Toxicity Detection in the Context of Social Media Discussions

Alokparna Bandyopadhyay (gq6225)
Jaydeb Sarker (gy3312)

A. Overview of the Problem

With the increased use of social networking over the past few decades, a flood of information is produced in a daily basis through internet arising from the online interactive communications among users, like chat messages, comments, etc. While this situation contributes significantly to the quality of human life, unfortunately it involves enormous dangers since online texts with high toxicity can cause personal attacks, online harassment, and bullying behaviors. A recent survey[1] finds that approximately four out of ten people in United States have personally faced online harassment. As a result, online platforms tend to struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments. In 2010, Google established the company Jigsaw LLC [2], a company dedicated to stopping online harassment. The goal of this project is to detect and identify toxic comments using both machine learning models and deep learning models. This will help in deterring people from posting toxic comments and thus help to facilitate more courteous and civil discussion on online forums.

Some developers of Google worked on building models like Perspective API [3] to classify the text as toxic or nontoxic. They also make their code publicly available [4]. The models can perform well in some specific tasks for detecting toxicity but makes some misclassifications. For that reason, we work on some machine learning and deep neural network models to improve the accuracy in toxic comment classification. We analyze those models and propose the best model for toxic content classification. Our goal is not only to improve the accuracy, but we have proposed our model to apply other domains for toxic content classification.

[1]https://www.pewresearch.org/internet/2017/07/11/online-harassment-2017/
[2]https://www.jigsawllc.com/
[3]https://www.perspectiveapi.com/
[4]https://conversationai.github.io/

A.1 Literature Review

Machine learning for detecting toxic comments has been a significant focus in Natural Language Processing research over the past few years. This is in part due to the availability of large corpora of online social interactions. Google Jigsaw [5] published two Kaggle competitions which have allowed researchers to gain access to datasets with 2.5 million training examples of toxic comments. In terms of methods, most research takes a text classification approach similar to sentiment analysis and spam detection.

Aroyo et al. from Google and Jigsaw studied the topic of toxicity in online conversations by addressing the problems of subjectivity, bias, and ambiguity inherent in this task [1]. They analyzed the characteristics of subjective assessment tasks (e.g. relevance judgment, toxicity judgment, sentiment assessment, etc). Whether something is perceived as relevant or as toxic can be influenced by almost infinite amounts of prior or current context, e.g. culture, background, experiences, education, etc. They surveyed recent works that tried to understand this phenomenon and outlined a number of open questions and challenges which shape the research perspectives in this multi-disciplinary field.

In 2018, Georgakopoulos et al. used a deep learning approach to discover toxic comments in a large pool of Wikipedia's talk page comments [2]. They used word representations and Convolutional Neural Networks (CNNs) for the toxicity classification and compared their results with the traditional bag-of-words approach for text analysis, combined with a selection of algorithms proven to be very effective in text classification. Their research proved that the CNN approach gives much better performance than traditional text classifiers.

In 2019, Vaidya et al. compared multiple classifiers with the specific focus of reducing unintended model bias within online conversations [3]. They proposed three multi-task learning models that outperform other base models at mitigating unintended bias and distinguishing between non-toxic and toxic comments. In addition, they also implemented an attention mechanism in one of their multi-task learning models with the intention of capturing hidden state dependencies.

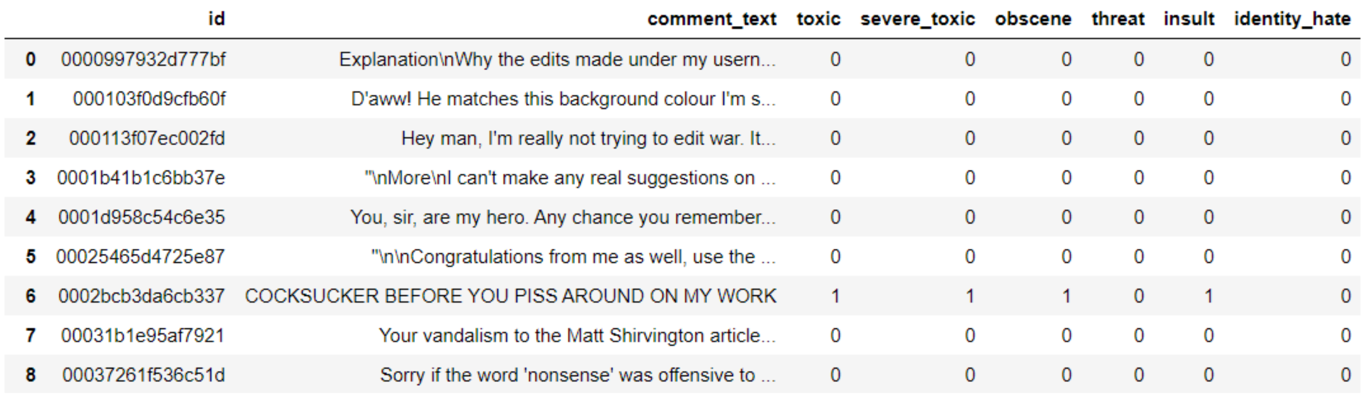
In 2020, Salminen et al. undertook the development of a cross-platform online hate classifier and experimented with various machine learning models like Logistic Regression, Naïve Bayes, Support-Vector Machines, XGBoost and Neural Networks [4]. They found the best performance with XGBoost as a classifier and BERT (Bidirectional Encoder Representations from Transformers) [5] features as the most impactful representation of hateful social media comments.

[5] `https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge`

In 2020, Kurita et al. showed that the performance of the state-of-the-art toxic content classifiers can be easily degraded in a model agnostic manner by using a background corpus of toxicity and introducing character-level perturbations as well as distractors [6] . To address these vulnerabilities, they proposed the Contextual Denoising Autoencoder (CDAE), a novel method for learning robust representation, which uses character-level and contextual information to de-noise obfuscated tokens. Aken et al. (2018) mentioned the challenges of toxic content classifciation [7]. They also perfomed different deep neural network model on two datasets and analyze those models.

B. Dataset Overview

In 2010 Google established the company Jigsaw LLC, a company dedicated to stopping online harassment. The dataset we used in this project is the Jigsaw/Conversation AI dataset provided for the Kaggle Toxic Comment Classification Challenge[6]. The dataset contains a large number of Wikipedia comments which have been labeled by human raters for toxic behavior. The dataset's only feature is the online comments and these comments are classified as one or more of the six classes, namely, toxic, severe_toxic, obscene, threat, insult, and identity_hate.The training data has 1,59,571 comments, whereas the testing data has 1,53,164 comments. The training data is labeled by human raters; whereas testing data is not labeled. Testing dataset only contains the comments.



	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0
5	00025465d4725e87	"\n\nCongratulations from me as well, use the ...	0	0	0	0	0	0
6	0002bcb3da6cb337	COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK	1	1	1	0	1	0
7	00031b1e95af7921	Your vandalism to the Matt Shirvington article...	0	0	0	0	0	0
8	00037261f536c51d	Sorry if the word 'nonsense' was offensive to ...	0	0	0	0	0	0

Figure 1: Sample Dataset

Figure 1 shows a portion of training comments. The value is either 0 or 1; 1 indicates that the comment consists that category and 0 indicates that the comment does not consist

[6] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data

on that category. The five categories severe_toxic, obscene, threat, insult, and identity_hate are under the big umbrella of toxicity. For that reason, we are focusing on toxic values and working on toxic content classification.

The goal of this project is to build a classifier that can identify toxic comments. We have used 80% of the training dataset (1,27,656 comments) for training our models and 20% for validating the results (31,915 comments) in conventional machine learning models. For deep neural networks models, the training size is increased up to 90%-95%.

C. Methods

We are working on a sequence of steps to build our models. We have summarized our research work in the flow diagram of Figure 2.

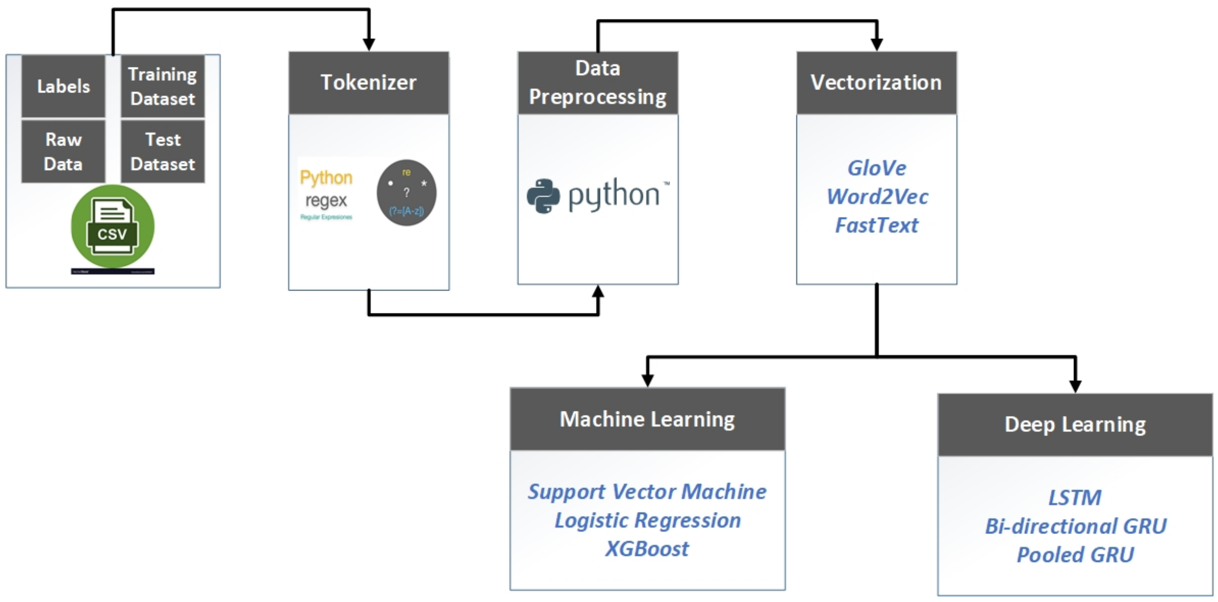


Figure 2: Model Pipeline

In Figure 2, we can see our work strategy for building classifiers. We define our process as followings:

1. Import the labeled Dataset

2. Tokenize the Dataset

3. Data Prepossessing

4. Vectorization
5. Classification Models
    - Machine Learning Models
    - Deep Neural Network Models

C.1 Import the labeled Dataset

In each model, we have imported dataset using python. The dataset is on .csv format.

C.2 Tokenize the Dataset

We use text.Tokenizer from keras model in python. The code snippet is given below:

```
tok=text.Tokenizer(num_words=max_features, lower=True)
tok.fit_on_texts(list(X_train)+list(X_test))
```

C.3 Data Pre-processing

We have done several methods to preprocess the comments before passing to the training model. First, we have done the text to sequences. Secondly, we have done padding of each text. The code snippet is attached below:

```
X_train=tok.texts_to_sequences(X_train)
X_test=tok.texts_to_sequences(X_test)
x_train=sequence.pad_sequences(X_train,maxlen=maxlen)
x_test=sequence.pad_sequences(X_test,maxlen=maxlen)
```

After completing this processings, we have passed our comment to pretrained word embeddings for vectorization of text.

C.4 Vectorization

In word vector representations, each word is represented by a vector which is concatenated or averaged with other word vectors in a context to form a resulting vector which is used to predict other words in the same context. These vectors allow capture word analogies, semantic associations and other hidden information about a language. In previous research, word vector representations have proved to boost the efficiency and accuracy of classification models. However, inconsistent performances are observed in some application contexts [8].

In this project, we explore three popular embedding models, namely, Word2Vec [9], GloVe [10] and FastText [11], combined with various classification models and tried to identify which combination of models work based for toxicity detection.

Here is a brief summary on how the models differ in principles.

Word2Vec was developed by Google in 2013 [9]. It is a group of related models based on two-layer neural networks that are trained to reconstruct linguistic contexts of words. Two model architectures can be used: continuous bag-of-words (CBOW) or continuous skip-gram (SG). In CBOW architecture, the model predicts the current word from a window of surrounding context words. As in other bag-of-words approaches, the order of context words does not influence prediction. In the continuous SG architecture, the model uses the current word to define the surrounding window of context words. The SG architecture weights nearby context words more heavily than more distant context words.

GloVe (Global Vectors for Word Representation) was developed in 2014 by the Stanford University [10]. It allows the user to obtain word vector representations by mapping words into a meaningful space where the distance between words is related to semantic similarity. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

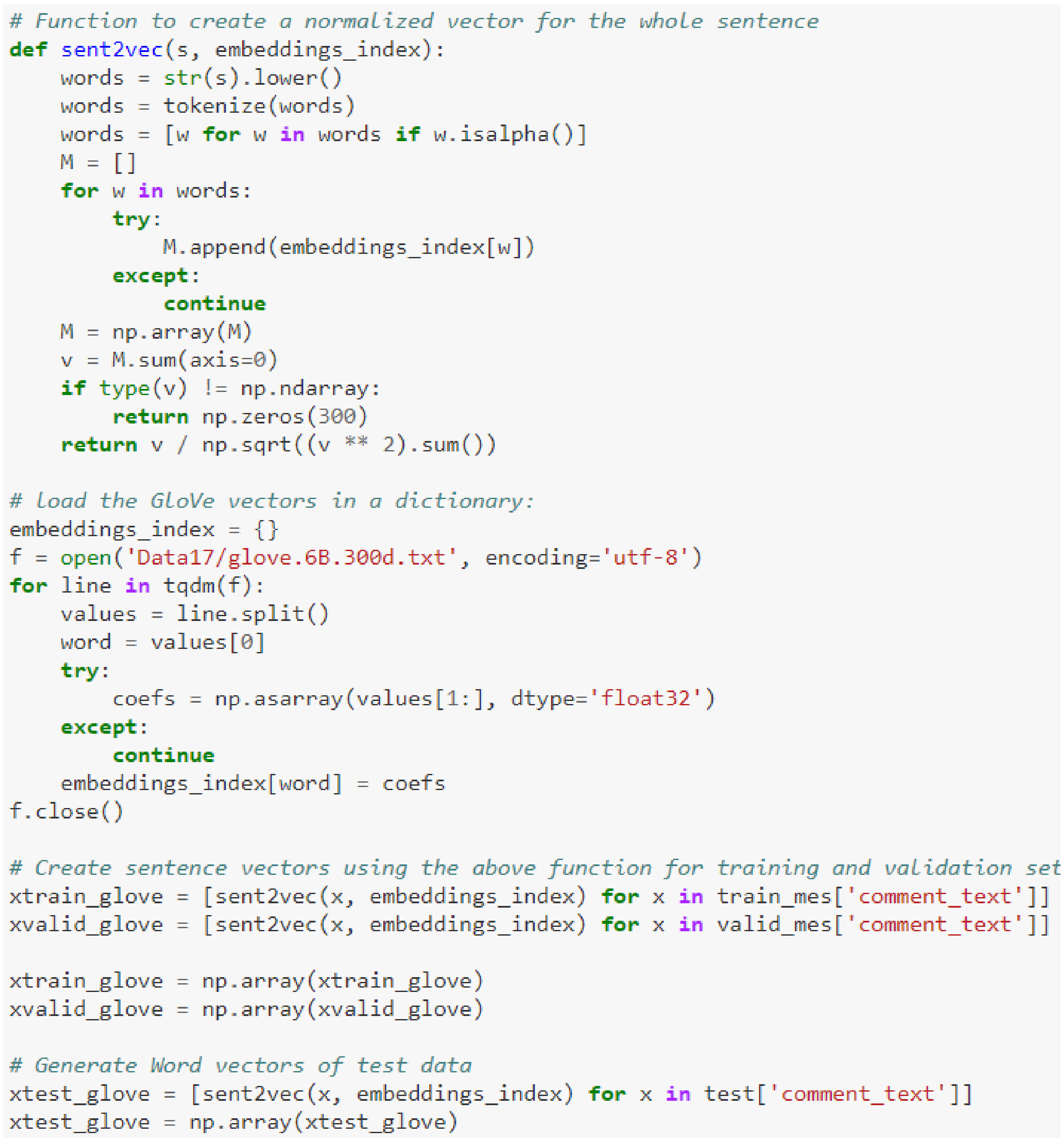
FastText is an extension of Word2Vec proposed by Facebook in 2017 [11]. It is based on the SG model, where each word is represented as a bag of character n-grams. A vector representation is associated to each character n-gram and words are represented as the sum of these vector representations. Pre-trained word embeddings on large training sets are publicly available, such as those produced for Word2Vec, GloVe or Wiki word vectors for FastText1.

For our project, we took the following steps for vectorization of user comments in machine learning models:

- Loaded the GloVeWord2VecFastText vectors in a dictionary called embedded_index.
- Defined a function called sent2vec which creates a normalized vector for a whole sentence.

- Created sentence vectors using the sent2vec function for training, validation and testing datasets and fed these vectorized datasets to the machine learning classification models

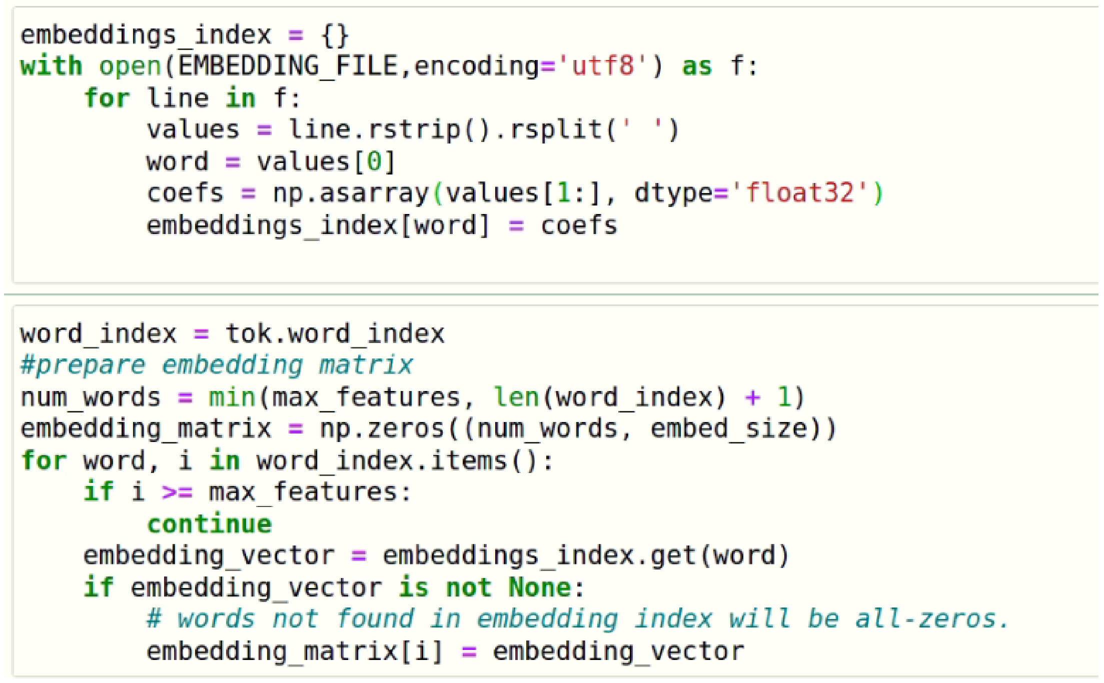
The code snippet is shown in Figure 3:



```python
# Function to create a normalized vector for the whole sentence
def sent2vec(s, embeddings_index):
    words = str(s).lower()
    words = tokenize(words)
    words = [w for w in words if w.isalpha()]
    M = []
    for w in words:
        try:
            M.append(embeddings_index[w])
        except:
            continue
    M = np.array(M)
    v = M.sum(axis=0)
    if type(v) != np.ndarray:
        return np.zeros(300)
    return v / np.sqrt((v ** 2).sum())

# load the GloVe vectors in a dictionary:
embeddings_index = {}
f = open('Data17/glove.6B.300d.txt', encoding='utf-8')
for line in tqdm(f):
    values = line.split()
    word = values[0]
    try:
        coefs = np.asarray(values[1:], dtype='float32')
    except:
        continue
    embeddings_index[word] = coefs
f.close()

# Create sentence vectors using the above function for training and validation set
xtrain_glove = [sent2vec(x, embeddings_index) for x in train_mes['comment_text']]
xvalid_glove = [sent2vec(x, embeddings_index) for x in valid_mes['comment_text']]

xtrain_glove = np.array(xtrain_glove)
xvalid_glove = np.array(xvalid_glove)

# Generate Word vectors of test data
xtest_glove = [sent2vec(x, embeddings_index) for x in test['comment_text']]
xtest_glove = np.array(xtest_glove)
```

Figure 3: Word Embeddings in Machine Learning Code

We have also done the word embeddings for our deep learning models. The code snippet is of embeddings on deep learning models is shown in Figure 4



```
embeddings_index = {}
with open(EMBEDDING_FILE,encoding='utf8') as f:
    for line in f:
        values = line.rstrip().rsplit(' ')
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
```

```
word_index = tok.word_index
#prepare embedding matrix
num_words = min(max_features, len(word_index) + 1)
embedding_matrix = np.zeros((num_words, embed_size))
for word, i in word_index.items():
    if i >= max_features:
        continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector
```

Figure 4: Word Embeddings in Deep Learning Models

C.5 Classification Models

We have used three conventional machine learning models and three deep neural network models to test performance. The model architectures are described in next sections.

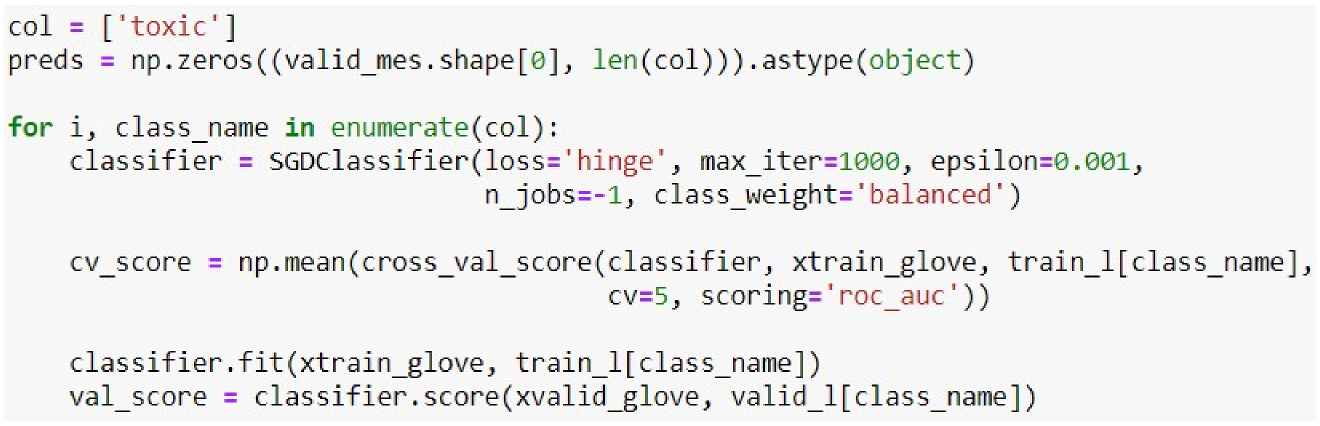
C.5.1 Machine Learning Models

In NLP literature, linear classifiers have always stood as strong baselines for text classification problems. These state-of-the-art models have proved their suitability and their robustness when they are combined with right features. In this research, we have used traditional linear classification models like Support Vector Machine (SVM), Logistic Regression and XGBoost classifiers combined with GloVe, Word2Vec and FastText word embedding approaches. Nine models were explored for the toxicity classification task. Each of the three word embedding models is coupled with each of the three linear classification models and their training (5 fold cross-validation) accuracy and testing accuracy are compared to find the best-performing model.

1. Support Vector Machine A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. SVMs have proved to be helpful in text and hypertext categorization in document classification.
The first approach we took in this project was to implement the SVM algorithm, so as to provide sort of a baseline that we can work with. The steps taken in the implementation are as follows:

- The normalized word embeddings for each sentence in the vectorized training dataset was fed as input into the scikit-learn's SGDClassifier model with 'Stochastic Gradient Descent' as optimizer, class_weight as 'balanced' and 'hinge' as loss function, to make sure that the classifier works as a Support Vector Machine.
- 5-fold cross-validation technique was used to determine the training accuracy score.
- After fitting the vectorized training dataset into the SGDClassifier model, we calculated the validation accuracy using the vectorized validation dataset.

We can see the code snippet of SVM model using 'hinge' loss in Figure 5:



```
col = ['toxic']
preds = np.zeros((valid_mes.shape[0], len(col))).astype(object)

for i, class_name in enumerate(col):
    classifier = SGDClassifier(loss='hinge', max_iter=1000, epsilon=0.001,
                               n_jobs=-1, class_weight='balanced')

    cv_score = np.mean(cross_val_score(classifier, xtrain_glove, train_l[class_name],
                                       cv=5, scoring='roc_auc'))

    classifier.fit(xtrain_glove, train_l[class_name])
    val_score = classifier.score(xvalid_glove, valid_l[class_name])
```

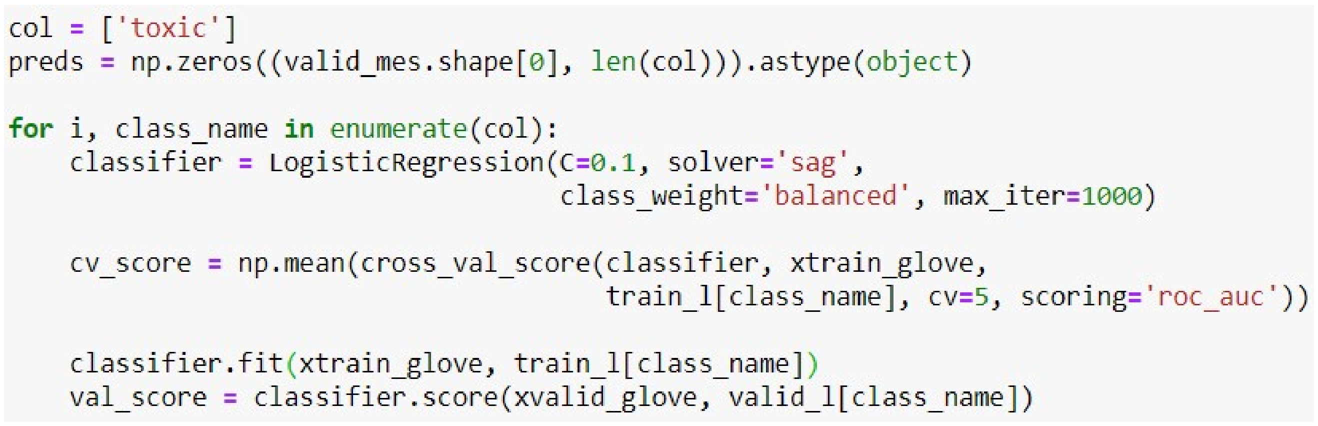
Figure 5: SMV Code Snippet

Figure 5 shows the code snippet for our SVM model with GloVe embeddings. This same procedure was repeated for the other two types of word embedding models used in this research, i.e. Word2Vec and FastText.

2. Logistic Regression Logistic Regression is another very commonly used supervised machine learning model for two-group classification problems and has been widely used

for text classification and sentiment analysis in the literature. Next, we implemented the Logistic Regression Classifier to compare its performance with SVM and see if it works better for toxicity classification.
The steps taken in the implementation are similar to the SVM implementation and are as follows:

- The normalized word embeddings for each sentence in the vectorized training dataset was fed as input into the scikit-learn's LogisticRegression classification model with 'Stochastic Average Gradient descent (sag)' as optimizer and class_weight as 'balanced'.
- 5-fold cross-validation technique was used to determine the training accuracy score.
- After fitting the vectorized training dataset into the LogisticRegression classifier, we calculated the validation accuracy using the vectorized validation dataset.



```
col = ['toxic']
preds = np.zeros((valid_mes.shape[0], len(col))).astype(object)

for i, class_name in enumerate(col):
    classifier = LogisticRegression(C=0.1, solver='sag',
                                    class_weight='balanced', max_iter=1000)

    cv_score = np.mean(cross_val_score(classifier, xtrain_glove,
                                       train_l[class_name], cv=5, scoring='roc_auc'))

    classifier.fit(xtrain_glove, train_l[class_name])
    val_score = classifier.score(xvalid_glove, valid_l[class_name])
```

Figure 6: LR Code Snippet

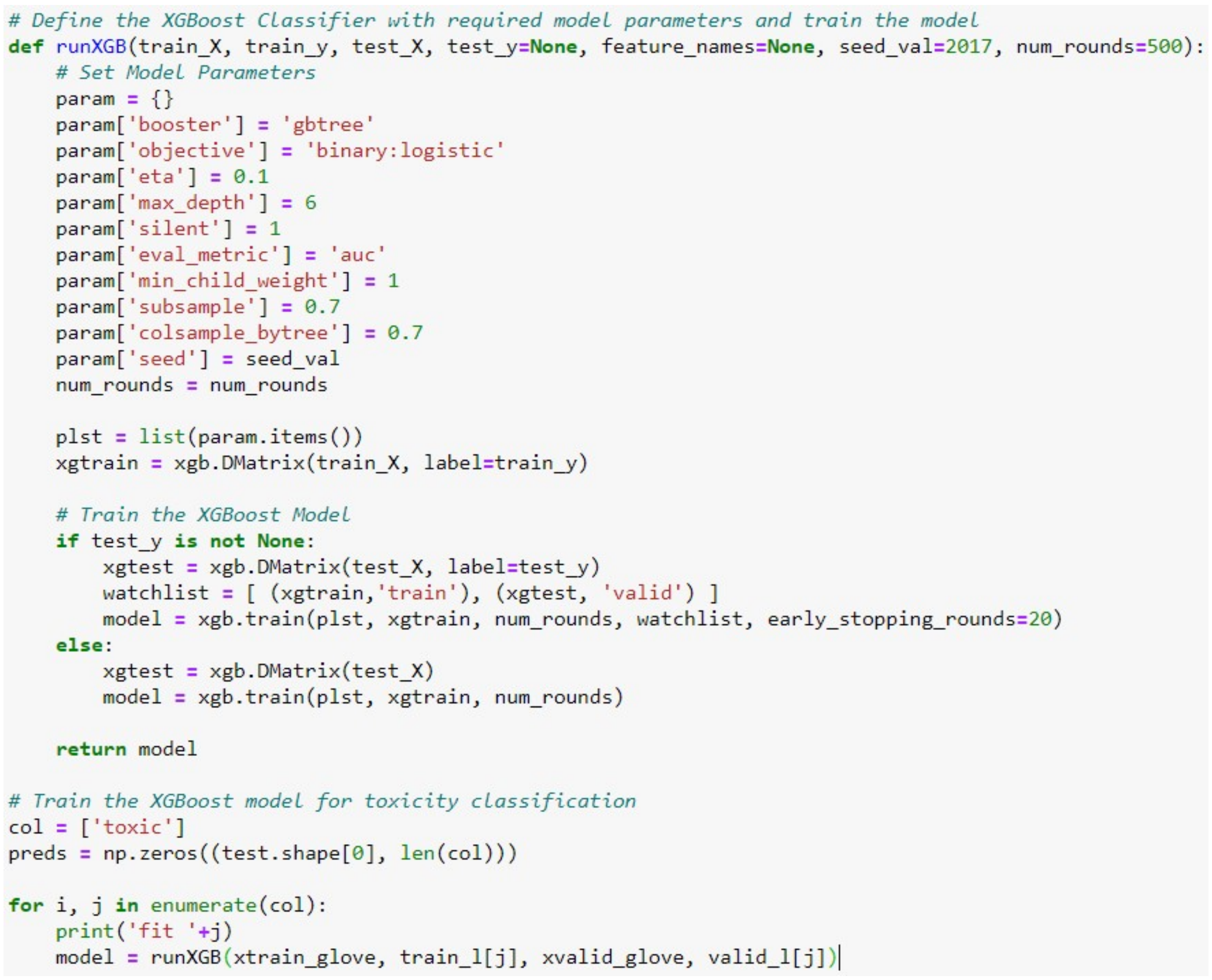
Figure 6 shows the code snippet for our LR model with GloVe embeddings. This same procedure was repeated for the other two types of word embedding models used in this research, i.e. Word2Vec and FastText.

3. XGBoost XGBoost (XGB) stands for eXtreme Gradient Boosting and is an implementation of gradient boosting machines that pushes the limits of computing power for boosted trees algorithms as it was built and developed for the sole purpose of model performance and computational speed. It is one of the most powerful available classifiers and has proved to be highly efficient in document classification over the literature. XGBoost

offers several advanced features for model tuning, computing environments and algorithm enhancement. It is capable of performing the three main forms of gradient boosting (Gradient Boosting (GB), Stochastic GB and Regularized GB) and it is robust enough to support fine tuning and addition of regularization parameters.
We have decided to use XGBoost for this project and evaluate its performance with respect to our baseline models SVM and LR. The steps taken in the implementation of XGB are as follows:

- Before running XGBoost, we must set three types of parameters: general parameters, booster parameters and task parameters. We created a python function for XGBoost training in which, we set the important parameters as below:
  - 'booster' is set to gbtree in order to use treebased models for gradient boosting.
  - 'eta' or the learning_rate is set to 0.1.
  - 'max_depth' is set to 6. Increasing this value will make the model more complex and more likely to overfit.
  - 'min_child_weight' is set to 1. This is the minimum sum of instance weight (hessian) needed in a child node to continue its partitioning.
  - 'subsample' (subsample ratio of the training instances) is set to 0.7.
  - 'colsample_bytree' (subsample ratio of columns when constructing each tree) is set to 0.7.
  - 'objective' (learning objective) is set to binary: logistic, which is logistic regression for binary classification with output probability
- The normalized word embeddings for each sentence in the vectorized training dataset (for model training) and validation dataset (for model testing) was fed as input into the XGBoost classification model with number of rounds as 500 and early stopping rounds as 20. When the validation accuracy does not change considerably for 20 rounds, the XGB model stops learning.

Figure 7 shows the code snippet for our XGBoost model with GloVe embeddings. This same procedure was repeated for the other two types of word embedding models used in this research, i.e. Word2Vec and FastText.



```python
# Define the XGBoost Classifier with required model parameters and train the model
def runXGB(train_X, train_y, test_X, test_y=None, feature_names=None, seed_val=2017, num_rounds=500):
    # Set Model Parameters
    param = {}
    param['booster'] = 'gbtree'
    param['objective'] = 'binary:logistic'
    param['eta'] = 0.1
    param['max_depth'] = 6
    param['silent'] = 1
    param['eval_metric'] = 'auc'
    param['min_child_weight'] = 1
    param['subsample'] = 0.7
    param['colsample_bytree'] = 0.7
    param['seed'] = seed_val
    num_rounds = num_rounds

    plst = list(param.items())
    xgtrain = xgb.DMatrix(train_X, label=train_y)

    # Train the XGBoost Model
    if test_y is not None:
        xgtest = xgb.DMatrix(test_X, label=test_y)
        watchlist = [ (xgtrain,'train'), (xgtest, 'valid') ]
        model = xgb.train(plst, xgtrain, num_rounds, watchlist, early_stopping_rounds=20)
    else:
        xgtest = xgb.DMatrix(test_X)
        model = xgb.train(plst, xgtrain, num_rounds)

    return model

# Train the XGBoost model for toxicity classification
col = ['toxic']
preds = np.zeros((test.shape[0], len(col)))

for i, j in enumerate(col):
    print('fit '+j)
    model = runXGB(xtrain_glove, train_l[j], xvalid_glove, valid_l[j])
```

Figure 7: XGBoost Model Snippet

C.5.2 Deep Neural Network Models

We have used three deep neural network models to perform our tasks. The models are:

1. Long Short Term Memory (LSTM) with Dropout
2. Bidirectional Gated Recurrent Unit (GRU)
3. Pooled Gated Recurrent Unit (GRU)

1. Long Short Term Memory (LSTM) with Dropout To reduce the problem with long term dependencies in Recurrent Neural Network (RNN), Long Short Term Memory (LSTM) was proposed by Sepp Hochreiter and Jürgen Schmidhuber [12]. LSTM consists of 3 units: Input Gate, Forget Gate and Output Gate.

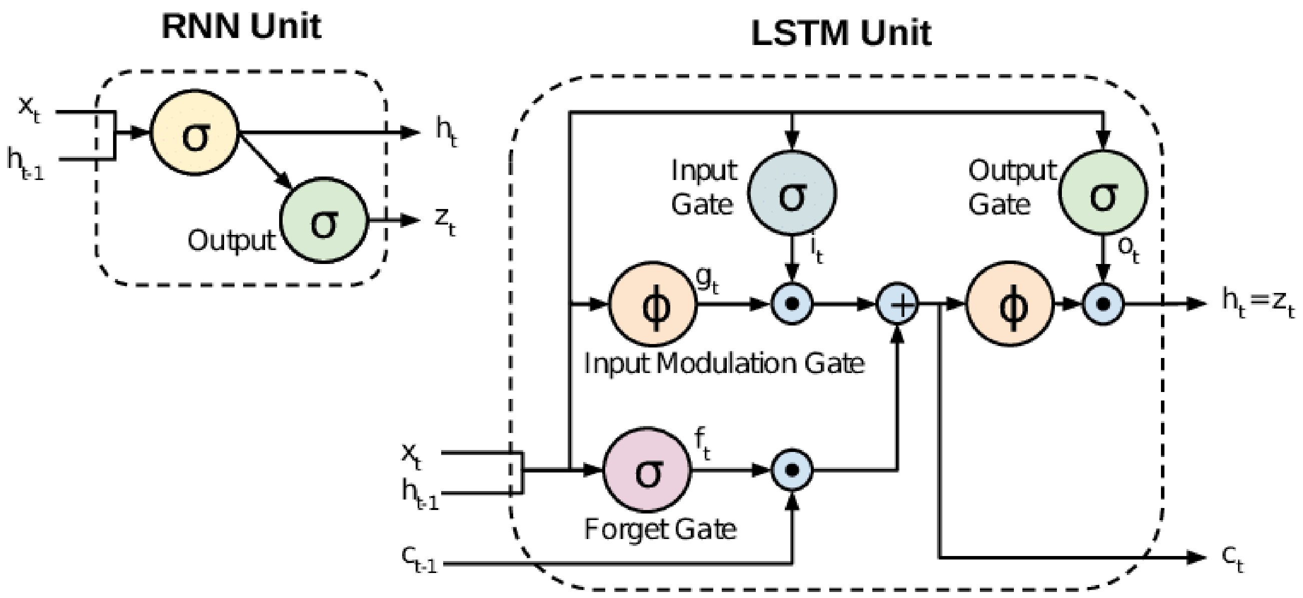


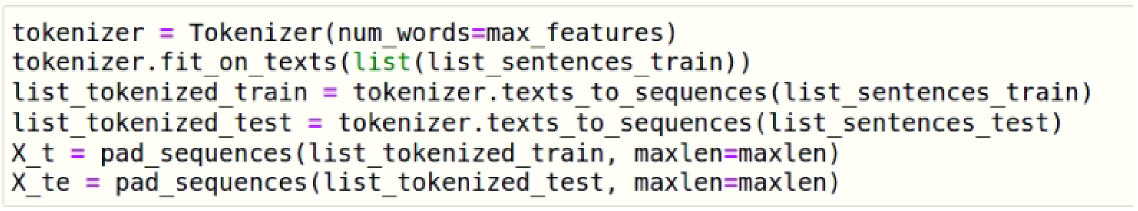
Figure 8: RNN and LSTM [`https://rb.gy/ql43x9`]

In Figure 8, we can see the comparison of LSTM with RNN. LSTM outperfoms RNN in long sequence of text. As we have done the text classification, LSTM is a better choice than RNN. We have classified the text as 'toxic' or 'nontoxic' where LSTM can detect long sequences for each comment's category.

We have used glove.840B.300d for word embedding in this model. In the model, we have fit the parameter as like

embed_size = 50 (how big is each word vector)
max_features = 20000 (how many unique words to use) (i.e num rows in embedding vector)
maxlen = 100

We prepossessed the data with padding and tokenization using Keras in Tensorflow. The code snippet is attached in Figure 9:



```
tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(list(list_sentences_train))
list_tokenized_train = tokenizer.texts_to_sequences(list_sentences_train)
list_tokenized_test = tokenizer.texts_to_sequences(list_sentences_test)
X_t = pad_sequences(list_tokenized_train, maxlen=maxlen)
X_te = pad_sequences(list_tokenized_test, maxlen=maxlen)
```

Figure 9: LSTM Preprocessing

Now, we build our LSTM model with a dropout layer in figure 10. We put the dropout layer to reduce the overfitting problem. In this model, we set one embedding layer, then bidirectional LSTM, two pooling layer before dropout layer.

We have used
loss='binary_crossentropy'
optimizer='adam'
metrics=['accuracy'].
The purpose of using 'binary_crossentropy' as loss function is that we have to classify two categories. In this scenario 'binary_crossentropy' performs well.

2. Bidirectional Gated Recurrent Unit (GRU) Kyunghyun Cho et al. (2014) introduced the Gated Recurrent Unit (GRU) that is similar with Long Short Term Memory (LSTM) with forget gate [13]. GRU has lower parameter than LSTM and easy to compute. GRU consists two gates: reset gate and update gate. In figure 11, we can look at the architecture of GRU having two gates. It has also a candidate hidden state and finally the output state produces the output after each input.

In this model, we have used glove.840B.300d for word embedding. Then we put

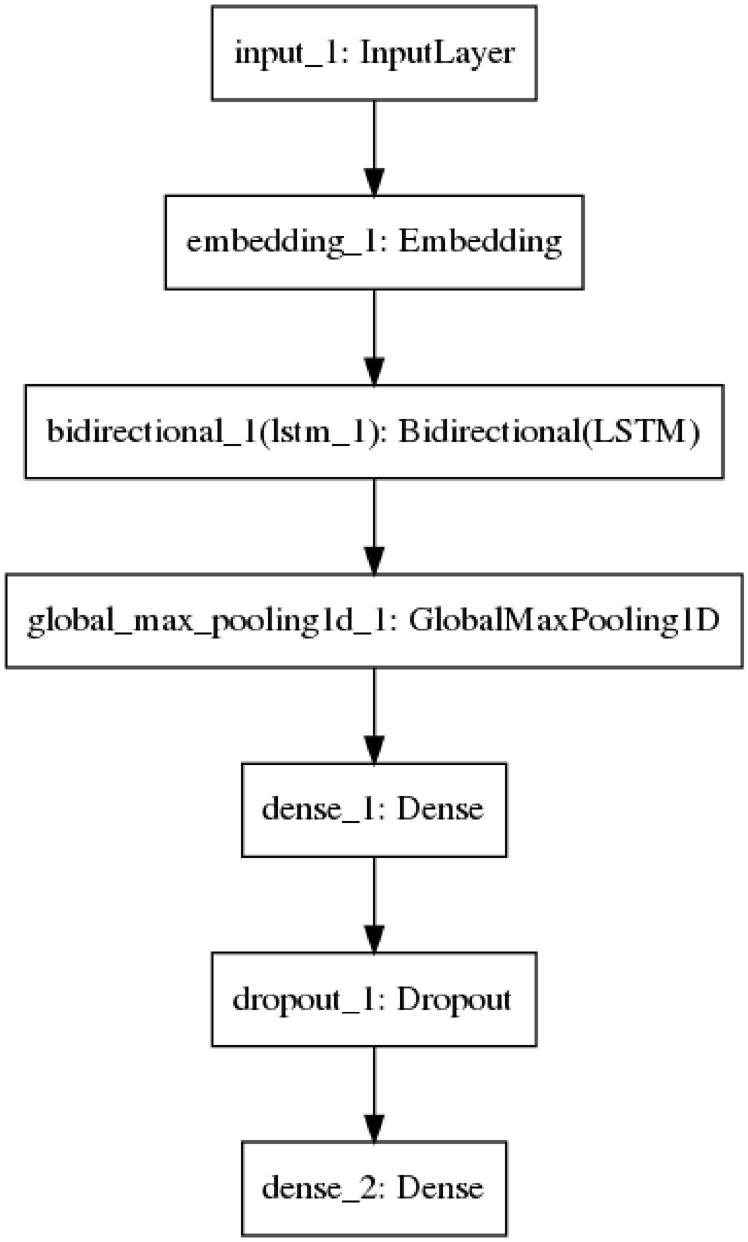


Figure 10: BiDirectional LSTM with dropout Layer

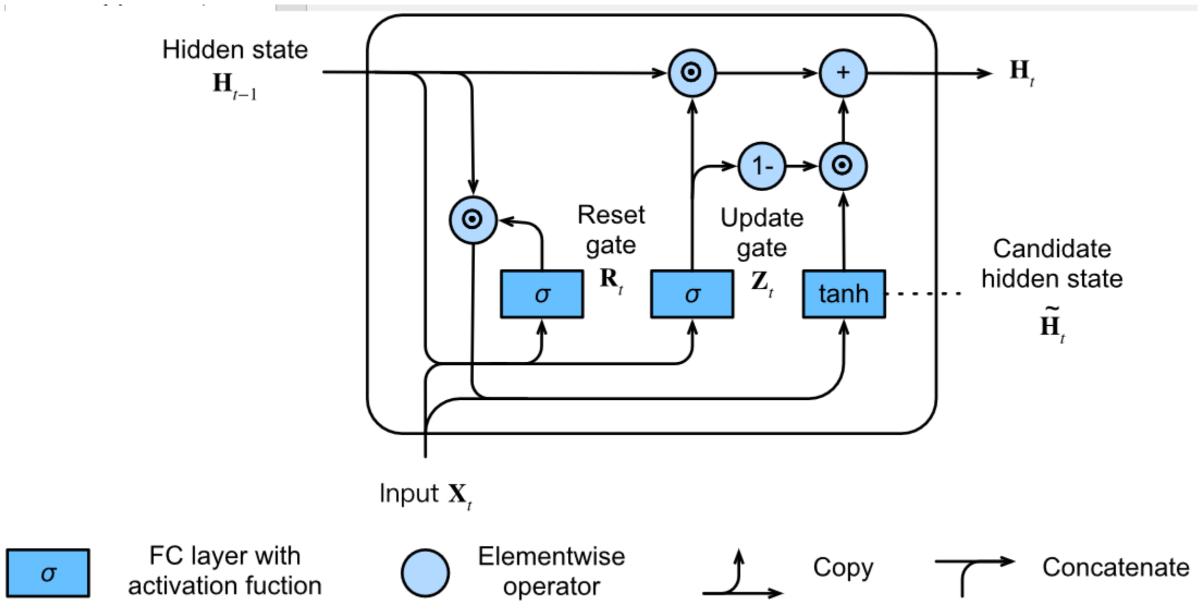

Hidden state $\mathbf{H}_{t-1}$
$\mathbf{H}_t$
1-
Reset gate $\mathbf{R}_t$
Update gate $\mathbf{Z}_t$
σ
σ
tanh
Candidate hidden state $\tilde{\mathbf{H}}_t$
Input $\mathbf{X}_t$
FC layer with activation fuction
Elementwise operator
Copy
Concatenate

Figure 11: Architecture of GRU [`https://rb.gy/nxiynr`]

max_features=100000 (maximumfeatures)
maxlen=150 (maximum length)
embed_size=300 (size of embedding)

We have done our preprosessing as followings:
First, we did tokenization and padding of each comments and then convert it vector by using word embeddings. Our Bi-Directional GRU architecture is depicted in figure 12. In this Bidirectional GRU model, we have set two pooling layers (average pooling and max pooling) layer after bidirectional GRU layer. Then we set dense layer. The batch size and number of epochs are shown in figure 13. We have set binarycrossentropy as loss function. We have done our experiment with 90% labled data as training and 10% data as testing.

C.5.3 Pooled Gated Recurrent Unit (GRU)

Pooled GRU is a simple GRU model with pooling layers. We also add dropout layer for reducing the overfitting problem. We have used FastText 300d as word embeddings for this model. The preprocessing steps are similar to Bidirection GRU model. The model is shown in figure 14:

In the model, we have set the followings:
batch_size = 32
epochs= 2

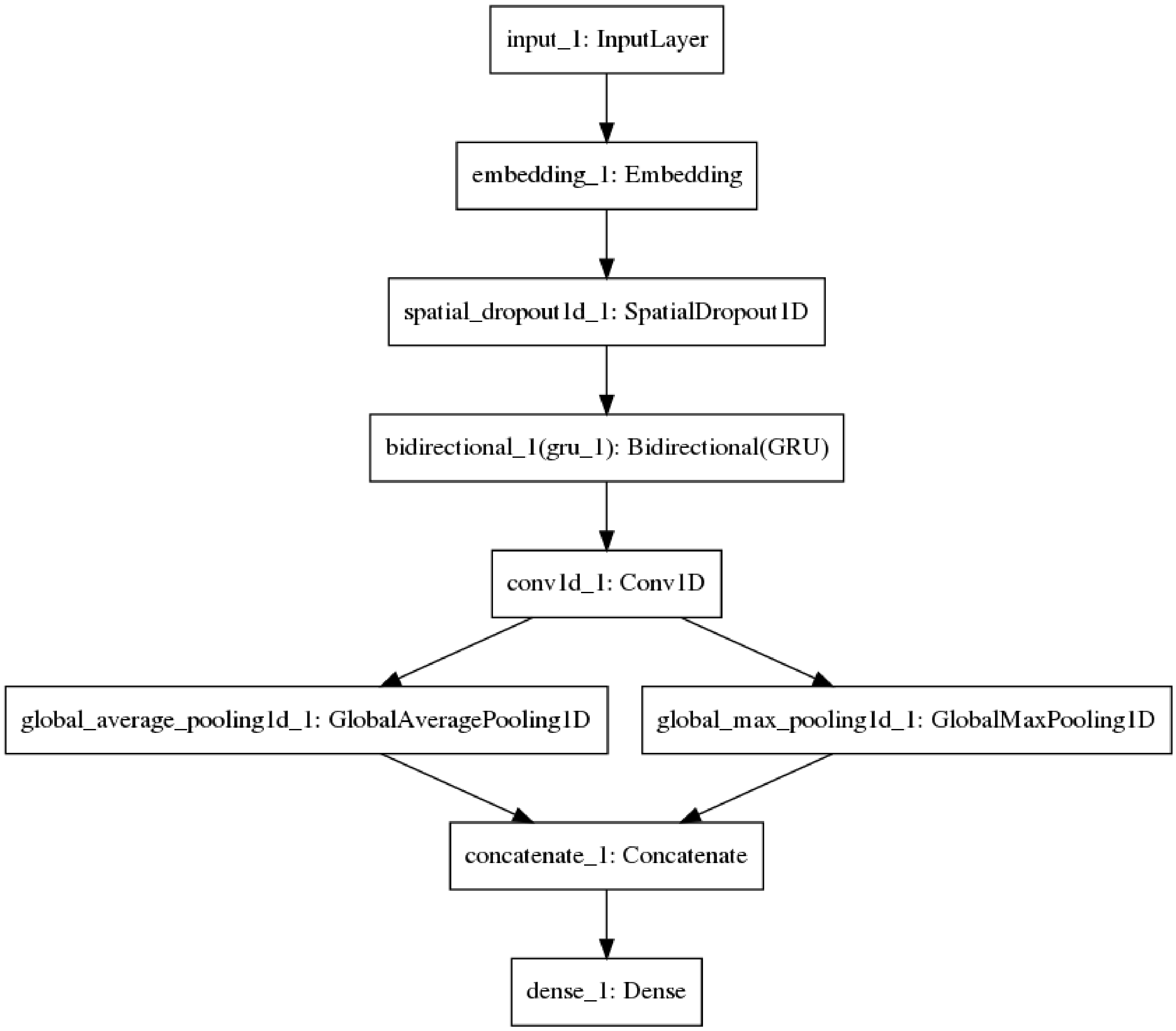
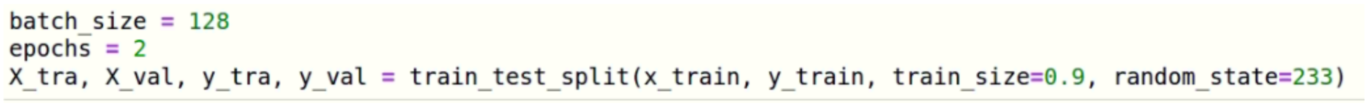

input_1: InputLayer
embedding_1: Embedding
spatial_dropout1d_1: SpatialDropout1D
bidirectional_1(gru_1): Bidirectional(GRU)
conv1d_1: Conv1D
global_average_pooling1d_1: GlobalAveragePooling1D
global_max_pooling1d_1: GlobalMaxPooling1D
concatenate_1: Concatenate
dense_1: Dense

Figure 12: Bidirectional GRU model



```
batch_size = 128
epochs = 2
X_tra, X_val, y_tra, y_val = train_test_split(x_train, y_train, train_size=0.9, random_state=233)
```

Figure 13: Bidirectional GRU model Code Snippet

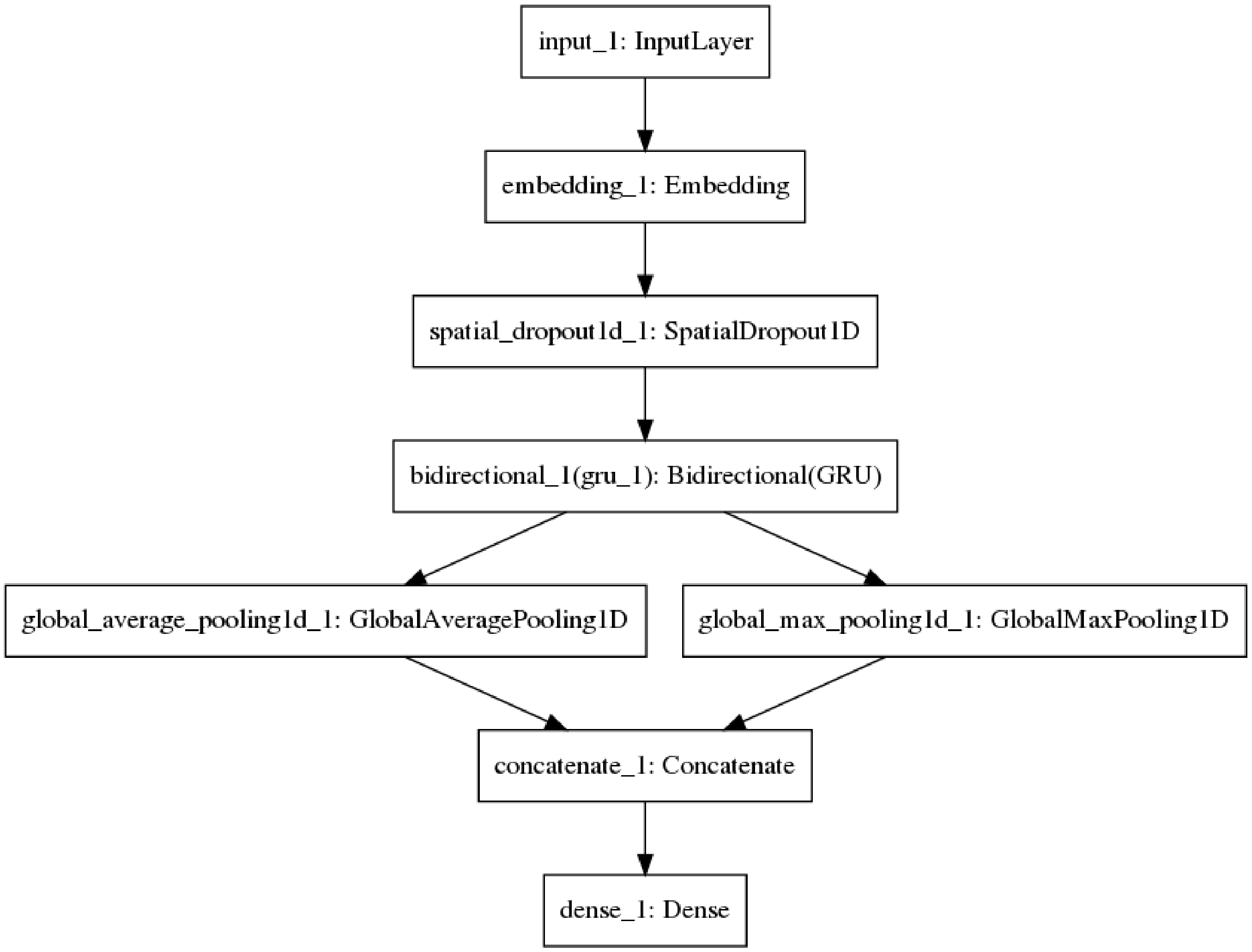

input_1: InputLayer
embedding_1: Embedding
spatial_dropout1d_1: SpatialDropout1D
bidirectional_1(gru_1): Bidirectional(GRU)
global_average_pooling1d_1: GlobalAveragePooling1D
global_max_pooling1d_1: GlobalMaxPooling1D
concatenate_1: Concatenate
dense_1: Dense

Figure 14: Pooled GRU model

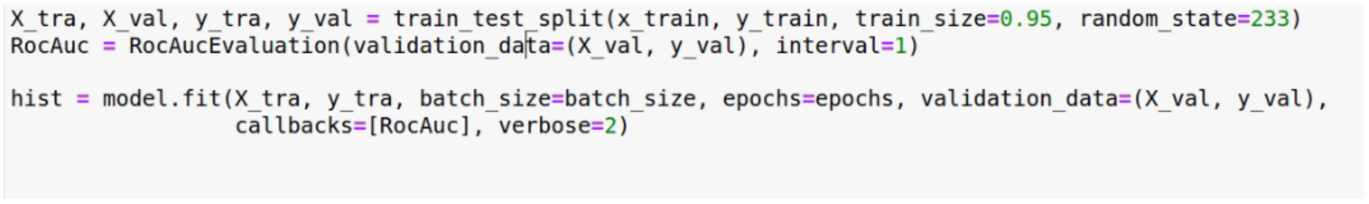

```
X_tra, X_val, y_tra, y_val = train_test_split(x_train, y_train, train_size=0.95, random_state=233)
RocAuc = RocAucEvaluation(validation_data=(X_val, y_val), interval=1)

hist = model.fit(X_tra, y_tra, batch_size=batch_size, epochs=epochs, validation_data=(X_val, y_val),
                 callbacks=[RocAuc], verbose=2)
```

Figure 15: Pooled GRU snippet

We have used the loss and optimizer as follows:
loss=‘binary_crossentropy’
optimizer=‘adam’

We have put 95% of training data and 5% for validate the model. The code for the validation of this Pooled GRU model is shown in figure 15.

D. Analysis Results and Comparison

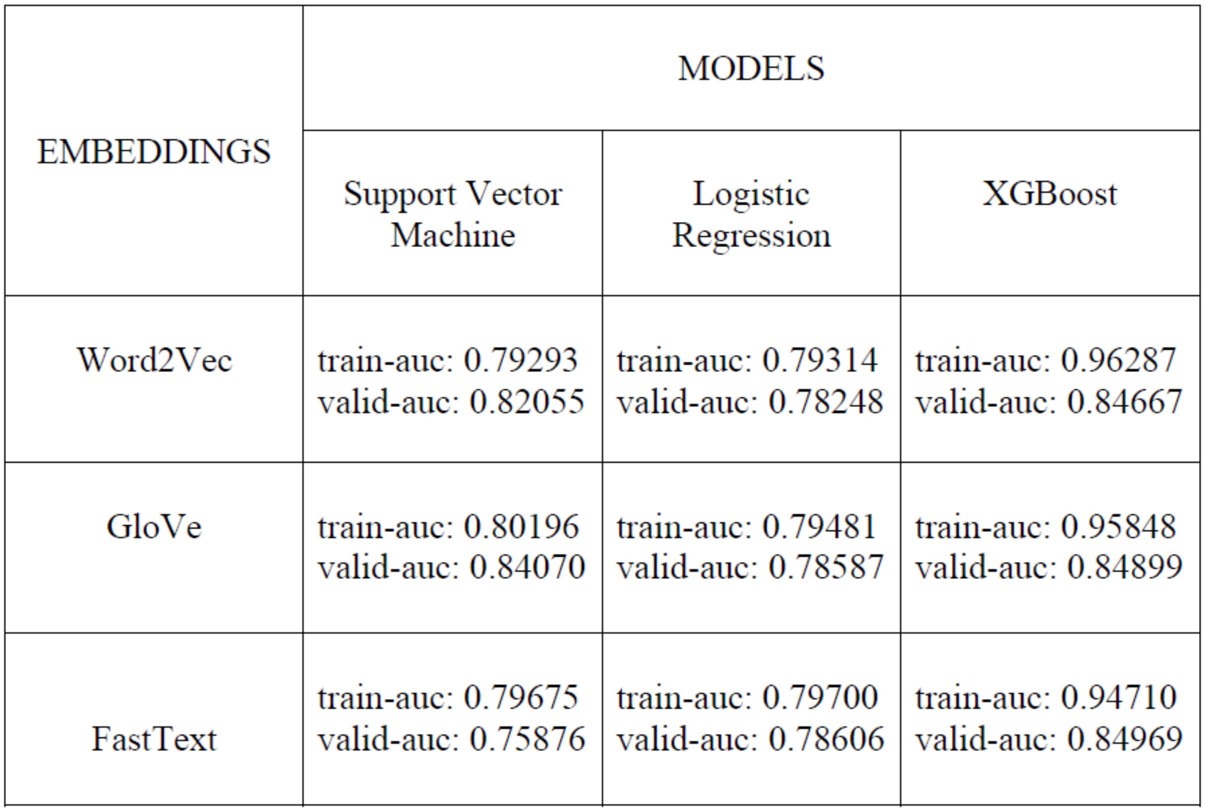
We have done validation on training dataset. The training dataset is labeled. We measure the results in Receiver Operating Characteristic (ROC) curve. The Area of Curve (AUC) is an effective measurement for binary classification. It plots the true positive rate against the false positive rate in various thresholds. So, we compare our models using the AUC values from each models output.

Environment Setup We have done our experiments using following environments:
pyhton=3.7
tensorflow 2.1
Jupyter Notebook
GPU: NVIDIA TITAN RTX

D.1 Machine Learning Models

The performance of three machine learning models with three different word embeddings are shown in Figure16. We have done total 9 experiments for three machine learning model with three different embeddings.

Among of the 9 experiments, we have found the best validation AUC score using XG-Boost (0.84969) using FastText word embeddings. This model can perform well than SVM and Logistic Regression. On the other hand support vector machine gives it’s best validation AUC score using GloVe (0.84070). Logistic Regression does not perform well for this classification models. It gives the best AUC score as 0.7858 using FastText embeddings. Figure 17 shows the best 3 machine learning models auc score with specific embeddings.



EMBEDDINGS	MODELS		
	Support Vector Machine	Logistic Regression	XGBoost
Word2Vec	train-auc: 0.79293 valid-auc: 0.82055	train-auc: 0.79314 valid-auc: 0.78248	train-auc: 0.96287 valid-auc: 0.84667
GloVe	train-auc: 0.80196 valid-auc: 0.84070	train-auc: 0.79481 valid-auc: 0.78587	train-auc: 0.95848 valid-auc: 0.84899
FastText	train-auc: 0.79675 valid-auc: 0.75876	train-auc: 0.79700 valid-auc: 0.78606	train-auc: 0.94710 valid-auc: 0.84969

Figure 16: Machine Learning Model Accuracy

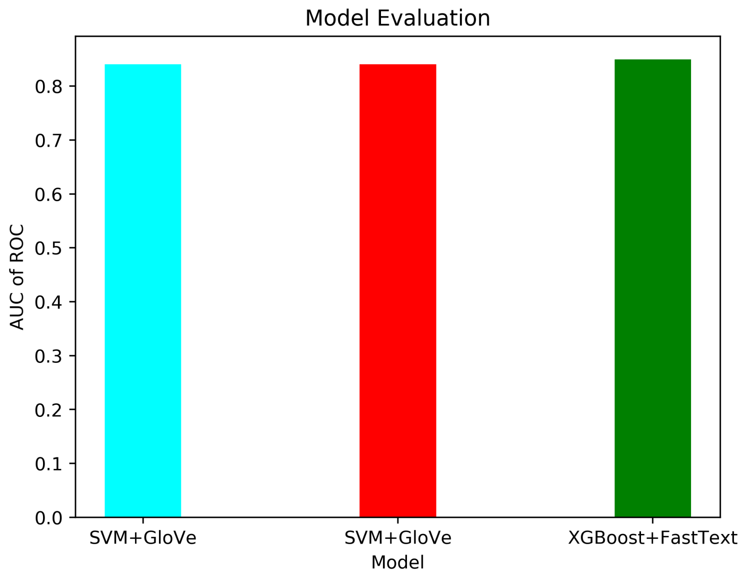


Figure 17: Best Machine Learning Model AUC Score

Table 1: Results of Deep Learning Models

SL	Model	Embeddings	AUC Score	Loss
1	LSTM with Dropout	GloVe 300d	0.9834	0.0453
2	Bidirectional GRU	GloVe 300d	0.97094	0.0537
3	Pooled GRU	FastText 300d	0.986516	0.045

D.2 Deep Learning Models

We have applied three deep learning models and compared the results among them. The prepossessing of our data are described in the methodology section. The deep learning models outperform the auc score than machine learning models.

In Table 1, we can see the performance of our models. We have splitted the train dataset into 90% as training size and 10% as validation size. Then we have found that LSTM gives slightly better accuracy than Bidirectional GRU model. Overall, Pooled GRU gives the best performance with AUC score 0.986516 which is more than 98.65% accurate. On the other hand, Pooled GRU also gives the best result for loss. It minimizes the loss to 0.045 whether Bidirectional GRU gives maximum loss 0.0537.

In Figure 18, we can observe the accuracy comparison of our deep neural network models. We have set dropout layer in each models to prevent the overfitting problems. The amount of dropout is 10%. By using this, we can prevent the overfitting of classification.

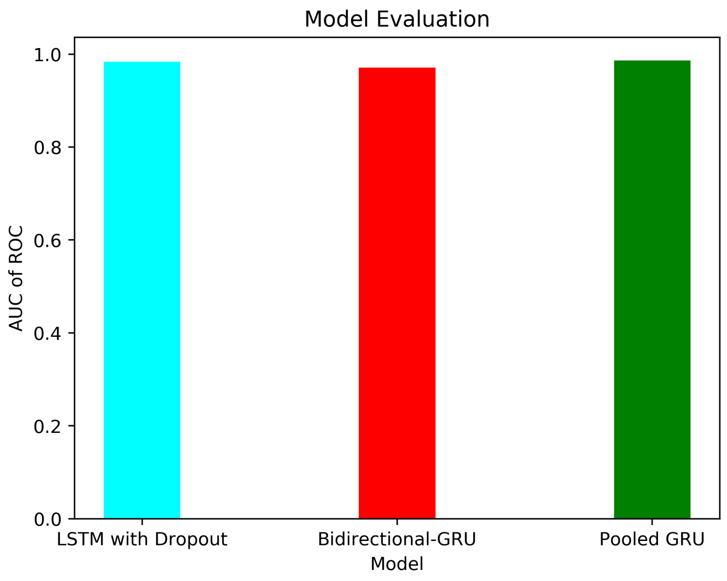


Figure 18: Performance Evaluation

Table 2: Number of Toxic Comments in Test dataset

SL	Model	Number of Toxic Comments
1	XGBoost	9156
2	LSTM with Dropout	32612
3	Bidirectional GRU	33517
4	Pooled GRU	37750

D.3 Result Analysis

In this work, we compare three machine learning models and three deep learning models. Among all the models, Pooled GRU gives the best performance for this specific dataset. For that reason, we are going to propose to use Pooled GRU model to do the classification of toxicity analysis. We also analyzed our results. We have found that traditional machine learning does not give good performance in text classification. Text classification needs long term dependency. Moreover, we use different word embedding techniques. Those techniques help us to improve the accuracy of our models. We have found Pooled GRU gives slightly better result than LSTM because it adds two pooling layers and it uses the FastText as it's word embedding. FastText embeddings provides well dictionary for this toxic content classification.

We have used our models to produce the output for test dataset. The data are not labeled in test dataset. We have produced the score for test dataset. The score is between 0 to 1 for each comment. As the score is not binary, we have fixed the thresold 0.5 or more as toxic score. Then we have compared our models according to the toxic data in test dataset. In Table 2, we can observe that the most number of toxic comments in test dataset are detected by Pooled GRU model. We (two of us) manually checked around 50 comments of them as toxic or nontoxic and found that Pooled GRU provides best accuracy for text dataset.

In Table 2, we put the number of comments in submission.csv file which has toxic score more than 0.5 using different models. We consider one machine learning models and three deep neural network models for this.

We can see the comparison of 4 models for finding toxic comments from test dataset in Figure 19. The total number of test data comments is 1,53,164. It shows that different model gives different number of comments as toxic in unlabled data. Among all, we manually observe that Pooled GRU outperforms other to classify the toxic comments.

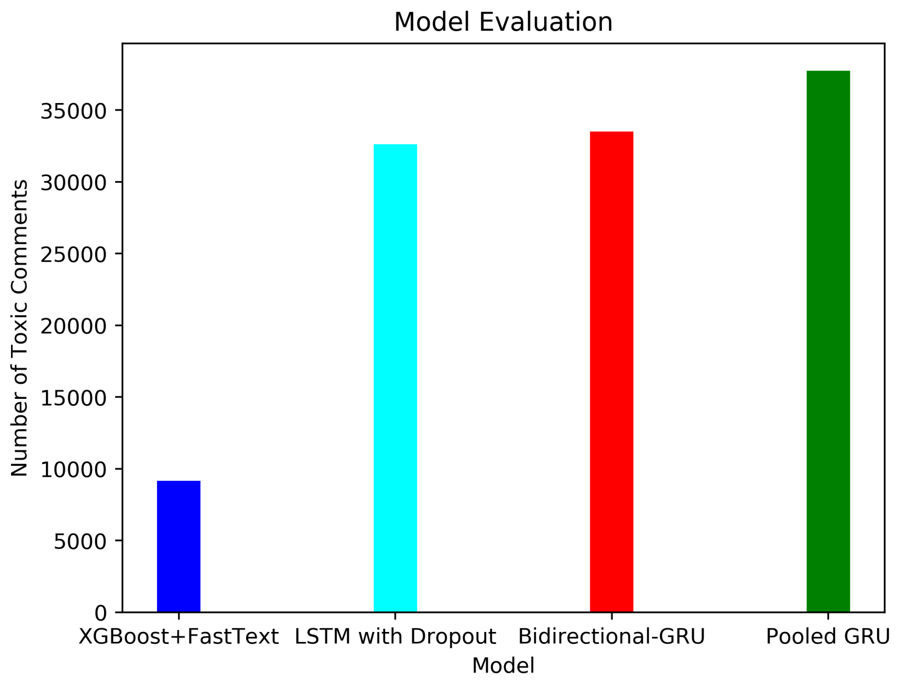


Figure 19: Number of Toxic Comments

D.4 Limitations and Future Work

In every NLP tasks, the classification varies on domain. Sometimes, the meaning of a text differs with respect to its consequences. We have evaluated our models on Jigsaw dataset which is based on online communications. We can not assure that our model can perform well for other domain's dataset. We will evaluate our models on other dataset. To get good accuracy on other domain for toxicity analysis, we may need to modify our models. Moreover, we will have to do some other techniques to prepossess the data. Apart from that, we have a plan to work on character level for finding toxic comments.

E. Conclusion

In this work, we have implemented six different models to perform the classification of toxic comments. We have found that deep neural network models outperform the conventional machine learning models. Moreover, Pooled GRU with FastText embedding gives the best result among 6 models. Before passing the comments to each, we work on different prepossessing to find a good result. We have compared the models by using AUC score.

Moreover, we also validate the models using unlabeled dataset and find out the number of toxic comment on that dataset. We propose to use Pooled GRU model for toxic content classification.

Repository

Go to the link [7] to get our code and output.

Member Contribution

- Alokparna Bandyopadhyay worked on literature review and building conventional machine learning models
- Jaydeb Sarker worked on preprocessing the dataset and building deep neural network models
- Both of them worked on analysis part

References

[1] Lora Aroyo, Lucas Dixon, Nithum Thain, Olivia Redfield, and Rachel Rosen. Crowdsourcing subjective tasks: the case study of understanding toxicity in online discussions. In *Companion Proceedings of The 2019 World Wide Web Conference*, pages 1100–1105, 2019.

[2] Spiros V Georgakopoulos, Sotiris K Tasoulis, Aristidis G Vrahatis, and Vassilis P Plagianakos. Convolutional neural networks for toxic comment classification. In *Proceedings of the 10th Hellenic Conference on Artificial Intelligence*, pages 1–6, 2018.

[3] Ameya Vaidya, Feng Mai, and Yue Ning. Empirical analysis of multi-task learning for reducing model bias in toxic comment detection. *arXiv preprint arXiv:1909.09758*, 2019.

[4] Joni Salminen, Maximilian Hopf, Shammur A Chowdhury, Soon-gyo Jung, Hind Almerekhi, and Bernard J Jansen. Developing an online hate classifier for multiple social media platforms. *Human-centric Computing and Information Sciences*, 10(1):1, 2020.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[7] `https://github.com/alokparna-wsu/Toxicity_Detection_Project`

[6] Keita Kurita, Anna Belova, and Antonios Anastasopoulos. Towards robust toxic content classification. *arXiv preprint arXiv:1912.06872*, 2019.

[7] Betty van Aken, Julian Risch, Ralf Krestel, and Alexander Löser. Challenges for toxic comment classification: An in-depth error analysis. *arXiv preprint arXiv:1809.07572*, 2018.

[8] Anaıs Ollagnier and Hywel Williams. Classification and event identification using word embedding. *neural networks*, 6:7, 2019.

[9] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[10] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[11] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[13] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.