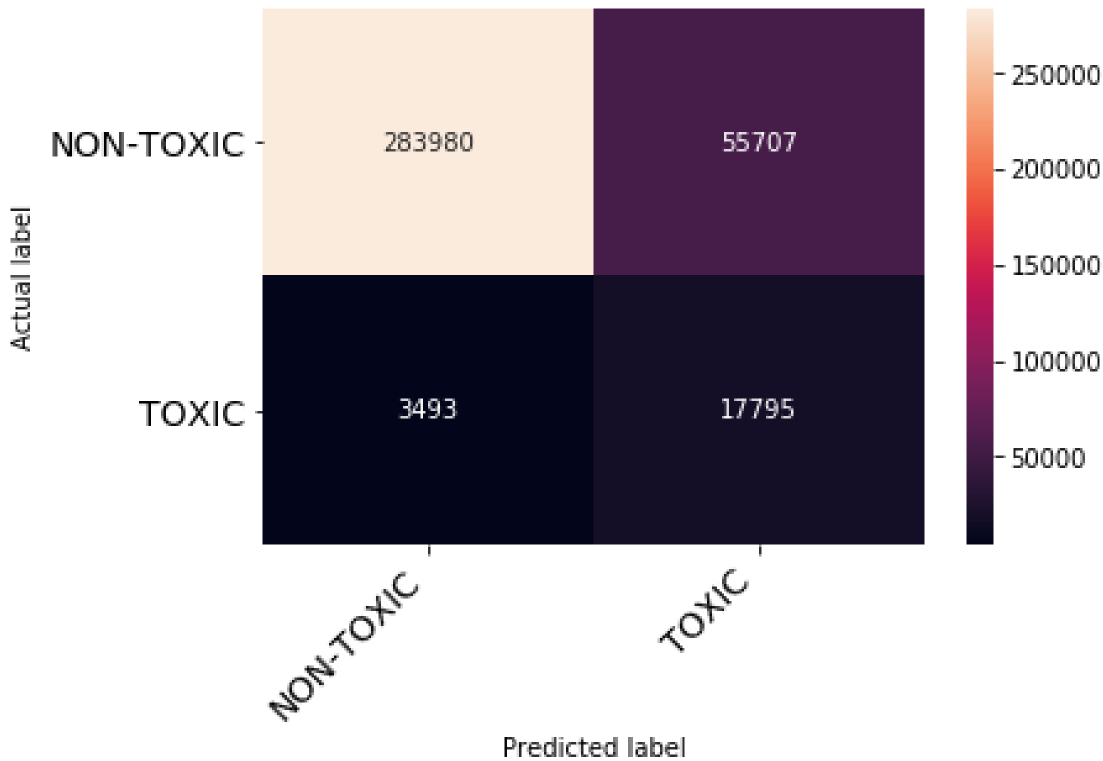
[30]:
```
pred_train = ␣
↪predict_with_best_t(predicted_train,tpr_train,fpr_train,threshold_train)
cm = confusion_matrix(y_train, pred_train)
print("\tTRAIN DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	TRAIN DATA CONFUSION MATRIX
```




[31]:
```
pred_test = ␣
↪predict_with_best_t(predicted_validation,tpr_test,fpr_test,threshold_test)
cm = confusion_matrix(y_validation, pred_test)
print("\ttest DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	test DATA CONFUSION MATRIX
```




#### 5.1.6 Stacking Classifier

**Models with best hyperparameters**

[28]:
```
nb_model = clf = MultinomialNB(alpha=1)
logistic_model = SGDClassifier(alpha=1e-5, class_weight='balanced', loss='log',␣
 ↪penalty='l2')
svm_model = SGDClassifier(alpha=1e-5, class_weight='balanced', loss='hinge',␣
 ↪penalty='l2')
xg_model = XGBClassifier(scale_pos_weight=99,n_estimators=2000, n_jobs=-1)
rf_model = RandomForestClassifier(n_estimators=1500, max_depth=12,␣
 ↪class_weight='balanced')
```

[29]:
```
import gc
gc.collect()
```

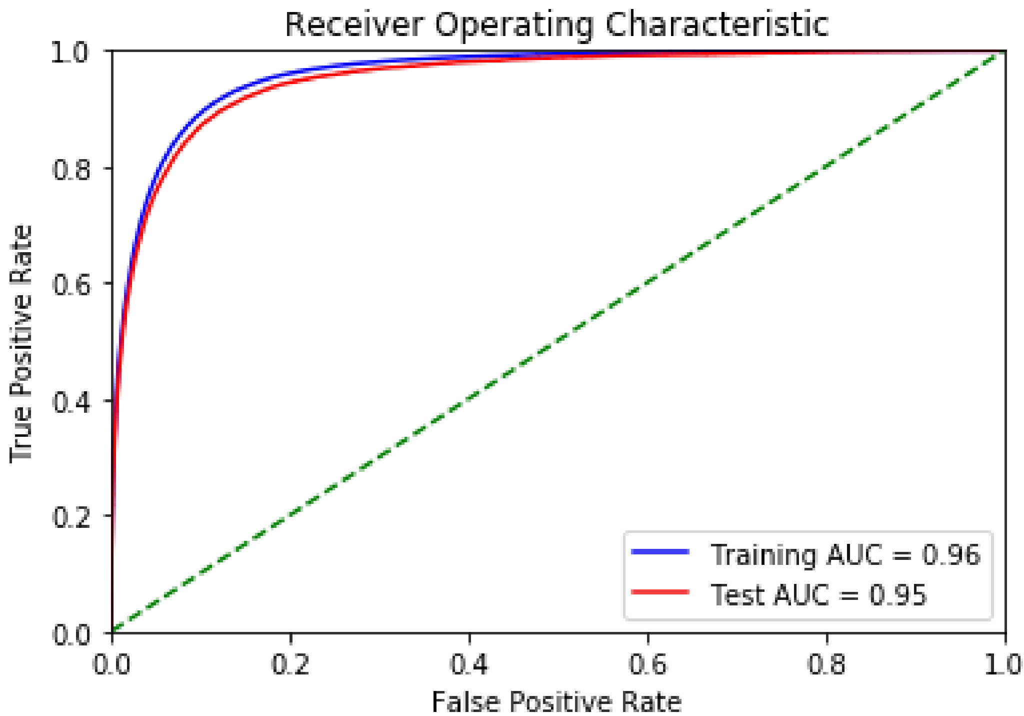
[29]: 367

**Stacking models**

[24]:
```
estimators = [
    ('nb', nb_model),
    ('lr', logistic_model),
    ('xg', xg_model),
    ('svm', CalibratedClassifierCV(svm_model, method='sigmoid'))
]
clf = StackingClassifier(
    estimators=estimators, final_estimator=LogisticRegression(), n_jobs=-1,␣
 ↪verbose=5
)
clf.fit(train_comment_tfidf_15000, y_train)

predicted_train = clf.predict_proba(train_comment_tfidf_15000)[:,1]
predicted_validation = clf.predict_proba(validation_comment_tfidf_15000)[:,1]

MODEL_NAME = 'stacking'
train_data[MODEL_NAME] = predicted_train
validation_data[MODEL_NAME] = predicted_validation
print(get_metric_value(train_data, identity_columns, MODEL_NAME))
print(get_metric_value(validation_data, identity_columns, MODEL_NAME))
```

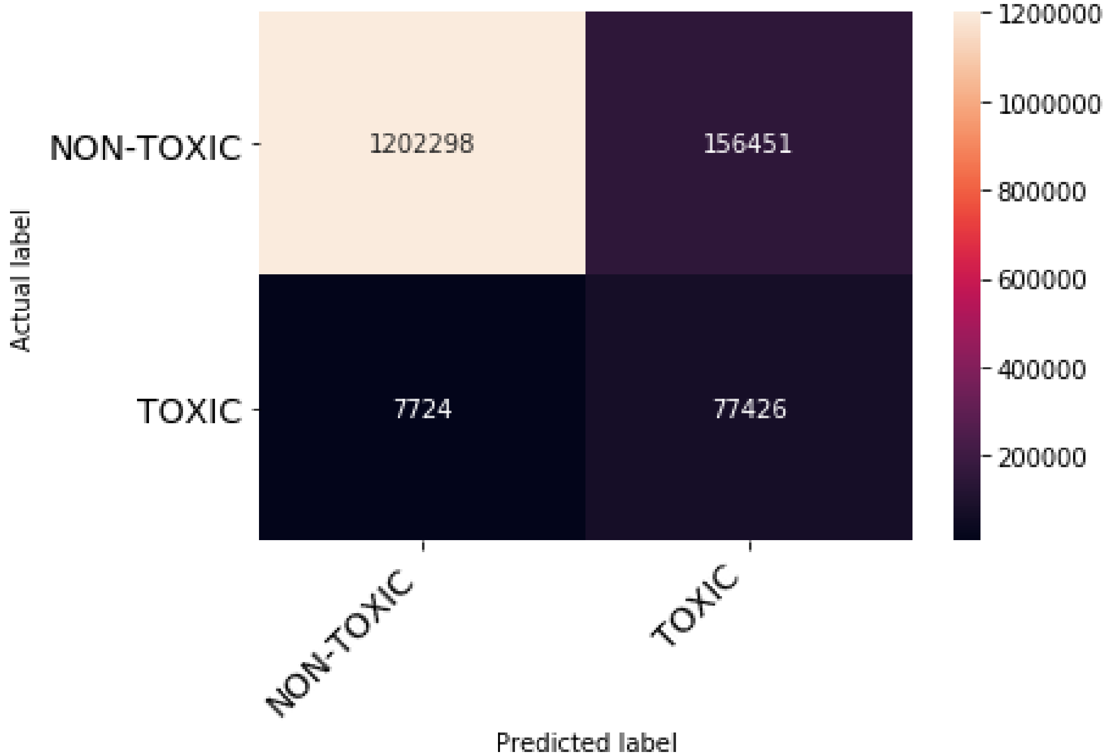
```
0.9168966098292373
0.9057013945975785
```

[25]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
 ↪predicted_validation)

roc_auc_train = auc(fpr_train, tpr_train)
roc_auc_test = auc(fpr_test, tpr_test)


plt.title('Receiver Operating Characteristic')

plt.plot(fpr_train, tpr_train, 'b', label = 'Training AUC = %0.2f' %␣
 ↪roc_auc_train)
plt.plot(fpr_test, tpr_test, 'r', label = 'Test AUC = %0.2f' % roc_auc_test)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



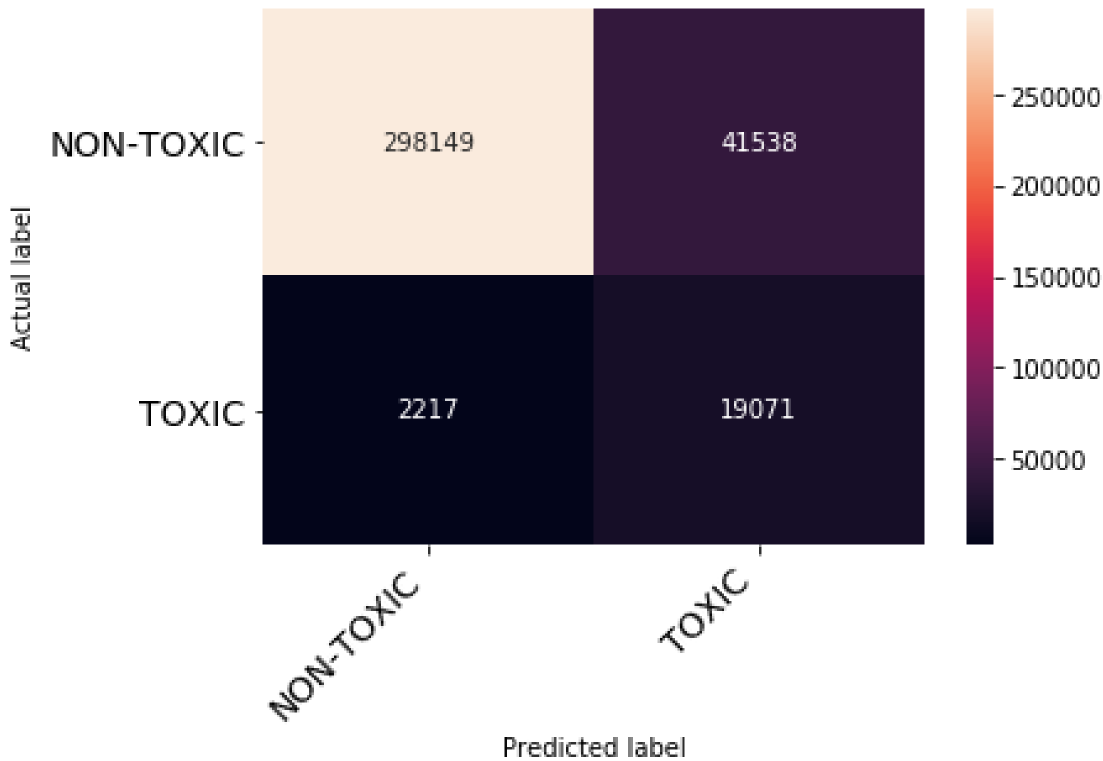

[26]:
```
pred_train =␣
↪predict_with_best_t(predicted_train,tpr_train,fpr_train,threshold_train)
cm = confusion_matrix(y_train, pred_train)
print("\tTRAIN DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	TRAIN DATA CONFUSION MATRIX
```




[27]:
```
pred_test =␣
 ↪predict_with_best_t(predicted_validation,tpr_test,fpr_test,threshold_test)
cm = confusion_matrix(y_validation, pred_test)
print("\ttest DATA CONFUSION MATRIX")
plot_confusion_matrix(cm,class_names=['NON-TOXIC','TOXIC'])
```

```
	test DATA CONFUSION MATRIX
```




# 6 Deep Learning Models

[43]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from keras import backend as K
import keras
print(tf.__version__)
# tf.compat.v1.disable_v2_behavior()
from sklearn.model_selection import train_test_split
from keras.models import Model
from keras.layers import Input, Dense, Embedding, SpatialDropout1D, add,␣
↪concatenate
from keras.layers import LSTM, Bidirectional, GlobalMaxPooling1D,␣
↪GlobalAveragePooling1D, GRU
from keras.layers import Conv1D, MaxPooling1D, AveragePooling1D, Flatten,␣
↪Dropout, Bidirectional
from keras.utils import to_categorical, plot_model
from keras.preprocessing import text, sequence
from gensim.models import KeyedVectors
```

```
from tqdm import tqdm
import pickle
import gc
gc.collect()

import re
import nltk
nltk.download('punkt')
nltk.download('wordnet')
from nltk.stem.wordnet import WordNetLemmatizer
from nltk import word_tokenize
from nltk.stem import PorterStemmer

from IPython.display import Image,YouTubeVideo,HTML

from sklearn import metrics
```

[2]:
```
import logging
logger = logging.getLogger("distributed.worker")
logger1 = logging.getLogger("distributed.utils_perf")
logger.setLevel(logging.ERROR)
logger1.setLevel(logging.ERROR)
```

[3]:
```
from dask.distributed import Client, progress
client = Client(processes=False, threads_per_worker=12, n_workers=1,␣
 ↪memory_limit='6GB')
client
```

```
/home/user/anaconda3/lib/python3.7/site-
packages/distributed/dashboard/core.py:79: UserWarning:
Port 8787 is already in use.
Perhaps you already have a cluster running?
Hosting the diagnostics dashboard on a random port instead.
  warnings.warn("\n" + msg)
```

[3]:
```
<Client: 'inproc://192.168.0.106/21124/1' processes=1 threads=12, memory=6.00
GB>
```

[3]:
```
EMBEDDING_FILES = [
    'crawl-300d-2M.gensim',
    'glove.840B.300d.gensim'
]
NUM_MODELS = 2
BATCH_SIZE = 60
LSTM_UNITS = 128
DENSE_HIDDEN_UNITS = 4 * LSTM_UNITS
EPOCHS = 4
```

```
MAX_LEN = 220
IDENTITY_COLUMNS = [
    'male', 'female', 'homosexual_gay_or_lesbian', 'christian', 'jewish',
    'muslim', 'black', 'white', 'psychiatric_or_mental_illness'
]
AUX_COLUMNS = ['target', 'severe_toxicity', 'obscene', 'identity_attack',␣
↪'insult', 'threat']
TEXT_COLUMN = 'comment_text'
TARGET_COLUMN = 'target'
CHARS_TO_REMOVE = '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n""'\'ꝏ÷•à– ³ ‘ ´°£€\× √²–
'
```

[4]:
```
def build_matrix(word_index, path):
    embedding_index = KeyedVectors.load(path, mmap='r')
    embedding_matrix = np.zeros((len(word_index) + 1, 300))
    for word, i in tqdm(word_index.items()):
        for candidate in [word, word.lower()]:
            if candidate in embedding_index:
                embedding_matrix[i] = embedding_index[candidate]
                break
    return embedding_matrix
```

### 6.1 Reading data

[6]:
```
train_data = pd.read_csv('../../train/train.csv')
```

[7]:
```
for column in IDENTITY_COLUMNS + [TARGET_COLUMN]:
    train_data[column] = np.where(train_data[column] >= 0.5, 1, 0)
```

### 6.2 Train test split (80% - 20%)

using stratified sampling to avoid bias while splitting data

[8]:
```
train_data, cv_df = train_test_split(train_data, test_size=0.2,␣
↪stratify=train_data.target.values)
print(train_data.shape)
print(cv_df.shape)
```

```
(1443899, 45)
(360975, 45)
```

Checking if test data is having approx same proportion of toxic comments compared to train data

[25]:
```
neg_train = train_data[train_data['target'] == 1]
neg_train.shape
```

[25]: (115467, 45)

```
[26]: neg_validation = cv_df[cv_df['target'] == 1]
      neg_validation.shape
```

[26]: (28867, 46)

```
[9]: x_validation = cv_df[TEXT_COLUMN].astype(str)
     y_validation = cv_df[TARGET_COLUMN].values
     x_train = train_data[TEXT_COLUMN].astype(str)
     y_train =train_data[TARGET_COLUMN].values
     x_test = test_df[TEXT_COLUMN].astype(str)
```

### 6.3 Data preparation

```
[11]: y_train = train_data[TARGET_COLUMN]
      y_train = to_categorical(y_train)

      y_validation = cv_df[TARGET_COLUMN]
      y_validation = to_categorical(y_validation)
```

```
[12]: sample_weights = np.ones(len(x_train), dtype=np.float32)
      sample_weights += train_data[IDENTITY_COLUMNS].sum(axis=1)
      sample_weights += train_data[TARGET_COLUMN] * (~train_data[IDENTITY_COLUMNS]).
       ↪sum(axis=1)
      sample_weights += (~train_data[TARGET_COLUMN]) * train_data[IDENTITY_COLUMNS].
       ↪sum(axis=1) * 5
      sample_weights /= sample_weights.mean()
```

```
[13]: tokenizer = text.Tokenizer(filters=CHARS_TO_REMOVE, lower=False)
      tokenizer.fit_on_texts(list(x_train) + list(x_test) + list(x_validation))

      x_train = tokenizer.texts_to_sequences(x_train)
      x_test = tokenizer.texts_to_sequences(x_test)
      x_validation = tokenizer.texts_to_sequences(x_validation)

      x_train = sequence.pad_sequences(x_train, maxlen=MAX_LEN)
      x_test = sequence.pad_sequences(x_test, maxlen=MAX_LEN)
      x_validation = sequence.pad_sequences(x_validation, maxlen=MAX_LEN)
```

```
[14]: embedding_matrix = (build_matrix(tokenizer.word_index,␣
       ↪EMBEDDING_FILES[0])+build_matrix(tokenizer.word_index, EMBEDDING_FILES[1]))/2
```

```
100%|      | 424070/424070 [02:31<00:00, 2793.53it/s]
100%|      | 424070/424070 [02:35<00:00, 2723.44it/s]
```

## 6.4 Models

### 6.4.1 CNN Model

[29]:
```
input_text = Input(shape=(MAX_LEN,), dtype='float32')
embedding_layer = Embedding(len(tokenizer.word_index) + 1,
                                    300,
                                    weights=[embedding_matrix],
                                    input_length=MAX_LEN,
                                    trainable=False)
x = embedding_layer(input_text)
x = Conv1D(128, 2, activation='relu', padding='same')(x)
x = MaxPooling1D(5, padding='same')(x)
x = Conv1D(128, 3, activation='relu', padding='same')(x)
x = MaxPooling1D(5, padding='same')(x)
x = Conv1D(128, 4, activation='relu', padding='same')(x)
x = MaxPooling1D(40, padding='same')(x)
x = Flatten()(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='relu')(x)
output = Dense(2, activation='softmax')(x)
```

```
WARNING:tensorflow:From /home/user/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/resource_variable_ops.py:435: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
```

[30]:
```
import keras
model = Model(inputs=[input_text], outputs=[output])
model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=[keras.metrics.AUC()])
print(model.summary())
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 220)               0
_________________________________________________________________
embedding_1 (Embedding)      (None, 220, 300)          127221300
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 220, 128)          76928
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 44, 128)           0
_________________________________________________________________
```

```
conv1d_2 (Conv1D)            (None, 44, 128)          49280
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 9, 128)           0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 9, 128)           65664
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 1, 128)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 128)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)              16512
_________________________________________________________________
dense_2 (Dense)              (None, 2)                258
=================================================================
Total params: 127,429,942
Trainable params: 208,642
Non-trainable params: 127,221,300
_________________________________________________________________
None
```

[31]:
```
CNN_model = model.fit(
            x_train,
            y_train,
            batch_size=BATCH_SIZE,
            epochs=5
        )
```

```
WARNING:tensorflow:From /home/user/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/5
1443899/1443899 [==============================] - 197s 137us/step - loss:
0.1382 - auc_1: 0.9879
Epoch 2/5
1443899/1443899 [==============================] - 196s 136us/step - loss:
0.1266 - auc_1: 0.9899
Epoch 3/5
1443899/1443899 [==============================] - 196s 136us/step - loss:
0.1228 - auc_1: 0.9905
Epoch 4/5
1443899/1443899 [==============================] - 197s 136us/step - loss:
0.1204 - auc_1: 0.9909
Epoch 5/5
```

```
1443899/1443899 [==============================] - 198s 137us/step - loss:
0.1185 - auc_1: 0.9912
```

[32]:
```
MODEL_NAME = 'cnn_model'
cv_df[MODEL_NAME] = model.predict(x_validation)[:, 1]
```

[42]:
```
bias_metrics_df = compute_bias_metrics_for_model(cv_df, identity_columns,␣
↪MODEL_NAME, TOXICITY_COLUMN)
```

[36]:
```
get_final_metric(bias_metrics_df, calculate_overall_auc(cv_df, MODEL_NAME))
```

[36]:
```
0.9098346247362036
```

[41]:
```
del model
```

### 6.4.2 Single layered LSTM

[42]:
```
import gc
gc.collect()
```

[42]:
```
1477
```

[43]:
```
from keras.regularizers import l2
input_text = Input(shape=(MAX_LEN,), dtype='float32')
embedding_layer = Embedding(len(tokenizer.word_index) + 1,
                            300,
                            weights=[embedding_matrix],
                            input_length=MAX_LEN,
                            trainable=False)
x = embedding_layer(input_text)
x = LSTM(LSTM_UNITS, return_sequences=True, kernel_regularizer=l2(0.001),␣
↪dropout=0.5)(x)
x = Flatten()(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='relu')(x)
output = Dense(2, activation='softmax')(x)
```

[44]:
```
model = Model(inputs=[input_text], outputs=[output])
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=[keras.metrics.AUC()])
print(model.summary())
```

```
Model: "model_4"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
input_3 (InputLayer)         (None, 220)              0
_________________________________________________________________
embedding_3 (Embedding)      (None, 220, 300)         127221300
_________________________________________________________________
lstm_2 (LSTM)                (None, 220, 128)         219648
_________________________________________________________________
flatten_3 (Flatten)          (None, 28160)            0
_________________________________________________________________
dropout_3 (Dropout)          (None, 28160)            0
_________________________________________________________________
dense_5 (Dense)              (None, 128)              3604608
_________________________________________________________________
dense_6 (Dense)              (None, 2)                258
=================================================================
Total params: 131,045,814
Trainable params: 3,824,514
Non-trainable params: 127,221,300
_________________________________________________________________
None
```

[45]:
```
LSTM_1_layer_model = model.fit(
            x_train,
            y_train,
            batch_size=BATCH_SIZE,
            epochs=1
        )
```

```
Epoch 1/1
1443899/1443899 [==============================] - 2283s 2ms/step - loss: 0.1994
- auc_3: 0.9793
```

[47]:
```
MODEL_NAME = 'LSTM_1_layer_model'
cv_df[MODEL_NAME] = LSTM_1_layer_model.model.predict(x_validation)[:, 1]
```

[41]:
```
bias_metrics_df = compute_bias_metrics_for_model(cv_df, identity_columns,␣
 ↪MODEL_NAME, TOXICITY_COLUMN)
```

[49]:
```
get_final_metric(bias_metrics_df, calculate_overall_auc(cv_df, MODEL_NAME))
```

[49]: 0.8869887233187211

[50]:
```
del model
gc.collect()
```

[50]: 45

### 6.4.3 Two layered Bi-Directional LSTM

[52]:
```
from keras.regularizers import l2
input_text = Input(shape=(MAX_LEN,), dtype='float32')
embedding_layer = Embedding(len(tokenizer.word_index) + 1,
                                    300,
                                    weights=[embedding_matrix],
                                    input_length=MAX_LEN,
                                    trainable=False)
x = embedding_layer(input_text)
x = Bidirectional(LSTM(LSTM_UNITS, return_sequences=True,␣
 ↪kernel_regularizer=l2(0.001), dropout=0.5))(x)
x = Bidirectional(LSTM(LSTM_UNITS, return_sequences=True,␣
 ↪kernel_regularizer=l2(0.001), dropout=0.5))(x)
x = GlobalMaxPooling1D()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='relu')(x)
output = Dense(2, activation='softmax')(x)
```

[53]:
```
model = Model(inputs=[input_text], outputs=[output])
model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=[keras.metrics.AUC()])
print(model.summary())
```

```
Model: "model_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         (None, 220)               0
_________________________________________________________________
embedding_5 (Embedding)      (None, 220, 300)          127221300
_________________________________________________________________
bidirectional_3 (Bidirection (None, 220, 256)          439296
_________________________________________________________________
bidirectional_4 (Bidirection (None, 220, 256)          394240
_________________________________________________________________
global_max_pooling1d_2 (Glob (None, 256)               0
_________________________________________________________________
dense_9 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_11 (Dense)             (None, 2)                 258
```

```
=================================================================
Total params: 128,104,502
Trainable params: 883,202
Non-trainable params: 127,221,300
_________________________________________________________________
None
```

[55]:
```
bi_dir_LSTM_2_layer_model = model.fit(
            x_train,
            y_train,
            batch_size=BATCH_SIZE,
            epochs=1
        )
```

```
Epoch 1/1
1443899/1443899 [==============================] - 8470s 6ms/step - loss: 0.1900
- auc_4: 0.9820
```

[56]:
```
MODEL_NAME = 'bi_dir_LSTM_2_layer_model'
cv_df[MODEL_NAME] = model.predict(x_validation)[:, 1]
```

[57]:
```
bias_metrics_df = compute_bias_metrics_for_model(cv_df, identity_columns,␣
 ↪MODEL_NAME, TOXICITY_COLUMN)
bias_metrics_df
```

[57]:
```
                        subgroup  subgroup_size  …  bpsn_auc  bnsp_auc
7                          white           5016  …  0.809932  0.945186
2      homosexual_gay_or_lesbian           2184  …  0.792816  0.948477
5                         muslim           4205  …  0.881415  0.907155
6                          black           3054  …  0.815275  0.943441
8  psychiatric_or_mental_illness           1002  …  0.916938  0.879761
4                         jewish           1583  …  0.884846  0.913397
0                           male           9049  …  0.871653  0.942291
1                         female          10791  …  0.889218  0.935191
3                      christian           8189  …  0.927190  0.906267

[9 rows x 5 columns]
```

[58]:
```
get_final_metric(bias_metrics_df, calculate_overall_auc(cv_df, MODEL_NAME))
```

[58]: 0.8877852468005003

[81]:
```
del model
gc.collect()
```

[81]: 17170

### 6.4.4 Research paper approach

https://www.theseus.fi/bitstream/handle/10024/226938/Quan_Do.pdf

[32]:
```
input_text = Input(shape=(MAX_LEN,), dtype='float32')
embedding_layer = Embedding(len(tokenizer.word_index) + 1,
                            300,
                            weights=[embedding_matrix],
                            input_length=MAX_LEN,
                            trainable=False)
x = embedding_layer(input_text)
x = SpatialDropout1D(0.2)(x)
x = Bidirectional(LSTM(LSTM_UNITS, return_sequences=True))(x)
x = Bidirectional(LSTM(LSTM_UNITS, return_sequences=True))(x)

hidden = concatenate([
        GlobalMaxPooling1D()(x),
        GlobalAveragePooling1D()(x),
    ])
hidden = add([hidden, Dense(DENSE_HIDDEN_UNITS, activation='relu')(hidden)])
hidden = add([hidden, Dense(DENSE_HIDDEN_UNITS, activation='relu')(hidden)])
result = Dense(2, activation='sigmoid')(hidden)
```

```
WARNING:tensorflow:From /home/user/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/resource_variable_ops.py:435: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
```

[35]:
```
model = Model(inputs=input_text, outputs=[result])
model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=[keras.metrics.AUC()])
print(model.summary())
```

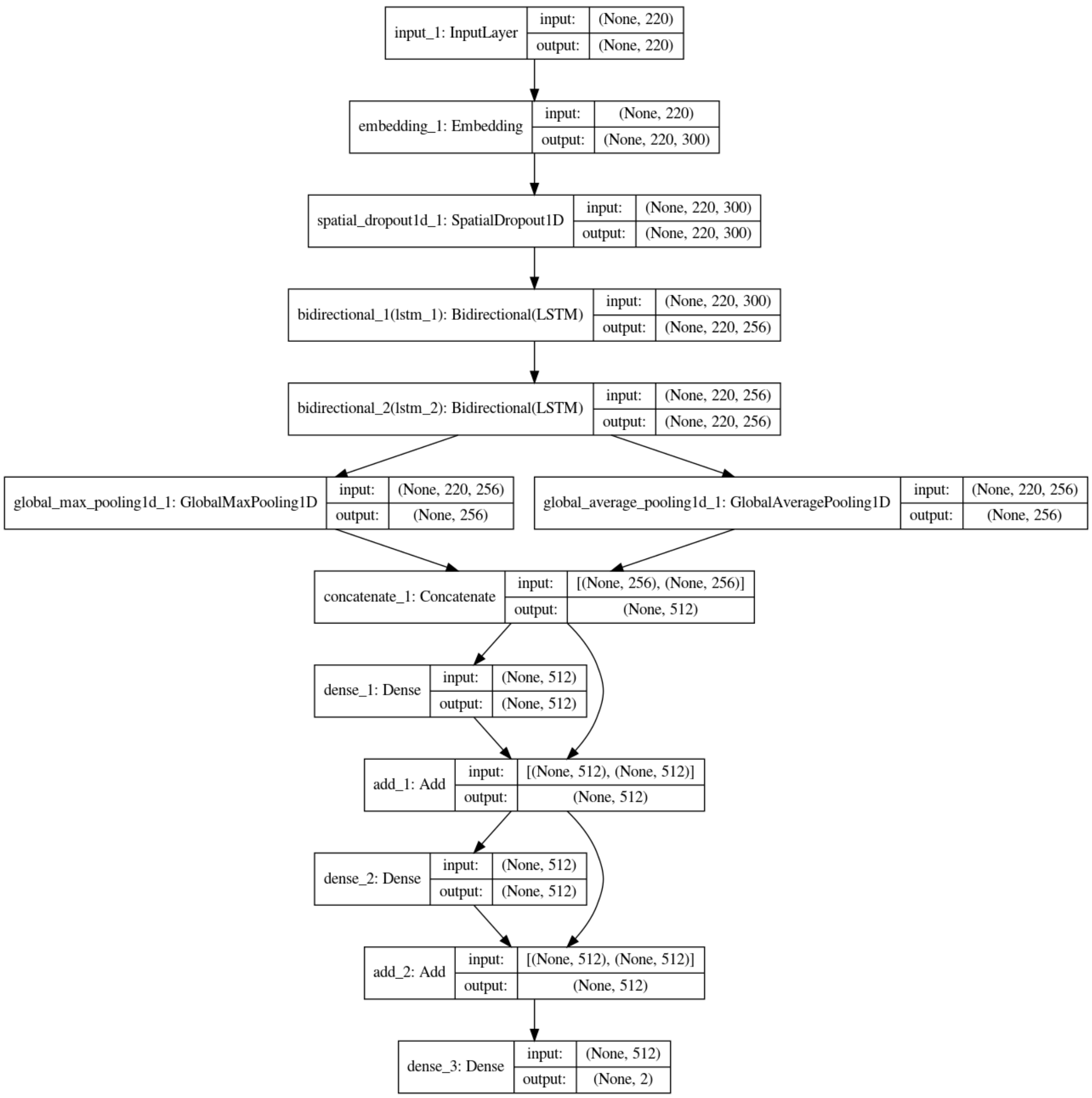
```
Model: "model_2"
________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
==================
input_1 (InputLayer)            (None, 220)          0
________________________________________________________________________________
__________________
embedding_1 (Embedding)         (None, 220, 300)     127221300   input_1[0][0]
________________________________________________________________________________
__________________
```

```
spatial_dropout1d_1 (SpatialDro (None, 220, 300)     0
embedding_1[0][0]

________________________________________________________________________________
__________________
bidirectional_1 (Bidirectional) (None, 220, 256)     439296
spatial_dropout1d_1[0][0]

________________________________________________________________________________
__________________
bidirectional_2 (Bidirectional) (None, 220, 256)     394240
bidirectional_1[0][0]

________________________________________________________________________________
__________________
global_max_pooling1d_1 (GlobalM (None, 256)          0
bidirectional_2[0][0]

________________________________________________________________________________
__________________
global_average_pooling1d_1 (Glo (None, 256)          0
bidirectional_2[0][0]

________________________________________________________________________________
__________________
concatenate_1 (Concatenate)     (None, 512)          0
global_max_pooling1d_1[0][0]
global_average_pooling1d_1[0][0]

________________________________________________________________________________
__________________
dense_1 (Dense)                 (None, 512)          262656
concatenate_1[0][0]

________________________________________________________________________________
__________________
add_1 (Add)                     (None, 512)          0
concatenate_1[0][0]
                                                                dense_1[0][0]

________________________________________________________________________________
__________________
dense_2 (Dense)                 (None, 512)          262656     add_1[0][0]

________________________________________________________________________________
__________________
add_2 (Add)                     (None, 512)          0          add_1[0][0]
                                                                dense_2[0][0]

________________________________________________________________________________
__________________
dense_3 (Dense)                 (None, 2)            1026       add_2[0][0]
================================================================================
==================
Total params: 128,581,174
Trainable params: 1,359,874
Non-trainable params: 127,221,300

________________________________________________________________________________
```

```
-------------------
None
```

[36]:
```
plot_model(model, show_shapes=True, to_file='research_paper_model.png')
```

[36]:




[37]:
```
research_paper_model = model.fit(
            x_train,
            y_train,
            batch_size=BATCH_SIZE,
            epochs=1)
```

```
WARNING:tensorflow:From /home/user/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
```

```
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
WARNING:tensorflow:From /home/user/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/math_grad.py:102: div (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.
Epoch 1/1
1443899/1443899 [==============================] - 8107s 6ms/step - loss: 0.1328
- auc_1: 0.9863
```

[38]:
```
MODEL_NAME = 'research_paper_model'
cv_df[MODEL_NAME] = model.predict(x_validation)[:, 1]
```

[39]:
```
bias_metrics_df = compute_bias_metrics_for_model(cv_df, identity_columns,
 ↪MODEL_NAME, TOXICITY_COLUMN)
bias_metrics_df
```

[39]:
```
                       subgroup  subgroup_size  subgroup_auc  bpsn_auc  \
6                         black           2956      0.839738  0.829836
2      homosexual_gay_or_lesbian           2148      0.844528  0.831435
5                        muslim           4133      0.857721  0.866696
7                         white           5001      0.858279  0.833863
4                        jewish           1543      0.894097  0.899702
8  psychiatric_or_mental_illness            990      0.916345  0.900627
1                        female          10652      0.925286  0.922746
0                          male           8998      0.926840  0.918693
3                     christian           8029      0.931650  0.949586

   bnsp_auc
6  0.971977
2  0.971242
5  0.965435
7  0.974571
4  0.964022
8  0.969396
1  0.966645
0  0.968962
3  0.951175
```

[40]:
```
get_final_metric(bias_metrics_df, calculate_overall_auc(cv_df, MODEL_NAME))
```

[40]:
```
0.9228624083847328
```

[49]:
```python
del model
gc.collect()
```

[49]: 1757

### 6.4.5 Research paper with attention layer

[18]:
```python
# https://www.kaggle.com/takuok/bidirectional-lstm-and-attention-lb-0-043
from keras.layers import Layer
from keras import initializers, regularizers, constraints
class Attention(Layer):
    def __init__(self, step_dim,
                 W_regularizer=None, b_regularizer=None,
                 W_constraint=None, b_constraint=None,
                 bias=True, **kwargs):
        self.supports_masking = True
        self.init = initializers.get('glorot_uniform')

        self.W_regularizer = regularizers.get(W_regularizer)
        self.b_regularizer = regularizers.get(b_regularizer)

        self.W_constraint = constraints.get(W_constraint)
        self.b_constraint = constraints.get(b_constraint)

        self.bias = bias
        self.step_dim = step_dim
        self.features_dim = 0
        super(Attention, self).__init__(**kwargs)

    def build(self, input_shape):
        assert len(input_shape) == 3

        self.W = self.add_weight(shape=(input_shape[-1],),
                                 initializer=self.init,
                                 name=f'{self.name}_W',
                                 regularizer=self.W_regularizer,
                                 constraint=self.W_constraint)
        self.features_dim = input_shape[-1]

        if self.bias:
            self.b = self.add_weight(shape=(input_shape[1],),
                                     initializer='zero',
                                     name='{}_b'.format(self.name),
                                     regularizer=self.b_regularizer,
                                     constraint=self.b_constraint)
        else:
```

```
            self.b = None

        self.built = True

    def compute_mask(self, input, input_mask=None):
        return None

    def call(self, x, mask=None):
        features_dim = self.features_dim
        step_dim = self.step_dim

        eij = K.reshape(K.dot(K.reshape(x, (-1, features_dim)),
                        K.reshape(self.W, (features_dim, 1))), (-1, step_dim))

        if self.bias:
            eij += self.b

        eij = K.tanh(eij)

        a = K.exp(eij)

        if mask is not None:
            a *= K.cast(mask, K.floatx())

        a /= K.cast(K.sum(a, axis=1, keepdims=True) + K.epsilon(), K.floatx())

        a = K.expand_dims(a)
        weighted_input = x * a
        return K.sum(weighted_input, axis=1)

    def compute_output_shape(self, input_shape):
        return input_shape[0],  self.features_dim
```

[19]:
```
input_text = Input(shape=(MAX_LEN,), dtype='float32')
embedding_layer = Embedding(len(tokenizer.word_index) + 1,
                            300,
                            weights=[embedding_matrix],
                            input_length=MAX_LEN,
                            trainable=False)
x = embedding_layer(input_text)
x = SpatialDropout1D(0.2)(x)
x = Bidirectional(LSTM(LSTM_UNITS, return_sequences=True))(x)
x = Bidirectional(LSTM(LSTM_UNITS, return_sequences=True))(x)
att = Attention(MAX_LEN)(x)
x = Conv1D(64, kernel_size = 3, padding = "valid", kernel_initializer =␣
↪"he_uniform")(x)
hidden = concatenate([att,
```

```
        GlobalMaxPooling1D()(x),
        GlobalAveragePooling1D()(x),
    ])
hidden = add([hidden, Dense(384, activation='relu')(hidden)])
hidden = Dropout(0.5)(hidden)
hidden = add([hidden, Dense(384, activation='relu')(hidden)])
result = Dense(2, activation='sigmoid')(hidden)
#     aux_result = Dense(num_aux_targets, activation='sigmoid')(hidden)
```

```
WARNING:tensorflow:From /home/user/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/resource_variable_ops.py:435: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
```

[20]:

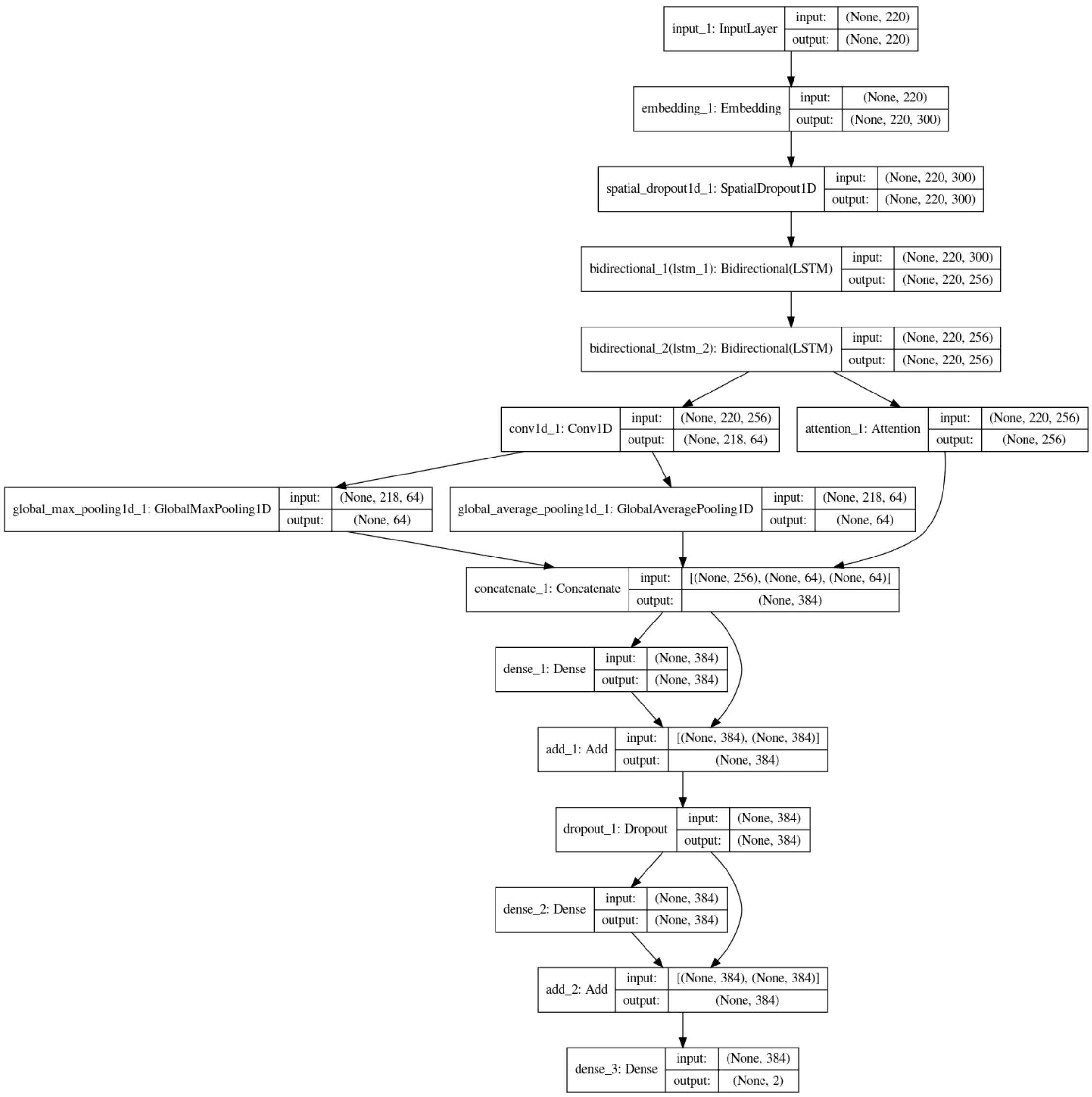
```
model = Model(inputs=input_text, outputs=[result])
model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=[keras.metrics.AUC()])
print(model.summary())
```

```
Model: "model_1"
________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
==================
input_1 (InputLayer)            (None, 220)          0

________________________________________________________________________________
__________________
embedding_1 (Embedding)         (None, 220, 300)     127221300   input_1[0][0]

________________________________________________________________________________
__________________
spatial_dropout1d_1 (SpatialDro (None, 220, 300)     0
embedding_1[0][0]

________________________________________________________________________________
__________________
bidirectional_1 (Bidirectional) (None, 220, 256)     439296
spatial_dropout1d_1[0][0]

________________________________________________________________________________
__________________
bidirectional_2 (Bidirectional) (None, 220, 256)     394240
bidirectional_1[0][0]

________________________________________________________________________________
__________________
conv1d_1 (Conv1D)               (None, 218, 64)      49216
bidirectional_2[0][0]
```

```
____________________________________________________________________________________________________
attention_1 (Attention)         (None, 256)          476
bidirectional_2[0][0]
____________________________________________________________________________________________________
global_max_pooling1d_1 (GlobalM (None, 64)           0            conv1d_1[0][0]
____________________________________________________________________________________________________
global_average_pooling1d_1 (Glo (None, 64)           0            conv1d_1[0][0]
____________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 384)          0
attention_1[0][0]
global_max_pooling1d_1[0][0]
global_average_pooling1d_1[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                 (None, 384)          147840
concatenate_1[0][0]
____________________________________________________________________________________________________
add_1 (Add)                     (None, 384)          0
concatenate_1[0][0]
                                                                  dense_1[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 384)          0            add_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                 (None, 384)          147840       dropout_1[0][0]
____________________________________________________________________________________________________
add_2 (Add)                     (None, 384)          0            dropout_1[0][0]
                                                                  dense_2[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                 (None, 2)            770          add_2[0][0]
====================================================================================================
Total params: 128,400,978
Trainable params: 1,179,678
Non-trainable params: 127,221,300
____________________________________________________________________________________________________
None
```

[21]:
```
plot_model(model, show_shapes=True,␣
↪to_file='research_paper_with_attention_model.png')
```

[21]:




[22]:
```
research_paper_model = model.fit(
            x_train,
            y_train,
            batch_size=BATCH_SIZE,
            epochs=4)
```

```
WARNING:tensorflow:From /home/user/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
```

```
Instructions for updating:
Use tf.cast instead.
WARNING:tensorflow:From /home/user/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/math_grad.py:102: div (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.
Epoch 1/4
1443899/1443899 [==============================] - 7886s 5ms/step - loss: 0.1370
- auc_1: 0.9824
Epoch 2/4
1443899/1443899 [==============================] - 7893s 5ms/step - loss: 0.1204
- auc_1: 0.9854
Epoch 3/4
1443899/1443899 [==============================] - 7890s 5ms/step - loss: 0.1148
- auc_1: 0.9858
Epoch 4/4
1443899/1443899 [==============================] - 7889s 5ms/step - loss: 0.1107
- auc_1: 0.9860
```

[23]:
```
MODEL_NAME = 'research_paper_with_attention'
cv_df[MODEL_NAME] = model.predict(x_validation)[:, 1]
```

[24]:
```
bias_metrics_df = compute_bias_metrics_for_model(cv_df, identity_columns,
 ↪MODEL_NAME, TOXICITY_COLUMN)
bias_metrics_df
```

[24]:
```
                     subgroup  subgroup_size  subgroup_auc  bpsn_auc  \
5                      muslim           4187      0.850103  0.879523
6                       black           3017      0.850858  0.827608
2       homosexual_gay_or_lesbian       2227      0.851779  0.850920
7                       white           4932      0.863925  0.849635
4                      jewish           1540      0.888502  0.920189
8  psychiatric_or_mental_illness         989      0.915460  0.921929
3                   christian           7955      0.925814  0.952348
1                      female          10754      0.931927  0.934992
0                        male           8883      0.933275  0.929630

   bnsp_auc
5  0.962804
6  0.976094
2  0.970679
7  0.974505
4  0.955366
8  0.964872
3  0.949032
```

```
1   0.966422
0   0.969869
```

[25]:
```
get_final_metric(bias_metrics_df, calculate_overall_auc(cv_df, MODEL_NAME))
```

[25]: 0.9269571528954396

### 6.5 Using Transfer Learning (BERT)

[64]:
```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
import sys
package_dir = "ppbert/pytorch-pretrained-bert/pytorch-pretrained-BERT"
sys.path.append(package_dir)
import torch.utils.data
import numpy as np
import pandas as pd
from tqdm import tqdm
import os
import warnings
from pytorch_pretrained_bert import BertTokenizer,␣
 ↪BertForSequenceClassification, BertAdam
from pytorch_pretrained_bert import BertConfig
import gc
from sklearn import metrics
from sklearn.model_selection import train_test_split

warnings.filterwarnings(action='once')
device = torch.device('cuda')
```

[2]:
```
IDENTITY_COLUMNS = [
    'transgender', 'female', 'homosexual_gay_or_lesbian', 'muslim', 'hindu',
    'white', 'black', 'psychiatric_or_mental_illness', 'jewish'
    ]
TARGET_COLUMN = 'target'
```

[4]:
```
for column in IDENTITY_COLUMNS + [TARGET_COLUMN]:
    train_df[column] = np.where(train_df[column] >=0.5, True, False)
```

[43]:
```
# cv_df.to_csv('cv_df.csv')
# train_data.to_csv('train_data.csv')
cv_df = pd.read_csv('cv_df.csv')
train_data = pd.read_csv('train_data.csv')
```

### 6.5.1 Bert Small And Large with fine tuned models

**Data Preparation**

[7]:
```
def convert_lines(example, max_seq_length,tokenizer):
    max_seq_length -=2
    all_tokens = []
    longer = 0
    for text in tqdm(example):
        tokens_a = tokenizer.tokenize(text)
        if len(tokens_a)>max_seq_length:
            tokens_a = tokens_a[:max_seq_length]
            longer += 1
        one_token = tokenizer.
 ↪convert_tokens_to_ids(["[CLS]"]+tokens_a+["[SEP]"])+[0] * (max_seq_length -␣
 ↪len(tokens_a))
        all_tokens.append(one_token)
    return np.array(all_tokens)
```

[8]:
```
MAX_SEQUENCE_LENGTH = 220
SEED = 1234
BATCH_SIZE = 32
BERT_MODEL_PATH = 'bert-pretrained-models/uncased_l-12_h-768_a-12/
 ↪uncased_L-12_H-768_A-12/'
LARGE_BERT_MODEL_PATH = 'bert-pretrained-models/uncased_l-24_h-1024_a-16/
 ↪uncased_L-24_H-1024_A-16/'
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
```

[9]:
```
# Pretrained BERT models - Google's pretrained BERT model
BERT_SMALL_PATH = 'bert-pretrained-models/uncased_l-12_h-768_a-12/
 ↪uncased_L-12_H-768_A-12/'
BERT_LARGE_PATH = 'bert-pretrained-models/uncased_l-24_h-1024_a-16/
 ↪uncased_L-24_H-1024_A-16/'
```

[10]:
```
# JIGSAW fine-tuned BERT models
JIGSAW_BERT_SMALL_MODEL_PATH =␣
 ↪'finetuned-bert-for-jigsaw-toxicity-classification/bert_pytorch.bin'
JIGSAW_BERT_LARGE_MODEL_PATH = 'pretrained-b-j/
 ↪jigsaw-bert-large-uncased-len-220-fp16/epoch-1/pytorch_model.bin'
JIGSAW_BERT_SMALL_JSON_PATH =␣
 ↪'finetuned-bert-for-jigsaw-toxicity-classification/bert_config.json'
JIGSAW_BERT_LARGE_JSON_PATH = 'pretrained-b-j/
 ↪jigsaw-bert-large-uncased-len-220-fp16/epoch-1/config.json'
NUM_BERT_MODELS = 2
```

```
INFER_BATCH_SIZE = 64
```

[11]:
```
cv_preds = np.zeros((cv_df.shape[0],NUM_BERT_MODELS))
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
```

### Predicting BERT large model

[12]:
```
# Prepare data
bert_config = BertConfig(JIGSAW_BERT_LARGE_JSON_PATH)
tokenizer = BertTokenizer.from_pretrained(BERT_LARGE_PATH,␣
↪cache_dir=None,do_lower_case=True)
X_cv = convert_lines(cv_df["comment_text"].fillna("DUMMY_VALUE"),␣
↪MAX_SEQUENCE_LENGTH, tokenizer)
cv = torch.utils.data.TensorDataset(torch.tensor(X_cv, dtype=torch.long))
```

```
100%|      | 360975/360975 [03:47<00:00, 1589.46it/s]
```

[44]:
```
# Load fine-tuned BERT model
gc.collect()
model = BertForSequenceClassification(bert_config, num_labels=1)
model.load_state_dict(torch.load(JIGSAW_BERT_LARGE_MODEL_PATH))
model.to(device)
for param in model.parameters():
    param.requires_grad = False
model.eval()
```

[14]:
```
# Predicting
gc.collect()
model_preds = np.zeros((len(X_cv)))
cv_loader = torch.utils.data.DataLoader(cv, batch_size=INFER_BATCH_SIZE,␣
↪shuffle=False)
tk0 = tqdm(cv_loader)
for i, (x_batch,) in enumerate(tk0):
        pred = model(x_batch.to(device), attention_mask=(x_batch > 0).
↪to(device), labels=None)
        model_preds[i * INFER_BATCH_SIZE:(i + 1) * INFER_BATCH_SIZE] = pred[:,␣
↪0].detach().cpu().squeeze().numpy()

cv_preds[:,0] = torch.sigmoid(torch.tensor(model_preds)).numpy().ravel()
del model
gc.collect()
```

```
100%|      | 5641/5641 [5:23:34<00:00,  3.44s/it]
```

[14]: 0

**Predicting BERT small model**

[15]:
```
bert_config = BertConfig(JIGSAW_BERT_SMALL_JSON_PATH)
tokenizer = BertTokenizer.from_pretrained(BERT_SMALL_PATH,␣
↪cache_dir=None,do_lower_case=True)
X_cv = convert_lines(cv_df["comment_text"].fillna("DUMMY_VALUE"),␣
↪MAX_SEQUENCE_LENGTH, tokenizer)
cv = torch.utils.data.TensorDataset(torch.tensor(X_cv, dtype=torch.long))
```

```
100%|      | 360975/360975 [03:47<00:00, 1584.12it/s]
```

[45]:
```
# # # Load fine-tuned BERT model
model = BertForSequenceClassification(bert_config, num_labels=1)
model.load_state_dict(torch.load(JIGSAW_BERT_SMALL_MODEL_PATH))
model.to(device)
for param in model.parameters():
    param.requires_grad = False
model.eval()
```

[17]:
```
# Predicting
model_preds = np.zeros((len(X_cv)))
cv_loader = torch.utils.data.DataLoader(cv, batch_size=INFER_BATCH_SIZE,␣
↪shuffle=False)
tk0 = tqdm(cv_loader)
for i, (x_batch,) in enumerate(tk0):
        pred = model(x_batch.to(device), attention_mask=(x_batch > 0).
↪to(device), labels=None)
        model_preds[i * INFER_BATCH_SIZE:(i + 1) * INFER_BATCH_SIZE] = pred[:,␣
↪0].detach().cpu().squeeze().numpy()

cv_preds[:,1] = torch.sigmoid(torch.tensor(model_preds)).numpy().ravel()

del model
gc.collect()
```

```
100%|      | 5641/5641 [1:45:48<00:00,  1.13s/it]
```

[17]: 0

[18]:
```
# Sub-model prediction
bert_submission = pd.DataFrame.from_dict({
'id': cv_df['id'],
'prediction': cv_preds.mean(axis=1)})
bert_submission.to_csv('bert_submission.csv')
```

[16]:
```
bert_submission = pd.read_csv('bert_submission.csv')
bert_submission.head()
```

[16]:

| | id | prediction |
|---|---|---|
| 0 | 6182394 | 0.174450 |
| 1 | 5328597 | 0.000077 |
| 2 | 4980998 | 0.051977 |
| 3 | 5520712 | 0.000070 |
| 4 | 5214775 | 0.000070 |

### 6.5.2 Research paper implementation

[39]:
```
from keras.preprocessing import text, sequence
from keras import backend as K
from keras.models import Model
from keras.layers import Input, Dense, Embedding, SpatialDropout1D, add,␣
 ↪concatenate
from keras.layers import CuDNNLSTM, Bidirectional, GlobalMaxPooling1D,␣
 ↪GlobalAveragePooling1D, LSTM, Conv1D
from keras.preprocessing import text, sequence
from keras.callbacks import LearningRateScheduler
from keras.engine.topology import Layer
from keras import initializers, regularizers, constraints, optimizers, layers
from tqdm._tqdm_notebook import tqdm_notebook as tqdm
import pickle
tqdm.pandas()
import gc
```

[4]:
```
EMBEDDING_PATHS = [
    '../convolutional_model/crawl-300d-2M.gensim',
    '../convolutional_model/glove.840B.300d.gensim'
]



NUM_MODELS = 2 # The number of classifiers we want to train
BATCH_SIZE = 512 # can be tuned
LSTM_UNITS = 128 # can be tuned
DENSE_HIDDEN_UNITS = 4*LSTM_UNITS # can betuned
EPOCHS = 4 # The number of epoches we want to train for each classifier
MAX_LEN = 220 # can ben tuned



IDENTITY_COLUMNS = [
    'transgender', 'female', 'homosexual_gay_or_lesbian', 'muslim', 'hindu',
    'white', 'black', 'psychiatric_or_mental_illness', 'jewish'
    ]
```

```python
AUX_COLUMNS = ['target',␣
 ↪'severe_toxicity','obscene','identity_attack','insult','threat']
TEXT_COLUMN = 'comment_text'
TARGET_COLUMN = 'target'
```

**Embedding**

[5]:
```python
def get_coefs(word, *arr):
    """
    Get word, word_embedding from a pretrained embedding file
    """
    return word, np.asarray(arr,dtype='float32')

def load_embeddings(path):
    if path.split('.')[-1] in ['txt','vec']: # for original pretrained␣
 ↪embedding files (extension .text, .vec)
        with open(path,'rb') as f:
            return dict(get_coefs(*line.strip().split(' ')) for line in f)
    if path.split('.')[-1] =='pkl': # for pickled pretrained embedding files␣
 ↪(extention pkl). Loading pickeled embeddings is faster than texts
        with open(path,'rb') as f:
            return pickle.load(f)



def build_matrix(word_index, path):
    embedding_index = KeyedVectors.load(path, mmap='r')
    embedding_matrix = np.zeros((len(word_index) + 1, 300))
    for word, i in tqdm(word_index.items()):
        for candidate in [word, word.lower()]:
            if candidate in embedding_index:
                embedding_matrix[i] = embedding_index[candidate]
                break
    return embedding_matrix
```

**Defining model architecture**

[6]:
```python
def build_model(embedding_matrix, num_aux_targets):#, loss_weight):
    """
    embedding layer
    dropout layer
    2 * bidirectional LSTM layers
    2 * pooling layers
    2 dense layers
    1 softmax layer
```

```
    """
    words = Input(shape=(MAX_LEN,))
    x = Embedding(*embedding_matrix.shape, weights = [embedding_matrix],␣
↪trainable=False)(words)
    x = SpatialDropout1D(0.1)(x)
    x = Bidirectional(LSTM(LSTM_UNITS, return_sequences=True))(x)
    x = Bidirectional(LSTM(LSTM_UNITS, return_sequences=True))(x)
    hidden = concatenate([
        GlobalMaxPooling1D()(x),
        GlobalAveragePooling1D()(x)
        ])
    hidden = add([hidden, Dense(DENSE_HIDDEN_UNITS, activation='relu')(hidden)])
    hidden = add([hidden, Dense(DENSE_HIDDEN_UNITS, activation='relu')(hidden)])
    hidden = add([hidden, Dense(DENSE_HIDDEN_UNITS, activation='relu')(hidden)])
    hidden = add([hidden, Dense(DENSE_HIDDEN_UNITS, activation='relu')(hidden)])
    result = Dense(1, activation='sigmoid')(hidden)
    aux_result =Dense(num_aux_targets, activation='sigmoid')(hidden)

    model = Model(inputs =words, outputs =[result, aux_result])
    model.compile(loss='binary_crossentropy', optimizer='adam')

    return model
```

### Text Tokanization

[7]:
```
x_train = train_data[TEXT_COLUMN].astype(str)
y_train = train_data[TARGET_COLUMN].values
y_aux_train = train_data[AUX_COLUMNS].values
x_cv = cv_df[TEXT_COLUMN].astype(str)
```

[8]:
```
# Return a Keras tokenizer class
CHARS_TO_REMOVE = '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n""’\'∞÷•à− ³‘´°£€\×√²−
'
tokenizer = text.Tokenizer(filters = CHARS_TO_REMOVE)
tokenizer.fit_on_texts(list(x_train)+ list(x_cv))
# Turn text to sequences of tokens
x_train = tokenizer.texts_to_sequences(x_train)
x_cv = tokenizer.texts_to_sequences(x_cv)
#Pad sequences to the same length
x_train = sequence.pad_sequences(x_train,maxlen=MAX_LEN)
x_cv= sequence.pad_sequences(x_cv, maxlen=MAX_LEN)
```

[9]:
```
x_train.shape
```

[9]: (1443899, 220)

[10]:
```
# Initialize weights
sample_weights = np.ones(len(x_train), dtype=np.float32)
# Add all the values of the identities along rows
sample_weights += train_data[IDENTITY_COLUMNS].sum(axis=1)
#Add all values of targets*~identity
sample_weights += train_data[TARGET_COLUMN]*(~train_data[IDENTITY_COLUMNS]).
↪sum(axis=1)
#Add all values ~targets*identity
sample_weights += (~train_data[TARGET_COLUMN])*train_data[IDENTITY_COLUMNS].
↪sum(axis=1)
#Normalize them
sample_weights/=sample_weights.mean()
```

[40]:
```
from gensim.models import KeyedVectors
embedding_matrix = np.concatenate([build_matrix(tokenizer.word_index,f) for f␣
↪in EMBEDDING_PATHS], axis =-1)
print("Embedding matrix shape:", embedding_matrix.shape)
del train_data, tokenizer
gc.collect()
```

### Model Training

[12]:
```
checkpoint_predictions = []
weights = []
NUM_MODELS = 1
for model_idx in range(NUM_MODELS):
    model = build_model(embedding_matrix, y_aux_train.shape[-1])
    for global_epoch in range(EPOCHS):
        model.fit(
            x_train,
            [y_train, y_aux_train],
            batch_size=BATCH_SIZE,
            epochs=1,
            sample_weight=[sample_weights.values, np.ones_like(sample_weights)],
            callbacks = [
                LearningRateScheduler(lambda _: 1e-3*(0.55**global_epoch)) #␣
↪Decayed learning rate
                ]
        )
        checkpoint_predictions.append(model.predict(x_cv, batch_size=2048)[0].
↪flatten())
        weights.append(2 ** global_epoch)
    del model
    gc.collect()
```

```
WARNING:tensorflow:From /home/user/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/resource_variable_ops.py:435: colocate_with (from
```

```
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
```

```
/home/user/anaconda3/lib/python3.7/site-
packages/tensorflow/python/framework/tensor_util.py:573: DeprecationWarning:
np.asscalar(a) is deprecated since NumPy v1.16, use a.item() instead
  append_fn(tensor_proto, proto_values)
```

```
WARNING:tensorflow:From /home/user/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
WARNING:tensorflow:From /home/user/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/math_grad.py:102: div (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.
Epoch 1/1
1443899/1443899 [==============================] - 1703s 1ms/step - loss: 0.3804
- dense_5_loss: 0.2900 - dense_6_loss: 0.0904
Epoch 1/1
1443899/1443899 [==============================] - 1712s 1ms/step - loss: 0.3337
- dense_5_loss: 0.2499 - dense_6_loss: 0.0839
Epoch 1/1
1443899/1443899 [==============================] - 1728s 1ms/step - loss: 0.3123
- dense_5_loss: 0.2303 - dense_6_loss: 0.0820
Epoch 1/1
1443899/1443899 [==============================] - 1707s 1ms/step - loss: 0.2943
- dense_5_loss: 0.2135 - dense_6_loss: 0.0807
```

[13]:
```
predictions = np.average(checkpoint_predictions, weights=weights, axis=0)
predictions.shape
```

[13]: (360975,)

[14]:
```
lstm_submission = pd.DataFrame.from_dict({
    'id': cv_df.id,
    'prediction': predictions
})
lstm_submission.to_csv('lstm_submission.csv')
```

[44]:
```
bert_submission = pd.read_csv('bert_submission.csv')
lstm_submission = pd.read_csv('lstm_submission.csv')
```

```
[45]: lstm_submission.head()
```

```
[45]:    Unnamed: 0       id  prediction
      0           0  6005154    0.000086
      1           1   851365    0.093943
      2           2   892430    0.000834
      3           3  5752256    0.997884
      4           4  5590246    0.002142
```

```
[46]: bert_submission.head()
```

```
[46]:    Unnamed: 0       id  prediction
      0     1538593  6005154    0.003758
      1      495446   851365    0.016163
      2      530578   892430    0.000078
      3     1339353  5752256    0.997755
      4     1206486  5590246    0.000212
```

```
[47]: submission = pd.DataFrame.from_dict({
      'id': cv_df['id'],
      'prediction': lstm_submission['prediction'].rank(pct=True)*0.3 +␣
       ↪bert_submission['prediction'].rank(pct=True)*0.7})
      submission.to_csv('submission.csv')
```

### Metric calculation

```
[75]: identity_columns = [
          'male', 'female', 'homosexual_gay_or_lesbian', 'christian', 'jewish',
          'muslim', 'black', 'white', 'psychiatric_or_mental_illness']
      # https://www.kaggle.com/c/jigsaw-unintended-bias-in-toxicity-classification/
       ↪discussion/90986#latest-527331
      SUBGROUP_AUC = 'subgroup_auc'
      BPSN_AUC = 'bpsn_auc'  # stands for background positive, subgroup negative
      BNSP_AUC = 'bnsp_auc'  # stands for background negative, subgroup positive
      TOXICITY_COLUMN = 'target'

      def compute_auc(y_true, y_pred):
          try:
              return metrics.roc_auc_score(y_true, y_pred)
          except ValueError:
              return np.nan

      def compute_subgroup_auc(df, subgroup, label, model_name):
          subgroup_examples = df[df[subgroup] != np.nan]
          return compute_auc(subgroup_examples[label], subgroup_examples[model_name])

      def compute_bpsn_auc(df, subgroup, label, model_name):
```

```
    """Computes the AUC of the within-subgroup negative examples and the␣
↪background positive examples."""
    subgroup_negative_examples = df[(df[subgroup] == True) & (df[label] ==␣
↪False)]
    non_subgroup_positive_examples = df[(df[subgroup] == False) & (df[label] ==␣
↪True)]
    examples = subgroup_negative_examples.append(non_subgroup_positive_examples)
    return compute_auc(examples[label], examples[model_name])

def compute_bnsp_auc(df, subgroup, label, model_name):
    """Computes the AUC of the within-subgroup positive examples and the␣
↪background negative examples."""
    subgroup_positive_examples = df[(df[subgroup] == True) & (df[label] ==␣
↪True)]
    non_subgroup_negative_examples = df[(df[subgroup] == False) & (df[label] ==␣
↪False)]
    examples = subgroup_positive_examples.append(non_subgroup_negative_examples)
    return compute_auc(examples[label], examples[model_name])

def compute_bias_metrics_for_model(dataset,
                                   subgroups,
                                   model,
                                   label_col,
                                   include_asegs=False):
    """Computes per-subgroup metrics for all subgroups and one model."""
    records = []
    for subgroup in subgroups:
        record = {
            'subgroup': subgroup,
            'subgroup_size': len(dataset[dataset[subgroup] != np.nan])
        }
        record[SUBGROUP_AUC] = compute_subgroup_auc(dataset, subgroup,␣
↪label_col, model)
        record[BPSN_AUC] = compute_bpsn_auc(dataset, subgroup, label_col, model)
        record[BNSP_AUC] = compute_bnsp_auc(dataset, subgroup, label_col, model)
        records.append(record)
    return pd.DataFrame(records).sort_values('subgroup_auc', ascending=True)
```

[76]:
```
def calculate_overall_auc(df, model_name):
    true_labels = df[TOXICITY_COLUMN]
    predicted_labels = df[model_name]
    return metrics.roc_auc_score(true_labels, predicted_labels)

def power_mean(series, p):
    total = sum(np.power(series, p))
    return np.power(total / len(series), 1 / p)
```

```
def get_final_metric(bias_df, overall_auc, POWER=-5, OVERALL_MODEL_WEIGHT=0.25):
    bias_score = np.average([
        power_mean(bias_df[SUBGROUP_AUC], POWER),
        power_mean(bias_df[BPSN_AUC], POWER),
        power_mean(bias_df[BNSP_AUC], POWER)
    ])
    return (OVERALL_MODEL_WEIGHT * overall_auc) + ((1 - OVERALL_MODEL_WEIGHT) *␣
↪bias_score)
```

[51]:
```
MODEL_NAME = 'research_paper_with_bert'
cv_df[MODEL_NAME] = submission['prediction'].values
```

[38]:
```
bias_metrics_df = compute_bias_metrics_for_model(cv_df, identity_columns,␣
↪MODEL_NAME, TOXICITY_COLUMN)
```

[78]:
```
get_final_metric(bias_metrics_df, calculate_overall_auc(cv_df, MODEL_NAME))
```

[78]:
```
0.9667060455662488
```

# 7 Result Summary

## 7.1 Machine Learning Simple models

[30]:
```
model_names = ['Naive Bayes', 'Logistic Regression', 'SVM', 'XG-Boost', 'Random␣
↪Forest', 'Stacking']
hyper_params = ['alpha=1','alpha=1e-5', 'apha=1e-5', 'scale_pos_weight=99,␣
↪n_estimators=2000', 'n_estimators=1500, max_depth=12', 'params got from␣
↪others']
train_metric_scores = [87.67, 91.20, 91.45, 91.05, 85.51, 91.68]
test_metric_scores = [86.24, 90.21, 90.26, 88.73, 83.90, 90.57]

results_summary = pd.DataFrame({'model_names':model_names, 'hyper_params':
↪hyper_params, 'train_metric_score':train_metric_scores, 'test_metric_score':
↪test_metric_scores})
results_summary.sort_values(by=['test_metric_score'], ascending=False)
```

[30]:
```
           model_names                             hyper_params  \
5             Stacking                   params got from others
2                  SVM                                apha=1e-5
1  Logistic Regression                               alpha=1e-5
3             XG-Boost  scale_pos_weight=99, n_estimators=2000
0          Naive Bayes                                  alpha=1
4        Random Forest          n_estimators=1500, max_depth=12
```

```
   train_metric_score  test_metric_score
5               91.68              90.57
2               91.45              90.26
1               91.20              90.21
3               91.05              88.73
0               87.67              86.24
4               85.51              83.90
```

### 7.2 Deep Learning Models

[3]:
```
model_names = ['CNN', 'Single layer LSTM', 'Two Layered Bi-Directional LSTM',␣
↪'Research Paper IMPL', 'Research Paper with Attention', 'Research paper +␣
↪BERT small + BERT large']
epochs = ['5','1', '1', '1', '4', '-']
test_metric_scores = [90.98, 88.70, 88.78, 92.28, 92.70, 96.67]

results_summary = pd.DataFrame({'model_names':model_names, 'epochs':epochs,␣
↪'test_metric_score':test_metric_scores})
results_summary.sort_values(by=['test_metric_score'], ascending=False)
```

[3]:

| | model_names | epochs | test_metric_score |
|---|---|---|---|
| 5 | Research paper + BERT small + BERT large | - | 96.67 |
| 4 | Research Paper with Attention | 4 | 92.70 |
| 3 | Research Paper IMPL | 1 | 92.28 |
| 0 | CNN | 5 | 90.98 |
| 2 | Two Layered Bi-Directional LSTM | 1 | 88.78 |
| 1 | Single layer LSTM | 1 | 88.70 |