Jigsaw Unintended bias in Toxicity Classification

1. Business Problem

Problem Description

- Social networking sites are the source of most of the recent trend. Almost every human is or these sites. Leading platforms gives people freedom to express themselves through posting that is good in an idea world, where no one is expected to abuse, such freedom but in real wo abuse of freedom often leads to hate spreading, racial slurring or verbal assault. These dang opinion or sharing any media for that matter which is harmful for leading platform as well as
- The Conversation AI team, a research initiative founded by Jigsaw and Google (both part of / in conversation. Their idea was to build an application that could detect and remove or limit of a particular site. Conversational AI uses machine learning, which provides a distinct advar detect comments that contain toxic content. Our solution makes use of machine learning,na preprocessing the data and deep learning approaches were used to train a model that could

Problem Statement

The model which was built by The Conversation AI team has the problem of unintended bias and t because of this the comments which are not actually toxic will be predicted as toxic.

1.2 Source / useful links

- https://www.kaggle.com/c/jigsaw-unintended-bias-in-toxicity-classification
- https://www.kaggle.com/gpreda/jigsaw-eda
- https://www.kaggle.com/kabure/simple-eda-hard-views-w-easy-code
- https://www.kaggle.com/ekhtiar/unintended-eda-with-tutorial-notes
- https://www.kaggle.com/dborkan/benchmark-kernel
- https://www.kaggle.com/thousandvoices/simple-lstm/log

1.3 Real World / Business Objectives and Constraints

- Predicting whether a comment is toxic or not with a probability score.
- Minimize unintended bias.
- No strict latency requirements.

2. Machine Learning problem

2.1 Data

2.1.1 Data Overview

- All of the data is in 2 files: Train and Test.
- Train.csv contains 45 columns: ['id' 'target' 'comment_text' 'severe_toxicity' 'obscene' 'identity 'bisexual' 'black' 'buddhist' 'christian' 'female' 'heterosexual' 'hindu' 'homosexual_gay_or_lesbi 'latino' 'male' 'muslim' 'other_disability' 'other_gender' 'other_race_or_ethnicity' 'other_religion' 'psychiatric_or_mental_illness' 'transgender' 'white' 'created_date' 'publication_id' 'parent_id' 'a 'disagree' 'sexual_explicit' 'identity_annotator_count' 'toxicity_annotator_count']
- Test.csv contains id,comment_text
- Size of Train.csv - 778.4MB
- Size of Test.csv - 28.54MB
- Number of rows in Train.csv = 1804874
- Number of rows in Test.csv = 97320

Data Field Explaination

The comments are stored in train and test in comment_text column. Additionally, in train we have t certain sensitive topic. The topic is related to five categories: race or ethnicity, gender, sexual orien

- race or ethnicity: asian, black, jewish, latino, other_race_or_ethnicity, white
- gender: female, male, transgender, other_gender
- sexual orientation: bisexual, heterosexual, homosexual_gay_or_lesbian, other_sexual_orienta
- religion: atheist,buddhist, christian, hindu, muslim, other_religion
- disability: intellectual_or_learning_disability, other_disability, physical_disability, psychiatric_c

We also have few article/comment identification information: created_date publication_id parent_i

Several user feedback information associated with the comments are provided:

- rating
- funny
- wow
- sad
- likes
- disagree
- sexual_explicit

In this dataset there are two fields related to annotations:

- identity_annotator_count
- toxicity_annotator_count

2.1.2 Example Data point

comment_text='This is so cool.'
target=0.0

2.2 Mapping the real-world problem to a Machine Learning Problem

2.2.1 Type of Machine Learning Problem

The task is classification and we need to give probabilies w.r.t. toxic level.

2.2.2 Performance metric

This competition uses a newly developed metric that combines several submetrics to balance ove
unintended bias.
Overall AUC
This is the ROC-AUC for the full evaluation set.

Bias AUCs
To measure unintended bias, we again calculate the ROC-AUC, this time on three specific subsets
capturing a different aspect of unintended bias.

a. Subgroup AUC — This calculates AUC on only the examples from the subgroup. It represents m
the group itself. A low value in this metric means the model does a poor job of distinguishing betw
mention the identity.

b. BNSP AUC — This calculates AUC on the positive examples from the background and the negat
here means that the model confuses toxic examples that mention the identity with non-toxic exam

c. BPSN AUC — This calculates AUC on the negative examples from the background and the posit
in this metric means that the model confuses non-toxic examples that mention the identity with to

d. Final Metrics — We combine the overall AUC with the generalized mean of the Bias AUCs to calc

score=w0AUCoverall+∑a=1AwaMp(ms,a) where:

A = number of submetrics (3)

ms,a = bias metric for identity subgroups using submetric a

wa = a weighting for the relative importance of each submetric; all four w values set to 0.25

3. Exploratory Data Analysis

3.1 Data Loading

```
!pip install emoji
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from tqdm import tqdm
from wordcloud import WordCloud
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
import string
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from sklearn import metrics
from sklearn.metrics import roc_auc_score,roc_curve,auc,confusion_matrix,classification_report,log_loss
from sklearn.manifold import TSNE
from sklearn import preprocessing
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
import re
from gensim.models import KeyedVectors
from wordcloud import WordCloud
from scipy.sparse import hstack
from sklearn.manifold import TSNE
from sklearn.preprocessing import Normalizer
import plotly.offline as py
import plotly
from plotly.offline import *
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
from sklearn.naive_bayes import MultinomialNB
import emoji
from tqdm.notebook import tqdm
tqdm.pandas()

import tensorflow
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Embedding, SpatialDropout1D, add, concatenate
from tensorflow.compat.v1.keras.layers import CuDNNLSTM, Bidirectional, GlobalMaxPooling1D, GlobalAverageP
from tensorflow.keras.preprocessing import text, sequence
from gensim.models import KeyedVectors
from tensorflow.keras.utils import plot_model
!pip install pyLDAvis
from pprint import pprint
```

```
# Gensim
import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel
from gensim.parsing.preprocessing import STOPWORDS
from nltk.stem import WordNetLemmatizer, SnowballStemmer
from nltk.stem.porter import *

# spacy for lemmatization
import spacy

# Plotting tools
import pyLDAvis
import pyLDAvis.gensim  # don't skip this
import matplotlib.pyplot as plt
%matplotlib inline

# Enable logging for gensim - optional
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.ERROR)
import nltk
nltk.download('wordnet')
from gensim.models import LdaModel
import tensorflow as tf
from textblob import TextBlob, Word, Blobber
from sklearn.preprocessing import StandardScaler

!pip show tensorflow
!pip install plot_model
!pip install tensorboardcolab
%load_ext tensorboard
!rm -rf ./logs/
import warnings
warnings.filterwarnings("ignore")
```

```
  Building wheel for emoji (setup.py) ... done
  Created wheel for emoji: filename=emoji-0.5.4-cp36-none-any.whl size=42176 sha256=c4b5063117a76b
  Stored in directory: /root/.cache/pip/wheels/2a/a9/0a/4f8e8cce8074232aba240caca3fade315bb49fac688(
Successfully built emoji
Installing collected packages: emoji
Successfully installed emoji-0.5.4
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing
  import pandas.util.testing as tm
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
Collecting pyLDAvis
  Downloading https://files.pythonhosted.org/packages/a5/3a/af82e070a8a96e13217c8f362f9a73e82d61a
     |████████████████████████████████| 1.6MB 2.7MB/s
Requirement already satisfied: wheel>=0.23.0 in /usr/local/lib/python3.6/dist-packages (from pyLDAvis) (0
Requirement already satisfied: numpy>=1.9.2 in /usr/local/lib/python3.6/dist-packages (from pyLDAvis) (1.
Requirement already satisfied: scipy>=0.18.0 in /usr/local/lib/python3.6/dist-packages (from pyLDAvis) (1.4
Requirement already satisfied: pandas>=0.17.0 in /usr/local/lib/python3.6/dist-packages (from pyLDAvis) (
Requirement already satisfied: joblib>=0.8.4 in /usr/local/lib/python3.6/dist-packages (from pyLDAvis) (0.1
Requirement already satisfied: jinja2>=2.7.2 in /usr/local/lib/python3.6/dist-packages (from pyLDAvis) (2.1
Requirement already satisfied: numexpr in /usr/local/lib/python3.6/dist-packages (from pyLDAvis) (2.7.1)
Requirement already satisfied: pytest in /usr/local/lib/python3.6/dist-packages (from pyLDAvis) (3.6.4)
Requirement already satisfied: future in /usr/local/lib/python3.6/dist-packages (from pyLDAvis) (0.16.0)
Collecting funcy
  Downloading https://files.pythonhosted.org/packages/ce/4b/6ffa76544e46614123de31574ad95758c421a
     |████████████████████████████████| 552kB 14.9MB/s
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/python3.6/dist-packages (from pan
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages (from pandas>=0.1
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.6/dist-packages (from jinja2>=
Requirement already satisfied: atomicwrites>=1.0 in /usr/local/lib/python3.6/dist-packages (from pytest->
Requirement already satisfied: more-itertools>=4.0.0 in /usr/local/lib/python3.6/dist-packages (from pytes
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages (from pytest->pyLDAv
Requirement already satisfied: pluggy<0.8,>=0.5 in /usr/local/lib/python3.6/dist-packages (from pytest->p
Requirement already satisfied: py>=1.5.0 in /usr/local/lib/python3.6/dist-packages (from pytest->pyLDAvis
Requirement already satisfied: attrs>=17.4.0 in /usr/local/lib/python3.6/dist-packages (from pytest->pyLD
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (from pytest->pyLDAvi
Building wheels for collected packages: pyLDAvis, funcy
  Building wheel for pyLDAvis (setup.py) ... done
  Created wheel for pyLDAvis: filename=pyLDAvis-2.1.2-py2.py3-none-any.whl size=97711 sha256=7db75b
  Stored in directory: /root/.cache/pip/wheels/98/71/24/513a99e58bb6b8465bae4d2d5e9dba8f0bef8179e3
  Building wheel for funcy (setup.py) ... done
  Created wheel for funcy: filename=funcy-1.14-py2.py3-none-any.whl size=32042 sha256=8d842ad19f7ff
  Stored in directory: /root/.cache/pip/wheels/20/5a/d8/1d875df03deae6f178dfdf70238cca33f948ef8a6f52
Successfully built pyLDAvis funcy
Installing collected packages: funcy, pyLDAvis
Successfully installed funcy-1.14 pyLDAvis-2.1.2
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk

Enter your authorization code:
..........
Mounted at /content/drive
```

```
Collecting plot_model
```

```
# Loading the train data into pandas dataframe
train = pd.read_csv('/content/drive/My Drive/train.csv')
# Loading the test data into pandas dataframe
test = pd.read_csv('/content/drive/My Drive/test.csv')
```

```
# We have 1.8 millions of data record in train dataset with 45 features given
print("Number of data points in train data", train.shape)
print('-'*50)
print("The attributes of train data :", train.columns.values)
```

```
Number of data points in train data (1804874, 45)
--------------------------------------------------
The attributes of train data : ['id' 'target' 'comment_text' 'severe_toxicity' 'obscene'
 'identity_attack' 'insult' 'threat' 'asian' 'atheist' 'bisexual' 'black'
 'buddhist' 'christian' 'female' 'heterosexual' 'hindu'
 'homosexual_gay_or_lesbian' 'intellectual_or_learning_disability'
 'jewish' 'latino' 'male' 'muslim' 'other_disability' 'other_gender'
 'other_race_or_ethnicity' 'other_religion' 'other_sexual_orientation'
 'physical_disability' 'psychiatric_or_mental_illness' 'transgender'
 'white' 'created_date' 'publication_id' 'parent_id' 'article_id' 'rating'
 'funny' 'wow' 'sad' 'likes' 'disagree' 'sexual_explicit'
 'identity_annotator_count' 'toxicity_annotator_count']
```

```
print("Sample train datapoint :")
train.head(1)
```

Sample train datapoint :

	id	target	comment_text	severe_toxicity	obscene	identity_attack	insult	threat	as
0	59848	0.0	This is so cool. It's like, 'would you want yo...	0.0	0.0	0.0	0.0	0.0	N

```
# We have 97k of data record in test dataset
print("Number of data points in test data", test.shape)
print(test.columns.values)
test.head(1)
```

```
Number of data points in test data (97320, 2)
['id' 'comment_text']
```

	id	comment_text
0	7097320	[Integrity means that you pay your debts.]\n\...

Exploratory Data Analysis

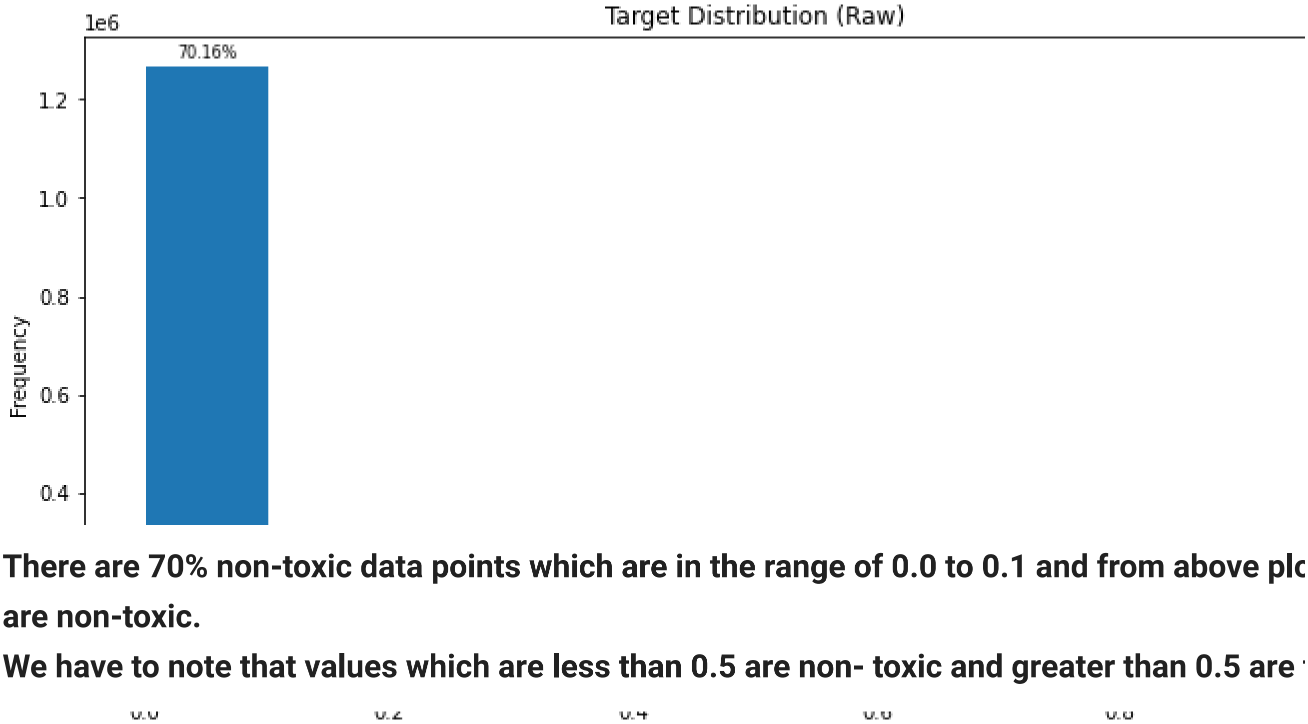
Percentage of NaN values

```
# its always a good idea to count the amount of missing values before diving into any analysis
```

```
# Its always a good idea to count the amount of missing values before diving into any analysis
# Lets also see how many missing values (in percentage) we are dealing with
miss_val_train_df = train.isnull().sum(axis=0) / len(train)
miss_val_train_df = miss_val_train_df[miss_val_train_df > 0] * 100
miss_val_train_df
```

```
asian                                  77.553558
atheist                                77.553558
bisexual                               77.553558
black                                  77.553558
buddhist                               77.553558
christian                              77.553558
female                                 77.553558
heterosexual                           77.553558
hindu                                  77.553558
homosexual_gay_or_lesbian              77.553558
intellectual_or_learning_disability    77.553558
jewish                                 77.553558
latino                                 77.553558
male                                   77.553558
muslim                                 77.553558
other_disability                       77.553558
other_gender                           77.553558
other_race_or_ethnicity                77.553558
other_religion                         77.553558
other_sexual_orientation               77.553558
physical_disability                    77.553558
psychiatric_or_mental_illness          77.553558
transgender                            77.553558
white                                  77.553558
parent_id                              43.141294
dtype: float64
```

```
plt.figure(figsize=(12,6))
plot = train.target.plot(kind='hist',bins=10)

ax = plot.axes

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

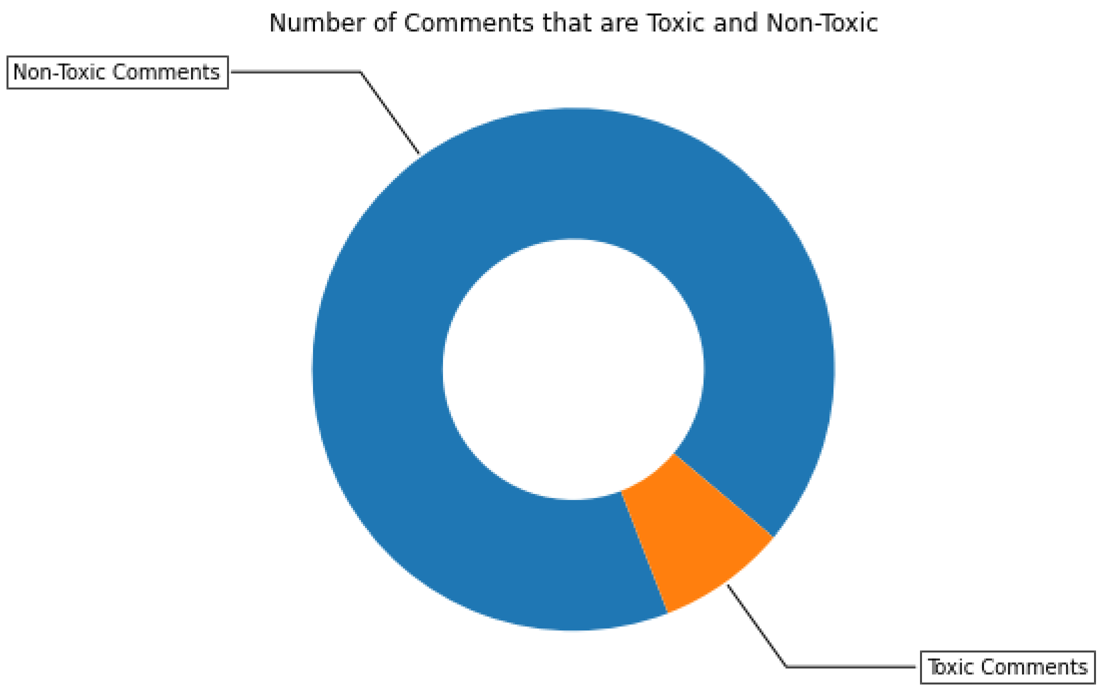


- **There are 70% non-toxic data points which are in the range of 0.0 to 0.1 and from above plo are non-toxic.**
- **We have to note that values which are less than 0.5 are non- toxic and greater than 0.5 are**

Lets assign binary values to target variable

```
def assign_class(target):
  '''this is for assigning class labels'''
    if target >= .5:
        return 1
    else:
        return 0

# we will create binary class column which will be our Y label
train['class'] = train.apply(lambda x: assign_class(x['target']), axis= 1)


class_specific_count = train['class'].value_counts()
print("Number of Non-Toxic comments ", class_specific_count[0],"(",((class_specific_count[0]/len(train))*100),"% )"
print("Number of Toxic comments ",class_specific_count[1], "(",((class_specific_count[1]/len(train))*100),"% )")
fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(aspect="equal"))
recipe = ["Non-Toxic Comments", "Toxic Comments"]
data = [class_specific_count[0], class_specific_count[1]]
wedges, texts = ax.pie(data, wedgeprops=dict(width=0.5), startangle=-40)
bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
kw = dict(xycoords='data', textcoords='data', arrowprops=dict(arrowstyle="-"),bbox=bbox_props, zorder=0, va=
for i, p in enumerate(wedges):
  ang = (p.theta2 - p.theta1)/2. + p.theta1
  y = np.sin(np.deg2rad(ang))
  x = np.cos(np.deg2rad(ang))
  horizontalalignment = {-1: "right", 1: "left"}[int(np.sign(x))]
  connectionstyle = "angle,angleA=0,angleB={}".format(ang)
  kw["arrowprops"].update({"connectionstyle": connectionstyle})
  ax.annotate(recipe[i], xy=(x, y), xytext=(1.35*np.sign(x), 1.4*y), horizontalalignment=horizontalalignment, **kw)
ax.set_title("Number of Comments that are Toxic and Non-Toxic")
plt.show()
```

```
Number of Non-Toxic comments  1660540 ( 92.00309827722046 % )
Number of Toxic comments  144334 ( 7.99690172277954 % )
```



Observation: we see that there are only 7% toxic data and 92% data is non-Toxic.Its clear that oui

Lets defining some categories of comments

```
#https://www.kaggle.com/kabure/simple-eda-hard-views-w-easy-code
etnics = ['asian' , 'latino' , 'black', 'white', 'other_race_or_ethnicity']

religions = ['atheist', 'buddhist', 'hindu', 'jewish', 'muslim', 'christian', 'other_religion']

sexual = ['female', 'male', 'other_gender']

sexual_orientation = ['heterosexual', 'bisexual', 'transgender', 'homosexual_gay_or_lesbian', 'other_sexual_orientat

disabilities = ['intellectual_or_learning_disability', 'physical_disability', 'psychiatric_or_mental_illness', 'other_disabi

reactions = ['funny', 'wow', 'sad', 'likes', 'disagree', 'sexual_explicit']


def bar_plot(features,title,xlabell):
  '''this function seperates toxic and non-toxic data and plots bar plot'''
  train_labeled_df = train.loc[:, ['target'] + features].dropna()
  toxic_df = train_labeled_df[train_labeled_df['target'] >= .5][features]
  non_toxic_df = train_labeled_df[train_labeled_df['target'] < .5][features]

  # at first, we just want to consider the identity tags in binary format. So if the tag is any value other than 0 we c
  toxic_count1 = toxic_df[features].where(train_labeled_df == 0, other = 1).sum()
  non_toxic_count1= non_toxic_df[features].where(train_labeled_df == 0, other = 1).sum()

  # now we can concat the two series together to get a toxic count vs non toxic count for each identity
```

```
toxic_vs_non_toxic = pd.concat([toxic_count1, non_toxic_count1], axis=1)
toxic_vs_non_toxic = toxic_vs_non_toxic.rename(index=str, columns={1: "non-toxic", 0: "toxic"})
# here we plot the stacked graph but we sort it by toxic comments to (perhaps) see something interesting
toxic_vs_non_toxic.sort_values(by='toxic').plot(kind='bar', stacked=False, figsize=(25,8), fontsize=20).legend(pro
plt.title(title, fontsize=26)
plt.ylabel('Counts', fontsize=26)
plt.xlabel(xlabell, fontsize=26)
plt.xticks(rotation=0)
```

Let's represent similarly the distribution of additional toxicity features.

```
bar_plot(etnics,"Ethnics Comments  by Toxic and Non-Toxic Classifivation",'Etnics')
```

From Etnics category white feature contains more toxic comments followed by black feature and

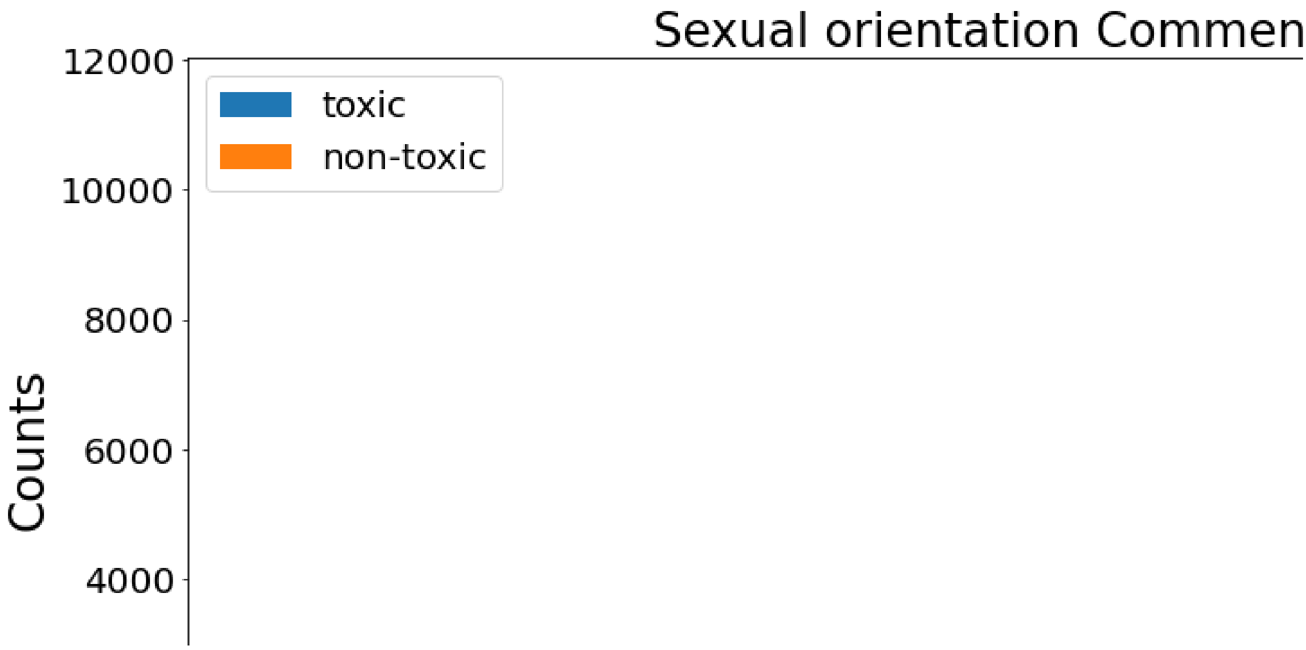
```
bar_plot(religions,"Religions Comments  by Toxic and Non-Toxic Classifivation",'Religions')
```

From Religions category muslim feature contains more toxic comments followed by cristian feat

```
bar_plot(sexual,"Sexual Comments  by Toxic and Non-Toxic Classifivation",'Sexual')
```

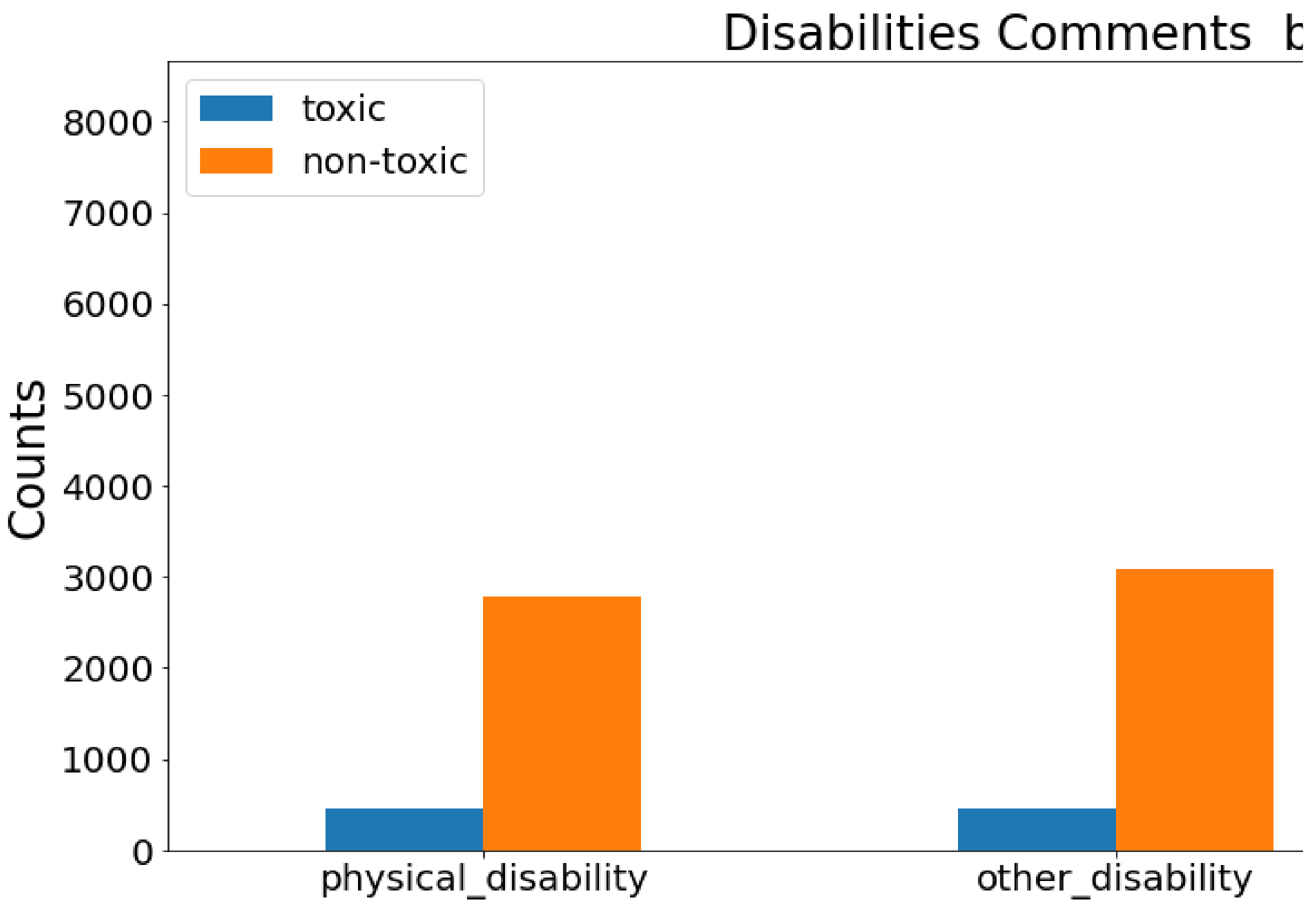
From Sexual category male feature contains more toxic comments followed by female feature

```
bar_plot(sexual_orientation,"Sexual orientation Comments  by Toxic and Non-Toxic Classifivation",'Sexual Orienta
```



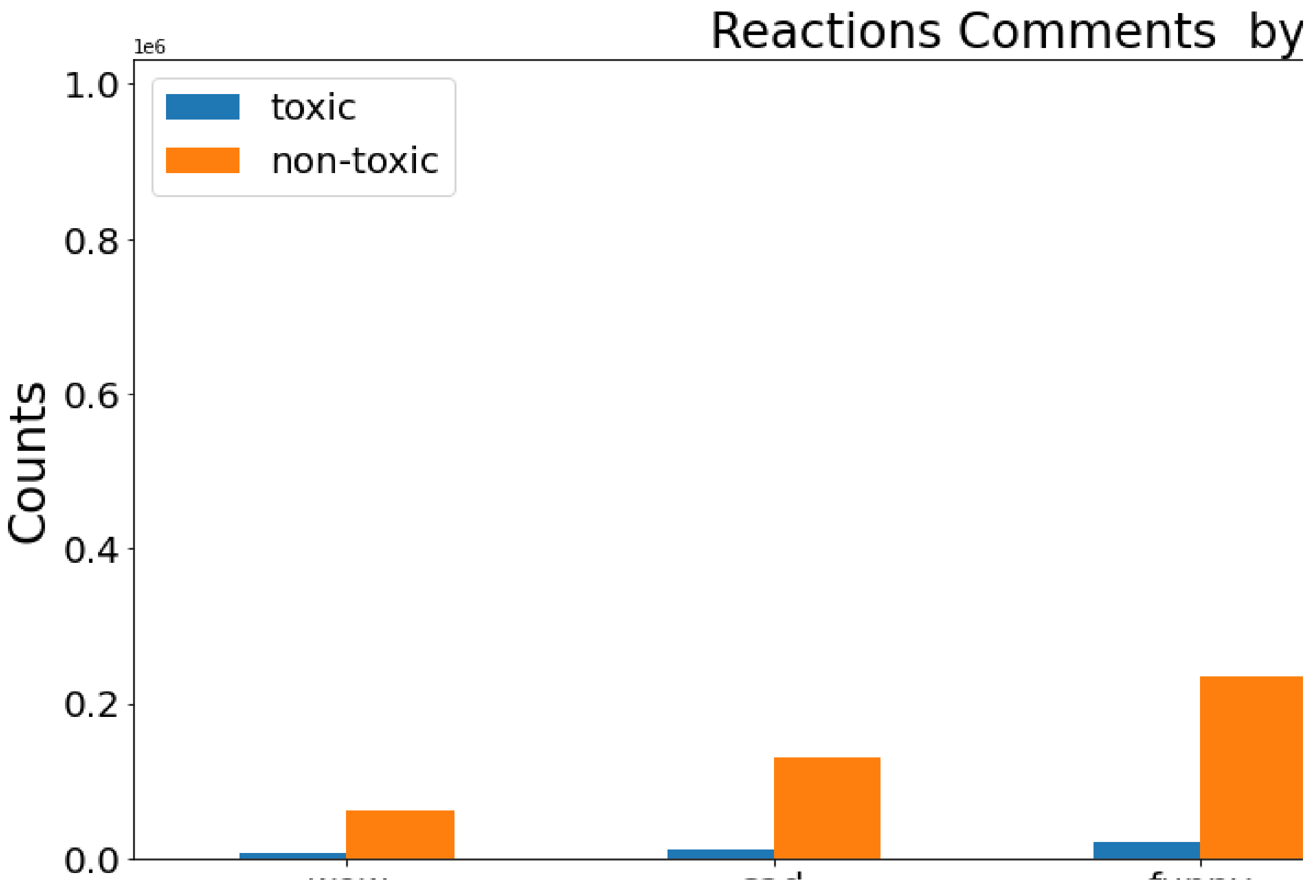
From sexual orientation category homosexual_gay_or_lesbian feature contains more toxic comm

```
bar_plot(disabilities,"Disabilities Comments  by Toxic and Non-Toxic Classifivation",'Disabilities')
```



From Disabilities category psychiatric_or_mental_illness feature contains more toxic comments f feature.

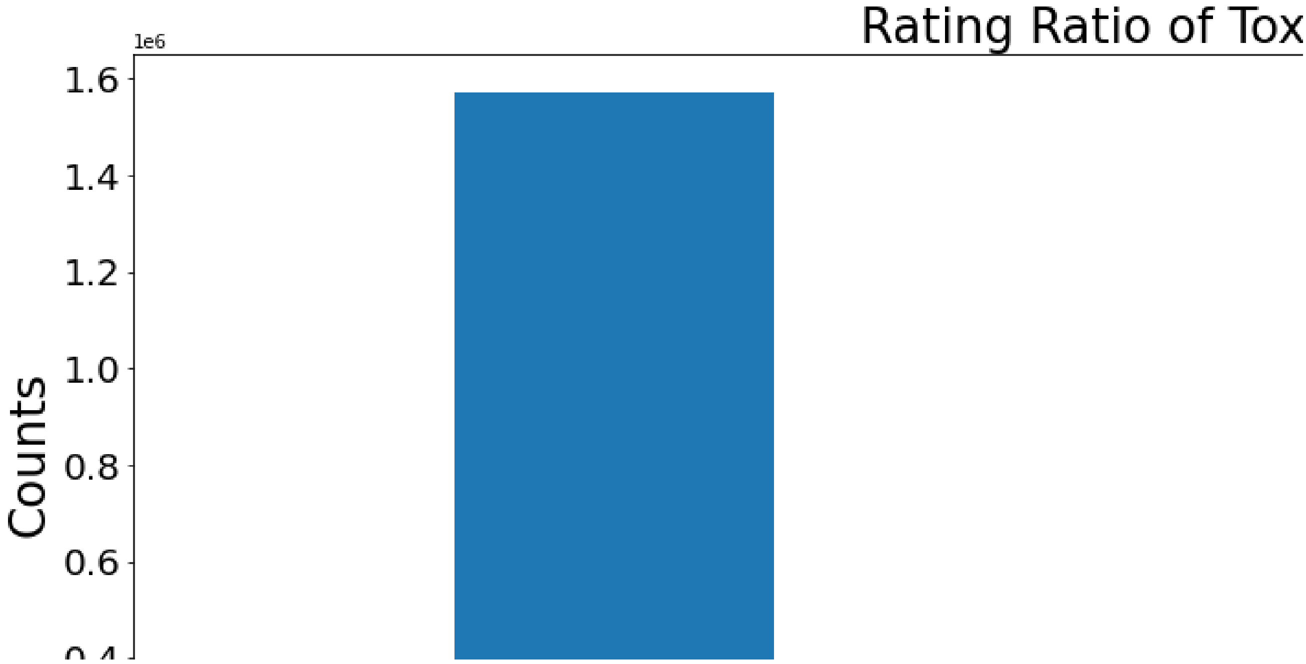
```
bar_plot(reactions,"Reactions Comments  by Toxic and Non-Toxic Classifivation",'Reactions')
```



From Reactions category likes feature contains more toxic comments followed by disagree featu

Rating distribution

```
def bar_plot(features,title,xlabell):
  '''this function seperate toxic and non-toxic data from rating feature'''
  train_labeled_df = train.loc[:, ['target'] + features].dropna()
  toxic_df = train_labeled_df[train_labeled_df['target'] >= .5][features]
  non_toxic_df = train_labeled_df[train_labeled_df['target'] < .5][features]

  counts=toxic_df['rating'].value_counts()
  counts1=non_toxic_df['rating'].value_counts()
  df=pd.DataFrame([["Approved",counts1[0],counts[0]],["Rejected",counts1[1],counts[1]]],columns=['comment', 'n
  # here we plot the stacked graph but we sort it by toxic comments to (perhaps) see something interesting
  df.plot(kind='bar', stacked=False, figsize=(25,8), fontsize=20).legend(prop={'size': 20})
  plt.title(title, fontsize=26)
  plt.ylabel('Counts', fontsize=26)
  plt.xlabel(xlabell, fontsize=26)
  plt.xticks([0,1],['Approved','Rejected'],rotation=0)
bar_plot(['rating'],'Rating Ratio of Toxic and Non-toxic Comments','Rating')
```



In Rating feature, approved category contains more toxic comments than rejected category

From above plots we saw toxic comments are more in white,black,muslim,cristian,male,female,ho psychiatric_or_mental_illness likes fetures so lets look into wordcloud of these features to know w between them.

Word Cloud

Let's show the wordcloud of frequent used words in the comments.

```
#https://www.kaggle.com/ekhtiar/unintended-eda-with-tutorial-notes

def generate_word_cloud(identity, toxic_comments, non_toxic_comments):
  '''this simple function is used  to generate the wordcloud per identity group'''
    # convert stop words to sets as required by the wordcloud library
    stop_words = set(stopwords.words("english"))
    # create toxic wordcloud
    wordcloud_toxic = WordCloud(max_font_size=100, max_words=100, background_color="black", stopwords=st
    # create non-toxic wordcloud
    wordcloud_non_toxic = WordCloud(max_font_size=100, max_words=100, background_color="black", stopwor
    # draw the two wordclouds side by side using subplot
    fig = plt.figure(figsize=[15,5])
    fig.add_subplot(1, 2, 1).set_title("Toxic Wordcloud", fontsize=26)
    plt.imshow(wordcloud_toxic, interpolation="bilinear")
    plt.axis("off")
    fig.add_subplot(1, 2, 2).set_title("Non Toxic Wordcloud", fontsize=26)
    plt.imshow(wordcloud_non_toxic, interpolation="bilinear")
    plt.axis("off")
    plt.subplots_adjust(top=0.85)
    plt.suptitle('Word Cloud - {} Feature'.format(identity), size = 26)
    plt.show()
```

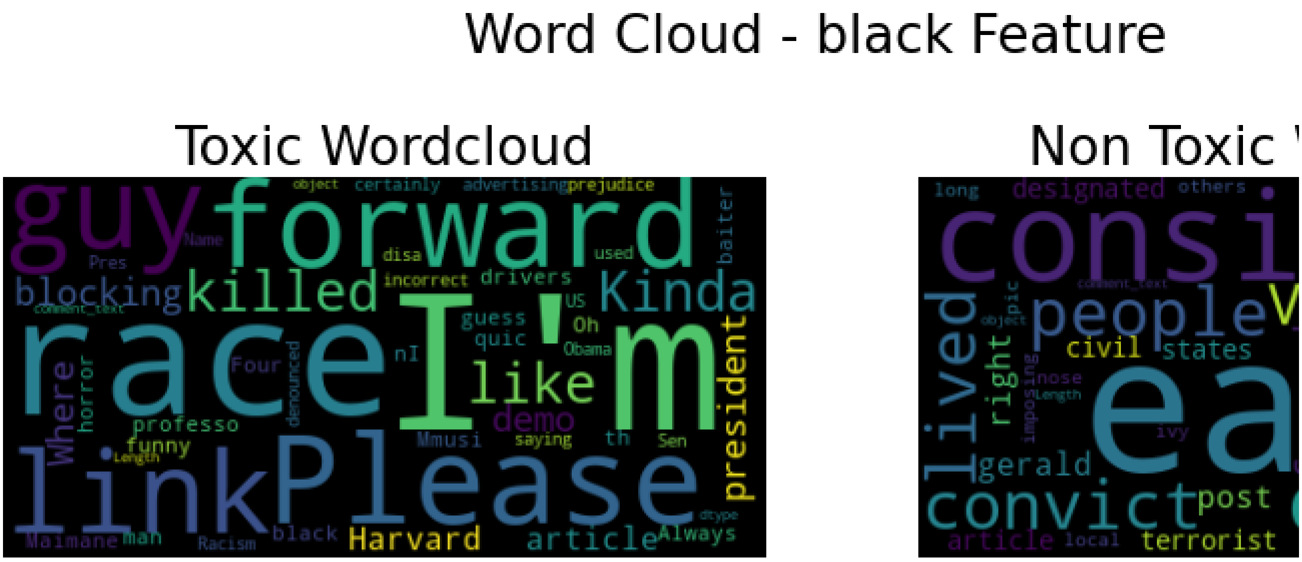
Lets start with white feature.

```
generate_word_cloud('white', train.loc[train['white'] > 0.5]['comment_text'].sample(20000), train.loc[train['white'] <
```

White feature:

- **In Toxic comments ,more frequent words are racist,Anti,male,white.**
- **In Non-toxic comments ,more frequent words are sore,latino,wow,another,eye.**

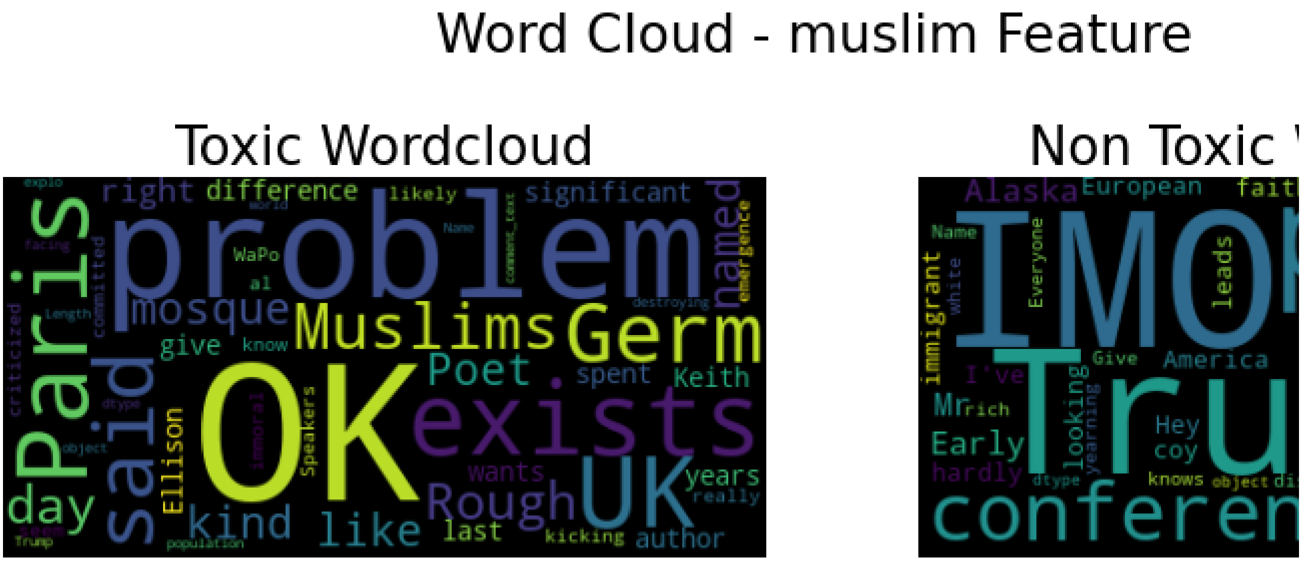
```
generate_word_cloud('black', train.loc[train['black'] > 0.5]['comment_text'].sample(10000), train.loc[train['black'] <
```



Black feature:

- **In Toxic comments ,more frequent words are race,forward,please,link.**
- **In Non-toxic comments ,more frequent words are considered,early,cost,vegas,convict.**

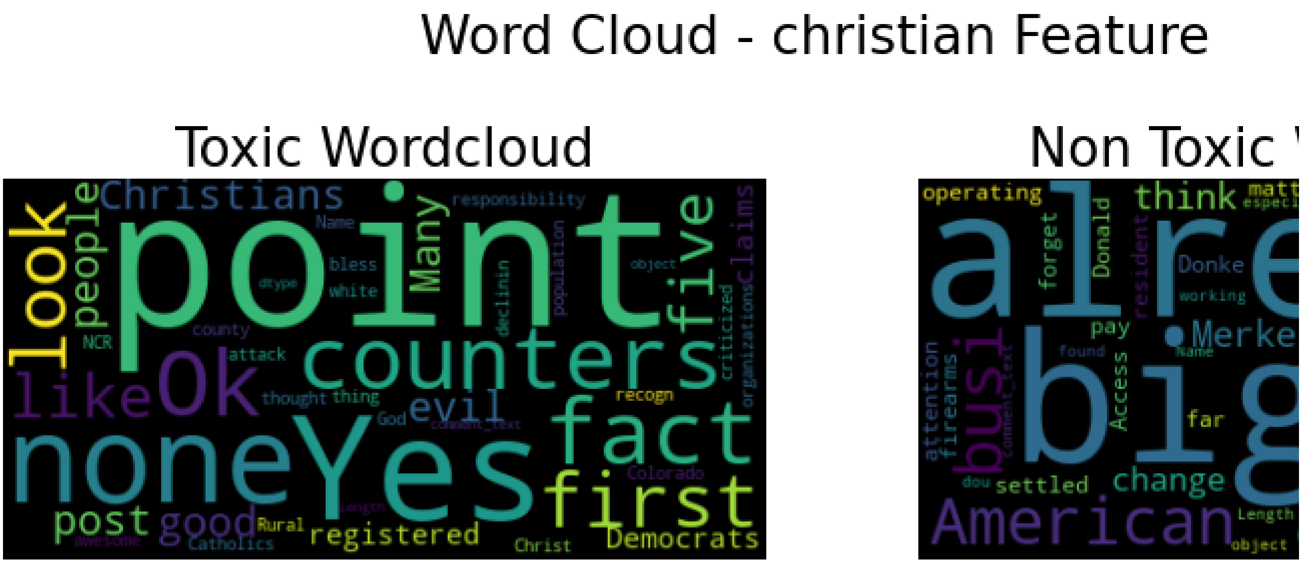
```
generate_word_cloud('muslim', train.loc[train['muslim'] > 0.5]['comment_text'].sample(10000), train.loc[train['mus
```



muslim feature:

- **In Toxic comments ,more frequent words are problem,ok,muslim,exists,paris,said,uk.**
- **In Non-toxic comments ,more frequent words are Trump,IMO,press,national,see,ck.**

```
generate_word_cloud('christian', train.loc[train['christian'] > 0.5]['comment_text'].sample(10000), train.loc[train['ch
```



Christian feature:

- **In Toxic comments ,more frequent words are point,yes,none,counters,fact,first,people.**
- **In Non-toxic comments ,more frequent words are already,big,observations,figures,perhaps.**

```
generate_word_cloud('male', train.loc[train['male'] > 0.5]['comment_text'].sample(10000), train.loc[train['male'] < (
```

male feature:

- **In Toxic comments ,more frequent words are men,male,say,female,ever,duc.**
- **In Non-toxic comments ,more frequent words are sacred,phoenix,documented,arizona.**

```
generate_word_cloud('female', train.loc[train['female'] > 0.5]['comment_text'].sample(10000), train.loc[train['femal
```

female feature:

- **In Toxic comments ,more frequent words are woman,care,less,sport,thinking,excuses,jude.**
- **In Non-toxic comments ,more frequent words are Every,Trump,black,wanted,safe.**

```
generate_word_cloud('homosexual_gay_or_lesbian', train.loc[train['homosexual_gay_or_lesbian'] > 0.5]['comment
```

homosexual_gay_or_lesbian feature:

- **In Toxic comments ,more frequent words are Ellis,Johnny,hard,choice,knew.**
- **In Non-toxic comments ,more frequent words are entire,nothing,comment,one,believe,cons**

```
generate_word_cloud('psychiatric_or_mental_illness', train.loc[train['psychiatric_or_mental_illness'] > 0.5]['commer
```

psychiatric_or_mental_illness feature:

- **In Toxic comments ,more frequent words are health,mental,free,blame,rant.**
- **In Non-toxic comments ,more frequent words are exactly,guess,sure,misplaced,claim.**

```
generate_word_cloud('likes', train.loc[train['likes'] > 0.5]['comment_text'].sample(9000), train.loc[train['likes'] < 0.5
```

likes feature:

- **In Toxic comments ,more frequent words are let,read,yes,go,sign,solved,calling.**
- **In Non-toxic comments ,more frequent words are bear,brown,think,electroc,anything.**

Text Preprocessing

- Convert to lower case
- Clean contractions
- Clean special charactor
- Convert small caps

```
contraction_mapping = {
    "ain't": "is not", "aren't": "are not","can't": "cannot", "'cause": "because", "could've": "could have", "couldn't": "cc
    "didn't": "did not",  "doesn't": "does not", "don't": "do not", "hadn't": "had not", "hasn't": "has not", "haven't": "h
    "he'd": "he would","he'll": "he will", "he's": "he is", "how'd": "how did", "how'd'y": "how do you", "how'll": "how v
    "I'd": "I would", "I'd've": "I would have", "I'll": "I will", "I'll've": "I will have","I'm": "I am", "I've": "I have", "i'd": "i wc
    "i would have", "i'll": "i will",  "i'll've": "i will have","i'm": "i am", "i've": "i have", "isn't": "is not", "it'd": "it would",
    "it'd've": "it would have", "it'll": "it will", "it'll've": "it will have","it's": "it is", "let's": "let us", "ma'am": "madam",
    "mayn't": "may not", "might've": "might have","mightn't": "might not","mightn't've": "might not have", "must've
    "mustn't": "must not", "mustn't've": "must not have", "needn't": "need not", "needn't've": "need not have",
    "o'clock": "of the clock", "oughtn't": "ought not", "oughtn't've": "ought not have", "shan't": "shall not",
    "sha'n't": "shall not", "shan't've": "shall not have", "she'd": "she would", "she'd've": "she would have",
    "she'll": "she will", "she'll've": "she will have", "she's": "she is", "should've": "should have", "shouldn't": "should n
    "shouldn't've": "should not have", "so've": "so have","so's": "so as", "this's": "this is","that'd": "that would",
    "that'd've": "that would have", "that's": "that is", "there'd": "there would", "there'd've": "there would have", "the
    "here's": "here is","they'd": "they would", "they'd've": "they would have", "they'll": "they will", "they'll've": "they v
    "they're": "they are", "they've": "they have", "to've": "to have", "wasn't": "was not", "we'd": "we would", "we'd've
    "we'll": "we will", "we'll've": "we will have", "we're": "we are", "we've": "we have", "weren't": "were not", "what'll":
    "what'll've": "what will have", "what're": "what are",  "what's": "what is", "what've": "what have", "when's": "wher
    "when've": "when have", "where'd": "where did", "where's": "where is", "where've": "where have", "who'll": "who
    "who's": "who is", "who've": "who have", "why's": "why is", "why've": "why have", "will've": "will have", "won't": "v
    "won't've": "will not have", "would've": "would have", "wouldn't": "would not", "wouldn't've": "would not have",
    "y'all": "you all", "y'all'd": "you all would","y'all'd've": "you all would have","y'all're": "you all are","y'all've": "you a
    "you'd": "you would", "you'd've": "you would have", "you'll": "you will", "you'll've": "you will have", "you're": "yo
    "Trump's": "trump is", "Obama's": "obama is", "Canada's": "canada is", "today's": "today is"
}


#https://stackoverflow.com/questions/11331982/how-to-remove-any-url-within-a-string-in-python
stop_words = set(stopwords.words('english'))
def clean_text(text):
  '''this for preprocessing text feature'''
  text=re.sub(r'http\S+', 'link', text)
  text = ' '.join(contraction_mapping[word] if word in contraction_mapping else word for word in text.split(" "))
  #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
  text = re.sub("\S*\d\S*", "", text).strip()
  text=emoji.demojize(text, delimiters=(" ", ""))
  text = re.sub('[^A-Za-z]+', ' ', text)
  text = ' '.join(e for e in text.split() if e not in stop_words)
```

```
    if len(text)<2:
      text='unknown'
    return text.lower().strip()

train["clean_text"] = train["comment_text"].progress_apply(lambda text: clean_text(text))
test["clean_text"] = test["comment_text"].progress_apply(lambda text: clean_text(text))
```

```
100%    1804874/1804874 [46:33<00:00, 646.20it/s]

100%    97320/97320 [03:26<00:00, 471.39it/s]
```

```
#pickle.dump( train , open( "/content/drive/My Drive/jigsaw/trpreprocessed", "wb" ))
#pickle.dump( test, open( "/content/drive/My Drive/jigsaw/tepreprocessed", "wb" ))
train_processed=pickle.load(open( "/content/drive/My Drive/jigsaw/trpreprocessed", "rb" ))
test_processed=pickle.load(open( "/content/drive/My Drive/jigsaw/tepreprocessed", "rb" ))
```

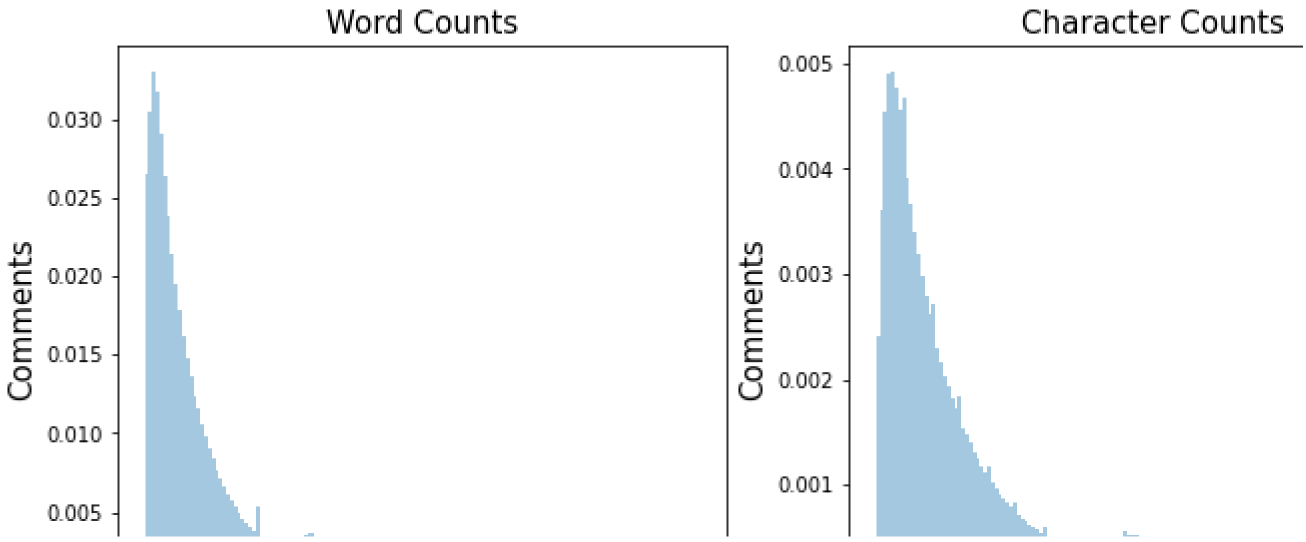
Feature Engineering

```
#is to add word count of comment data
train_processed['comment_word_count'] = train_processed.clean_text.apply(lambda x: len(x.split()))
test_processed['comment_word_count'] = test_processed.clean_text.apply(lambda x: len(x.split()))

#is to add word count of comment data
train_processed['comment_char_count'] = train_processed.clean_text.apply(lambda x: len(x))
test_processed['comment_char_count'] = test_processed.clean_text.apply(lambda x: len(x))
```

Univariate Analysis: comment_word_count and comment_ch

```
#How to calculate number of words in a string in DataFrame: https://stackoverflow.com/a
plt.figure(figsize=[12,5])

plt.subplot(1, 2, 1).set_title("Word Counts", fontsize=15)
sns.distplot(train_processed['comment_word_count'], kde=False, bins=150, label='Train word count', norm_hist=1
plt.ylabel('Comments', fontsize=15)
plt.xlabel('Number of words in comments', fontsize=15)

plt.subplot(1, 2, 2).set_title("Character Counts", fontsize=15)
sns.distplot(train_processed['comment_char_count'], kde=False, bins=150, label='Train char count', norm_hist=Tru
plt.ylabel('Comments', fontsize=15)
plt.xlabel('Number of characters in comments', fontsize=15)
```

Text(0.5, 0, 'Number of characters in comments')



Observation : We can see that most of the words have length starting from 1 to 120 and character

Lets see real counts of both the features as follows

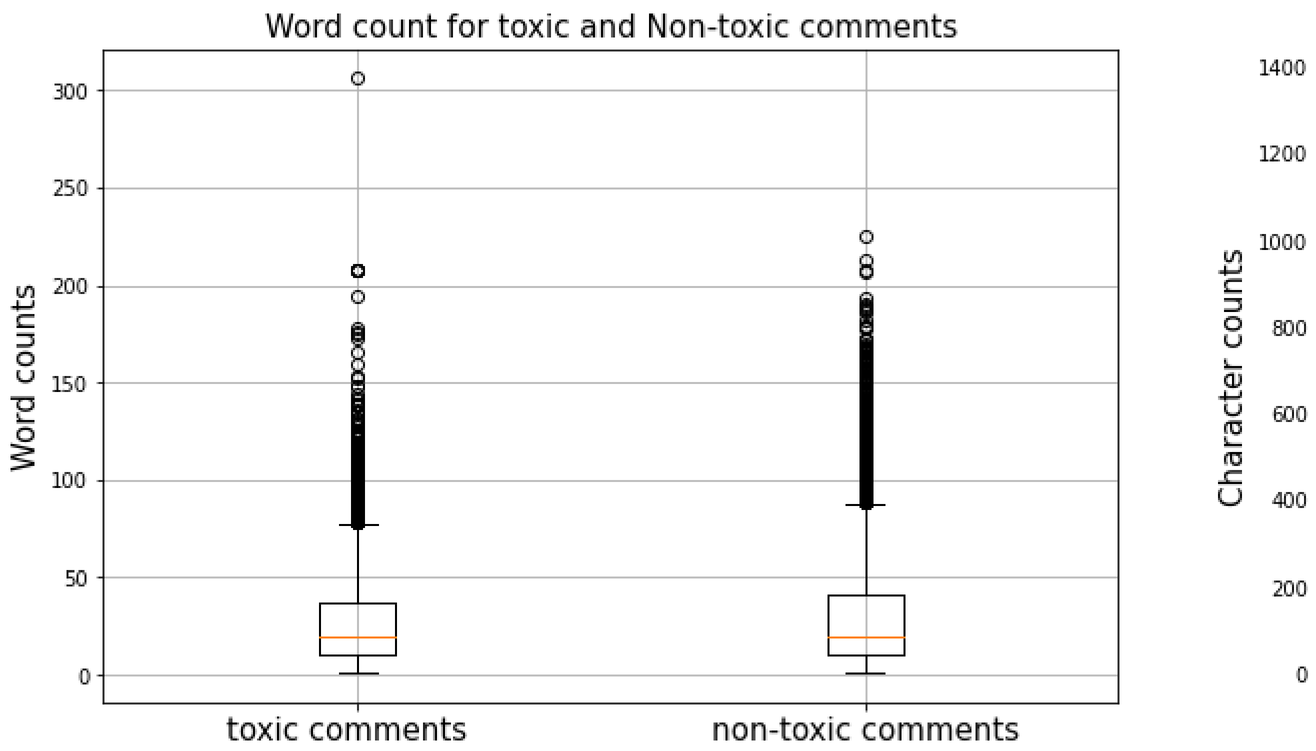
```
print("\n Maximum length of words :",train_processed['comment_word_count'].max() )
print("\n Minimum length of words :",train_processed['comment_word_count'].min() )
print("\n-----------------------------------")
print("\n Maximum length of characters :",train_processed['comment_char_count'].max() )
print("\n Minimum length of characters :",train_processed['comment_char_count'].min() )
```

```
 Maximum length of words : 306

 Minimum length of words : 1

-----------------------------------

 Maximum length of characters : 1372

 Minimum length of characters : 2
```

Boxplot

```
toxic_word_count1 = train_processed[train_processed['class']==1]['comment_word_count'].values
non_toxic_word_count1 = train_processed[train_processed['class']==0]['comment_word_count'].values

plt.figure(figsize=(20,6))
plt.subplot(1,2,1)
#getting boxplot based on word count of each class.
plt.boxplot([toxic_word_count1, non_toxic_word_count1])
plt.xticks([1,2],('toxic comments  ','non-toxic comments'), fontsize=15)
plt.ylabel('Word counts', fontsize=15)
plt.title('Word count for toxic and Non-toxic comments', fontsize=15)
plt.grid()
```

```
toxic_char_count2 = train_processed[train_processed['class']==1]['comment_char_count'].values
non_toxic_char_count2 = train_processed[train_processed['class']==0]['comment_char_count'].values
plt.subplot(122)
plt.boxplot([toxic_char_count2, non_toxic_char_count2])
plt.xticks([1,2],('toxic comments  ','non-toxic comments'), fontsize=15)
plt.ylabel('Character counts',fontsize=15)
plt.title('Character count for toxic and Non-toxic comments', fontsize=15)
plt.grid()
```



Observation

- In first plot,Word counts of toxic and non toxic comments are overlapping.so we cant disting
- In second plot, character counts are also overlapping but we can say that 75% of non-toxic c

Lets see real picture of above plot

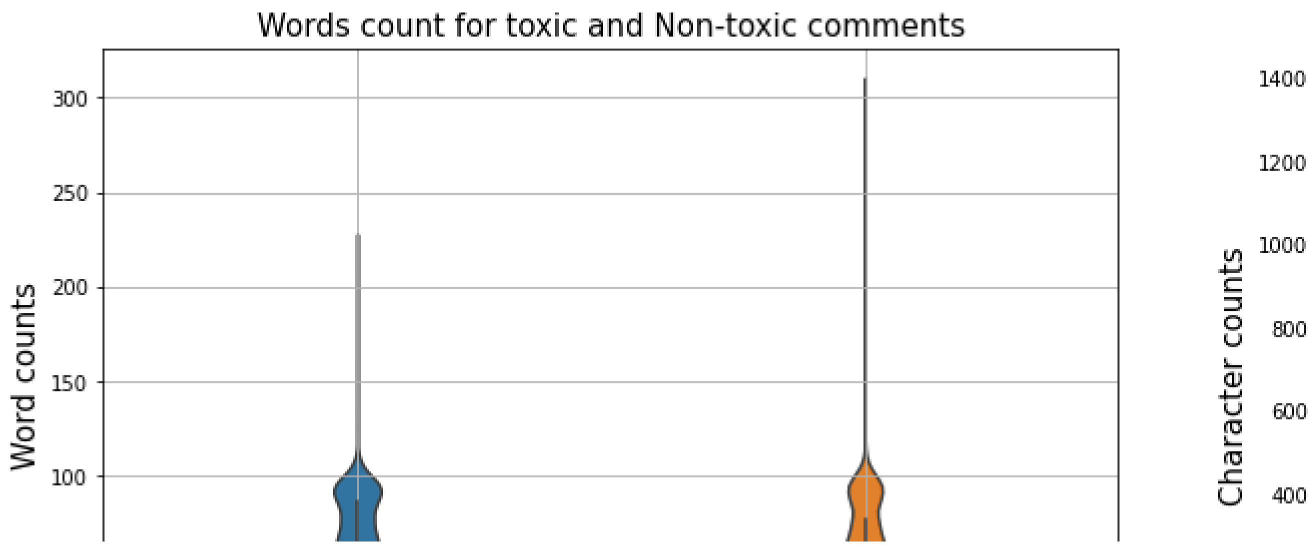
```
print("Toxic Word counts:")
print("\n Maximum length of words :",toxic_word_count1.max() )
print("\n Minimum length of words :",toxic_word_count1.min() )
print("-----------------------------------")
print("Non-Toxic Word counts:")
print("\n Maximum length of characters :",non_toxic_word_count1.max() )
print("\n Minimum length of characters :",non_toxic_word_count1.min() )
print("=======================")
print("Toxic Char counts:")
print("\n Maximum length of words :",toxic_char_count2.max() )
print("\n Minimum length of words :",toxic_char_count2.min() )
print("-----------------------------------")
print("Non-Toxic Char counts:")
```

```
print("\n Maximum length of characters :",non_toxic_char_count2.max() )
print("\n Minimum length of characters :",non_toxic_char_count2.min() )
```

```
Toxic Word counts:

 Maximum length of words : 306

 Minimum length of words : 1
-----------------------------------
Non-Toxic Word counts:

 Maximum length of characters : 225

 Minimum length of characters : 1
=======================
Toxic Char counts:

 Maximum length of words : 1372

 Minimum length of words : 2
-----------------------------------
Non-Toxic Char counts:

 Maximum length of characters : 1372

 Minimum length of characters : 2
```

Violin Plot

```
#getting violin plot on train data with comment_word_countfeature

plt.figure(figsize=(20,6))
plt.subplot(1,2,1)
sns.violinplot(x = 'class', y = 'comment_word_count', data = train_processed[0:])
plt.xticks([0,1],('toxic comments  ','non-toxic comments'), fontsize=15)
plt.ylabel('Word counts', fontsize=15)
plt.title('Words count for toxic and Non-toxic comments', fontsize=15)
plt.grid()

plt.subplot(1,2,2)
sns.violinplot(x = 'class', y = 'comment_char_count', data = train_processed[0:])
plt.xticks([0,1],('toxic comments  ','non-toxic comments'), fontsize=15)
plt.ylabel('Character counts',fontsize=15)
plt.title('Character count for toxic and Non-toxic comments', fontsize=15)
plt.grid()
```

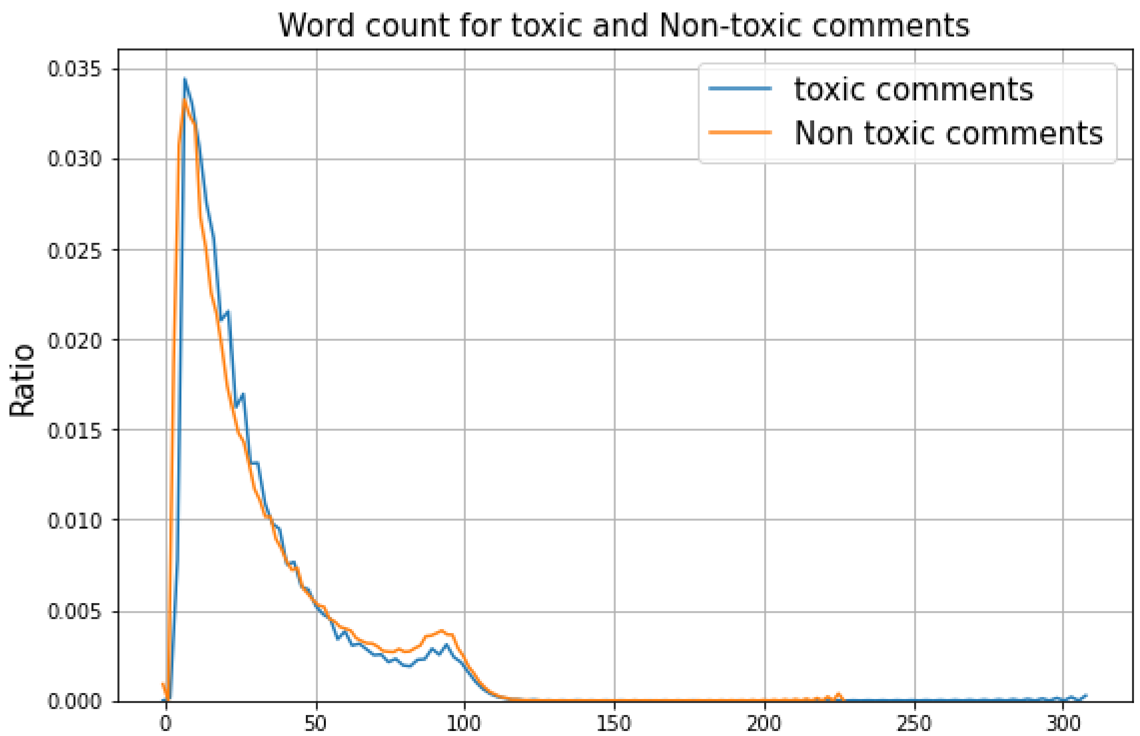


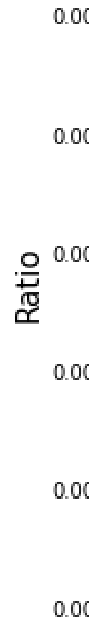
Observation

- This plot giving more clear picture than box plot which says that word count distribution is ve
  distribution is peaked at count 50.
- It means there are more comments with nearly 10 words and 50 characters.
- Distributions of toxic and non-toxic comments are almost same.

Kernel Density Estimate Plot

```
toxic_word_count1 = train_processed[train_processed['class']==1]['comment_word_count'].values
non_toxic_word_count1 = train_processed[train_processed['class']==0]['comment_word_count'].values

plt.figure(figsize=(20,6))
plt.subplot(1,2,1)
sns.kdeplot(toxic_word_count1,label="toxic comments", bw=0.6)
sns.kdeplot(non_toxic_word_count1,label="Non toxic comments", bw=0.6)
plt.legend(fontsize=15)
plt.xlabel('Number of Words', fontsize=15)
plt.ylabel('Ratio', fontsize=15)
plt.title('Word count for toxic and Non-toxic comments', fontsize=15)
plt.grid()


toxic_char_count2 = train_processed[train_processed['class']==1]['comment_char_count'].values
non_toxic_char_count2 = train_processed[train_processed['class']==0]['comment_char_count'].values
plt.subplot(122)
sns.kdeplot(toxic_char_count2,label="toxic comments", bw=0.6)
sns.kdeplot(non_toxic_char_count2,label="Non toxic comments", bw=0.6)
plt.legend(fontsize=15)
plt.xlabel('Number of characters', fontsize=15)
plt.ylabel('Ratio',fontsize=15)
plt.title('Character count for toxic and Non-toxic comments', fontsize=15)
plt.grid()
```





Observation

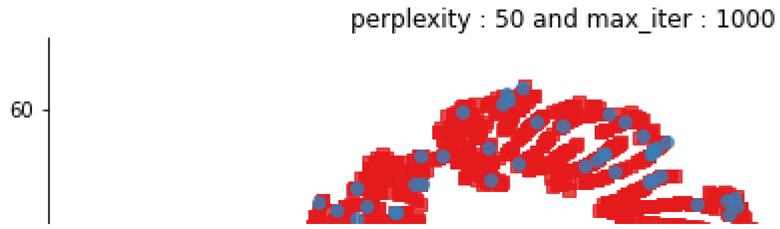
- PDF of both toxic and Non-toxic comments is overlapping,so its hard to distinguish results.
- Vey high peek of both the pdf's has been seen near word count with 10 and character count

Visualization on train data

```
dfp_subsampled = train[0:5000]
X = MinMaxScaler().fit_transform(dfp_subsampled[['comment_word_count','comment_char_count']])
y = dfp_subsampled['class'].values

tsne2d = TSNE(
    n_components=2,
    perplexity=50,
    init='random', # pca
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
    angle=0.5
).fit_transform(X)
```

```
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.002s...
[t-SNE] Computed neighbors for 5000 samples in 0.127s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.003408
[t-SNE] Computed conditional probabilities in 0.718s
[t-SNE] Iteration 50: error = 72.4004135, gradient norm = 0.0295096 (50 iterations in 1.603s)
[t-SNE] Iteration 100: error = 61.1872292, gradient norm = 0.0104540 (50 iterations in 1.280s)
[t-SNE] Iteration 150: error = 58.3941269, gradient norm = 0.0054704 (50 iterations in 1.233s)
[t-SNE] Iteration 200: error = 56.9852371, gradient norm = 0.0040134 (50 iterations in 1.199s)
[t-SNE] Iteration 250: error = 56.1403809, gradient norm = 0.0030811 (50 iterations in 1.237s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 56.140381
[t-SNE] Iteration 300: error = 1.1669676, gradient norm = 0.0013977 (50 iterations in 1.395s)
[t-SNE] Iteration 350: error = 0.7491656, gradient norm = 0.0005055 (50 iterations in 1.443s)
[t-SNE] Iteration 400: error = 0.5812745, gradient norm = 0.0002911 (50 iterations in 1.405s)
[t-SNE] Iteration 450: error = 0.5015921, gradient norm = 0.0001858 (50 iterations in 1.407s)
[t-SNE] Iteration 500: error = 0.4548135, gradient norm = 0.0001451 (50 iterations in 1.401s)
[t-SNE] Iteration 550: error = 0.4287824, gradient norm = 0.0001253 (50 iterations in 1.412s)
[t-SNE] Iteration 600: error = 0.4118314, gradient norm = 0.0001145 (50 iterations in 1.377s)
```

```
df = pd.DataFrame({'x':tsne2d[:,0], 'y':tsne2d[:,1] ,'label':y})

# draw the plot in appropriate place in the grid
sns.lmplot(data=df, x='x', y='y', hue='label', fit_reg=False, size=8,palette="Set1",markers=['s','o'])
plt.title("perplexity : {} and max_iter : {}".format(50, 1000))
plt.show()
```



As we can see that non-toxic comments are somewhere seperated from toxic comments.

Train and Cv Split

```
IDENTITY_COLUMNS = ['male', 'female', 'homosexual_gay_or_lesbian', 'christian', 'jewish', 'muslim', 'black', 'white',
TARGET_COLUMN = 'target'
for column in IDENTITY_COLUMNS + [TARGET_COLUMN]:
    train_processed[column] = np.where(train_processed[column] >= 0.5, True, False)

Y = train_processed['class'].values
X_train, X_cv, Y_train, Y_cv = train_test_split(train_processed, Y,stratify=Y, test_size=0.2,random_state=42)
print("shape of train data :",X_train.shape,Y_train.shape)
print("shape of cv data : ",X_cv.shape,Y_cv.shape)
```

```
shape of train data : (1443899, 49) (1443899,)
shape of cv data :  (360975, 49) (360975,)
```

Make Data Model Ready: Encoding numerical, text feature

Encoding numerical feature: comment_word_count

```
normalizer = Normalizer()
normalizer.fit(X_train['comment_word_count'].values.reshape(1,-1))
X_train_word_count_norm = (normalizer.transform(X_train['comment_word_count'].values.reshape(1,-1))).transpo
X_cv_word_count_norm = (normalizer.transform(X_cv['comment_word_count'].values.reshape(1,-1))).transpose()
test_word_count_norm = (normalizer.transform(test_processed['comment_word_count'].values.reshape(1,-1))).tra
print("After vectorizations")
print(X_train_word_count_norm.shape, Y_train.shape)
print(X_cv_word_count_norm.shape, Y_cv.shape)
print(test_word_count_norm.shape)
print("="*100)
```

```
After vectorizations
(1443899, 1) (1443899,)
(360975, 1) (360975,)
(97320, 1)
================================================================================
```

Encoding numerical feature: comment_char_count

```
normalizer = Normalizer()
normalizer.fit(X_train['comment_char_count'].values.reshape(1,-1))
X_train_char_count_norm = (normalizer.transform(X_train['comment_char_count'].values.reshape(1,-1))).transpose
X_cv_char_count_norm = (normalizer.transform(X_cv['comment_char_count'].values.reshape(1,-1))).transpose()
test_char_count_norm = (normalizer.transform(test_processed['comment_char_count'].values.reshape(1,-1))).trans
print("After vectorizations")
print(X_train_char_count_norm.shape, Y_train.shape)
print(X_cv_char_count_norm.shape, Y_cv.shape)
print(test_char_count_norm.shape)
print("="*100)
```

```
After vectorizations
(1443899, 1) (1443899,)
(360975, 1) (360975,)
(97320, 1)
==========================================================================
```

Encoding text feature: comment_text

- I have tried below approches by training model and checked model score but max_features=
- type1=fidfVectorizer(ngram_range=(1,1),min_df=3, max_df=0.9, strip_accents='unicode', use_
- type2=fidfVectorizer(ngram_range=(1,2),min_df=3, max_df=0.9, strip_accents='unicode', use_
- type 3: TfidfVectorizer(ngram_range=(1,2),max_features=1500000)
- type 4: TfidfVectorizer(ngram_range=(1,2),max_features=150000)
- type 5: TfidfVectorizer(ngram_range=(1,1))
- type 6:TfidfVectorizer(max_features=50000)
- type 7: TfidfVectorizer(max_features=23075)
- type 8: TfidfVectorizer(max_features=76918)
- type 9:TfidfVectorizer(max_features=115377)

```
vectorizer =  TfidfVectorizer(max_features=76918)
vectorizer.fit(X_train['clean_text'].values)
X_train_comment_tfidf = vectorizer.transform(X_train['clean_text'].values)
X_cv_comment_tfidf = vectorizer.transform(X_cv['clean_text'].values)
test_comment_tfidf = vectorizer.transform(test_processed['clean_text'].values)
```

```
from scipy.sparse import hstack
#concatenate numerical and categorical features
x_tr = hstack((X_train_word_count_norm,X_train_char_count_norm,X_train_comment_tfidf)).tocsr()
x_cv = hstack((X_cv_word_count_norm,X_cv_char_count_norm,X_cv_comment_tfidf)).tocsr()
x_te = hstack(( test_word_count_norm,test_char_count_norm ,test_comment_tfidf)).tocsr()
```

Machine Learning Models

Metrics definition

```
#https://www.kaggle.com/dborkan/benchmark-kernel
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]]
    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)


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)


def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-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))
    return t


def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions


def error_plot(alpha,auc_array_train,auc_array_cv):
  plt.plot(alpha, auc_array_train, label='Train AUC')
  plt.plot(alpha, auc_array_cv, label='CV AUC')
  plt.scatter(alpha, auc_array_train, label='Train AUC points')
  plt.scatter(alpha, auc_array_cv, label='CV AUC points')
  plt.legend()
  plt.xlabel("alpha: hyperparameter")
  plt.ylabel("AUC")
  plt.title("ERROR PLOTS")
  plt.grid()
  plt.show()


def roc_curve_plot(Y_train,y_train_pred,Y_cv,y_cv_pred):
  train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_pred)
  cv_fpr, cv_tpr, cv_thresholds = roc_curve(Y_cv, y_cv_pred)

  plt.title('Receiver Operating Characteristic Curve')
  plt.plot(train_fpr, train_tpr,'b', label="Train AUC =%0.2f" % auc(train_fpr, train_tpr))
  plt.plot(cv_fpr, cv_tpr,'r', label="Test AUC =%0.2f" % auc(cv_fpr, cv_tpr))

  plt.legend(loc = 'lower right')
```

```
  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.grid()
  plt.show()

  print("="*100)
  best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
  plt.figure(figsize=(10,6))
  plt.title('Train confusion matrix', fontsize=15)
  cf1=confusion_matrix(Y_train, predict_with_best_t(y_train_pred, best_t))
  sn.heatmap(cf1, annot=True, fmt="d")

  cf2=confusion_matrix(Y_cv, predict_with_best_t(y_cv_pred, best_t))
  plt.figure(figsize = (10,6))
  plt.title('CV confusion matrix', fontsize=15)
  sn.heatmap(cf2, annot=True,fmt="d")
```

Machine Learning Model 1: Logistic Regression

Hyper parameter tuning

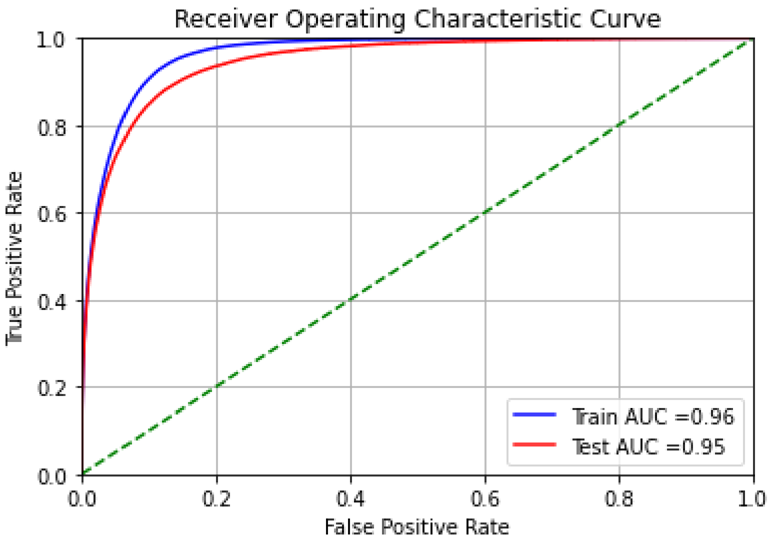
```
alpha = [10 ** x for x in range(-7, -3)]
auc_array_train=[]
auc_array_cv=[]
for i in alpha:
  clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42,class_weight='balanced')
  clf.fit(x_tr, Y_train)

  sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
  sig_clf.fit(x_tr, Y_train)

  predict_y_train = sig_clf.predict_proba(x_tr)[:,1]
  predict_y_cv = sig_clf.predict_proba(x_cv)[:,1]

  auc_array_train.append(roc_auc_score(Y_train, predict_y_train))
  auc_array_cv.append(roc_auc_score(Y_cv, predict_y_cv))
  print('For values of alpha = ', i, "The auc score on CV is:",roc_auc_score(Y_cv, predict_y_cv))

error_plot(alpha,auc_array_train,auc_array_cv)
```

```
For values of alpha =  1e-07 The auc score on CV is: 0.9396586321126278
For values of alpha =  1e-06 The auc score on CV is: 0.9476303773747572
For values of alpha =  1e-05 The auc score on CV is: 0.9422021511049676
For values of alpha =  0.0001 The auc score on CV is: 0.9145650900568598
```


```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
clf = SGDClassifier(alpha=0.000001, penalty='l2', loss='log', random_state=42,class_weight='balanced')
clf.fit(x_tr, Y_train)

sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_tr, Y_train)

y_train_pred = sig_clf.predict_proba(x_tr)[:,1]
y_cv_pred = sig_clf.predict_proba(x_cv)[:,1]
```

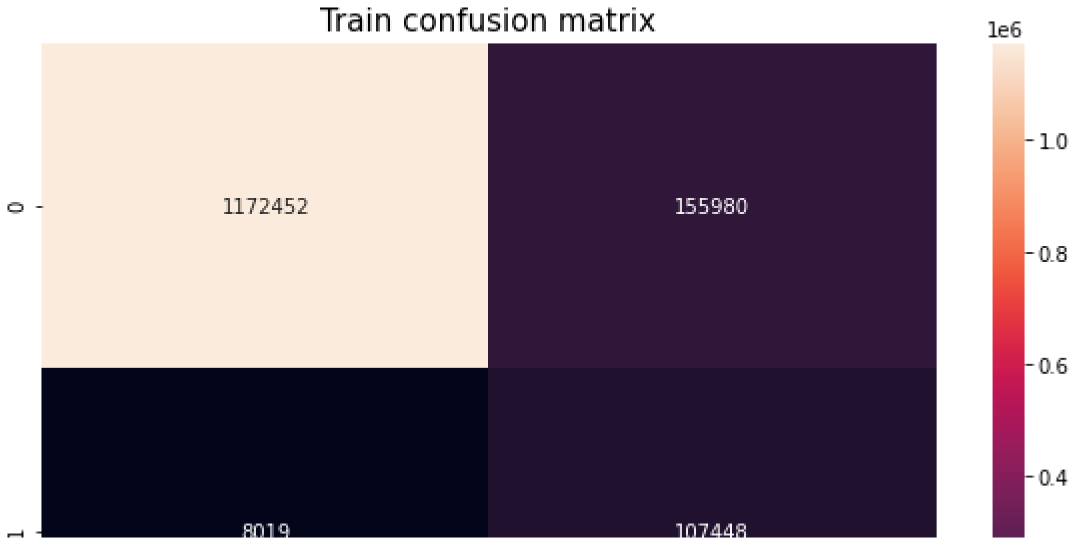
Firstly, we check how our model is performing using ROC cu

```
roc_curve_plot(Y_train,y_train_pred,Y_cv,y_cv_pred)
```



===

the maximum value of tpr*(1-fpr) 0.8212893609196633 for threshold 0.069



Now we use custom metric designed in our kaggle competit

```
MODEL_NAME = 'LR_model'
X_cv[MODEL_NAME] = y_cv_pred
```

```
X_cv.head(2)
```

	id	target	comment_text	severe_toxicity	obscene	identity_attack	insult	thi
1538593	6005154	False	So no O-line, no running game, no TE (or slot ...	0.0	0.0	0.0	0.0	
495446	851365	False	Canuckistan	0.0	0.0	0.1	0.0	

```
bias_metrics_df = compute_bias_metrics_for_model(X_cv, IDENTITY_COLUMNS, MODEL_NAME, TARGET_COLUMN
bias_metrics_df
```

	subgroup	subgroup_size	subgroup_auc	bpsn_auc	bnsp_auc
2	homosexual_gay_or_lesbian	2163	0.782008	0.760467	0.962429
6	black	3079	0.787232	0.745137	0.967607
7	white	5018	0.817864	0.772038	0.967427
5	muslim	4209	0.823175	0.802359	0.962746
4	jewish	1512	0.851224	0.854121	0.951192
8	psychiatric_or_mental_illness	918	0.871861	0.836761	0.963380
0	male	8912	0.891339	0.877872	0.957811
1	female	10795	0.897445	0.890568	0.954890
3	christian	8163	0.899208	0.921535	0.934436

```
get_final_metric(bias_metrics_df, calculate_overall_auc(X_cv, MODEL_NAME))
```

0.8904733235626956

Naive Bayes

Hyper parameter tuning

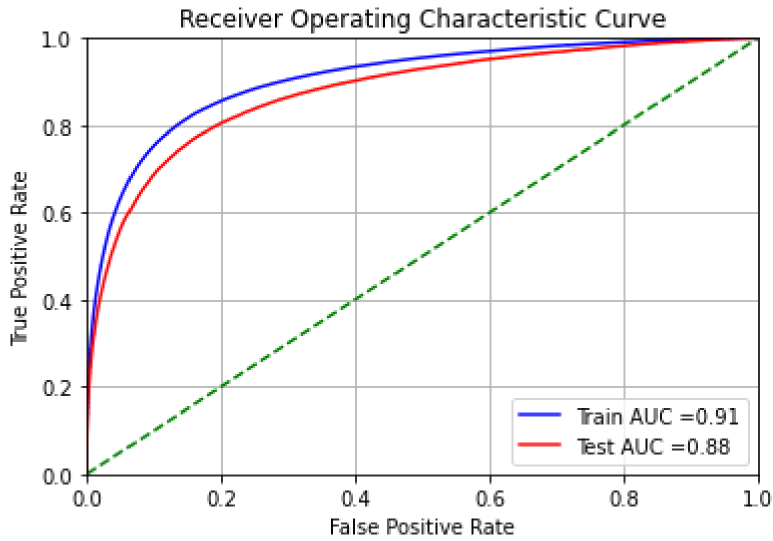
```
alpha = [10 ** x for x in range(-2, 1)]
auc_array_train=[]
auc_array_cv=[]
for i in alpha:
  clf = MultinomialNB(alpha=i)
  clf.fit(x_tr, Y_train)

  sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
  sig_clf.fit(x_tr, Y_train)

  predict_y_train = sig_clf.predict_proba(x_tr)[:,1]
  predict_y_cv = sig_clf.predict_proba(x_cv)[:,1]

  auc_array_train.append(roc_auc_score(Y_train, predict_y_train))
  auc_array_cv.append(roc_auc_score(Y_cv, predict_y_cv))
  print('For values of alpha = ', i, "The auc score on CV is:",roc_auc_score(Y_cv, predict_y_cv))
error_plot(alpha,auc_array_train,auc_array_cv)
```

For values of alpha = 0.01 The auc score on CV is: 0.8638602745525787
For values of alpha = 0.1 The auc score on CV is: 0.8775647543438234
For values of alpha = 1 The auc score on CV is: 0.8627768026044911

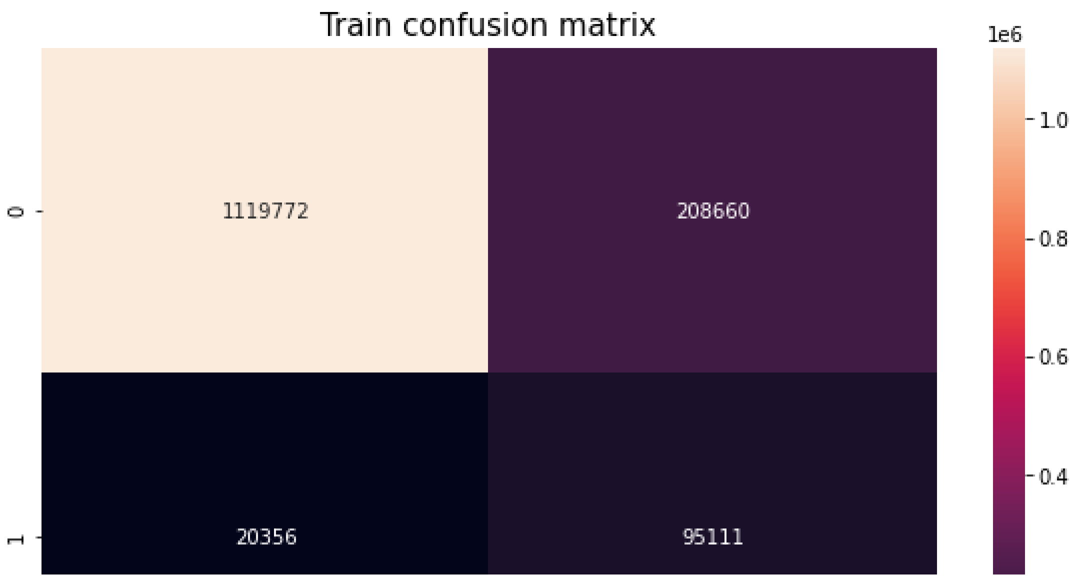

```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
clf = MultinomialNB(alpha=0.1)
clf.fit(x_tr, Y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_tr, Y_train)

y_train_pred = sig_clf.predict_proba(x_tr)[:,1]
y_cv_pred = sig_clf.predict_proba(x_cv)[:,1]
```

ROC Curve

```
roc_curve_plot(Y_train,y_train_pred,Y_cv,y_cv_pred)
```



==
the maximum value of tpr*(1-fpr) 0.6943255401190821 for threshold 0.066



Custom metric

```
# Prediction on CV data
MODEL_NAME = 'NB_model'
X_cv[MODEL_NAME] = y_cv_pred
```

```
X_cv.head(2)
```

```
bias_metrics_df = compute_bias_metrics_for_model(X_cv, IDENTITY_COLUMNS, MODEL_NAME, TARGET_COLUMN
bias_metrics_df
```

	subgroup	subgroup_size	subgroup_auc	bpsn_auc	bnsp_auc
2	homosexual_gay_or_lesbian	2163	0.770932	0.744431	0.904679
5	muslim	4209	0.771887	0.727883	0.917432
6	black	3079	0.776339	0.716623	0.921781
7	white	5018	0.785281	0.692370	0.936333
4	jewish	1512	0.804850	0.786438	0.894269
1	female	10795	0.844507	0.821241	0.899532
0	male	8912	0.846546	0.803253	0.913716
3	christian	8163	0.853535	0.891002	0.832497
8	psychiatric_or_mental_illness	918	0.862032	0.804329	0.921201

```
get_final_metric(bias_metrics_df, calculate_overall_auc(X_cv, MODEL_NAME))
```

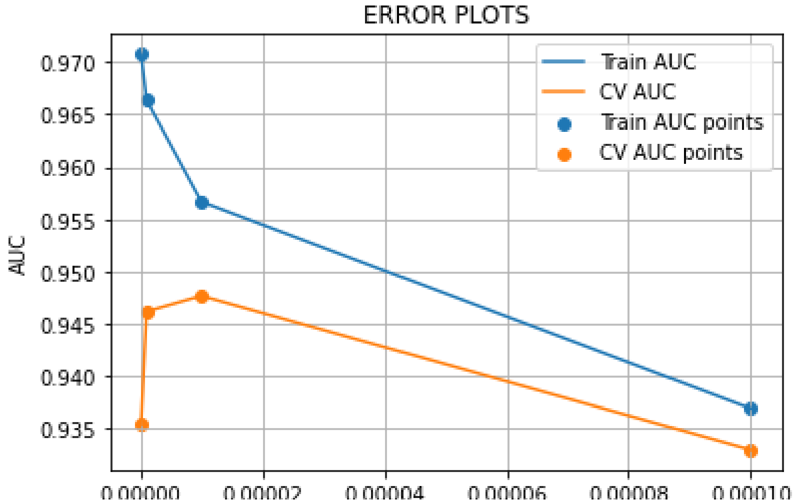
0.837811746713348

Linear Support Vector Machines

Hyper paramter tuning

```
alpha = [10 ** x for x in range(-7, -3)]
auc_array_train=[]
auc_array_cv=[]
for i in alpha:
  clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=5,class_weight='balanced')
  clf.fit(x_tr, Y_train)
  sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
  sig_clf.fit(x_tr, Y_train)

  predict_y_train = sig_clf.predict_proba(x_tr)[:,1]
  predict_y_cv = sig_clf.predict_proba(x_cv)[:,1]

  auc_array_train.append(roc_auc_score(Y_train, predict_y_train))
  auc_array_cv.append(roc_auc_score(Y_cv, predict_y_cv))
  print('For values of alpha = ', i, "The auc score on CV is:",roc_auc_score(Y_cv, predict_y_cv))

error_plot(alpha,auc_array_train,auc_array_cv)
```

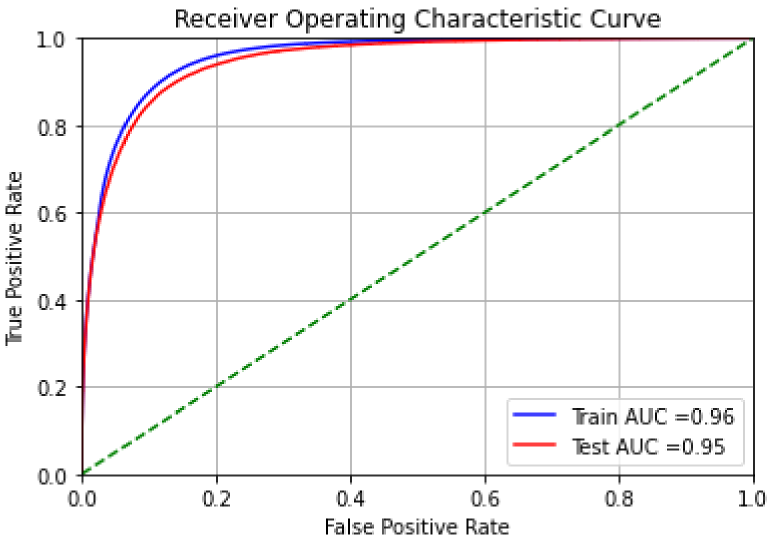
For values of alpha = 1e-07 The auc score on CV is: 0.9353775229814637
For values of alpha = 1e-06 The auc score on CV is: 0.9461685173473453
For values of alpha = 1e-05 The auc score on CV is: 0.9476579787160421
For values of alpha = 0.0001 The auc score on CV is: 0.9329576727848292




```
clf = SGDClassifier(alpha=0.00001, penalty='l2', loss='hinge', random_state=42,class_weight='balanced')
clf.fit(x_tr, Y_train)

sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_tr, Y_train)

y_train_pred = sig_clf.predict_proba(x_tr)[:,1]
y_cv_pred = sig_clf.predict_proba(x_cv)[:,1]
```

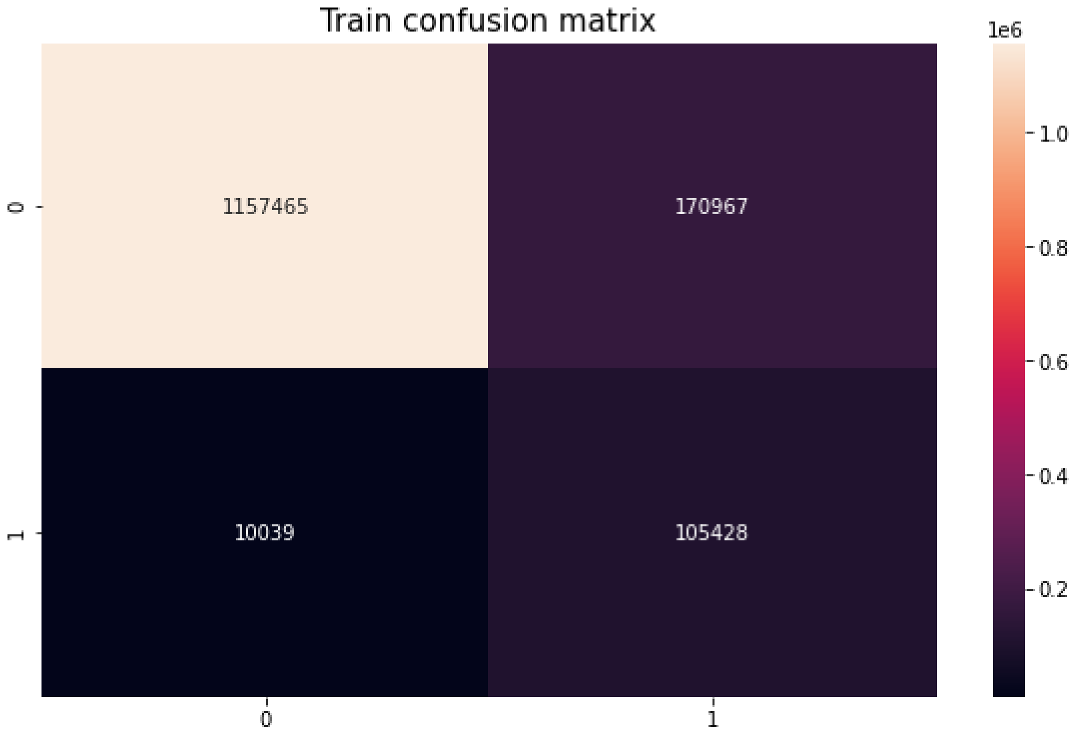
ROC Curve

```
roc_curve_plot(Y_train,y_train_pred,Y_cv,y_cv_pred)
```



==

the maximum value of tpr*(1-fpr) 0.7955484326347957 for threshold 0.059



Custom metric


```
# Prediction on CV data
MODEL_NAME = 'SVM_model'
X_cv[MODEL_NAME] = y_cv_pred
```

```
X_cv.head(2)
```

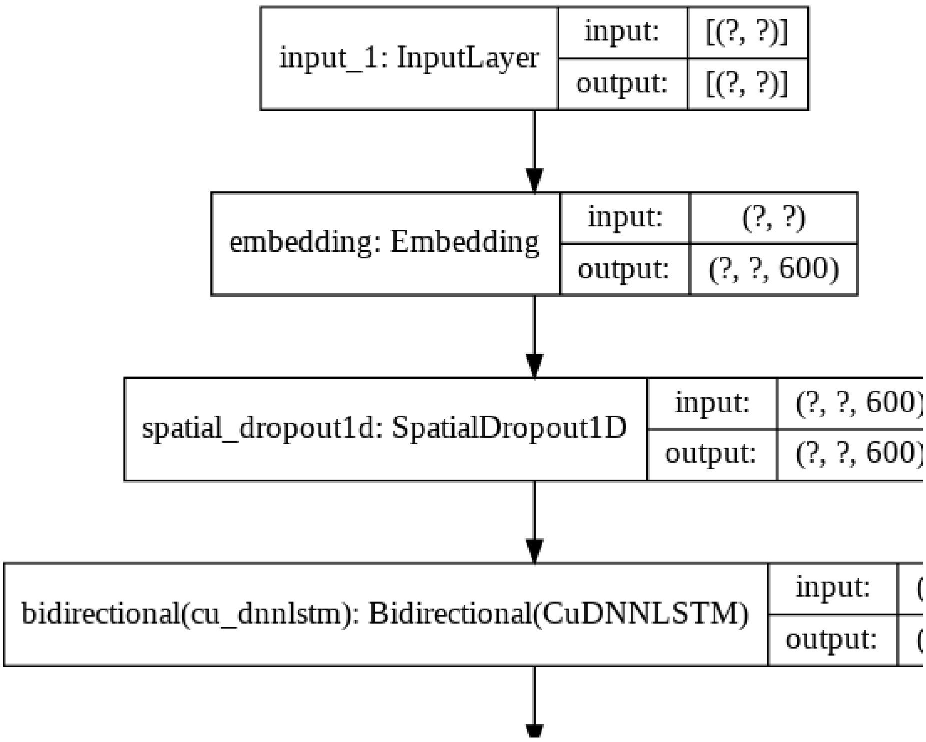
```
bias_metrics_df = compute_bias_metrics_for_model(X_cv, IDENTITY_COLUMNS, MODEL_NAME, TARGET_COLUMN
bias_metrics_df
```

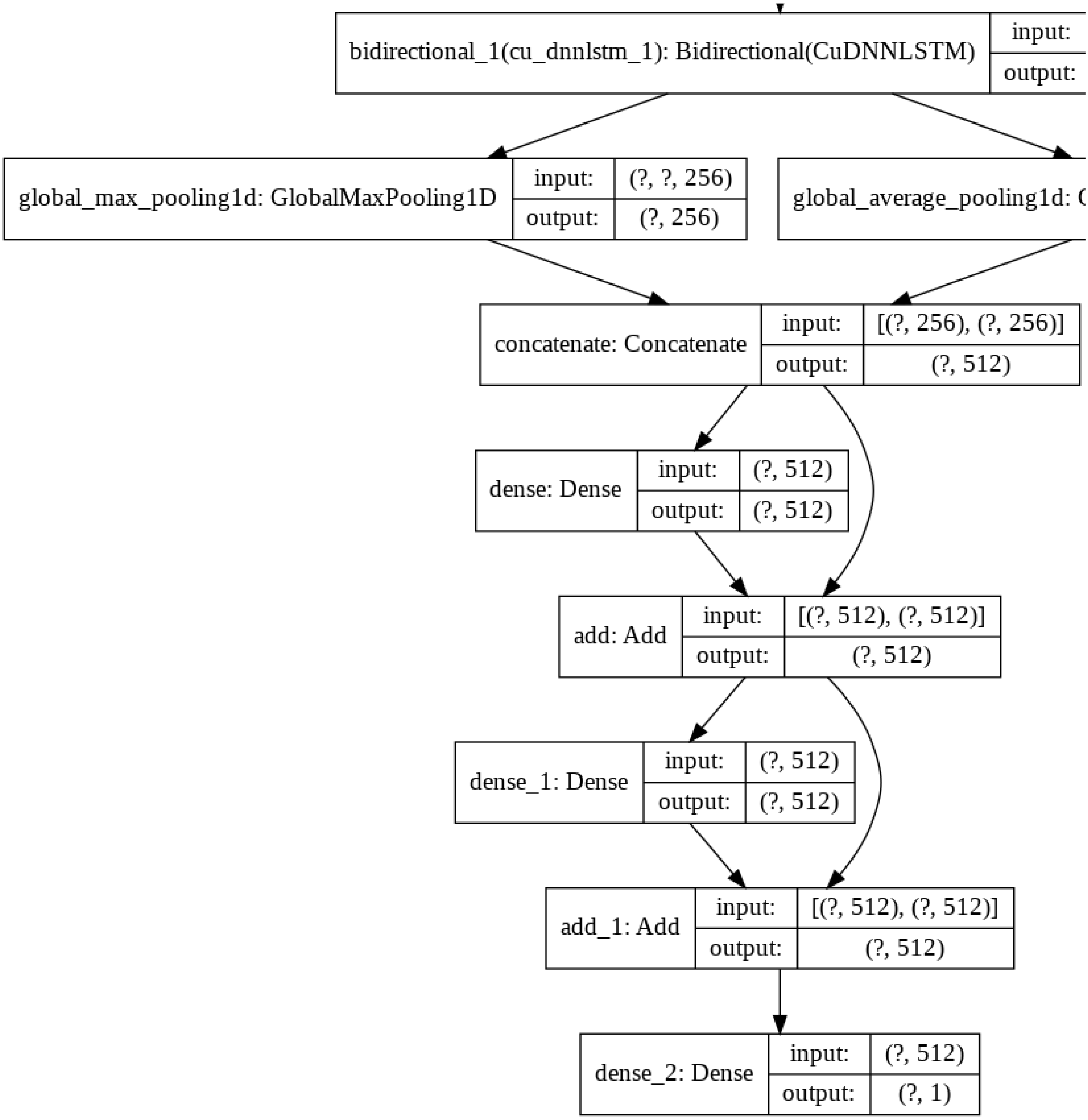
	subgroup	subgroup_size	subgroup_auc	bpsn_auc	bnsp_auc
6	black	3079	0.777690	0.690604	0.975239
2	homosexual_gay_or_lesbian	2163	0.783777	0.726070	0.969826
7	white	5018	0.810377	0.717118	0.975577
5	muslim	4209	0.815634	0.755065	0.971256
4	jewish	1512	0.846175	0.837240	0.956459
8	psychiatric_or_mental_illness	918	0.870516	0.812753	0.969329
0	male	8912	0.889711	0.870435	0.959899
1	female	10795	0.894940	0.885971	0.955470
3	christian	8163	0.898399	0.918577	0.936055

```
get_final_metric(bias_metrics_df, calculate_overall_auc(X_cv, MODEL_NAME))
```

0.8814130929571368

Deep Learning model 1: Using Only Text Feature






```
#https://www.kaggle.com/thousandvoices/simple-lstm
EMBEDDING_FILES = [
    '/content/drive/My Drive/jigsaw/crawl-300d-2M.gensim',
    '/content/drive/My Drive/jigsaw/glove.840B.300d.gensim'
]
NUM_MODELS = 2
BATCH_SIZE = 512
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',

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

```
TARGET_COLUMN= target
CHARS_TO_REMOVE = '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n“”’\'∞θ÷α•à−β∅³π‘₹´°£€\×™√²—'

X_train, X_te = train_test_split(train, test_size=0.1,random_state=42)
X_train, X_cv = train_test_split(X_train, test_size=0.1,random_state=42)

x_train = X_train[TEXT_COLUMN].astype(str)
y_train = X_train[TARGET_COLUMN].values


x_cv = X_cv[TEXT_COLUMN].astype(str)
y_cv = X_cv[TARGET_COLUMN].values

x_te = X_te[TEXT_COLUMN].astype(str)
y_te = X_te[TARGET_COLUMN].values

for column in IDENTITY_COLUMNS + [TARGET_COLUMN]:
    X_train[column] = np.where(X_train[column] >= 0.5, True, False)
    X_cv[column] = np.where(X_cv[column] >= 0.5, True, False)
    X_te[column] = np.where(X_te[column] >= 0.5, True, False)


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 word_index.items():
        for candidate in [word, word.lower()]:
            if candidate in embedding_index:
                embedding_matrix[i] = embedding_index[candidate]
                break
    return embedding_matrix


tokenizer = text.Tokenizer(filters=CHARS_TO_REMOVE, lower=False)
tokenizer.fit_on_texts(list(x_train))

x_train = tokenizer.texts_to_sequences(x_train)
x_cv = tokenizer.texts_to_sequences(x_cv)
x_te = tokenizer.texts_to_sequences(x_te)

x_train = sequence.pad_sequences(x_train, maxlen=MAX_LEN)
x_cv = sequence.pad_sequences(x_cv, maxlen=MAX_LEN)
x_te = sequence.pad_sequences(x_te, maxlen=MAX_LEN)


#pickle.dump(tokenizer,open("/content/drive/My Drive/jigsaw/dltokenizer","wb"))
sample_weights = np.ones(len(x_train), dtype=np.float32)
sample_weights += X_train[IDENTITY_COLUMNS].sum(axis=1)
sample_weights += X_train[TARGET_COLUMN] * (~X_train[IDENTITY_COLUMNS]).sum(axis=1)
sample_weights += (~X_train[TARGET_COLUMN]) * X_train[IDENTITY_COLUMNS].sum(axis=1) * 5
sample_weights /= sample_weights.mean()

embedding_matrix = np.concatenate([build_matrix(tokenizer.word_index, f) for f in EMBEDDING_FILES], axis=-1)


  def build_model(embedding_matrix):
    words = Input(shape=(None,))
```

```
    x = Embedding(*embedding_matrix.shape, weights=[embedding_matrix], trainable=False)(words)
    x = SpatialDropout1D(0.2)(x)
    x = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x)
    x = Bidirectional(CuDNNLSTM(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(1, activation='sigmoid')(hidden)

    model = Model(inputs=words, outputs=result)
    model.compile(loss='binary_crossentropy', optimizer='adam')
    #plot_model(model, to_file='/content/drive/My Drive/jigsaw/model1.png', show_shapes=True)
    return model
```

```
!pip show tensorflow
!pip install plot_model
!pip install tensorboardcolab
%load_ext tensorboard
!rm -rf ./logs/
import warnings
warnings.filterwarnings("ignore")
```

```
Name: tensorflow
Version: 2.2.0
Summary: TensorFlow is an open source machine learning framework for everyone.
Home-page: https://www.tensorflow.org/
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: /usr/local/lib/python3.6/dist-packages
Requires: tensorboard, absl-py, keras-preprocessing, opt-einsum, h5py, grpcio, numpy, gast, protobuf, tens
Required-by: fancyimpute
Requirement already satisfied: plot_model in /usr/local/lib/python3.6/dist-packages (0.20)
Requirement already satisfied: tensorboardcolab in /usr/local/lib/python3.6/dist-packages (0.0.22)
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
checkpoint_predictions = []
weights = []
checkpoint = tensorflow.keras.callbacks.ModelCheckpoint('/content/drive/My Drive/jigsaw/Model11.hdf5', monit
log_dir="/content/drive/My Drive/jigsaw/Model11/logs/fit/" + datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1, write_graph=True,write

model = build_model(embedding_matrix)
model.fit(x_train, y_train,batch_size=BATCH_SIZE,epochs=5,verbose=2,validation_data=(x_cv,y_cv),callbacks=[tens
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.
2020-05-31 10:31:57,422 : WARNING : `write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard`
Epoch 1/5

Epoch 00001: val_loss improved from inf to 0.24552, saving model to /content/drive/My Drive/jigsaw/Mod
2856/2856 - 829s - loss: 0.4226 - val_loss: 0.2455
Epoch 2/5

Epoch 00002: val_loss improved from 0.24552 to 0.24449, saving model to /content/drive/My Drive/jigsaw,
2856/2856 - 813s - loss: 0.4088 - val_loss: 0.2445
Epoch 3/5

Epoch 00003: val_loss improved from 0.24449 to 0.24227, saving model to /content/drive/My Drive/jigsaw,
2856/2856 - 813s - loss: 0.4039 - val_loss: 0.2423
Epoch 4/5

Epoch 00004: val_loss did not improve from 0.24227
2856/2856 - 809s - loss: 0.3998 - val_loss: 0.2475
Epoch 5/5
```

```
%tensorboard --logdir='/content/drive/My Drive/jigsaw/Model11/logs/fit'
```

```
def load_file(path):
  fp=open(path,'r')
  all_lines=fp.readlines()
  words_list=[]
  for line in all_lines:
    words_list.append(line.strip())
  fp.close()
  return words_list

pos_words=load_file(pos_path)
neg_words=load_file(neg_path)

#count  number of positive and negative words in each comment
def pos_word_count(comment):
  count=0
  for word in comment.split():
    if word in pos_words:
      count=count+1
  return count

def neg_word_count(comment):
  count=0
  for word in comment.split():
    if word in neg_words:
      count=count+1
  return count


# pos_word_count
count=[]
for i in X_train['clean_text'].values:
  count.append(pos_word_count(str(i)))
train_features['pos_word_count']=count

count=[]
for i in X_cv['clean_text'].values:
  count.append(pos_word_count(str(i)))
cv_features['pos_word_count']=count

count=[]
for i in X_te['clean_text'].values:
  count.append(pos_word_count(str(i)))
te_features['pos_word_count']=count

count=[]
for i in test['clean_text'].values:
  count.append(pos_word_count(str(i)))
test_features['pos_word_count']=count

# neg_word_count
count=[]
for i in X_train['clean_text'].values:
  count.append(neg_word_count(str(i)))
train_features['neg_word_count']=count
```

TensorBoard SCALARS GRAPHS DISTRIBUTIONS HISTOGRAMS

Histogram mode

OVERLAY OFFSET

Filter tags (regular expressions supporte

bidirectional_1

```
# load the model
model1=tensorflow.keras.models.load_model('/content/drive/My Drive/jigsaw/Model11.hdf5')

predictions=model1.predict(x_te, batch_size=2048).flatten()
MODEL_NAME = 'score'
X_te[MODEL_NAME] = predictions
bias_metrics_df = compute_bias_metrics_for_model(X_te, IDENTITY_COLUMNS, MODEL_NAME, TARGET_COLUMN
get_final_metric(bias_metrics_df, calculate_overall_auc(X_te, MODEL_NAME))
```

0.9319064728629746

Deep Learning Model 2 : Text feature + Additional Features

Additional Features:

- topic features
- positive word count
- negative word count
- sentiment of each comment
- word count,character count

```
train=pickle.load(open( "/content/drive/My Drive/trpreprocessed", "rb" ))
test=pickle.load(open( "/content/drive/My Drive/jigsaw/tepreprocessed", "rb" ))
```

bidirectional_1/forward_cu_dnnlstm_1/kernel_(

Train cv split

```
X_train, X_te = train_test_split(train, test_size=0.1,random_state=42)

X_train, X_cv = train_test_split(X_train, test_size=0.1,random_state=42)
```

Topic modeling (Unsupervised Clustering Method)

- LDA (Latent Dirichlet Allocation) is an unsupervised machine-learning model that automatic and to derive hidden patterns exhibited by a text corpus. Thus, assisting better decision mak
- we will model our clean_text into 5 different topics and then take these topics as features.

```
#https://github.com/sonalijathar01/Toxic-comment-classification/blob/master/Jigsaw_UnIntended_Bias_Toxicity_
%%time
stemmer = SnowballStemmer("english")
def lemmatize_stemming(text):
    return stemmer.stem(WordNetLemmatizer().lemmatize(text, pos='v'))

# Tokenize and lemmatize
def preprocess(text):
    '''this is for preprocessing text using gensim.utils.simple_preprocess() function'''
    result=[]
    for token in gensim.utils.simple_preprocess(text) :
        if token not in gensim.parsing.preprocessing.STOPWORDS and len(token) > 3:
            result.append(lemmatize_stemming(token))

    return result


data = X_train.clean_text.values.tolist()
processed_docs = []
for doc in data:
    processed_docs.append(preprocess(doc))
```

```
CPU times: user 11min 3s, sys: 2.11 s, total: 11min 5s
Wall time: 11min 5s
```

Create the Dictionary and Corpus needed for Topic Modeling

```
%%time
# Create Dictionary
dictionary = gensim.corpora.Dictionary(processed_docs)

# Create Corpus
texts = processed_docs

# Term Document Frequency
# Gensim creates a unique id for each word in the document. The produced corpus shown above is a mapping o
corpus = [dictionary.doc2bow(text) for text in texts]

# View
print(corpus[:1])
```

```
[[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1)]]
CPU times: user 1min 12s, sys: 1.14 s, total: 1min 13s
Wall time: 1min 13s
```

```
pickle.dump(dictionary,open('/content/drive/My Drive/jigsaw/dictionary','wb'))
#dictionary=pickle.load(open('/content/drive/My Drive/jigsaw/dictionary','rb'))
```

```
# Human readable format of corpus (term-frequency)
[[(dictionary[id], freq) for id, freq in cp] for cp in corpus[:1]]
```

```
[[('alaska', 1),
  ('career', 1),
  ('collegi', 1),
  ('good', 1),
  ('gymnast', 1),
  ('luck', 1),
  ('repres', 1),
  ('thank', 1)]]
```

Building topic model

```
#this code took me 10 hours to run

# Build LDA model
lda_model = LdaModel(corpus=corpus, id2word=dictionary, num_topics=5, random_state=100, update_every=1,
                                           chunksize=100, passes=10, alpha='auto', per_word_topics=True)


#pickle.dump(lda_model,open('/content/drive/My Drive/jigsaw/ldamodelall','wb'))
lda_model=pickle.load(open('/content/drive/My Drive/jigsaw/ldamodelall','rb'))
```

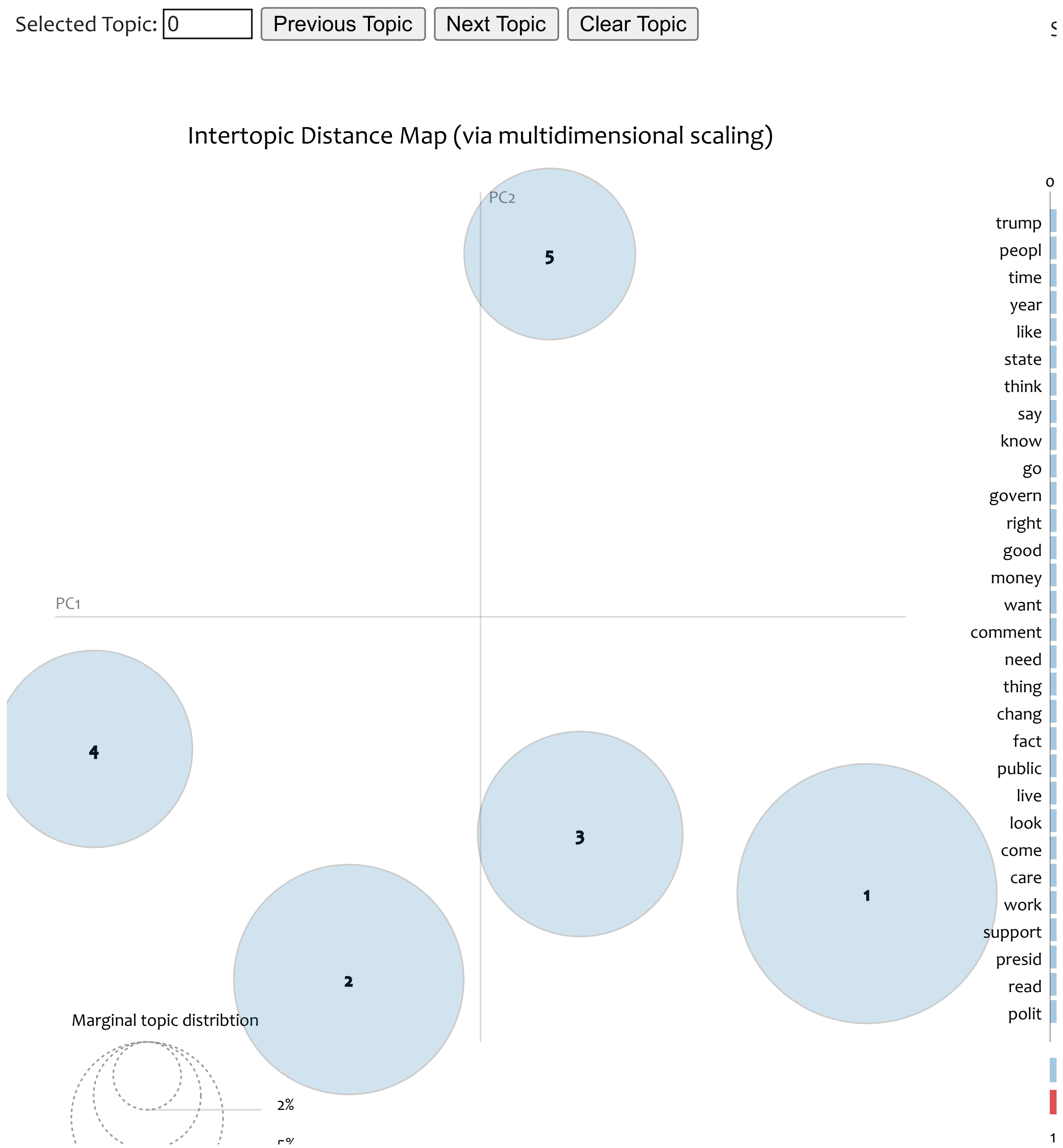
```
from gensim.models import CoherenceModel

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model, texts=processed_docs, dictionary=dictionary, cohe
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

```
Coherence Score:  0.4943335322897525
```

```
# Print the Keyword in the 5 topics
print(lda_model.print_topics())
doc_lda = lda_model[corpus]
```

```
[(0, '0.012*"life" + 0.011*"million" + 0.011*"question" + 0.009*"women" + 0.009*"church" + 0.007*"line" + 0
```

```
# Visualize the topics on train
 pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim.prepare(lda_model, corpus, dictionary)
vis
```



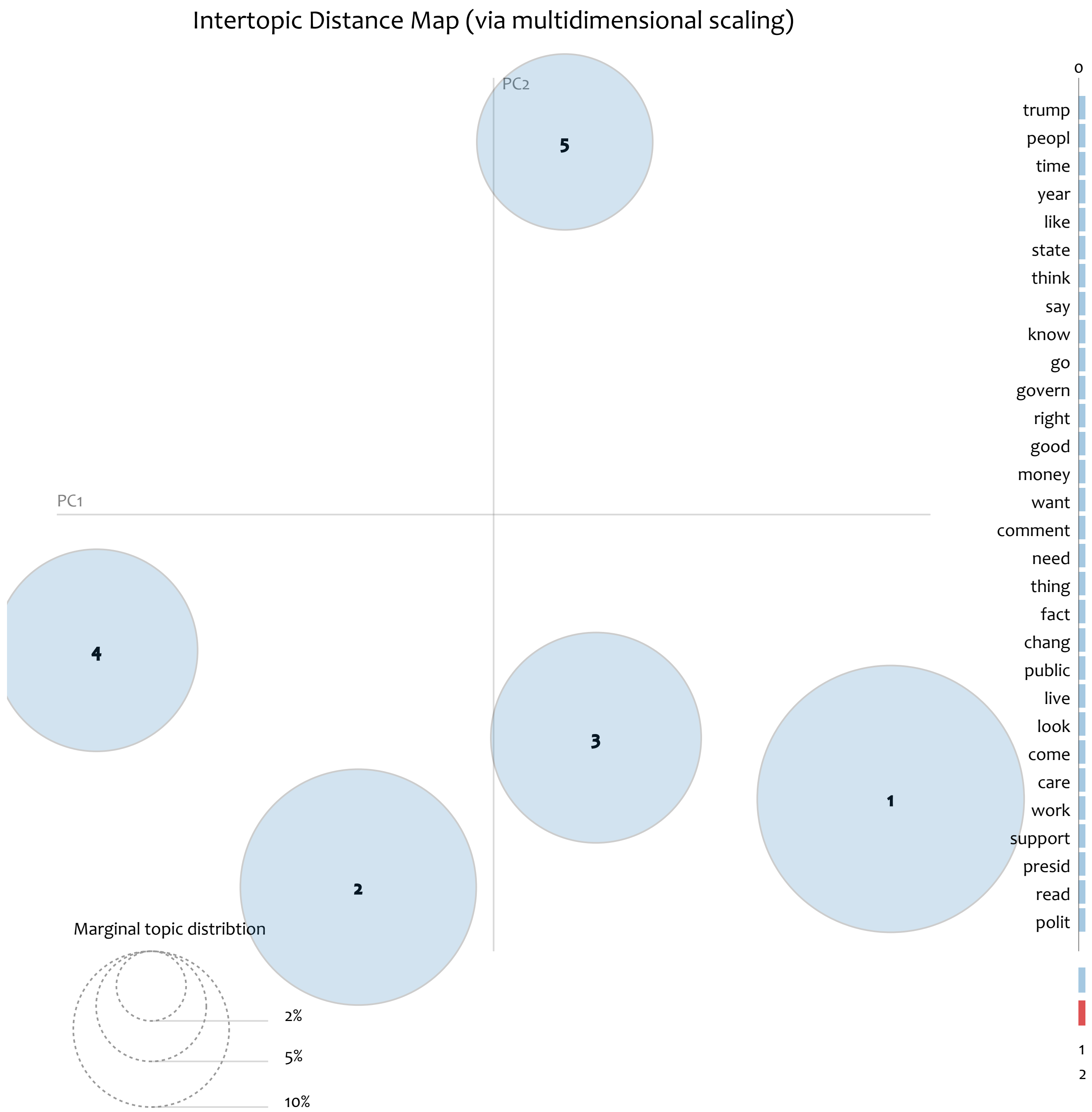
CV data preprocessing

```
%%time
data1 = X_cv.clean_text.values.tolist()
processed_docs1 = []
for doc in data1:
    processed_docs1.append(preprocess(doc))

print(processed_docs1[:1])
corpus1= [dictionary.doc2bow(text) for text in processed_docs1]
```

```
[['fals', 'doctrin', 'attack', 'catholic', 'contain', 'articl']]
CPU times: user 1min 19s, sys: 386 ms, total: 1min 19s
Wall time: 1min 19s
```

```
vis = pyLDAvis.gensim.prepare(lda_model, corpus1, dictionary)
vis
```

Selected Topic: 0 Previous Topic Next Topic Clear Topic



Test Data Preprocessing

```
%%time
data11 = X_te.clean_text.values.tolist()
```

```
processed_docs11 = []
for doc in data11:
    processed_docs11.append(preprocess(doc))

print(processed_docs11[:1])
corpus11= [dictionary.doc2bow(text) for text in processed_docs11]
```

```
[['breath', 'fresh', 'embrac', 'common', 'sens', 'valu', 'instead', 'leadership', 'canada', 'clear', 'differ', 'page', 'rea
CPU times: user 1min 24s, sys: 393 ms, total: 1min 25s
Wall time: 1min 25s
```

Actual test data

```
#will use for submission
%%time
data_test = test.clean_text.values.tolist()
processed_docs_test = []
for doc in data_test:
    processed_docs_test.append(preprocess(doc))

print(processed_docs_test[:1])
corpus_test= [dictionary.doc2bow(text) for text in processed_docs_test]
```

```
[['integr', 'mean', 'debt', 'appli', 'presid', 'trump']]
CPU times: user 45.9 s, sys: 238 ms, total: 46.2 s
Wall time: 46.2 s
```

Converting Topics to Feature Vectors

```
# For train vectors
train_vecs = []

for i in range(len(X_train)):
    top_train_topics = lda_model.get_document_topics(corpus[i], minimum_probability=0.0)
    topic_train_vec = [top_train_topics[i][1] for i in range(5)]
    train_vecs.append(topic_train_vec)
# Printing top five train vectors
train_vecs[:5]
```

```
[[0.2572527, 0.116501085, 0.15880889, 0.23540507, 0.23203227],
 [0.1085511, 0.12808271, 0.23760512, 0.205904, 0.31985703],
 [0.10038725, 0.13197713, 0.24514905, 0.28512138, 0.23736522],
 [0.100387305, 0.13197719, 0.24514884, 0.23232232, 0.29016432],
 [0.09079904, 0.11937169, 0.22173429, 0.21013263, 0.35796234]]
```

```
# For cv vectors
cv_vecs = []
for i in range(len(X_cv)):
    top_cv_topics = lda_model.get_document_topics(corpus1[i], minimum_probability=0.0)
```

```
    topic_cv_vec = [top_cv_topics[i][1] for i in range(5)]
    cv_vecs.append(topic_cv_vec)
# Printing top five test vectors
cv_vecs[:5]
```

```
[[0.09080016, 0.16712835, 0.17397994, 0.30563962, 0.2624519],
 [0.23157655, 0.07987076, 0.23264326, 0.25131276, 0.20459664],
 [0.132223, 0.1595133, 0.16605367, 0.33729813, 0.2049119],
 [0.06035246, 0.14152181, 0.17819482, 0.3855083, 0.23442256],
 [0.1206431, 0.13538024, 0.18808436, 0.24763924, 0.30825308]]
```

```
# For test vectors
te_vecs = []

for i in range(len(X_te)):
    top_te_topics = lda_model.get_document_topics(corpus11[i], minimum_probability=0.0)
    topic_te_vec = [top_te_topics[i][1] for i in range(5)]
    te_vecs.append(topic_te_vec)
# Printing top five test vectors
te_vecs[:5]
```

```
[[0.1436283, 0.28519964, 0.16165379, 0.14947933, 0.26003894],
 [0.075519, 0.11585588, 0.19756171, 0.23977299, 0.37129042],
 [0.19178723, 0.16131727, 0.134504, 0.26750937, 0.24488218],
 [0.16126405, 0.080642395, 0.21566236, 0.2366619, 0.30576932],
 [0.070576765, 0.2041459, 0.17327477, 0.2737662, 0.27823636]]
```

```
# For actual test vectors
test_vecs = []

for i in range(len(test)):
    top_test_topics = lda_model.get_document_topics(corpus_test[i], minimum_probability=0.0)
    topic_test_vec = [top_test_topics[i][1] for i in range(5)]
    test_vecs.append(topic_test_vec)


# Create the new df with 5 topics
train_features = pd.DataFrame(train_vecs,columns=['Topic-1','Topic-2','Topic-3','Topic-4','Topic-5'])
cv_features = pd.DataFrame(cv_vecs,columns=['Topic-1','Topic-2','Topic-3','Topic-4','Topic-5'])
te_features = pd.DataFrame(te_vecs,columns=['Topic-1','Topic-2','Topic-3','Topic-4','Topic-5'])
test_features = pd.DataFrame(test_vecs,columns=['Topic-1','Topic-2','Topic-3','Topic-4','Topic-5'])
```

Count number of positive and negative words in each comm

- https://gist.github.com/mkulakowski2/4289441
- https://gist.github.com/mkulakowski2/4289437

```
#I have created 2 files one for  positive words taken from https://gist.github.com/mkulakowski2/4289437 ,anothe

#load pos,neg words
pos_path='/content/drive/My Drive/jigsaw/pos.txt'
neg_path='/content/drive/My Drive/jigsaw/neg.txt'
```

```
count=[]
for i in X_cv['clean_text'].values:
  count.append(neg_word_count(str(i)))
cv_features['neg_word_count']=count

count=[]
for i in X_te['clean_text'].values:
  count.append(neg_word_count(str(i)))
te_features['neg_word_count']=count

count=[]
for i in test['clean_text'].values:
  count.append(pos_word_count(str(i)))
test_features['neg_word_count']=count
```

Find the sentiment of each comment

```
#https://www.pluralsight.com/guides/natural-language-processing-extracting-sentiment-from-text-data
%%time
sentiment_count=[]
for i in X_train['clean_text'].values:
  sentiment_count.append(TextBlob(i).sentiment[0])
train_features['sentiment']=sentiment_count

sentiment_count=[]
for i in X_cv['clean_text'].values:
  sentiment_count.append(TextBlob(i).sentiment[0])
cv_features['sentiment']=sentiment_count

sentiment_count=[]
for i in X_te['clean_text'].values:
  sentiment_count.append(TextBlob(i).sentiment[0])
te_features['sentiment']=sentiment_count

sentiment_count=[]
for i in test['clean_text'].values:
  sentiment_count.append(TextBlob(i).sentiment[0])
test_features['sentiment']=sentiment_count
```

```
CPU times: user 19min 6s, sys: 2.87 s, total: 19min 9s
Wall time: 19min 9s
```

Features calculated during the EDA

```
from tqdm import tqdm
#comment_word_count
count=[]
for i in tqdm(X_train['clean_text'].values):
  count.append(len(i.split()))
```

```
count.append(len(i.split()))
train_features['comment_word_count']=count

count=[]
for i in tqdm(X_cv['clean_text'].values):
  count.append(len(i.split()))
cv_features['comment_word_count']=count

count=[]
for i in tqdm(X_te['clean_text'].values):
  count.append(len(i.split()))
te_features['comment_word_count']=count

count=[]
for i in tqdm(test['clean_text'].values):
  count.append(len(i.split()))
test_features['comment_word_count']=count

#comment_char_count

count=[]
for i in tqdm(X_train['clean_text'].values):
  count.append(len(i))
train_features['comment_char_count']=count

count=[]
for i in tqdm(X_cv['clean_text'].values):
  count.append(len(i))
cv_features['comment_char_count']=count

count=[]
for i in tqdm(X_te['clean_text'].values):
  count.append(len(i))
te_features['comment_char_count']=count

count=[]
for i in tqdm(test['clean_text'].values):
  count.append(len(i))
test_features['comment_char_count']=count
```

```
100%|██████████| 1461947/1461947 [00:03<00:00, 379699.89it/s]
100%|██████████| 162439/162439 [00:00<00:00, 351337.65it/s]
100%|██████████| 180488/180488 [00:00<00:00, 354918.62it/s]
100%|██████████| 97320/97320 [00:00<00:00, 390665.21it/s]
100%|██████████| 1461947/1461947 [00:00<00:00, 1607047.00it/s]
100%|██████████| 162439/162439 [00:00<00:00, 1488215.71it/s]
100%|██████████| 180488/180488 [00:00<00:00, 1514697.43it/s]
100%|██████████| 97320/97320 [00:00<00:00, 1503715.79it/s]
```

```
test_features
```

	Topic-1	Topic-2	Topic-3	Topic-4	Topic-5	pos_word_count	neg_word_count	se
0	0.149603	0.176198	0.133320	0.316583	0.224297	1	1	(
1	0.195654	0.175510	0.132552	0.220671	0.275613	0	0	(
2	0.105486	0.196813	0.275211	0.191042	0.231447	3	3	
3	0.183384	0.153363	0.218330	0.211242	0.233680	5	5	
4	0.178390	0.231265	0.128876	0.252196	0.209274	2	2	
...	...	...	...	...	...	...	...	
97315	0.095354	0.141435	0.132553	0.304900	0.325758	0	0	(
97316	0.136550	0.126796	0.222251	0.236765	0.277638	2	2	
97317	0.212566	0.177888	0.139512	0.281822	0.188212	1	1	(

Encoding Numerical features

97320 rows × 10 columns

```
numerical_train_1=train_features['Topic-1'].values.reshape(-1, 1)
numerical_train_2=train_features['Topic-2'].values.reshape(-1, 1)
numerical_train_3=train_features['Topic-3'].values.reshape(-1, 1)
numerical_train_4=train_features['Topic-4'].values.reshape(-1, 1)
numerical_train_5=train_features['Topic-5'].values.reshape(-1, 1)
numerical_train_6=train_features['pos_word_count'].values.reshape(-1, 1)
numerical_train_7=train_features['neg_word_count'].values.reshape(-1, 1)
numerical_train_8=train_features['sentiment'].values.reshape(-1, 1)
numerical_train_9=train_features['comment_word_count'].values.reshape(-1, 1)
numerical_train_10=train_features['comment_char_count'].values.reshape(-1, 1)

numerical_cv_1=cv_features['Topic-1'].values.reshape(-1, 1)
numerical_cv_2=cv_features['Topic-2'].values.reshape(-1, 1)
numerical_cv_3=cv_features['Topic-3'].values.reshape(-1, 1)
numerical_cv_4=cv_features['Topic-4'].values.reshape(-1, 1)
numerical_cv_5=cv_features['Topic-5'].values.reshape(-1, 1)
numerical_cv_6=cv_features['pos_word_count'].values.reshape(-1, 1)
numerical_cv_7=cv_features['neg_word_count'].values.reshape(-1, 1)
numerical_cv_8=cv_features['sentiment'].values.reshape(-1, 1)
numerical_cv_9=cv_features['comment_word_count'].values.reshape(-1, 1)
numerical_cv_10=cv_features['comment_char_count'].values.reshape(-1, 1)

numerical_te_1=te_features['Topic-1'].values.reshape(-1, 1)
numerical_te_2=te_features['Topic-2'].values.reshape(-1, 1)
numerical_te_3=te_features['Topic-3'].values.reshape(-1, 1)
numerical_te_4=te_features['Topic-4'].values.reshape(-1, 1)
numerical_te_5=te_features['Topic-5'].values.reshape(-1, 1)
numerical_te_6=te_features['pos_word_count'].values.reshape(-1, 1)
numerical_te_7=te_features['neg_word_count'].values.reshape(-1, 1)
numerical_te_8=te_features['sentiment'].values.reshape(-1, 1)
numerical_te_9=te_features['comment_word_count'].values.reshape(-1, 1)
numerical_te_10=te_features['comment_char_count'].values.reshape(-1, 1)
```

```
numerical_test_1=test_features['Topic-1'].values.reshape(-1, 1)
numerical_test_2=test_features['Topic-2'].values.reshape(-1, 1)
numerical_test_3=test_features['Topic-3'].values.reshape(-1, 1)
numerical_test_4=test_features['Topic-4'].values.reshape(-1, 1)
numerical_test_5=test_features['Topic-5'].values.reshape(-1, 1)
numerical_test_6=test_features['pos_word_count'].values.reshape(-1, 1)
numerical_test_7=test_features['neg_word_count'].values.reshape(-1, 1)
numerical_test_8=test_features['sentiment'].values.reshape(-1, 1)
numerical_test_9=test_features['comment_word_count'].values.reshape(-1, 1)
numerical_test_10=test_features['comment_char_count'].values.reshape(-1, 1)


num_tr=np.concatenate((numerical_train_1,numerical_train_2,numerical_train_3,numerical_train_4,numerical_train
num_cv=np.concatenate((numerical_cv_1,numerical_cv_2,numerical_cv_3,numerical_cv_4,numerical_cv_5,numerica
num_te=np.concatenate((numerical_te_1,numerical_te_2,numerical_te_3,numerical_te_4,numerical_te_5,numerical_
num_test=np.concatenate((numerical_test_1,numerical_test_2,numerical_test_3,numerical_test_4,numerical_test_5

numerical=StandardScaler()
numerical_train=numerical.fit_transform(num_tr)
numerical_cv=numerical.transform(num_cv)
numerical_te=numerical.transform(num_te)
numerical_test=numerical.transform(num_test)
```

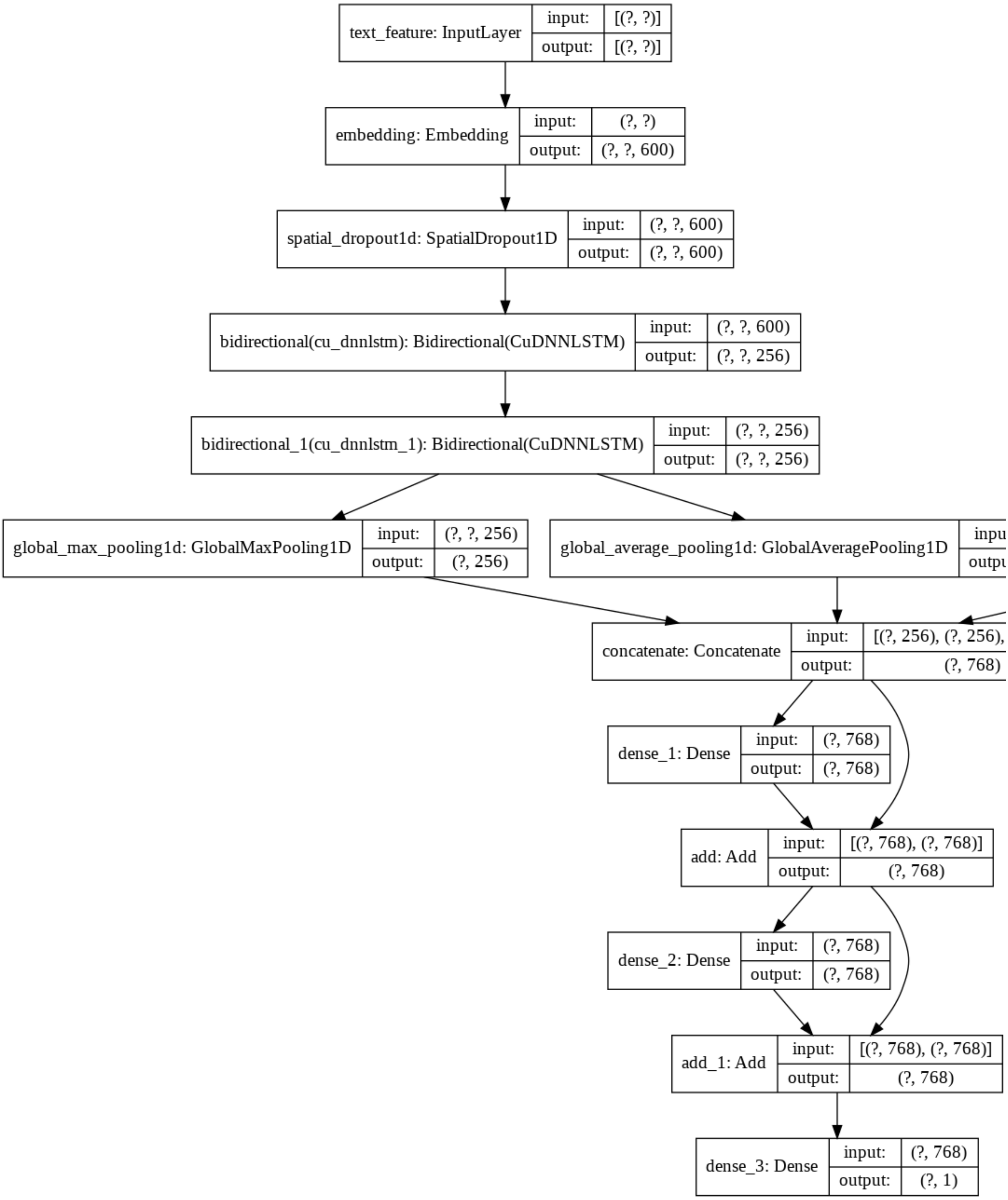
Deep Learning Model Data Preparation

```
EMBEDDING_FILES = [
    '/content/drive/My Drive/jigsaw/crawl-300d-2M.gensim',
    '/content/drive/My Drive/jigsaw/glove.840B.300d.gensim'
]
NUM_MODELS = 2
BATCH_SIZE = 512
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',

TEXT_COLUMN = 'comment_text'
TARGET_COLUMN='target'
CHARS_TO_REMOVE = '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n“”’\'∞θ÷α•à−β∅³π‘₹´°£€\×™√²—'

x_train = X_train[TEXT_COLUMN].astype(str)
y_train = X_train[TARGET_COLUMN].values

x_cv = X_cv[TEXT_COLUMN].astype(str)
y_cv = X_cv[TARGET_COLUMN].values

x_te = X_te[TEXT_COLUMN].astype(str)
x_test = test[TEXT_COLUMN].astype(str)

for column in IDENTITY_COLUMNS + [TARGET_COLUMN]:
```

```
    X_train[column] = np.where(X_train[column] >= 0.5, True, False)
    X_cv[column] = np.where(X_cv[column] >= 0.5, True, False)
    X_te[column] = np.where(X_te[column] >= 0.5, True, False)


tokenizer = text.Tokenizer(filters=CHARS_TO_REMOVE, lower=False)
tokenizer.fit_on_texts(list(x_train))

x_train = tokenizer.texts_to_sequences(x_train)
x_cv = tokenizer.texts_to_sequences(x_cv)
x_te = tokenizer.texts_to_sequences(x_te)
x_test = tokenizer.texts_to_sequences(x_test)

x_train = sequence.pad_sequences(x_train, maxlen=MAX_LEN)
x_cv = sequence.pad_sequences(x_cv, maxlen=MAX_LEN)
x_te = sequence.pad_sequences(x_te, maxlen=MAX_LEN)
x_test = sequence.pad_sequences(x_test, maxlen=MAX_LEN)


pickle.dump(tokenizer,open("/content/drive/My Drive/jigsaw/tokenizer","wb"))


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

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

embedding_matrix = np.concatenate([build_matrix(tokenizer.word_index, f) for f in EMBEDDING_FILES], axis=-1)
```

```
/usr/local/lib/python3.6/dist-packages/smart_open/smart_open_lib.py:253: UserWarning:

This function is deprecated, use smart_open.open instead. See the migration notes for details: https://githu
```

Deep Learning Model 2 : Dropout 0.2 + text feature + Additio

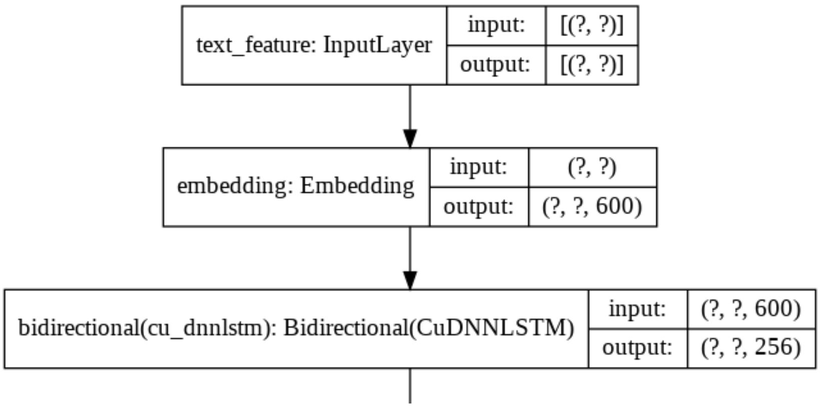

text_feature: InputLayer
input: [(?, ?)]
output: [(?, ?)]
embedding: Embedding
input: (?, ?)
output: (?, ?, 600)
spatial_dropout1d: SpatialDropout1D
input: (?, ?, 600)
output: (?, ?, 600)
bidirectional(cu_dnnlstm): Bidirectional(CuDNNLSTM)
input: (?, ?, 600)
output: (?, ?, 256)
bidirectional_1(cu_dnnlstm_1): Bidirectional(CuDNNLSTM)
input: (?, ?, 256)
output: (?, ?, 256)
global_max_pooling1d: GlobalMaxPooling1D
input: (?, ?, 256)
output: (?, 256)
global_average_pooling1d: GlobalAveragePooling1D
concatenate: Concatenate
input: [(?, 256), (?, 256),
output: (?, 768)
dense_1: Dense
input: (?, 768)
output: (?, 768)
add: Add
input: [(?, 768), (?, 768)]
output: (?, 768)
dense_2: Dense
input: (?, 768)
output: (?, 768)
add_1: Add
input: [(?, 768), (?, 768)]
output: (?, 768)
dense_3: Dense
input: (?, 768)
output: (?, 1)

```
def build_model(embedding_matrix):
    words = Input(shape=(None,),name="text_feature")
    x = Embedding(*embedding_matrix.shape, weights=[embedding_matrix], trainable=False)(words)
    x = SpatialDropout1D(0.2)(x)
    x = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x)
    x = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x)

    numerical_feats = Input(shape=(10,),name="numerical_features")
    numerical_featss = Dense(256,activation="relu",kernel_initializer="he_normal")(numerical_feats)

    hidden = concatenate([GlobalMaxPooling1D()(x),GlobalAveragePooling1D()(x),numerical_featss,])
    hidden = add([hidden, Dense(768, activation='relu')(hidden)])
    hidden = add([hidden, Dense(768, activation='relu')(hidden)])
    result = Dense(1, activation='sigmoid')(hidden)

    model = Model(inputs=[words,numerical_feats], outputs=[result])#, aux_result])
    model.compile(loss='binary_crossentropy', optimizer='adam')
    plot_model(model, to_file='/content/drive/My Drive/jigsaw/Model2.png', show_shapes=True)

    return model
```

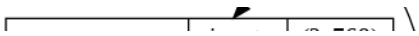
```
#all epochs to fit once
from datetime import datetime, timedelta

checkpoint = tensorflow.keras.callbacks.ModelCheckpoint('/content/drive/My Drive/jigsaw/Model22.hdf5', monit
log_dir="/content/drive/My Drive/jigsaw/Model22/logs/fit/" + datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1, write_graph=True,write
model = build_model(embedding_matrix)
model.fit([x_train,numerical_train], y_train,batch_size=BATCH_SIZE,epochs=5,verbose=2,validation_data=([x_cv,nu
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.
2020-05-31 12:09:21,855 : WARNING : `write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard`
Epoch 1/5

Epoch 00001: val_loss improved from inf to 0.25208, saving model to /content/drive/My Drive/jigsaw/Mod
2856/2856 - 915s - loss: 0.4270 - val_loss: 0.2521
Epoch 2/5

Epoch 00002: val_loss improved from 0.25208 to 0.24228, saving model to /content/drive/My Drive/jigsaw,
2856/2856 - 818s - loss: 0.4096 - val_loss: 0.2423
Epoch 3/5

Epoch 00003: val_loss did not improve from 0.24228
2856/2856 - 816s - loss: 0.4045 - val_loss: 0.2423
```

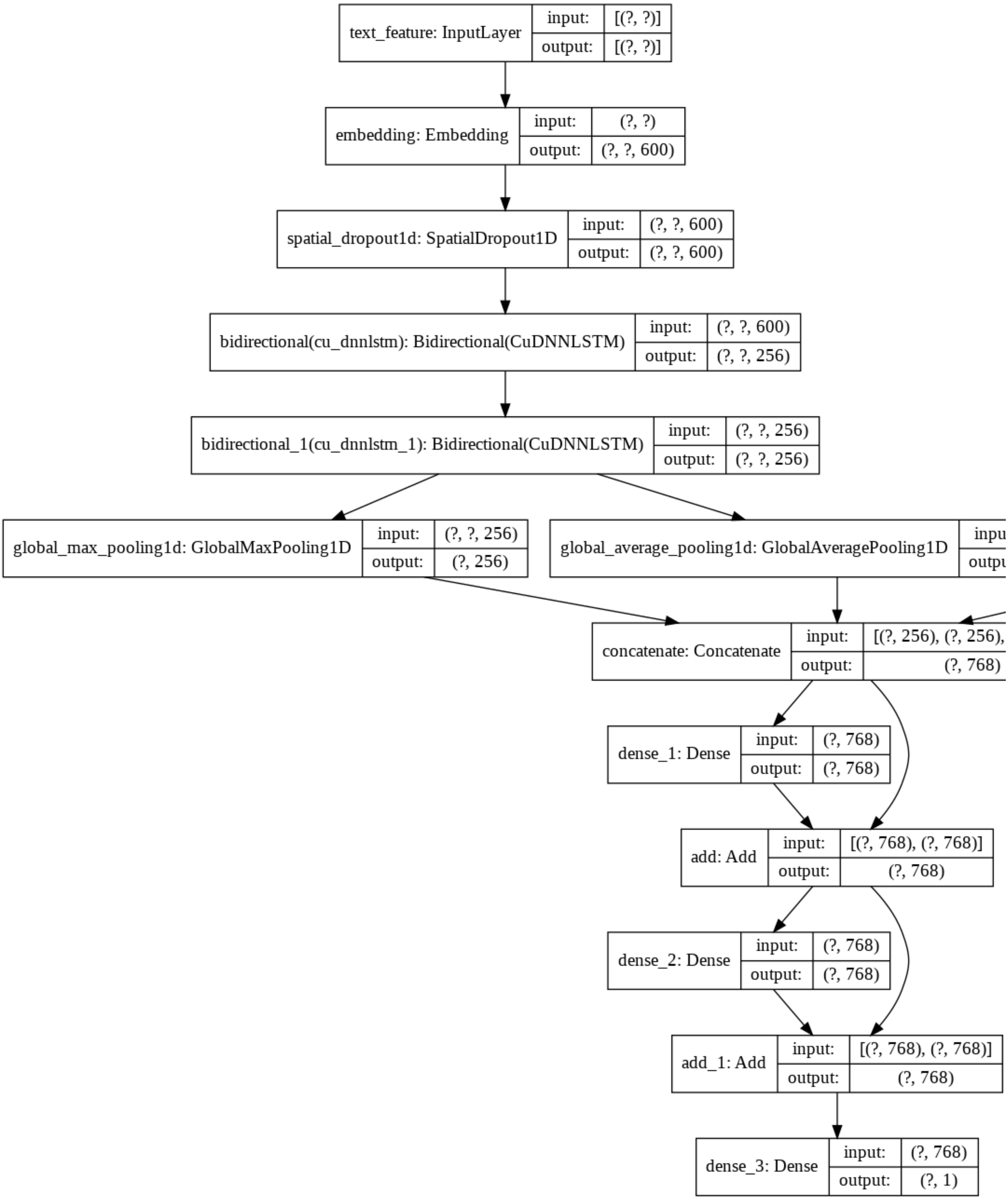
```
%tensorboard --logdir='/content/drive/My Drive/jigsaw/Model22/logs/fit'
```




```
def build_model(embedding_matrix):
    words = Input(shape=(None,),name="text_feature")
    x = Embedding(*embedding_matrix.shape, weights=[embedding_matrix], trainable=False)(words)
    x = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x)
    x = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x)

    numerical_feats = Input(shape=(10,),name="numerical_features")
    numerical_featss = Dense(256,activation="relu",kernel_initializer="he_normal")(numerical_feats)

    hidden = concatenate([GlobalMaxPooling1D()(x),GlobalAveragePooling1D()(x),numerical_featss,])
    hidden = add([hidden, Dense(768, activation='relu')(hidden)])
    hidden = add([hidden, Dense(768, activation='relu')(hidden)])
    result = Dense(1, activation='sigmoid')(hidden)

    model = Model(inputs=[words,numerical_feats], outputs=result)
    model.compile(loss='binary_crossentropy', optimizer='adam')
    plot_model(model, to_file='/content/drive/My Drive/jigsaw/model33.png', show_shapes=True)
    return model
```



```
#all epochs to fit once
from datetime import datetime, timedelta

checkpoint = tensorflow.keras.callbacks.ModelCheckpoint('/content/drive/My Drive/jigsaw/Model33.hdf5', monit
log_dir="/content/drive/My Drive/jigsaw/Model33/logs/fit/" + datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tensorflow.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1, write_graph=
model = build_model(embedding_matrix)
model.fit([x_train,numerical_train], y_train,batch_size=BATCH_SIZE,epochs=15,verbose=2,validation_data=([x_cv,n
```


text_feature: InputLayer
input: [(?, ?)]
output: [(?, ?)]
embedding: Embedding
input: (?, ?)
output: (?, ?, 600)
spatial_dropout1d: SpatialDropout1D
input: (?, ?, 600)
output: (?, ?, 600)
bidirectional(cu_dnnlstm): Bidirectional(CuDNNLSTM)
input: (?, ?, 600)
output: (?, ?, 256)
bidirectional_1(cu_dnnlstm_1): Bidirectional(CuDNNLSTM)
input: (?, ?, 256)
output: (?, ?, 256)
global_max_pooling1d: GlobalMaxPooling1D
input: (?, ?, 256)
output: (?, 256)
global_average_pooling1d: GlobalAveragePooling1D
concatenate: Concatenate
input: [(?, 256), (?, 256),
output: (?, 768)
dense_1: Dense
input: (?, 768)
output: (?, 768)
add: Add
input: [(?, 768), (?, 768)]
output: (?, 768)
dense_2: Dense
input: (?, 768)
output: (?, 768)
add_1: Add
input: [(?, 768), (?, 768)]
output: (?, 768)
dense_3: Dense
input: (?, 768)
output: (?, 1)

```
def build_model(embedding_matrix):
    words = Input(shape=(None,),name="text_feature")
    x = Embedding(*embedding_matrix.shape, weights=[embedding_matrix], trainable=False)(words)
    x = SpatialDropout1D(0.5)(x)
    x = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x)
    x = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x)

    numerical_feats = Input(shape=(10,),name="numerical_features")
    numerical_featss = Dense(256,activation="relu",kernel_initializer="he_normal")(numerical_feats)

    hidden = concatenate([GlobalMaxPooling1D()(x),GlobalAveragePooling1D()(x),numerical_featss,])
    hidden = add([hidden, Dense(768, activation='relu')(hidden)])
    hidden = add([hidden, Dense(768, activation='relu')(hidden)])
    result = Dense(1, activation='sigmoid')(hidden)

    model = Model(inputs=[words,numerical_feats], outputs=result)
    model.compile(loss='binary_crossentropy', optimizer='adam')
    plot_model(model, to_file='/content/drive/My Drive/jigsaw/model44.png', show_shapes=True)

    return model
```

```
from datetime import datetime, timedelta

checkpoint = tensorflow.keras.callbacks.ModelCheckpoint('/content/drive/My Drive/jigsaw/Model44.hdf5', monit
log_dir="/content/drive/My Drive/jigsaw/Model44/logs/fit/" + datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tensorflow.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1, write_graph=
model = build_model(embedding_matrix)
model.fit([x_train,numerical_train], y_train,batch_size=BATCH_SIZE,epochs=5,verbose=2,validation_data=([x_cv,nu
```

TensorBoard SCALARS GRAPHS DISTRIBUTIONS HISTOGRAMS

```
# load the model
model1=tensorflow.keras.models.load_model('/content/drive/My Drive/jigsaw/Model22.hdf5')
MODEL_NAME = 'with_DO1'
X_te[MODEL_NAME] =model1.predict([x_te,numerical_te], batch_size=2048).flatten()
bias_metrics_df = compute_bias_metrics_for_model(X_te, IDENTITY_COLUMNS, MODEL_NAME, TARGET_COLUMN
get_final_metric(bias_metrics_df, calculate_overall_auc(X_te, MODEL_NAME))
```

0.932494270491741

Deep Learning Model 3: Without Dropout + text feature + Ad

1) **Business Problem**: First go through business problem, understand problem statement,define bu
understand data fields.
2)**Map the real-world problem to a Machine Learning Problem**: understand what type of Machine
metric.
3)**Work on Exploratory Data Analysis**: like loading data,understanding its toxic and non-toxic featu
and non-toxic words by plotting wordcloud, perform text preprocessing in which replace links with
which contains characters and numbers together,demojize i.e. convert emoji's into words,remove p
because having stopwords doesnt help in text classification.
4)**Feature Engineering**: we have added 'comment_word_count' ,'comment_char_count' new feature
by having distribution plot,boxplot,violinplot,kernel density estimate plot and then we visualize usir
5)**Machine Learning Models**

- **Train and Cv Split**: we do 80:20 split.
- **Make Data Model Ready**: encoding numerical, text features
- **Apply ML models**:Tune Hyperparameters of ML models,plot ROC curve and confusion matrix
  after that use custom metric to see how it will score on kaggle board.

6)**Give a try to Deep Learning Model**:build a model ,evauate using custom metric.
7)**Summarize Reults**

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.
2020-06-01 05:37:32,131 : WARNING : `write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard`
Epoch 1/15

Epoch 00001: val_loss improved from inf to 0.23934, saving model to /content/drive/My Drive/jigsaw/Mod
2856/2856 - 776s - loss: 0.4260 - val_loss: 0.2393
Epoch 2/15

Epoch 00002: val_loss did not improve from 0.23934
2856/2856 - 767s - loss: 0.4070 - val_loss: 0.2395
Epoch 3/15

Epoch 00003: val_loss improved from 0.23934 to 0.23773, saving model to /content/drive/My Drive/jigsaw,
2856/2856 - 772s - loss: 0.4009 - val_loss: 0.2377
Epoch 4/15

Epoch 00004: val_loss improved from 0.23773 to 0.23526, saving model to /content/drive/My Drive/jigsaw,
2856/2856 - 773s - loss: 0.3952 - val_loss: 0.2353
Epoch 5/15

Epoch 00005: val_loss did not improve from 0.23526
2856/2856 - 768s - loss: 0.3895 - val_loss: 0.2416
Epoch 6/15

Epoch 00006: val_loss did not improve from 0.23526
2856/2856 - 769s - loss: 0.3838 - val_loss: 0.2380
Epoch 7/15

Epoch 00007: val_loss did not improve from 0.23526
2856/2856 - 768s - loss: 0.3784 - val_loss: 0.2405
Epoch 8/15

Epoch 00008: val_loss did not improve from 0.23526
2856/2856 - 768s - loss: 0.3732 - val_loss: 0.2396
Epoch 9/15

Epoch 00009: val_loss did not improve from 0.23526
2856/2856 - 768s - loss: 0.3691 - val_loss: 0.2394
Epoch 10/15

Epoch 00010: val_loss did not improve from 0.23526
2856/2856 - 768s - loss: 0.3656 - val_loss: 0.2437
Epoch 11/15

Epoch 00011: val_loss did not improve from 0.23526
2856/2856 - 768s - loss: 0.3628 - val_loss: 0.2428
Epoch 12/15
```

```
%tensorboard --logdir='/content/drive/My Drive/jigsaw/Model33/logs/fit'
```

TensorBoard SCALARS GRAPHS DISTRIBUTIONS HISTOGRAMS

Histogram mode

OVERLAY OFFSET

Offset time axis

STEP RELATIVE WALL

Runs

Write a regex to filter runs

20200601-053732/train

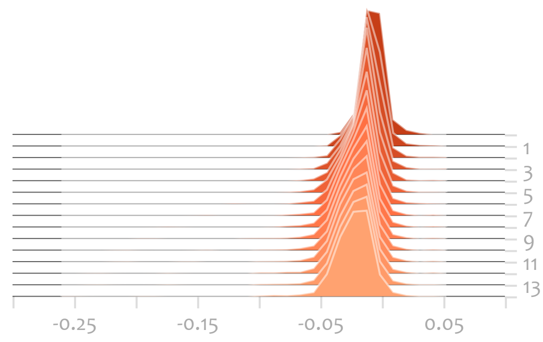
20200601-053732/validation

TOGGLE ALL RUNS

/content/drive/My Drive/jigsaw/Model33/logs/fit

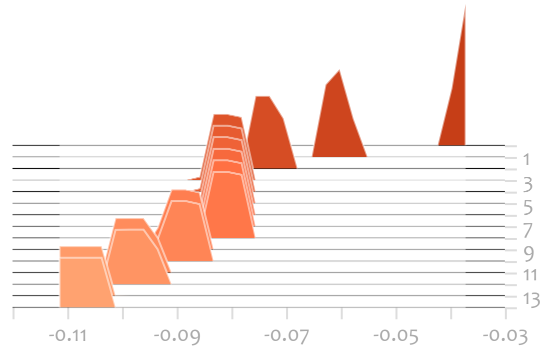
dense_2

dense_2/bias_0 20200601-053732/trai



dense_3

dense_3/bias_0 20200601-053732/trai




```
# load the model
model1=tensorflow.keras.models.load_model('/content/drive/My Drive/jigsaw/Model33.hdf5')
MODEL_NAME = 'withoutDP'
X_te[MODEL_NAME] =model1.predict([x_te,numerical_te], batch_size=2048).flatten()
bias_metrics_df = compute_bias_metrics_for_model(X_te, IDENTITY_COLUMNS, MODEL_NAME, TARGET_COLUMN
get_final_metric(bias_metrics_df, calculate_overall_auc(X_te, MODEL_NAME))
```

0.930495739941519

Deep Learning Model 4: With 0.5 Dropout + text feature + Ad

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.
2020-06-01 08:59:33,564 : WARNING : `write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard`
Epoch 1/5

Epoch 00001: val_loss improved from inf to 0.25312, saving model to /content/drive/My Drive/jigsaw/Mod
2856/2856 - 793s - loss: 0.4400 - val_loss: 0.2531
Epoch 2/5

Epoch 00002: val_loss improved from 0.25312 to 0.25297, saving model to /content/drive/My Drive/jigsaw,
2856/2856 - 794s - loss: 0.4153 - val_loss: 0.2530
Epoch 3/5

Epoch 00003: val_loss did not improve from 0.25297
2856/2856 - 788s - loss: 0.4106 - val_loss: 0.2542
Epoch 4/5

Epoch 00004: val_loss improved from 0.25297 to 0.25020, saving model to /content/drive/My Drive/jigsaw,
2856/2856 - 793s - loss: 0.4076 - val_loss: 0.2502
Epoch 5/5
```

```
%tensorboard --logdir='/content/drive/My Drive/jigsaw/Model44/logs/fit'
```

TensorBoard SCALARS GRAPHS DISTRIBUTIONS HISTOGRAMS

Histogram mode

OVERLAY OFFSET

dense_5

dense_5/bias_0 20200601-085933/trai

```
# load the model
model1=tensorflow.keras.models.load_model('/content/drive/My Drive/jigsaw/Model44.hdf5')
MODEL_NAME = 'DP'
X_te[MODEL_NAME] =model1.predict([x_te,numerical_te], batch_size=2048).flatten()
bias_metrics_df = compute_bias_metrics_for_model(X_te, IDENTITY_COLUMNS, MODEL_NAME, TARGET_COLUMN
get_final_metric(bias_metrics_df, calculate_overall_auc(X_te, MODEL_NAME))
```

0.9317927713596027

20200601-085933/train

From above trained models Model 2(Dropout 0.2 + text feature + Additional Fe lets load that trained model and predict on test data.

```
test = pd.read_csv('/content/drive/My Drive/test.csv')
x_test=pickle.load(open('/content/drive/My Drive/jigsaw/xtest','rb'))
numerical_test=pickle.load(open('/content/drive/My Drive/jigsaw/numericaltest','rb'))
```

```
# load the model
model1=tensorflow.keras.models.load_model('/content/drive/My Drive/jigsaw/Model22.hdf5')
predictions=model1.predict([x_test,numerical_test], batch_size=2048).flatten()

submission = pd.DataFrame.from_dict({
    'id': test.id,
    'prediction': predictions
})
submission.to_csv('/content/drive/My Drive/jigsaw/submission.csv', index=False)
```

On kaggle kernel we got a score of 0.93112

Submission and Description	Private Score	Public Score
kernel1a8ed46047 (version 2/2) a few seconds ago by Priyankaad From "kernel1a8ed46047" Script	0.93112	0.00000

Results

```
from prettytable import PrettyTable
import pandas as pd

x = PrettyTable()
x.field_names = ["Model", "ROC-AUC Score", "Custom  Metric Score"]
x.add_row(["Logistic Regression", 0.95,  0.8904733235626956])
x.add_row(["Naive Bayes\t", 0.88, 0.837811746713348])
x.add_row(["SVM\t",      0.95,   0.8814130929571368])
x.add_row(["Deep Learning", " - \t",  0.9319064728629746])
print(x)
```

```
+---------------------+---------------+----------------------+
|        Model        | ROC-AUC Score | Custom  Metric Score |
+---------------------+---------------+----------------------+
| Logistic Regression |      0.95     |  0.8904733235626956  |
|     Naive Bayes     |      0.88     |  0.837811746713348   |
|         SVM         |      0.95     |  0.8814130929571368  |
|    Deep Learning    |       -       |  0.9319064728629746  |
+---------------------+---------------+----------------------+
```

After adding additional features

```
x = PrettyTable()
x.field_names = ["Model\t\t\t\t\t", "\tCustom  Metric Score"]
x.add_row(["Dropout 0.2 + text feature + Additional Features\t\t", 0.9324942704917411])
x.add_row(["Without Dropout + text feature + Additional Features\t", 0.930495739941519])
x.add_row(["With 0.5 Dropout + text feature + Additional Features\t", 0.9317927713596027])
print(x)
```

```
+-------------------------------------------------------+-----------------------+
|                         Model                         |   Custom  Metric Score |
+-------------------------------------------------------+-----------------------+
|   Dropout 0.2 + text feature + Additional Features    |   0.9324942704917411  |
| Without Dropout + text feature + Additional Features  |   0.930495739941519   |
| With 0.5 Dropout + text feature + Additional Features |   0.9317927713596027  |
+-------------------------------------------------------+-----------------------+
```

Conclusion

1) As we have seen machine learning models performed very well when we use roc-auc metric but given compartively low score.
2) We have tried three machine learning models out of them Logistic regression has given higher s
3) To improve performance on custom metric we have trained **five** deep learning models out of wh **+ Additional Features"**which has given **0.9324 on custom metric** ,we can see that this score is mu models. 4)**After adding those additional features ,we have improved our model from 0.93190647**

Step by Step Procedure to solve this case study