Toxic Comments Classification using Deep Learning and Natural Language Processing

Project Team

Sl. No.	Reg. No.	Student Name
01	16ETCS002064	Lakshmikanth H V
02	16ETCS002069	Manoj M
03	16ETCS002074	Modali Krishna Sai
04	16ETCS002078	Muli Madhava Reddy

Supervisor: Mrs. Vaishali R Kulkarni

May – 2020

B. Tech. in Computer Science and Engineering

FACULTY OF ENGINEERING AND TECHNOLOGY

M. S. RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Bengaluru -560 054

FACULTY OF ENGINEERING AND TECHNOLOGY

FET

Certificate

*This is to certify that the Project titled "Toxic Comments Classification using Deep Learning and Natural Language Processing" is a bonafide work carried out in the Department of Computer Science and Engineering by **Mr. Lakshmikanth H V bearing Reg. No. 16ETCS002064** in partial fulfilment of requirements for the award of B. Tech. Degree in Computer Science and Engineering of Ramaiah University of Applied Sciences.*

May – 2020

Mrs. Vaishali R Kulkarni
Supervisor

Dr. P. V. R. Murthy
Professor and Head – Dept. of CSE

Dr. H. M. Rajashekara Swamy
Professor and Dean-FET

FACULTY OF ENGINEERING AND TECHNOLOGY

Certificate

*This is to certify that the Project titled "Toxic Comments Classification using Deep Learning and Natural Language Processing" is a bonafide work carried out in the Department of Computer Science and Engineering by **Mr. Manoj M bearing Reg. No. 16ETCS002069** in partial fulfilment of requirements for the award of B. Tech. Degree in Computer Science and Engineering of Ramaiah University of Applied Sciences.*

May – 2020

Mrs. Vaishali R Kulkarni
Supervisor

Dr. P. V. R. Murthy
Professor and Head – Dept. of CSE

Dr. H. M. Rajashekara Swamy
Professor and Dean-FET

FACULTY OF ENGINEERING AND TECHNOLOGY

FET

Certificate

This is to certify that the Project titled "Toxic Comments Classification using Deep Learning and Natural Language Processing" is a bonafide work carried out in the Department of Computer Science and Engineering by ***Mr. Modali Krishna Sai bearing Reg. No. 16ETCS002074*** *in partial fulfilment of requirements for the award of B. Tech. Degree in Computer Science and Engineering of Ramaiah University of Applied Sciences.*

May – 2020

Mrs. Vaishali R Kulkarni
Supervisor

Dr. P. V. R. Murthy
Professor and Head – Dept. of CSE

Dr. H. M. Rajashekara Swamy
Professor and Dean-FET

FACULTY OF ENGINEERING AND TECHNOLOGY

FET

Certificate

This is to certify that the Project titled "Toxic Comments Classification using Deep Learning and Natural Language Processing" is a bonafide work carried out in the Department of Computer Science and Engineering by ***Mr. Muli Madhava Reddy bearing Reg. No. 16ETCS002078*** *in partial fulfilment of requirements for the award of B. Tech. Degree in Computer Science and Engineering of Ramaiah University of Applied Sciences.*

May – 2020

Mrs. Vaishali R Kulkarni
Supervisor

Dr. P. V. R. Murthy
Professor and Head – Dept. of CSE

Dr. H. M. Rajashekara Swamy
Professor and Dean-FET

Declaration

Toxic Comments Classification using Deep Learning and Natural Language Processing

The project work is submitted in partial fulfilment of academic requirements for the award of B. Tech. Degree in the Department of Computer Science and Engineering of the Faculty of Engineering and Technology of Ramaiah University of Applied Sciences. The project report submitted herewith is a result of our own work and in conformance to the guidelines on plagiarism as laid out in the University Student Handbook. All sections of the text and results which have been obtained from other sources are fully referenced. We understand that cheating and plagiarism constitute a breach of University regulations, hence this project report has been passed through plagiarism check and the report has been submitted to the supervisor.

Sl. No.	Reg. No.	Student Name	Signature
01	16ETCS002064	Lakshmikanth H V	
02	16ETCS002069	Manoj M	
03	16ETCS002074	Modali Krishna Sai	
04	16ETCS002078	Muli Madhava Reddy	

Date: May 2020

Acknowledgements

We would like to thank our supervisor Mrs. Vaishali R Kulkarni Associate Professor, Department of Computer Science and Engineering, RUAS, for constant guidance and advice on improving the project and encouraging us to do more.

We would like to acknowledge all those who have contributed directly or indirectly to successfully complete the project.

We would like to acknowledge Dr. P V R Murthy, HOD of Computer Science and Engineering department, RUAS and Dr. H M Rajashekara Swamy, Faculty of Engineering Dean, RUAS for providing us the opportunity to research and work on the project.

We would like to express our gratitude to the friends and family who have helped us in the process of the project.

We would like express heartfelt thanks to Mr. Pavan C S working in Juniper Networks for constant guidance in implementation and improvement of the project.

Summary

Toxic comments classification using Deep Learning and Natural Language Processing is the project title, where the comments or paragraphs of text are classified as either toxic or not based on the percentage of toxicity of the sentence. The threat of abuse, racial slurs, cyber bullying or harassment online means that many people stop expressing themselves and give up on seeking opinions that matter. Many celebrities and common people have under gone severe depression after the toxic comments received on the photos, videos and tweets they posted online. To address this issue, we have used Deep Learning models and Natural Language Processing to classify the text and display the toxic percentage of the comment.

Identification and classification of toxic comments using different deep learning models is the main objective of the project. The below mentioned steps are followed to achieve the said objective. The bad words dataset and the comments dataset are first cleaned and tokenized using some python libraries, then feature extraction is done by using Google's word2vec which is a word embedding python library to identify the important features required for the problem statement. Processed dataset is then fed into Convolutional Neural Networks and Recurrent Neural Networks separately to classify the comments into either toxic or not.

As Convolutional Neural Networks are used applied majorly to image visualisations and classification, applying the model on textual data is the main highlight of the project. More efficient models of Recurrent Neural Networks like Long-Short Term Memory and Gated Recurrent Unit models are used on the dataset to get more accurate results.

Table of Contents

List of Tables

List of Figures

Abbreviation and Acronyms

CNN	Convolutional Neural Networks
RNN	Recurrent Neural Networks
LSTM	Long-Short Term Memory
GRU	Gated Recurrent Unit
NLP	Natural Language Processing
CBOW	Continuous Bag of Words
GUI	Graphic User Interface
AI	Artificial Intelligence

1. Introduction

This chapter provides a brief on toxic comments that are posted online and why measures should be taken to avoid the comments using deep learning models and Natural Language Processing (NLP). Motivation behind this dissertation are presented in this chapter along with the scope of the work. Outline on the various chapters in this document are also provided.

1.1 Introduction

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 victims for online abuse range from celebrities to common people. This gives rise to a new problem of identifying and deleting the toxic comments posted online.

Identification and classification of toxic comments using different deep learning models is the main objective of the project. As Convolutional Neural Networks are used applied majorly to image visualisations and classification, applying the model on textual data is the main highlight of the project. More efficient models of Recurrent Neural Networks like Long-Short Term Memory and Gated Recurrent Unit models are used on the dataset to get more accurate results.

1.2 Motivation

Social media is a relatively new phenomenon that has swept the world during the past decade. Social media fuses technology with social interaction via Internet-based applications that allow the creation and exchange of user-generated content. Social media platforms, such as chat rooms, blogging Web sites (e.g., Blogspot), video sites (e.g., YouTube), social networking sites (e.g., Facebook, Myspace, Twitter, Google+), and electronic bulletin boards or forums, as well as e-mail, text messaging, and video chat, have transformed traditional methods of communication by allowing the instantaneous and interactive sharing of information created and controlled by individuals, groups, organizations, and governments. At the end of 2004, the popular social networking site Facebook had close to 1 million users; by June 2011, that number had risen to nearly 700 million users worldwide. Facebook has reported that an average of 30 billion pieces of content (e.g., Web links, news reports, photo albums, blog postings) are shared every month via the social media site. Social media has become fundamental in the way many people and organizations communicate and share opinions, ideas, and information.

As the above mentioned statistics suggests that more and more people are opting for social media to get a daily dose of current affairs, entertainment and much more. This gives rise to anonymous people who try to spread hatred on some of the issues or community or even just a post online. The hatred spread online disturbs the people or community and can send them to depression. There are instances where people committed suicide just because an anonymous person hated there post and criticised to the extent.

The overall idea and motive behind the project is to use data science techniques to handle large corpus of data and Deep Learning models to identify toxic comments to make online platforms a safe place to share information.

1.3 Scope

The main objective of the project is to identify comments entered by the user as either toxic or not by calculating toxic probabilities of the comment.

For the above mentioned objective, the dataset is taken from the Kaggle website and is cleaned according to the requirements. Then important features are extracted using word embedding models. The extracted features are then split into train, validation and test dataset.

The train and validation dataset are they fed into deep learning models with minor tweaks to classify the comments with better accuracy.

Using graphs to compare the results of all the deep learning models implemented to improve and analyse the best model available.

1.4 Organization of the report

This report is organized in different chapters and sections and they are as follows. Literature review with the prior background of the chosen work is discussed in the **Chapter 2**. Critical review of literature is done and summary of that is provided. The research gaps are identified based on the critical analysis of the literature and the problem statement for the dissertation is formed in the Chapter 2. Based on the problem statement, the title, aim and objectives and various methods and methodologies that are used in developing the solution are provided in the **Chapter 3**. A succinct explanation on the various techniques used for developing the various deep learning models and the step wise implementation details of the developed model is also provided in the **Chapter 4**. The results and analysis made on the deep learning models are provided in the **Chapter 5** with the discussion on the obtained results. Cost incurred in implementing the project is provided in **Chapter 6**. Based on the results, a conclusion is drawn. The conclusion and the future direction of work are discussed in the **Chapter 7** in detail.

2.Background Theory

This chapter provides the details on the prior literature work done in the related field of text classification. A critical review of the literature work is done and based on this, research gaps are identified and the problem statement is formulated for this dissertation work.

2.1 Background Theory

Many applications using natural language processing and deep learning find the need of classification of text in the development of this toxic comment classification model.

In this project we implement Convolutional Neural Network (CNN), Gated Recurrent Unit (GRU) and Long-Short Term Memory (LSTM) network to accurately classify the toxic words and we demonstrate the efficiency and F1 scores of the deep learning models mentioned before.

Based on the literature review, we have proposed the following model and the steps on how we intend to proceed

1. To conduct literature survey on feature selection methods and deep learning models for toxic comments classification.
2. To gather data, store and preprocess the acquired datasets.
3. To extract features from the pre-processed data.
4. To develop and train a deep learning model using the extracted features.
5. To interpret results and validate the developed model.
6. To develop a GUI interface for demonstration.

2.2 Few challenges of toxic comments classification:

- Out of vocabulary words: A common problem for the task is the occurrence of words that are not present in the training data. These words include slang or misspellings, but also intentionally obfuscated content.
- Long-range dependencies: The toxicity of a comment often depends on expressions made in early parts of the comment. This is especially problematic for longer comments where the influence of earlier parts on the result can vanish.
- Multi word phrases: We see many occurrences of multi-word phrases in both data-sets. Our algorithms can detect their toxicity only if they can recognize multiple words as a single hateful phrase.

3. Aim and Objectives

This chapter of the record provides the details on the problem statement with identified Aim, objectives and method and methodologies used to analyse and implement the solution using deep learning and NLP.

3.1 Title

- ❖ Toxic comments classification using Deep Learning and Natural Language Processing

3.2 Aim

- ❖ To design and develop a deep learning model to classify toxic words using Deep learning algorithms and Natural language processing approach

3.3 Objectives

1. To conduct literature survey on data preprocessing, feature selection models and deep learning models and its application in toxic comments classification.
2. To acquire, store and preprocess data that comprises of bad words and comments.
3. To design and develop feature selection model to detect and extract important features from the pre-processed data.
4. To design and develop deep learning models to train the extracted features for toxic comments classification.
5. To estimate base line F1 score for NLP based deep learning models.
6. To develop GUI for demonstration.
7. To document the report by comparing the parameters and results of all the models developed

3.4 Methods and Methodology/Approach to attain each objective

Table 3.1: Methods and methodologies.

Objective No.	Statement of the Objective	Method/ Methodology	Resources Utilised
1	To conduct literature survey on data preprocessing, feature selection models and deep learning models and its application in toxic comments classification.	1. To research on data preprocessing techniques and feature selection methods to apply on the natural language dataset. 2. To research on deep learning classification models which can be applied to classify text.	1. Books on toxic comment classification using Deep learning models. 2. Online blogs like Quora, Medium, towards data science etc. 3. YouTube videos. 4. Online courses.
2	To acquire, store and preprocess data that comprises of bad words and comments.	1. To acquire datasets made available on the Kaggle website. 2. To clean the data by removing special characters/symbols, converting the text to lowercase and	1. The dataset of bad words and comments are taken from the website Kaggle.com

		removing stop words. 3. To load the data and split the data into train, validation and test dataset. 4. Tokenizing the dataset.	
3	To design and develop feature selection model to detect and extract important features from the pre-processed data.	1. To implement word embedding using a pre-trained Word2Vec Model.	1. Pre-trained word2vec model provided by Google.
4	To design and develop deep learning models to train the extracted features for toxic comments classification.	1. To design and develop some of the deep learning models like CNN, LSTM and GRU to train and test extracted features for toxic comments classification.	1. Language – Python 2. Libraries – Re, Keras, Sci-kit learn 3. Deep learning models – CNN, LSTM and GRU
5	To estimate base line F1 score for NLP	1. To interpret and compare base line	

	based deep learning models.	F1 score of some of the NLP deep learning models. 2. To choose the model with best parameter values i.e. higher F1 score for both training and testing data.	
6	To develop GUI for demonstration.	1. To create simple GUI to input the comments and get the toxic, insult and obscene level of the comment.	1. PyQt – A python library to create GUI.
7	To document the report by comparing the parameters and results of all the models developed.	1. To document the observations made on all the models implemented.	

4. Problem Solving

This chapter of the report provides the processes involved in the design and development of the deep learning models for toxic comments classification. A brief introduction on the various concepts and techniques used to solve this problem are also discussed in this chapter.

4.1 Data pre-processing.

Data preprocessing is a data mining technique which is used to transform the raw data in a useful and efficient manner. It involves removing all the unwanted text from the large corpus of data and retaining only the important data. The data pre-processing phase is the most important process of the whole project as accuracy of classification of the toxic comments dataset highly depends on data pre-processing.

Steps involved in data pre-processing

1. Removal of special characters. Special characters don't add a special value to the semantic meaning of the sentence so they are removed to reduce the size of sentence in turn reducing the size of dataset.
2. Converting all the sentences in dataset to lowercase. As a sentence which contains both upper and lowercase letters don't differ much from the sentence containing only lowercase letters and also it requires more processing conditions to classify both upper and lowercase letters.
3. Removal of numbers from the dataset. As numbers doesn't add value to the semantic meaning of the sentence.
4. Removal of stop words from dataset. Stop words are for humans to understand a sentence but for a machine they are irrelevant data and add very little semantic meaning to the sentence.

5. After the dataset is reduced by removing all the stop words, numbers and special characters, the words in the dataset are given numbers based on the frequency of their occurrence, this process is called Tokenization.
6. After the tokenization, sentences are reformed with corresponding word indices.
7. To make processing easier the sentences are padded with zeros to make all of the sentences to have same length.

4.2 Feature extraction.

- Feature is an individual measurable property or characteristic of a phenomenon being observed. Choosing informative, discriminating and independent features is a crucial step for effective algorithms in pattern recognition, classification and regression.
- Feature extraction refers to the extraction of linguistic items from the documents to provide a representative sample of their content. Distinctive vocabulary items found in a document are assigned to the different categories by measuring the importance of those items to the document content.
- Feature extraction used in the project is word embeddings, Word embedding is the collective name for a set of language modelling and feature learning techniques in natural language processing where words or phrases from the vocabulary are mapped to vectors of real numbers.
- Pre trained Word2Vec model provided by Google is used to implement word embedding on the dataset.
- Word2Vec contains 300 dimensional vectors for 3 million words. It is a group of related models that are used to produce word embeddings, these models are two-layer neural networks that are trained to reconstruct

linguistic contexts of words. Its objective function causes the words that occur in similar context have similar embeddings.

- A famous example for Word2Vec embeddings is, word vector for the word KING – word vector for the word MAN + word vector for word WOMAN = word vector for word QUEEN. Word2vec groups words with similar contexts to bring out relations out of the sentences and help classify text better.
- Types of Word2Vec
    1. Continuous Bag of Words (CBOW) – In CBOW model tries to predict the target word using the context words.
    2. Skipgram – In Skipgram model tries to predict context words from target word.
- The neural network takes the target word as input and tries to predict the context words and continuously updates weights of the hidden layer, the weights updated are recorded in a new matrix. The weight matrix is then multiplied with the number vector of the word to get the word embedding vector.
- As there are 17,471 words in English dictionary as of 2019, it will require more RAM and separate storage to implement Word2Vec model so Google provides a pre-trained model which is free to use.

4.3 Deep learning models.

Deep learning is an Artificial Intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in AI that has networks capable of learning unsupervised from data that is unstructured or unlabelled. Also known as deep neural learning or deep neural network.

4.3.1 Convolutional Neural Networks (CNN)

- A convolutional neural network is a class of deep neural networks, most commonly applied to analyse visual imagery.
- CNN consists of,
    1. Convolutional layer
    2. Pooling layer
    3. Dense output layer
- Steps involved in classification of comments using CNN,
    1. In Convolution layer, a small window of a sentence which is converted into a 2d matrix of numbers is selected as filters or features to help classify. Features or filters are learnt automatically by the neural networks during the training process.
    2. The numbers in the filters are multiplied with corresponding numbers in the input sentence matrix and the product is recorded in a different matrix which is same size as the filter, this process is repeated for all the filters chosen. At the end of this process there will n resultant matrices if there were n filters chosen.
    3. Then the values in the resultant matrix are added and divided by the size of the matrix. The value got from dividing is placed in a new matrix with the size of input sentence matrix. These steps are repeated for all the resultant matrices present. By the end of this process there will n resultant matrices with the size of input sentence matrix if there were n filters chosen.
    4. All the resultant matrices are fed into an activation function, in our project we are using ReLU which stands for Rectified Linear Unit function as the activation function. ReLU function converts all the numbers in resultant matrix which are below zero to zero. In the next

step all the resultant matrices from activation function are fed into the Pooling layer.

5. In the pooling layer a window size is chosen which is smaller than the resultant matrix and a stride is chosen, stride is number of indices shift over the resultant matrix using window. Now the window is slid across the matrix and the maximum values out of the window is chosen and recorded in a new matrix. The process is repeated for all the resultant matrix produced from the activation function. Pooling layer helps in shrinking down the size of the resultant matrix while preserving the meaning of the matrix.
6. If required, output matrix from the pooling layer can fed into another round of convolution layer, ReLU activation function and pooling layer to even shrink down the size of the matrix.
7. After the before mentioned steps the matrices from all the filters of pooling layer are stacked up into a single column matrix which represents the input sentence.
8. Then the column matrix is compared with actual trained and classified sentences. Comparisons are done by adding all the indices with 1 in actual sentence vector and the indices corresponding to the column matric, then sum of column matrix is divided by the sum of actual sentence vector. This comparison is repeated with all the trained sentences and maximum value is chosen to classify the sentence.

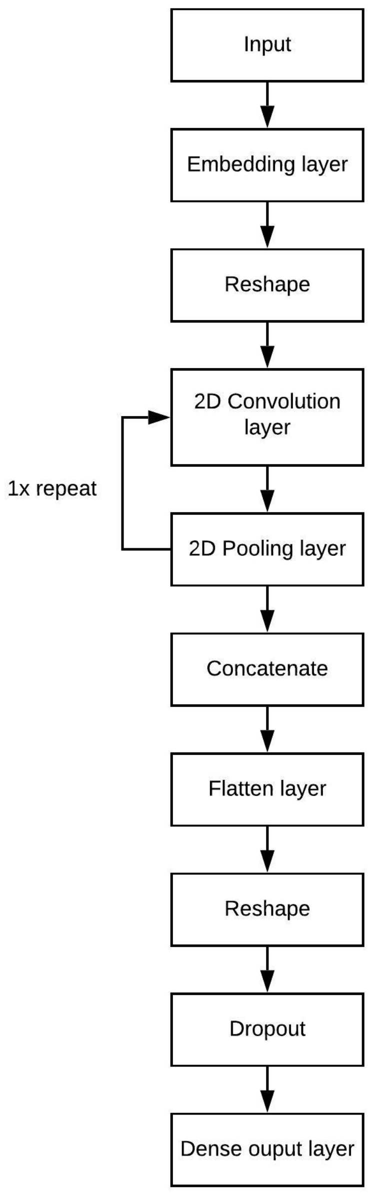


Figure 4.1: Steps of CNN to classify comments dataset.

4.3.2 Long-Short Term Memory (LSTM)

- In Deep Learning, other than convolutional neural networks, we use other kind neural network called Recurrent Neural Networks (RNNs), where the present output is fed as input to the next Step. In CNNs, we see that the outputs and inputs are independent to each other, and the hidden layers perform independently, which work on its set of weights, and biases, do not memorize the previous outputs. In RNNs the hidden layers memorize the outputs and compute the subsequent steps. We can consider the RNN Architecture as a sequential model. The RNNs are usually implemented for the time series models, speech recognition, Natural Language Processing Applications.
- It turns out that one of the problems with a basic RNN model is that it turns into vanishing gradient problems.
- In the forward propagation like in CNN, we will calculate the cost function. To update the weights, back propagation of the cost function is necessary. The neurons which all have included in the forward propagation, need to update their weights in order to minimise the error. Characteristic feature of RNNs is that it’s not just the neurons directly below the output layer that contributed but all of the neurons further back in time. So, it has to propagate all the way back through time to these neurons.
- In backpropagation method, each of the neuron's weight has to update itself proportional to the partial derivative of the error function with respect to the current weight. Backpropagation does this using chain rule of derivatives. This has an effect of multiplying the numbers to compute the gradients. For a n-layer neural network, gradient decreases exponentially with n. So, the value of the gradients for the front layers will be close to zero. This is Vanishing gradient Problem. Lower the gradient is, harder it is for the network to update the weights and the longer it takes to get to the final result.

- One of the solutions for the Vanishing Gradient Problem in RNN is to use Long Short-term Memory Networks (LSTMs).
- LSTMs are a special kind of RNN capable of learning Long term Dependencies. Remembering for longer period of time is its behaviour. LSTM Architecture have the chain of LSTM Cells, instead of single Activation, there are four interacting operations.

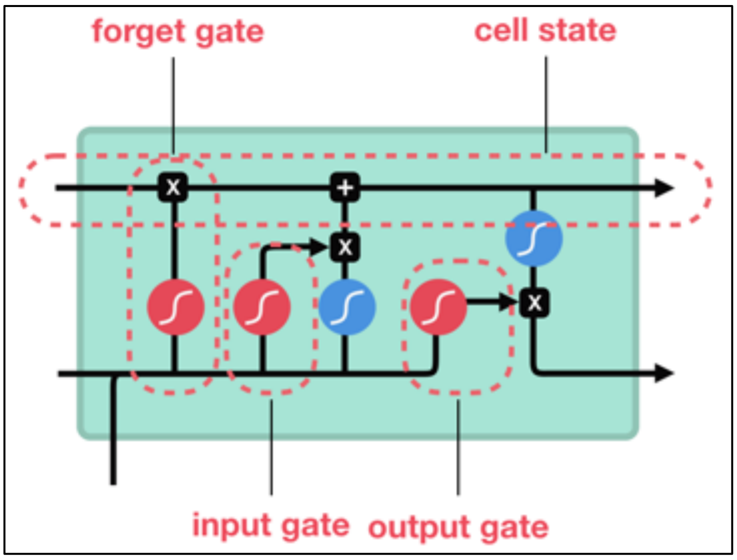


Figure 4.2: The Structure of an LSTM cell

- The key feature in the LSTM unit is the Cell State. It runs straight down the entire chain with only some minor linear interactions. LSTM has the ability to remove or add information to the cell state, carefully regulated by the gates. Each LSTM cell consists of 3 gates, namely, forget gate, update gate, output gate.

Data flow in LSTM cell is explained below.

1. Initially, in the cell, LSTM decides what information is not required by forget gate, the operation of the forget gate uses the sigmoid activation function.

$$f_t = \sigma\left(W_t * [h_{t-1}, x_t] + b_f\right)$$

2. Next step of the LSTM cell is to decide what information is kept along. This is a combination of two computations, First, a sigmoid layer called the input gate layer decides which values to update. Next, a tanh layer creates a vector of new candidate values, C'_t , that could be added to the state.

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i)$$
$$C'_t = \tanh\ (W_c . [h_{t-1}, x_t] + b_c)$$

3. Further, we need to update the cell state, we multiply the old state byf_t , forgetting the things we decided to forget earlier. Then we add $i_t * C'_t$. This is the new candidate values, scaled by how much we decided to update each state value.

$$C_t = f_t * C_{t-1} + i_t * C'_t$$

4. Finally, we need to decide on the output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state will be going to output. Then, we put the cell state through tanh function and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

$$O_t = \sigma(W_o * [h_{t-1}, x_t] + b_o)$$

$$h_t = O_t * \tanh\ (C_t)$$

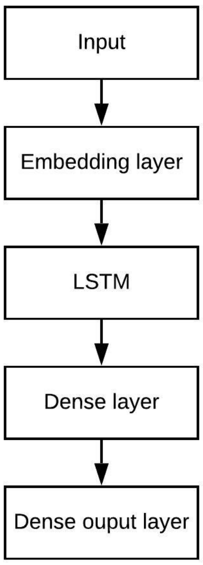


Figure 4.3: Steps for LSTM to classify comments dataset.

4.3.3 Gated Recurrent Unit (GRU)

- Gated Recurrent Unit is another model of RNN which is similar to LSTM but a simpler version of it.

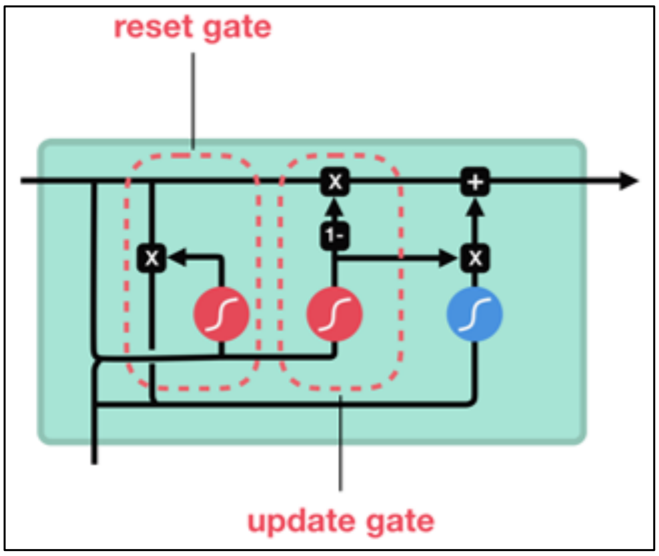


Figure 4.4: The Structure of a GRU cell.

- To solve the vanishing gradient problem of a standard RNN, GRU uses, gates called, **update gate and reset gate**.
- Basically, these are two vectors which decide what information should be passed to the output.

Data flow in GRU cell is explained below,

1. Initially, in the cell, LSTM decides what information is not required by forget gate, the operation of the forget gate uses the sigmoid activation function.

$$G_r = \sigma(W_r * [h_{t-1}, x_t] + b_r).$$

2. Then, the update gate is to be computed with the following mathematical equation.
   $G_u = \sigma(W_u * [h_{t-1}, x_t] + b_u).$
3. Next, a tanh layer creates a vector of new candidate values, C'_t , that could be added to the state with following equation,

$$C'_t = \tanh\,(W_c . [G_r * h_{t-1}, x_t] + b_c)$$

4. Further, we need to update the cell state, we multiply the old state by $(1 - G_r)$, forgetting the things we decided to forget earlier. Then we add $G_u * C'_t$. This is the new candidate values, scaled by how much we decided to update each state value.

$$C_t = (1 - G_u) * C_{t-1} + G_u * C'_t$$

5. Finally, we need to decide what will be going to output. This output will be based on our cell state,

$$O_t = (C_t)$$

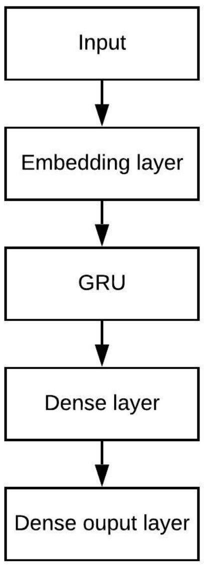
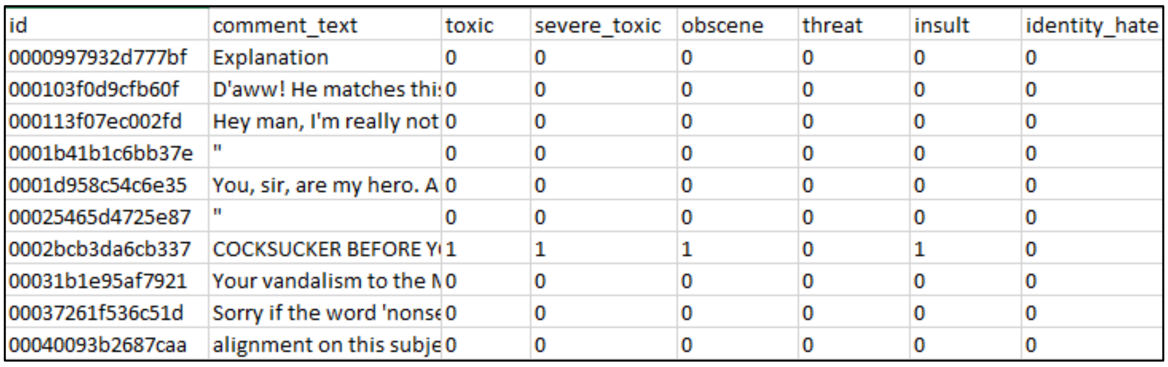


Figure 4.5: Steps for GRU to classify comments dataset.

4.4 Implementation.

Note: All the important parts of the code are explained using comments.

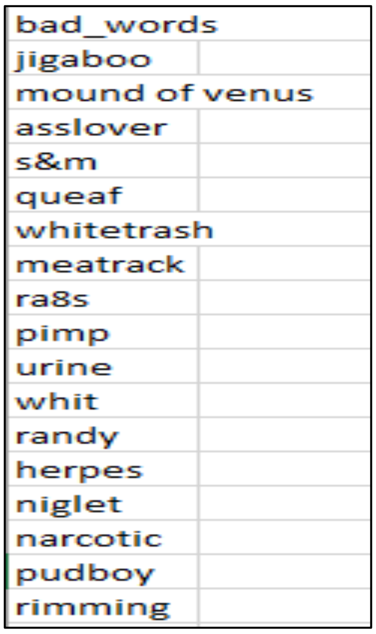
4.4.1 Data pre-processing.



id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0000997932d777bf	Explanation	0	0	0	0	0	0
000103f0d9cfb60f	D'aww! He matches thi	0	0	0	0	0	0
000113f07ec002fd	Hey man, I'm really not	0	0	0	0	0	0
0001b41b1c6bb37e	"	0	0	0	0	0	0
0001d958c54c6e35	You, sir, are my hero. A	0	0	0	0	0	0
00025465d4725e87	"	0	0	0	0	0	0
0002bcb3da6cb337	COCKSUCKER BEFORE Y	1	1	1	0	1	0
00031b1e95af7921	Your vandalism to the M	0	0	0	0	0	0
00037261f536c51d	Sorry if the word 'nonse	0	0	0	0	0	0
00040093b2687caa	alignment on this subje	0	0	0	0	0	0

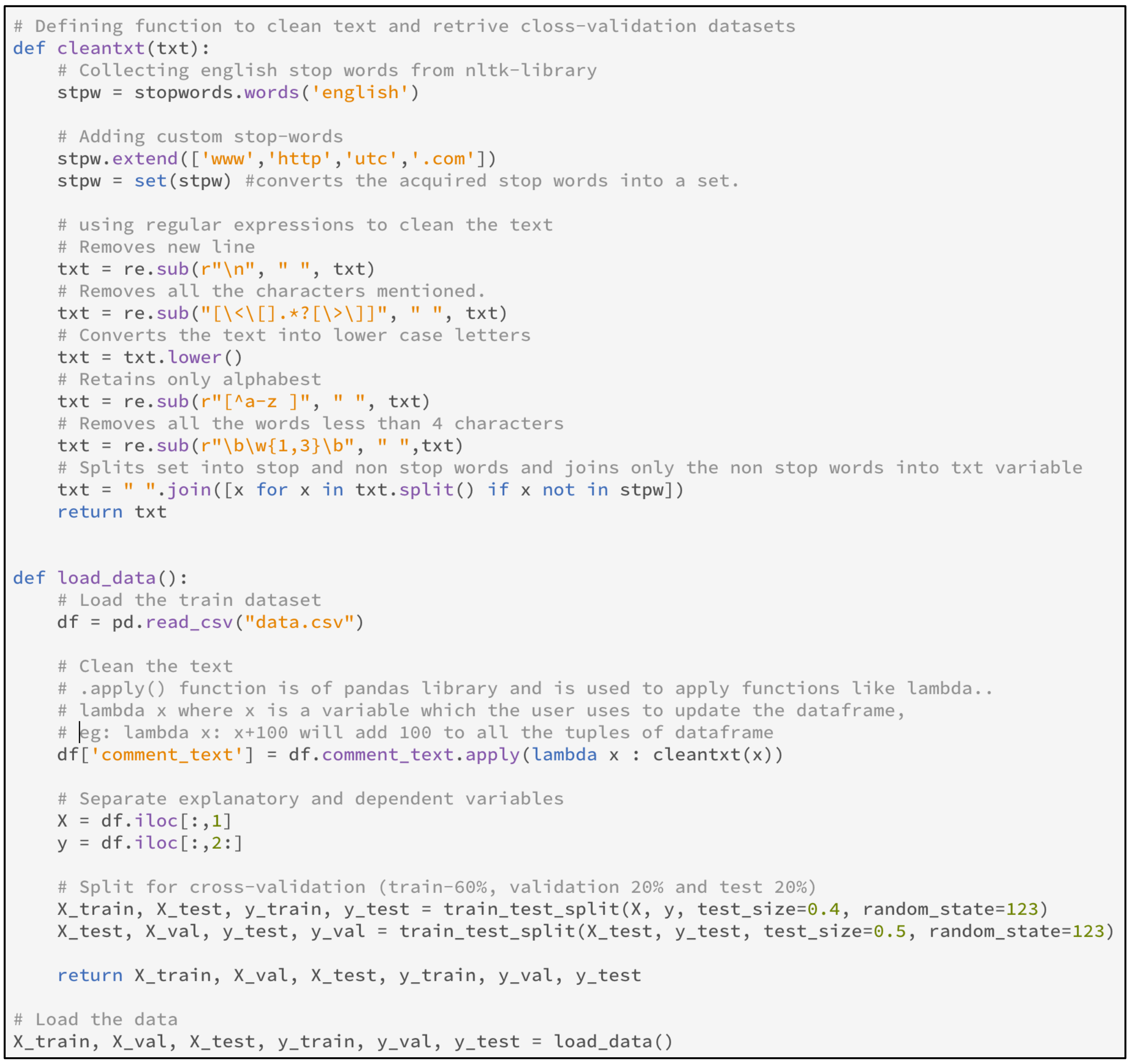
Figure 4.6: Structure of comments dataset.

- 3 attributes that are taken into consideration are Toxic, obscene and insult as comments for other attributes in the dataset are unbalanced.



bad_words
jigaboo
mound of venus
asslover
s&m
queaf
whitetrash
meatrack
ra8s
pimp
urine
whit
randy
herpes
niglet
narcotic
pudboy
rimming

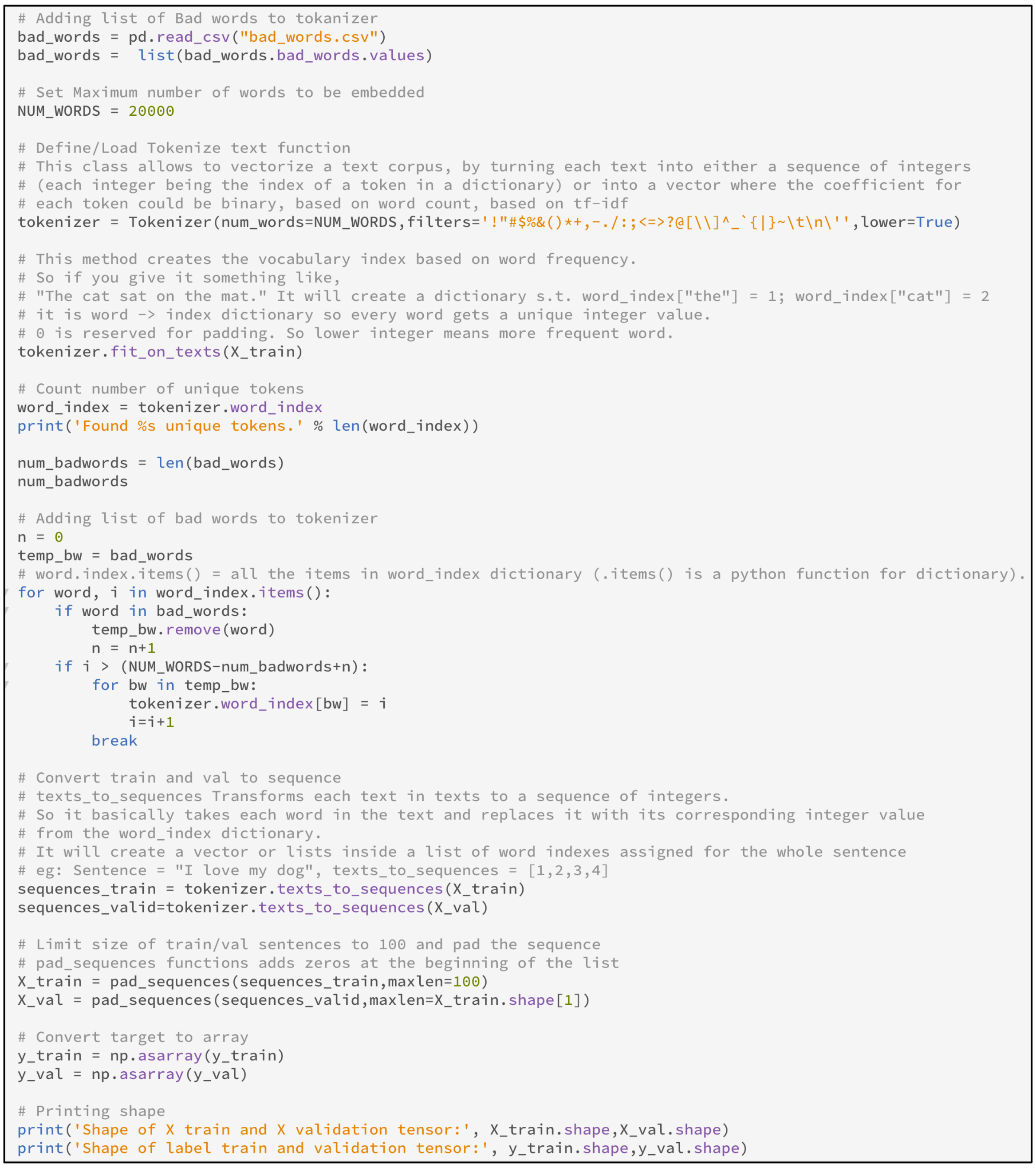
Figure 4.7: Structure of bad words dataset.



```
# Defining function to clean text and retrive closs-validation datasets
def cleantxt(txt):
    # Collecting english stop words from nltk-library
    stpw = stopwords.words('english')

    # Adding custom stop-words
    stpw.extend(['www','http','utc','.com'])
    stpw = set(stpw) #converts the acquired stop words into a set.

    # using regular expressions to clean the text
    # Removes new line
    txt = re.sub(r"\n", " ", txt)
    # Removes all the characters mentioned.
    txt = re.sub("[\<\[].*?[\>\]]", " ", txt)
    # Converts the text into lower case letters
    txt = txt.lower()
    # Retains only alphabest
    txt = re.sub(r"[^a-z ]", " ", txt)
    # Removes all the words less than 4 characters
    txt = re.sub(r"\b\w{1,3}\b", " ",txt)
    # Splits set into stop and non stop words and joins only the non stop words into txt variable
    txt = " ".join([x for x in txt.split() if x not in stpw])
    return txt


def load_data():
    # Load the train dataset
    df = pd.read_csv("data.csv")

    # Clean the text
    # .apply() function is of pandas library and is used to apply functions like lambda..
    # lambda x where x is a variable which the user uses to update the dataframe,
    # eg: lambda x: x+100 will add 100 to all the tuples of dataframe
    df['comment_text'] = df.comment_text.apply(lambda x : cleantxt(x))

    # Separate explanatory and dependent variables
    X = df.iloc[:,1]
    y = df.iloc[:,2:]

    # Split for cross-validation (train-60%, validation 20% and test 20%)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=123)
    X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.5, random_state=123)

    return X_train, X_val, X_test, y_train, y_val, y_test

# Load the data
X_train, X_val, X_test, y_train, y_val, y_test = load_data()
```

Figure 4.8: Data cleaning and loading code snippet.

- In the above figure, method cleantxt() will take sentences as input and clean the text using "re" python library which stands for Regular expression.
- In data cleaning process, special characters, stop words, numbers are removed from the dataset and the sentences in dataset are converted to lower case.
- The load_data() method will read the dataset using "pd" library which stands for pandas and split the dataset into train, validation and test dataset.



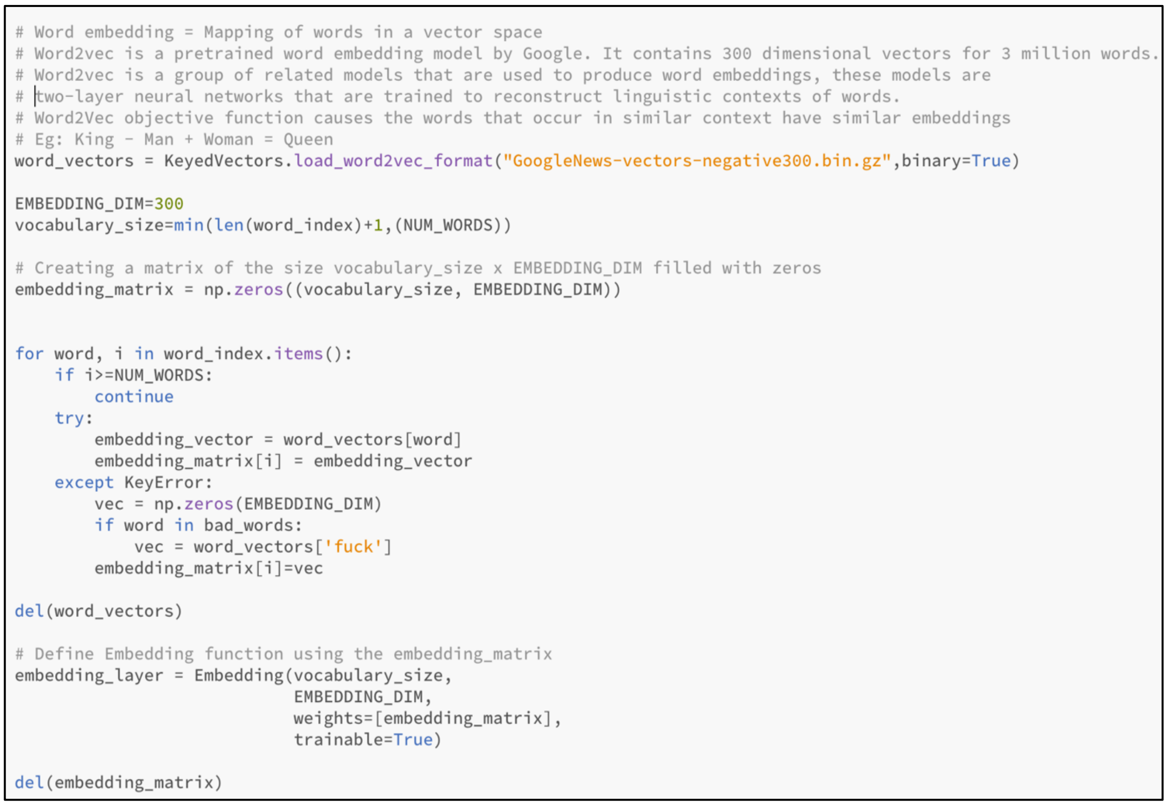
```python
# Adding list of Bad words to tokanizer
bad_words = pd.read_csv("bad_words.csv")
bad_words =  list(bad_words.bad_words.values)

# Set Maximum number of words to be embedded
NUM_WORDS = 20000

# Define/Load Tokenize text function
# This class allows to vectorize a text corpus, by turning each text into either a sequence of integers
# (each integer being the index of a token in a dictionary) or into a vector where the coefficient for
# each token could be binary, based on word count, based on tf-idf
tokenizer = Tokenizer(num_words=NUM_WORDS,filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n\'',lower=True)

# This method creates the vocabulary index based on word frequency.
# So if you give it something like,
# "The cat sat on the mat." It will create a dictionary s.t. word_index["the"] = 1; word_index["cat"] = 2
# it is word -> index dictionary so every word gets a unique integer value.
# 0 is reserved for padding. So lower integer means more frequent word.
tokenizer.fit_on_texts(X_train)

# Count number of unique tokens
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

num_badwords = len(bad_words)
num_badwords

# Adding list of bad words to tokanizer
n = 0
temp_bw = bad_words
# word.index.items() = all the items in word_index dictionary (.items() is a python function for dictionary).
for word, i in word_index.items():
    if word in bad_words:
        temp_bw.remove(word)
        n = n+1
    if i > (NUM_WORDS-num_badwords+n):
        for bw in temp_bw:
            tokenizer.word_index[bw] = i
            i=i+1
        break

# Convert train and val to sequence
# texts_to_sequences Transforms each text in texts to a sequence of integers.
# So it basically takes each word in the text and replaces it with its corresponding integer value
# from the word_index dictionary.
# It will create a vector or lists inside a list of word indexes assigned for the whole sentence
# eg: Sentence = "I love my dog", texts_to_sequences = [1,2,3,4]
sequences_train = tokenizer.texts_to_sequences(X_train)
sequences_valid=tokenizer.texts_to_sequences(X_val)

# Limit size of train/val sentences to 100 and pad the sequence
# pad_sequences functions adds zeros at the beginning of the list
X_train = pad_sequences(sequences_train,maxlen=100)
X_val = pad_sequences(sequences_valid,maxlen=X_train.shape[1])

# Convert target to array
y_train = np.asarray(y_train)
y_val = np.asarray(y_val)

# Printing shape
print('Shape of X train and X validation tensor:', X_train.shape,X_val.shape)
print('Shape of label train and validation tensor:', y_train.shape,y_val.shape)
```

Figure 4.9: Tokenization code snippet.

- In tokenization the sentences in the dataset are divided into words and word index are assigned to every unique word.
- The texts_to_sequences() function combines the word indices into a single list to form a sentence.

- The pad_sequences() function will pad the sentences below the mentioned length with zeros.

4.4.2 Word2Vec word embedding model

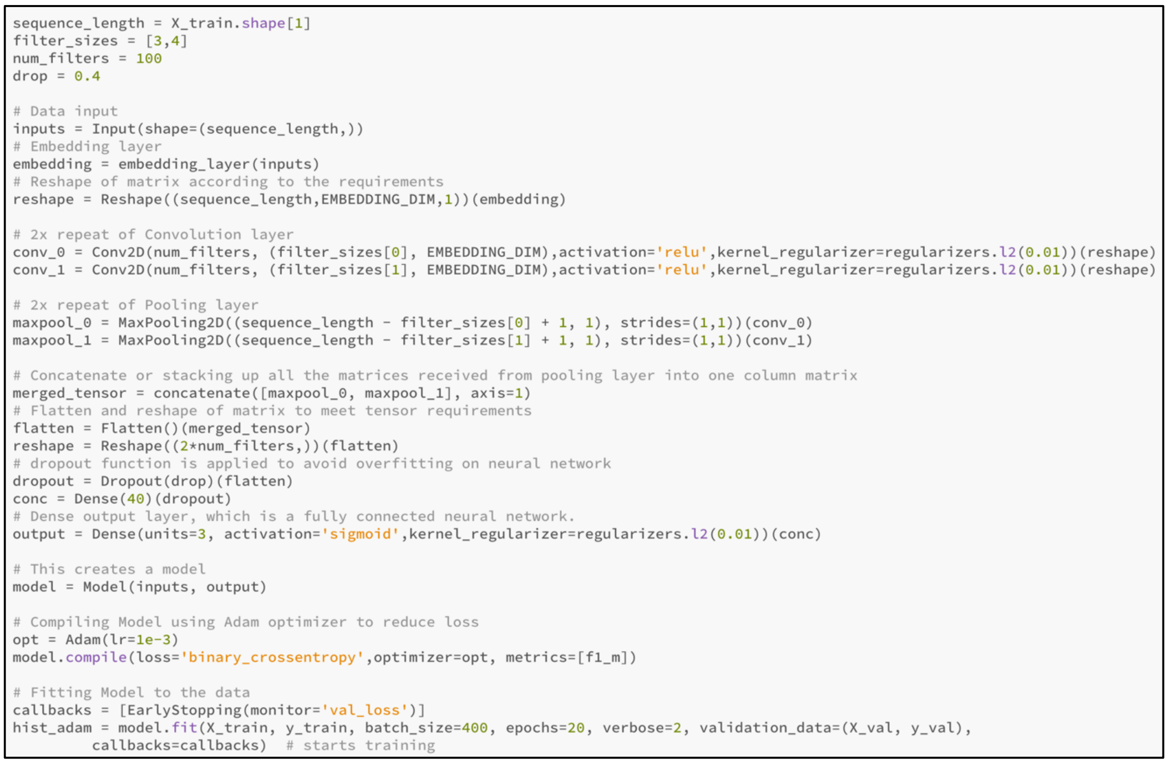


```python
# Word embedding = Mapping of words in a vector space
# Word2vec is a pretrained word embedding model by Google. It contains 300 dimensional vectors for 3 million words.
# Word2vec is a group of related models that are used to produce word embeddings, these models are
# two-layer neural networks that are trained to reconstruct linguistic contexts of words.
# Word2Vec objective function causes the words that occur in similar context have similar embeddings
# Eg: King - Man + Woman = Queen
word_vectors = KeyedVectors.load_word2vec_format("GoogleNews-vectors-negative300.bin.gz",binary=True)

EMBEDDING_DIM=300
vocabulary_size=min(len(word_index)+1,(NUM_WORDS))

# Creating a matrix of the size vocabulary_size x EMBEDDING_DIM filled with zeros
embedding_matrix = np.zeros((vocabulary_size, EMBEDDING_DIM))


for word, i in word_index.items():
    if i>=NUM_WORDS:
        continue
    try:
        embedding_vector = word_vectors[word]
        embedding_matrix[i] = embedding_vector
    except KeyError:
        vec = np.zeros(EMBEDDING_DIM)
        if word in bad_words:
            vec = word_vectors['fuck']
        embedding_matrix[i]=vec

del(word_vectors)

# Define Embedding function using the embedding_matrix
embedding_layer = Embedding(vocabulary_size,
                            EMBEDDING_DIM,
                            weights=[embedding_matrix],
                            trainable=True)

del(embedding_matrix)
```

Figure 4.10: Word2Vec word embeddings code snippet.

- Pre-trained word2vec model is imported to extract important features from the dataset.

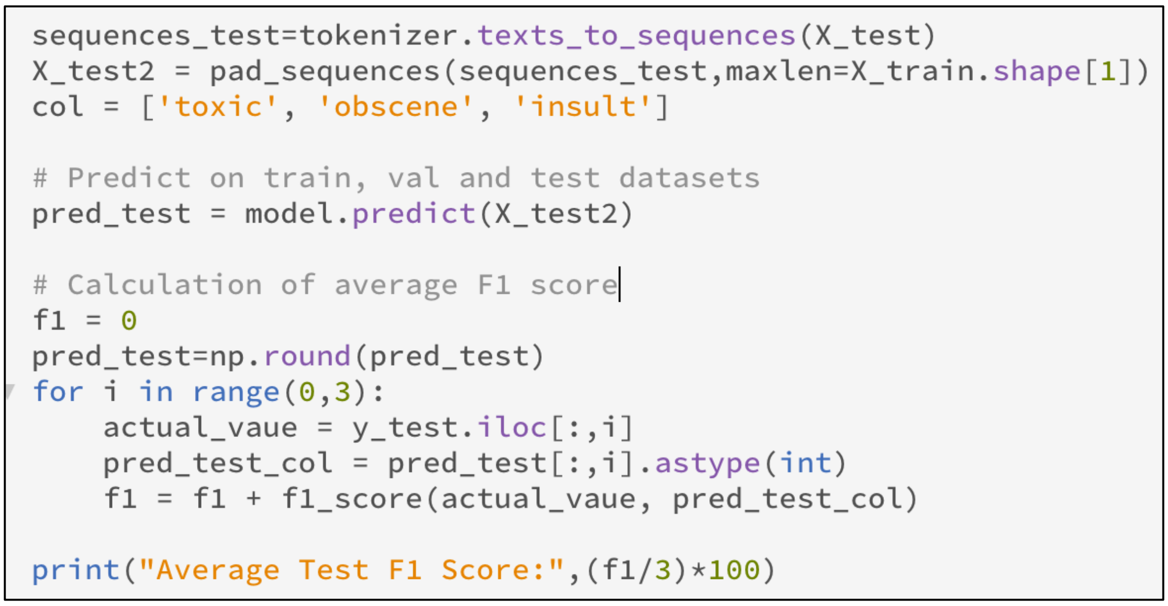
4.4.3 CNN implementation.



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

# Data input
inputs = Input(shape=(sequence_length,))
# Embedding layer
embedding = embedding_layer(inputs)
# Reshape of matrix according to the requirements
reshape = Reshape((sequence_length,EMBEDDING_DIM,1))(embedding)

# 2x repeat of Convolution layer
conv_0 = Conv2D(num_filters, (filter_sizes[0], EMBEDDING_DIM),activation='relu',kernel_regularizer=regularizers.l2(0.01))(reshape)
conv_1 = Conv2D(num_filters, (filter_sizes[1], EMBEDDING_DIM),activation='relu',kernel_regularizer=regularizers.l2(0.01))(reshape)

# 2x repeat of Pooling layer
maxpool_0 = MaxPooling2D((sequence_length - filter_sizes[0] + 1, 1), strides=(1,1))(conv_0)
maxpool_1 = MaxPooling2D((sequence_length - filter_sizes[1] + 1, 1), strides=(1,1))(conv_1)

# Concatenate or stacking up all the matrices received from pooling layer into one column matrix
merged_tensor = concatenate([maxpool_0, maxpool_1], axis=1)
# Flatten and reshape of matrix to meet tensor requirements
flatten = Flatten()(merged_tensor)
reshape = Reshape((2*num_filters,))(flatten)
# dropout function is applied to avoid overfitting on neural network
dropout = Dropout(drop)(flatten)
conc = Dense(40)(dropout)
# Dense output layer, which is a fully connected neural network.
output = Dense(units=3, activation='sigmoid',kernel_regularizer=regularizers.l2(0.01))(conc)

# This creates a model
model = Model(inputs, output)

# Compiling Model using Adam optimizer to reduce loss
opt = Adam(lr=1e-3)
model.compile(loss='binary_crossentropy',optimizer=opt, metrics=[f1_m])

# Fitting Model to the data
callbacks = [EarlyStopping(monitor='val_loss')]
hist_adam = model.fit(X_train, y_train, batch_size=400, epochs=20, verbose=2, validation_data=(X_val, y_val),
         callbacks=callbacks)  # starts training
```

Figure 4.11: CNN code snippet.

- After the features are extracted, it is fed into convolution layer and pooling layer of CNN for 2 times to reduce the size of the matrix.
- The matrices from the pooling layer are then concatenated, flattened and reshaped to meet tensor flow requirements.
- Then the matrices are fed into fully connected neural network (Dense output layer) with Adam optimiser to reduce the loss.
- At the last step the model is fit with number of epochs mentioned.

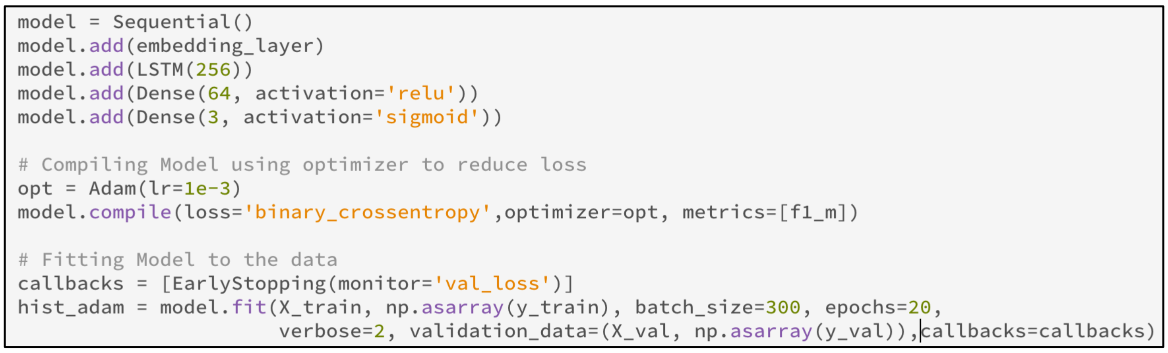


```
sequences_test=tokenizer.texts_to_sequences(X_test)
X_test2 = pad_sequences(sequences_test,maxlen=X_train.shape[1])
col = ['toxic', 'obscene', 'insult']

# Predict on train, val and test datasets
pred_test = model.predict(X_test2)

# Calculation of average F1 score
f1 = 0
pred_test=np.round(pred_test)
for i in range(0,3):
    actual_vaue = y_test.iloc[:,i]
    pred_test_col = pred_test[:,i].astype(int)
    f1 = f1 + f1_score(actual_vaue, pred_test_col)

print("Average Test F1 Score:",(f1/3)*100)
```

Figure 4.12: Model prediction code snippet.

- The model.predict() function is applied on test dataset to classify the comments.
- Average F1 score are calculated for test dataset on 3 attributes.

4.4.4 LSTM implementation.



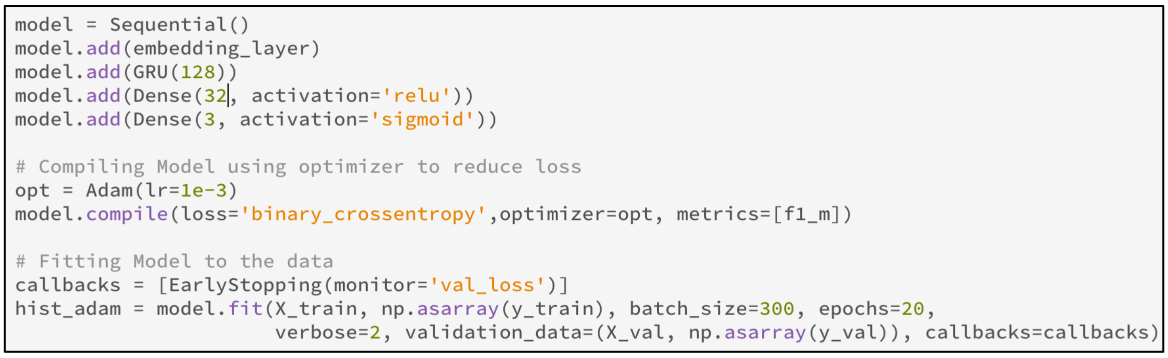
```
model = Sequential()
model.add(embedding_layer)
model.add(LSTM(256))
model.add(Dense(64, activation='relu'))
model.add(Dense(3, activation='sigmoid'))

# Compiling Model using optimizer to reduce loss
opt = Adam(lr=1e-3)
model.compile(loss='binary_crossentropy',optimizer=opt, metrics=[f1_m])

# Fitting Model to the data
callbacks = [EarlyStopping(monitor='val_loss')]
hist_adam = model.fit(X_train, np.asarray(y_train), batch_size=300, epochs=20,
                      verbose=2, validation_data=(X_val, np.asarray(y_val)),callbacks=callbacks)
```

Figure 4.13: LSTM code snippet.

- After the features are extracted, we feed the featured vector to the embedding layer (word2vec) and the output is fed into LSTM layer with 256 neurons(cells).
- The matrices which were the output from the LSTM layer is given as input to the Dense layer with the 64 neurons.

- Then the matrices are fed into Dense output layer with 3 neurons with Adam optimiser to reduce the loss and the final classification is done.

4.4.5 GRU implementation.

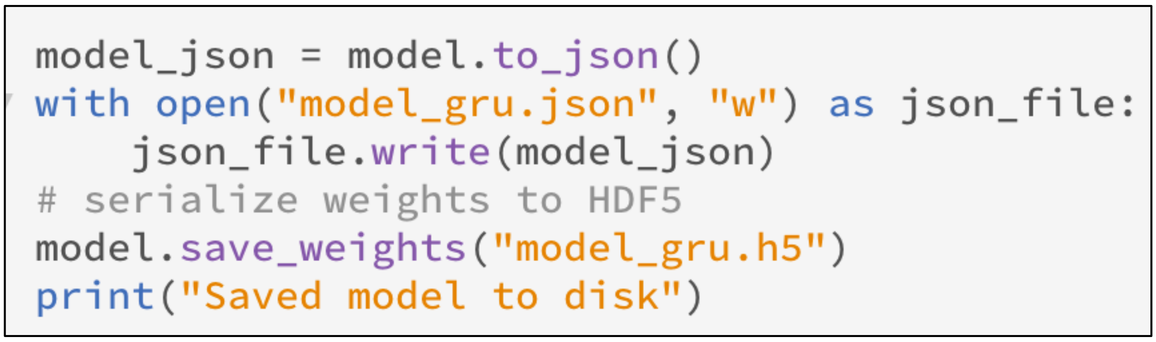


```
model = Sequential()
model.add(embedding_layer)
model.add(GRU(128))
model.add(Dense(32, activation='relu'))
model.add(Dense(3, activation='sigmoid'))

# Compiling Model using optimizer to reduce loss
opt = Adam(lr=1e-3)
model.compile(loss='binary_crossentropy',optimizer=opt, metrics=[f1_m])

# Fitting Model to the data
callbacks = [EarlyStopping(monitor='val_loss')]
hist_adam = model.fit(X_train, np.asarray(y_train), batch_size=300, epochs=20,
                      verbose=2, validation_data=(X_val, np.asarray(y_val)), callbacks=callbacks)
```

Figure 4.14: GRU code snippet.

- After the features are extracted, we feed the featured vector to the embedding layer (word2vec) and the output is fed into GRU layer with 128 neurons(cells).
- The matrices which were the output from the GRU layer is given as input to the Dense layer with the 32 neurons.
- Then the matrices are fed into Dense output layer with 3 neurons with Adam optimiser to reduce the loss and the final classification is done.

4.4.6 GUI implementation.



```
model_json = model.to_json()
with open("model_gru.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model_gru.h5")
print("Saved model to disk")
```

Figure 4.15: Model save code snippet.

- After the model is trained, the weights of the neural network are saved to test the model on user given comments.
- The model is saved with. json extension.

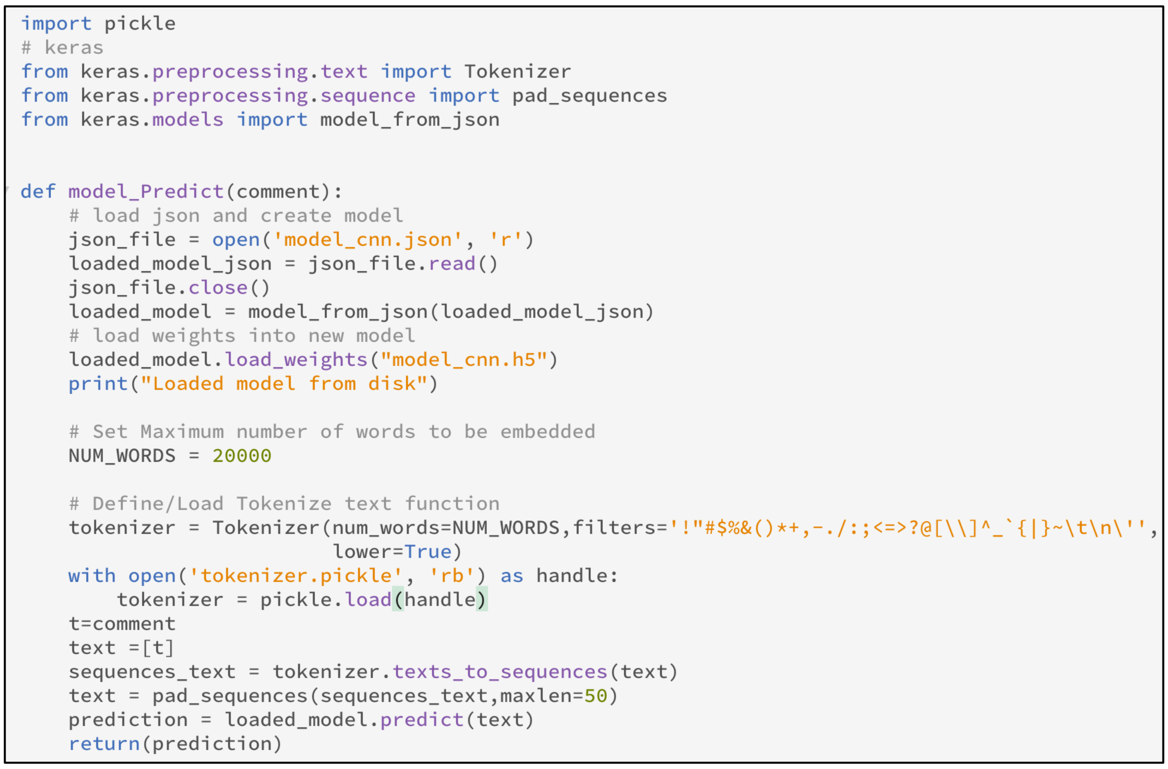
```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Dialog(object):
    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.resize(700, 463)

        self.label_1 = QtWidgets.QLabel(Dialog)
        self.label_1.setGeometry(QtCore.QRect(240, 60, 241, 61))
        font = QtGui.QFont()
        font.setPointSize(12)
        font.setBold(True)
        font.setWeight(75)
        self.label_1.setFont(font)
        self.label_1.setObjectName("label_1")

        self.label_2 = QtWidgets.QLabel(Dialog)
        self.label_2.setGeometry(QtCore.QRect(130, 160, 175, 16))
        self.label_2.setFont(font)
        self.label_2.setObjectName("label_2")

        font = QtGui.QFont()
        font.setPointSize(10)

        self.lineEdit1 = QtWidgets.QLineEdit(Dialog)
        self.lineEdit1.setGeometry(QtCore.QRect(310, 160, 300, 20))
        self.lineEdit1.setObjectName("lineEdit1")

        self.pushButton = QtWidgets.QPushButton(Dialog)
        self.pushButton.setGeometry(QtCore.QRect(290, 225, 112, 34))
        font = QtGui.QFont()
        font.setPointSize(10)
        self.pushButton.setFont(font)
        self.pushButton.setObjectName("pushButton")

        font = QtGui.QFont()
        font.setPointSize(10)
        font.setWeight(50)
```

Figure 4.16: GUI design code snippet.

```
        self.label_3 = QtWidgets.QLabel(Dialog)
        self.label_3.setGeometry(QtCore.QRect(200, 275, 210, 50))
        self.label_3.setFont(font)

        self.label_4 = QtWidgets.QLabel(Dialog)
        self.label_4.setGeometry(QtCore.QRect(200, 300, 210, 50))
        self.label_4.setFont(font)

        self.label_5 = QtWidgets.QLabel(Dialog)
        self.label_5.setGeometry(QtCore.QRect(200, 325, 210, 50))
        self.label_5.setFont(font)

        font = QtGui.QFont()
        font.setPointSize(12)
        font.setBold(True)
        font.setWeight(75)
        self.label_6 = QtWidgets.QLabel(Dialog)
        self.label_6.setGeometry(QtCore.QRect(15, 380, 700, 50))
        self.label_6.setFont(font)
        self.retranslateUi(Dialog)
        QtCore.QMetaObject.connectSlotsByName(Dialog)

    def retranslateUi(self, Dialog):
        _translate = QtCore.QCoreApplication.translate
        Dialog.setWindowTitle(_translate("Dialog", "Dialog"))
        self.label_1.setText(_translate("Dialog", "Toxic Comment Classfication"))
        self.label_2.setText(_translate("Dialog", "Enter the comment"))
        self.label_3.setText(_translate("Dialog", ""))
        self.label_4.setText(_translate("Dialog", ""))
        self.label_5.setText(_translate("Dialog", ""))
        self.label_6.setText(_translate("Dialog", ""))
        self.pushButton.setText(_translate("Dialog", "Classify"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    Dialog = QtWidgets.QDialog()
    ui = Ui_Dialog()
    ui.setupUi(Dialog)
    Dialog.show()
    sys.exit(app.exec_())
```

Figure 4.17: GUI design code snippet.

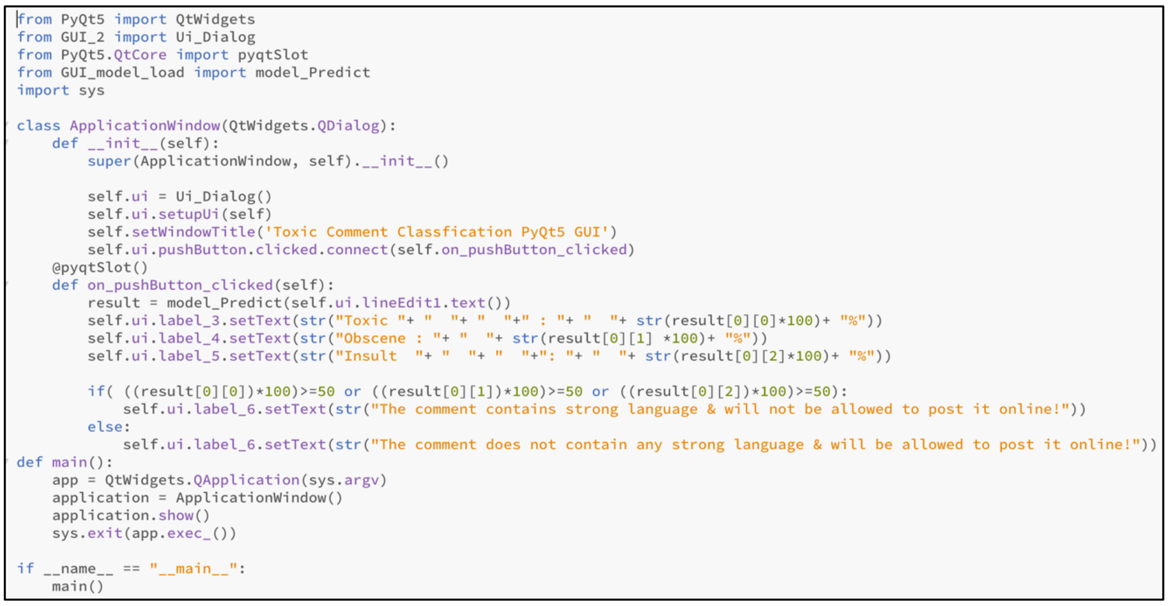
- Figure 4.8 and 4.9 are python code for GUI design using PyQt library and its functions.



```
import pickle
# keras
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import model_from_json


def model_Predict(comment):
    # load json and create model
    json_file = open('model_cnn.json', 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)
    # load weights into new model
    loaded_model.load_weights("model_cnn.h5")
    print("Loaded model from disk")

    # Set Maximum number of words to be embedded
    NUM_WORDS = 20000

    # Define/Load Tokenize text function
    tokenizer = Tokenizer(num_words=NUM_WORDS,filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n\'',
                          lower=True)
    with open('tokenizer.pickle', 'rb') as handle:
        tokenizer = pickle.load(handle)
    t=comment
    text =[t]
    sequences_text = tokenizer.texts_to_sequences(text)
    text = pad_sequences(sequences_text,maxlen=50)
    prediction = loaded_model.predict(text)
    return(prediction)
```

Figure 4.18: Loading the deep learning code snippet.

- The model saved which is showed in figure 4.7 is loaded into the GUI design using the above written code.
- The comment entered by the user is tokenised and classified using the saved weights.



```python
from PyQt5 import QtWidgets
from GUI_2 import Ui_Dialog
from PyQt5.QtCore import pyqtSlot
from GUI_model_load import model_Predict
import sys

class ApplicationWindow(QtWidgets.QDialog):
    def __init__(self):
        super(ApplicationWindow, self).__init__()

        self.ui = Ui_Dialog()
        self.ui.setupUi(self)
        self.setWindowTitle('Toxic Comment Classfication PyQt5 GUI')
        self.ui.pushButton.clicked.connect(self.on_pushButton_clicked)
    @pyqtSlot()
    def on_pushButton_clicked(self):
        result = model_Predict(self.ui.lineEdit1.text())
        self.ui.label_3.setText(str("Toxic "+ "  "+ "  "+" : "+ "  "+ str(result[0][0]*100)+ "%"))
        self.ui.label_4.setText(str("Obscene : "+ "  "+ str(result[0][1] *100)+ "%"))
        self.ui.label_5.setText(str("Insult  "+ "  "+ "  "+": "+ "  "+ str(result[0][2]*100)+ "%"))

        if( ((result[0][0])*100)>=50 or ((result[0][1])*100)>=50 or ((result[0][2])*100)>=50):
            self.ui.label_6.setText(str("The comment contains strong language & will not be allowed to post it online!"))
        else:
            self.ui.label_6.setText(str("The comment does not contain any strong language & will be allowed to post it online!"))
def main():
    app = QtWidgets.QApplication(sys.argv)
    application = ApplicationWindow()
    application.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()
```

Figure 4.19: GUI code snippet.

- Codes which are shown in figure 4.8, 4.9 and 4.10 are imported into a single python file.
- After the probabilities of toxic, insult and obscene is printed out by the model, it is converted into percentage and a threshold is put whether the sentence can be allowed to post online.

Table 5.1: Confusion matrix.

Predicted values

Actual values

	Negative	Positive
Negative	True negative	False positive
Positive	False negative	True positive

- Positive and negative classes are toxic and non-toxic comments in the database.
- True negative – Model correctly predicts the negative class.
- True positive – Model correctly predicts the positive class.
- False negative – Model incorrectly predicts the negative class.
- False positive – Model incorrectly predicts the positive class.
- Precision – The percentage of results that are relevant,

$$P = \frac{True\ positive}{True\ positive\ +\ False\ positive}$$

- Recall – The percentage of total relevant results correctly by the model,

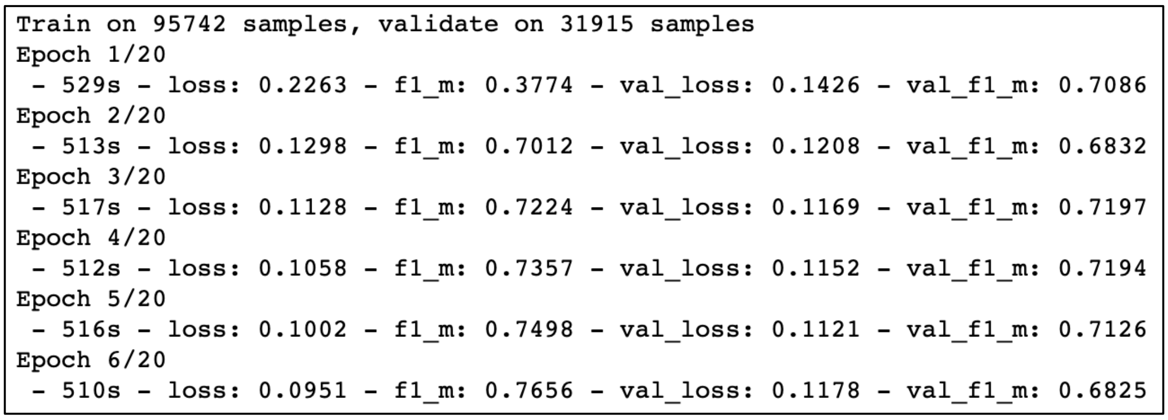
$$R = \frac{True\ positive}{True\ positive\ +\ False\ negative}$$

- F1 score – It is a function of precision and recall,

$$F1 = 2 * \frac{precision\ *\ recall}{precision\ +\ recall}$$

- As F1 score contains both precision and recall it is used as the evaluation metric for all the deep learning models.

5.1 Results and analysis of CNN model.



```
Train on 95742 samples, validate on 31915 samples
Epoch 1/20
 - 529s - loss: 0.2263 - f1_m: 0.3774 - val_loss: 0.1426 - val_f1_m: 0.7086
Epoch 2/20
 - 513s - loss: 0.1298 - f1_m: 0.7012 - val_loss: 0.1208 - val_f1_m: 0.6832
Epoch 3/20
 - 517s - loss: 0.1128 - f1_m: 0.7224 - val_loss: 0.1169 - val_f1_m: 0.7197
Epoch 4/20
 - 512s - loss: 0.1058 - f1_m: 0.7357 - val_loss: 0.1152 - val_f1_m: 0.7194
Epoch 5/20
 - 516s - loss: 0.1002 - f1_m: 0.7498 - val_loss: 0.1121 - val_f1_m: 0.7126
Epoch 6/20
 - 510s - loss: 0.0951 - f1_m: 0.7656 - val_loss: 0.1178 - val_f1_m: 0.6825
```

Figure 5.1: Epochs of CNN model implemented on training and validation dataset.

- The model is trained on 95742 comments and validated on 31915 sentences.
- The model runs for 6 epochs and stops as an early stopping function is applied to stop the model whenever the Validation loss increases after the decrease.
- F1 score on training dataset – 74.98%
- F1 score on validation dataset – 71.26%

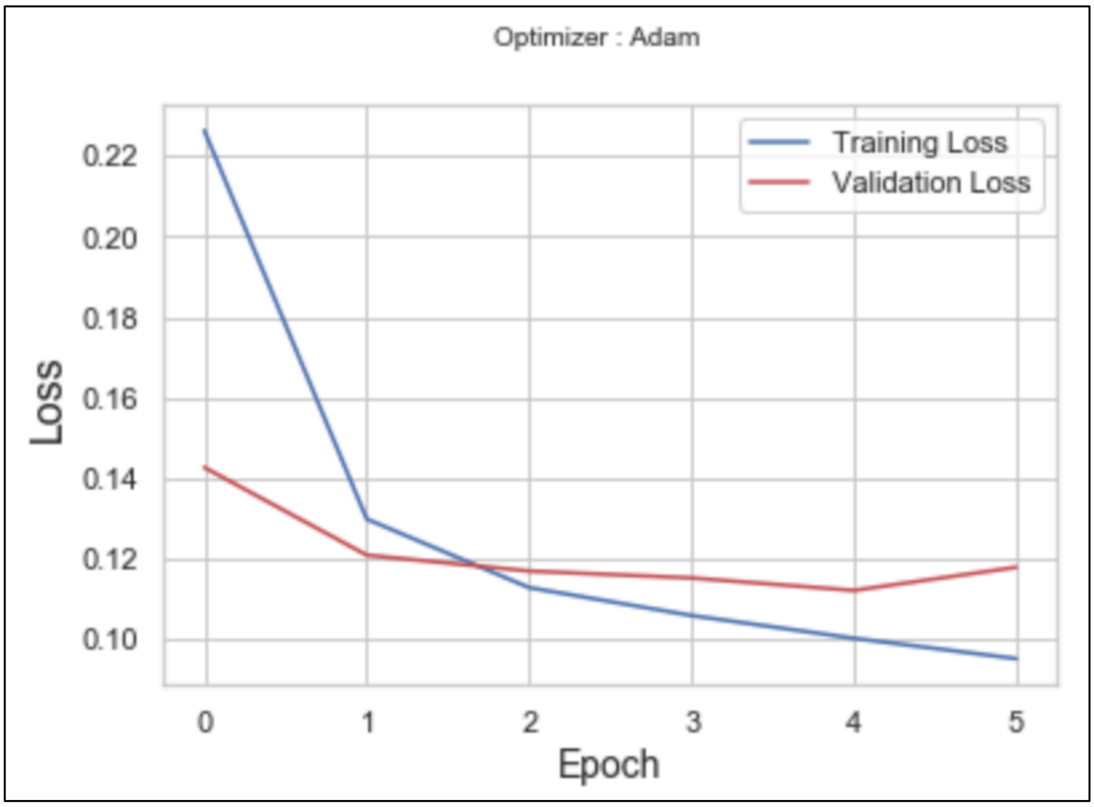
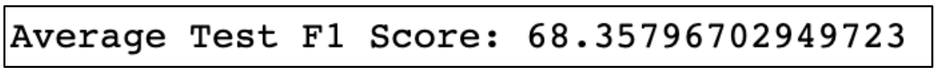


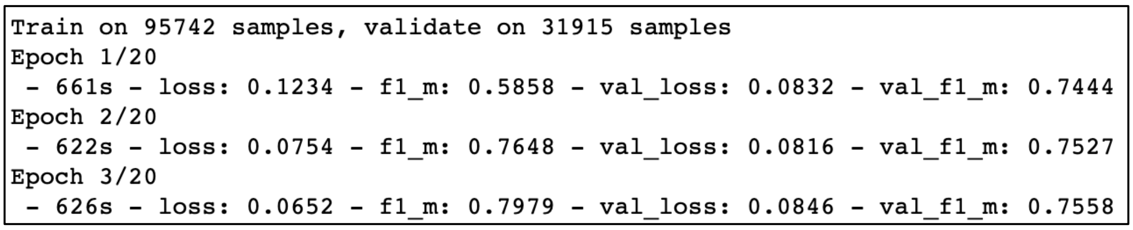
Figure 5.2: Loss V/S Epoch graph of CNN implementation on training and validation dataset. Training loss is 10.02% and Validation loss is 11.21%.



```
Average Test F1 Score: 68.35796702949723
```

Figure 5.3: Average test F1 score of CNN model implemented on test dataset.

5.2 Results and analysis of LSTM model

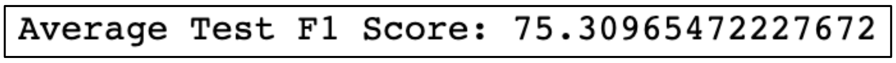


```
Train on 95742 samples, validate on 31915 samples
Epoch 1/20
 - 661s - loss: 0.1234 - f1_m: 0.5858 - val_loss: 0.0832 - val_f1_m: 0.7444
Epoch 2/20
 - 622s - loss: 0.0754 - f1_m: 0.7648 - val_loss: 0.0816 - val_f1_m: 0.7527
Epoch 3/20
 - 626s - loss: 0.0652 - f1_m: 0.7979 - val_loss: 0.0846 - val_f1_m: 0.7558
```

Figure 5.4: Epochs of LSTM model implemented on training and validation dataset.

- The model is trained on 95742 comments and validated on 31915 sentences.

- The model runs for 3 epochs and stops as an early stopping function is applied to stop the model whenever the Validation loss increases after the decrease.
- F1 score on training dataset – 76.48 %
- F1 score on validation dataset – 75.27%



```
Average Test F1 Score: 75.30965472227672
```

Figure 5.5: Average test F1 score of LSTM model implemented on test dataset.

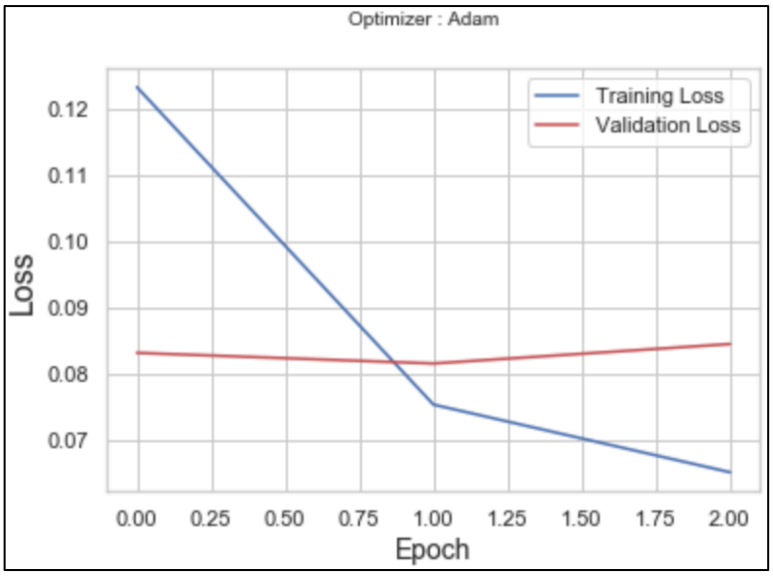
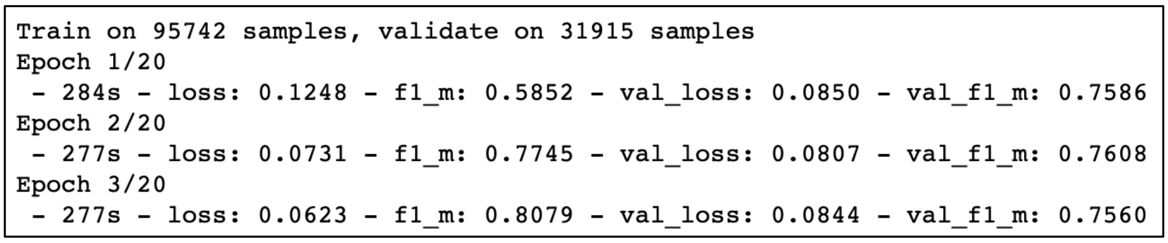


Figure 5.6: Loss V/S Epoch graph of LSTM implementation on training and validation dataset. Training loss is 6.52% and Validation loss is 8.46%.

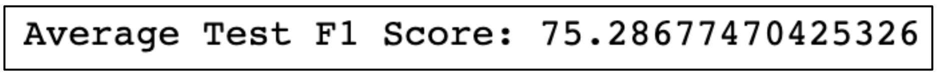
5.3 Results and analysis of GRU model.



```
Train on 95742 samples, validate on 31915 samples
Epoch 1/20
 - 284s - loss: 0.1248 - f1_m: 0.5852 - val_loss: 0.0850 - val_f1_m: 0.7586
Epoch 2/20
 - 277s - loss: 0.0731 - f1_m: 0.7745 - val_loss: 0.0807 - val_f1_m: 0.7608
Epoch 3/20
 - 277s - loss: 0.0623 - f1_m: 0.8079 - val_loss: 0.0844 - val_f1_m: 0.7560
```

Figure 5.7: Epochs of GRU model implemented on training and validation dataset.

- The model is trained on 95742 comments and validated on 31915 sentences.
- The model runs for 3 epochs and stops as an early stopping function is applied to stop the model whenever the Validation loss increases after the decrease.
- F1 score on training dataset – 77.45 %
- F1 score on validation dataset – 76.08%



```
Average Test F1 Score: 75.28677470425326
```

Figure 5.8: Average test F1 score of GRU model implemented on test dataset

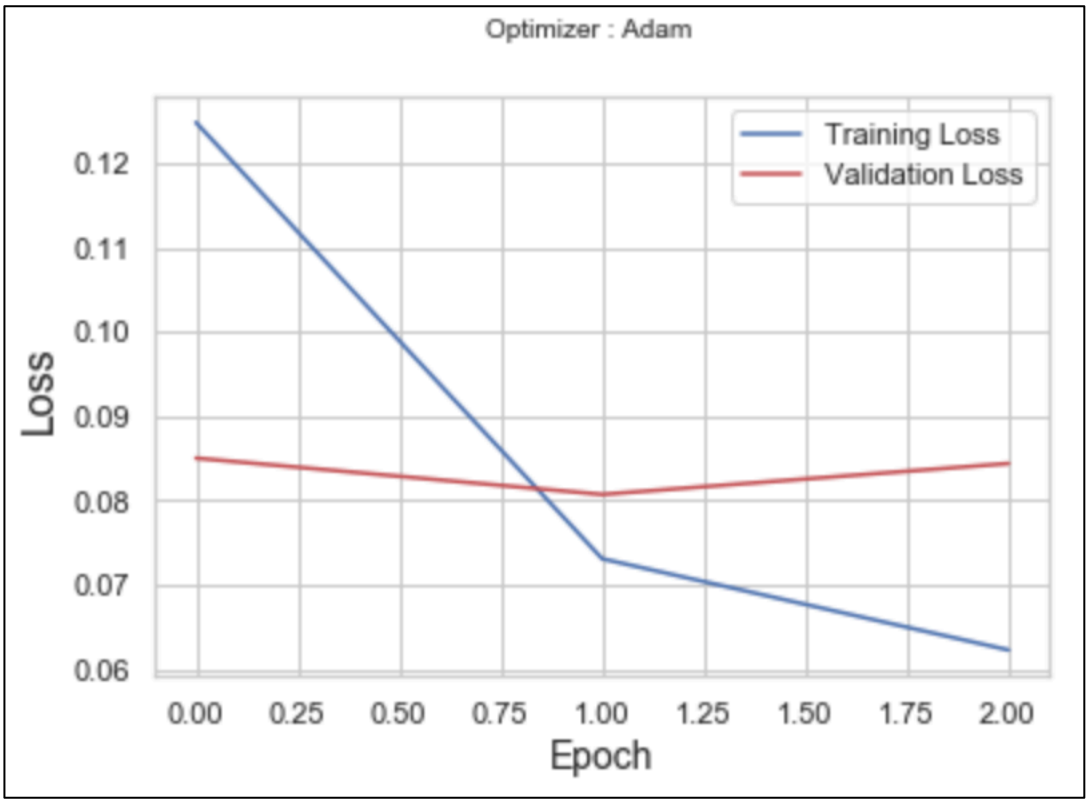


Figure 5.9: Loss V/S Epoch graph of GRU implementation on training and validation dataset. Training loss is 7.31% and Validation loss is 8.07%.

1.4 Comparison between CNN, LSTM and GRU.

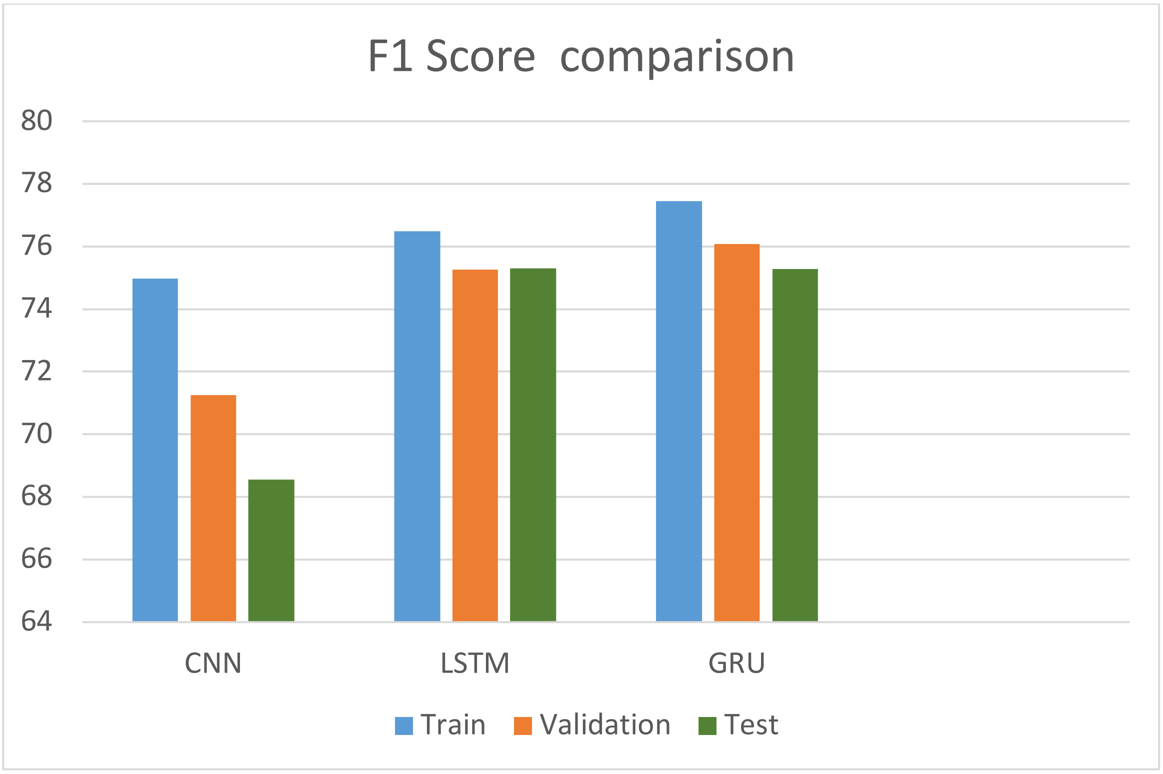


Figure 5.10: Bar chart with F1 scores of train, validation and test dataset of CNN, LSTM and GRU.

- When compared based on F1 scores LSTM model has the highest F1 score for train, validation and test dataset among all the models.

1.5 GUI design

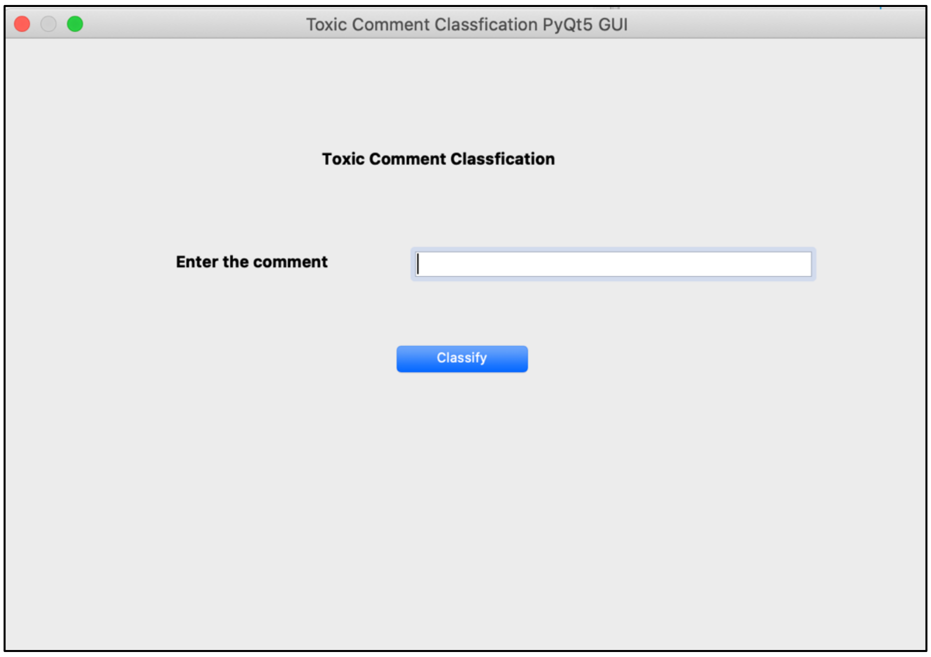
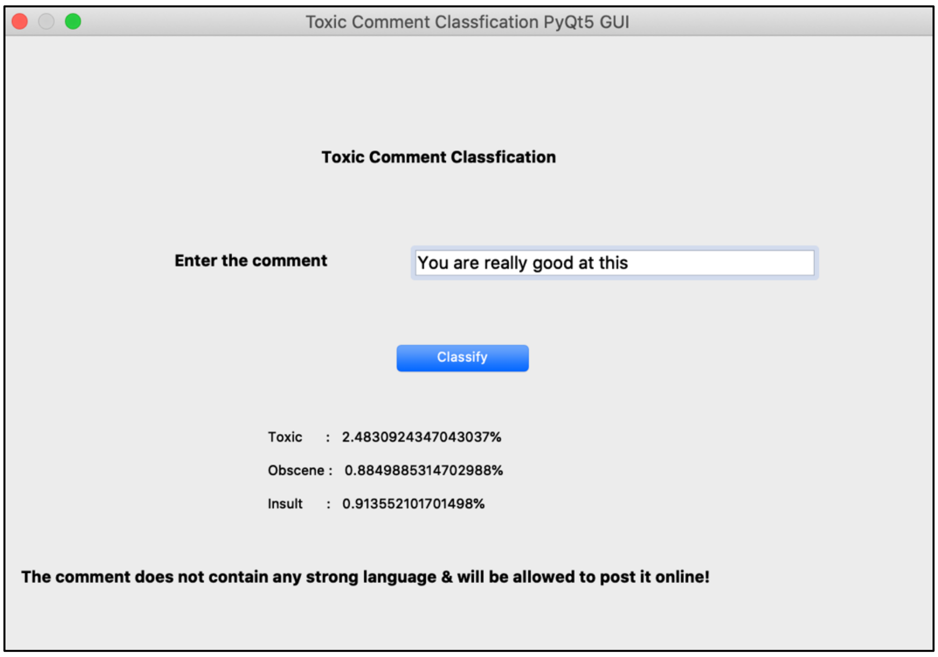

Toxic Comment Classfication PyQt5 GUI
Toxic Comment Classfication
Enter the comment
Classify

Figure 5.11: GUI design of the project using PyQt python library.

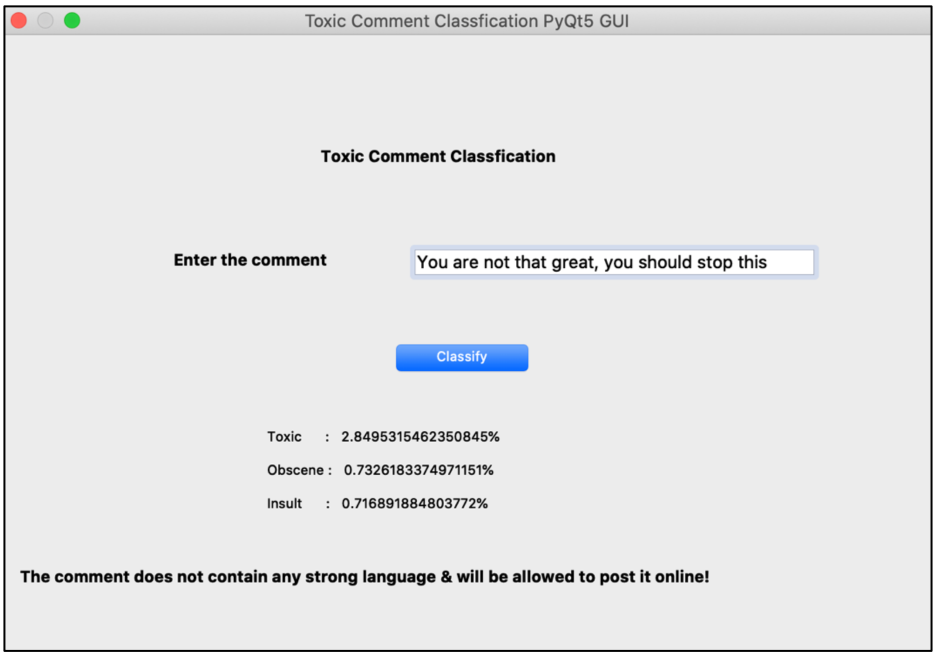

Toxic Comment Classfication PyQt5 GUI
Toxic Comment Classfication
Enter the comment
Classify

You are really good at this

```
Toxic    :  2.4830924347043037%
Obscene :  0.8849885314702988%
Insult   :  0.913552101701498%
```

The comment does not contain any strong language & will be allowed to post it online!

Figure 5.12: User comment tested using the saved model of LSTM with a non-toxic comment.



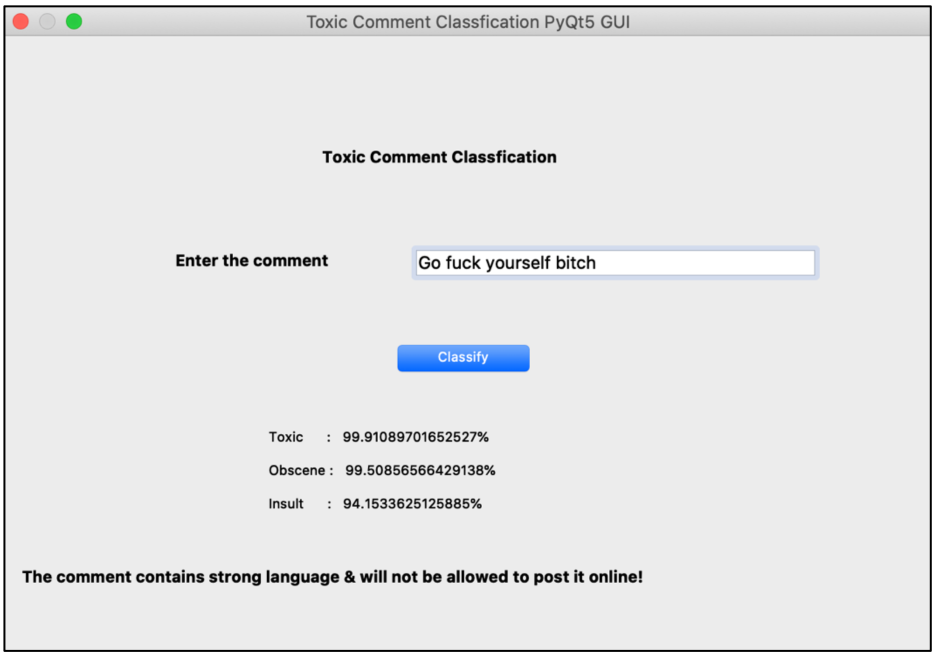
Toxic Comment Classfication

Enter the comment: You are not that great, you should stop this

```
Toxic    :  2.8495315462350845%
Obscene :  0.7326183374971151%
Insult   :  0.716891884803772%
```

The comment does not contain any strong language & will be allowed to post it online!

Figure 5.13: User comment tested using the saved model of LSTM with a non-toxic comment but with a slight hatred included.



Toxic Comment Classfication

Enter the comment: Go fuck yourself bitch

```
Toxic    :  99.91089701652527%
Obscene :  99.50856566429138%
Insult   :  94.1533625125885%
```

The comment contains strong language & will not be allowed to post it online!

Figure 5.14: User comment tested using the saved model of LSTM with a toxic comment.

6. Project Costing

Table 6.1: Project costing

Sl. No	Component	Estimated Cost
1	Literature survey	4 persons*2500/-
2	Data gathering, data cleaning and data pre-processing	2 persons*3000/-
3	Deep learning model implementation	4 persons*5000/-
4	GUI design	3 persons*4000/-
5	Testing	4 persons*3000/-
6	Documentation	4 persons*1500/-
7	Maintenance	4 persons*500/-
	Total:	68,000

7. Conclusions and Suggestions for Future Work

This chapter provides the conclusions on the results obtained from the different deep learning models. The future direction of work that can be carried out based on this dissertation is also discussed further.

7.1 Conclusions

- As per the literature survey conducted deep learning algorithms found to provide more accurate results compared to machine learning algorithms.
- Pre-processing of the comment's dataset is done using regular expressions and sentence tokenizer of sklearn library.
- The words in the dataset are converted into one hot encoded vector using pre-trained word2vec word embedding as it contains 300 dimensional vectors for 3 million words.
- LSTM model provides better average F1 score when implemented on test dataset out of the 3 deep learning models i.e. LSTM, GRU and CNN.
- GUI is implemented using PyQt library to test the LSTM model on user input comments.

7.2 Suggestions for future work

- To incorporate ensemble approach to combine the models to get even better results.
- To develop a plugin for web browsers to incorporate toxic comments identification.

References

1. Spiros, V. Georgakopoulos; Sotiris, K. Tasoulis; Aristidis, G. Vrahatis; Vassilis, P. Plagianakos. (2018) Convolutional Neural Networks for Toxic Comment Classification, Available at https://dl.acm.org/doi/abs/10.1145/3200947.3208069
2. Li, Siyuan. And Wu, Yingnian. (2018) Application of Recurrent Neural Networks in Toxic Comment Classification, Available at https://escholarship.org/uc/item/5f87h061
3. Dhruvil Karani. (2018) Introduction to Word Embedding and Word2Vec, from https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa
4. Kaggle Toxic Comment Classification Challenge, https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge
5. Dataset for training and testing, https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data
6. The complete PyQt5 tutorial – Create GUI applications with Python, https://www.learnpyqt.com/
7. Michael Phi. (2018) Illustrated Guide to LSTM's and GRU's: A step by step explanation, https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
8. Abhishek Perambai. (2019) A deep dive into the world of gated Recurrent Neural Networks: LSTM and GRU, https://medium.com/analytics-vidhya/lstm-and-gru-a-step-further-into-the-world-of-gated-rnns-99d07dac6b91
9. Purva Huilgol. (2019) Accuracy vs. F1-Score, https://medium.com/analytics-vidhya/accuracy-vs-f1-score-6258237beca2

10. Iris Fu. (2019) Deep Learning for Natural Language Processing: Part 2 – RNN, https://medium.com/gumgum-tech/deep-learning-for-natural-language-processing-part-2-rnn-ed2e7bd4d017
11. David, S. Batista. (2018) Convolutional Neural Networks for Text Classification, http://www.davidsbatista.net/blog/2018/03/31/SentenceClassificationConvNets/
12. Colah. (2015) Understanding LSTM Networks, http://colah.github.io/posts/2015-08-Understanding-LSTMs/
13. Simeon Kostadinov. (2017) Understanding GRU Networks, https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be