**TOXIC COMMENT CLASSIFICATION**

**ABSTRACT**

Today's generation spends most of their time in social networking applications or websites, people share their life experiences, express their feelings and their lifestyle by posting photos, stories, Live Streaming, profile pictures etc.

There is freedom for everyone to post the comments on the social network sites that may hurt people feelings and may disturb their life too which may cause huge loss.

The freedom for people to comment on any topics or words without any restriction has become an issue and this is extremely been affected many lives in cases as cyber bullying, harassment.

This project presents a framework for using deep learning to automatically identify and categorize toxic online comments using Convolution Neural Network & Gated Recurrent Unit.

1. **INTRODUCTION**

Many website that display user-submitted content must deal with toxic or abusive comments. These online comments contain explicit language which may hurt the readers and online texts with high toxicity can cause personal attacks, online harassment and bullying behaviors. To protect users from being exposed to offensive language on online forums or social media sites, companies have started flagging comments and blocking users who are found guilty of using unpleasant language.

Potential automated solutions will require not just binary classification (acceptable vs blocked) but fine-grained comment classifications to maintain civility without interfering with normal discourse and provide explanations to users when their posts are censored. Different websites may wish to have different policies for dealing with different kinds of toxic content.

In this project, the attempt is identification of toxic online comments in a dataset provided by a Kaggle challenge. The dataset consists of 159,571 comments from Wikipedia talk page edits which have been labelled by human raters for the presence of toxic behavior. The 6 types of toxicity are: *toxic*, *severe*\_*toxic*, *obscene*, *threat*, *insult* and *identity*\_*hate*.

This project will describe a baseline approach to this multi-label classification task using Convolution Neural Network (CNN) and Bi-directional Gated Recurrent Unit (bi-GRU).

Convolutional neural networks (CNN) has been widely used for Image recognition or Image Processing in computer vision, it is core most model used in computer vision. Recently convolutional neural networks (CNN) have been interestingly used in NLP and have gained quite good and accurate results. Convolutional Neural Networks (CNN) has been widely applied for classification problems in different fields like text classification.

Recurrent neural network (RNNs) has proved to be a powerful sequence model for Natural Language Processing. Gated recurrent unit (GRU) is one kind of RNNs which has achieved excellent performance in NLP. This structure of RNN only contains two gates. The update gate controls the information that flows into memory, while the reset gate controls the information that flows out of memory. Similarly to the LSTM unit, GRU has gating units that modulate the flow of information inside the unit, however, without having a separate memory cell. Bidirectional GRU’s are a type of bidirectional recurrent neural networks with only the input and forget gates. It allows for the use of information from both previous time steps and later time steps to make predictions about the current state.

1. **LITERATURE SURVEY**

The paper *Khurana, D., Koli, A., Khatter, K. et al. Natural language processing: state of the art, current trends and challenges. Multimed Tools Appl (2022)* [1] conclude with three objectives. The first objective gives insights of the various important terminologies of NLP and NLG, and can be useful for the readers interested to start their early career in NLP and work relevant to its applications. The second objective of this paper focuses on the history, applications, and recent developments in the field of NLP. The third objective is to discuss datasets, approaches and evaluation metrics used in NLP. The relevant work done in the existing literature with their findings and some of the important applications and projects in NLP are also discussed in the paper. This paper distinguish four phases by discussing different levels of NLP by presenting the history and evolution of NLP.

Then *Keiron O’Shea, Ryan Nash arXiv:1511.08458* [2] describes Convolution Neural Network(CNN) as one of the most impressive forms of Artificial Neural Network(ANN). This document provides a brief introduction to CNNs, discussing recently published papers and newly formed techniques in developing these brilliantly fantastic image recognition models. This paper outlined the basic concepts of Convolutional Neural Networks, explaining the layers required to build one and detailing how best to structure the network in most image analysis tasks. This paper concludes that Convolutional Neural Networks differ to other forms of Artificial Neural Network in that instead of focusing on the entirety of the problem domain, knowledge about the specific type of input is exploited. This in turn allows for a much simpler network architecture to be set up.

In *arXiv:1408.5882* [3], Kim used CNN's to address a series of sentence-level classification tasks.

Gated Recurrent Unit (GRU) was introduced by Cho, in 2014, *arXiv:1406.1078* [4]. The aim was to solve the vanishing gradient problem which comes with a standard recurrent network. The paper propose a novel neural network model called RNN Encoder–Decoder that consists of two recurrent neural networks (RNN). One RNN encodes a sequence of symbols into a fixed length vector representation, and the other decodes the representation into another sequence of symbols. The encoder and decoder of the proposed model are jointly trained to maximize the conditional probability of a target sequence given a source sequence. The performance of a statistical machine translation system is empirically found to improve by using the conditional probabilities of phrase pairs computed by the RNN Encoder–Decoder as an additional feature in the existing log-linear model.

In *Zhang, Liujie & Zhou, Yanquan & Duan, Xiuyu & Chen, Ruiqi. (2018). A Hierarchical multi-input and output Bi-GRU Model for Sentiment Analysis on Customer Reviews. IOP Conference Series: Materials Science and Engineering. 322. 062007. 10.1088/1757-899X/322/6/062007* [5] paper proposes a multi-input and output model which applies two independent bi-GRU layer to generate part of speech and sentence representation.

1. **PROBLEM STATEMENT**

Anyone who has been the target of abuse or harassment online will know that it doesn’t go away when you log off or switch off your phone. Discussing things you care about can be difficult. 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.

This project’s focus is to solve the Toxic comment classification challenge posted on Kaggle. 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 behaviors, 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).

In this project, the challenge is to build a multi-headed model that’s capable of detecting different types of toxicity like threats, obscenity, insults, and identity-based hate.

1. **OBJECTIVES**

Given a group of sentences or paragraphs, used as a comment by a user in an online platform, classify it to belong to one or more of the following categories — toxic, severe-toxic, obscene, threat, insult or identity-hate.

As the task was to figure out whether the data belongs to zero, one, or more than one categories out of the six listed above, the first step before working on the problem was to distinguish between multi-label and multi-class classification.

In multi-class classification, one basic assumption is that the data can belong to only one label out of all the labels. For example, a given picture of a fruit may be an apple, orange or guava only and not a combination of these.

In multi-label classification, data can belong to more than one label simultaneously. For example, in this dataset a comment may be toxic, obscene and insulting at the same time. It may also happen that the comment is non-toxic and hence does not belong to any of the six labels.

So the objective of the project is to build a multi-label classification of toxic comments.

1. **METHODOLOGY**
   1. **Pre-Processing Text**

Pre-processing of the text is the first step that is performed on the dataset. The dataset is cleaned and prepared for the classification tasks by removing punctuation, imputing missing values, etc. Besides these common preprocessing functions there are other techniques that are used specifically for deep learning classification.

**5.1.1 Tokenization**

Tokenization is the process of converting a text corpus to a set of distinct tokens of any size. These tokens are usually numbers which are assigned to the words present in the text. As a computer cannot understand a language, this method helps us to map all the words to distinct numbers which makes it easier for the computer to understand. So the result of this process is a dictionary of fixed size that contains a mapping from words to numbers.

**5.1.2 Embedding**

Every word in the dataset is embedded into feature vectors, this is done by creating an embedding matrix. An embedding matrix is a list of words and their corresponding embeddings. Embeddings usually refer to n-dimensional dense vectors. The embedding matrix is of shape (max\_features, embedding\_dim). Here max\_features is the number of words in the dictionary that are obtained from the tokenization method and embedding\_dim is the number of features into which the words will be embedded. There are a lot of pre-trained word embeddings available with different embedding sizes like the GloVe (Global Vectors for Word Representation), word2vec, Fasttext-crawl, etc.

This project uses GloVe. GloVe word embedding is downloaded and a dictionary is created with those embeddings. In the next step word embedding matrix for each word is created and if the word doesn’t have an embedding in GloVe it will be presented with a zero matrix.

* 1. **Convolution Neural Network**

CNN were initially developed in the neural network image processing community where they achieved break-through results in recognizing an object from a pre-defined category (e.g., cat, bicycle, etc.).

A Convolutional Neural Network typically involves two operations, which can be thought of as feature extractors: convolution and pooling. The output of this sequence of operations is then typically connected to a fully connected layer which is in principle the same as the traditional multi-layer perceptron neural network (MLP).

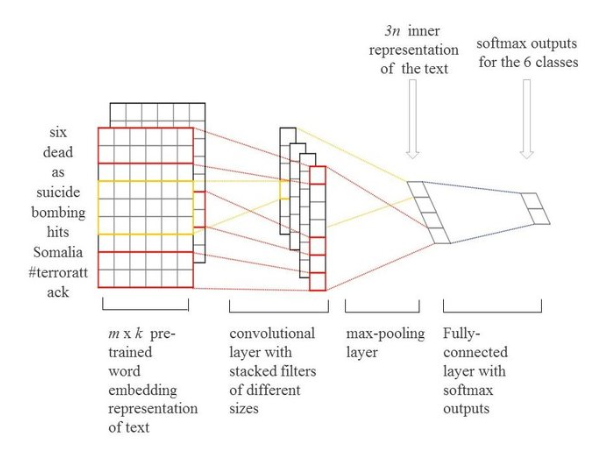
In the case of NLP tasks, i.e., when applied to text instead of images, we have a 1 dimensional array representing the text. Here the architecture of the Convolution Networks is changed to 1D convolutional-and-pooling operations.

Here, GloVe which is an open-sourced pre-defined word embedding available from the library is used. When we do dot product of vectors representing text, they might turn out zero even when they belong to same class but if you do dot product of those embedded word vectors to find similarity between them then you will be able to find the interrelation of words for a specific class.

Then, we slide the filter/ kernel over these embeddings to find convolutions and these are further dimensionally reduced in order to reduce complexity and computation by the Global Max Pooling layer.

Lastly, we have the fully connected layers and the activation function on the outputs that will give values for each class.

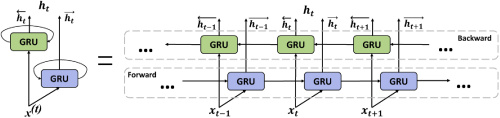
A simple CNN architecture for classifying texts is shown below in Figure 1.



*Figure 1: CNN architecture for text classification*

* 1. **Bidirectional Gated Recurrent Unit**

Models with bi-directional structure have the ability to learn information from previous and subsequent data when dealing with the current data. The bi-GRU model is determined based on the state of two GRUs, which are unidirectional in opposite directions. One GRU that moves forward, beginning from the start of the data sequence, the other GRU that moves backward, beginning from the end of the data sequence. This allows the information from both future and past to impact the current states. The bi-directional Gated Recurrent Unit (bi-GRU) can capture long range dependencies with lesser model parameters. The structure of the bi-GRU model diagram in Figure 2.



*Figure 2: Structure of bi-GRU*

1. **ALGORITHMS**
   1. **Convolutional Neural Network**

Convolutional Neural Network (CNN) is a deep neural network that is usually applied to images. CNNs were inspired by the human brain. Like the human brain, CNN consists of interconnected neurons in different layers. Each neuron in a layer is a perceptron that performs some computation to the weights that are passed to it. Although CNNs are mostly used for image classification, they can also be used for text classification by passing the feature vectors of input text to CNN. The CNN then computes weights for different neurons which are used to determine a function that maps the feature vectors to the output. A CNN usually consists of the following layers: -

1. Convolutional Layer: - The purpose of a Convolutional layer is to extract and learn features from the input vectors. A convolutional layer computes outputs of neurons by performing dot product operations to the weights and passes this output to an activation function.
2. Activation Function: - The output of a convolutional layer is passed to an activation function. An activation function is used to add non-linearity to the output of the Convolutional Layer. The most common activation function is the Rectified Linear Unit (ReLu) function. A ReLu function can be defined as follows:

f(x) = max (0, x)

1. Pooling Layer: - A pooling layer is used to reduce the dimensions of the input by preserving the important features. A Convolutional layer is often succeeded by a pooling layer to reduce the size and number of parameters from the previous layer.
2. Embedding Layer: - It is a special component of the CNNs for text classification problems. The purpose of an embedding layer is to transform the text inputs into a suitable form for the CNN. Here, each word of a text document is transformed into a dense vector of fixed size.
3. Fully-Connected Layer: - It is a classic Feed-Forward Neural Network (FNN) hidden layer. It can be interpreted as a special case of the convolutional layer with kernel size 3 X 3. This type of layer belongs to the class of trainable layer weights and it is used in the final stages of CNNs.
   1. **Bidirectional Gated Recurrent Unit**

GRUs are a gating mechanism in recurrent neural networks, improved version of standard recurrent neural network. The GRU is like a long short-term memory (LSTM) with a forget gate but has fewer parameters than LSTM, as it lacks an output gate. To solve the vanishing gradient problem of a standard RNN, GRU uses

1. Update Gate: - It is a combination of Forget Gate and Input Gate. Forget gate decides what information to ignore and what information to add in memory.
2. Reset Gate: - This Gate resets the past information in order to get rid of gradient explosion. Reset Gate determines how much past information should be forgotten.

Basically, these are two vectors which decide what information should be passed to the output. The special thing about them is that they can be trained to keep information from long ago, without washing it through time or remove information which is irrelevant to the prediction.

A Bidirectional GRU, or bi-GRU, is a sequence processing model that consists of two GRUs. One taking the input in a forward direction, and the other in a backwards direction. It is a bidirectional recurrent neural network with only the input and forget gates.

1. **IMPLEMENTATION**

Keras framework was used to implement the CNN and GRU models. Keras is an open-sourced neural network library built on python which provides user-friendly, high-level APIs which enable easy implementations of different deep neural network algorithms.

The dataset used for classification in this paper is given as part of the competition hosted by Jigsaw and it contains 159571 comments taken from Wikipedia. Each comment is classified into one of 6 labels based on their level of toxicity. The results are then validated against a test data set of 153164 new examples.

* 1. **Convolution Neural Network**

The first step of this algorithm is tokenization. This generates a sequence of numbers for each comment. As each comment may vary is their lengths, the output of tokenization is padded to a fixed length of 400. This is then passed to a Keras Embedding Layer which learns embeddings. The embedding size used was 300. The embedding layer is provided with a matrix. After the embedding layer, we then use a series of convolutional layers in conjunction with pooling layers and dropout layers. Dense layer of 6 units with a sigmoid activation function is used which predicts the probabilities of each label.

This CNN model achieved an accuracy of 96.94%.

* 1. **Bidirectional Gated Recurrent Unit**

Similar to CNN the first layer will be the embedding layer where sentences will be represented as max\_features by embedding\_dim vectors. The embedding layer turns positive integers (indexes) into dense vectors of fixed size. This embedding layer will encode the input sequence into a sequence of dense embedding\_dim vectors. A bidirectional wrapper of RNN layer, i.e., GRU along with 1D convolution layer and global averaging pooling & global max pooling are used. Finally, a Dense layer, which implements the operation, of 6 units with a sigmoid activation function predicts the probabilities of each label.

This bi-GRU model achieved an accuracy of 93.76%.

* 1. **Learning Curve**

In machine learning, a learning curve (or training curve) plots the optimal value of a model's loss function for a training set against this loss function evaluated on a validation data set with same parameters as produced the optimal function. It is a tool to find out how much a machine model benefits from adding more training data and whether the estimator suffers more from a variance error or a bias error. If both the validation score and the training score converge to a value that is too low with increasing size of the training set, it will not benefit much from more training data.

The machine learning curve is useful for many purposes including comparing different algorithms, choosing model parameters during design, adjusting optimization to improve convergence, and determining the amount of data used for training.

In the machine learning domain, there are two implications of learning curves differing in the x-axis of the curves, with experience of the model graphed either as the number of training examples used for learning or the number of iterations used in training the model.

In this project learning curve is plotted for both the models.

* 1. **ROC Curve**

The Receiver Operator Characteristic (ROC) curve is an evaluation metric for binary classification problems. It is a probability curve that plots the True Positive Rate (TPR) against False Positive Rate (FPR) at various threshold values and essentially separates the 'signal' from the 'noise'.

In this project ROC curve is used as an evaluation metric and the curve is plotted for both the models

1. **EMPLOY CODE**

# Import the required libraries

import re

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from wordcloud import WordCloud, STOPWORDS

import sklearn

from sklearn.model\_selection import train\_test\_split

import tensorflow as tf

from tensorflow.keras.preprocessing import text, sequence

from tensorflow.keras.models import Sequential

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Embedding

from tensorflow.keras.layers import Dense, Dropout, Activation, RNN, LSTM, GRU, Input, concatenate

from tensorflow.keras.layers import Conv1D, GlobalMaxPooling1D, MaxPooling1D, GlobalAveragePooling1D, Bidirectional

from keras.callbacks import ReduceLROnPlateau

from keras import initializers, regularizers, constraints, optimizers, layers

from sklearn.metrics import roc\_curve, auc

from sklearn.metrics import accuracy\_score, confusion\_matrix,classification\_report

# Mount google drive

from google.colab import drive

drive.mount('/content/drive')

# Load the training dataset

train\_df = pd.read\_csv("/content/drive/MyDrive/Project JNTUH AIML/train.csv")

train\_df.head(10)

# Columns, total entries & data type in training data

train\_df.info()

# Check the distribution of train data

train\_df.describe()

# Check for unique values

train\_df.nunique()

# Load testing dataset

test\_df = pd.read\_csv("/content/drive/MyDrive/Project JNTUH AIML/test.csv")

test\_df.head(10)

# Check the distribution of test data

test\_df.describe()

# Load test labels

test\_labels\_df = pd.read\_csv("/content/drive/MyDrive/Project JNTUH AIML/test\_labels.csv")

test\_labels\_df.head()

# Check the distribution of test labels

test\_labels\_df.describe()

# Check the shape of data

print("Training data shape: ",train\_df.shape)

print("Testing data shape: ",test\_df.shape)

print("Testing label data shape: ",test\_labels\_df.shape)

# Drop id column, as not required for analysis

train\_df = train\_df.drop(['id'],axis = 1)

test\_df = test\_df.drop(['id'],axis = 1)

test\_labels\_df = test\_labels\_df.drop(['id'],axis = 1)

# Check for null values

print("Null values in training set: ")

train\_df.isnull().sum()

print("Null values in testing set: ")

test\_df.isnull().sum()

print("Null values in test label set: ")

test\_labels\_df.isnull().sum()

# Function to obtain a cleaner text

def clean\_text(line):

clean = ""

line = line.replace(",", " ")

line = line.replace(".", " ")

line = line.replace("'", " ")

line = line.replace(":", " ")

line = line.replace(";", " ")

line = line.replace("=", " ")

line = line.replace("-", " ")

line = line.replace("+", " ")

line = line.replace("/", " ")

line = line.replace("|", " ")

line = line.replace("!", " ")

line = line.replace("@", " ")

line = line.replace("#", " ")

line = line.replace("$", " ")

line = line.replace("%", " ")

line = line.replace("^", " ")

line = line.replace("\*", " ")

line = line.replace("?", " ")

line = line.replace("\t", " ")

line = line.replace("\n", " ")

line = line.lower()

for char in line:

if char in "qwertyuiopasdfghjklzxcvbnm ":

clean += char

else:

clean += " "

clean = re.sub(" +"," ",clean)

return clean

# Clean text

train\_df['comment\_text'] = train\_df['comment\_text'].apply(lambda x: clean\_text(x))

test\_df['comment\_text'] = test\_df['comment\_text'].apply(lambda x: clean\_text(x))

# Store training features in x

x = train\_df["comment\_text"].values

print(x)

# Store training labels in y

y = train\_df[train\_df.columns[1:]].values

print(y)

x\_test = test\_df['comment\_text'].values

print(x\_test)

y\_test = test\_labels\_df.values

print(y\_test)

print("Training data shape: ",x.shape)

print("Training data labels shape: ",y.shape)

print("Testing data shape: ",x\_test.shape)

print("Testing data labels shape: ",y\_test.shape)

# Plot the most frequent words in the dataset using word cloud

def word\_plot(column,text):

comments = train\_df['comment\_text'].loc[column == 1].values

word\_cloud = WordCloud( width = 640, height = 640, background\_color = 'black',

stopwords = STOPWORDS).generate(str(comments))

fig = plt.figure( figsize = (8, 5), facecolor = 'k', edgecolor = 'k')

plt.subplot()

plt.imshow(word\_cloud, interpolation = 'bilinear')

plt.suptitle("Most frequent words in " + text , y = 1.06,color = "white")

plt.tight\_layout(pad = 0)

plt.axis('off')

plt.show()

word\_plot(train\_df['toxic'], "Toxic comments")

word\_plot(train\_df['severe\_toxic'],"severe Toxic comments")

word\_plot(train\_df['obscene'],"Obscene comments")

word\_plot(train\_df['threat'],"Threat comments")

word\_plot(train\_df['insult'],"Insult comments")

word\_plot(train\_df['identity\_hate'],"Identity hate comments")

# Exlore distribution of training labels

colors\_ = ["deep blue","blue", "teal", "lightblue","sea blue","purple"]

palette= sns.xkcd\_palette(colors\_)

n = train\_df.iloc[:,1:].sum()

plt.figure(figsize=(10,7))

ax= sns.barplot(n.index, n.values,palette=palette)

plt.title("Class Distribution in Training set")

plt.xlabel('Label ')

plt.ylabel('No. of comments', fontsize=11)

patch = ax.patches

labels = n.values

for rect, label in zip(patch, labels):

height = rect.get\_height()

ax.text(rect.get\_x() + rect.get\_width()/2, height + 10, label, va='bottom',ha='center')

plt.show()

# Explore multi label comments

rowsums = train\_df.iloc[:,1:].sum(axis=1)

m=rowsums.value\_counts()

plt.figure(figsize=(8,5))

ax = sns.barplot(m.index, m.values)

plt.title("Multiple categories per comment")

plt.ylabel('No. of Occurrences', fontsize=12)

plt.xlabel('No. of categories', fontsize=12)

# Heatmap for training data

fig = plt.figure(figsize = (9,7))

sns.heatmap(train\_df.corr(), annot=True,cmap="Blues")

plt.suptitle('Heatmap of Training label Class Correlation',size = 14)

plt.xlabel("Classes")

plt.ylabel("Classes")

plt.show()

correlation\_val = train\_df.corr()

correlation\_val

abs(correlation\_val) >= 0.6

# Distribution of Classes in Percentage

print("Distribution of Training Classes in Percentage:")

print((train\_df['toxic'].value\_counts()/159571)\*100)

print(train\_df['severe\_toxic'].value\_counts()/159571 \*100)

print(train\_df['obscene'].value\_counts()/159571 \*100)

print(train\_df['threat'].value\_counts()/159571 \*100)

print(train\_df['insult'].value\_counts()/159571 \*100)

print(train\_df['identity\_hate'].value\_counts()/159571 \*100)

print("Distribution of Test Classes in Percentage:")

print((test\_labels\_df['toxic'].value\_counts())/153164 \*100)

print(test\_labels\_df['severe\_toxic'].value\_counts() /153164\*100)

print(test\_labels\_df['obscene'].value\_counts() /153164 \*100)

print(test\_labels\_df['threat'].value\_counts() /153164 \*100)

print(test\_labels\_df['insult'].value\_counts() /153164 \*100)

print(test\_labels\_df['identity\_hate'].value\_counts() /153164 \*100)

print('Percentage of comments that are not labelled:')

print(len(train\_df[(train\_df['toxic']==0) & (train\_df['severe\_toxic']==0) & (train\_df['obscene']==0) & (train\_df['threat']== 0) & (train\_df['insult']==0) & (train\_df['identity\_hate']==0)]) / len(train\_df))

# Check where training labels have a row with only zeros

zero\_rows\_y = np.where(~y.any(axis=1))[0]

zero\_rows\_y

y[159568]

# Check shape of comments that are not labelled

print("Rows with zeros in training labels: ",zero\_rows\_y.shape)

print("Shape of training labels : ", y.shape)

# Perform Under Sampling by dropping a few rows of data

# Drop 35% of comments which have labels as all zeros

# Calculating 35% of labels with rows with only zeros

(143346\*35)/100

# Get index where rows of training labels are zeros

drop\_rows\_indx = zero\_rows\_y[0:50000] # store indexes of first 50 such rows

drop\_rows\_indx

drop\_rows\_indx.shape

# Dropping the first 50000 rows of training data

for indx in drop\_rows\_indx:

train\_df = train\_df.drop([indx], axis=0)

train\_df.shape

train\_df

print("Distribution of Training Classes in Percentage:")

print()

print((train\_df['toxic'].value\_counts()/109571)\*100)

print()

print(train\_df['severe\_toxic'].value\_counts()/109571 \*100)

print()

print(train\_df['obscene'].value\_counts()/109571 \*100)

print()

print(train\_df['threat'].value\_counts()/109571 \*100)

print()

print(train\_df['insult'].value\_counts()/109571 \*100)

print()

print(train\_df['identity\_hate'].value\_counts()/109571 \*100)

print()

# Check null values

null\_train = train\_df.isnull().sum()

print("Null values in training : ")

print(null\_train)

print()

# Store training features in x

x = train\_df["comment\_text"].values

print(x)

# Store training labels in y

list\_class = ["toxic", "severe\_toxic", "obscene", "threat", "insult", "identity\_hate"]

y = train\_df[list\_class].values

print(y)

print("Training data shape after undersampling: ", x.shape)

print("Training data labels shape after undersampling: ", y.shape)

max\_features = 20000 # most freq 20k words in the dataset would be kept

max\_text\_length = 400

# Change each text into a sequence of integers, each integer being index of token in a dictionary

x\_tokenizer = text.Tokenizer(max\_features)

x\_tokenizer.fit\_on\_texts(list(x))

x\_tokenized = x\_tokenizer.texts\_to\_sequences(x)

x\_train\_val = sequence.pad\_sequences(x\_tokenized,maxlen = max\_text\_length) #padding so that all comments have same lenght of 400

# Tokenise and pad the text in test dataset so that all comments are sequences have the same lenght of 400

x\_test\_tokenized = x\_tokenizer.texts\_to\_sequences(x\_test)

x\_testing = sequence.pad\_sequences(x\_test\_tokenized,maxlen = max\_text\_length)

# Prepare embedding matrix using GloVe embeddings

# Code to download glove data

!wget http://nlp.stanford.edu/data/glove.6B.zip

!unzip glove.6B.zip

embedding\_dim = 100

embeddings\_index = dict()

f = open('glove.6B.100d.txt')

for line in f:

values = line.split()

word = values[0]

coefs = np.asarray(values[1:],dtype = 'float32')

embeddings\_index[word] = coefs

f.close()

print(f'Found {len(embeddings\_index)} word vectors')

# Prepare embedding\_matrix

embedding\_matrix = np.zeros((max\_features,embedding\_dim))

for word,index in x\_tokenizer.word\_index.items():

if index> max\_features -1:

break

else:

embedding\_vector = embeddings\_index.get(word)

if embedding\_vector is not None:

embedding\_matrix[index] = embedding\_vector

print(embedding\_matrix)

x\_train\_val

# Check shape of data

print(x\_train\_val.shape, y.shape)

# Train test split

x\_train,x\_val,y\_train,y\_val = train\_test\_split(x\_train\_val, y,

test\_size = 0.2,

shuffle = True,

random\_state = 1)

print("Training Data Shape: ")

print(x\_train.shape, y\_train.shape)

print("Validation Data Shape: ")

print(x\_val.shape, y\_val.shape)

filters\_cnn = 250

kernel\_size\_cnn = 3

hidden\_dims\_cnn = 250

batch\_size\_cnn = 32

n\_epochs\_cnn = 4

drop\_rate\_cnn = 0.4

# Build the model - CNN

model\_cnn = Sequential()

# create embedding layer

model\_cnn.add(Embedding(max\_features, embedding\_dim, trainable = False

, embeddings\_initializer = tf.keras.initializers.Constant(embedding\_matrix)))

# 1st dropout

model\_cnn.add(Dropout(drop\_rate\_cnn))

# 1st convolutional 1-D layer

model\_cnn.add(Conv1D(filters\_cnn, kernel\_size\_cnn, padding = 'valid', activation = 'relu'))

# max pooling layer

model\_cnn.add(MaxPooling1D())

# 2nd convolutional 1-D layer

model\_cnn.add(Conv1D(filters\_cnn, kernel\_size\_cnn, padding = 'valid', activation = 'relu'))

# global max pooling layer

model\_cnn.add(GlobalMaxPooling1D())

# 1st dense layer

model\_cnn.add(Dense(hidden\_dims\_cnn, activation = 'relu'))

# 2nd dropout

model\_cnn.add(Dropout(drop\_rate\_cnn))

# final dense layer

model\_cnn.add(Dense(6,activation = 'sigmoid'))

# Compile the model

model\_cnn.compile(loss = 'binary\_crossentropy',

optimizer = 'adam',

metrics = ['accuracy'])

# Fit the model

model\_cnn.fit(x\_train,y\_train,

batch\_size = batch\_size\_cnn,

epochs = n\_epochs\_cnn,

validation\_data=(x\_val,y\_val))

# Evaluate the model

model\_cnn.evaluate(x\_val, y\_val, verbose=0)

# The values less than 0 would be changed to 0

y\_test[y\_test<0] = 0

# Predict test values

y\_pred\_cnn = model\_cnn.predict(x\_testing, verbose = 1, batch\_size = 32)

y\_pred\_cnn

y\_pred\_cnn[0]

y\_test[0]

y\_pred\_cnn[21]

y\_test[21]

y\_pred\_cnn[48]

y\_test[48]

y\_pred\_cnn[10]

y\_test[10]

batch\_size\_gru = 32

n\_epoch\_gru = 3

kernel\_size\_gru = 3

drop\_rate\_gru = 0.1

# Build the model - GRU

inp = Input(shape=max\_text\_length,)

g = Embedding(max\_features, embedding\_dim, weights=[embedding\_matrix])(inp)

g = Bidirectional(GRU(128, return\_sequences=True, dropout=drop\_rate\_gru, recurrent\_dropout=0.1))(g)

g = Conv1D(64, kernel\_size = kernel\_size\_gru, padding = "valid", activation="relu")(g)

g = concatenate([GlobalAveragePooling1D()(g), GlobalMaxPooling1D()(g)])

g = Dense(6, activation="sigmoid")(g)

model\_gru = Model(inputs=inp, outputs=g)

model\_gru.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

model\_gru.summary()

reduce\_lr = ReduceLROnPlateau(monitor='val\_loss', factor=0.2, patience=1, min\_lr=1e-8)

model\_gru.fit(x\_train\_val, y, batch\_size=batch\_size\_gru, epochs=n\_epoch\_gru, validation\_split=0.1,

callbacks=[reduce\_lr])

y\_pred\_gru = model\_gru.predict(x\_testing, batch\_size=32)

y\_pred\_gru

y\_pred\_gru[21]

y\_test[21]

y\_pred\_gru[7549]

y\_test[7549]

# Plot accuracy and loss curves

def plot\_history(history):

plt.figure(figsize=(8,5),linewidth = 7, edgecolor="whitesmoke")

n = len(history.history['accuracy'])

plt.plot(np.arange(0,n)+1,history.history['accuracy'], color='orange',marker=".")

plt.plot(np.arange(0,n)+1,history.history['loss'],'b',marker=".")

# offset both validation curves

plt.plot(np.arange(0,n)+ 1,history.history['val\_accuracy'],'r')

plt.plot(np.arange(0,n)+ 1,history.history['val\_loss'],'g')

plt.legend(['Train Acc','Train Loss','Val Acc','Val Loss'])

plt.grid(True)

# set vertical limit to 1

plt.gca().set\_ylim(0,1)

plt.xlabel("Number of Epochs")

plt.ylabel("Value")

plt.suptitle("Learning Curve", size=16, y=0.927)

plt.show()

plot\_history(history\_cnn)

plot\_history(history\_gru)

# Plot roc curve

def plot\_roc(y, y\_pred, colors):

fpr, tpr, thresholds = roc\_curve(y, y\_pred)

plt.plot(fpr, tpr, color=colors, lw = 1.5, label='ROC Curve')

plt.plot([0, 1], [0, 1], color='lightgrey', lw = 1, linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

# ROC curve for CNN model

plot\_roc(y\_test[:, 0], y\_pred\_cnn[:, 0],'gold')

plot\_roc(y\_test[:, 1], y\_pred\_cnn[:, 1],"pink")

plot\_roc(y\_test[:, 2], y\_pred\_cnn[:, 2],"purple")

plot\_roc(y\_test[:, 3], y\_pred\_cnn[:, 3],"b")

plot\_roc(y\_test[:, 4], y\_pred\_cnn[:, 4],"g")

plot\_roc(y\_test[:, 5], y\_pred\_cnn[:, 5],"lightblue")

# ROC curve for biGRU model

plot\_roc(y\_test[:, 0], y\_pred\_gru[:, 0],'gold')

plot\_roc(y\_test[:, 1], y\_pred\_gru[:, 1],"pink")

plot\_roc(y\_test[:, 2], y\_pred\_gru[:, 2],"purple")

plot\_roc(y\_test[:, 3], y\_pred\_gru[:, 3],"b")

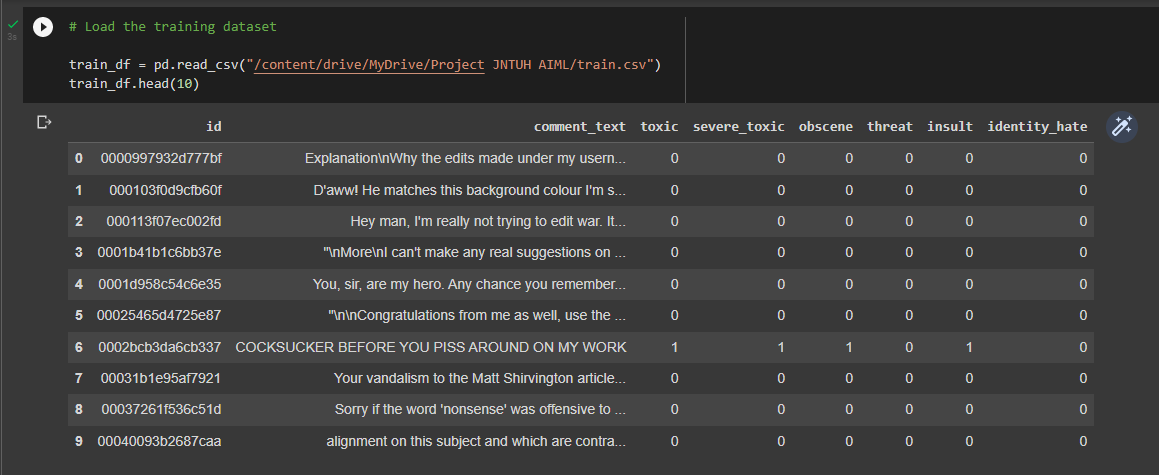
plot\_roc(y\_test[:, 4], y\_pred\_gru[:, 4],"g")

plot\_roc(y\_test[:, 5], y\_pred\_gru[:, 5],"lightblue")

1. **RESULTS**

Below are the screenshots of the output:

Training dataset contain 159571 entries with total 8 columns having no null values.



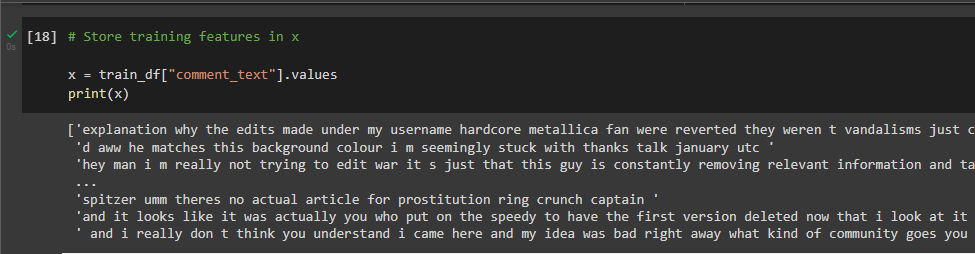
*Figure 3: Training dataset*

Testing dataset have 153164 entries.



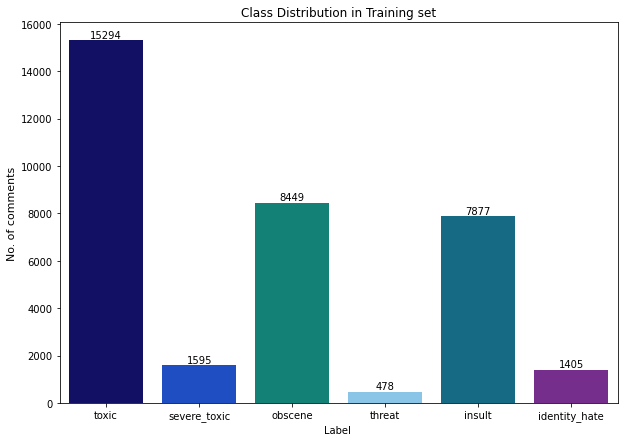
*Figure 4: Test Dataset*

Training data after preprocessing which involved removal of punctuation, special characters and converting the text into lower case.



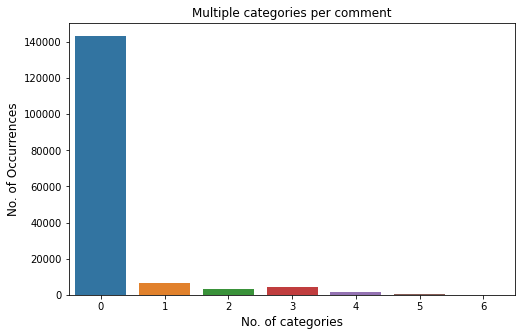
*Figure 5: Clean dataset*

Exploring the distribution of training labels.



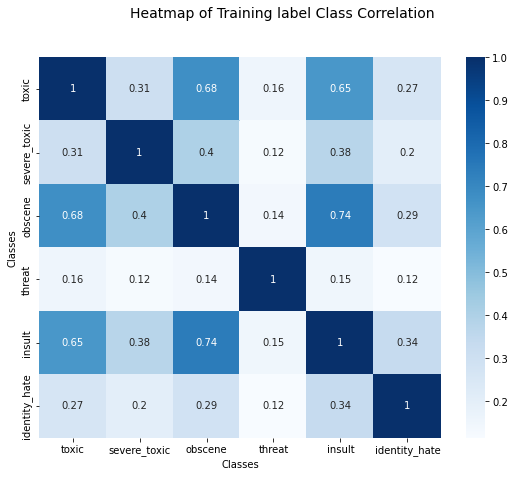
*Figure 6: Class distribution*

Below graph shows number of multi label comments.



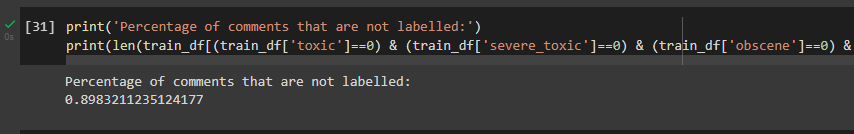
*Figure 7: Explore multi label comments*

The class correlation heatmap shows that some classes are highly positively correlated to others. For correlation greater than 60%, toxic and insult have 0.64 correlation, toxic and obscene have 0.67, whereas obscene and insult have 0.74 correlation. This means that is a comment is toxic then there is a 67% chance it is also obscene and 64% chance that it is also classified as insult. Obscene and insult comments are the most correlated with correlation of 74%.

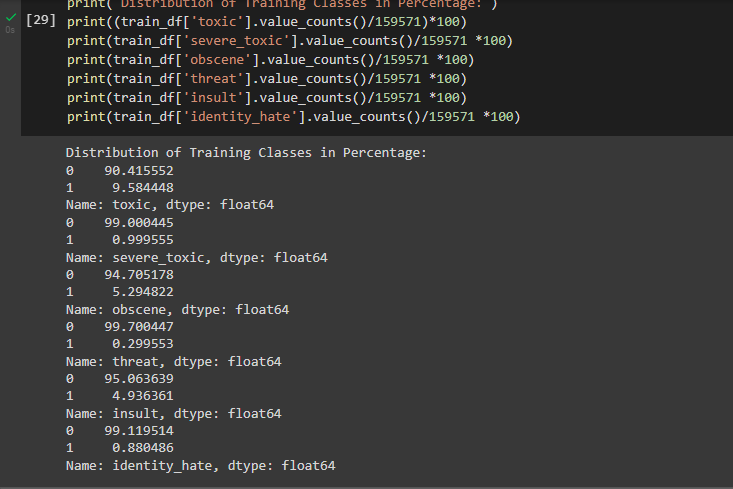


*Figure 8: Heatmap for training data*

It can be observed that most of the values are 0, very few percentage of values in classes as labelled as 1 which means there is a high imbalance of classes. Almost 90% of data is not labelled or in other words are not toxic. There is a high chance of overfitting and getting a good accuracy with a model which predicts almost all comments as not toxic. It is important to fit the data such that it also predicts the toxicity and the type of toxicity accurately. Some undersampling should be performed to remove the imbalance along with stratified sampling by splitting data into training and validation.

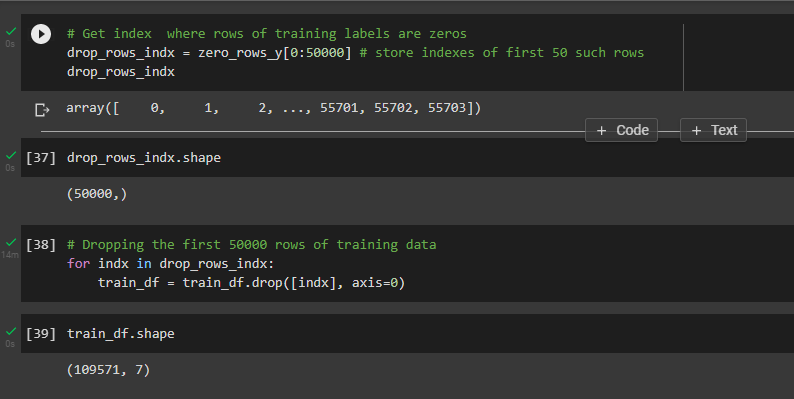


*Figure 9: Percentage of non-labeled comments*



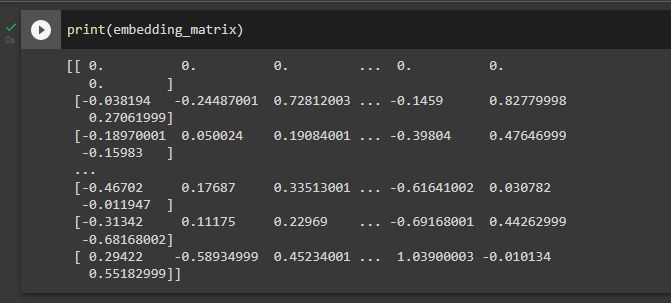
*Figure 10: Training classes distribution in percentage*

Here undersampling is performed by dropping 35% of comments or rows of data which have labels as all zeros.



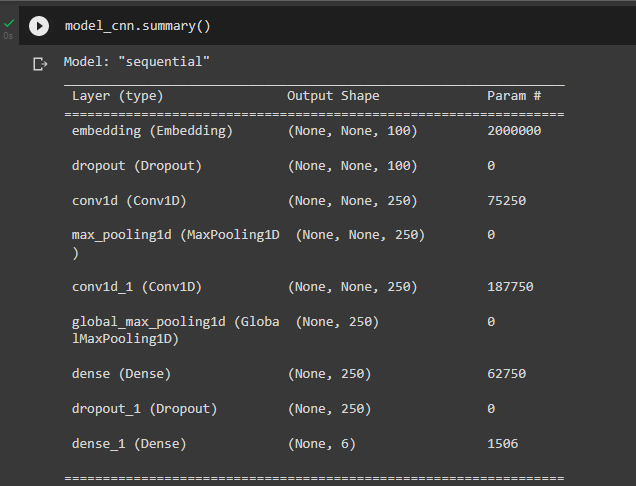
*Figure 11: Undersampling*

Embedding matrix is prepared using GloVe embeddings.



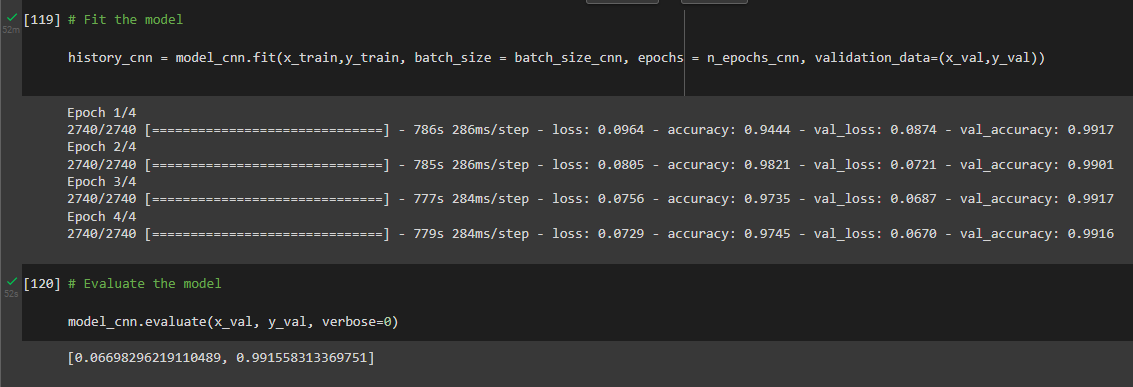
*Figure 12: Embedding matrix*

Convolution neural network model summary mentioning all the layers used.



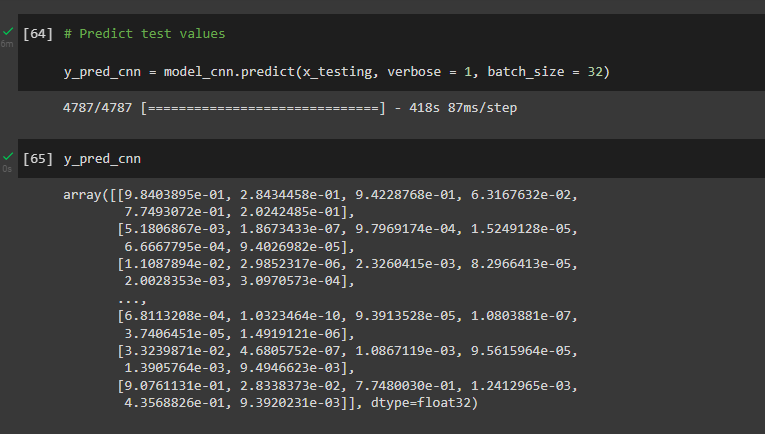
*Figure 13: CNN model summary*

Convolution Neural Network model training.



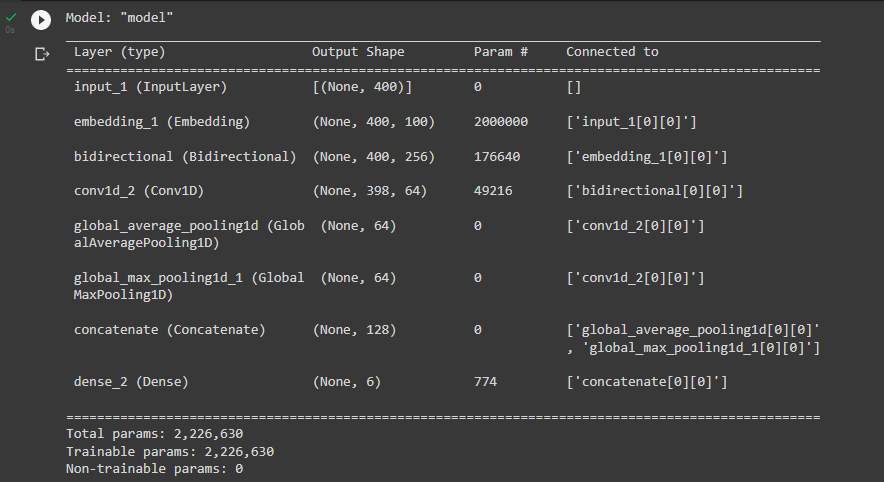
*Figure 14: CNN model training*

CNN predicted values.



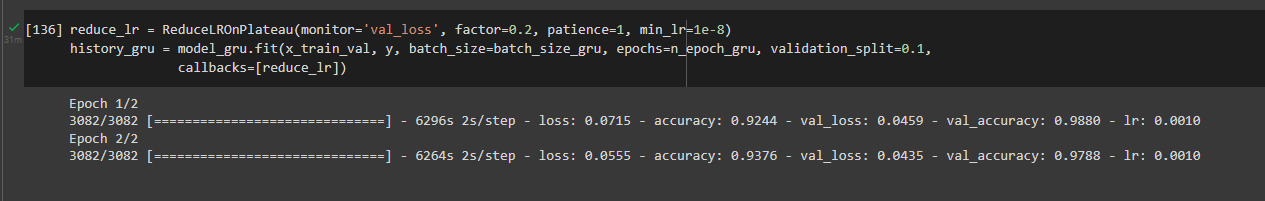
*Figure 15: CNN prediction*

Bidirectional Gated Recurrent Unit model training and summary.



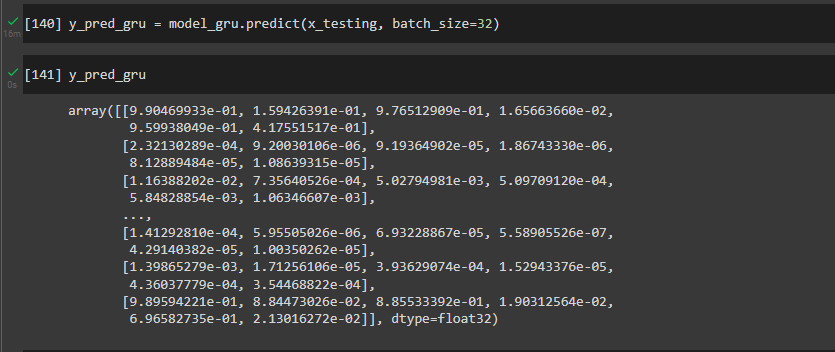
*Figure 16: Bi-GRU model summary*

Next step is to train bi-GRU model.



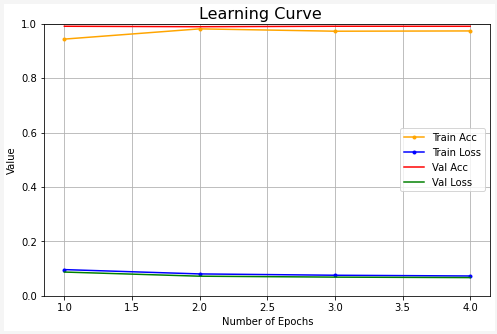
*Figure 17: Bi-GRU model training*

Bi-GRU predictions:

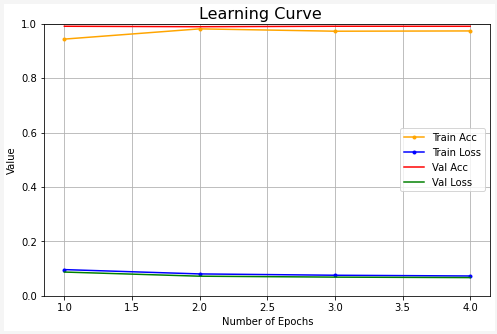


*Figure 18: Bi-GRU prediction*

Accuracy and lost curves are plotted:

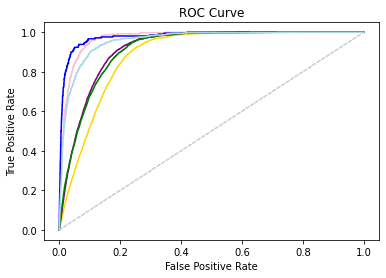


*Figure 19: Learning curve for CNN*

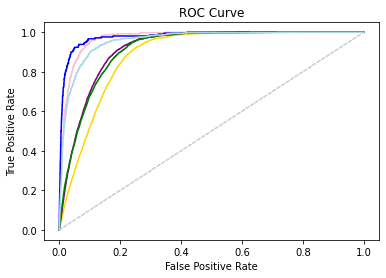


*Figure 20: Learning curve for bi-GRU*

ROC curve is used as evaluation matrix:



*Figure 21: ROC curve for CNN*



*Figure 22: ROC curve for bi-GRU*

1. **CONCLUSION & FUTURE SCOPE**

This project mainly addresses the toxic comments made online using NLP, CNN and bi-GRU. Both the models developed are a multi-labeled classification scheme that can predict the different types of toxicity and levels of toxicity in a comment. CNN achieved an accuracy of 97.45%. On the other, hand the bi-GRU achieved an accuracy of 93.76%.

The advantage of CNN is that it takes into account local structure since a convolution is usually taken over adjacent word embeddings, hence information contained in adjacent words is learned effectively. In images, local structure matters as useful semantic information is contained in adjacent pixels. However, in sentences, words often don’t need to be adjacent to be related. The promising results are motivating for further development of CNN based methodologies for text mining in near future.

The combination of a GRU flowing signals in a specific direction with another GRU carrying information in the opposite direction forms the bidirectional gated recurrent unit (BiGRU). This process that generated the BiGRU network can provide more efficient short-term electric load forecasts than the original GRU. But because the network is more complicated, the model time cost is increased to a certain extent. In the future, it is the goal of the next step to study how to improve the classification accuracy and reduce the time cost.

**References**

[1] Khurana, D., Koli, A., Khatter, K. et al. Natural language processing: state of the art, current trends and challenges. Multimed Tools Appl (2022). <https://doi.org/10.1007/s11042-022-13428-4>

[2] An Introduction to Convolutional Neural Networks Keiron O'Shea, Ryan Nash arXiv:1511.08458 <https://doi.org/10.48550/arXiv.1511.08458>

[3] Convolutional Neural Networks for Sentence Classification Yoon Kim arXiv:1408.5882 <https://doi.org/10.48550/arXiv.1408.5882>

[4] Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio arXiv:1406.1078 <https://doi.org/10.48550/arXiv.1406.1078>

[5] Zhang, Liujie & Zhou, Yanquan & Duan, Xiuyu & Chen, Ruiqi. (2018). A Hierarchical multi-input and output Bi-GRU Model for Sentiment Analysis on Customer Reviews. IOP Conference Series: Materials Science and Engineering. 322. 062007. 10.1088/1757-899X/322/6/062007 <https://arxiv.org/abs/1406.1078>

* Annexure – I (List of figures)
  + Figure 1: CNN architecture for text classification 7
  + Figure 2: Structure of bi-GRU 7
  + Figure 3: Training dataset 23
  + Figure 4: Test Dataset 23
  + Figure 5: Clean dataset 24
  + Figure 6: Class distribution 24
  + Figure 7: Explore multi label comments 25
  + Figure 8: Heatmap for training data 25
  + Figure 9: Percentage of non-labeled comments 26
  + Figure 10: Training classes distribution in percentage 26
  + Figure 11: Undersampling 27
  + Figure 12: Embedding matrix 27
  + Figure 13: CNN model summary 28
  + Figure 14: CNN model training 28
  + Figure 15: CNN prediction 29
  + Figure 16: Bi-GRU model summary 29
  + Figure 17: Bi-GRU model training 30
  + Figure 18: Bi-GRU prediction 30
  + Figure 19: Learning curve for CNN 30
  + Figure 20: Learning curve for bi-GRU 31
  + Figure 21: ROC curve for CNN 31
  + Figure 22: ROC curve for bi-GRU 32
* Annexure –II (List of Symbols/Acronyms)
  + 1D 1 Dimensional
  + ANN Artificial Neural Network
  + API Application Programming Interface
  + Bi-GRU Bidirectional Gated Recurrent Unit
  + CNN Convolution Neural Network
  + GloVe Global Vectors for Word Representation
  + GRU Gated Recurrent Unit
  + MLP Multi-Layer Perceptron
  + NLP Natural Language Processing
  + ReLu Rectified Linear Unit
  + RNN Recurrent Neural Network
  + ROC Receiver Operator Characteristic