jigsaw-unintended-bias-toxicity-classification

March 28, 2020

[2]:
```
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

[3]:
```
# 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', \
```

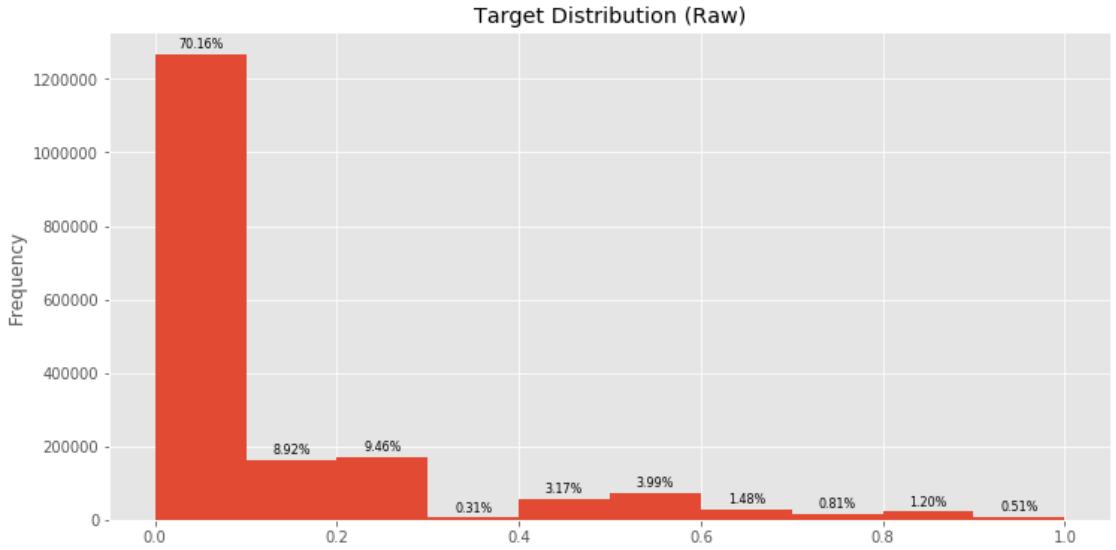
```
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',␣
 ↪'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'into', 'through', 'during',␣
 ↪'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on',␣
 ↪'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"]
```

[4]:
```
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

[13]:
```
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_data.shape[0]:.2f}%',
                (p.get_x() + p.get_width() / 2., p.get_height()),
```

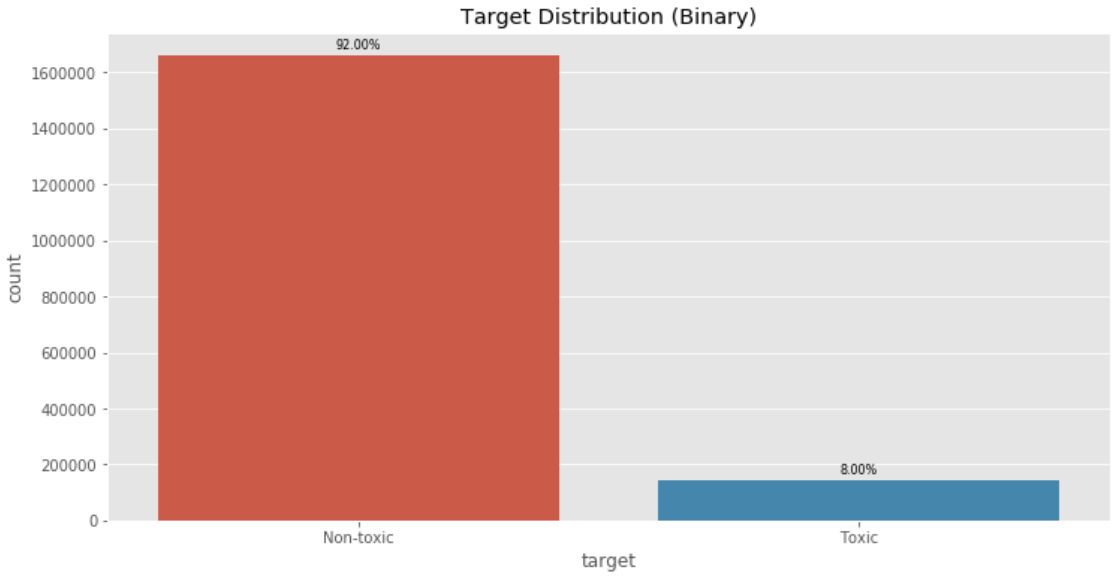
```
                ha='center',
                va='center',
                fontsize=8,
                color='black',
                xytext=(0,7),
                textcoords='offset points')
plt.title('Target Distribution (Raw)')
plt.show()
```



[6]:
```
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)
```

[16]:
```
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_data.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
```

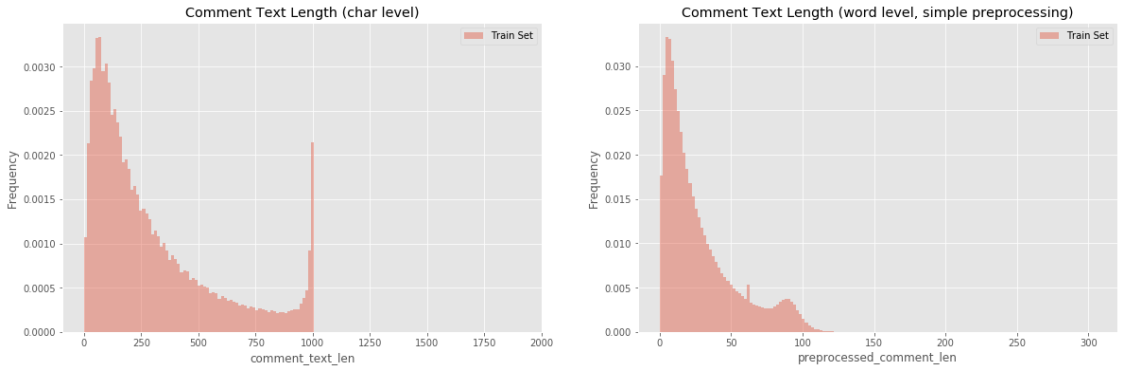
[15]:
```
def cleanComments(text):
    sent = decontracted(text)
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords).
↪lower().strip()
    return sent
```

[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

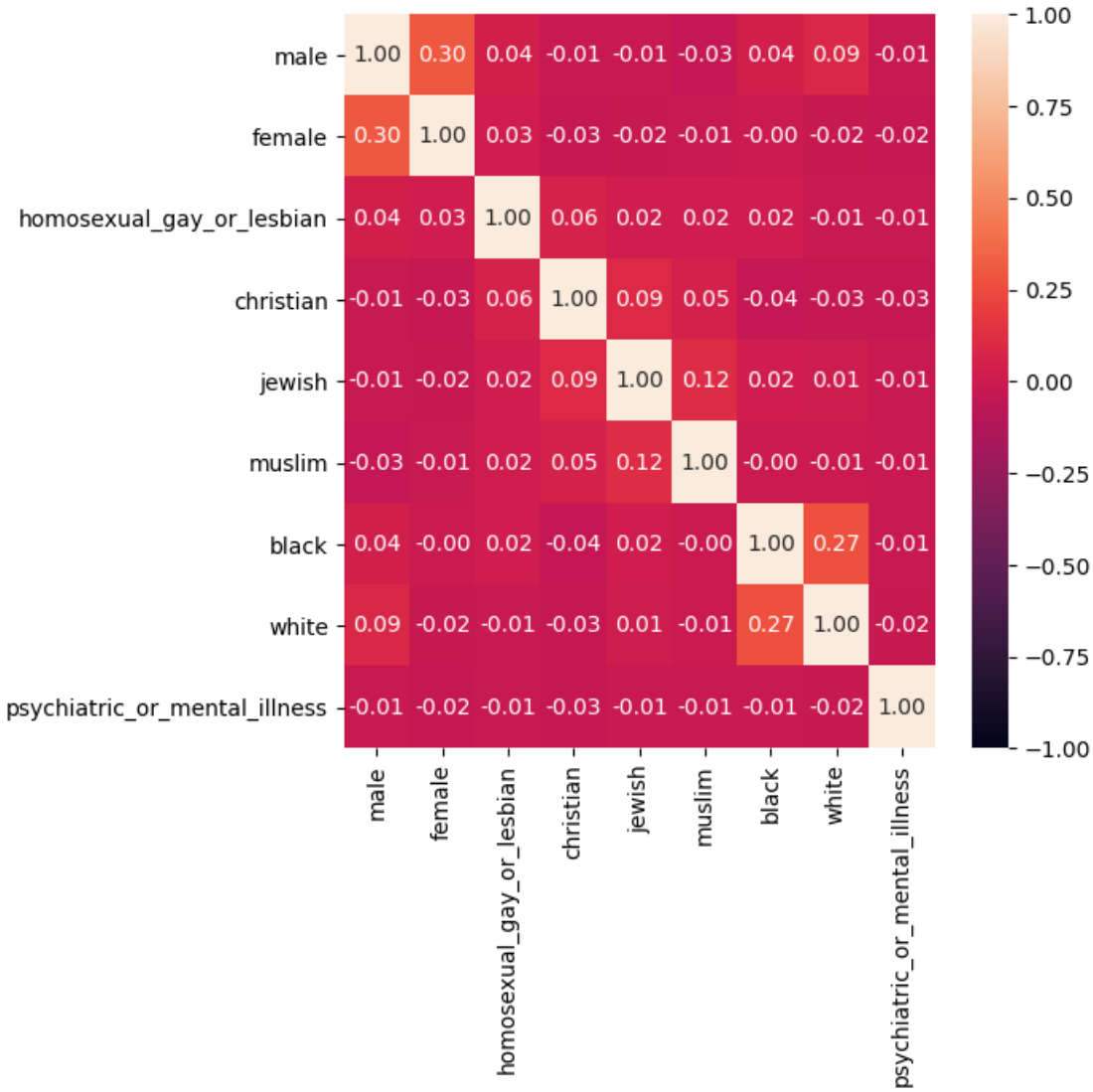
[35]:
```
for identity in selected_identities:
    counts = train_data[identity].sum()
    percentage = train_data[identity].sum() / train_data[identity].count() * 100
    print(f'{identity:<30}: {percentage:.2f}% , {counts}')
```

```
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()
```

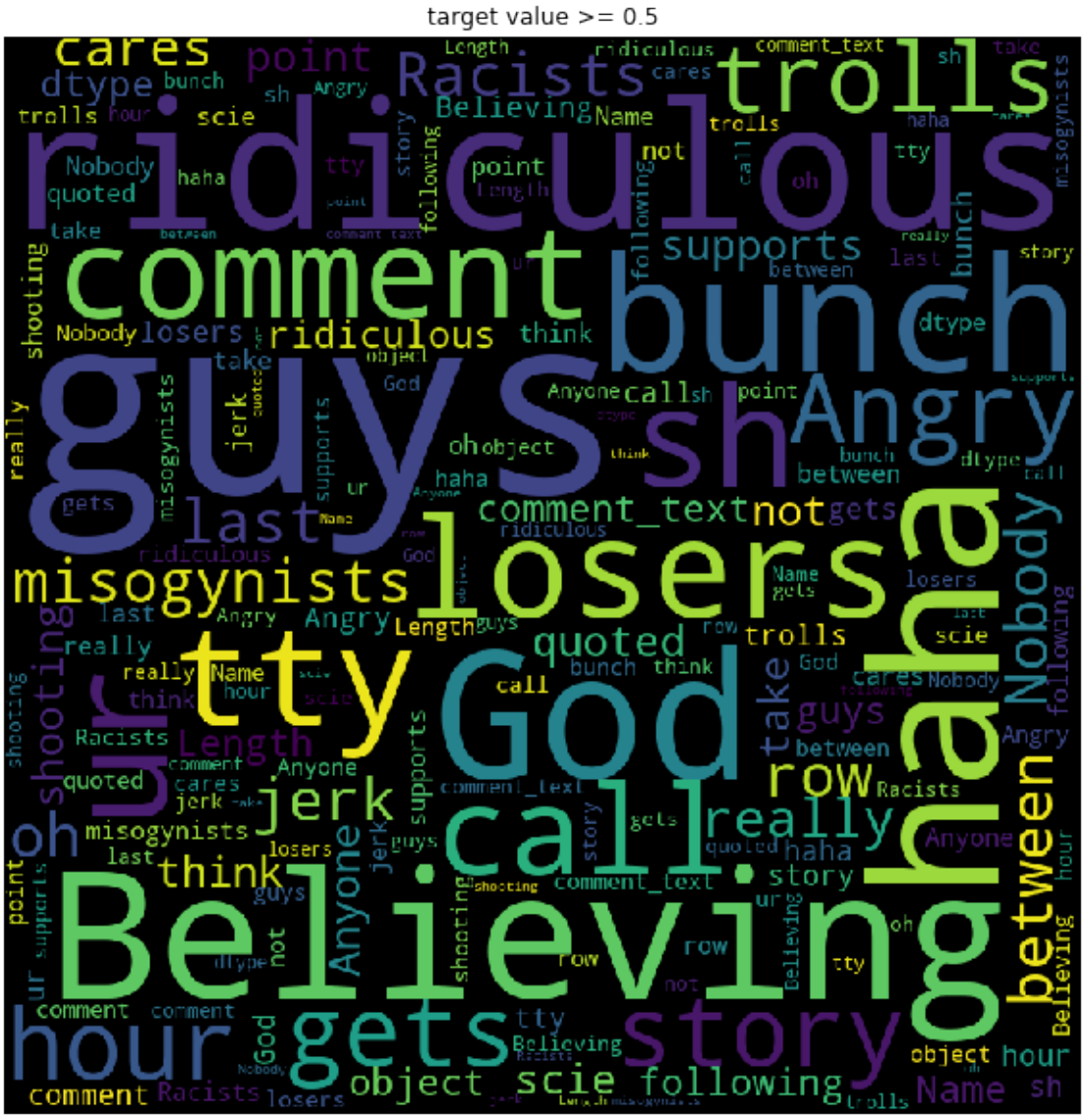


1.5 Word Cloud

[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)
```

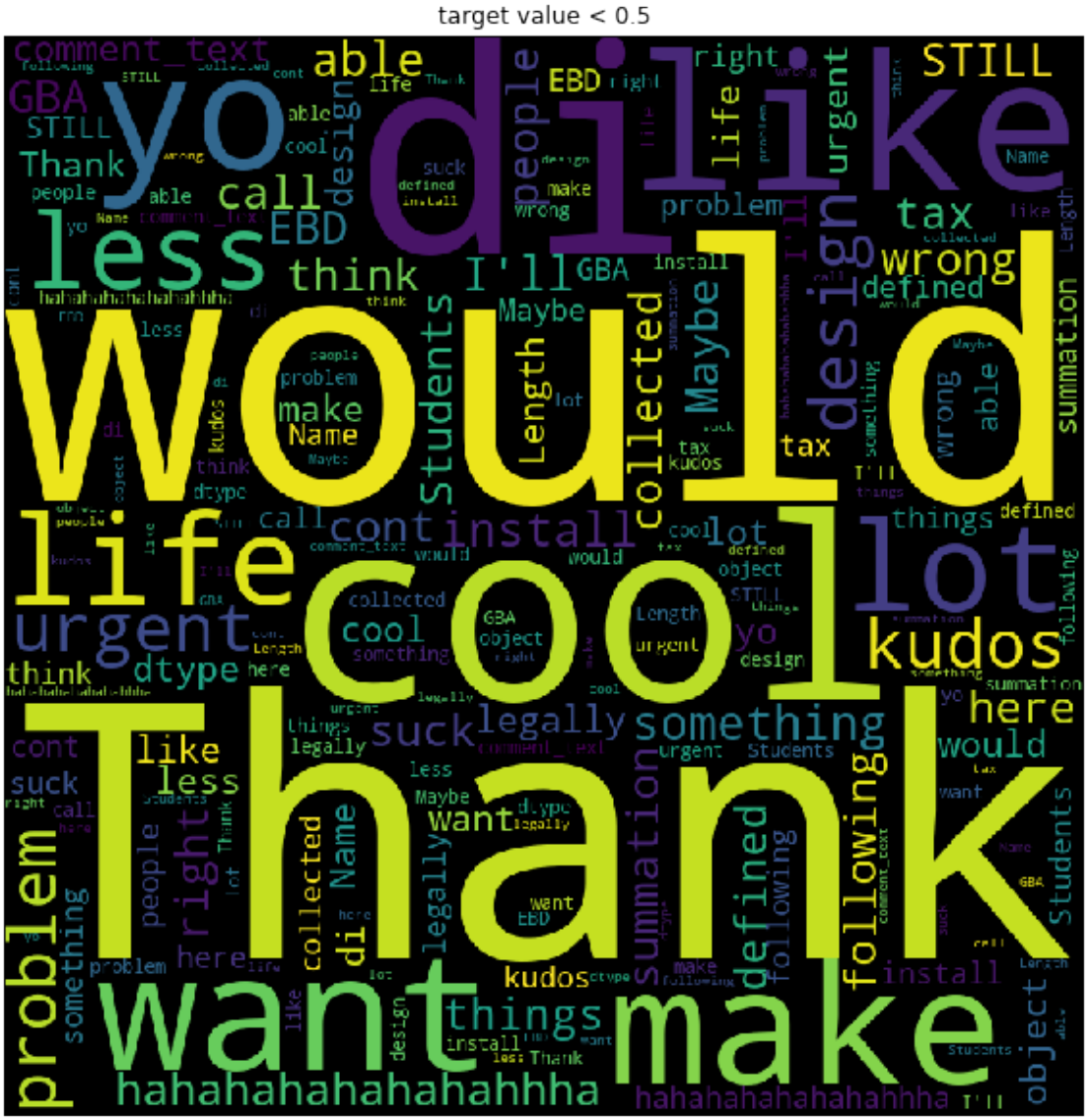
```
    plt.title(title)
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.tight_layout(pad = 0)
    plt.show()
```

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

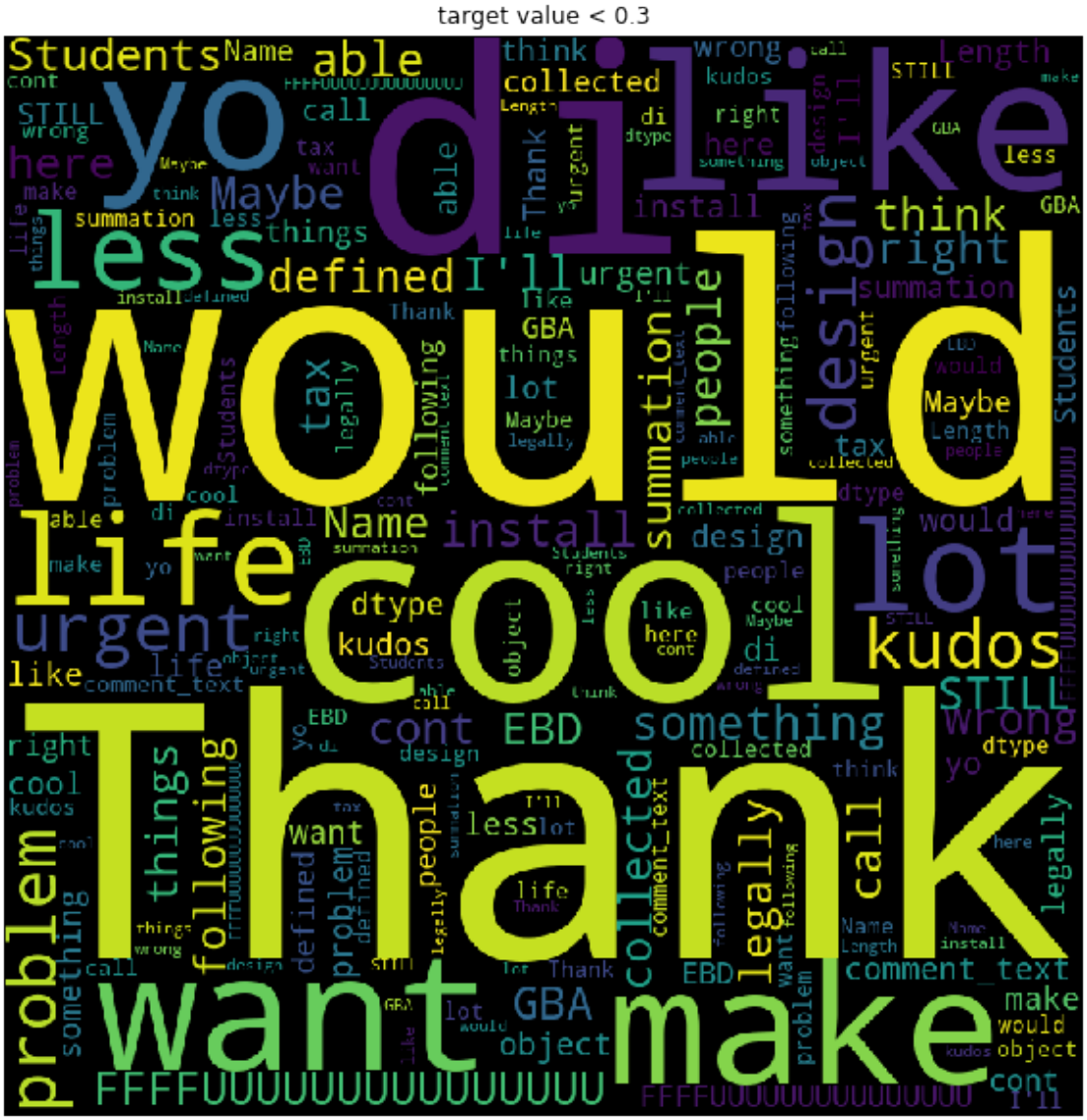


As we can see most used words having target >= 0.5 are slangs or related to religion or related to someone's behaviour and believes

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



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



most of the words used in comments having target<0.5 are normal english words that we use in our regular speaking. So we can say that our dataset doesnot has abnormal words i.e most probably toxic words and non toxic comments are labeled correctly.

[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

[7]:
```
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
```

[9]:
```
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
```

[10]:
```
cleaned_comments = cleanComments(train_data, 'comment_text')
cleaned_comments_test = cleanComments(test_data, 'comment_text')
```

```
100%|      | 1804874/1804874 [27:05<00:00, 1110.34it/s]
100%|      | 97320/97320 [01:27<00:00, 1107.66it/s]
```

[11]:
```
train_data['comment_text'] = cleaned_comments
test_data['comment_text'] = cleaned_comments_test
```

[12]:
```
train_data.comment_text.values[20383]
```

[12]:
```
'forward nicer littl peopl ok look pressur'
```

[13]:
```
train_data.comment_text.values[20000]
```

[13]:
```
'weapon start mr mani firearm take treat serious would becom cool peopl class
order elrey around u publicli seriou tool requir carri agre seen fashion comfort
god statement help'
```

[14]:
```
train_data.shape
```

[14]:
```
(1804874, 45)
```

[15]:
```
train_data.to_csv('train_data_cleaned.csv', index_label=False)
test_data.to_csv('test_data_cleaned.csv', index_label=False)
```

[16]:
```
train_data=pd.read_csv('train_data_cleaned.csv')
test_data=pd.read_csv('test_data_cleaned.csv')
```

3 Train test split (80% - 20%)

using stratified sampling to avoid bias while splitting data

[17]:
```
train_data, validation_data = train_test_split(train_data, test_size=0.2,␣
↪stratify=train_data.target.values, random_state=2020)
print(train_data.shape)
print(validation_data.shape)
```

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

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

[18]:
```
neg_train = train_data[train_data['target'] == True]
neg_train.shape
```

[18]:
```
(115467, 45)
```

[19]:
```
neg_validation = validation_data[validation_data['target'] == True]
neg_validation.shape
```

[19]:
```
(28867, 45)
```

[20]:
```
train_data.to_csv('train_data_splited.csv', index_label=False)
validation_data.to_csv('validation_data_splitted.csv', index_label=False)
```

[5]:
```
train_data=pd.read_csv('train_data_splited.csv')
validation_data=pd.read_csv('validation_data_splitted.csv')
test_data = pd.read_csv('test/test.csv')
```

[6]:
```
train_data.head()
```

[6]:

	id	target	comment_text	\
86452	348166	False	favorit equal one post misfortun part previou …	
1156017	5529565	False	justin abomin trudeau huge joke	
111702	378780	False	www theguardian jun news 2015 http com count 0…	
780699	5076044	False	oh mainland higher tax mental	
282234	587953	False	seen stock focus list valu fabric blinder clai…	

	severe_toxicity	obscene	identity_attack	insult	threat	asian	\
86452	0.000000	0.0625	0.000000	0.18750	0.0	NaN	
1156017	0.014493	0.0000	0.014493	0.42029	0.0	NaN	
111702	0.000000	0.0000	0.000000	0.00000	0.0	NaN	
780699	0.000000	0.0000	0.000000	0.00000	0.0	NaN	
282234	0.000000	0.0000	0.000000	0.00000	0.0	NaN	

	atheist	…	article_id	rating	funny	wow	sad	likes	disagree	\
86452	NaN	…	138562	approved	0	0	0	0	0	
1156017	NaN	…	351636	rejected	0	0	0	0	0	
111702	NaN	…	140837	approved	0	0	0	0	0	
780699	NaN	…	323299	approved	0	1	0	0	0	
282234	NaN	…	151058	approved	0	0	0	1	0	

	sexual_explicit	identity_annotator_count	toxicity_annotator_count
86452	0.234375	0	64
1156017	0.000000	0	69
111702	0.000000	0	4
780699	0.000000	0	4
282234	0.000000	0	4

[5 rows x 45 columns]

[7]:
```
y_train = train_data['target']
y_validation = validation_data['target']
```

4 Defining Evaluation Metric

[8]:
```
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
```

[9]:
```
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

[10]:
```
def vectorizeData(train, validation, test, 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)
        test_data_bow = bow_vectorizer.transform(test)
```

```
        return train_data_bow, validation_data_bow, test_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)
        test_data_tfidf = tfidf_vectorizer.transform(test)
        return train_data_tfidf, validation_data_tfidf, test_data_tfidf
```

[11]:
```
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

[12]:
```
train_comment_bow_25000, validation_comment_bow_25000, test_comment_bow_25000 =␣
↪vectorizeData(train_data['comment_text'], validation_data['comment_text'],␣
↪test_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}')
print(f'test_bow : {test_comment_bow_25000.shape}')

train_comment_tfidf_25000, validation_comment_tfidf_25000,␣
↪test_comment_tfidf_25000 = vectorizeData(train_data['comment_text'],␣
↪validation_data['comment_text'], test_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}')
print(f'test_tfidf : {test_comment_tfidf_25000.shape}')
```

```
train_bow : (1443899, 25000)
validation_bow : (360975, 25000)
test_bow : (97320, 25000)
train_tfidf : (1443899, 25000)
validation_tfidf : (360975, 25000)
test_tfidf : (97320, 25000)
```

15000 top words in bow and tfidf

[13]:
```
train_comment_bow_15000, validation_comment_bow_15000, test_comment_bow_15000 =␣
↪vectorizeData(train_data['comment_text'], validation_data['comment_text'],␣
↪test_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}')
```

```
print(f'test_bow : {test_comment_bow_15000.shape}')

train_comment_tfidf_15000, validation_comment_tfidf_15000,␣
↪test_comment_tfidf_15000 = vectorizeData(train_data['comment_text'],␣
↪validation_data['comment_text'], test_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}')
print(f'test_tfidf : {test_comment_tfidf_15000.shape}')
```

```
train_bow : (1443899, 15000)
validation_bow : (360975, 15000)
test_bow : (97320, 15000)
train_tfidf : (1443899, 15000)
validation_tfidf : (360975, 15000)
test_tfidf : (97320, 15000)
```

10000 top words in bow and tfidf

[14]:
```
train_comment_bow_10000, validation_comment_bow_10000, test_comment_bow_10000 =␣
↪vectorizeData(train_data['comment_text'], validation_data['comment_text'],␣
↪test_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}')
print(f'test_bow : {test_comment_bow_10000.shape}')

train_comment_tfidf_10000, validation_comment_tfidf_10000,␣
↪test_comment_tfidf_10000 = vectorizeData(train_data['comment_text'],␣
↪validation_data['comment_text'], test_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}')
print(f'test_tfidf : {test_comment_tfidf_10000.shape}')
```

```
train_bow : (1443899, 10000)
validation_bow : (360975, 10000)
test_bow : (97320, 10000)
train_tfidf : (1443899, 10000)
validation_tfidf : (360975, 10000)
test_tfidf : (97320, 10000)
```

[17]:
```
#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')
```

[18]:
```
# 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

[31]:
```
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 [01:08<00:00, 11.49s/it]
```

[32]:
```
import gc
gc.collect()
```

[32]:
```
20
```

[33]:
```
score = pd.DataFrame({'name':names, 'train_score':train_auc_list, 'test_score':
 ↪validation_auc_list}).sort_values(by=['test_score'])
score
```

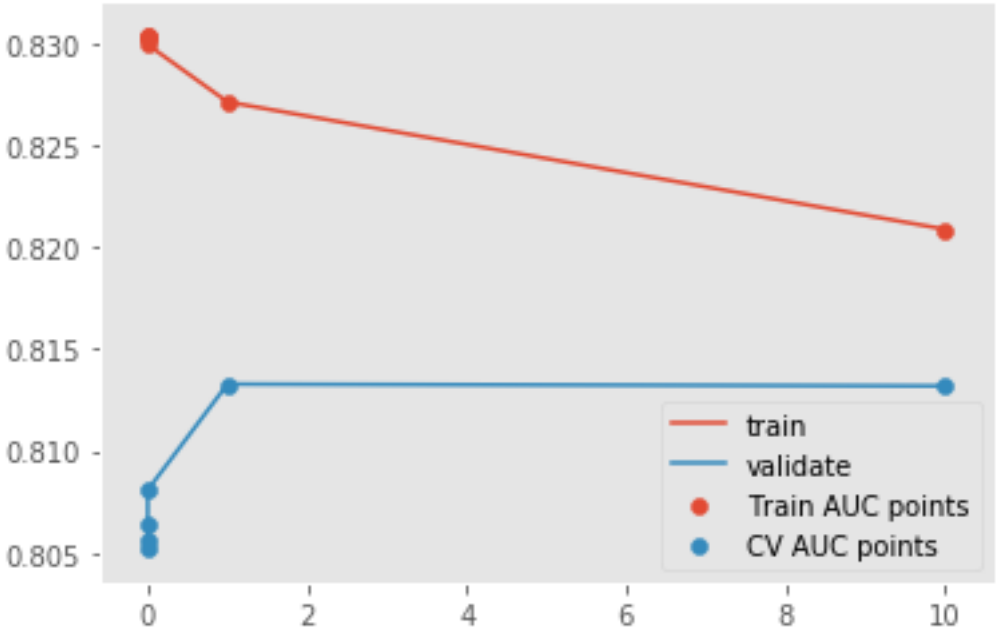
[33]:

	name	train_score	test_score
0	NB-BOW_25k_1e-09	0.830385	0.805230
1	NB-BOW_25k_1e-07	0.830369	0.805561
2	NB-BOW_25k_1e-05	0.830297	0.806373
3	NB-BOW_25k_0.001	0.829962	0.808092
5	NB-BOW_25k_10	0.820896	0.813198
4	NB-BOW_25k_1	0.827153	0.813289

[34]:
```
print(train_auc_list,validation_auc_list)
print(f'best hyperparameter got = {score.name.values[-1]} ##### Best cv score␣
 ↪got = {score.test_score.values[-1]}')
plt.plot(alpha, train_auc_list, label='train')
plt.plot(alpha, validation_auc_list, label='validate')
plt.scatter(alpha, train_auc_list, label='Train AUC points')
plt.scatter(alpha, validation_auc_list, label='CV AUC points')
plt.legend()
plt.grid()
plt.show()
```

```
[0.8303848235597261, 0.8303685885882189, 0.8302966785002497, 0.829961728468076,
0.827152860228968, 0.8208963212816531] [0.8052299001325809, 0.8055613532368453,
```

```
0.8063728503898459, 0.8080922245271929, 0.8132894711382146, 0.8131982861162018]
best hyperparameter got = NB-BOW_25k_1 ##### Best cv score got =
0.8132894711382146
```



[35]:
```
MODEL_NAME = 'NB_BOW_25k'
clf = MultinomialNB(alpha=1)
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]
```

[36]:
```
train_data[MODEL_NAME] = predicted_train
validation_data[MODEL_NAME] = predicted_validation
print(f'Train score = {get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME)}')
print(f'Validation score = {get_metric_value(validation_data, identity_columns,␣
 ↪MODEL_NAME)}')
```

```
Train score = 0.827152860228968
Validation score = 0.8132894711382146
```

[37]:
```
predicted_test = clf.predict_proba(test_comment_bow_25000)[:,1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/NB_BOW_25k_submission.csv', index=False)
```

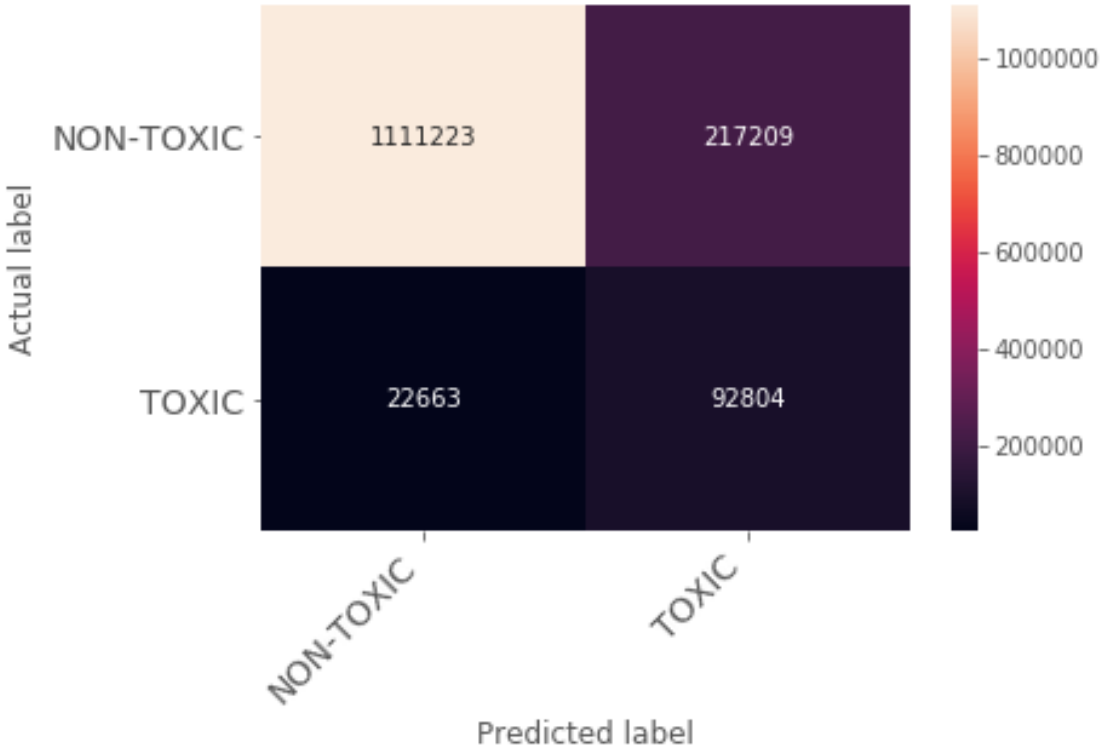
[38]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
```

```
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()
```

[39]:
```
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'])
```

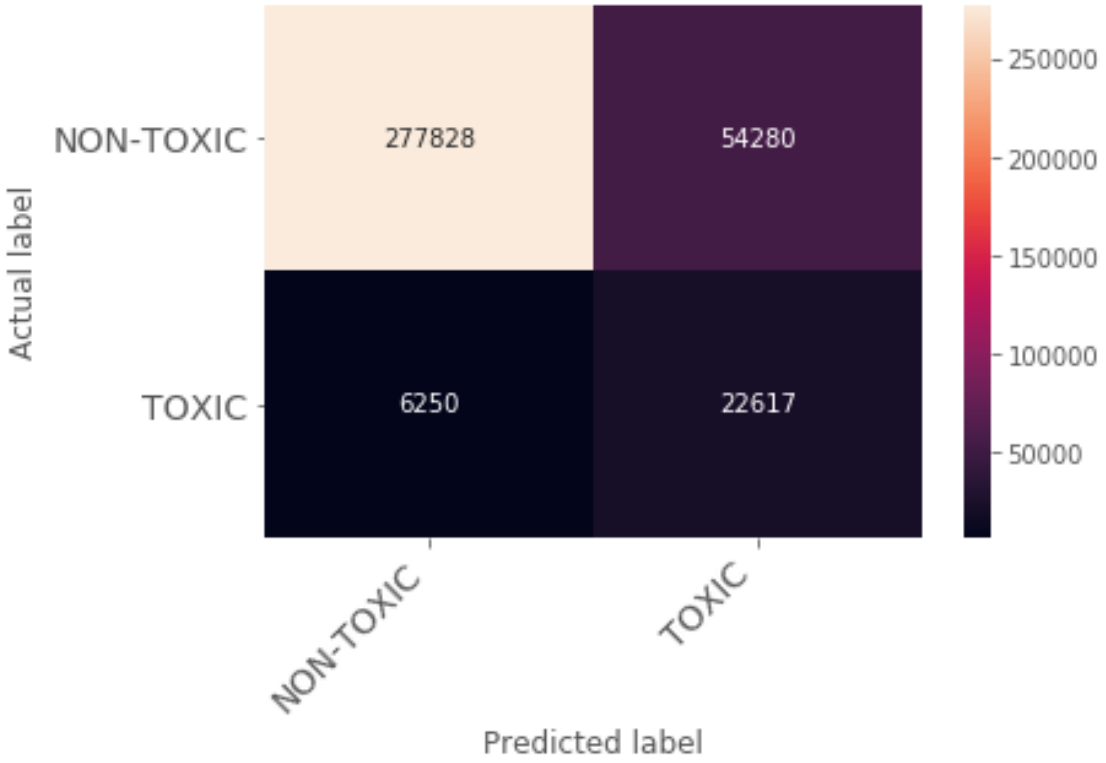
```
        TRAIN DATA CONFUSION MATRIX
```



=> 83.64 % of non-toxic comments predicted correctly => 80.37% of toxic comments predicted correctly

[40]:
```
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
```



=> 83.65 % of non-toxic comments predicted correctly => 78.37% of toxic comments predicted correctly

15000 features

[41]:
```
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 [01:11<00:00, 11.86s/it]
```

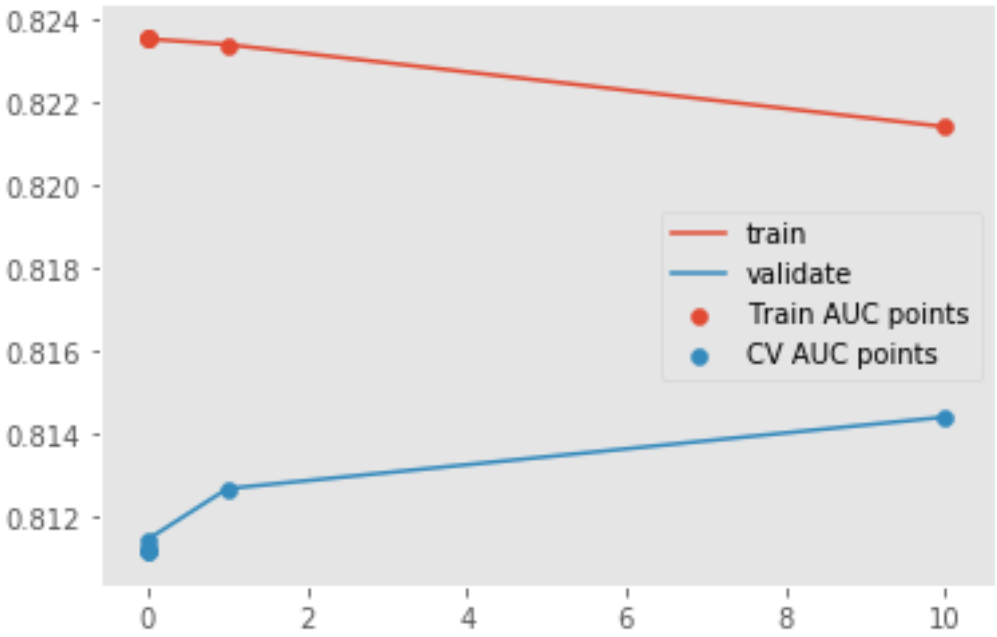
[42]:
```
score = pd.DataFrame({'name':names, 'train_score':train_auc_list, 'test_score':
 ↪validation_auc_list}).sort_values(by=['test_score'])
score
```

[42]:

	name	train_score	test_score
0	NB-BOW_15k_1e-09	0.823568	0.811151
1	NB-BOW_15k_1e-07	0.823568	0.811177
2	NB-BOW_15k_1e-05	0.823565	0.811245
3	NB-BOW_15k_0.001	0.823551	0.811434
4	NB-BOW_15k_1	0.823416	0.812662
5	NB-BOW_15k_10	0.821424	0.814395

[43]:
```
print(train_auc_list,validation_auc_list)
print(f'best hyperparameter got = {score.name.values[-1]} ##### Best cv score␣
 ↪got = {score.test_score.values[-1]}')
plt.plot(alpha, train_auc_list, label='train')
plt.plot(alpha, validation_auc_list, label='validate')
plt.scatter(alpha, train_auc_list, label='Train AUC points')
plt.scatter(alpha, validation_auc_list, label='CV AUC points')
plt.legend()
plt.grid()
plt.show()
```

```
[0.8235684449002456, 0.8235677783155043, 0.8235650158577439, 0.8235510348300056,
0.8234156579630094, 0.8214236834988156] [0.811151273030675, 0.8111772725244157,
0.8112452836045302, 0.8114336536220598, 0.8126621147166926, 0.8143945561931168]
best hyperparameter got = NB-BOW_15k_10 ##### Best cv score got =
0.8143945561931168
```



[44]:
```
MODEL_NAME = 'NB_BOW_15k'
clf = MultinomialNB(alpha=10)
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]
```

[45]:
```
train_data[MODEL_NAME] = predicted_train
validation_data[MODEL_NAME] = predicted_validation
print(f'Train score = {get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME)}')
print(f'Validation score = {get_metric_value(validation_data, identity_columns,␣
 ↪MODEL_NAME)}')
```

```
Train score = 0.8214236834988156
Validation score = 0.8143945561931168
```

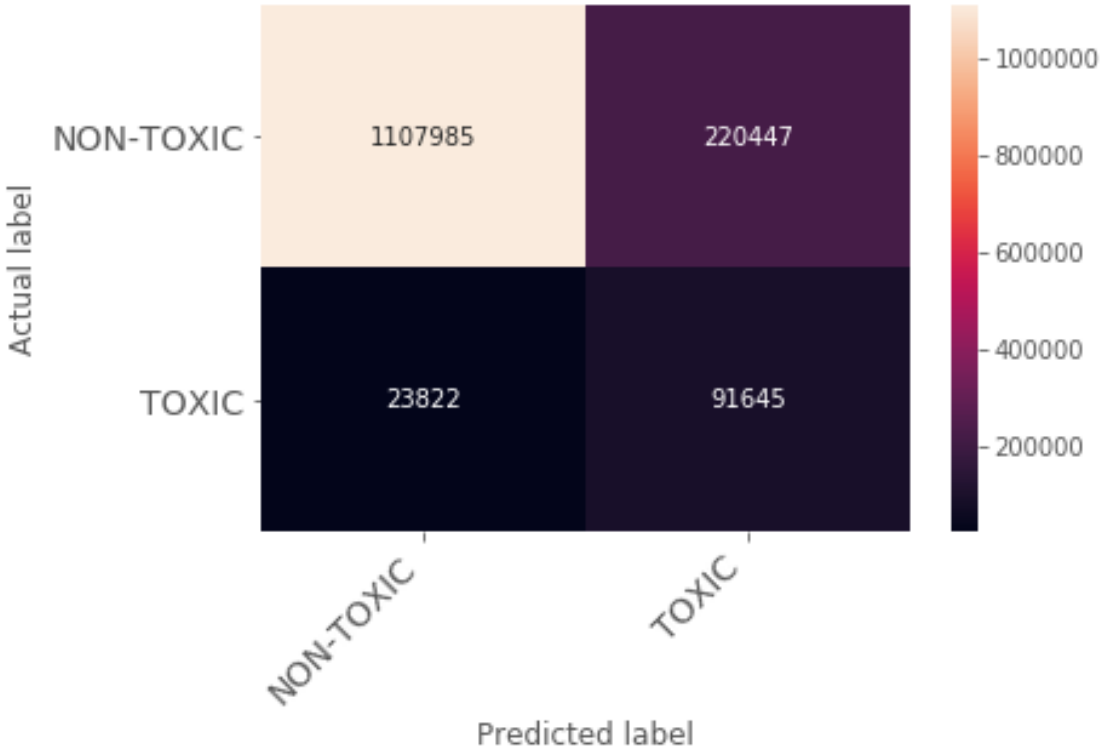
[46]:
```
predicted_test = clf.predict_proba(test_comment_bow_15000)[:,1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/NB_BOW_15k_submission.csv', index=False)
```

[47]:
```
# 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()
```

[48]:
```
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'])
```

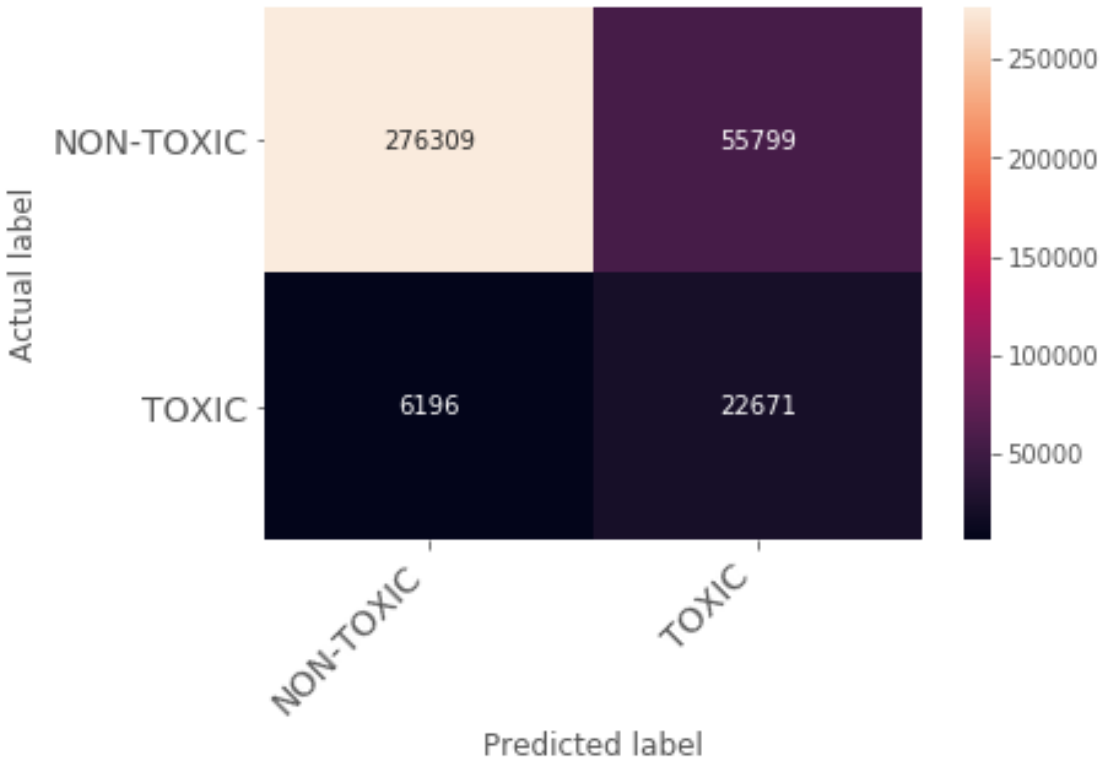
```
TRAIN DATA CONFUSION MATRIX
```



=> 83.64 % of non-toxic comments predicted correctly => 80.37% of toxic comments predicted correctly

[49]:
```
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
```



=> 83.64 % of non-toxic comments predicted correctly => 80.37% of toxic comments predicted correctly

10000 features

[50]:
```
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 [01:16<00:00, 12.79s/it]
```

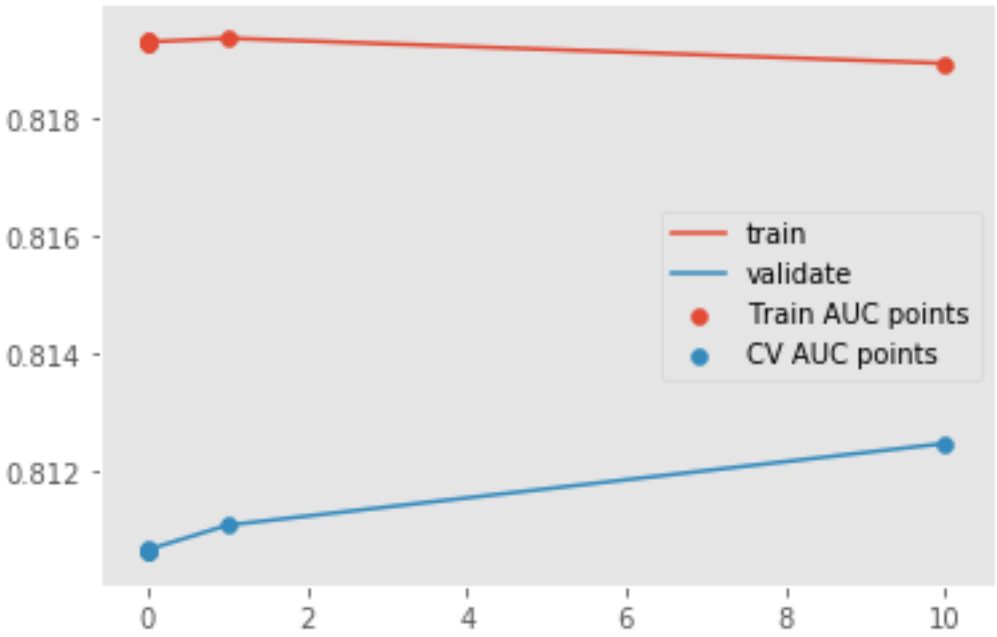
[51]:
```
score = pd.DataFrame({'name':names, 'train_score':train_auc_list, 'test_score':
 ↪validation_auc_list}).sort_values(by=['test_score'])
score
```

[51]:

	name	train_score	test_score
0	NB-BOW_10k_1e-09	0.819319	0.810637
1	NB-BOW_10k_1e-07	0.819319	0.810639
2	NB-BOW_10k_1e-05	0.819319	0.810644
3	NB-BOW_10k_0.001	0.819316	0.810660
4	NB-BOW_10k_1	0.819380	0.811077
5	NB-BOW_10k_10	0.818952	0.812470

[52]:
```
print(train_auc_list,validation_auc_list)
print(f'best hyperparameter got = {score.name.values[-1]} ##### Best cv score␣
 ↪got = {score.test_score.values[-1]}')
plt.plot(alpha, train_auc_list, label='train')
plt.plot(alpha, validation_auc_list, label='validate')
plt.scatter(alpha, train_auc_list, label='Train AUC points')
plt.scatter(alpha, validation_auc_list, label='CV AUC points')
plt.legend()
plt.grid()
plt.show()
```

```
[0.8193192247971082, 0.8193191310344902, 0.8193187500257502, 0.8193163314649758,
0.8193802291582439, 0.8189522244639298] [0.8106369318316213, 0.8106388164386731,
0.8106442853842235, 0.8106595574355672, 0.8110768916283557, 0.8124702812251251]
best hyperparameter got = NB-BOW_10k_10 ##### Best cv score got =
0.8124702812251251
```



[53]:
```
MODEL_NAME = 'NB_BOW_10k'
clf = MultinomialNB(alpha=1)
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]
```

[54]:
```
train_data[MODEL_NAME] = predicted_train
validation_data[MODEL_NAME] = predicted_validation
print(f'Train score = {get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME)}')
print(f'Validation score = {get_metric_value(validation_data, identity_columns,␣
 ↪MODEL_NAME)}')
```

```
Train score = 0.8193802291582439
Validation score = 0.8110768916283557
```

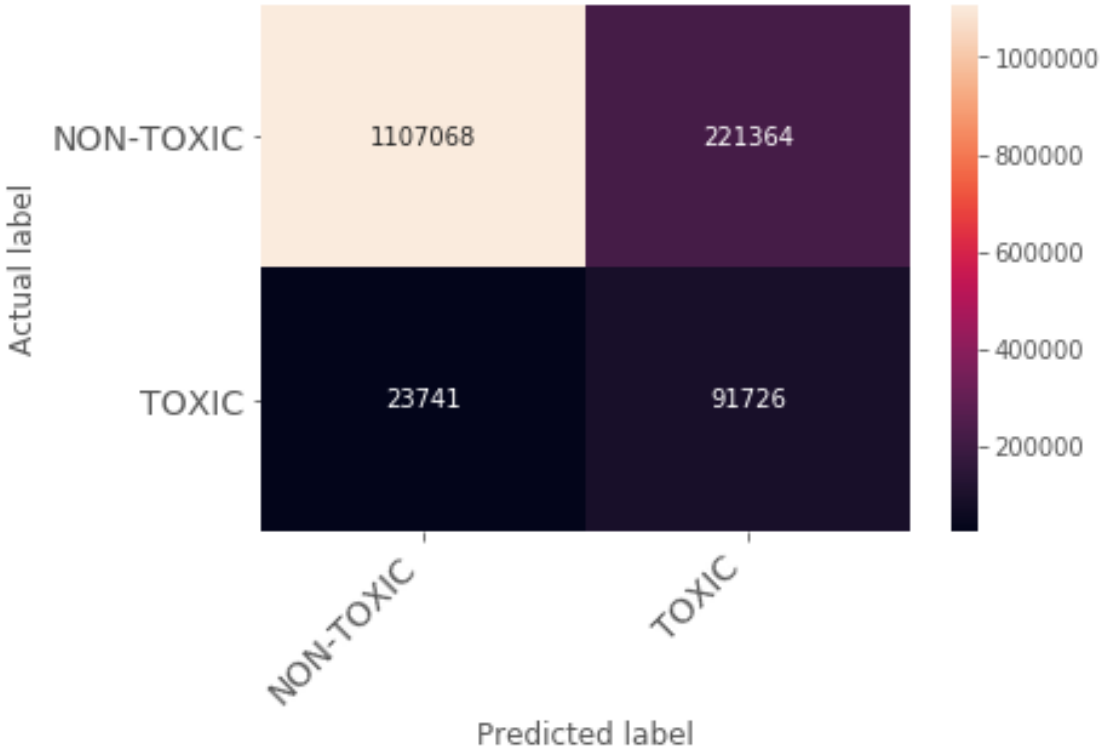
[55]:
```
predicted_test = clf.predict_proba(test_comment_bow_10000)[:,1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/NB_BOW_10k_submission.csv', index=False)
```

[56]:
```
# 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()
```

[57]:
```
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'])
```

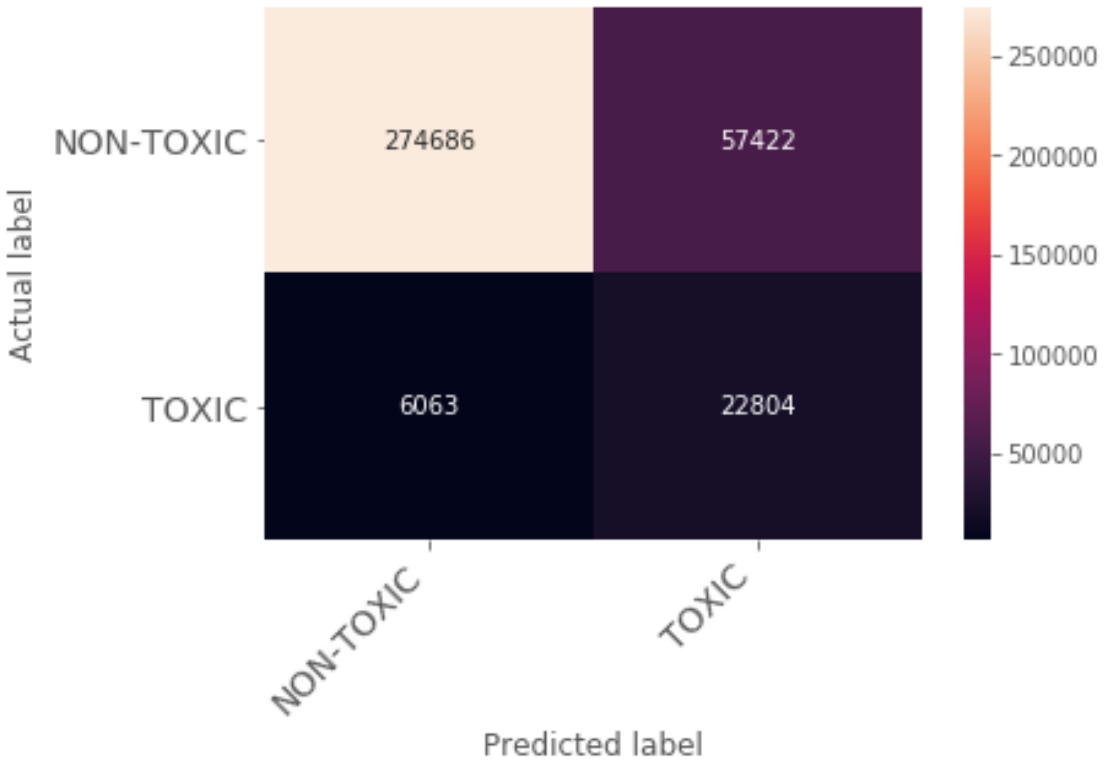
```
TRAIN DATA CONFUSION MATRIX
```



=> 83.64 % of non-toxic comments predicted correctly => 80.37% of toxic comments predicted correctly

[58]:
```
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
```



=> 83.64 % of non-toxic comments predicted correctly => 80.37% of toxic comments predicted correctly

Considering TFIDF

25000 features

[59]:
```
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 [01:19<00:00, 13.22s/it]
```

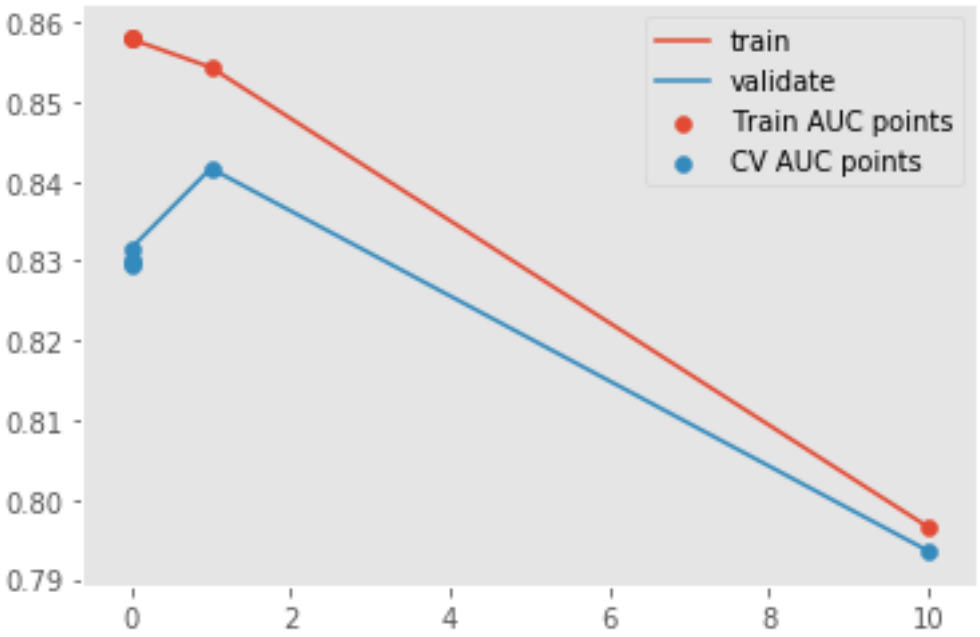
[60]:
```
score = pd.DataFrame({'name':names, 'train_score':train_auc_list, 'test_score':
↪validation_auc_list}).sort_values(by=['test_score'])
score
```

[60]:

	name	train_score	test_score
5	NB-tfidf_25k_10	0.796550	0.793550
0	NB-tfidf_25k_1e-09	0.858006	0.829614
1	NB-tfidf_25k_1e-07	0.858005	0.829701
2	NB-tfidf_25k_1e-05	0.857992	0.830077
3	NB-tfidf_25k_0.001	0.857891	0.831597
4	NB-tfidf_25k_1	0.854349	0.841650

[61]:
```
print(train_auc_list,validation_auc_list)
print(f'best hyperparameter got = {score.name.values[-1]} ##### Best cv score␣
↪got = {score.test_score.values[-1]}')
plt.plot(alpha, train_auc_list, label='train')
plt.plot(alpha, validation_auc_list, label='validate')
plt.scatter(alpha, train_auc_list, label='Train AUC points')
plt.scatter(alpha, validation_auc_list, label='CV AUC points')
plt.legend()
plt.grid()
plt.show()
```

```
[0.8580058604880463, 0.8580045470157118, 0.8579923573881973, 0.8578913522811471,
0.8543485224611782, 0.7965498426042015] [0.8296141165672792, 0.829700624425765,
0.8300772749678597, 0.831597373696867, 0.841650375369775, 0.7935504787284842]
best hyperparameter got = NB-tfidf_25k_1 ##### Best cv score got =
0.841650375369775
```



[62]:
```
MODEL_NAME = 'NB_tfidf_25k'
clf = MultinomialNB(alpha=1)
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]
```

[63]:
```
train_data[MODEL_NAME] = predicted_train
validation_data[MODEL_NAME] = predicted_validation
print(f'Train score = {get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME)}')
print(f'Validation score = {get_metric_value(validation_data, identity_columns,␣
 ↪MODEL_NAME)}')
```

```
Train score = 0.8543485224611782
Validation score = 0.841650375369775
```

[64]:
```
predicted_test = clf.predict_proba(test_comment_tfidf_25000)[:,1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/NB_tfidf_25k_submission.csv', index=False)
```

[ ]:
```

```

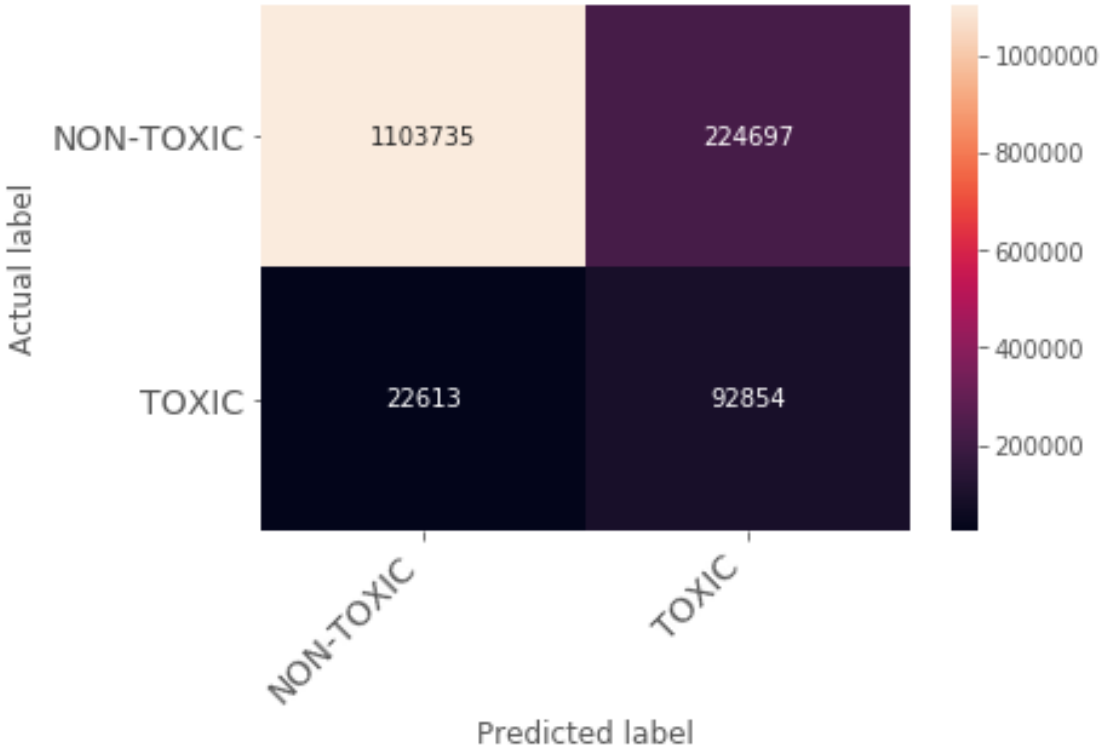
[65]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
 ↪predicted_validation)
```

```
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()
```

[66]:
```
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'])
```

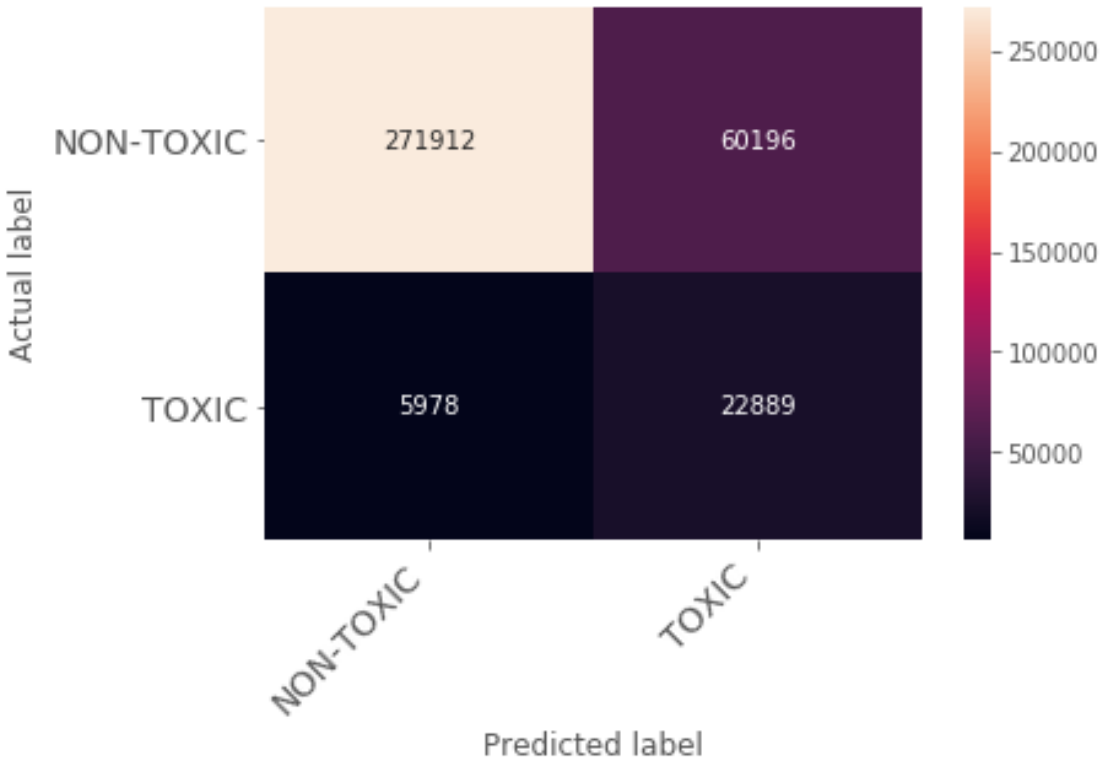
```
TRAIN DATA CONFUSION MATRIX
```



=> 83.08 % of non-toxic comments predicted correctly => 80.37% of toxic comments predicted correctly

[67]:
```
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
```



=> 83.64 % of non-toxic comments predicted correctly => 81.37% of toxic comments predicted correctly

15000 features

[68]:
```
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 [01:20<00:00, 13.44s/it]
```

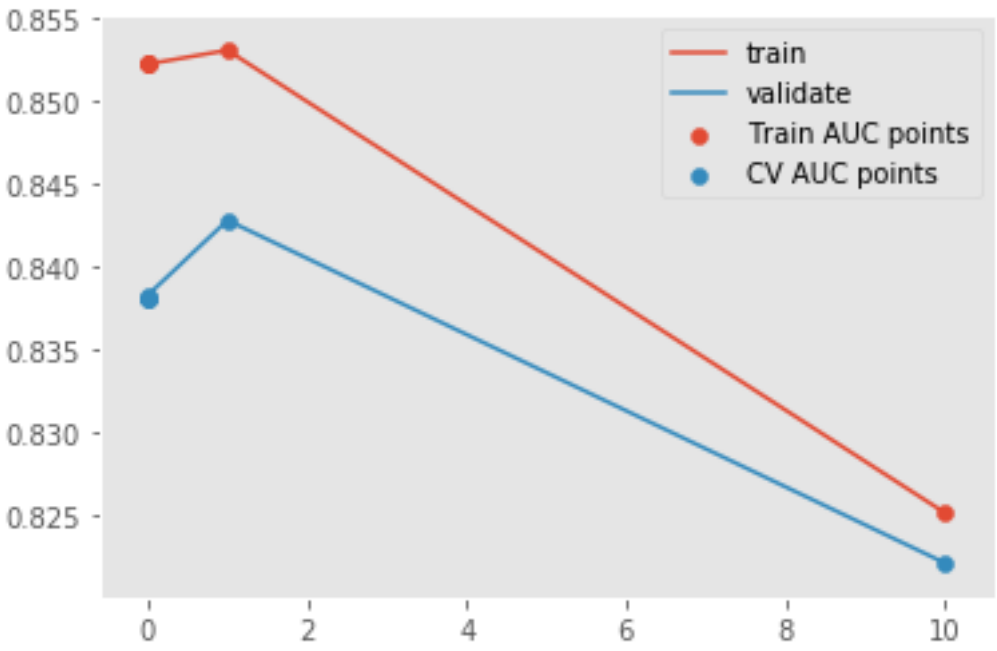
[69]:
```
score = pd.DataFrame({'name':names, 'train_score':train_auc_list, 'test_score':
 ↪validation_auc_list}).sort_values(by=['test_score'])
score
```

[69]:

	name	train_score	test_score
5	NB-tfidf_15k_10	0.825208	0.822186
0	NB-tfidf_15k_1e-09	0.852262	0.838132
1	NB-tfidf_15k_1e-07	0.852262	0.838139
2	NB-tfidf_15k_1e-05	0.852261	0.838179
3	NB-tfidf_15k_0.001	0.852260	0.838334
4	NB-tfidf_15k_1	0.853095	0.842844

[70]:
```
print(train_auc_list,validation_auc_list)
print(f'best hyperparameter got = {score.name.values[-1]} ##### Best cv score␣
 ↪got = {score.test_score.values[-1]}')
plt.plot(alpha, train_auc_list, label='train')
plt.plot(alpha, validation_auc_list, label='validate')
plt.scatter(alpha, train_auc_list, label='Train AUC points')
plt.scatter(alpha, validation_auc_list, label='CV AUC points')
plt.legend()
plt.grid()
plt.show()
```

```
[0.8522619696619693, 0.8522619191058425, 0.8522614506947731, 0.8522597430300503,
0.8530951191819393, 0.8252077260229658] [0.8381319855023279, 0.8381391608991691,
0.8381789805232116, 0.8383343802973273, 0.8428440899522812, 0.8221860833729862]
best hyperparameter got = NB-tfidf_15k_1 ##### Best cv score got =
0.8428440899522812
```



[71]:
```
MODEL_NAME = 'NB_tfidf_15k'
clf = MultinomialNB(alpha=1)
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]
```

[72]:
```
train_data[MODEL_NAME] = predicted_train
validation_data[MODEL_NAME] = predicted_validation
print(f'Train score = {get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME)}')
print(f'Validation score = {get_metric_value(validation_data, identity_columns,␣
 ↪MODEL_NAME)}')
```

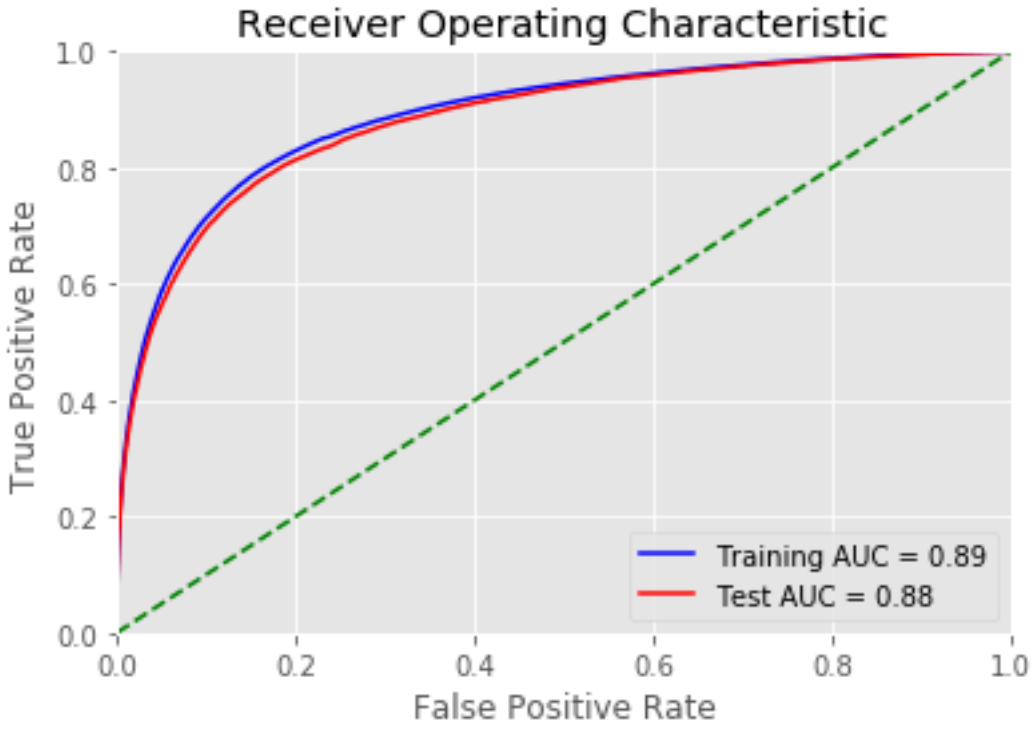
```
Train score = 0.8530951191819393
Validation score = 0.8428440899522812
```

[73]:
```
predicted_test = clf.predict_proba(test_comment_tfidf_15000)[:,1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/NB_tfidf_15k_submission.csv', index=False)
```

[ ]:
```

```

[74]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
```

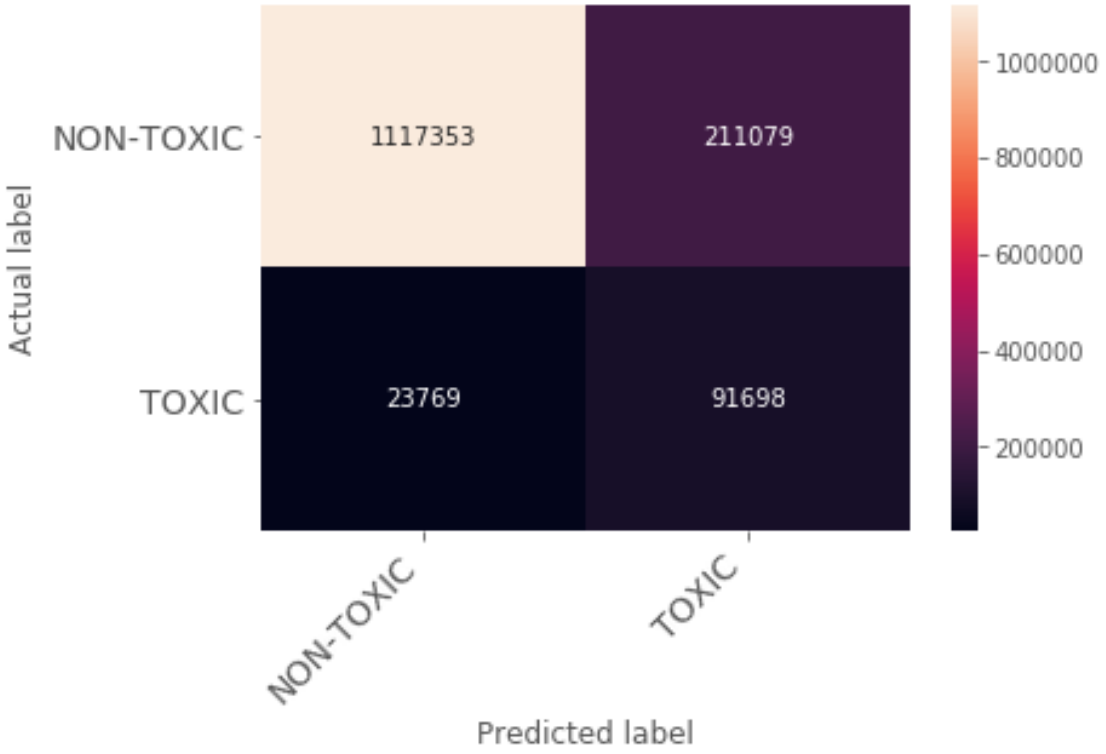
```
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
↪predicted_validation)

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()
```



[75]:
```
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'])
```

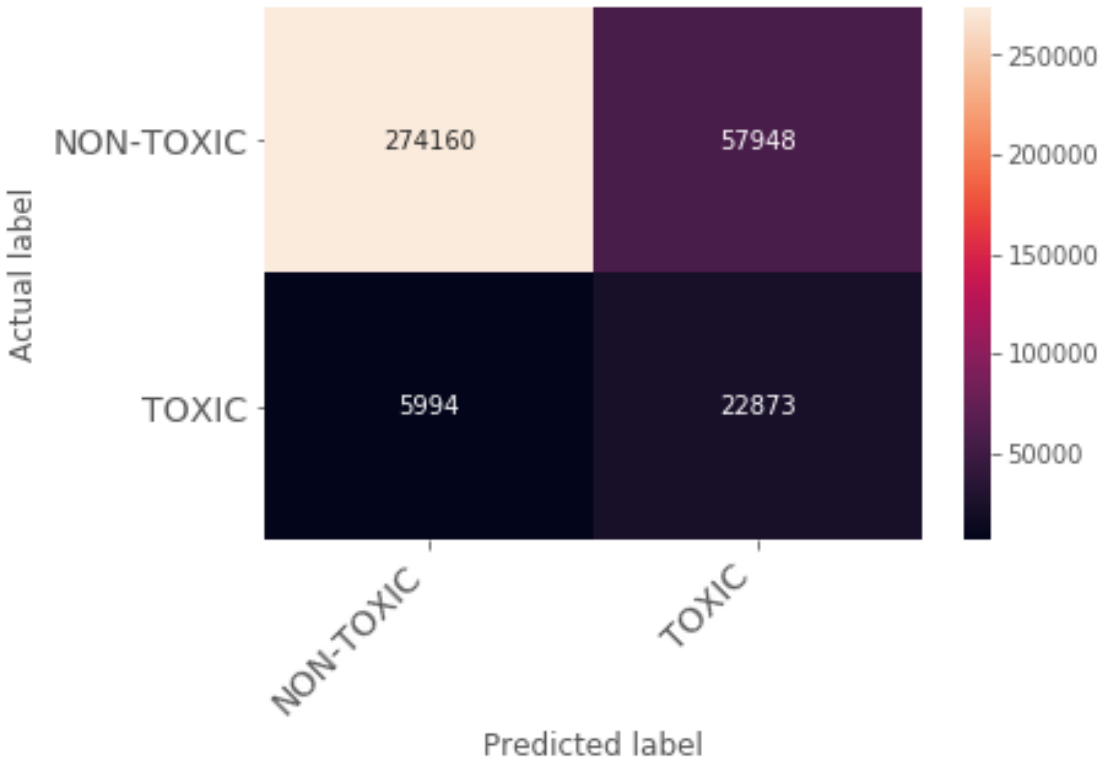
```
        TRAIN DATA CONFUSION MATRIX
```



=> 84.11 % of non-toxic comments predicted correctly => 79.41% of toxic comments predicted correctly

[76]:
```
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
```



=> 82.55% of non-toxic comments predicted correctly => 79.73% of toxic comments predicted correctly

10000 features

[77]:
```
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 [01:23<00:00, 13.96s/it]
```

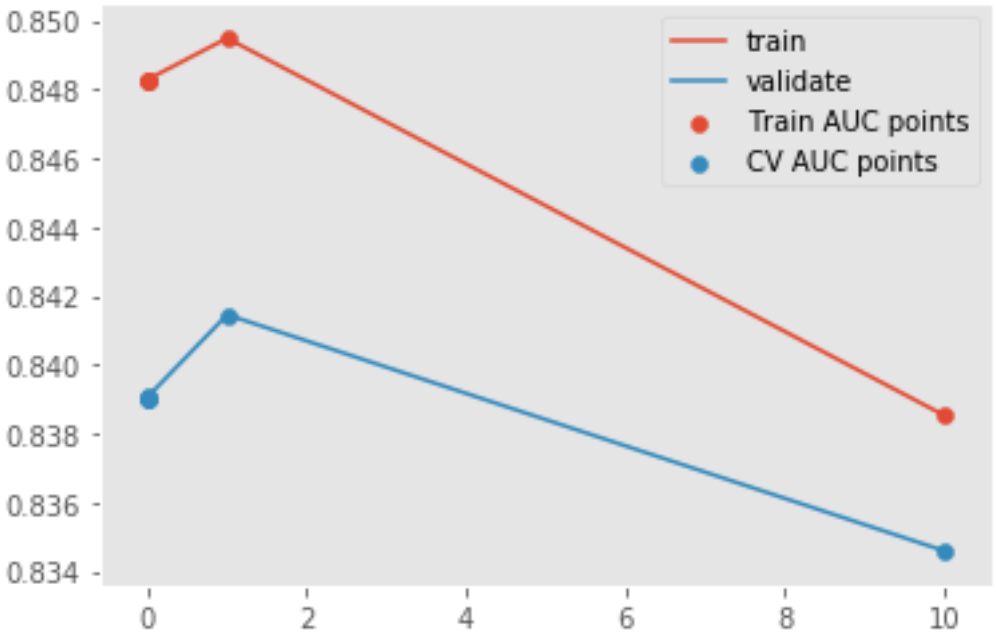
[78]:
```
score = pd.DataFrame({'name':names, 'train_score':train_auc_list, 'test_score':
 ↪validation_auc_list}).sort_values(by=['test_score'])
score
```

[78]:

	name	train_score	test_score
5	NB-tfidf_10k_10	0.838529	0.834594
0	NB-tfidf_10k_1e-09	0.848289	0.839049
1	NB-tfidf_10k_1e-07	0.848289	0.839049
2	NB-tfidf_10k_1e-05	0.848289	0.839050
3	NB-tfidf_10k_0.001	0.848290	0.839067
4	NB-tfidf_10k_1	0.849492	0.841459

[79]:
```
print(train_auc_list,validation_auc_list)
print(f'best hyperparameter got = {score.name.values[-1]} ##### Best cv score␣
 ↪got = {score.test_score.values[-1]}')
plt.plot(alpha, train_auc_list, label='train')
plt.plot(alpha, validation_auc_list, label='validate')
plt.scatter(alpha, train_auc_list, label='Train AUC points')
plt.scatter(alpha, validation_auc_list, label='CV AUC points')
plt.legend()
plt.grid()
plt.show()
```

```
[0.8482890240897429, 0.848289017167702, 0.8482889367196303, 0.8482898011152245,
0.8494916500190987, 0.8385287368673903] [0.839048724447654, 0.83904880083159,
0.8390503286699291, 0.8390668321924513, 0.8414590279205327, 0.8345943754450087]
best hyperparameter got = NB-tfidf_10k_1 ##### Best cv score got =
0.8414590279205327
```



[80]:
```
MODEL_NAME = 'NB_tfidf_10k'
clf = MultinomialNB(alpha=1)
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]
```

[81]:
```
train_data[MODEL_NAME] = predicted_train
validation_data[MODEL_NAME] = predicted_validation
print(f'Train score = {get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME)}')
print(f'Validation score = {get_metric_value(validation_data, identity_columns,␣
 ↪MODEL_NAME)}')
```

```
Train score = 0.8494916500190987
Validation score = 0.8414590279205327
```

[82]:
```
predicted_test = clf.predict_proba(test_comment_tfidf_10000)[:,1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/NB_tfidf_10k_submission.csv', index=False)
```

[ ]:
```

```

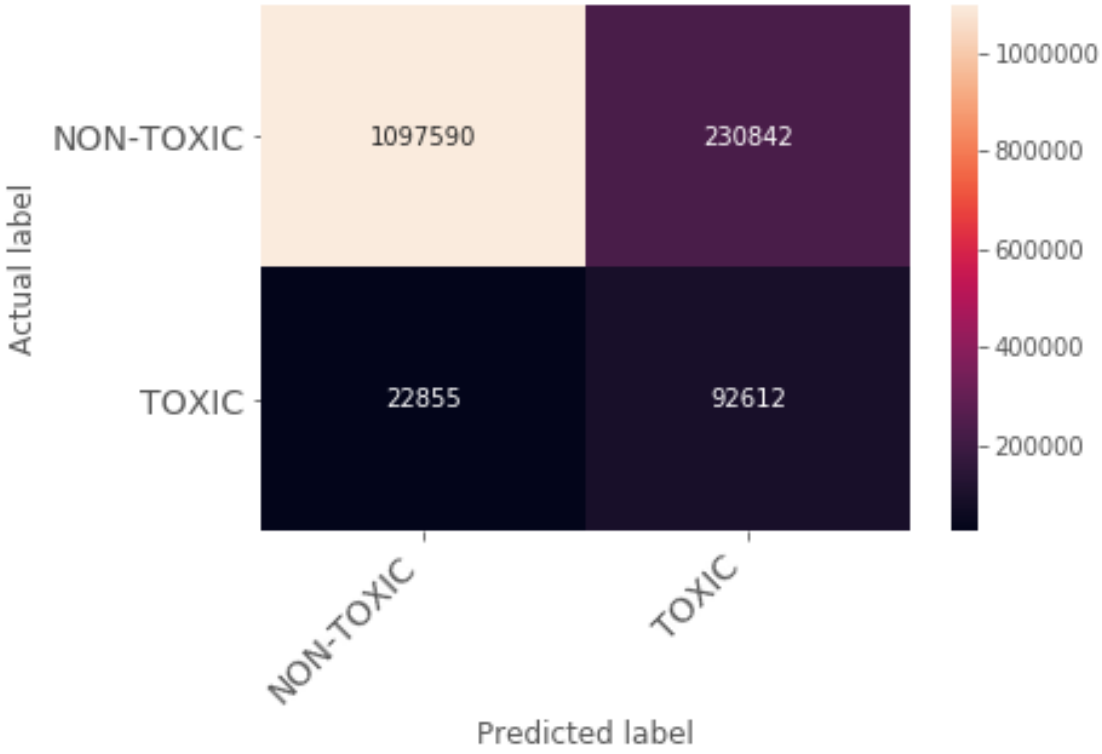
[83]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
 ↪predicted_validation)
```

```
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()
```

[84]:
```
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'])
```

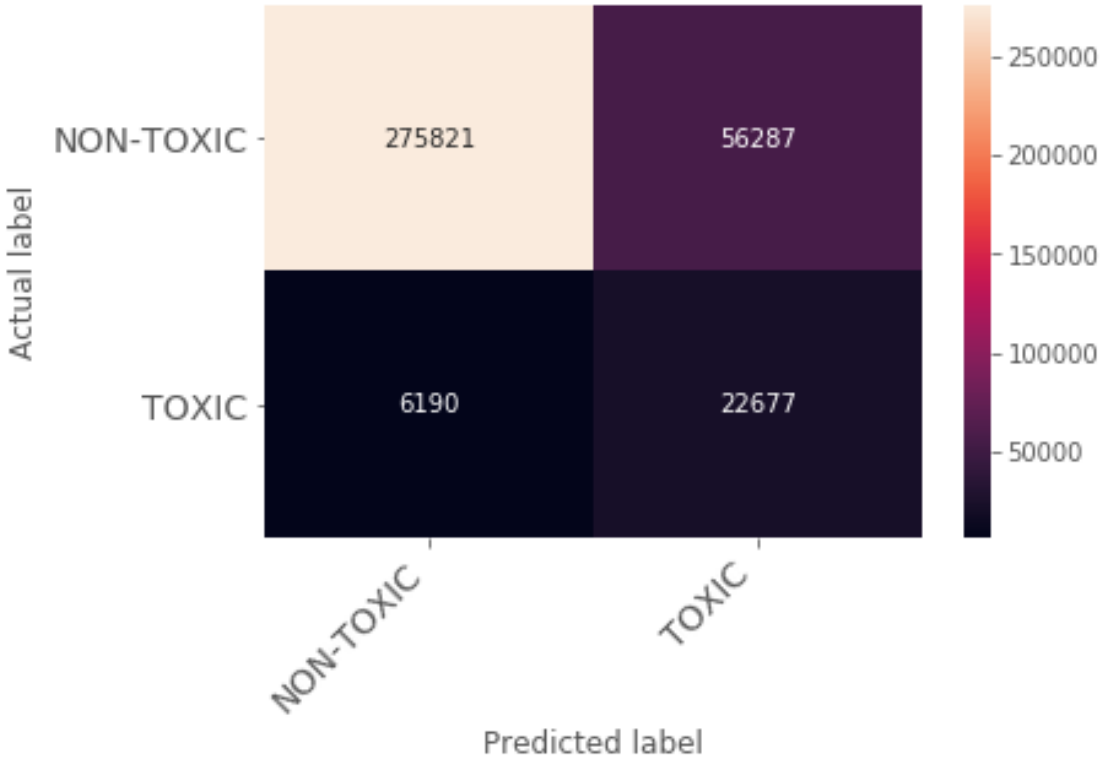
```
TRAIN DATA CONFUSION MATRIX
```



=> 82.52 % of non-toxic comments predicted correctly => 80.20% of toxic comments predicted correctly

[85]:
```
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
```



=> 83.05 % of non-toxic comments predicted correctly => 79.04% of toxic comments predicted correctly

5.1.2 Logistic Regression

Considering BOW

25000 features

[86]:
```
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 [02:13<00:00, 19.07s/it]
```

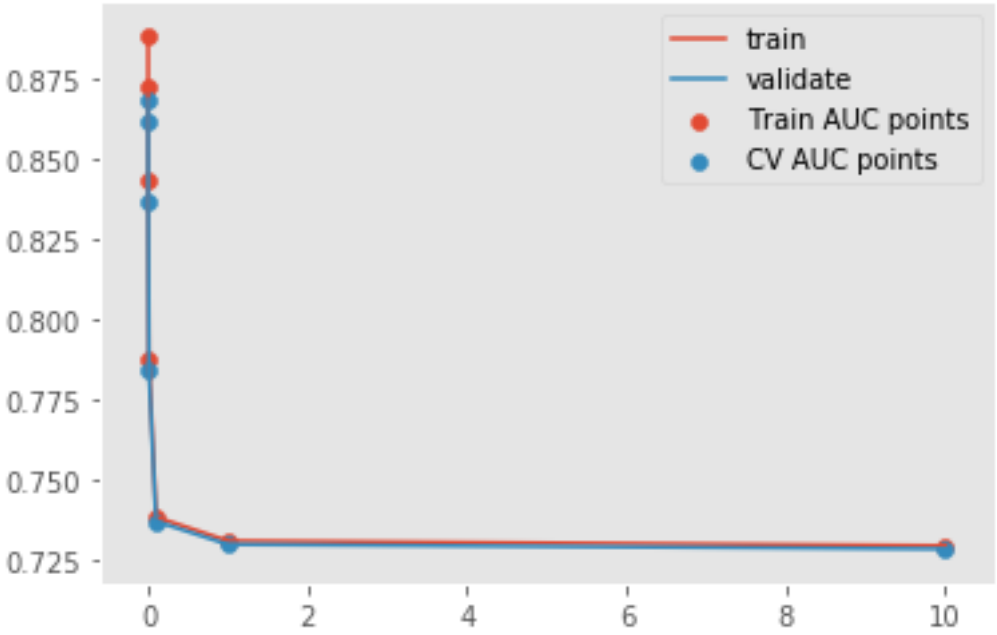
[87]:
```
score = pd.DataFrame({'name':names, 'train_score':train_auc_list, 'test_score':
 ↪validation_auc_list}).sort_values(by=['test_score'])
score
```

[87]:

	name	train_score	test_score
6	LR-BOW_25k_10	0.729246	0.728246
5	LR-BOW_25k_1	0.730751	0.729667
4	LR-BOW_25k_0.1	0.738181	0.736869
3	LR-BOW_25k_0.01	0.787471	0.784409
2	LR-BOW_25k_0.001	0.843153	0.836826
1	LR-BOW_25k_0.0001	0.872860	0.861781
0	LR-BOW_25k_1e-05	0.888376	0.868502

[88]:
```
print(train_auc_list,validation_auc_list)
print(f'best hyperparameter got = {score.name.values[-1]} ##### Best cv score␣
 ↪got = {score.test_score.values[-1]}')
plt.plot(alpha, train_auc_list, label='train')
plt.plot(alpha, validation_auc_list, label='validate')
plt.scatter(alpha, train_auc_list, label='Train AUC points')
plt.scatter(alpha, validation_auc_list, label='CV AUC points')
plt.legend()
plt.grid()
plt.show()
```

```
[0.8883764040261166, 0.8728596102026178, 0.8431525884311164, 0.78747086488579,
0.7381814282033289, 0.7307513181859522, 0.7292459221422967] [0.8685024682817755,
0.8617813205230369, 0.8368256005198109, 0.784409386454797, 0.7368686535128023,
0.7296670343536258, 0.7282463762307846]
best hyperparameter got = LR-BOW_25k_1e-05 ##### Best cv score got =
0.8685024682817755
```



[89]:
```
MODEL_NAME = 'LR_bow_25k'
clf = SGDClassifier(alpha=1e-05, class_weight='balanced', loss='log',␣
 ↪penalty='l2')
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]
```

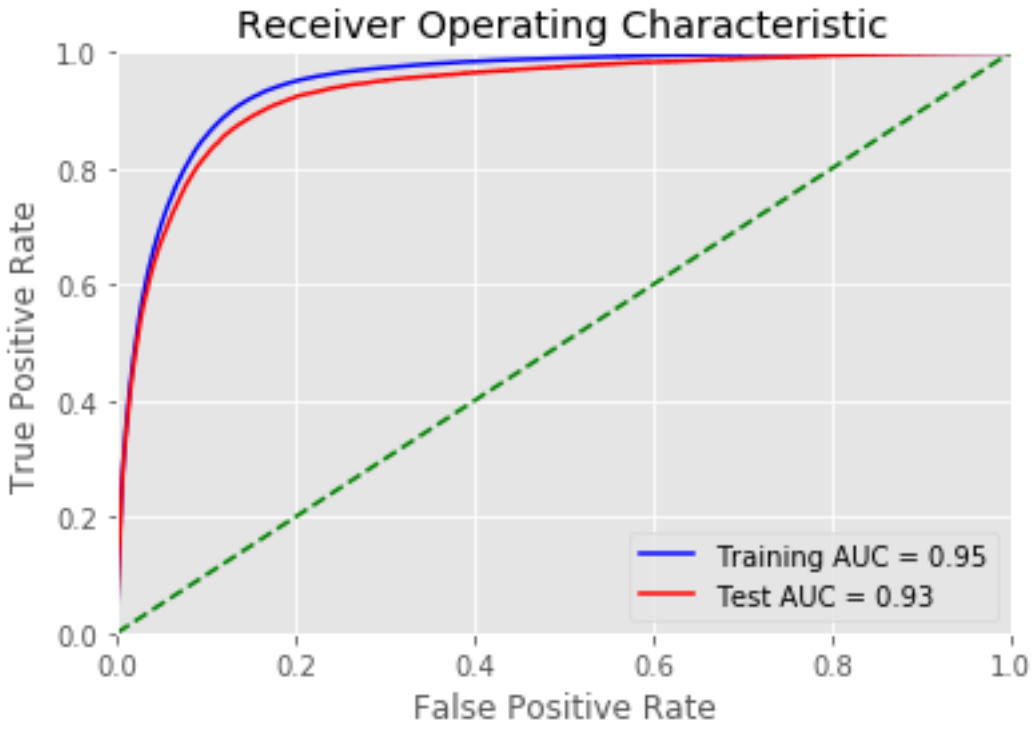
[90]:
```
train_data[MODEL_NAME] = predicted_train
validation_data[MODEL_NAME] = predicted_validation
print(f'Train score = {get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME)}')
print(f'Validation score = {get_metric_value(validation_data, identity_columns,␣
 ↪MODEL_NAME)}')
```

```
Train score = 0.8855870072660815
Validation score = 0.8658516271542865
```

[91]:
```
predicted_test = clf.predict_proba(test_comment_bow_25000)[:,1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/LR_bow_25k_submission.csv', index=False)
```

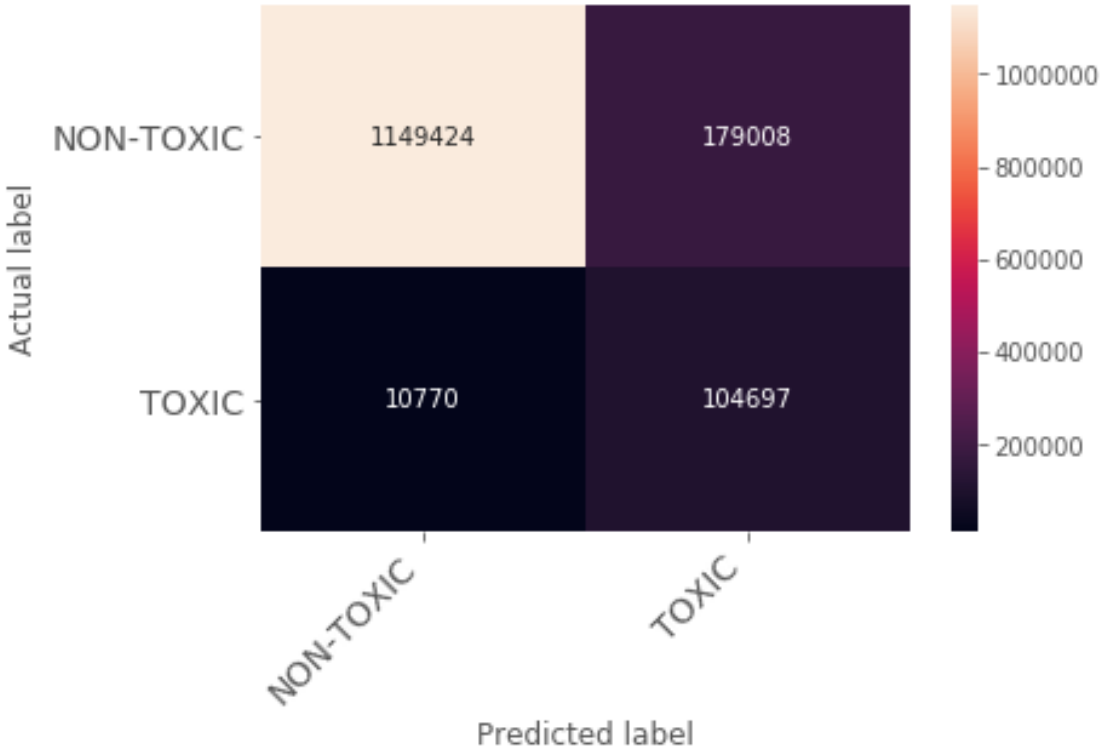
[92]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
fpr_test, tpr_test, threshold_test = roc_curve(y_validation,␣
 ↪predicted_validation)
```

```
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()
```



[93]:
```
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'])
```

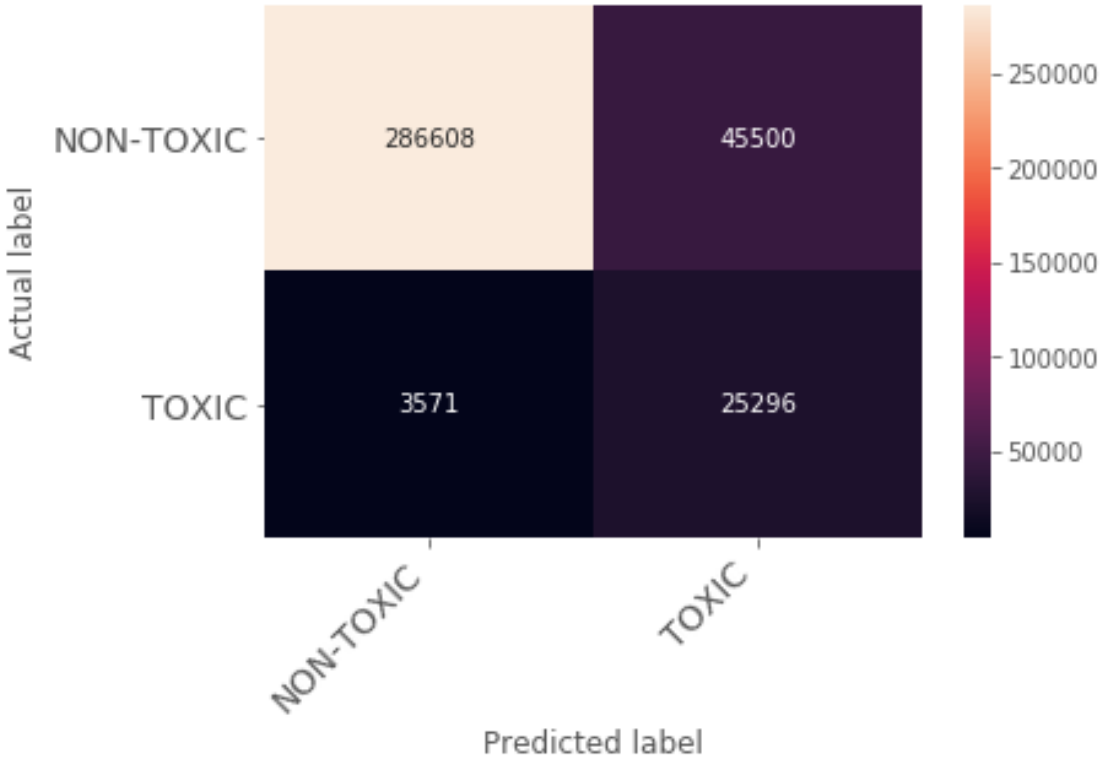
```
TRAIN DATA CONFUSION MATRIX
```



=> 86.52 % of non-toxic comments predicted correctly => 90.17% of toxic comments predicted correctly

[94]:
```
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
```



=> 86.29 % of non-toxic comments predicted correctly => 88.17% of toxic comments predicted correctly

15000 features

[95]:
```
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 [02:18<00:00, 19.83s/it]
```

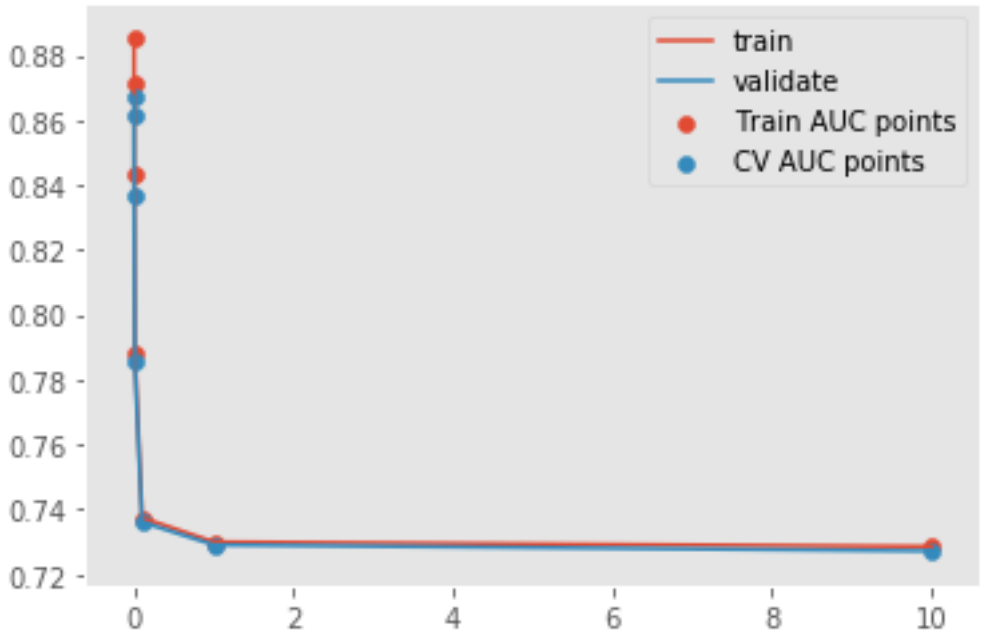
[96]:
```
score = pd.DataFrame({'name':names, 'train_score':train_auc_list, 'test_score':
↪validation_auc_list}).sort_values(by=['test_score'])
score
```

[96]:

	name	train_score	test_score
6	LR-BOW_15k_10	0.728575	0.727211
5	LR-BOW_15k_1	0.729960	0.729090
4	LR-BOW_15k_0.1	0.737433	0.736097
3	LR-BOW_15k_0.01	0.788353	0.785380
2	LR-BOW_15k_0.001	0.842992	0.836769
1	LR-BOW_15k_0.0001	0.871725	0.861307
0	LR-BOW_15k_1e-05	0.885256	0.867154

[97]:
```
print(train_auc_list,validation_auc_list)
print(f'best hyperparameter got = {score.name.values[-1]} ##### Best cv score␣
↪got = {score.test_score.values[-1]}')
plt.plot(alpha, train_auc_list, label='train')
plt.plot(alpha, validation_auc_list, label='validate')
plt.scatter(alpha, train_auc_list, label='Train AUC points')
plt.scatter(alpha, validation_auc_list, label='CV AUC points')
plt.legend()
plt.grid()
plt.show()
```

```
[0.8852563636754947, 0.8717246904498954, 0.8429924070403463, 0.7883527587420291,
0.7374330897002882, 0.7299600382169944, 0.7285752730332522] [0.867154022484939,
0.8613073302574822, 0.8367693709621655, 0.7853799758740808, 0.736096702505924,
0.7290902929699002, 0.7272107612840208]
best hyperparameter got = LR-BOW_15k_1e-05 ##### Best cv score got =
0.867154022484939
```



[98]:
```
MODEL_NAME = 'LR_bow_15k'
clf = SGDClassifier(alpha=1e-05, class_weight='balanced', loss='log',␣
 ↪penalty='l2')
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]
```

[99]:
```
train_data[MODEL_NAME] = predicted_train
validation_data[MODEL_NAME] = predicted_validation
print(f'Train score = {get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME)}')
print(f'Validation score = {get_metric_value(validation_data, identity_columns,␣
 ↪MODEL_NAME)}')
```

```
Train score = 0.884694041115146
Validation score = 0.8668547299807178
```

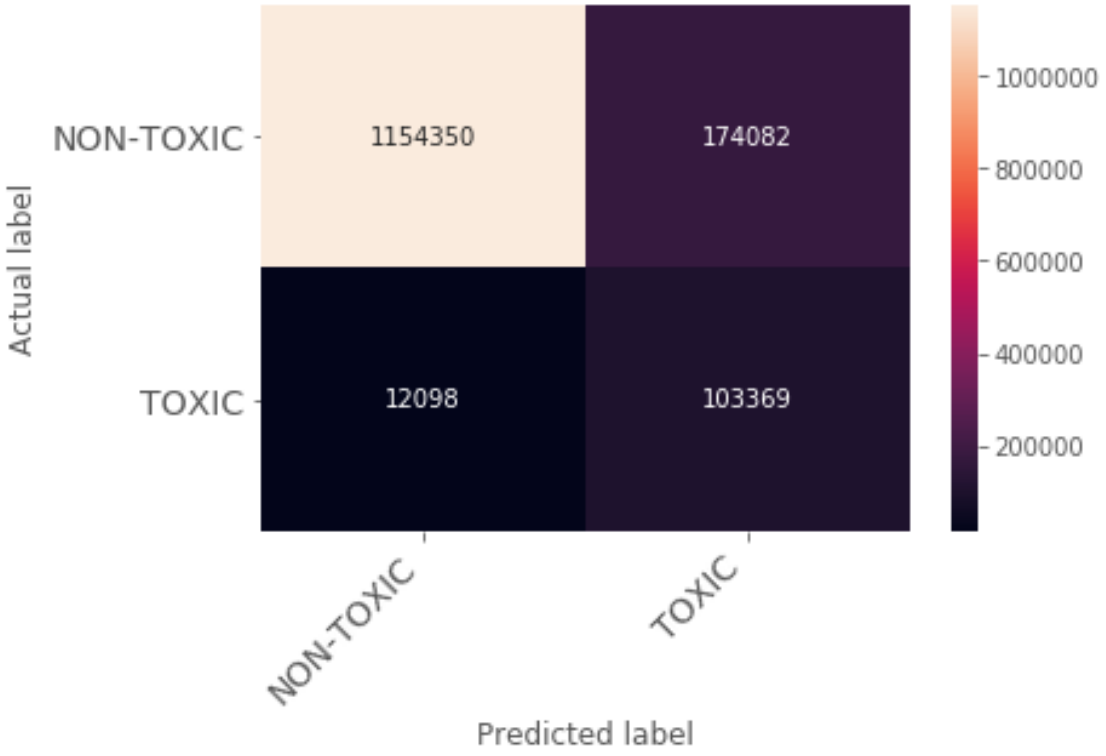
[100]:
```
predicted_test = clf.predict_proba(test_comment_bow_15000)[:,1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/LR_bow_15k_submission.csv', index=False)
```

[101]:
```
# 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()
```

[102]:
```
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'])
```

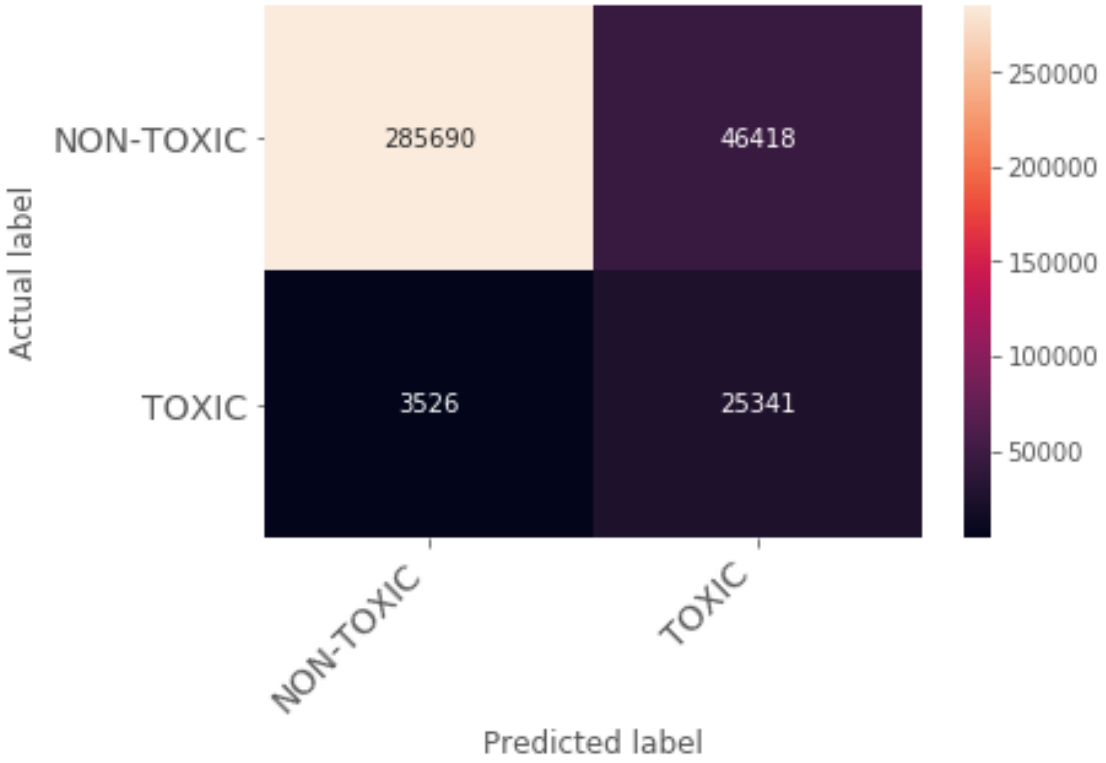
```
TRAIN DATA CONFUSION MATRIX
```



=> 86.89 % of non-toxic comments predicted correctly => 89.52% of toxic comments predicted correctly

[103]:
```
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
```



=> 86.02 % of non-toxic comments predicted correctly => 88.33% of toxic comments predicted correctly

10000 features

[104]:
```
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 [02:22<00:00, 20.33s/it]
```

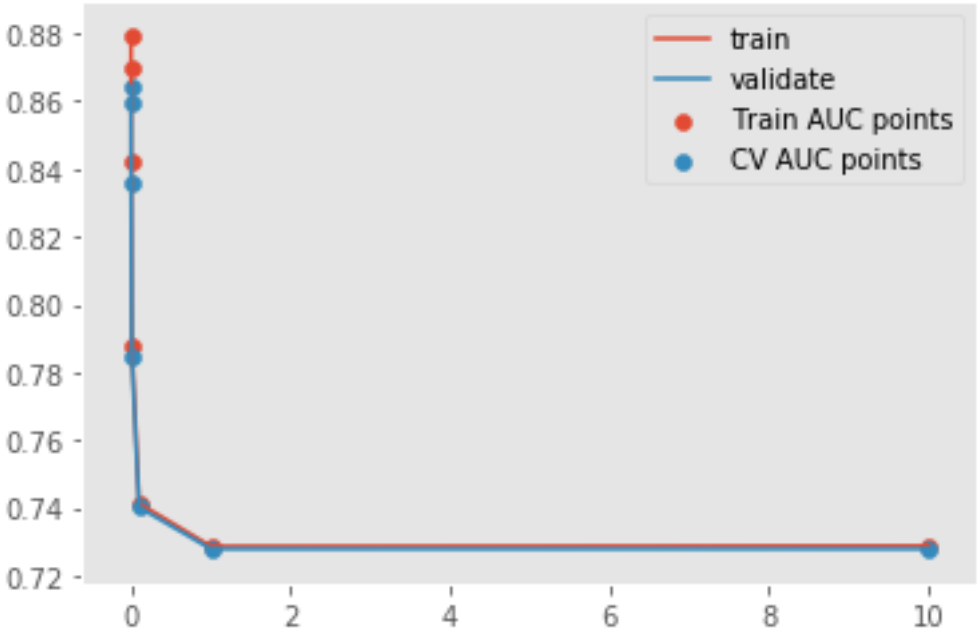
[105]:
```
score = pd.DataFrame({'name':names, 'train_score':train_auc_list, 'test_score':
↪validation_auc_list}).sort_values(by=['test_score'])
score
```

[105]:

	name	train_score	test_score
5	LR-BOW_10k_1	0.728729	0.727951
6	LR-BOW_10k_10	0.728852	0.727953
4	LR-BOW_10k_0.1	0.741374	0.740229
3	LR-BOW_10k_0.01	0.787858	0.784922
2	LR-BOW_10k_0.001	0.841920	0.835783
1	LR-BOW_10k_0.0001	0.869799	0.859878
0	LR-BOW_10k_1e-05	0.879191	0.864436

[106]:
```
print(train_auc_list,validation_auc_list)
print(f'best hyperparameter got = {score.name.values[-1]} ##### Best cv score␣
↪got = {score.test_score.values[-1]}')
plt.plot(alpha, train_auc_list, label='train')
plt.plot(alpha, validation_auc_list, label='validate')
plt.scatter(alpha, train_auc_list, label='Train AUC points')
plt.scatter(alpha, validation_auc_list, label='CV AUC points')
plt.legend()
plt.grid()
plt.show()
```

```
[0.8791914041561264, 0.8697988757289951, 0.8419202161656001, 0.7878577939714233,
0.7413741273602321, 0.7287285347560886, 0.7288519182154476] [0.8644363381705658,
0.8598782548103915, 0.8357830918384859, 0.7849215563767598, 0.7402294893350438,
0.7279511226767879, 0.7279532162599258]
best hyperparameter got = LR-BOW_10k_1e-05 ##### Best cv score got =
0.8644363381705658
```



[107]:
```
MODEL_NAME = 'LR_bow_10k'
clf = SGDClassifier(alpha=1e-05, class_weight='balanced', loss='log',␣
 ↪penalty='l2')
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]
```

[108]:
```
train_data[MODEL_NAME] = predicted_train
validation_data[MODEL_NAME] = predicted_validation
print(f'Train score = {get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME)}')
print(f'Validation score = {get_metric_value(validation_data, identity_columns,␣
 ↪MODEL_NAME)}')
```

```
Train score = 0.8751021308860036
Validation score = 0.859595458254724
```

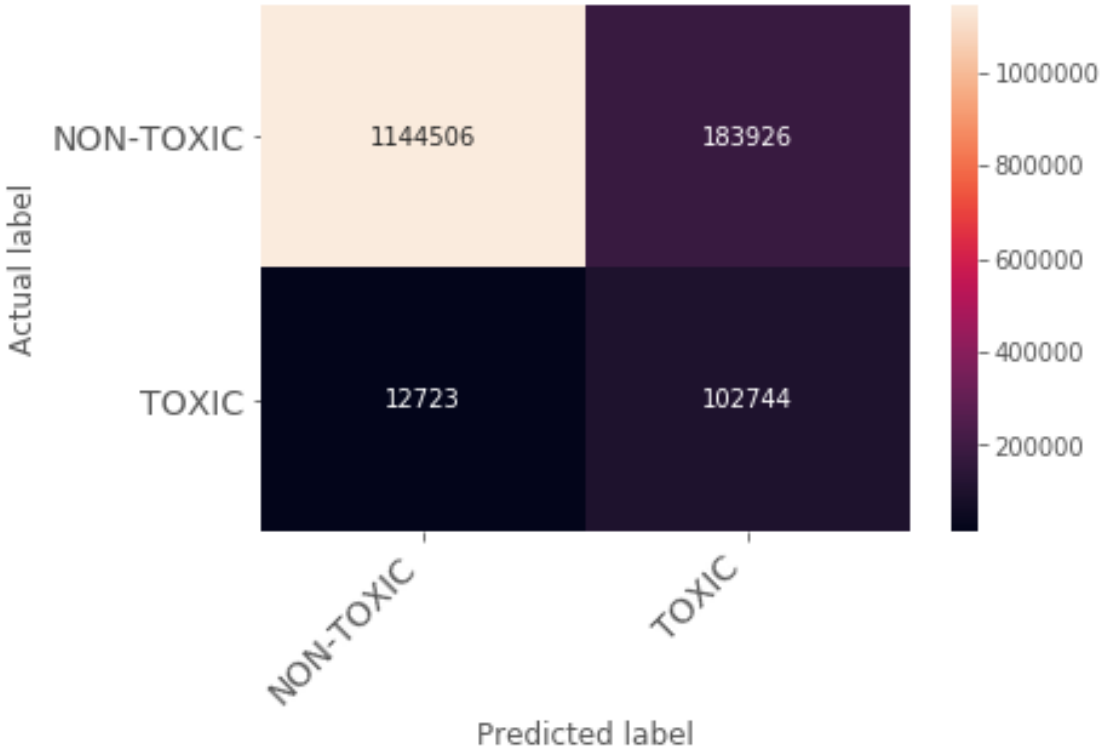
[109]:
```
predicted_test = clf.predict_proba(test_comment_bow_10000)[:,1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/LR_bow_10k_submission.csv', index=False)
```

[110]:
```
# 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()
```

[111]:
```
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'])
```

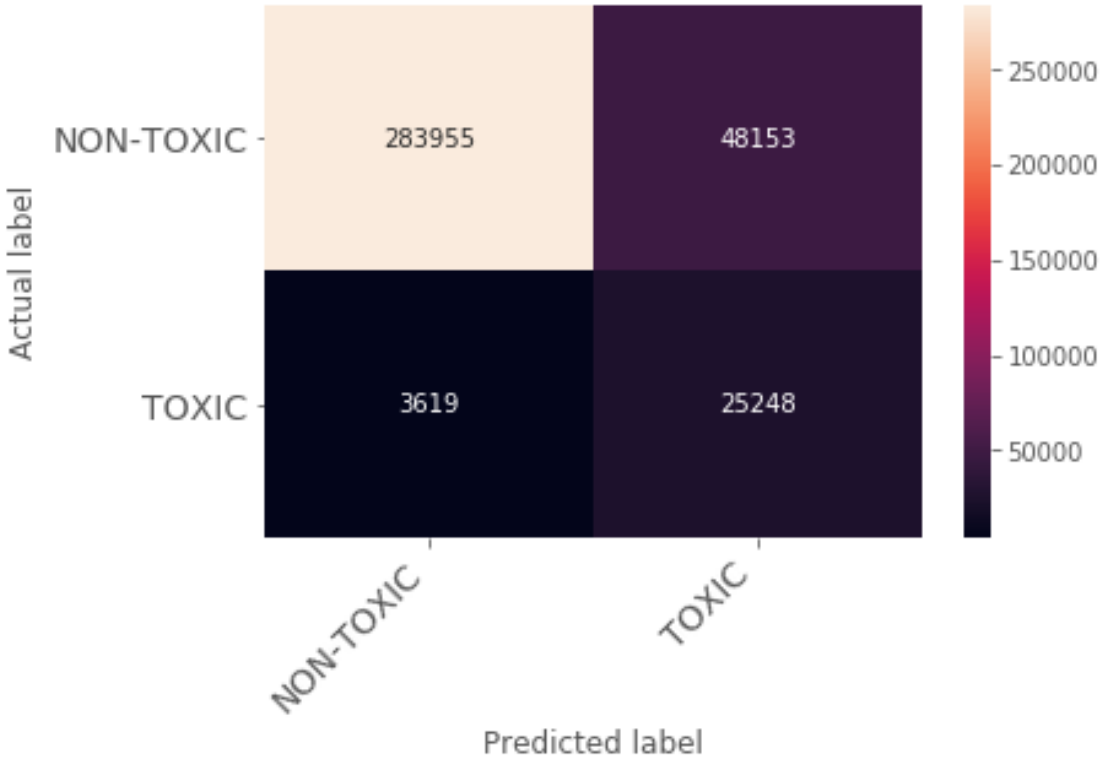
```
TRAIN DATA CONFUSION MATRIX
```



=> 86.15 % of non-toxic comments predicted correctly => 88.98% of toxic comments predicted correctly

[112]:
```
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
```



=> 85.50 % of non-toxic comments predicted correctly => 88.01% of toxic comments predicted correctly

Considering TFIDF

25000 features

[113]:
```
import gc
gc.collect()
```

[113]: 26411

[114]:
```
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 [02:17<00:00, 19.64s/it]
```

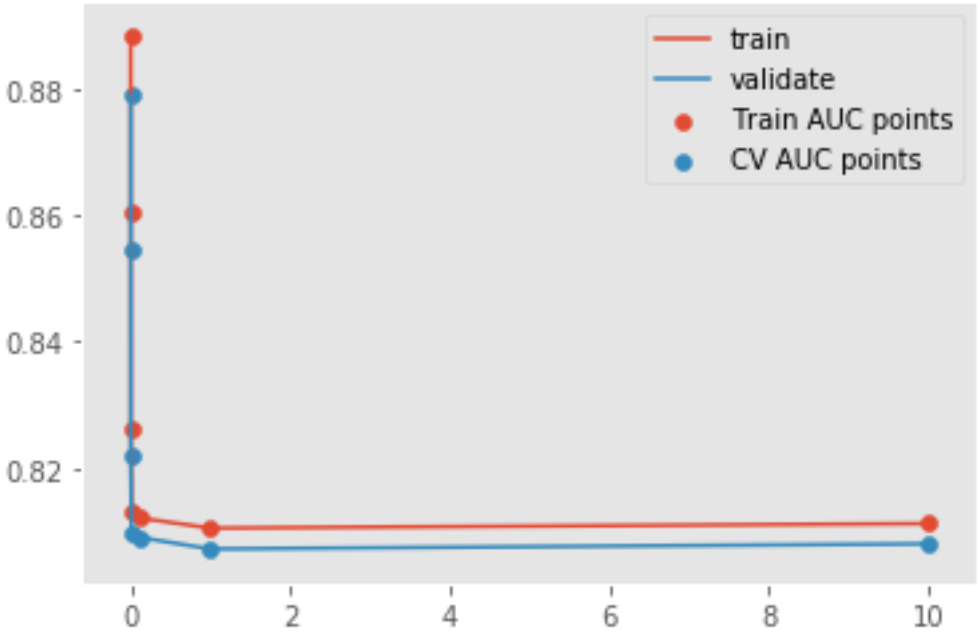
[115]:
```
score = pd.DataFrame({'name':names, 'train_score':train_auc_list, 'test_score':
 ↪validation_auc_list}).sort_values(by=['test_score'])
score
```

[115]:

	name	train_score	test_score
5	LR-tfidf_25k_1	0.810516	0.807215
6	LR-tfidf_25k_10	0.811231	0.808033
4	LR-tfidf_25k_0.1	0.812138	0.809069
3	LR-tfidf_25k_0.01	0.812970	0.809628
2	LR-tfidf_25k_0.001	0.826055	0.822004
1	LR-tfidf_25k_0.0001	0.860375	0.854431
0	LR-tfidf_25k_1e-05	0.888359	0.879074

[116]:
```
print(train_auc_list,validation_auc_list)
print(f'best hyperparameter got = {score.name.values[-1]} ##### Best cv score␣
 ↪got = {score.test_score.values[-1]}')
plt.plot(alpha, train_auc_list, label='train')
plt.plot(alpha, validation_auc_list, label='validate')
plt.scatter(alpha, train_auc_list, label='Train AUC points')
plt.scatter(alpha, validation_auc_list, label='CV AUC points')
plt.legend()
plt.grid()
plt.show()
```

```
[0.8883589444562898, 0.8603753586108874, 0.826055031353244, 0.8129701080826565,
0.8121382918349923, 0.8105158008134189, 0.8112311832421755] [0.8790736879001076,
0.8544308878174316, 0.8220039261864243, 0.8096283463665845, 0.8090685059079621,
0.8072150818648656, 0.8080333922373485]
best hyperparameter got = LR-tfidf_25k_1e-05 ##### Best cv score got =
0.8790736879001076
```



[117]:
```
MODEL_NAME = 'LR_tfidf_25k'
clf = SGDClassifier(alpha=1e-05, class_weight='balanced', loss='log',␣
 ↪penalty='l2')
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]
```

[118]:
```
train_data[MODEL_NAME] = predicted_train
validation_data[MODEL_NAME] = predicted_validation
print(f'Train score = {get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME)}')
print(f'Validation score = {get_metric_value(validation_data, identity_columns,␣
 ↪MODEL_NAME)}')
```

```
Train score = 0.8884860097510427
Validation score = 0.8792506623862287
```

[119]:
```
predicted_test = clf.predict_proba(test_comment_tfidf_25000)[:,1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/LR_tfidf_25k_submission.csv', index=False)
```

[ ]:
```

```

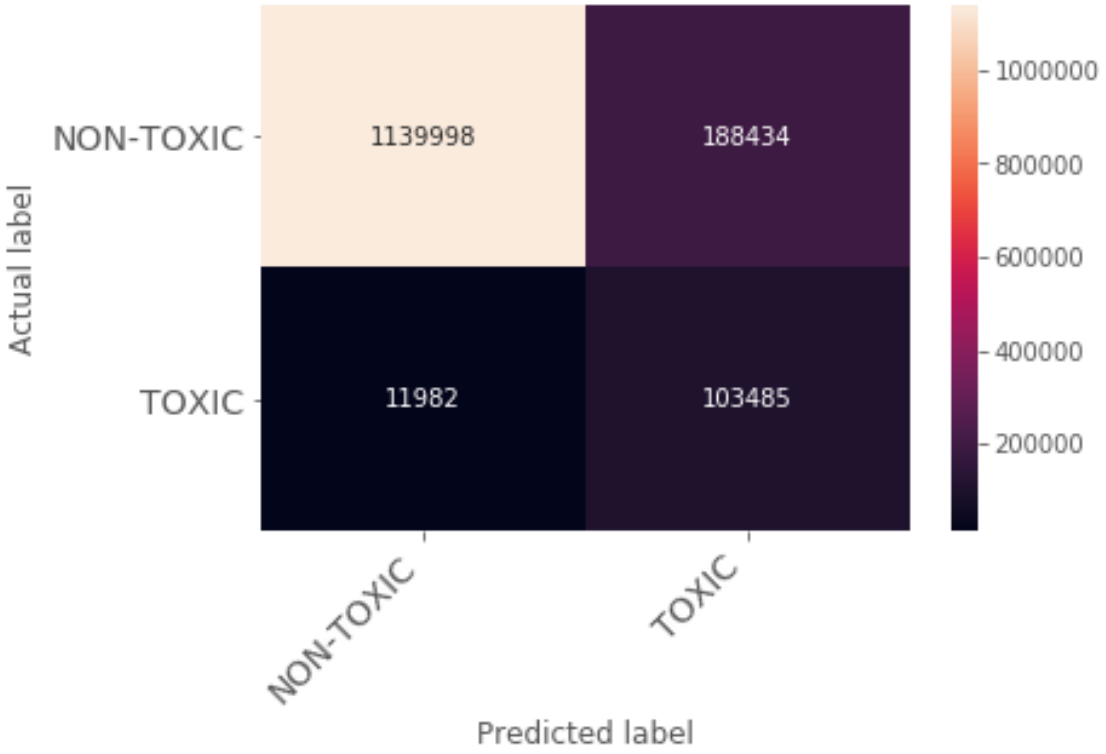
[120]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
```

```
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()
```

[121]:
```
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'])
```

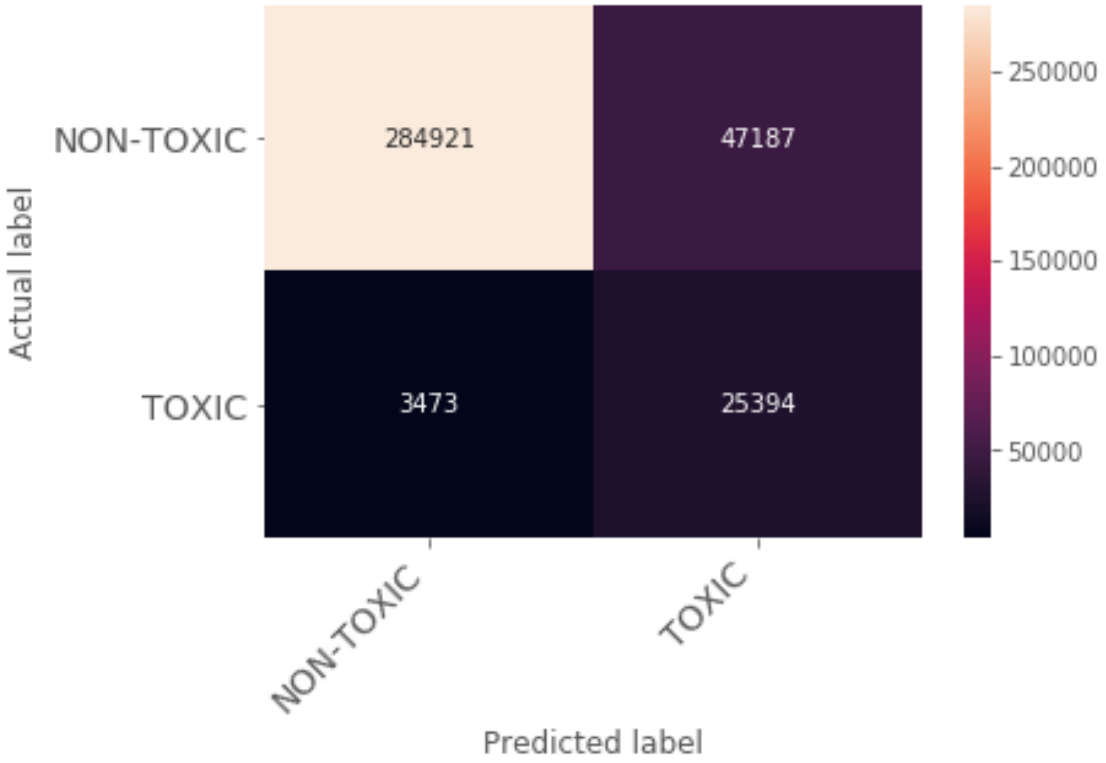
```
        TRAIN DATA CONFUSION MATRIX
```



=> 85.81 % of non-toxic comments predicted correctly => 89.62% of toxic comments predicted correctly

[122]:
```
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
```



=> 85.79 % of non-toxic comments predicted correctly => 88.52% of toxic comments predicted correctly

15000 features

[123]:
```
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 [02:22<00:00, 20.42s/it]
```

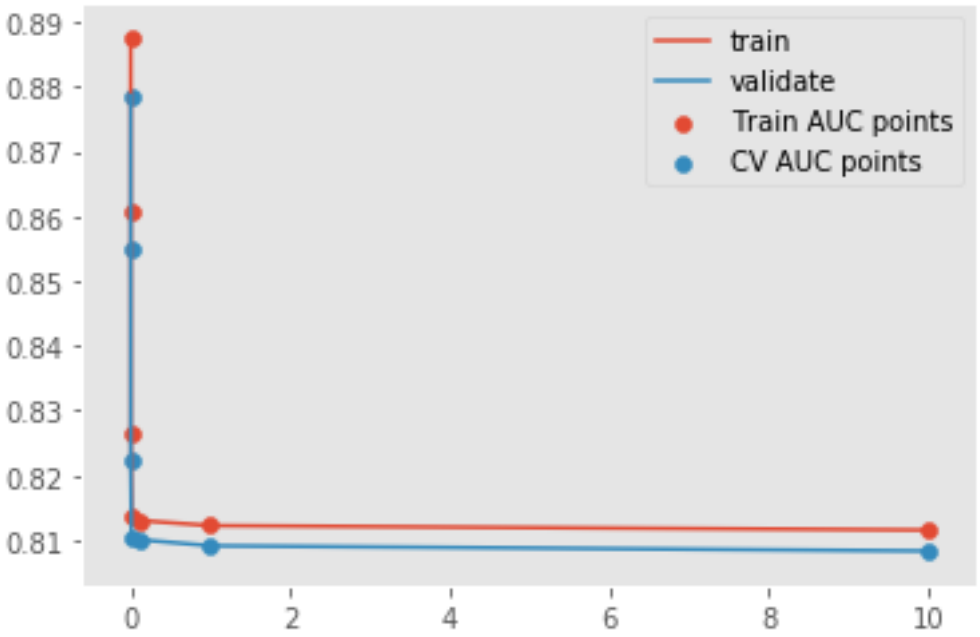
[124]:
```
score = pd.DataFrame({'name':names, 'train_score':train_auc_list, 'test_score':
↪validation_auc_list}).sort_values(by=['test_score'])
score
```

[124]:

	name	train_score	test_score
6	LR-tfidf_15k_10	0.811563	0.808329
5	LR-tfidf_15k_1	0.812242	0.809136
4	LR-tfidf_15k_0.1	0.813041	0.810073
3	LR-tfidf_15k_0.01	0.813771	0.810451
2	LR-tfidf_15k_0.001	0.826632	0.822499
1	LR-tfidf_15k_0.0001	0.860811	0.854865
0	LR-tfidf_15k_1e-05	0.887478	0.878648

[125]:
```
print(train_auc_list,validation_auc_list)
print(f'best hyperparameter got = {score.name.values[-1]} ##### Best cv score␣
↪got = {score.test_score.values[-1]}')
plt.plot(alpha, train_auc_list, label='train')
plt.plot(alpha, validation_auc_list, label='validate')
plt.scatter(alpha, train_auc_list, label='Train AUC points')
plt.scatter(alpha, validation_auc_list, label='CV AUC points')
plt.legend()
plt.grid()
plt.show()
```

```
[0.8874782707651374, 0.8608110760758877, 0.8266316949731487, 0.8137705074207171,
0.8130409528654141, 0.8122418726520166, 0.811562758955933] [0.878648271939579,
0.8548654582687194, 0.8224987463709735, 0.8104508401330929, 0.81007337461985,
0.8091362953089876, 0.8083285875956953]
best hyperparameter got = LR-tfidf_15k_1e-05 ##### Best cv score got =
0.878648271939579
```



[126]:
```
MODEL_NAME = 'LR_tfidf_15k'
clf = SGDClassifier(alpha=1e-05, class_weight='balanced', loss='log',␣
 ↪penalty='l2')
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]
```

[127]:
```
train_data[MODEL_NAME] = predicted_train
validation_data[MODEL_NAME] = predicted_validation
print(f'Train score = {get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME)}')
print(f'Validation score = {get_metric_value(validation_data, identity_columns,␣
 ↪MODEL_NAME)}')
```

```
Train score = 0.8872380229794584
Validation score = 0.878449056596368
```

[128]:
```
predicted_test = clf.predict_proba(test_comment_tfidf_15000)[:,1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/LR_tfidf_15k_submission.csv', index=False)
```

[ ]:
```

```

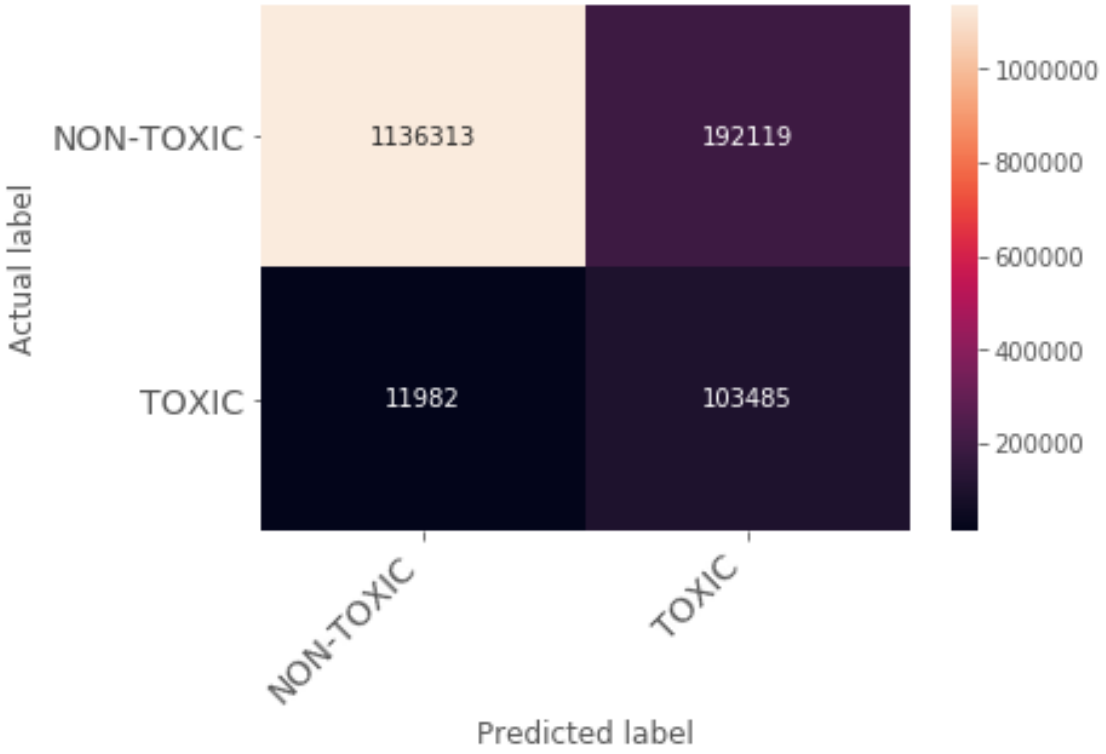
[129]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
```

```
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'])
```

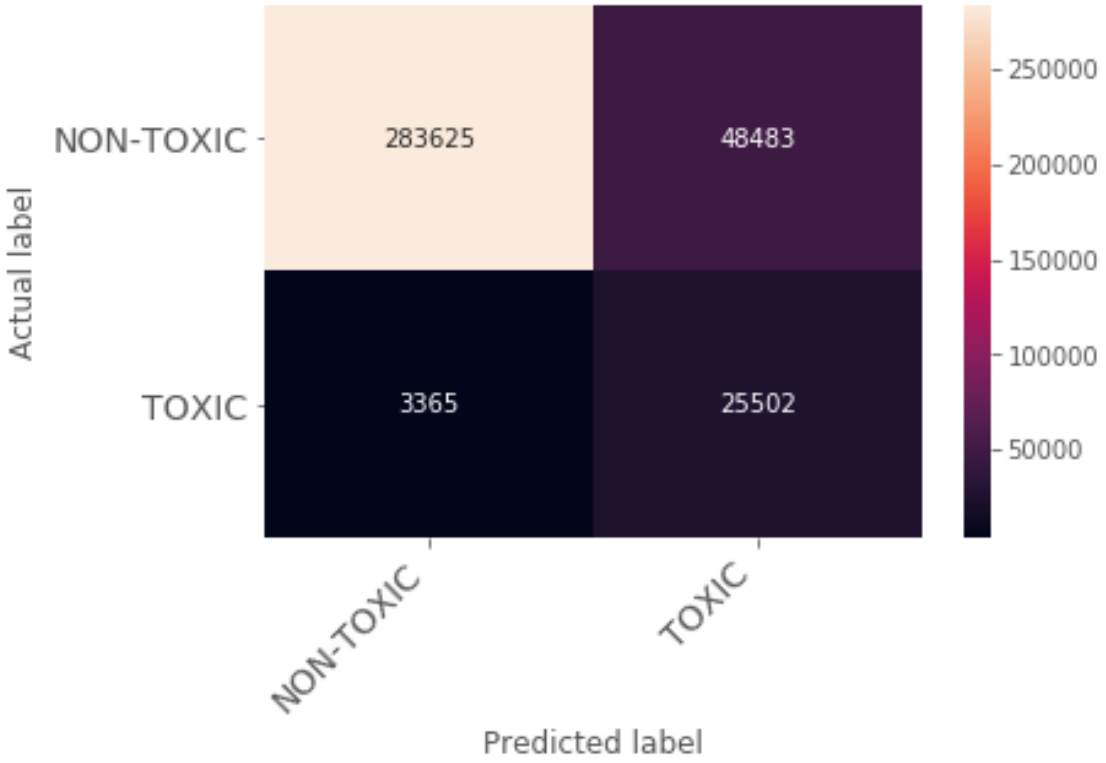
```
        TRAIN DATA CONFUSION MATRIX
```



=> 85.53 % of non-toxic comments predicted correctly => 89.62% of toxic comments predicted correctly

[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
```



=> 85.40 % of non-toxic comments predicted correctly => 88.89% of toxic comments predicted correctly

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-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 [02:26<00:00, 20.93s/it]
```

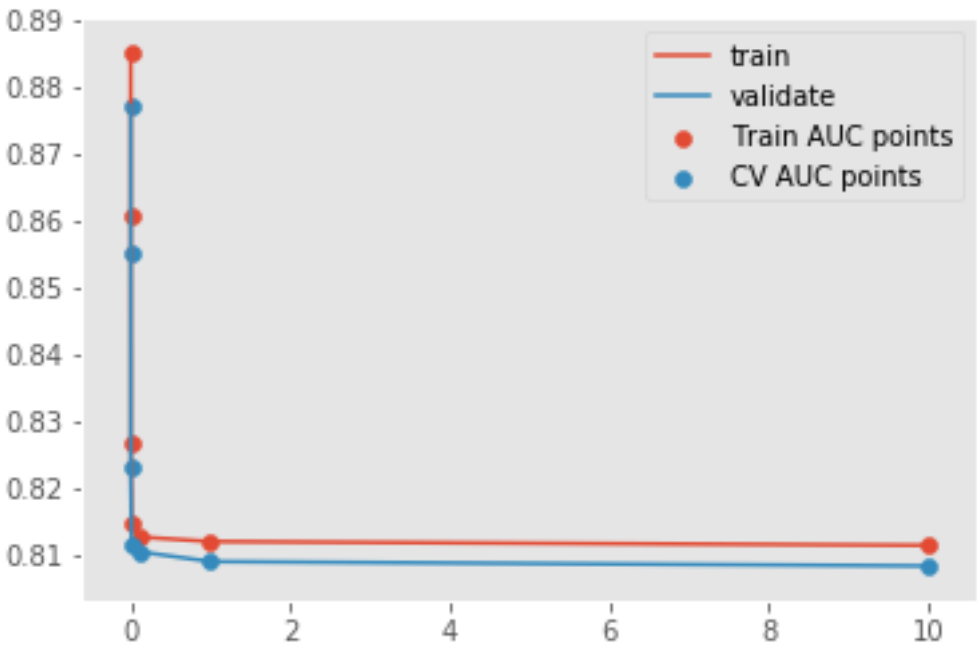
[133]:
```
score = pd.DataFrame({'name':names, 'train_score':train_auc_list, 'test_score':
↪validation_auc_list}).sort_values(by=['test_score'])
score
```

[133]:

	name	train_score	test_score
6	LR-tfidf_10k_10	0.811468	0.808384
5	LR-tfidf_10k_1	0.812026	0.809038
4	LR-tfidf_10k_0.1	0.812718	0.810515
3	LR-tfidf_10k_0.01	0.814661	0.811691
2	LR-tfidf_10k_0.001	0.826914	0.822966
1	LR-tfidf_10k_0.0001	0.860607	0.855021
0	LR-tfidf_10k_1e-05	0.885048	0.877142

[134]:
```
print(train_auc_list,validation_auc_list)
print(f'best hyperparameter got = {score.name.values[-1]} ##### Best cv score␣
↪got = {score.test_score.values[-1]}')
plt.plot(alpha, train_auc_list, label='train')
plt.plot(alpha, validation_auc_list, label='validate')
plt.scatter(alpha, train_auc_list, label='Train AUC points')
plt.scatter(alpha, validation_auc_list, label='CV AUC points')
plt.legend()
plt.grid()
plt.show()
```

```
[0.8850477949471514, 0.860606632317165, 0.8269141138245488, 0.8146609570366554,
0.8127180876243518, 0.8120258043083588, 0.8114678002992664] [0.8771421110683988,
0.8550209766014671, 0.8229664609181242, 0.8116905364534729, 0.8105154686961777,
0.8090381136644589, 0.8083837093406189]
best hyperparameter got = LR-tfidf_10k_1e-05 ##### Best cv score got =
0.8771421110683988
```



[135]:
```
MODEL_NAME = 'LR_tfidf_10k'
clf = SGDClassifier(alpha=1e-05, class_weight='balanced', loss='log',␣
 ↪penalty='l2')
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]
```

[136]:
```
train_data[MODEL_NAME] = predicted_train
validation_data[MODEL_NAME] = predicted_validation
print(f'Train score = {get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME)}')
print(f'Validation score = {get_metric_value(validation_data, identity_columns,␣
 ↪MODEL_NAME)}')
```

```
Train score = 0.8848560381994712
Validation score = 0.8768401612236583
```

[137]:
```
predicted_test = clf.predict_proba(test_comment_tfidf_10000)[:,1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/LR_tfidf_10k_submission.csv', index=False)
```

[ ]:
```

```

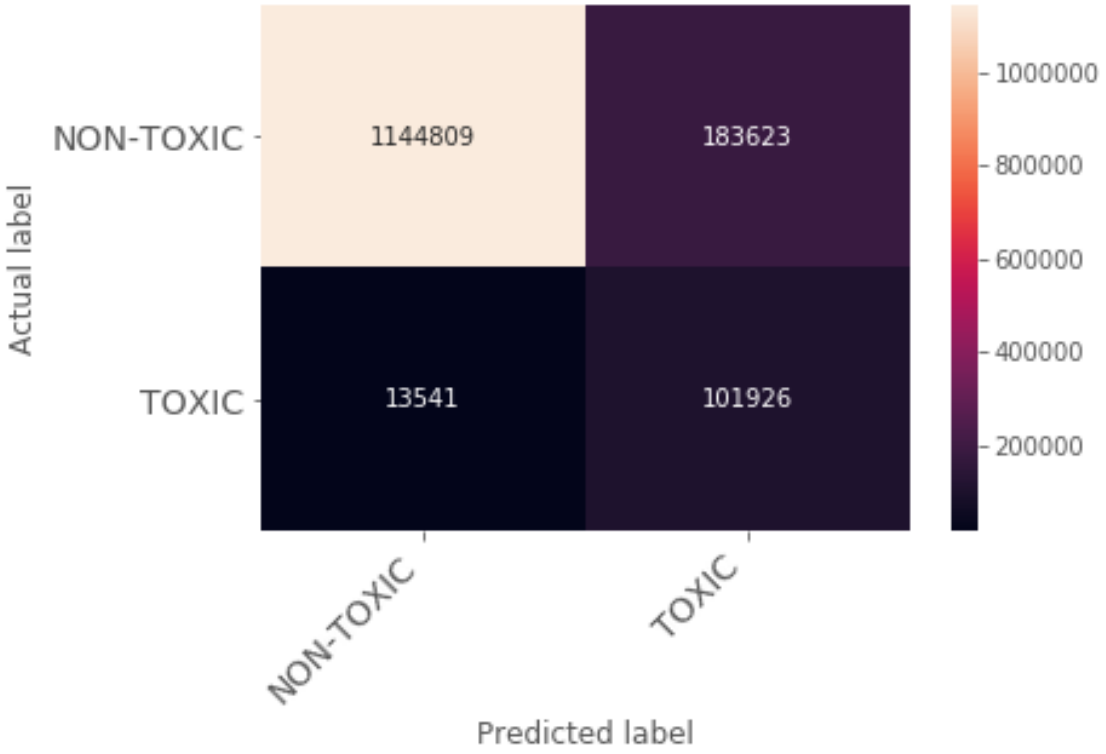
[138]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr_train, tpr_train, threshold_train = roc_curve(y_train, predicted_train)
```

```
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()
```

[139]:
```
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'])
```

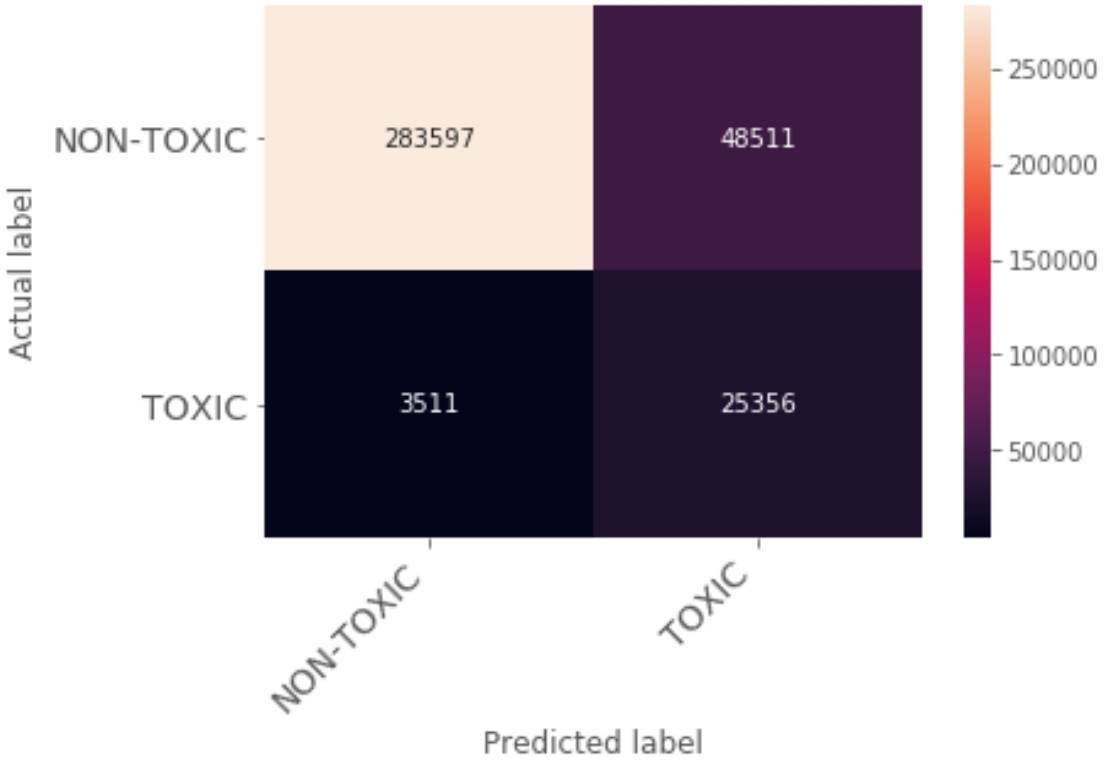
```
        TRAIN DATA CONFUSION MATRIX
```



=> 86.17 % of non-toxic comments predicted correctly => 88.27% of toxic comments predicted correctly

[140]:
```
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
```



=> 85.39 % of non-toxic comments predicted correctly => 88.38% of toxic comments predicted correctly

[141]:
```
gc.collect()
```

[141]: 39299

5.1.3 SVM

Considering TFIDF

25000 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'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 [04:42<00:00, 40.36s/it]
```

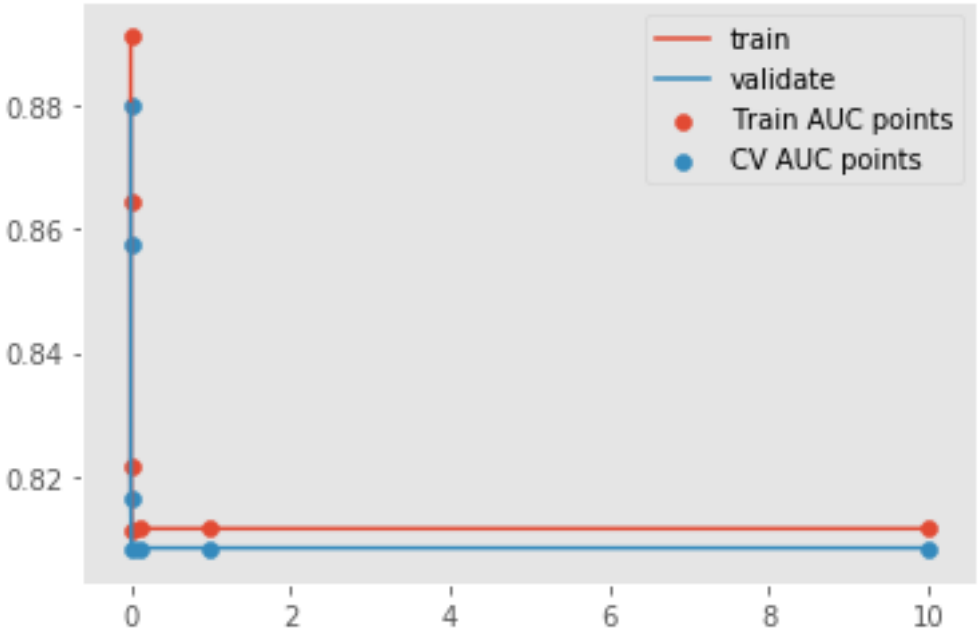
[143]:
```
score = pd.DataFrame({'name':names, 'train_score':train_auc_list, 'test_score':
↪validation_auc_list}).sort_values(by=['test_score'])
score
```

[143]:

	name	train_score	test_score
3	svm-tfidf_25k_0.01	0.811327	0.808182
4	svm-tfidf_25k_0.1	0.811490	0.808345
5	svm-tfidf_25k_1	0.811490	0.808345
6	svm-tfidf_25k_10	0.811490	0.808345
2	svm-tfidf_25k_0.001	0.821717	0.816406
1	svm-tfidf_25k_0.0001	0.864645	0.857521
0	svm-tfidf_25k_1e-05	0.891313	0.880202

[144]:
```
print(train_auc_list,validation_auc_list)
print(f'best hyperparameter got = {score.name.values[-1]} ##### Best cv score␣
↪got = {score.test_score.values[-1]}')
plt.plot(alpha, train_auc_list, label='train')
plt.plot(alpha, validation_auc_list, label='validate')
plt.scatter(alpha, train_auc_list, label='Train AUC points')
plt.scatter(alpha, validation_auc_list, label='CV AUC points')
plt.legend()
plt.grid()
plt.show()
```

```
[0.8913132106095535, 0.864644777765612, 0.8217171085751638, 0.8113269134720448,
0.8114898882220797, 0.8114898881974566, 0.8114898882237095] [0.880202263518974,
0.857520541444115, 0.8164061631885171, 0.8081815897401121, 0.8083445796631972,
0.8083445796631972, 0.8083445796631972]
best hyperparameter got = svm-tfidf_25k_1e-05 ##### Best cv score got =
0.880202263518974
```



[145]:
```
MODEL_NAME = 'svm_tfidf_25k'
clf = SGDClassifier(alpha=1e-05, 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]
```

[146]:
```
train_data[MODEL_NAME] = predicted_train
validation_data[MODEL_NAME] = predicted_validation
print(f'Train score = {get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME)}')
print(f'Validation score = {get_metric_value(validation_data, identity_columns,␣
 ↪MODEL_NAME)}')
```

```
Train score = 0.8913848127985691
Validation score = 0.8801859160326639
```

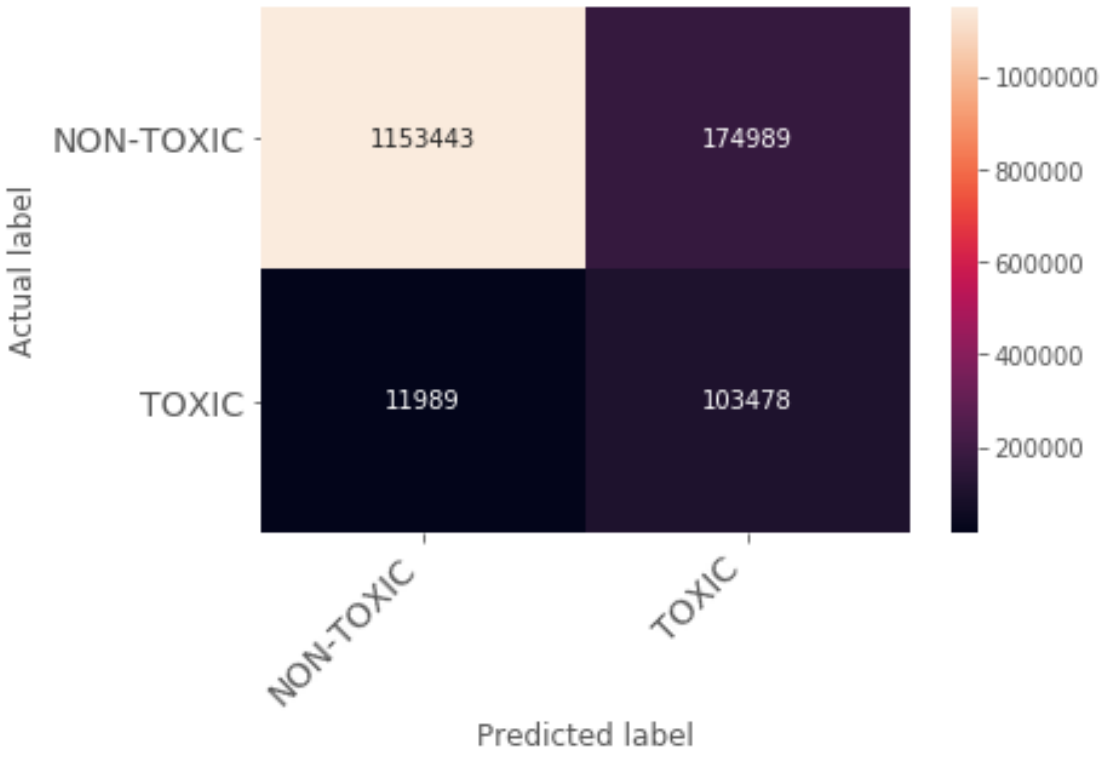
[147]:
```
predicted_test = clf.predict_proba(test_comment_tfidf_25000)[:,1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/svm_tfidf_25k_submission.csv', index=False)
```

[ ]:
```

```

[148]:
```
# 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()
```

[149]:
```
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'])
```

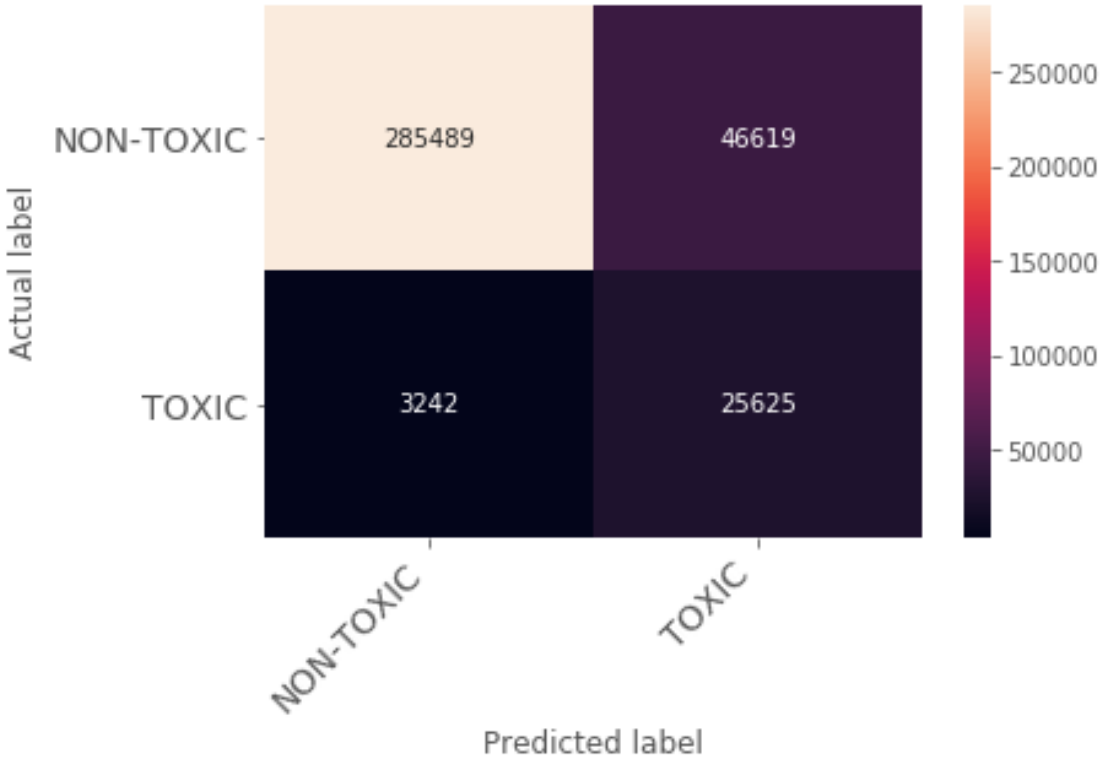
```
        TRAIN DATA CONFUSION MATRIX
```



=> 86.82 % of non-toxic comments predicted correctly => 89.61% of toxic comments predicted correctly

[150]:
```
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
```



=> 85.96 % of non-toxic comments predicted correctly => 89.32% of toxic comments predicted correctly

15000 features

[151]:
```
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 [04:38<00:00, 39.75s/it]
```

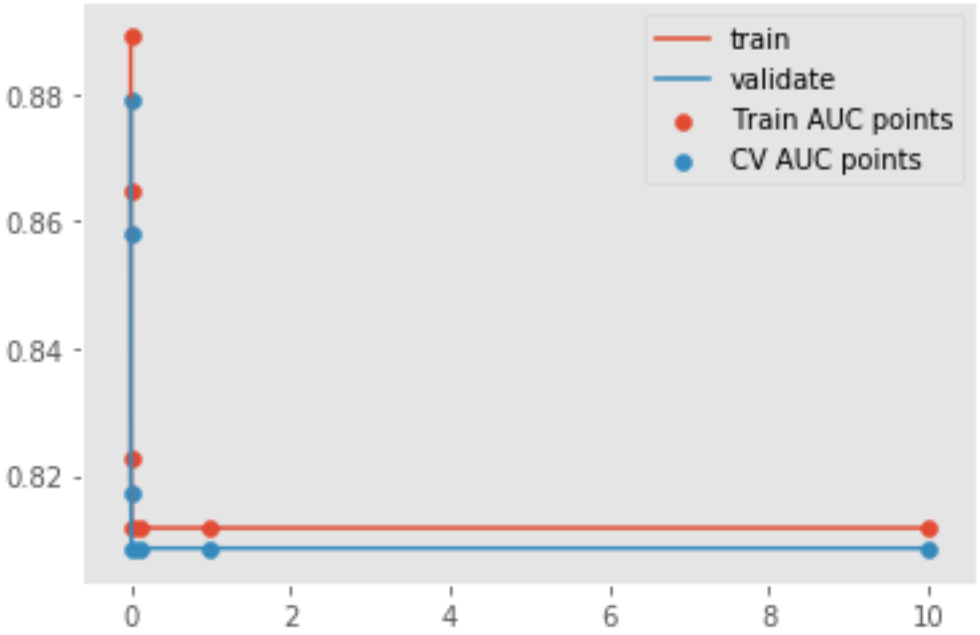
[152]:
```
score = pd.DataFrame({'name':names, 'train_score':train_auc_list, 'test_score':
↪validation_auc_list}).sort_values(by=['test_score'])
score
```

[152]:

	name	train_score	test_score
3	svm-tfidf_15k_0.01	0.811666	0.808471
4	svm-tfidf_15k_0.1	0.811782	0.808587
5	svm-tfidf_15k_1	0.811782	0.808587
6	svm-tfidf_15k_10	0.811782	0.808587
2	svm-tfidf_15k_0.001	0.822798	0.817372
1	svm-tfidf_15k_0.0001	0.864762	0.857834
0	svm-tfidf_15k_1e-05	0.889035	0.878969

[153]:
```
print(train_auc_list,validation_auc_list)
print(f'best hyperparameter got = {score.name.values[-1]} ##### Best cv score␣
↪got = {score.test_score.values[-1]}')
plt.plot(alpha, train_auc_list, label='train')
plt.plot(alpha, validation_auc_list, label='validate')
plt.scatter(alpha, train_auc_list, label='Train AUC points')
plt.scatter(alpha, validation_auc_list, label='CV AUC points')
plt.legend()
plt.grid()
plt.show()
```

```
[0.8890348991369909, 0.8647621003386462, 0.8227981573703355, 0.8116660775481254,
0.8117819975992921, 0.8117819975992921, 0.8117819975992921] [0.8789687931833012,
0.8578336021805566, 0.8173721678374075, 0.8084708725513468, 0.8085867979393189,
0.8085867979393189, 0.8085867979393189]
best hyperparameter got = svm-tfidf_15k_1e-05 ##### Best cv score got =
0.8789687931833012
```



[154]:
```
MODEL_NAME = 'svm_tfidf_15k'
clf = SGDClassifier(alpha=1e-05, 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]
```

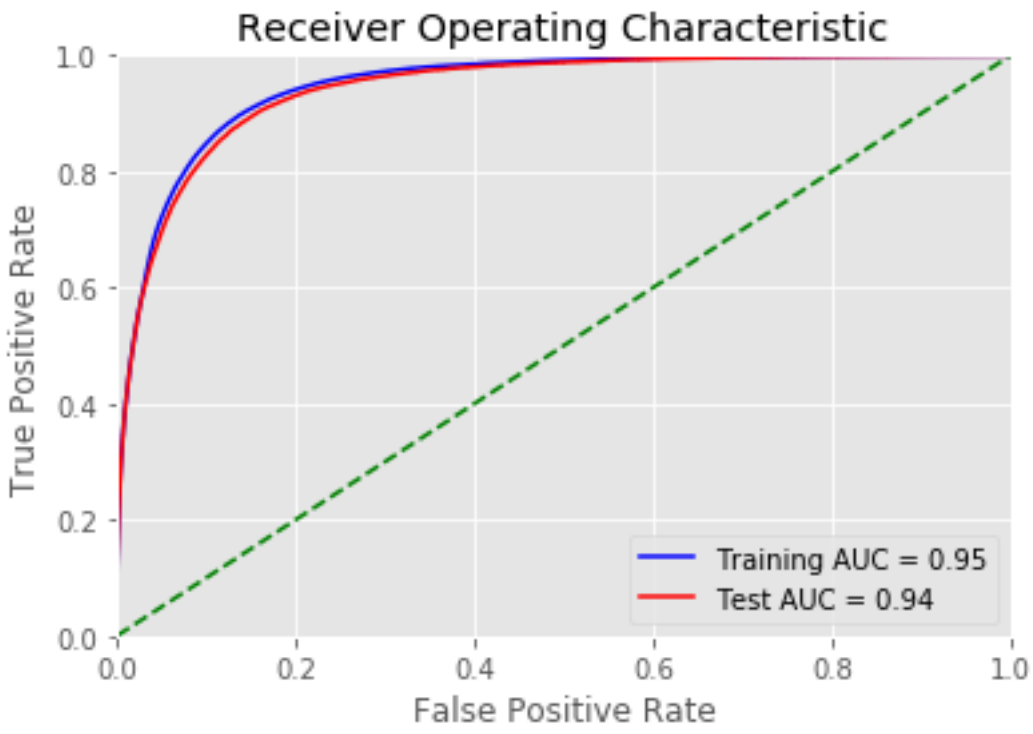
[155]:
```
train_data[MODEL_NAME] = predicted_train
validation_data[MODEL_NAME] = predicted_validation
print(f'Train score = {get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME)}')
print(f'Validation score = {get_metric_value(validation_data, identity_columns,␣
 ↪MODEL_NAME)}')
```

```
Train score = 0.8895580758634982
Validation score = 0.879360781648051
```

[156]:
```
predicted_test = clf.predict_proba(test_comment_tfidf_15000)[:,1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/svm_tfidf_15k_submission.csv', index=False)
```

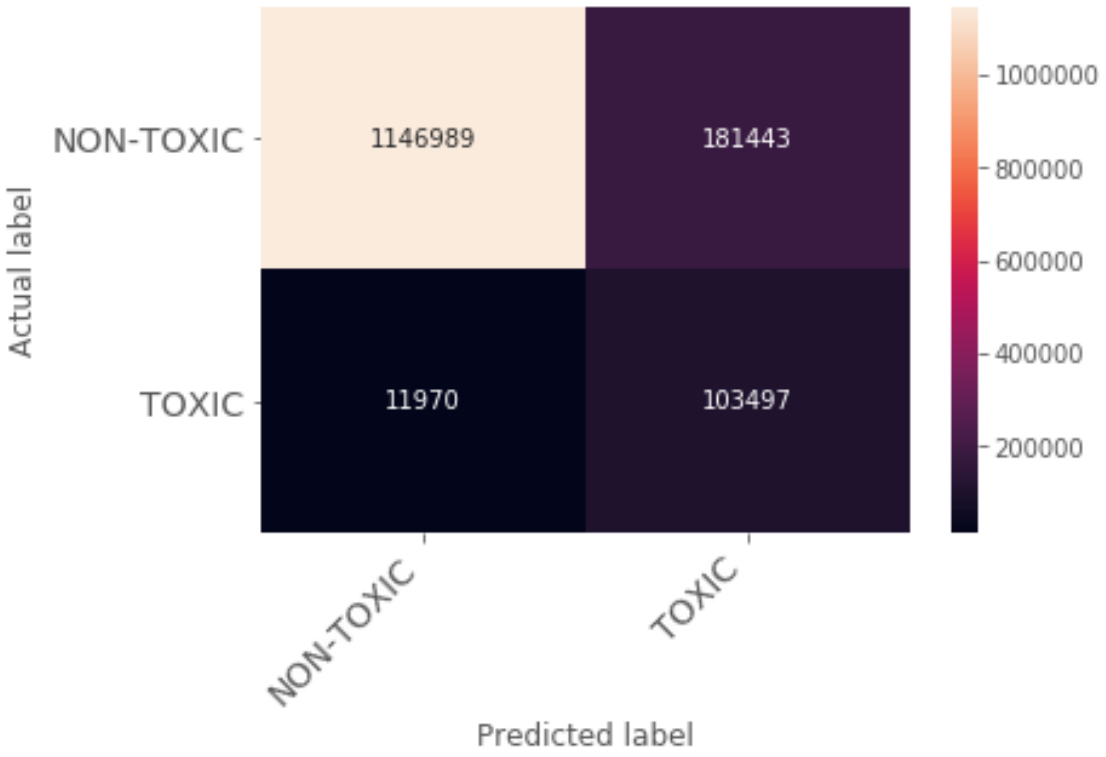
[ ]:
```

```

[157]:
```
# 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()
```



[158]:
```
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'])
```

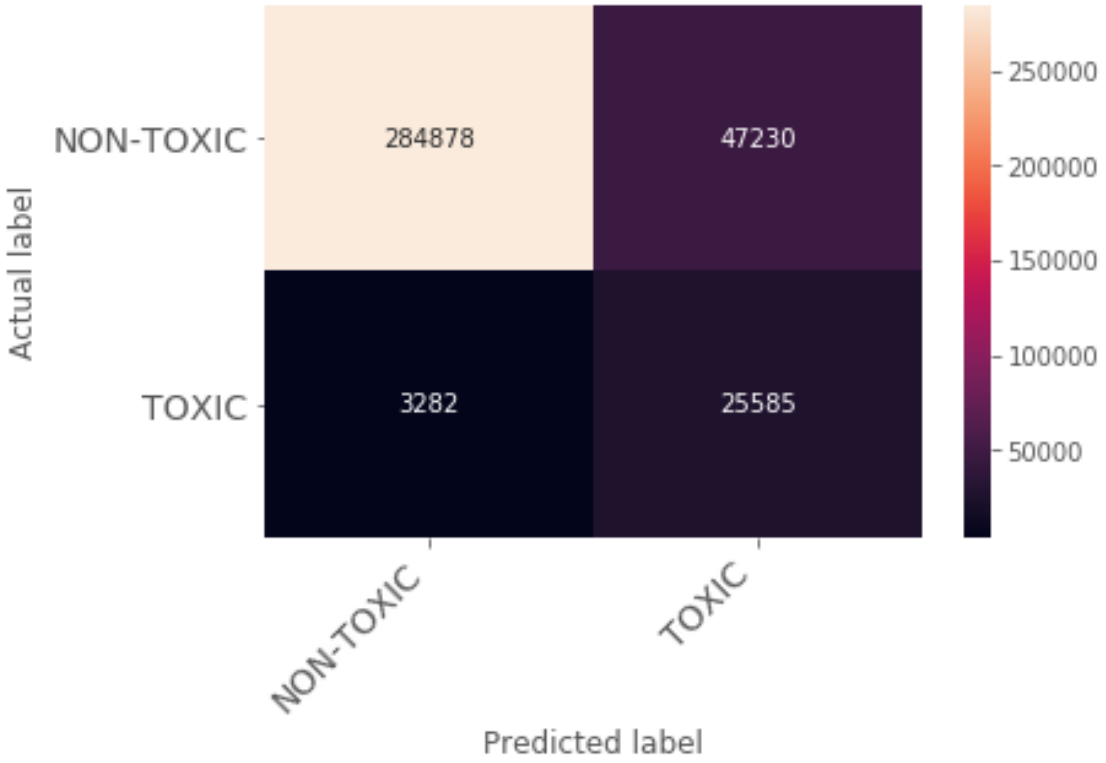
```
        TRAIN DATA CONFUSION MATRIX
```



=> 86.34 % of non-toxic comments predicted correctly => 89.63% of toxic comments predicted correctly

[159]:
```
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
```



=> 85.77 % of non-toxic comments predicted correctly => 89.18% of toxic comments predicted correctly

10000 features

[160]:
```
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 [04:51<00:00, 41.67s/it]
```

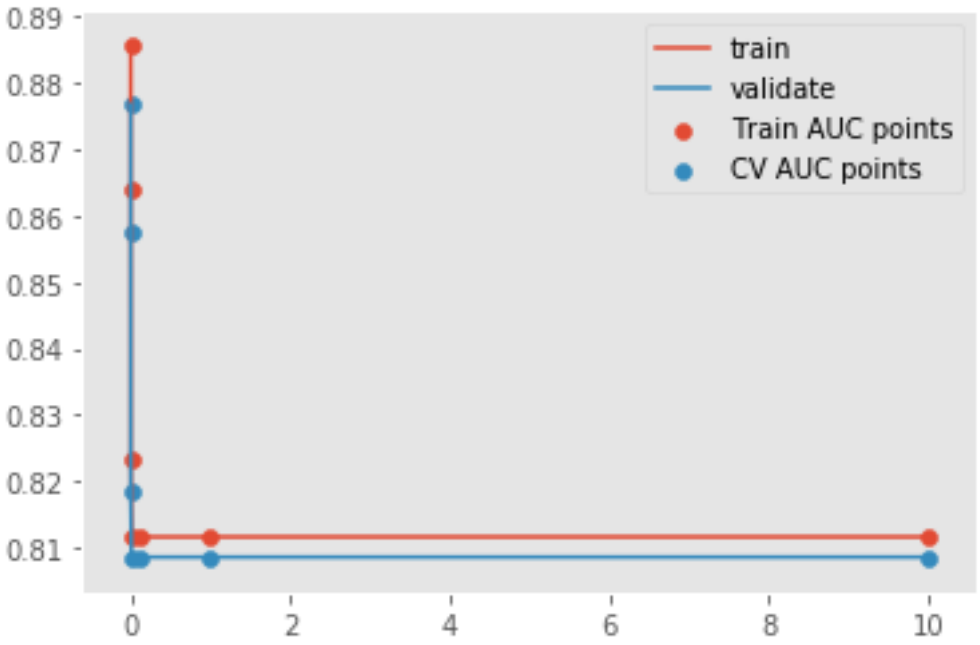
[161]:
```
score = pd.DataFrame({'name':names, 'train_score':train_auc_list, 'test_score':
↪validation_auc_list}).sort_values(by=['test_score'])
score
```

[161]:

	name	train_score	test_score
3	svm-tfidf_10k_0.01	0.811532	0.808482
6	svm-tfidf_10k_10	0.811668	0.808620
5	svm-tfidf_10k_1	0.811668	0.808620
4	svm-tfidf_10k_0.1	0.811668	0.808620
2	svm-tfidf_10k_0.001	0.823545	0.818405
1	svm-tfidf_10k_0.0001	0.863934	0.857572
0	svm-tfidf_10k_1e-05	0.885669	0.876793

[162]:
```
print(train_auc_list,validation_auc_list)
print(f'best hyperparameter got = {score.name.values[-1]} ##### Best cv score␣
↪got = {score.test_score.values[-1]}')
plt.plot(alpha, train_auc_list, label='train')
plt.plot(alpha, validation_auc_list, label='validate')
plt.scatter(alpha, train_auc_list, label='Train AUC points')
plt.scatter(alpha, validation_auc_list, label='CV AUC points')
plt.legend()
plt.grid()
plt.show()
```

```
[0.8856686504648588, 0.8639337539335741, 0.8235451574133681, 0.811531834901979,
0.8116679298145992, 0.8116679298594802, 0.8116679298234563] [0.8767930396434684,
0.8575715048044434, 0.8184052179126222, 0.8084815746247286, 0.8086197295460223,
0.8086197294835358, 0.8086197294574587]
best hyperparameter got = svm-tfidf_10k_1e-05 ##### Best cv score got =
0.8767930396434684
```



[163]:
```
MODEL_NAME = 'svm_tfidf_10k'
clf = SGDClassifier(alpha=1e-05, 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]
```

[164]:
```
train_data[MODEL_NAME] = predicted_train
validation_data[MODEL_NAME] = predicted_validation
print(f'Train score = {get_metric_value(train_data, identity_columns,␣
 ↪MODEL_NAME)}')
print(f'Validation score = {get_metric_value(validation_data, identity_columns,␣
 ↪MODEL_NAME)}')
```

```
Train score = 0.8853153367543201
Validation score = 0.8764942399219395
```

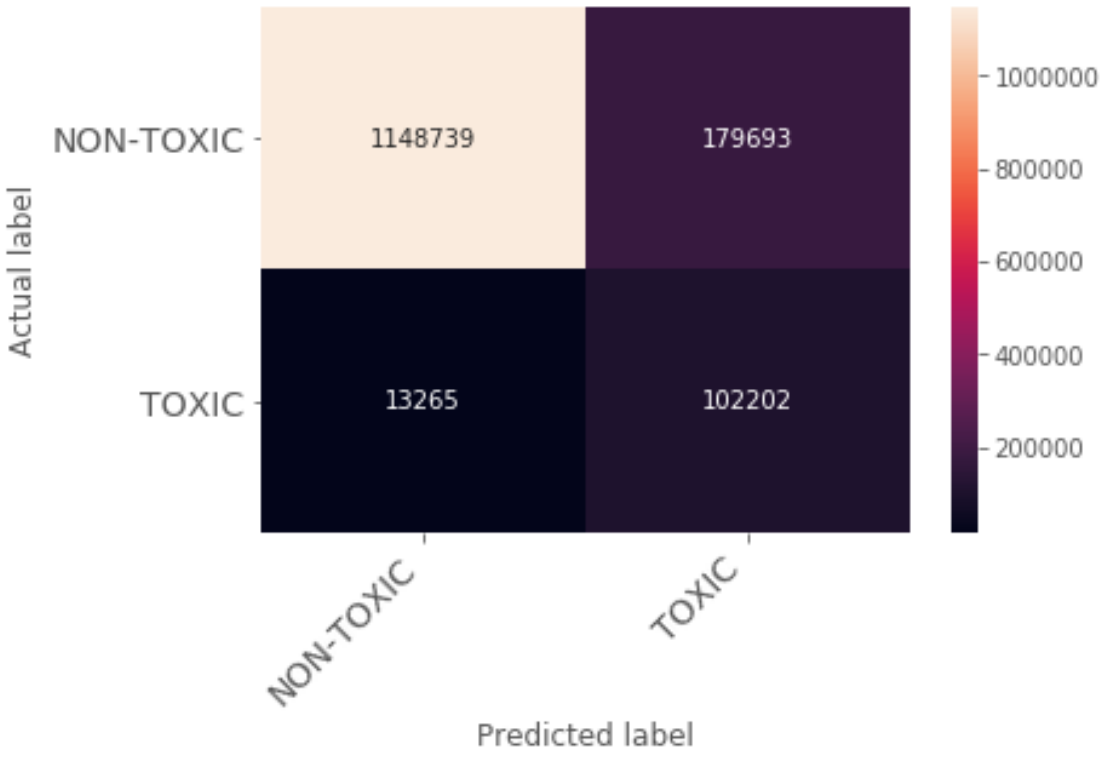
[165]:
```
predicted_test = clf.predict_proba(test_comment_tfidf_10000)[:,1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/svm_tfidf_10k_submission.csv', index=False)
```

[ ]:
```

```

[166]:
```
# 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()
```

[167]:
```
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'])
```

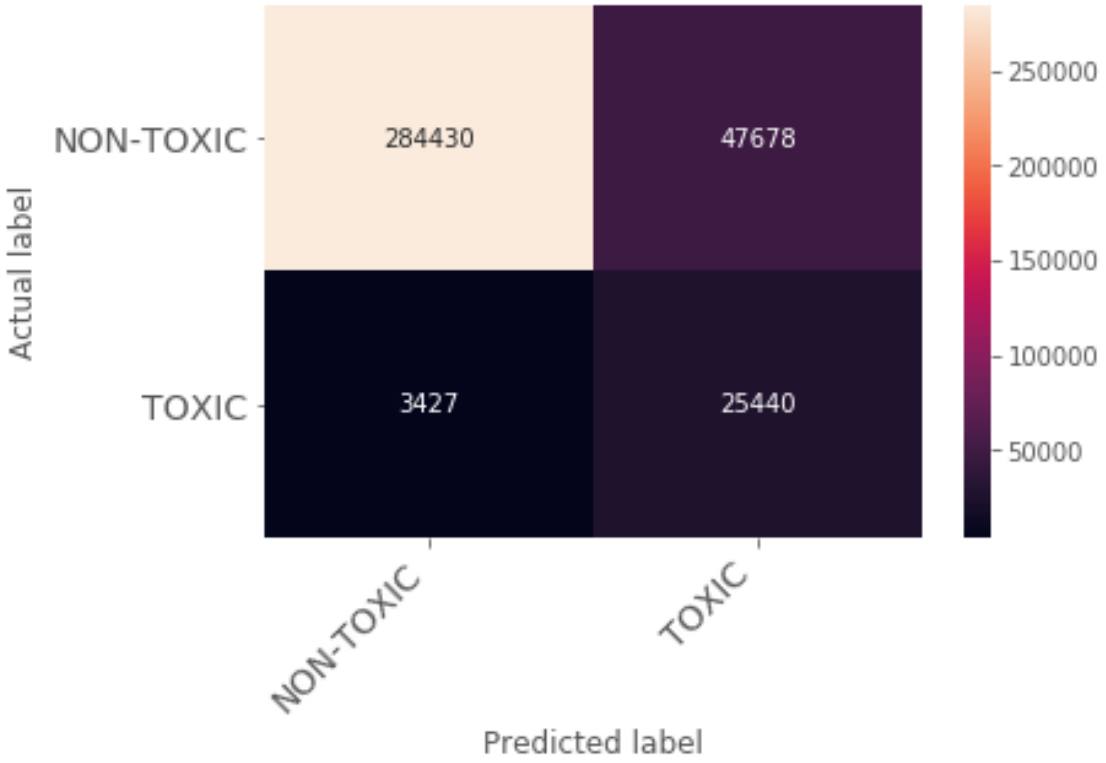
```
        TRAIN DATA CONFUSION MATRIX
```



=> 86.47 % of non-toxic comments predicted correctly => 88.51% of toxic comments predicted correctly

[168]:
```
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
```



=> 85.64 % of non-toxic comments predicted correctly => 88.68% of toxic comments predicted correctly

[169]:
```
gc.collect()
```

[169]: 2753

5.1.4 XG-Boost

- Because of the momery constraint I am going to train XG-Boost and Random Forest models with single parameter at a time and finally pick up the best hyperparameter to be used in the final models.

[170]:
```
# train_auc_list = []
# validation_auc_list = []
MODEL_NAME = f'xgb_15k'
clf = XGBClassifier(scale_pos_weight=99,n_estimators=600, n_jobs=-1)
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

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

```
0.8454389696554394
0.8308588587754047
```

[171]:
```
gc.collect()
```

[171]:
```
20
```

[172]:
```
# train_auc_list = []
# validation_auc_list = []
MODEL_NAME = f'xgb_15k'
clf = XGBClassifier(scale_pos_weight=99,n_estimators=1000, n_jobs=-1)
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

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

```
0.8629956494551478
0.846324172468981
```

[173]:
```
gc.collect()
```

[173]:
```
20
```

[174]:
```
# train_auc_list = []
# validation_auc_list = []
MODEL_NAME = f'xgb_15k'
clf = XGBClassifier(scale_pos_weight=99,n_estimators=1500, n_jobs=-1)
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
```

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

```
0.8750813511870685
0.855662883466987
```

[175]:
```
gc.collect()
```

[175]:
```
20
```

[27]:
```
# 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))
```

```
0.8834831095144419
0.8618897095328113
```

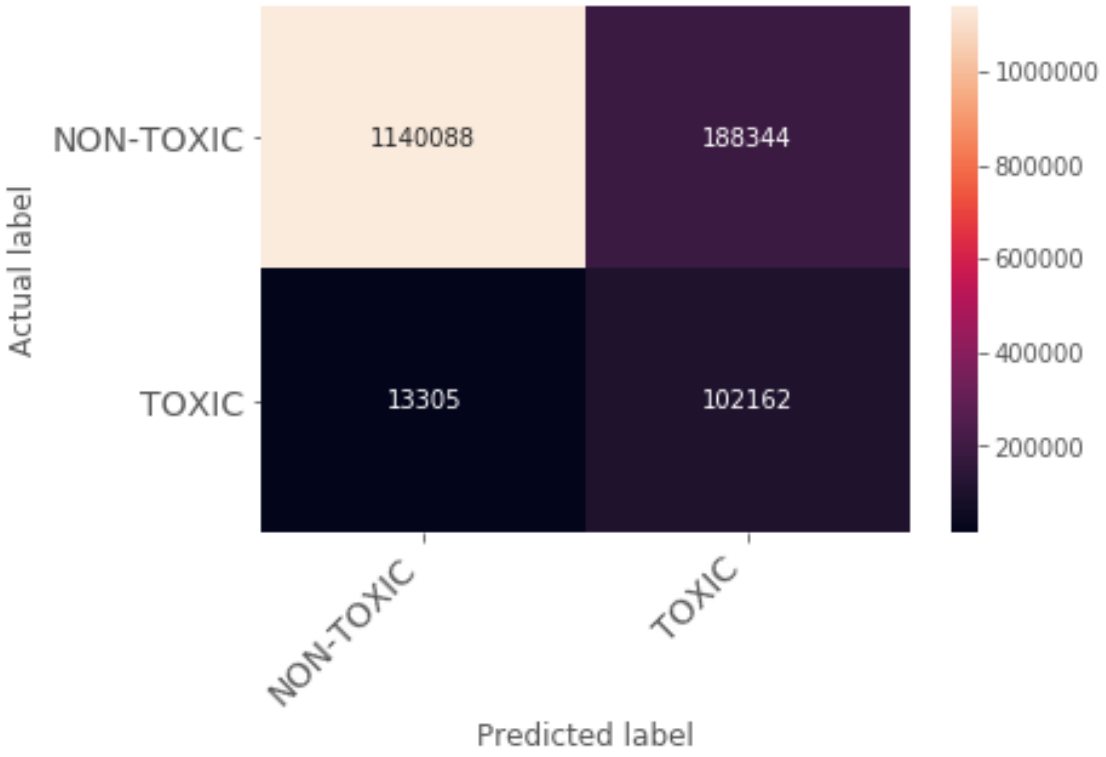
[28]:
```
predicted_test = clf.predict_proba(test_comment_tfidf_10000)[:,1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/xgb_10k_submission.csv', index=False)
```

[177]:
```
# 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()
```

[178]:
```
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'])
```

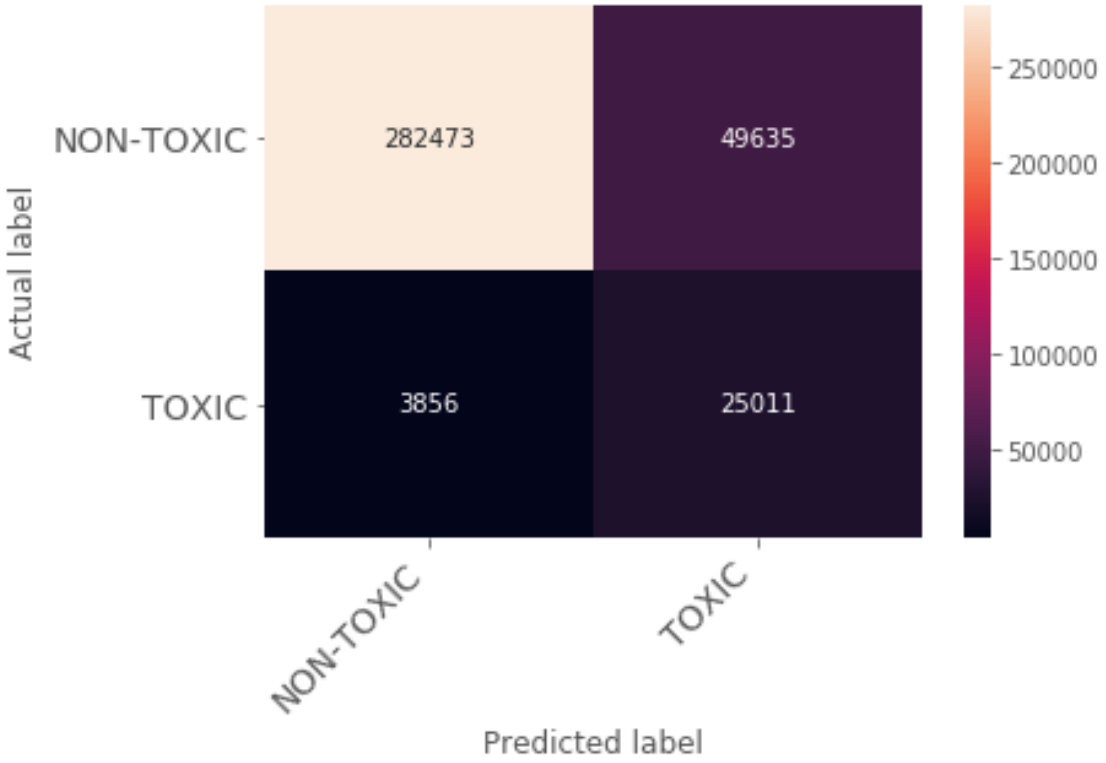
```
        TRAIN DATA CONFUSION MATRIX
```



=> 85.82 % of non-toxic comments predicted correctly => 88.47% of toxic comments predicted correctly

[179]:
```
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
```



=> 85.05 % of non-toxic comments predicted correctly => 87.18% of toxic comments predicted correctly

[180]:
```
gc.collect()
```

[180]: 10071

5.1.5 RandomForest Classifier

[181]:
```
n_estimators = 1000
max_depth= 6
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))
```

```
0.780364485192717
0.7710341053802323
```

[182]:
```
gc.collect()
```

[182]: 188

[183]:
```
n_estimators = 1500
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))
```

```
0.803692756142839
0.7863449270650648
```

[184]:
```
gc.collect()
```

[184]: 92

[185]:
```
n_estimators = 2000
max_depth= 6
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))
```

```
0.7867186851683692
0.7774001591947981
```

[186]:
```
gc.collect()
```

[186]:
```
68
```

[29]:
```
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))
```

```
0.8039419785272921
0.7866102845131482
```

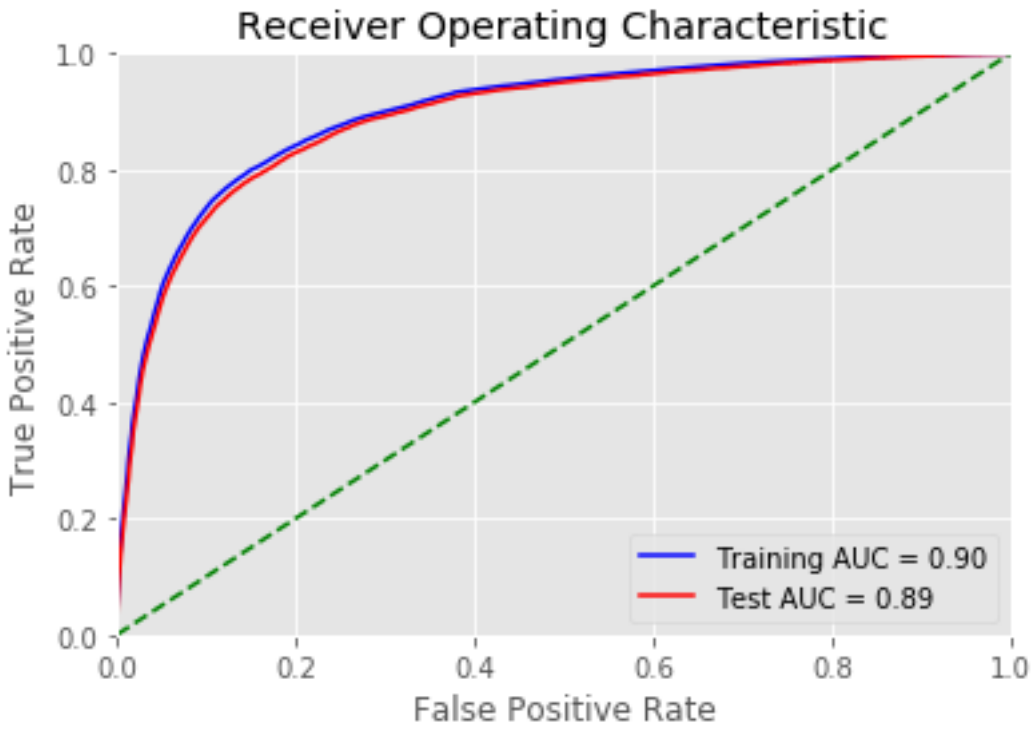
[30]:
```
predicted_test = clf.predict_proba(test_comment_tfidf_10000)[:,1]
test_data['prediction'] = predicted_test
```

```
test_data.to_csv('test_preds/rf_10k_submission.csv', index=False)
```

[188]:
```
gc.collect()
```

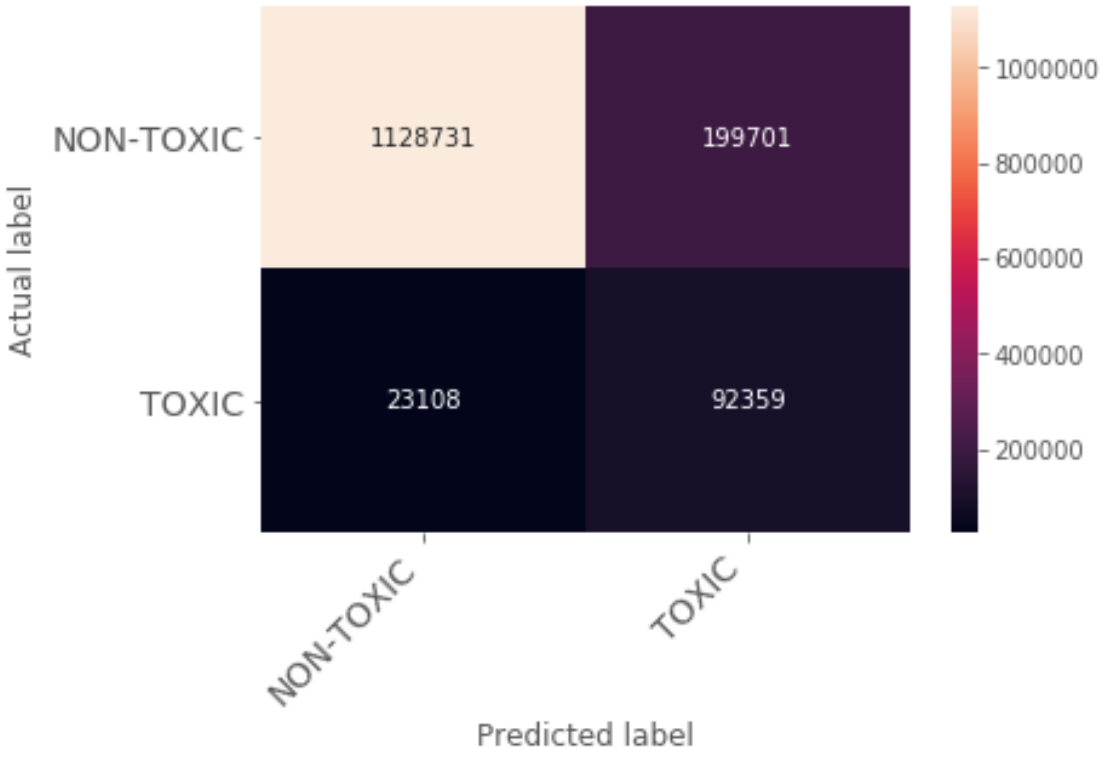
[188]: 68

[189]:
```
# 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()
```



[190]:
```
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'])
```

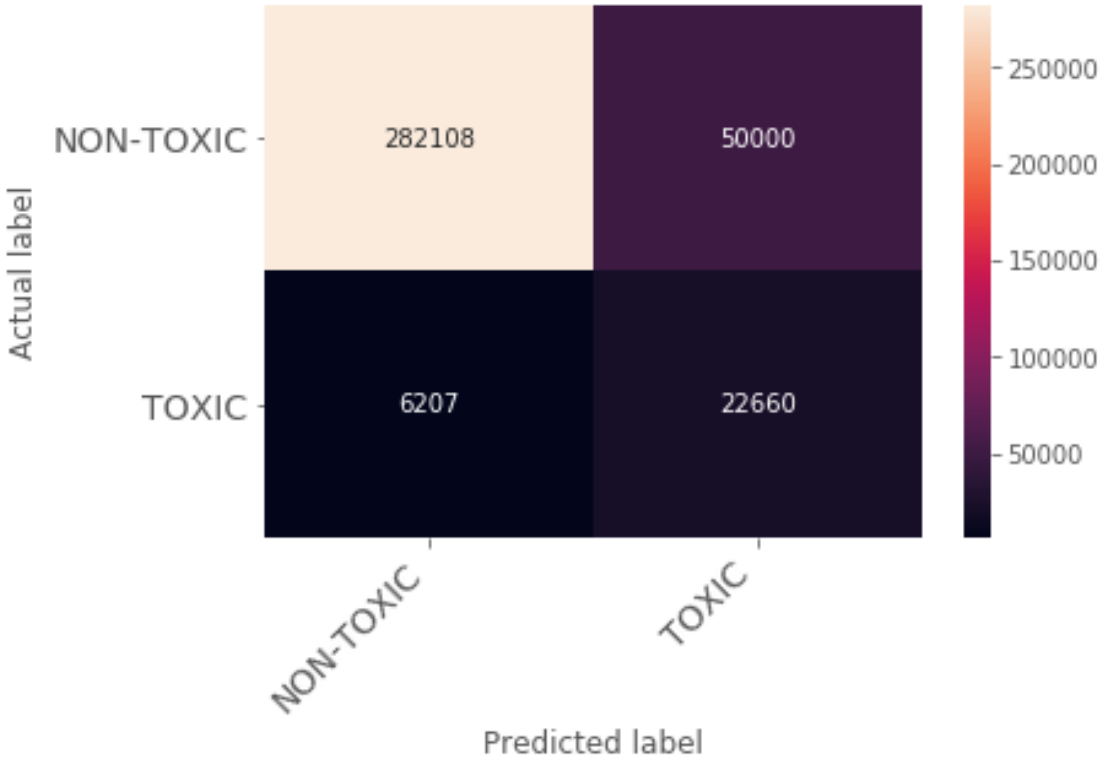
```
        TRAIN DATA CONFUSION MATRIX
```



=> 84.96 % of non-toxic comments predicted correctly => 79.98% of toxic comments predicted correctly

[191]:
```
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
```



=> 84.94 % of non-toxic comments predicted correctly => 78.99% of toxic comments predicted correctly

5.1.6 Stacking Classifier

Models with best hyperparameters

[19]:
```
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)
```

[20]:
```
import gc
gc.collect()
```

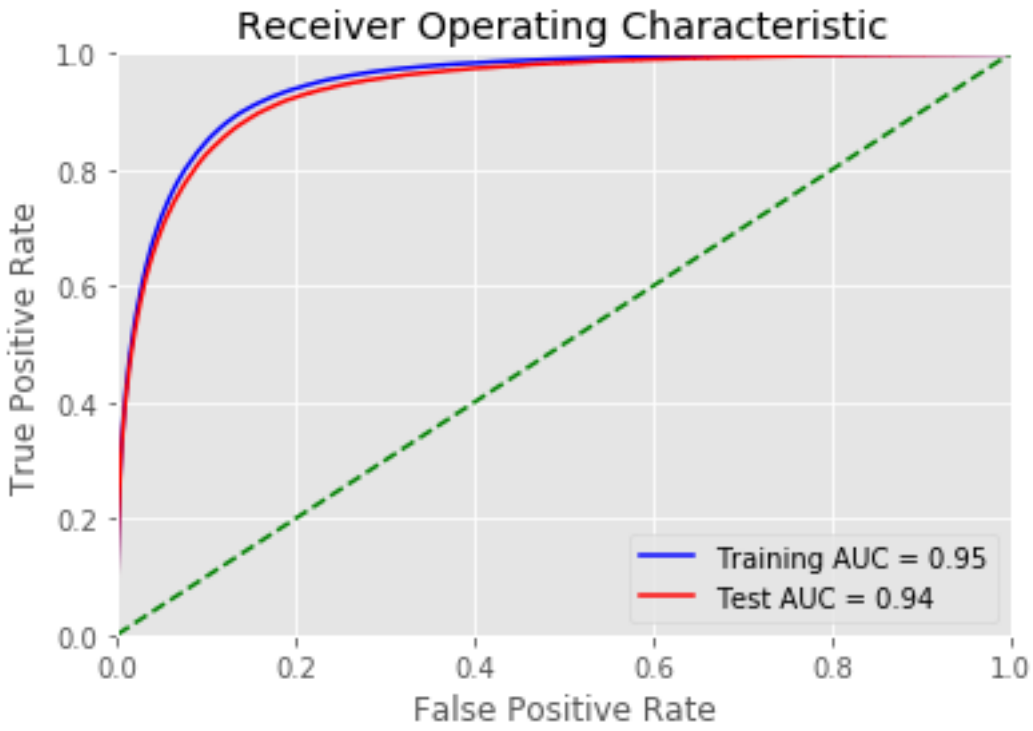
[20]: 40

Stacking models

[21]:
```
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))
```

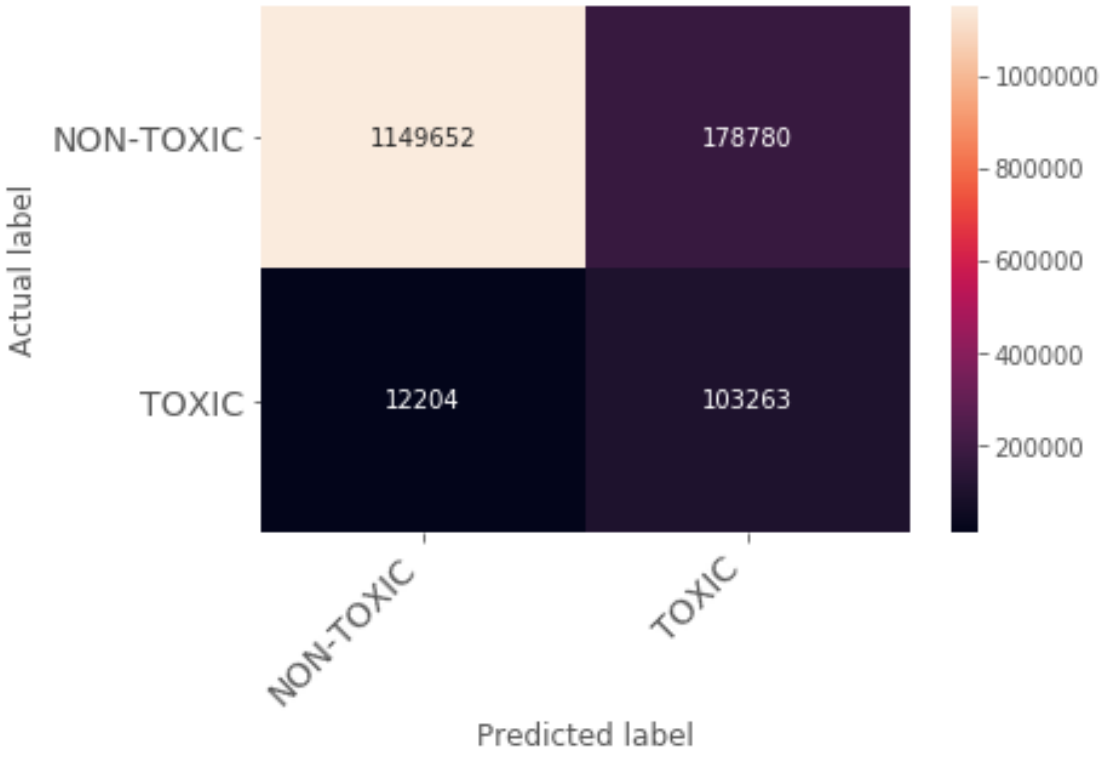
```
0.8917818215411302
0.8789805612696694
```

[22]:
```
# 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()
```



[23]:
```
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'])
```

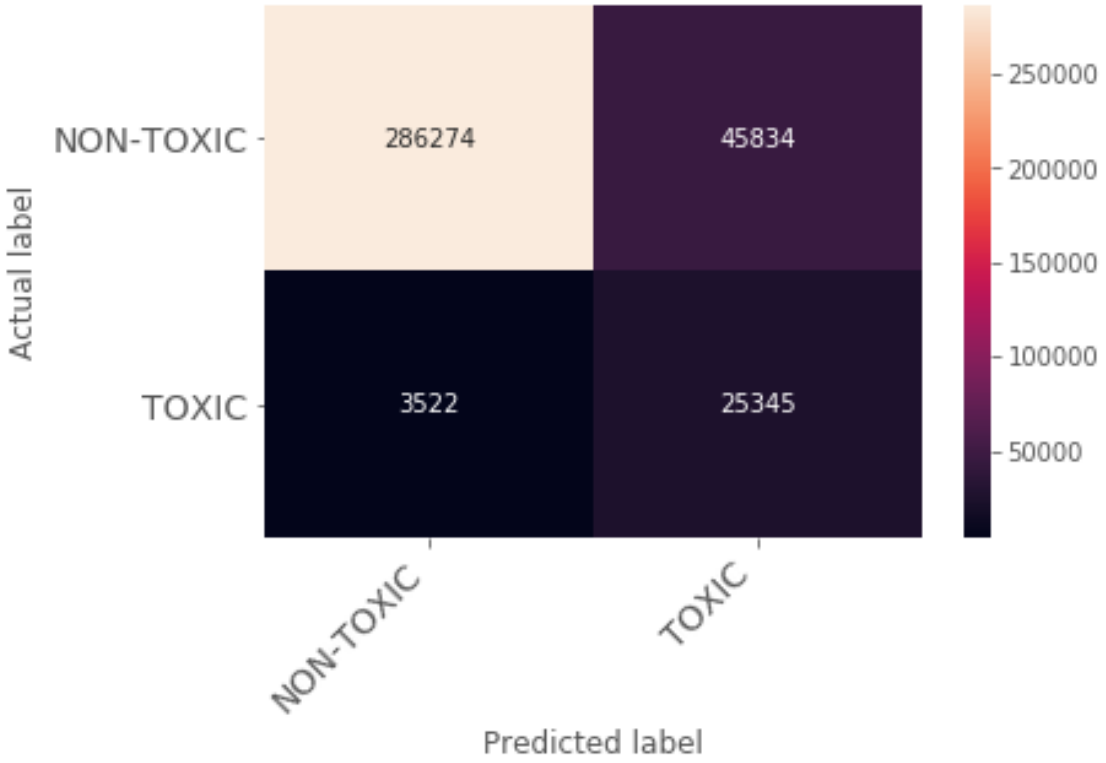
```
        TRAIN DATA CONFUSION MATRIX
```



=> 86.73 % of non-toxic comments predicted correctly => 89.43% of toxic comments predicted correctly

[25]:
```
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
```



=> 86.19 % of non-toxic comments predicted correctly => 88.35% of toxic comments predicted correctly

[26]:
```
predicted_test = clf.predict_proba(test_comment_tfidf_15000)[:,1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/stacking_15k_submission.csv', index=False)
```

6 Deep Learning Models

[1]:
```
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
```

```
/home/user/anaconda3/lib/python3.7/site-
packages/tensorflow/python/framework/dtypes.py:526: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/user/anaconda3/lib/python3.7/site-
packages/tensorflow/python/framework/dtypes.py:527: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/user/anaconda3/lib/python3.7/site-
packages/tensorflow/python/framework/dtypes.py:528: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/user/anaconda3/lib/python3.7/site-
packages/tensorflow/python/framework/dtypes.py:529: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/user/anaconda3/lib/python3.7/site-
packages/tensorflow/python/framework/dtypes.py:530: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
```

```
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/user/anaconda3/lib/python3.7/site-
packages/tensorflow/python/framework/dtypes.py:535: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
Using TensorFlow backend.
```

```
1.13.1
```

```
[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!
```

[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
```

[3]:
```
<Client: 'inproc://192.168.0.107/24002/1' processes=1 threads=12, memory=6.00
GB>
```

[26]:
```
EMBEDDING_FILES = [
    'deep learning/convolutional_model/crawl-300d-2M.gensim',
    'deep learning/convolutional_model/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""’\'ω÷•à− ³ ‘ ´°£€\×√²−
'
```

[5]:
```
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

[14]:
```
train_data = pd.read_csv('train/train.csv')
test_df = pd.read_csv('test/test.csv')
```

[15]:
```
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

[16]:
```
train_data, cv_df = train_test_split(train_data, test_size=0.2,␣
↪stratify=train_data.target.values, random_state=2020)
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

[17]:
```
neg_train = train_data[train_data['target'] == 1]
neg_train.shape
```

[17]:
```
(115467, 45)
```

[18]:
```
neg_validation = cv_df[cv_df['target'] == 1]
neg_validation.shape
```

[18]:
```
(28867, 45)
```

[19]:
```
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

[20]:
```
y_train = train_data[TARGET_COLUMN]
y_train = to_categorical(y_train)

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

[21]:
```
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()
```

[22]:
```
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)
```

[27]:
```
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, 2807.92it/s]
100%|      | 424070/424070 [02:36<00:00, 2709.57it/s]
```

6.4 Models

6.4.1 CNN Model

[28]:
```
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.
```

[29]:
```
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
```

[30]:
```
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 [==============================] - 199s 138us/step - loss:
0.1380 - auc_1: 0.9879
Epoch 2/5
1443899/1443899 [==============================] - 199s 138us/step - loss:
0.1264 - auc_1: 0.9899
Epoch 3/5
1443899/1443899 [==============================] - 199s 138us/step - loss:
0.1230 - auc_1: 0.9905
Epoch 4/5
1443899/1443899 [==============================] - 200s 138us/step - loss:
0.1203 - auc_1: 0.9909
Epoch 5/5
```

```
1443899/1443899 [==============================] - 199s 138us/step - loss:
0.1184 - auc_1: 0.9912
```

[31]:
```
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
```

[33]:
```
test_data = test_df
predicted_test = model.predict(x_test)[:, 1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/cnn_submission.csv', index=False)
```

[34]:
```
del model
```

6.4.2 Single layered LSTM

[35]:
```
import gc
gc.collect()
```

[35]:
```
26
```

[36]:
```
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)
```

[37]:
```
model = Model(inputs=[input_text], outputs=[output])
model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=[keras.metrics.AUC()])
```

```
print(model.summary())
```

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 220)               0
_________________________________________________________________
embedding_2 (Embedding)      (None, 220, 300)          127221300
_________________________________________________________________
lstm_1 (LSTM)                (None, 220, 128)          219648
_________________________________________________________________
flatten_2 (Flatten)          (None, 28160)             0
_________________________________________________________________
dropout_2 (Dropout)          (None, 28160)             0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               3604608
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 258
=================================================================
Total params: 131,045,814
Trainable params: 3,824,514
Non-trainable params: 127,221,300
_________________________________________________________________
None
```

[38]:
```
LSTM_1_layer_model = model.fit(
            x_train,
            y_train,
            batch_size=BATCH_SIZE,
            epochs=1
        )
```

```
Epoch 1/1
1443899/1443899 [==============================] - 2269s 2ms/step - loss: 0.1985
- auc_2: 0.9794
```

[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

[39]:
```
predicted_test = model.predict(x_test)[:, 1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/lstm_1_layer_submission.csv', index=False)
```

[40]:
```
del model
gc.collect()
```

[40]: 606

6.4.3 Two layered Bi-Directional LSTM

[41]:
```
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)
```

[42]:
```
model = Model(inputs=[input_text], outputs=[output])
model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=[keras.metrics.AUC()])
print(model.summary())
```

```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, 220)               0
_________________________________________________________________
embedding_3 (Embedding)      (None, 220, 300)          127221300
_________________________________________________________________
bidirectional_1 (Bidirection (None, 220, 256)          439296
_________________________________________________________________
bidirectional_2 (Bidirection (None, 220, 256)          394240
```

```
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 256)               0
_________________________________________________________________
dense_5 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_7 (Dense)              (None, 2)                 258
=================================================================
Total params: 128,104,502
Trainable params: 883,202
Non-trainable params: 127,221,300
_________________________________________________________________
None
```

[43]:
```
bi_dir_LSTM_2_layer_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_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 [==============================] - 8428s 6ms/step - loss: 0.1993
- auc_3: 0.9806
```

[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
```

[44]:
```
predicted_test = model.predict(x_test)[:, 1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/lstm_2_layer_submission.csv', index=False)
```

[45]:
```
del model
gc.collect()
```

[45]:
```
523
```

6.4.4 Research paper approach

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

[46]:
```
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)
```

[47]:
```
model = Model(inputs=input_text, outputs=[result])
model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
```

```
                  metrics=[keras.metrics.AUC()])
print(model.summary())
```

```
Model: "model_4"
_________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
=================================================================================
==================
input_4 (InputLayer)            (None, 220)          0
_________________________________________________________________________________
__________________
embedding_4 (Embedding)         (None, 220, 300)     127221300   input_4[0][0]
_________________________________________________________________________________
__________________
spatial_dropout1d_1 (SpatialDro (None, 220, 300)     0
embedding_4[0][0]
_________________________________________________________________________________
__________________
bidirectional_3 (Bidirectional) (None, 220, 256)     439296
spatial_dropout1d_1[0][0]
_________________________________________________________________________________
__________________
bidirectional_4 (Bidirectional) (None, 220, 256)     394240
bidirectional_3[0][0]
_________________________________________________________________________________
__________________
global_max_pooling1d_2 (GlobalM (None, 256)          0
bidirectional_4[0][0]
_________________________________________________________________________________
__________________
global_average_pooling1d_1 (Glo (None, 256)          0
bidirectional_4[0][0]
_________________________________________________________________________________
__________________
concatenate_1 (Concatenate)     (None, 512)          0
global_max_pooling1d_2[0][0]
global_average_pooling1d_1[0][0]
_________________________________________________________________________________
__________________
dense_8 (Dense)                 (None, 512)          262656
concatenate_1[0][0]
_________________________________________________________________________________
__________________
add_1 (Add)                     (None, 512)          0
concatenate_1[0][0]
                                                     dense_8[0][0]
```

```
__________________________________________________________________________________
__________________
dense_9 (Dense)                 (None, 512)          262656      add_1[0][0]
__________________________________________________________________________________
__________________
add_2 (Add)                     (None, 512)          0           add_1[0][0]
                                                                 dense_9[0][0]
__________________________________________________________________________________
__________________
dense_10 (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]:

[48]:
```
research_paper_model = model.fit(
            x_train,
            y_train,
            batch_size=BATCH_SIZE,
            epochs=1)
```

```
Epoch 1/1
1443899/1443899 [==============================] - 8081s 6ms/step - loss: 0.1329
- auc_4: 0.9861
```

[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	bnsp_auc
6	black	2956	0.839738	0.829836	0.971977
2	homosexual_gay_or_lesbian	2148	0.844528	0.831435	0.971242
5	muslim	4133	0.857721	0.866696	0.965435
7	white	5001	0.858279	0.833863	0.974571
4	jewish	1543	0.894097	0.899702	0.964022
8	psychiatric_or_mental_illness	990	0.916345	0.900627	0.969396
1	female	10652	0.925286	0.922746	0.966645
0	male	8998	0.926840	0.918693	0.968962
3	christian	8029	0.931650	0.949586	0.951175

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

[40]:
```
0.9228624083847328
```

[49]:
```
predicted_test = model.predict(x_test)[:, 1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/research_paper_submission.csv', index=False)
```

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

[50]:
```
165
```

6.4.5 Research paper with attention layer

[51]:
```
# 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
```

[52]:
```
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)
```

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

```
print(model.summary())
```

```
Model: "model_5"
_________________________________________________________________________________
_________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================
=================
input_5 (InputLayer)         (None, 220)               0
_________________________________________________________________________________
_________________
embedding_5 (Embedding)      (None, 220, 300)          127221300   input_5[0][0]
_________________________________________________________________________________
_________________
spatial_dropout1d_2 (SpatialDro (None, 220, 300)       0
embedding_5[0][0]
_________________________________________________________________________________
_________________
bidirectional_5 (Bidirectional) (None, 220, 256)       439296
spatial_dropout1d_2[0][0]
_________________________________________________________________________________
_________________
bidirectional_6 (Bidirectional) (None, 220, 256)       394240
bidirectional_5[0][0]
_________________________________________________________________________________
_________________
conv1d_4 (Conv1D)            (None, 218, 64)           49216
bidirectional_6[0][0]
_________________________________________________________________________________
_________________
attention_1 (Attention)      (None, 256)               476
bidirectional_6[0][0]
_________________________________________________________________________________
_________________
global_max_pooling1d_3 (GlobalM (None, 64)             0           conv1d_4[0][0]
_________________________________________________________________________________
_________________
global_average_pooling1d_2 (Glo (None, 64)             0           conv1d_4[0][0]
_________________________________________________________________________________
_________________
concatenate_2 (Concatenate)  (None, 384)               0
attention_1[0][0]
global_max_pooling1d_3[0][0]
global_average_pooling1d_2[0][0]
_________________________________________________________________________________
_________________
dense_11 (Dense)             (None, 384)               147840
```

```
concatenate_2[0][0]
__________________________________________________________________________________
__________________
add_3 (Add)                     (None, 384)          0
concatenate_2[0][0]
                                                                 dense_11[0][0]
__________________________________________________________________________________
__________________
dropout_4 (Dropout)             (None, 384)          0           add_3[0][0]
__________________________________________________________________________________
__________________
dense_12 (Dense)                (None, 384)          147840      dropout_4[0][0]
__________________________________________________________________________________
__________________
add_4 (Add)                     (None, 384)          0           dropout_4[0][0]
                                                                 dense_12[0][0]
__________________________________________________________________________________
__________________
dense_13 (Dense)                (None, 2)            770         add_4[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]:

[54]:
```
research_paper_model = model.fit(
            x_train,
            y_train,
            batch_size=BATCH_SIZE,
            epochs=4)
```

```
Epoch 1/4
1443899/1443899 [==============================] - 8155s 6ms/step - loss: 0.1360
- auc_5: 0.9835
Epoch 2/4
1443899/1443899 [==============================] - 8184s 6ms/step - loss: 0.1201
- auc_5: 0.9855
Epoch 3/4
```

```
1443899/1443899 [==============================] - 8136s 6ms/step - loss: 0.1143
- auc_5: 0.9849
Epoch 4/4
1443899/1443899 [==============================] - 8136s 6ms/step - loss: 0.1102
- auc_5: 0.9846
```

[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	bnsp_auc
5	muslim	4187	0.850103	0.879523	0.962804
6	black	3017	0.850858	0.827608	0.976094
2	homosexual_gay_or_lesbian	2227	0.851779	0.850920	0.970679
7	white	4932	0.863925	0.849635	0.974505
4	jewish	1540	0.888502	0.920189	0.955366
8	psychiatric_or_mental_illness	989	0.915460	0.921929	0.964872
3	christian	7955	0.925814	0.952348	0.949032
1	female	10754	0.931927	0.934992	0.966422
0	male	8883	0.933275	0.929630	0.969869

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

[25]:
```
0.9269571528954396
```

[55]:
```
predicted_test = model.predict(x_test)[:, 1]
test_data['prediction'] = predicted_test
test_data.to_csv('test_preds/research_paper_with_attan_submission.csv',␣
↪index=False)
```

6.5 Using Transfer Learning (BERT)

[64]:
```
# https://www.kaggle.com/prithvi1029/unprocessed-comments-worked-well
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'
    ]

AUX_COLUMNS = ['target',␣
↪'severe_toxicity','obscene','identity_attack','insult','threat']
```

```
TEXT_COLUMN = 'comment_text'
TARGET_COLUMN = 'target'
```

Embedding

[5]:
```
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]:
```
def build_model(embedding_matrix, num_aux_targets):#, loss_weight):
    """
    embedding layer
    droput 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""’\'ϖ ÷ •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]:
```
# https://www.kaggle.com/prithvi1029/unprocessed-comments-worked-well
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

[20]:
```
from prettytable import PrettyTable
x = PrettyTable()

column_names = ["model_names", "hyper_params", "train_metric_score",␣
 ↪"test_metric_score", "kaggle_submission_score"]
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␣
 ↪n_estimators=2000', 'n_estimators=1500,\n max_depth=12', 'params got from \n␣
 ↪others']
train_metric_scores = [85.3, 88.72, 88.97, 88.05, 80.51, 89.68]
test_metric_scores = [84.24, 87.84, 87.9, 86.73, 78.90, 87.57]
kaggle_scores = [83.52, 87.80, 88.03, 73.40, 68.30, 75.35]
x.add_column(column_names[0], model_names)
x.add_column(column_names[1], hyper_params)
x.add_column(column_names[2], train_metric_scores)
x.add_column(column_names[3], test_metric_scores)
x.add_column(column_names[4], kaggle_scores)
print(x.get_string(sortby="kaggle_submission_score", reversesort = True))
```

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

```
+---------------------+----------------------+--------------------+-------------
------+--------------------------+
|     model_names     |     hyper_params     | train_metric_score |
test_metric_score | kaggle_submission_score |
+---------------------+----------------------+--------------------+-------------
------+--------------------------+
|         SVM         |      apha=1e-5       |       88.97        |        87.9
|          88.03          |
| Logistic Regression |      alpha=1e-5      |       88.72        |       87.84
|           87.8          |
|     Naive Bayes     |       alpha=1        |        85.3        |       84.24
|          83.52          |
|       Stacking      |   params got from    |       89.68        |       87.57
|          75.35          |
|                     |        others        |                    |
|                         |
|       XG-Boost      | scale_pos_weight=99, |       88.05        |       86.73
|           73.4          |
|                     |   n_estimators=2000  |                    |
|                         |
|    Random Forest    |  n_estimators=1500,  |       80.51        |        78.9
|           68.3          |
|                     |     max_depth=12     |                    |
|                         |
+---------------------+----------------------+--------------------+-------------
------+--------------------------+
```

7.2 Deep Learning Models

[21]:
```
column_names = ["model_names", "epochs", "test_metric_score",␣
 ↪"kaggle_submission_score"]
model_names = ['CNN', 'Single layer LSTM', 'Two Layered \n Bi-Directional␣
 ↪LSTM', 'LSTM', 'LSTM \n with Attention', '(LSTM + \n BERT small + \n BERT␣
 ↪large)']
epochs = ['5','1', '1', '1', '4', '-']
test_metric_scores = [90.98, 88.70, 88.78, 92.28, 92.70, 96.67]
kaggle_scores = [91.14, 87.66, 88.47, 92.14, 92.63, 94.17]

x = PrettyTable()

x.add_column(column_names[0], model_names)
```

```
x.add_column(column_names[1], epochs)
x.add_column(column_names[2], test_metric_scores)
x.add_column(column_names[3], kaggle_scores)
print(x.get_string(sortby="kaggle_submission_score", reversesort = True))
```

```
+----------------------+--------+-------------------+--------------------------+
|     model_names      | epochs | test_metric_score | kaggle_submission_score |
+----------------------+--------+-------------------+--------------------------+
|       (LSTM +        |   -    |       96.67       |          94.17           |
|     BERT small +     |        |                   |                          |
|      BERT large)     |        |                   |                          |
|        LSTM          |   4    |        92.7       |          92.63           |
|    with Attention    |        |                   |                          |
|         LSTM         |   1    |       92.28       |          92.14           |
|         CNN          |   5    |       90.98       |          91.14           |
|     Two Layered      |   1    |       88.78       |          88.47           |
|  Bi-Directional LSTM |        |                   |                          |
|  Single layer LSTM   |   1    |        88.7       |          87.66           |
+----------------------+--------+-------------------+--------------------------+
```

7.3 Conclusion

- We are getting best result from the combination of BERT small, BERT large and LSTM.
- The weight initialization also helped us in LSTM model as we were able to inculcate some information about the identities.
- We are using weighted average for prediction in LSTM model
- We are using pct ranking for for both bert as well as lstm predictions and given 30% and 70% weightage to lstm and bert predicted values respectively. This approach has scope of experimentation.
- We must note that we are getting decent score with simple CNN for just 5 epochs. So there may be a scope of improvement there.

[ ]: