Unintended Bias in Toxicity Classification

Team Number (or name): Group 2

Team Members:

Abdul Rahman Gulam Mohammed Hussain
Briana Haldaman
Maryam Hashemitaheri
Satya Kotha
Shilpa Patil

Course: DSBA/ MBAD 6211- Advanced Business Analytics

Instructor: Dr. Mousavi

Date Submitted: December 3rd, 2019

1 Executive Summary:

1.1 Business Problem Statement

In a world where communication is right at your fingertips, individuals have the ability to express their views and feelings across the globe many times without putting a name to those comments. This level of accessibility has resulted in an increased level of cyber-bullying. One in three young people experience cyberbullying through online platforms. Cyber-bullying takes many forms, some of which include sending mean messages, spreading rumors, posting harmful or threatening messages and photos. Those who experience this form or harassment can suffer from anxiety and depression into their adult years. Those who conduct the bullying also tend to engage in other poor decision making and behaviors such as; alcohol and drug abuse, early sexual activity, and abuse towards their significant other.

1.2 Business Goal

Through this project we seek to identify words that are used in online platforms, identify those considered toxic, and ensure to eliminate any bias in how those words are classified. This will ensure words that are not being used with a negative intention are not classified as toxic and in turn accurately protect online communities from cyberbullies. One of the recommended ways to remove AI model's bias is keeping humans in the loop. Our dataset includes target variable that represents the fraction of human raters who believed the attribute applied to the given comment. Comments with target value >0.5 are considered to be in positive class for toxicity. Each comment was labeled by 10 annotators. We believe if done properly, this model could identify users who demonstrate negative intentions in their posts towards others via online communities and platforms giving social media administrators the tools they need to be able to remove those offenders and their comments, and allow them to protect their users.

1.3 Data Profile

Our dataset was found on Kaggle and included 1,804,874 rows and 45 variable columns. The main variables that we will be using for our model include:

- target: toxicity label, based on this we are supposed to predict toxicity level for the test data. It is in fraction form and represents the fraction of human raters who believed the attribute applied to the given comment. Comments with target value >0.5 are considered to be in positive class for toxicity.
  Each comment was labeled by 10 annotators (some comments were shown to more than 10 annotators, due to sampling and strategies used to enforce rater accuracy).
  Annotators were asked to: "Rate the toxicity of this comment"
    - Very Toxic (a very hateful, aggressive, or disrespectful comment that is very likely to make you leave a discussion or give up on sharing your perspective)

- Toxic (a rude, disrespectful, or unreasonable comment that is somewhat likely to make you leave a discussion or give up on sharing your perspective)
- Hard to Say
- Not Toxic

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

- comment_text: Text from the individual comment

1.4 Results

After running the model, the confusion matrix and performance indicates an accuracy of 0.91. We were left with 1835 false positive cases in which the model predicted a word as toxic when it was not toxic. We also had 1908 false negatives in which the model predicted a word as non-toxic when in fact it was toxic.

	T	F
P	18204	1835
N	1908	18053

	precision	recall	f1-score	Support
0	0.91	0.91	0.91	20039
1	0.91	0.90	0.91	19961
accuracy			0.91	40000
macro avg	0.91	0.91	0.91	40000
weighted avg	0.91	0.91	0.91	40000

2 Project Report

2.1 Introduction

Today's modern world revolves around the internet, you can communicate across the entire world with one click of a button. Expressing one's views is as easy as typing on a smartphone. With this huge paradigm shift, it is important to recognize that new developing technologies come with both positive and negative aspects. People have a greater ability to express themselves over the internet via feedback, conversations, or comments. While the majority of users have positive intent and are able to articulate themselves well (the positive expressors), there are also many who have negative behaviors with the intention of being abusive, disrespectful and rude to others, making others to leave the online conversation or online community. This accessibility has caused increased levels of cyber-bullying. One in three young people experience cyberbullying through online platforms. Cyber-bullying takes many forms, some of which include sending mean messages, spreading rumors, posting harmful or threatening messages and photos. Those who experience this form or harassment can suffer from anxiety and depression into their adult years. Those who conduct the bullying also tend to engage in other poor decision making and behaviors such as; alcohol and drug abuse, early sexual activity, and abuse towards their significant other.

Currently, there are machine learning models built to identify the negative remarks/words. However, the models can incorrectly associate the names of frequently attacked identities with toxicity for example; LGBT community is one of the most attacked, which makes the machine think the word 'gay' is toxic. As such, sentences like *"We are proud gay couple",* are predicted with high toxicity when in reality they should not be categorized as toxic. Our project is designed to build a model that recognizes toxicity appropriately and help to minimize unintended bias with respect to mention of specific identities. Removing the bias in the model will allow for more accuracy in how cyberbullies are identified and removed from platforms where they can hurt others.

While identifying negative comments is important that we also respect the rights of the commenter's views and beliefs. If people limit themselves or completely stop expressing their views due to toxic comments from others, it can have downstream impacts. Therefore, we want to minimize the bias in identifying the toxicity which helps preserve the positive expression. We anticipate that this project will help to improve the prediction of the toxic words with accurate classification and minimize the bias.

2.2 Background

Data science has come a long way in the area of text analytics. There are similar projects and efforts being made to identify the bias of toxic language. Students from Hong Kong University of

Science and technology have worked on a project titled; *"Reducing gender bias in abusive language"*. In that project they were able to analyze the gender biases and worked on three bias mitigation methods. First, to debiased word embedding methods. Next, to gender swap data augmentation and finally, to fine tune with a larger corpus. Applying these methods, they estimated the gender bias was reduced by 90-98%. But their work is limited to the gender bias, and these classifiers/models are not designed to identify and predict other biases in the toxic comments. [1]

Students from Stanford University worked on a project titled: *"Toxic Speech Detection",* in this experiment they were able to use deep learning models in a cascaded way to achieve a high performing classifier on the data. While they were able to achieve the task, this work limited to a data set from a specific website, and it was not guaranteed that the same level of performance can be achieved for the other social media websites. [2]

There are other works done by many to debias word associations and many of them were all working with gender bias. Most of them were using the machine learning, deep learning and neural network-based approaches to build the models/classifiers. Classical methods like Regression, Bayesian and Random Forest etc.can be effective in use but they need a significant amount of work for building the models and classifiers.

Our approach is to start with pre-processing the data by cleaning, tokenizing, and removing the stop words. On the cleaned data, we will use stemming and lemmatization to combine similar tokens. Through binary classification we will split the data into toxic and non-toxic data. Then we will build an artificial neural network to process the train data. We wanted to use 'R' programming to build out models.

2.3 Data

The available data sets on this project website are 'train.csv', 'test.csv', and 'sample_submission'.

'train.csv' is the training data set upon which we built our model. There are 1,804,874 rows and 45 variable columns.

Besides the attributes mentioned in the Data Profile section, there are some additional toxicity subtype attributes. Since our test data doesn't have these attributes, we did not use them for in our model. Some comments have identity attributes, based on the identities mentioned in the comments. Similar to the target label, they are meant to collect the identity labels, annotators were asked to indicate identities mentioned in the comment. What gender was mentioned in the comment? (Male, female, Transgender, other gender, no gender mentioned). Those were

aggregated to fractional values representing the fraction of raters who said that particular identity was mentioned in the comment.

severe_toxicity	heterosexual	psychiatric_or_mental_illness
obscene	homosexual_gay_or_lesbian	transgender
identity_attack	intellectual_or_learning_disability	White
insult	jewish	created_date
threat	latino	Publication_id
sexual_explicit	male	parent_id
Asian	muslim	article_id
atheist	other_disability	rating
bisexual	other_gender	funny
black	other_race_or_ethnicity	wow
Buddhist	other_religion	sad
Christian	other_sexual_orientation	likes
female	physical_disability	disagree
toxicity_annotator_count	identity_annotator_count	

Some examples of how these labels are given:

I. “I picture this clown yelling "fathisht" with a massive lisp, what being on the Autism spectrum & all...”
    a. Toxicity label: 0.6
    b. Identity labels: Intellectually disabled: 0.7, other disability: 0.1
II. “Holy crap. How can anyone be dumb enough that they don't understand the difference between someone's genetic traits, and what essentially amounts to a fashion choice? It's unbelievable. \n\nNot to mention - you people have spent the last 20 years denying gays rights because it's supposedly a ""choice"". So your little hissy fit reveals one thing - that you're the real hypocrite here. \n\nBut you are a republican, so it's required for party membership. I won't hold it against you.”
    a. Toxicity label: 0.56
    b. Identity labels: homosexual_gay_or_lesbian: 0.83,

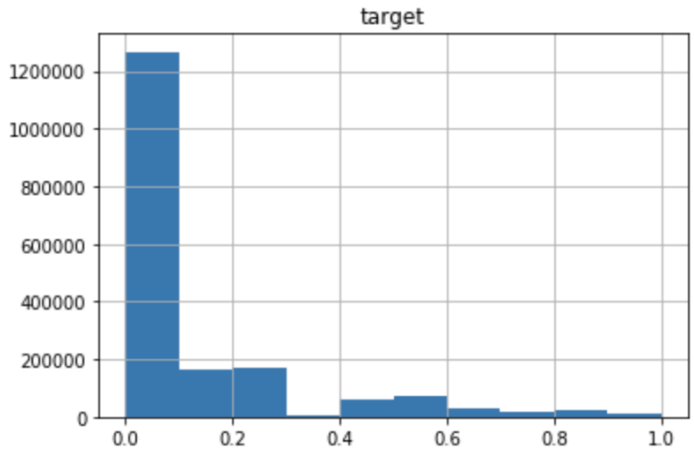
In addition to these labels, the dataset also provides metadata from Jigsaw's annotation: toxicity_annotator_count and identity_annotator_count, and metadata from Civil Comments: created_date, publication_id, parent_id, article_id, rating, funny, wow, sad, likes, disagree. Civil Comments' label 'rating'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

Cleaning and exploring the data:

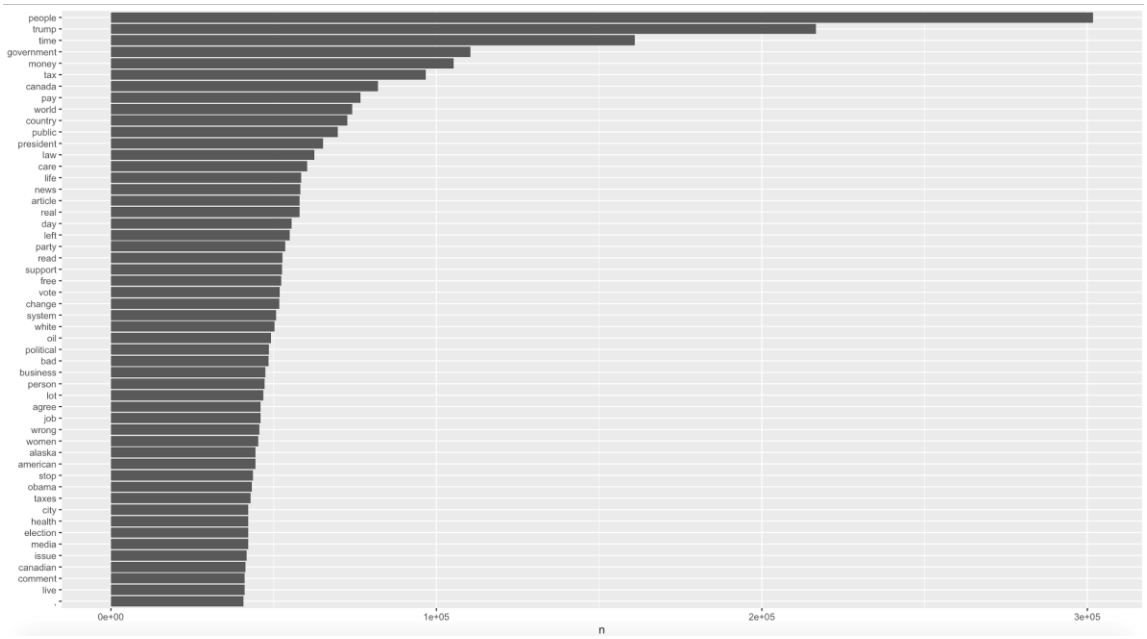
For this portion we used R programming.

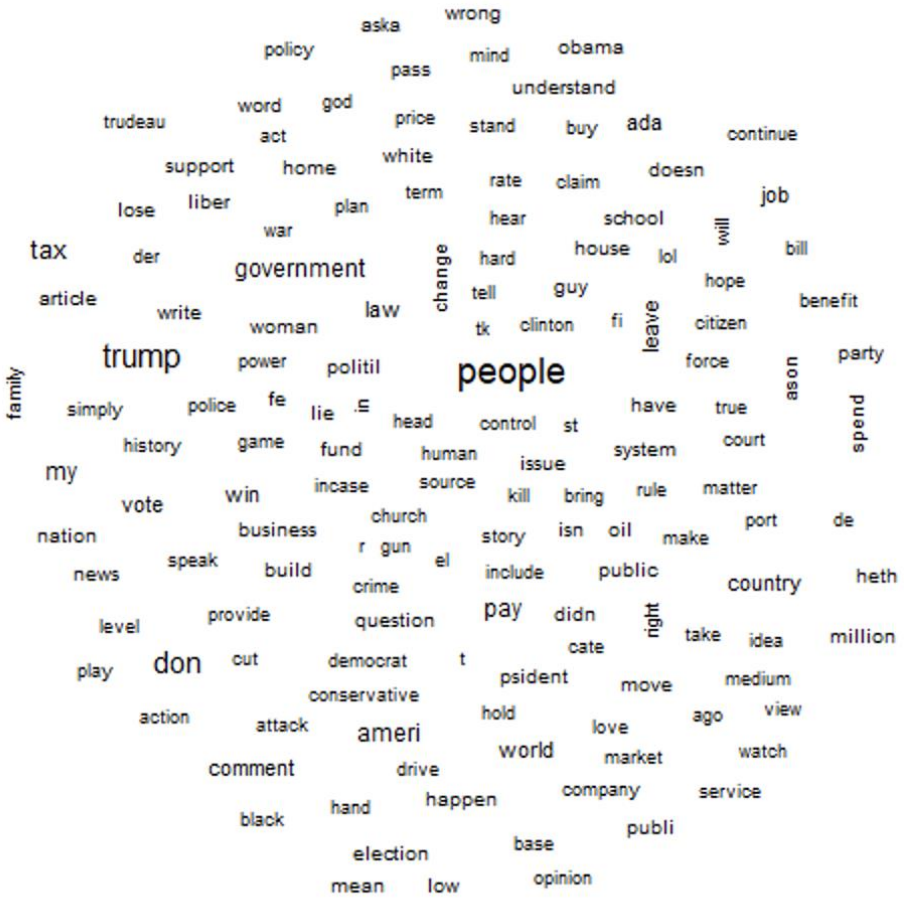
- Converted target values > 0.5 to 1 and <= 0.5 to 0.
- Comment_text attribute:
    - Removed punctuations, special characters, numbers, stop words
    - converted to lowercase
    - Tokenized
    - Then removed customized list of words (time, my, https, www, http, can, see, now, come, just, get, much, even, say, one, want, like, thing, run, kid, child, http, al, ca, fe, tk, re, line, life, day, call, bad, live, read, canadian, cost, post, start, lot, stop, agree, city, person)
    - Lemmatized the tokens

The histogram of the distribution of our target variable shows the distribution of the variable is largely skewed right with most observations between 0 and 0.3.



As we continue to explore the data, we find that the most common words plot and the word cloud shows the most frequent words in our dataset are: "people", "trump", "tax", and "government".


people
trump
time
government
money
tax
canada
pay
world
country
public
president
law
care
life
news
article
real
day
left
party
read
support
free
vote
change
system
white
oil
political
bad
business
person
lot
agree
job
wrong
women
alaska
american
stop
obama
taxes
city
health
election
media
issue
canadian
comment
live
0e+00
1e+05
2e+05
3e+05
n


wrong
aska
policy
mind
obama
pass
understand
word
god
price
stand
buy
ada
continue
trudeau
act
white
support
home
rate
claim
doesn
job
term
liber
plan
hear
school
lose
war
will
tax
der
government
change
hard
house
lol
bill
hope
article
write
law
tell
guy
benefit
woman
tk
clinton
fi
leave
citizen
trump
power
politil
people
force
party
family
simply
police
fe
lie
head
control
st
have
true
ason
spend
history
game
fund
human
system
court
my
issue
incase
source
kill
bring
rule
matter
vote
win
church
port
de
nation
business
r
gun
el
story
isn
oil
make
speak
build
include
public
country
heth
news
crime
pay
didn
right
level
provide
question
take
idea
million
cate
play
don
cut
democrat
t
psident
move
medium
conservative
action
attack
ameri
hold
love
ago
view
world
market
watch
comment
drive
company
service
happen
hand
black
publi
election
base
mean
low
opinion

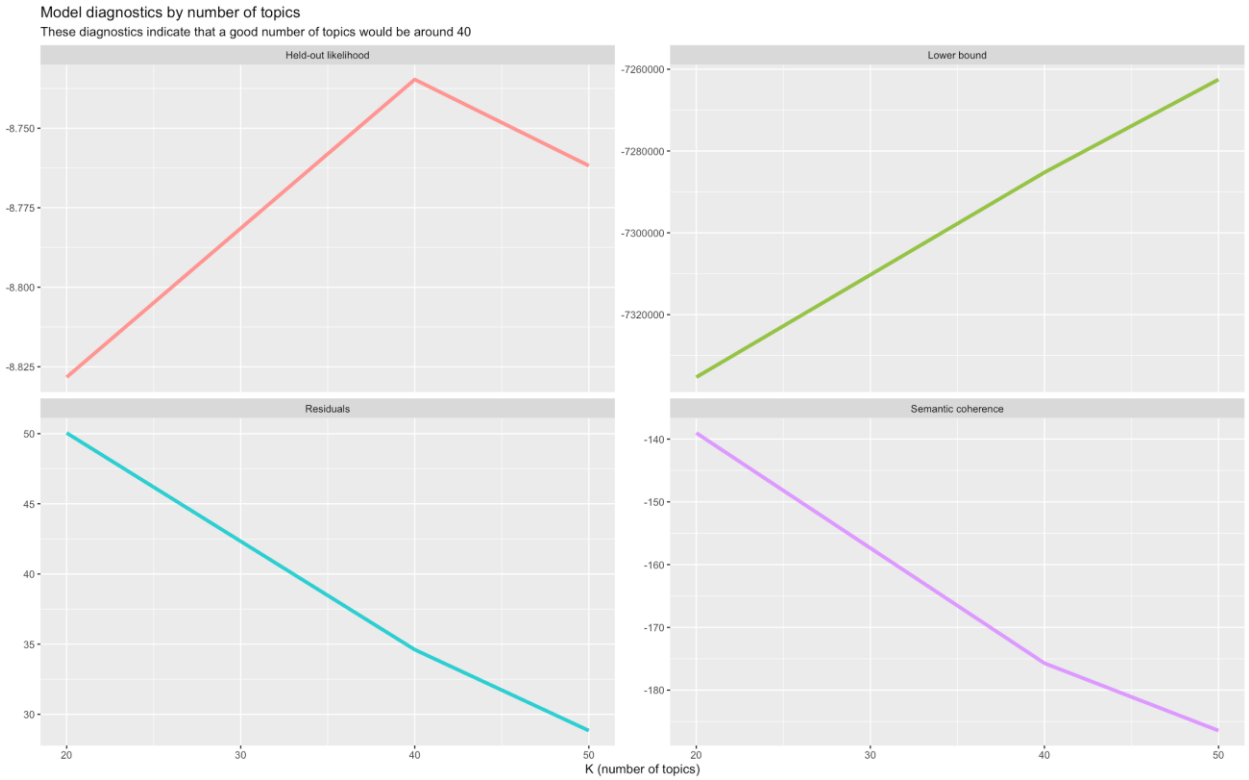
When reviewing the sentiment of the most frequent words, “support”, “win”, “love”, and “right” show as frequent positive words while “issue”, “wrong”, and “lose” are examples of some of the most frequent negatively associated words.

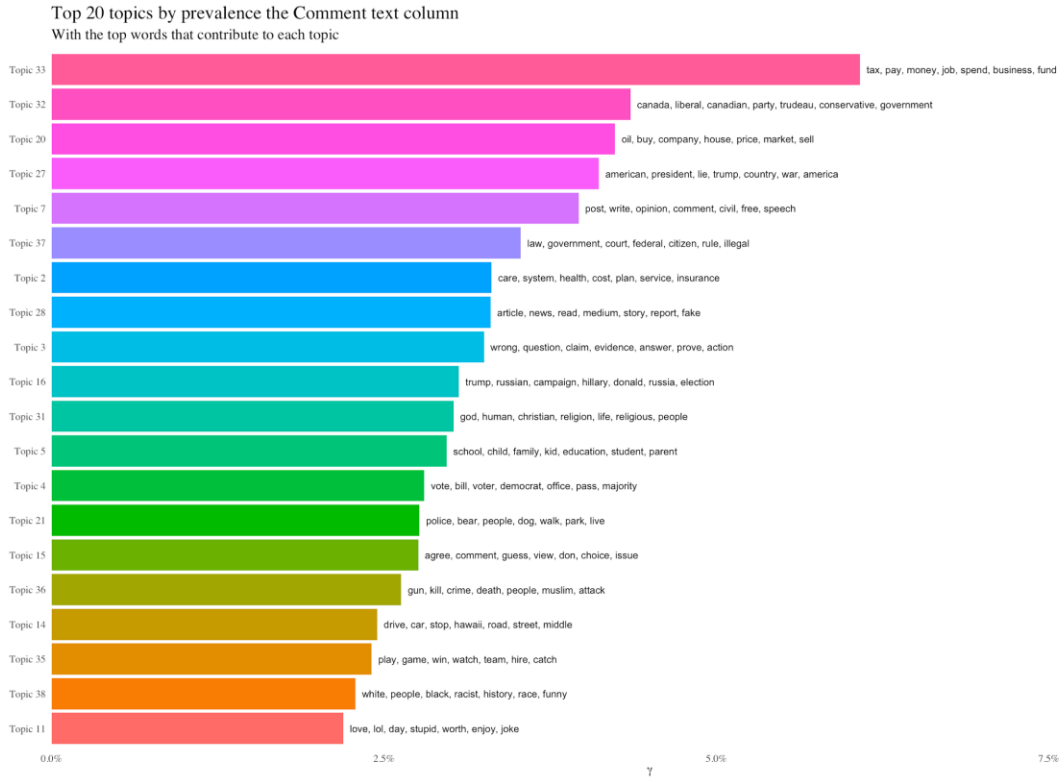
Structural Topic Model

We built an STM model to better understand causal inferences or covariates in our social media text. This visualization describes the prevalence of the topic within the entire corpus as well as the top seven words associated with the topic. This model is more useful than just understanding word frequency as it allows us to understand word groupings within the data.

We evaluated the model for several topic numbers based on residuals, semantic coherence of the topics (slows down at 40), held-out likelihood(highest), etc. found out 40 as the best number.

The graph for model diagnostics and top 20 topics is shown here.


Model diagnostics by number of topics
These diagnostics indicate that a good number of topics would be around 40
Held-out likelihood
Lower bound
Residuals
Semantic coherence
-8.750
-8.775
-8.800
-8.825
-7260000
-7280000
-7300000
-7320000
50
45
40
35
30
-140
-150
-160
-170
-180
20
30
40
50
K (number of topics)


Top 20 topics by prevalence the Comment text column
With the top words that contribute to each topic
Topic 33 tax, pay, money, job, spend, business, fund
Topic 32 canada, liberal, canadian, party, trudeau, conservative, government
Topic 20 oil, buy, company, house, price, market, sell
Topic 27 american, president, lie, trump, country, war, america
Topic 7 post, write, opinion, comment, civil, free, speech
Topic 37 law, government, court, federal, citizen, rule, illegal
Topic 2 care, system, health, cost, plan, service, insurance
Topic 28 article, news, read, medium, story, report, fake
Topic 3 wrong, question, claim, evidence, answer, prove, action
Topic 16 trump, russian, campaign, hillary, donald, russia, election
Topic 31 god, human, christian, religion, life, religious, people
Topic 5 school, child, family, kid, education, student, parent
Topic 4 vote, bill, voter, democrat, office, pass, majority
Topic 21 police, bear, people, dog, walk, park, live
Topic 15 agree, comment, guess, view, don, choice, issue
Topic 36 gun, kill, crime, death, people, muslim, attack
Topic 14 drive, car, stop, hawaii, road, street, middle
Topic 35 play, game, win, watch, team, hire, catch
Topic 38 white, people, black, racist, history, race, funny
Topic 11 love, lol, day, stupid, worth, enjoy, joke
0.0%
2.5%
5.0%
7.5%
γ

2.4 Method

The goal is to predict '**target**' for the test data. In the train data we have 43 more variables than in the test data. These 43 variables include '**target**' and 42 auxiliary variables. In Kaggle competition, the most common approach was to (1) extract 42 auxiliary variables for the test data and then (2) predict '**target**' for test records based on the extracted 42 auxiliary variables.

Here, we can use a completely different approach ignoring 42 auxiliary variables. We can only use '**comment_text**' to predict '**target**'. For this purpose, we better use word2vec representation of '**comment_text**' than using tf-idf or topic modeling.

we have a large data set of post ('**comment_text**') annotated by predicted class ('**target**'). We run a convolutional neural network (CNN) on the word-embedding representations of the '**comment_text**' trying to classify '**target**'. The model may use some pre-trained D-dimensional embedding vectors of each token, so a '**comment_text**' with the length of L (L tokens) will be represented by a matrix of size L×D. So, the CNN model will use L×D features to classify '**target**.

Each word in each '**comment_text**' is converted to a 300-D numeric vector, using pre-trained word embedding model **ConceptNet NumberBatch**. The logic behind using **NumberBatch** is to reduce the bias as much as possible. It is less biased compared to all the other well-known word embedding models.

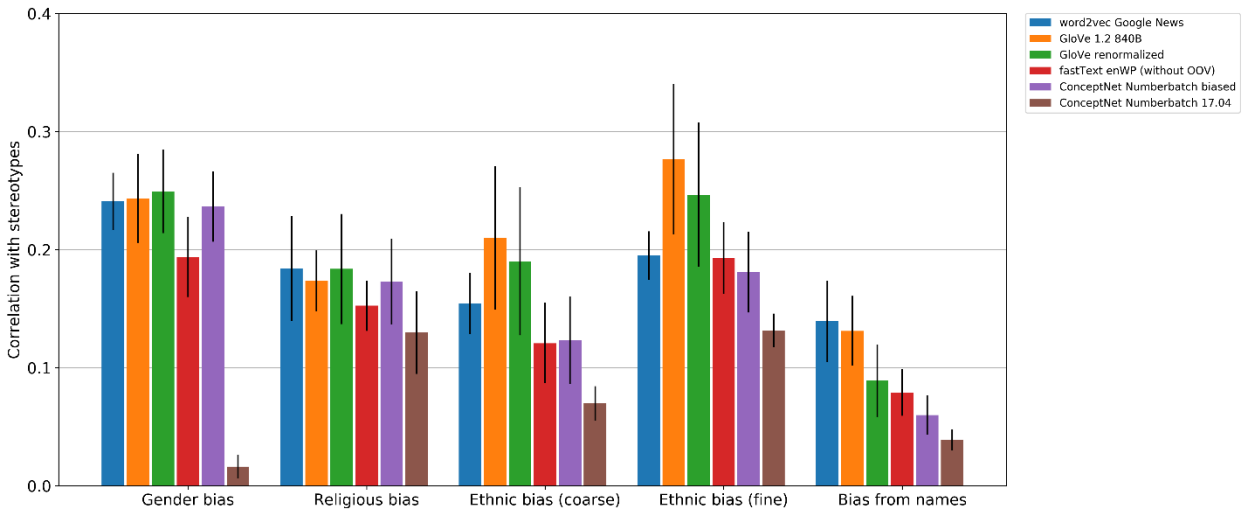


Figure 1: Comparison of bias introduced by different word embedding models.
http://blog.conceptnet.io/posts/2017/conceptnet-numberbatch-17-04-better-less-stereotyped-word-vectors/

As discussed before, **ConceptNet NumberBatch** is less biased compared to any other word2vec model, with respect to:

- Gender and sexual orientation
- Religion
- Ethnicity

- Special Names

Also, we can use a model that processes the joint effect of n-grams. For instance, in our CNN model, using convolution filter sizes of 2,3,... will enable the model to look at the words in the context that they appear. Therefore, the bias introduced by particular words will significantly decrease

Convolutional Neural Network

Our CNN model contains the following layers:

Input ↠Embedding ↠ Convolutional Filters ↠ Drop Out ↠ Dense Layer ↠ Sigmoid ↠ Output

For our CNN model, we use:

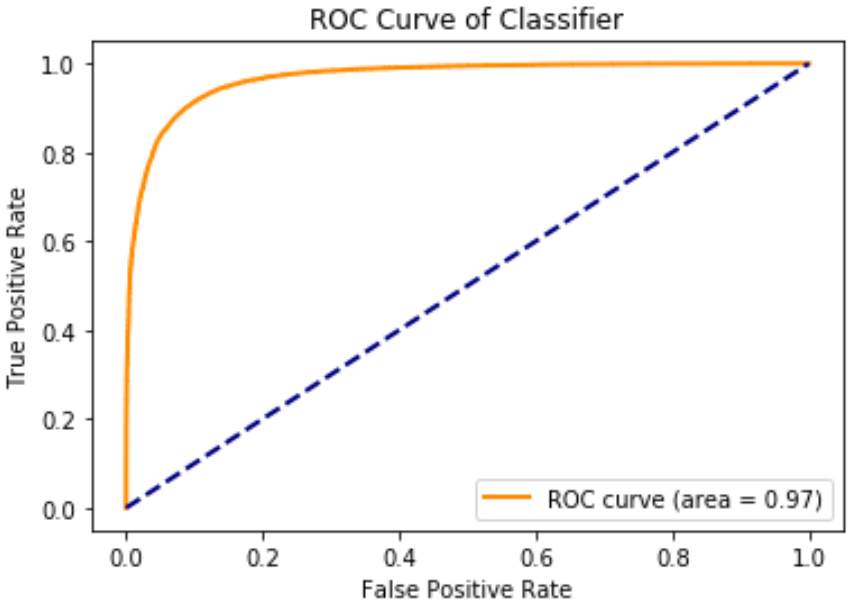
- Pad size = 150: Only 150 initial words for each record is used
- Convolutional Filters: We use filter sizes of 2,3, and 4 (100 filters of each size).
- Drop out = 40%: removing 40% of filters output will notably prevent over-fitting.
- Epochs = 10: The models iterated 10 times through the data.
- Batch size = 30: In each iteration, the model reads the data in blocks of 30 records.

2.5 Results & Discussions

After running the model, the confusion matrix and performance indicates an accuracy of 0.91 with an AUC of 0.97 as seen in the charts below.

	T	F
P	18204	1835
N	1908	18053

	precision	recall	f1-score	Support
0	0.91	0.91	0.91	20039
1	0.91	0.90	0.91	19961
accuracy			0.91	40000
macro avg	0.91	0.91	0.91	40000
weighted avg	0.91	0.91	0.91	40000



2.6 Conclusion

We believe our mode can assist social media platforms by classifying words into toxic and non-toxic and identify those users who are frequent offenders of toxic behaviors and language and ultimately eliminate them from their platforms. This will create a more positive social experience and make it more difficult for cyberbullies to persist.

3 Appendix

Citations:

1. Reference: Fung, P., Park J.H., Shin, J. 2018. "Reducing Gender Bias in Abusive Language Detection," Centre for Artificial Intelligence Research, pp. 1-6.
2. Reference: Koratana, A., Hu, K.2018. "Toxic Speech Detection," Stanford University, pp. 1-9.
3. Julia Silge - Training, evaluating, and interpreting topic models. (2019). Retrieved 4 December 2019, from https://juliasilge.com/blog/evaluating-stm/
4. Speer, R. (2019). ConceptNet Numberbatch 17.04: better, less-stereotyped word vectors. Retrieved 4 December 2019, from http://blog.conceptnet.io/posts/2017/conceptnet-numberbatch-17-04-better-less-stereotyped-word-vectors/

R code:

DSBA/ MBAD 6211: Advanced Business Analytics - Project Report

Unintended Bias in Toxicity Classification

project_stm.docx

Python Code:

```
import csv

import numpy as np

import pandas as pd

import scipy

import os

import re

import sklearn


import matplotlib.pyplot as plt

from matplotlib import cm, transforms


from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report


# from tensorflow import set_random_seed

# from tensorflow.keras import backend

from keras import backend

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad_sequences

from keras.layers import Embedding, Conv1D, Conv2D, MaxPooling1D, MaxPooling2D, MaxPool2D,
Dense, Input, GlobalMaxPooling1D, Dropout, Concatenate, Flatten

from keras.models import Model

from keras.layers.core import Reshape

# from tensorflow.keras.layers import Reshape

from keras import regularizers
```

```
from sklearn.model_selection import train_test_split

from keras.callbacks import ModelCheckpoint

from keras.initializers import Constant

from keras.optimizers import Adam


from sklearn.model_selection import train_test_split

C:\ProgramData\Anaconda3\lib\site-packages\h5py\__init__.py:34: FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.

  from ._conv import register_converters as _register_converters

Using TensorFlow backend.

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:458: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

  _np_qint8 = np.dtype([("qint8", np.int8, 1)])

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:459: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:460: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

  _np_qint16 = np.dtype([("qint16", np.int16, 1)])

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:461: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:462: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:465: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

```
path = 'C:\\Users\\Sh\\Desktop\\AdvBI\\py'
os.chdir(path)
```

Loading the data

```
train = pd.read_csv('_tr.csv')
test  = pd.read_csv('_ts.csv')
X_train = train['comment_text']
X_test  = test ['comment_text']
y_train = train['target']
# y_test  = test['target']
```

Making Balanced subsets

```
samples = 20000

X_train_pos = X_train[y_train >  0.5]
X_train_neg = X_train[y_train <= 0.5]

y_train_pos = y_train[y_train >  0.5]
y_train_neg = y_train[y_train <= 0.5]

X_train_pos = X_train_pos[:samples]
y_train_pos = y_train_pos[:samples]
X_train_neg = X_train_neg[:samples]
y_train_neg = y_train_neg[:samples]

X_train = pd.concat([X_train_pos, X_train_neg], axis=0)
y_train = pd.concat([y_train_pos, y_train_neg], axis=0)
```

```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.50, random_state=42)

X_test  = X_test [:(2*samples)]

data = pd.concat([X_train, X_val, X_test], axis=0)

Set Binary response for classification

y_train[y_train >  0.5] = 1

y_train[y_train <= 0.5] = 0

y_val  [y_val >  0.5] = 1

y_val  [y_val <= 0.5] = 0

Tokenizing the data, and making the dictionart

NUM_WORDS = 50000 # Max number of words to embed

tokenizer = Tokenizer(num_words=NUM_WORDS, filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',
lower=True)

tokenizer.fit_on_texts(data)

unique_words = tokenizer.word_index

del data

train_data = tokenizer.texts_to_sequences(X_train)

test_data  = tokenizer.texts_to_sequences(X_test)

val_data  = tokenizer.texts_to_sequences(X_val)

pad_size = 150

train_data = pad_sequences(train_data, maxlen = pad_size)

test_data  = pad_sequences(test_data , maxlen = pad_size)

val_data   = pad_sequences(val_data  , maxlen = pad_size)

Loading ConceptNet NumberBatch pre-trained vectors

Embedding_Dim = 300

exceptions = 0

embeddings_index = {}


with open('numberbatch-en-17.06_correct.txt', encoding="UTF-8") as f:
```

```
    for line in f:
        try:
            values = line.split(',')
            word = values[0]
            coefs = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = coefs
        except:
            exceptions = exceptions + 1


exceptions
25
Making the embedding matix, covering only the data dictionary
num_words = min(NUM_WORDS, len(unique_words)) + 1
embedding_matrix = np.zeros((num_words, Embedding_Dim))
for word, i in unique_words.items():
    if i > NUM_WORDS:
        continue
    embedding_vector = embeddings_index.get(word)

    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector

np.save('embedding_matrix.npy', embedding_matrix)
del embeddings_index
# get the number of words that are available in pre-trained embedding:
x = embedding_matrix[:,0]
print(len(x))
```

```
print(len(x[x!= 0]))
```

50001

36883

Converting the train and test data to emdedding data

```
train_data.shape
```

(20000, 150)

```
train_emb = np.zeros([train_data.shape[0], pad_size, Embedding_Dim])
test_emb  = np.zeros([test_data.shape[0] , pad_size, Embedding_Dim])
val_emb   = np.zeros([val_data.shape[0]  , pad_size, Embedding_Dim])


for i in np.arange(train_emb.shape[0]):
   train_emb[i,:,:] = [embedding_matrix[int(j),:] for j in train_data[i,:]]
for i in np.arange(test_emb.shape[0]):
   test_emb[i,:,:]  = [embedding_matrix[int(j),:] for j in test_data[i,:] ]
for i in np.arange(val_emb.shape[0]):
   val_emb[i,:,:]  = [embedding_matrix[int(j),:] for j in val_data[i,:] ]
print(train_emb.shape)
print(test_emb.shape)
print(val_emb.shape)
```

(20000, 150, 300)

(40000, 150, 300)

(20000, 150, 300)

Building the model

```
sequence_length = train_data.shape[1]
filter_sizes = [2,3,4]
num_filters = 100
drop = 0.4
```

```
epochs = 10

batch_size = 30


embedding = Input(shape=((sequence_length, Embedding_Dim,)))

layer_reshape = Reshape((sequence_length, Embedding_Dim, 1))

reshaped = layer_reshape(embedding)



layer_conv_0 = Conv2D(num_filters, kernel_size=(filter_sizes[0], Embedding_Dim),

                padding='valid', kernel_initializer='normal', activation='relu')

layer_conv_1 = Conv2D(num_filters, kernel_size=(filter_sizes[1], Embedding_Dim),

                padding='valid', kernel_initializer='normal', activation='relu')

layer_conv_2 = Conv2D(num_filters, kernel_size=(filter_sizes[2], Embedding_Dim),

                padding='valid', kernel_initializer='normal', activation='relu')


conv_0 = layer_conv_0(reshaped)

conv_1 = layer_conv_1(reshaped)

conv_2 = layer_conv_2(reshaped)


layer_maxpool_0 = MaxPool2D(pool_size=(sequence_length - filter_sizes[0] + 1, 1), strides=(1,1),
padding='valid')

layer_maxpool_1 = MaxPool2D(pool_size=(sequence_length - filter_sizes[1] + 1, 1), strides=(1,1),
padding='valid')

layer_maxpool_2 = MaxPool2D(pool_size=(sequence_length - filter_sizes[2] + 1, 1), strides=(1,1),
padding='valid')


maxpool_0 = layer_maxpool_0(conv_0)

maxpool_1 = layer_maxpool_1(conv_1)

maxpool_2 = layer_maxpool_2(conv_2)
```

```
layer_concatenated_tensor = Concatenate(axis=1)
concatenated_tensor = layer_concatenated_tensor([maxpool_0, maxpool_1, maxpool_2])

layer_flatten = Flatten()
flatten = layer_flatten(concatenated_tensor)
layer_dropout = Dropout(drop)
dropout = layer_dropout(flatten)

# layer_output = Dense(units =2, activation='softmax')
layer_output = Dense(units =1, activation='sigmoid')
# layer_output = Dense(units =1, activation='linear')
output = layer_output(dropout)

# model = Model(inputs=inputs, outputs=output)
model = Model(inputs=embedding, outputs=output)

checkpoint = ModelCheckpoint('weights.{epoch:03d}-{val_acc:.4f}.hdf5',
                             monitor = 'val_acc',
                             verbose=1,
                             save_best_only=True,
                             mode='auto')
adam = Adam(lr=1e-4, beta_1=0.9, beta_2=0.999, epsilon=1e-8, decay=0.0)
model.compile(optimizer = adam, loss='binary_crossentropy', metrics=['accuracy'])


# opt = Adam(lr=1e-3, decay=1e-3 / 200)
# model.compile(loss="mean_absolute_percentage_error", optimizer=opt)
Train the model on train_data
```

```
model.fit(train_emb, np.asarray(y_train),
        batch_size=batch_size,
        epochs=epochs, verbose=1,
#         callbacks=[checkpoint],
        validation_data = (val_emb, np.asarray(y_val)))
```

```
Train on 20000 samples, validate on 20000 samples
Epoch 1/10
20000/20000 [==============================] - 309s 15ms/step - loss: 0.6281 - acc: 0.6887 - val_loss: 0.5250 - val_acc: 0.8266
Epoch 2/10
20000/20000 [==============================] - 326s 16ms/step - loss: 0.4435 - acc: 0.8388 - val_loss: 0.3806 - val_acc: 0.8524
Epoch 3/10
20000/20000 [==============================] - 303s 15ms/step - loss: 0.3563 - acc: 0.8590 - val_loss: 0.3308 - val_acc: 0.8700
Epoch 4/10
20000/20000 [==============================] - 321s 16ms/step - loss: 0.3175 - acc: 0.8747 - val_loss: 0.3044 - val_acc: 0.8757
Epoch 5/10
20000/20000 [==============================] - 299s 15ms/step - loss: 0.2923 - acc: 0.8825 - val_loss: 0.2871 - val_acc: 0.8852
Epoch 6/10
20000/20000 [==============================] - 311s 16ms/step - loss: 0.2765 - acc: 0.8927 - val_loss: 0.2760 - val_acc: 0.8884
Epoch 7/10
20000/20000 [==============================] - 315s 16ms/step - loss: 0.2653 - acc: 0.8961 - val_loss: 0.2683 - val_acc: 0.8910
Epoch 8/10
20000/20000 [==============================] - 294s 15ms/step - loss: 0.2517 - acc: 0.9004 - val_loss: 0.2615 - val_acc: 0.8938
Epoch 9/10
```

20000/20000 [==============================] - 312s 16ms/step - loss: 0.2415 - acc: 0.9056 - val_loss: 0.2567 - val_acc: 0.8961

Epoch 10/10

20000/20000 [==============================] - 317s 16ms/step - loss: 0.2355 - acc: 0.9071 - val_loss: 0.2528 - val_acc: 0.8976

<keras.callbacks.History at 0x25a81b44b00>

Evaluation on val data (We do not have labels for test data.)

```
plt.hist(y_train)

plt.show()


pred = model.predict(val_emb)

# pred[pred > 1] = 1.0

# pred[pred < 0] = 0.0


conf_matrix = confusion_matrix(np.round(y_val), np.round(pred))

evaluation = classification_report(np.round(y_val), np.round(pred))

print(conf_matrix)

print(evaluation)
```

```
[[9030  974]
 [1073 8923]]
              precision    recall  f1-score   support

         0.0       0.89      0.90      0.90     10004
         1.0       0.90      0.89      0.90      9996

    accuracy                           0.90     20000
   macro avg       0.90      0.90      0.90     20000
weighted avg       0.90      0.90      0.90     20000
```

```
plt.hist(pred)
plt.show()


from sklearn import metrics
from sklearn.metrics import roc_auc_score


fpr_rf, tpr_rf, thresholds_rf = metrics.roc_curve(np.round(y_val), pred, pos_label=1)
roc_auc=roc_auc_score(np.round(y_val),pred)
print(roc_auc)


plt.figure()
lw = 2
plt.plot(fpr_rf, tpr_rf, color='darkorange',
        lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve of Classifier')
plt.legend(loc="lower right")
plt.savefig('__ROC.png')
plt.show()
0.9624480039916807


test_text = np.asarray(tokenizer.sequences_to_texts(test_data)
import csv
import numpy as np
import pandas as pd
import scipy
```

```
import os

import re

import sklearn


import matplotlib.pyplot as plt

from matplotlib import cm, transforms


from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report


# from tensorflow import set_random_seed

# from tensorflow.keras import backend

from keras import backend

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad_sequences

from keras.layers import Embedding, Conv1D, Conv2D, MaxPooling1D, MaxPooling2D, MaxPool2D,
Dense, Input, GlobalMaxPooling1D, Dropout, Concatenate, Flatten

from keras.models import Model

from keras.layers.core import Reshape

# from tensorflow.keras.layers import Reshape

from keras import regularizers

from sklearn.model_selection import train_test_split

from keras.callbacks import ModelCheckpoint

from keras.initializers import Constant

from keras.optimizers import Adam


from sklearn.model_selection import train_test_split
```

```
C:\ProgramData\Anaconda3\lib\site-packages\h5py\__init__.py:34: FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.

  from ._conv import register_converters as _register_converters

Using TensorFlow backend.

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:458: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

  _np_qint8 = np.dtype([("qint8", np.int8, 1)])

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:459: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:460: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

  _np_qint16 = np.dtype([("qint16", np.int16, 1)])

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:461: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:462: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

  _np_qint32 = np.dtype([("qint32", np.int32, 1)])

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:465: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

In [2]:

```
path = 'C:\\Users\\Sh\\Desktop\\AdvBI\\py'

os.chdir(path)
```

c. Loading the data

In [3]:

```
train = pd.read_csv('_tr.csv')

test  = pd.read_csv('_ts.csv')
```

In [4]:

```
X_train = train['comment_text']

# X_test  = test ['comment_text']

y_train = train['target']
```

III. Making Balanced subsets

In [5]:

```
samples = 40000


X_train_pos = X_train[y_train >  0.5]

X_train_neg = X_train[y_train <= 0.5]


y_train_pos = y_train[y_train >  0.5]

y_train_neg = y_train[y_train <= 0.5]


X_train_pos = X_train_pos[:samples]

y_train_pos = y_train_pos[:samples]

X_train_neg = X_train_neg[:samples]

y_train_neg = y_train_neg[:samples]


X_train = pd.concat([X_train_pos, X_train_neg], axis=0)

y_train = pd.concat([y_train_pos, y_train_neg], axis=0)
```

```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.50, random_state=42)

# X_test  = X_test [:(2*samples)]
```

In [6]:

```
# data = pd.concat([X_train, X_val, X_test], axis=0)

data = pd.concat([X_train, X_val], axis=0)
```

IV. Set Binary response for classification

In [7]:

```
y_train[y_train >  0.5] = 1

y_train[y_train <= 0.5] = 0

y_val  [y_val >  0.5] = 1

y_val  [y_val <= 0.5] = 0
```

V. Tokenizing the data, and making the dictionart

In [8]:

```
NUM_WORDS = 50000 # Max number of words to embed

tokenizer = Tokenizer(num_words=NUM_WORDS, filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',
lower=True)

tokenizer.fit_on_texts(data)

unique_words = tokenizer.word_index
```

In [9]:

```
del data
```

In [10]:

```
train_data = tokenizer.texts_to_sequences(X_train)

# test_data  = tokenizer.texts_to_sequences(X_test)

val_data  = tokenizer.texts_to_sequences(X_val)
```

In [11]:

```
pad_size = 150

train_data = pad_sequences(train_data, maxlen = pad_size)

# test_data  = pad_sequences(test_data , maxlen = pad_size)

val_data   = pad_sequences(val_data  , maxlen = pad_size)
```

a. Loading ConceptNet NumberBatch pre-trained vectors

In [12]:

```
Embedding_Dim = 300

exceptions = 0

embeddings_index = {}


with open('numberbatch-en-17.06_correct.txt', encoding="UTF-8") as f:
    for line in f:
        try:
            values = line.split(',')
            word = values[0]
            coefs = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = coefs
        except:
            exceptions = exceptions + 1
```

In [13]:

```
exceptions
```

Out[13]:

25

b. Making the embedding matix, covering only the data dictionary

In [14]:

```
num_words = min(NUM_WORDS, len(unique_words)) + 1
embedding_matrix = np.zeros((num_words, Embedding_Dim))
for word, i in unique_words.items():
    if i > NUM_WORDS:
        continue
    embedding_vector = embeddings_index.get(word)

    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector
```

In [15]:

```
np.save('embedding_matrix.npy', embedding_matrix)
del embeddings_index
```

In [16]:

```
# get the number of words that are available in pre-trained embedding:
x = embedding_matrix[:,0]
print(len(x))
print(len(x[x!= 0]))
```

50001

36205

VI. Converting the train and test data to emdedding data

In [17]:

```
train_data.shape
```

Out[17]:

(40000, 150)

In [18]:

```
train_emb = np.zeros([train_data.shape[0], pad_size, Embedding_Dim])
# test_emb  = np.zeros([test_data.shape[0] , pad_size, Embedding_Dim])
val_emb   = np.zeros([val_data.shape[0]  , pad_size, Embedding_Dim])

for i in np.arange(train_emb.shape[0]):
    train_emb[i,:,:] = [embedding_matrix[int(j),:] for j in train_data[i,:]]
# for i in np.arange(test_emb.shape[0]):
#     test_emb[i,:,:]  = [embedding_matrix[int(j),:] for j in test_data[i,:] ]
for i in np.arange(val_emb.shape[0]):
    val_emb[i,:,:]  = [embedding_matrix[int(j),:] for j in val_data[i,:] ]
```

In [19]:

```
print(train_emb.shape)
# print(test_emb.shape)
print(val_emb.shape)
```

(40000, 150, 300)

(40000, 150, 300)

VII. Building the model

In [20]:

```
sequence_length = train_data.shape[1]
filter_sizes = [2,3,4]
num_filters = 100
drop = 0.4
```

```
epochs = 10

batch_size = 30


embedding = Input(shape=((sequence_length, Embedding_Dim,)))

layer_reshape = Reshape((sequence_length, Embedding_Dim, 1))

reshaped = layer_reshape(embedding)



layer_conv_0 = Conv2D(num_filters, kernel_size=(filter_sizes[0], Embedding_Dim),

                padding='valid', kernel_initializer='normal', activation='relu')

layer_conv_1 = Conv2D(num_filters, kernel_size=(filter_sizes[1], Embedding_Dim),

                padding='valid', kernel_initializer='normal', activation='relu')

layer_conv_2 = Conv2D(num_filters, kernel_size=(filter_sizes[2], Embedding_Dim),

                padding='valid', kernel_initializer='normal', activation='relu')


conv_0 = layer_conv_0(reshaped)

conv_1 = layer_conv_1(reshaped)

conv_2 = layer_conv_2(reshaped)


layer_maxpool_0 = MaxPool2D(pool_size=(sequence_length - filter_sizes[0] + 1, 1), strides=(1,1),
padding='valid')

layer_maxpool_1 = MaxPool2D(pool_size=(sequence_length - filter_sizes[1] + 1, 1), strides=(1,1),
padding='valid')

layer_maxpool_2 = MaxPool2D(pool_size=(sequence_length - filter_sizes[2] + 1, 1), strides=(1,1),
padding='valid')


maxpool_0 = layer_maxpool_0(conv_0)

maxpool_1 = layer_maxpool_1(conv_1)

maxpool_2 = layer_maxpool_2(conv_2)
```

```
layer_concatenated_tensor = Concatenate(axis=1)
concatenated_tensor = layer_concatenated_tensor([maxpool_0, maxpool_1, maxpool_2])

layer_flatten = Flatten()
flatten = layer_flatten(concatenated_tensor)
layer_dropout = Dropout(drop)
dropout = layer_dropout(flatten)

# layer_output = Dense(units =2, activation='softmax')
layer_output = Dense(units =1, activation='sigmoid')
# layer_output = Dense(units =1, activation='linear')
output = layer_output(dropout)

# model = Model(inputs=inputs, outputs=output)
model = Model(inputs=embedding, outputs=output)

checkpoint = ModelCheckpoint('weights.{epoch:03d}-{val_acc:.4f}.hdf5',
                             monitor = 'val_acc',
                             verbose=1,
                             save_best_only=True,
                             mode='auto')
adam = Adam(lr=1e-4, beta_1=0.9, beta_2=0.999, epsilon=1e-8, decay=0.0)
model.compile(optimizer = adam, loss='binary_crossentropy', metrics=['accuracy'])


# opt = Adam(lr=1e-3, decay=1e-3 / 200)
# model.compile(loss="mean_absolute_percentage_error", optimizer=opt)
```

VIII. Train the model on train_data

In [21]:

```
model.fit(train_emb, np.asarray(y_train),
          batch_size=batch_size,
          epochs=epochs, verbose=1,
#           callbacks=[checkpoint],
          validation_data = (val_emb, np.asarray(y_val)))
```

Train on 40000 samples, validate on 40000 samples

Epoch 1/10

40000/40000 [==============================] - 600s 15ms/step - loss: 0.5343 - acc: 0.7638 - val_loss: 0.3791 - val_acc: 0.8545

Epoch 2/10

40000/40000 [==============================] - 639s 16ms/step - loss: 0.3376 - acc: 0.8660 - val_loss: 0.3053 - val_acc: 0.8760

Epoch 3/10

40000/40000 [==============================] - 623s 16ms/step - loss: 0.2900 - acc: 0.8836 - val_loss: 0.2774 - val_acc: 0.8874

Epoch 4/10

40000/40000 [==============================] - 633s 16ms/step - loss: 0.2658 - acc: 0.8921 - val_loss: 0.2634 - val_acc: 0.8948

Epoch 5/10

40000/40000 [==============================] - 607s 15ms/step - loss: 0.2500 - acc: 0.8996 - val_loss: 0.2519 - val_acc: 0.8986

Epoch 6/10

40000/40000 [==============================] - 606s 15ms/step - loss: 0.2374 - acc: 0.9055 - val_loss: 0.2455 - val_acc: 0.9004

Epoch 7/10

40000/40000 [==============================] - 621s 16ms/step - loss: 0.2275 - acc: 0.9091 - val_loss: 0.2399 - val_acc: 0.9029

Epoch 8/10

40000/40000 [==============================] - 635s 16ms/step - loss: 0.2198 - acc: 0.9125 - val_loss: 0.2360 - val_acc: 0.9047

Epoch 9/10

40000/40000 [==============================] - 639s 16ms/step - loss: 0.2123 - acc: 0.9165 - val_loss: 0.2327 - val_acc: 0.9048

Epoch 10/10

40000/40000 [==============================] - 642s 16ms/step - loss: 0.2068 - acc: 0.9183 - val_loss: 0.2304 - val_acc: 0.9064

Out[21]:

<keras.callbacks.History at 0x1e68a53bb38>

a. Evaluation on val data (We do not have labels for test data.)

In [22]:

```
plt.hist(y_train)

plt.show()
```

In [23]:

```
pred = model.predict(val_emb)

# pred[pred > 1] = 1.0

# pred[pred < 0] = 0.0


conf_matrix = confusion_matrix(np.round(y_val), np.round(pred))

evaluation = classification_report(np.round(y_val), np.round(pred))

print(conf_matrix)

print(evaluation)
```

```
[[18204  1835]
```

```
[ 1908 18053]]
              precision    recall  f1-score   support

         0.0       0.91      0.91      0.91     20039
         1.0       0.91      0.90      0.91     19961

    accuracy                           0.91     40000
   macro avg       0.91      0.91      0.91     40000
weighted avg       0.91      0.91      0.91     40000
```

In [25]:

```
plt.hist(pred)
plt.show()
```

In [26]:

```
from sklearn import metrics
from sklearn.metrics import roc_auc_score

fpr_rf, tpr_rf, thresholds_rf = metrics.roc_curve(np.round(y_val), pred, pos_label=1)
roc_auc=roc_auc_score(np.round(y_val),pred)
print(roc_auc)

plt.figure()
lw = 2
plt.plot(fpr_rf, tpr_rf, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
```

```
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve of Classifier')
plt.legend(loc="lower right")
plt.savefig('__ROC2.png')
plt.show()
```

```
0.9677975525501936
```

In [27]:

```
val_text = np.asarray(tokenizer.sequences_to_texts(val_data)
```