Project report towards Udacity Machine Learning Advanced Nanodegree Program Capstone project.

Author: Prashant Tripathi (xprashanttr@gmail.com)

This page is intentionally left blank

Table of Contents

(1) Definition

(1.1) Project Overview

The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions. Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments.

The Conversation AI team, a research initiative founded by Jigsaw and Google (both a part of Alphabet) are working on tools to help improve online conversation. One area of focus is the study of negative online behaviours, like toxic comments (i.e. comments that are rude, disrespectful or otherwise likely to make someone leave a discussion). So far they've built a range of publicly available models served through the Perspective API, including toxicity. But the current models still make errors, and they don't allow users to select which types of toxicity they're interested in finding (e.g. some platforms may be fine with profanity, but not with other types of toxic content).

I have chosen "Toxic Comment Classification Challenge" on Kaggle for my Advanced Machine Learning Nanodegree Capstone Project.

Kaggle Competition title: Jigsaw Toxic Comment Classification Challenge

Kaggle Competition URL: https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge

My Motivation:

Text analytics and processing always excited me. In 2015, I did a basic reading on text analytics and came across many concepts on why text universe is different from other areas of analytics.

My Journey in Udacity Nanodegree and particularly this capstone project has been full of learning excitement. I initially wanted to build a chatbot and started working on that but faced a lot of infrastructure issues (memory, processing power etc – I own a 4GB RAM, Intel Core i5-4210U CPU laptop). I almost gave up on Toxic comment project also until Udacity Mentor told me about Google-Colab platform.

This project has been run on Google-Colab infrastructure.

My next goal is to build a classifier for fake news detection. Toxic comment classification project is a step towards it.

I believe this project has significant importance in terms of the value that it can add to social media platforms; biggest of them would be creating social media platforms more censored hence suitable for everyone in today's connected world.

(1.2) Problem Statement

Problem statement is to build a multi-headed model that's capable of detecting different types of toxicity like threats, obscenity, insults, and identity-based hate. This is a multi-label classification problem whereby each comment can belong to one or more class. There are 6 classes as given below.

toxic	severe_toxic	obscene	threat	insult	identity_hate

Dataset provided is of comments from Wikipedia's talk page edits.

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

Data provided has large number of Wikipedia comments which have been labelled by human raters for toxic behaviour into 6 classes mentioned above. Data files being provided are train, test and sample submission files.

train.csv - the training set, contains comments with their binary labels. Below is structure.

id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate

Sample Screenshot:

id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0000997932d777bf	Explanation	0	0	0	0	0	0
000103f0d9cfb60f	D'aww! He matches t	0	0	0	0	0	0
000113f07ec002fd	Hey man, I'm really n	0	0	0	0	0	0
0001b41b1c6bb37e	"	0	0	0	0	0	0
0001d958c54c6e35	You, sir, are my hero.	0	0	0	0	0	0
00025465d4725e87	"	0	0	0	0	0	0

test.csv - the test set, model must predict the toxicity probabilities for these comments. To deter hand labelling, the test set contains some comments which are not included in scoring. Below is structure.

id	comment_text

Sample Screenshot:

id	comment_text
000634272d0d44eb	==Current Position==
000663aff0fffc80	this other one from 1897
000689dd34e20979	== Reason for banning throwing
000834769115370c	:: Wallamoose was changing the cite
000844b52dee5f3f	|blocked]] from editing Wikipedia.

sample_submission.csv - A sample submission file in the correct format. Below is structure.

id	toxic	severe_toxic	obscene	threat	insult	identity_hate

Sample Screenshot:

id	toxic	severe_toxic	obscene	threat	insult	identity_hate
000634272d0d44eb	0.5	0.5	0.5	0.5	0.5	0.5
000663aff0fffc80	0.5	0.5	0.5	0.5	0.5	0.5
000689dd34e20979	0.5	0.5	0.5	0.5	0.5	0.5
000834769115370c	0.5	0.5	0.5	0.5	0.5	0.5
000844b52dee5f3f	0.5	0.5	0.5	0.5	0.5	0.5

In this submission, I have used Stratified Shuffle Split for preparing training and evaluation dataset. I have used tfidf Vectorizer for creating word2vec embeddings. I have used 2 classifiers – RF and NN within a pipeline as model. Details are mentioned in section 2.2.

(1.3) Metrics

It is a multi label classification problem (not multi class classification), where each record can belong to more than one class. I have used ROC AUC (Receiver Operating Characteristic – Area Under Curve) to evaluate performance of the model.

ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- True Positive Rate
- False Positive Rate

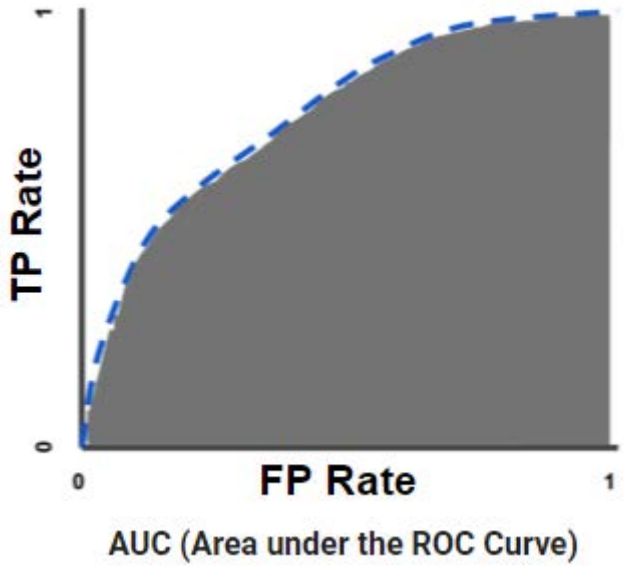
True Positive Rate (TPR) is a synonym for recall and is therefore defined as follows:

$$TPR = \frac{TP}{TP + FN}$$

False Positive Rate (FPR) is defined as follows:

$$FPR = \frac{FP}{FP + TN}$$

AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve from (0,0) to (1,1).



AUC (Area under the ROC Curve)

AUC is desirable for the following two reasons:

- AUC is **scale-invariant**. It measures how well predictions are ranked, rather than their absolute values.

- AUC is **classification-threshold-invariant**. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.

  Kaggle competition moved to ROC AUC in mid of competition, after test dataset was revised. The fundamental reason was that the test set and the training set were drawn from somewhat different distributions and there were cross validation issues. Details can be found at Kaggle Discussion Forum.

  As mentioned in the proposal document - I intended to achieve AUC of min 0.80. This value has been met.

(2)Analysis

(2.1)Data Exploration & Exploratory Visualization

Below is high level overview of training data provided.

Data counts in train.csv	
Header present?	Yes
Total number of data records	159571
Data records classified as toxic	15294
Data records classified as severe_toxic	1595
Not classified as Toxic but classified as severe_toxic	0
Data records classified as obscene	8449
Data records classified as threat	478
Data records classified as insult	7877
Data records classified as identity_hate	1405
Data records classified in one or more classes	16225
Data records not classified in any class	143346

test.csv has 153164 data records and one header row. Sample submission file has ID column with 6 given classes against it. Model should predict probability of comments corresponding to ID column for all records.

(a)Sample of training data:

A	B	C	D	E	F	G	H
id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
00040093b2687caa	alignment on this sut	0	0	0	0	0	0
0005300084f90edc	"	0	0	0	0	0	0
00054a5e18b50dd4	bbq	0	0	0	0	0	0
0005c987bdfc9d4b	Hey... what is it..	1	0	0	0	0	0
0006f16e4e9f292e	Before you start	0	0	0	0	0	0
00070ef96486d6f9	Oh, and the girl abov	0	0	0	0	0	0

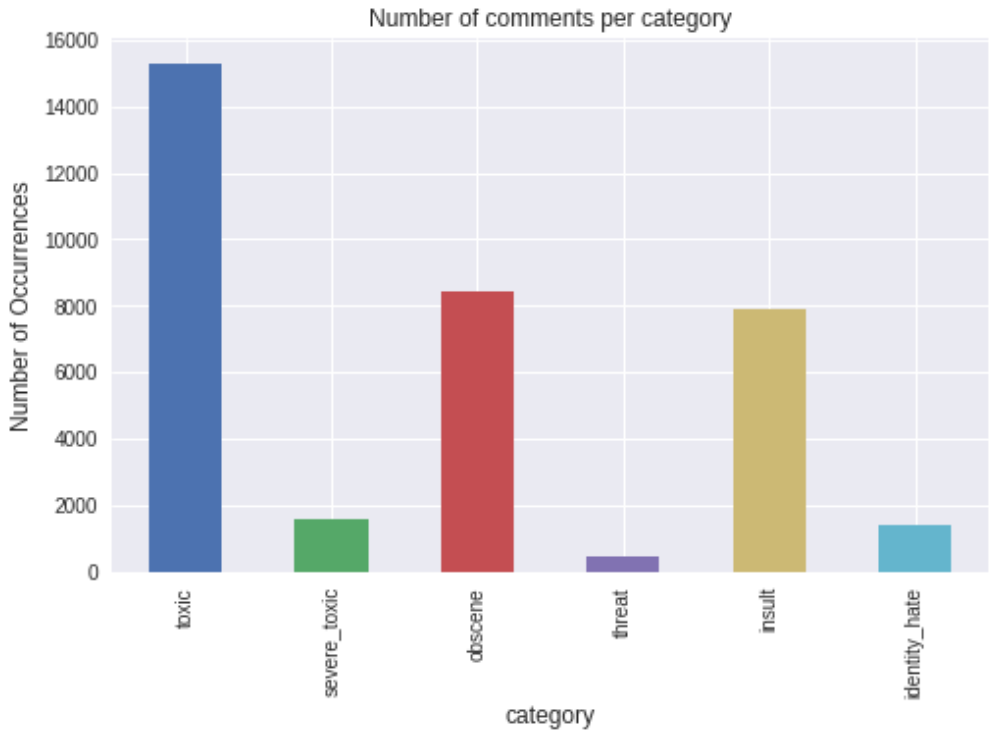
(b)Sample of data classified in more than one classes:

A	B	C	D	E	F	G	H
id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
001810bf8c45bf5f	You are gay or	1	0	1	0	1	1
00190820581d90ce	FUCK YOUR FILTHY M	1	0	1	0	1	0
001dc38a83d420cf	GET FUCKED UP. GET	1	0	1	0	0	0
0020e7119b96eeeb	Stupid peace of shit s	1	1	1	0	1	0
0020fd96ed3b8c8b	=Tony Sidaway is obv	1	0	1	0	1	0
0021fe88bc4da3e6	My Band Page's	1	0	1	0	0	0
0028d62e8a5629aa	All of my edits are go	1	0	1	0	1	0
00472b8e2d38d1ea	A pair of jew-hating v	1	0	1	0	1	1

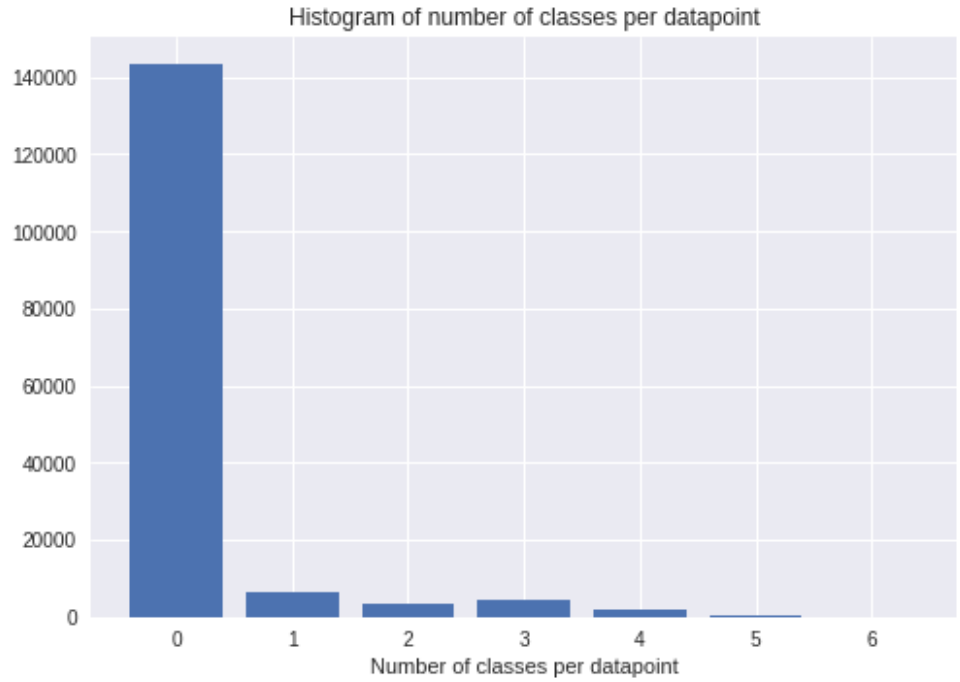
(c)Sample of data classified under none of the classes:

id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
00169857adbc989b	Hi Explicit, can you bl	0	0	0	0	0	0
0016e01b742b8da3	Notability of Rurika	0	0	0	0	0	0
001735f961a23fc4	"	0	0	0	0	0	0
00173958f46763a2	TFD	0	0	0	0	0	0
001b2dd65d9d925c	I don't believe the	0	0	0	0	0	0
001c419c445b5a59	You had a point, and	0	0	0	0	0	0
001c557175094f10	In other words, you'r	0	0	0	0	0	0
001cadfd324f8087	"	0	0	0	0	0	0
001d874a4d3e8813	":::Jmabel; in	0	0	0	0	0	0
001d8e7be417776a	"	0	0	0	0	0	0

(d) Number of comments per category : Below is distribution of #records per class label -



(e) Number of class labels VS data records: Below is distribution –



It suggests that there is major portion training data which is not labelled against any Class. Upon analysis it was found that 89+ % training records are not labelled against any class.

```
Percentage of comments that are not labelled:
0.8983211235124177
```

This is a very interesting situation. Just predicting any comment as not under any of the categories can give us misleading metric for model verification. E.g. Accuracy can be considerable if we just mark a comment as not under any class. Hence it shouldn't be a metric for this problem.

(2.2)Algorithms and Techniques

Data Preparation: I have used Stratified Shuffle Split {sklearn.model_selection.StratifiedShuffleSplit} for training and evaluation data creation.

Embedding: I have used tfidf word2vec conversion { sklearn.feature_extraction.text.TfidfVectorizer}, both with and without ngram.

Classifier: I have used two stage classifier – First stage with RF{sklearn.ensemble.RandomForestClassifier} and second stage with NN{ sklearn.neural_network.MLPClassifier}.

Basic idea behind using all above algorithms in combination is to provide better coverage from bias outside of one model. Parameters are mentioned in "section 3.2 & 4.1" of this document.

A **random forest** is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are by default drawn with replacement.

MLPClassifier is a multi-layer perceptron (MLP) **Neural Network** supervised algorithm that trains using Backpropagation. It trains using some form of gradient descent and the gradients are calculated

using Backpropagation. For classification, it minimizes the Cross-Entropy loss function, giving a vector of probability estimates per sample. The reason I have used MLPClassifier is that it supports multi-label classification in which a sample can belong to more than one class. For each class, the raw output passes through the logistic function. Values larger or equal to 0.5 are rounded to 1, otherwise to 0.

Parameters used for these functions for multiple runs before coming to submission code version are mentioned in the section “section 3.2 & 4.1” of this document.

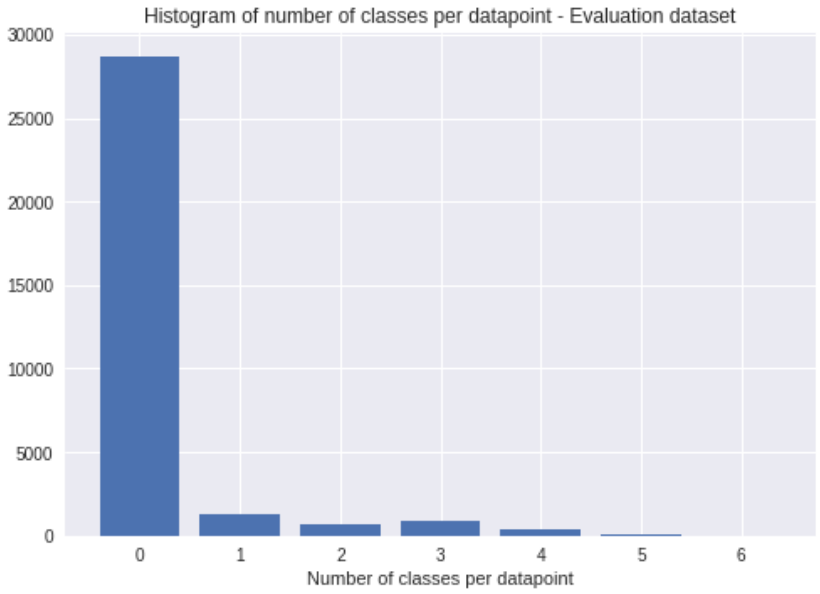
Libraries used in this submitted code :

imports

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import scipy.sparse as sp
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
import sklearn.metrics as metrics
from sklearn.multioutput import MultiOutputClassifier
import time
from sklearn.pipeline import Pipeline
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression
import seaborn as sns
```

Below are **basic characteristic of train and evaluation data** created:

```
---------------- Training Data ----------------
Number of records in Training data set: 127656
Number of records of type toxic: 12218
Number of records of type severe_toxic: 1271
Number of records of type obscene: 6788
Number of records of type threat: 402
Number of records of type insult: 6292
Number of records of type identity_hate: 1108
---------------- Evaluation Data ----------------
Number of records in Evaluation data set: 31915
Number of records of type toxic: 3076
Number of records of type severe_toxic: 324
Number of records of type obscene: 1661
Number of records of type threat: 76
Number of records of type insult: 1585
Number of records of type identity_hate: 297
```



These are proportionally in sync with distribution of overall training data provided. There are more distributions mentioned in the notebook.

(2.3)Benchmark

As part of Conversation AI project (a collaborative research effort exploring ML as a tool for better discussions online) , There are existing text classification models , existing datasets etc. One of the project I came across was on Github : https://github.com/conversationai/unintended-ml-bias-analysis.This reference is mentioned in Kaggle Competition Description page.

I have used tfidf vectoriser { sklearn.feature_extraction.text.TfidfVectorizer } for embedding and logistic regression { sklearn.linear_model.LogisticRegression } model as benchmark model. It produced AUC of 0.700092. This model is present in Notebook under section “Benchmark model”.

This will be baseline metric. As mentioned before – I intended to achieve AUC of min 0.80 in my solution. This benchmark value has been met.

(3)Methodology

(3.1)Data Pre-processing

Since it is a text analysis and processing problem on social media platform, I have not done any pre-processing.

For word to vector embedding – I have used tfidf {sklearn.feature_extraction.text.TfidfVectorizer} . Have used this with and without ngram both.

(3.2)Implementation & Refinement

Implementation steps with descriptive comments in Notebook file attached in project submission zip file. Project is implemented on Google Colab { https://colab.research.google.com } as advised by Udacity mentor. I faced lot of infrastructure issues while setting up project on my laptop.

Some unix commands are also used. So code is not as-it-is re-executable on other platforms.

One of the **challenges** faced during projects: I wanted to use pre-trained vectors for word to vector embedding. There are 2 sets of pre-trained vectors I could find. These are widely used as per my findings:

Pre-Trained vectors mentioned on Facebook research page- https://github.com/facebookresearch/fastText/blob/master/docs/english-vectors.md

Pre-Trained vectors from Stanford research (GloVe)– https://nlp.stanford.edu/projects/glove/

But faced MemoryError in loading of embedding files (I tried with files with highest number of tokens).

There are two stages in which code was executed:

First (initial) stage was – only plain tfidf embedding(without ngram), default parameters in RF and NN models, and single split stratified shuffle with 80% data for testing.

Second (current) stage was – plain and ngram tfidf embeddings, tunned parameters in RR and NN models, and 10 split stratified shuffle with 80% train and 20% test data.

Second stage of code was achieved after multiple runs and parameter improvement. Below is a summary of how I ran the code and tuned it to submission. I could not use Grid Search CV in this submission because of time constraints. I did some reading about it though after Review1. This seems to be major value addition for me. I will use it later. It will save a lot of time also.

Run Order	StratifiedShuffleSplit	RF	NN	tfidf_vectorizer	ngram_tfidf_vectorizer	ROC-AUC
1	v_n_splits=1 v_train_size=0.8 v_test_size=0.2	Default n_estimators = 10 n_jobs=-1	Default random_state=1 #seed	Default max_features = 5000	Default max_features = 5000 ngram_range=(2,3)	0.94453
2	v_n_splits=5	Default	Default	Default	Default	0.945543

	v_train_size=0.8 v_test_size=0.2	n_estimators = 10 n_jobs=-1	random_state=1 #seed	max_features = 5000	max_features = 5000 ngram_range=(2,3)	
3	v_n_splits=10 v_train_size=0.8 v_test_size=0.2	Default n_estimators = 10 n_jobs=-1	Default random_state=1 #seed	Default max_features = 5000	Default max_features = 5000 ngram_range=(2,3)	0.944654
4	v_n_splits=10 v_train_size=0.8 v_test_size=0.2	Default n_estimators = 10 n_jobs=-1	Default random_state=1 #seed	max_features = 5000 max_df=0.5 min_df=2	max_features = 5000 ngram_range=(2,3) max_df=0.5 min_df=1	0.944689
5	v_n_splits=10 v_train_size=0.8 v_test_size=0.2	n_estimators = 100 n_jobs=-1	Default random_state=1 #seed	max_features = 5000 max_df=0.5 min_df=2	max_features = 5000 ngram_range=(2,3) max_df=0.5 min_df=1	0.947627
6	v_n_splits=10 v_train_size=0.8 v_test_size=0.2	n_estimators = 100 n_jobs=-1	hidden_layer_sizes=(100,100) random_state=1 #seed	max_features = 5000 max_df=0.5 min_df=2	max_features = 5000 ngram_range=(2,3) max_df=0.5 min_df=1	0.952642

7	v_n_splits=10
	v_train_size=0.8
	v_test_size=0.2
		n_estimators = 100
		n_jobs=-1
			hidden_layer_sizes=(100,100)
			random_state=1 #seed
			activation='logistic'
				max_features = 5000
				max_df=0.5
				min_df=2
					max_features = 5000
					ngram_range=(2,3)
					max_df=0.5
					min_df=1
						0.952908

8	v_n_splits=10
	v_train_size=0.8
	v_test_size=0.2
		n_estimators = 100
		n_jobs=-1
			hidden_layer_sizes=(100,100)
			random_state=1 #seed
			activation='logistic'
			solver='sgd'
			learning_rate='adaptive'
				max_features = 5000
				max_df=0.5
				min_df=2
					max_features = 5000
					ngram_range=(2,3)
					max_df=0.5
					min_df=1
						0.95911

9	v_n_splits=10
	v_train_size=0.8
	v_test_size=0.2
		n_estimators = 100
		n_jobs=-1
			hidden_layer_sizes=(100,100)
			random_state=1 #seed
			activation='logistic'
			solver='sgd'
			learning_rate='adaptive'
			momentum=0.9, alpha=1e-6, tol=1e-15
				max_features = 5000
				max_df=0.5
				min_df=2
					max_features = 5000
					ngram_range=(2,3)
					max_df=0.5
					min_df=1
						0.95911

Submission 1	n_splits=10
	train_size=0.8
	test_size=0.2
		n_estimators = 100
		n_jobs=-1
			hidden_layer_sizes=(100,100)
			random_state=1 #seed
			activation='logistic'
			solver='sgd'
			learning_rate='adaptive'
			momentum=0.9, alpha=1e-6
				max_features = 5000
				max_df=0.5
				min_df=2
					max_features = 5000
					ngram_range=(2,3)
					max_df=0.5
					min_df=1
						0.95992

Submission 2	*Same parameters as above* Pipeline is used				RF then NN	0.95958

(4)Results

(4.1)Model Evaluation and Validation

The submitted model has below parameters.

StratifiedShuffleSplit	RF	NN	tfidf_vectorizer	ngram_tfidf_vectorizer
n_splits=10	n_estimators = 100	hidden_layer_sizes=(100,100)	max_features = 5000	max_features = 5000
train_size=0.8	n_jobs=-1	random_state=1 #seed	max_df=0.5	ngram_range=(2,3)
test_size=0.2		activation='logistic'	min_df=2	max_df=0.5
		solver='sgd'		min_df=1
		learning_rate='adaptive'		
		momentum=0.9, alpha=1e-6		

These values have been put after multiple runs and observations, starting with default values for all.

(4.2)Justification

As mentioned in the proposal document and in this document, objective was to achieve AUC of min 0.80.

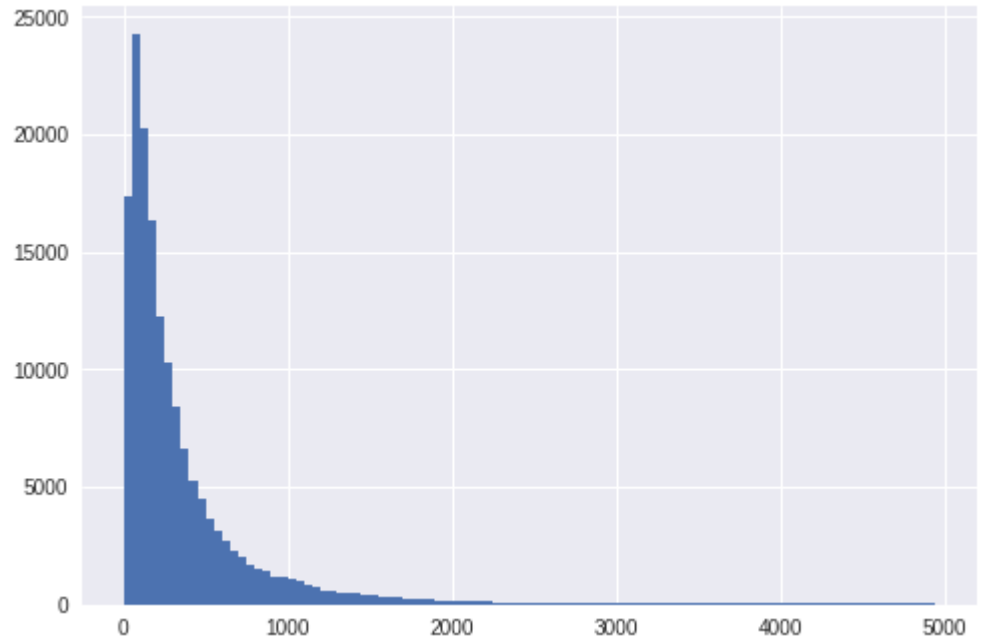
Benchmark model has AUC of 0.700092.

The AUC of submitted model is 0.95992. Submitted data has score of 0.9479 on Kaggle. It suggests that submitted model has solved the problem to acceptable standards.

(5)Conclusion

(5.1)Free-Form Visualization

One of the important characteristic for this data was length of comments. Below is a distribution of training data provided.



- Large volume of data has comment length less than 500 charecters. Some outliers are present with more than 4000 charesters also.
- Mean length of comments is 394 charecters.

```
[46] comment_length.mean()

     394.0732213246768
```

- There are no records without any comment.

```
'''
missing comment in comment text column.
'''

print('Number of missing comments in training data:')
train_data_full_df['comment_text'].isnull().sum()

Number of missing comments in training data:
0
```

When the data was divided into training and evaluation sets – there was replication of original training data in both the sets.

(5.2)Reflection

Basic idea behind having 2 versions of embeddings, 2 models is to reduce dependency on one approach and have less model bias. There can be multiple things that can be added on top of it – e.g. more models. I am yet to explore Pipelining of models.

Very interesting thing and value addition out of this entire Nanodegree for me is – Udacity Reviews. As a developer – we are many times confined in our own space and having reviews, feedbacks breaks that mirror and pushes to go extra miles. Its great to have unbiased and straight review comments. Programming world is very humbling. I wish to attain such level to programming, functional and domain expertise one day.

(5.3)Improvement

There are multiple areas of improvements in submitted solution:

(A) Code: Unix commands. Can be improved to make it only python dependent code.

(B) Model :

(B.1) Use of Pre trained vectors should be analysed and tested instead of data specific embeddings.

(B.2) Use of log loss and F1 for performance metrics in addition to ROC AUC.

(B.3) Parameter tuning of input parameters (embedding feature size, estimator in random forest, etc).

(B.4) Use of Grid Search and Random Search for hyperparameter tunning.

(C) Data cleaning should also be done.

(6)Acknowledgements and References

I want to acknowledge the support and guidance from Udacity team – creators of the course, reviewers and mentor. Below are few reference URL's used while completing this project.

[1]https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge
[2] https://www.kdnuggets.com/2018/04/right-metric-evaluating-machine-learning-models-1.html
[3] https://www.scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html
[4] scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
[5] scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
[6] scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[7] https://colab.research.google.com
[8] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/discussion/48049#275730
[9] https://medium.com/@move37timm/using-kaggle-api-for-google-colaboratory-d18645f93648
[10] http://scikit-learn.org/stable/modules/neural_networks_supervised.html#classification
[11] https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc
[12] http://scikit-learn.org/stable/modules/multiclass.html