University of Pisa

Human Language Technology Course

Toxic Comment classification Challenge Report

Course 649AA A.A 2021/2022

Dawit Anelay ID:608791 Email: d.anelay@studenti.unipi.it
Yohannis Kifle ID:621821 Email: y.telila@studenti.unipi.it

July 8, 2021

Abstract

This report is an analysis of different model performance on the task of toxic comment classification challenge from Kaggle[1]. The task is multi-label classification task with 6 possible labels. For this task we mainly used Keras, nltk, Scikit-Learn and gensim popular frameworks in the machine learning and natural language processing taks. We experimented with different potential models from simpler models to deep neural network models. With a proper selection of convolutional neural networks(CNN) architecture and proper embedding layer we can remarkably get a good result in a text classification[Yon Kim, 2014]. We also explored the performance of Recurrent neural networks(RNN) in the text classification task[Minaee et al, 2021]. Hybrid model of RNN and CNN is also another architecture we explored in this report. We identified the best performing model based on our metrics. We then deployed the the selected model for the real world use. We built a telegram bot that moderates a group discussion and take appropriate actions for the users who send inappropriate comments in a discussion.

Keywords: Recurrent neural networks, CNN, Embeddings, Bot

I. Introduction

On many of current online platforms it is common activity to collect users feed backs in the form of comments. People express their feelings on what they think of a product, services, videos, articles and on other people's opinion on the platform. If these sections of the platform are not regulated or managed it could turn into a place of harassment, abuse and threat.

In this competition, we're challenged to build a multi-headed model that's capable of detecting different types of of toxicity like threats, obscenity, insults, and identity-based hate.The task of this challenge falls under multi-label prediction since a comment can have multiple labels. We have compared our result from 7 different models to get the best performing model for the challenge. We also tried to explore the effect of using embedding layers for the CNN and RNN architectures we have tried. We experimented with the following models.

1. Binary Relevance Method with SVM classifier..
2. Classifier chain with Logistic Regression classifier
3. Label Powerset with MultinomialNB classifier.

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

4. CNN with different embedding and kernel sizes
5. RNN With different embedding and kernel sizes.
6. Combination of RNN and CNN with different embeddings.

We optimized all of the models using hyper-parameter tuning techniques in machine learning. To train the Hyper parameters of all models two different solutions were chosen: a Grid Search approach was used, defining the ranges of the chose parameters and progressively narrowing the range of values around the best results obtained in the various search; a Randomized Search approach was also used to introduce some stochastic randomness to evade from saddle points and local minimums in the model function that was being tested. Once the models were all evaluated, according to the results obtained, the best ones were chosen. Hyper-parameter tuning some of the model were difficult as they require huge amount of computational power. For those models we analysed the learning curve and tried to control the complexity of the model and select appropriate mode parameters by analysing learning curve.

II. Evaluation Method

We defined 4 metrics to evaluate our models to make sure our model is operating correctly and smoothly. We chose hamming loss, Accuracy, log-loss and F1-score evaluation metrics.
Let D be a multi-label dataset consisting n multi-label examples (x_i, Y_i), 1$\leq$ i $\leq$ n,x$_i$ $\in X, y_i \in Y = \{0,1\}^k$),with a labelset L, $\mid L \mid$=k. Let h be a multi-label classifier and $Z_i = h(x_i) = \{0,1\}^k$ be the set of label memberships predicted by h for the example x_i.

I. Hamming Loss

Hamming Loss(HL): Hamming Loss reports how many times on average, the relevance of an example to a class label is incorrectly predicted.Hamming loss takes into account the prediction error(an incorrect label) and the missing error(a relevant label not predicted),normalized over total number of classes and total number of examples.

$$HammingLoss, HL = \frac{1}{kn}\sum_{i=1}^{n}\sum_{l=1}^{k}[I(l \in Z_i \wedge l \wedge Y_i) + I(l \wedge Z_i \wedge l \in Y_i)] \tag{1}$$

where I is the indicator function . Ideally,we would expect hamming loss,HL=0 which would imply no error;parctically the smaller the value of hamming loss,the better the performance of the learning algorithm.

II. Accuracy

Accuracy(A): Accuracy for each instance is defined as the proportion of the predicted correct labels to the total number (predicted and actual) of labels for that instance . overall accuracy is the average across all instances.

$$Accuracy, A = \frac{1}{n}\sum_{i=1}^{n}\frac{\mid Y_i \cap Z_i \mid}{\mid Y_i \cup Z_i \mid} \tag{2}$$

III. Log Loss

Log Loss quantifies the accuracy of a classifier by penalizing false classifications.

$$LogLoss = -\frac{1}{n}\sum_{i=1}^{n}\sum_{l=1}^{k} y_{ij} \log p_{ij} \tag{3}$$

IV. F1-score

F1-Measure(F):The F1 score is the harmonic mean of precision and recall.

$$F1 = \frac{1}{n}\sum_{i=1}^{n} 2\frac{\mid Y_i \cap Z_i \mid}{\mid Y_i \mid + \mid Z_i \mid} \tag{4}$$

The higher the value of accuracy,precision,recall and F1-score, the better performance of the learning algorithm.

We have put all of the metrics in a method *evaluate_score(Y_test,predict)* where *Y_test* is the true label and *predict* is the predicted label, and returns all the calculated metrics score.

III. EDA - Exploratory Data Analysis

The data consists of 8 columns and 159571 entry for training and 63987 for testing. The first column is id which is an 8 digit integer value that uniquely identify the person who wrote the comment. The next column is comment_text and it's multi line text which contains the comment given by the user. The rest columns are toxic, severe_toxic, obscene ,threat,insult,identity_hate which are binary label containing 0/1 indicating whether the comment belong to the that label. The comment_text will be our input to the model and the labels will be the prediction of our model.

I. Visualization

In this we explore some of the visualization to find out more about our data. First lets see the amount of comments in each labels.

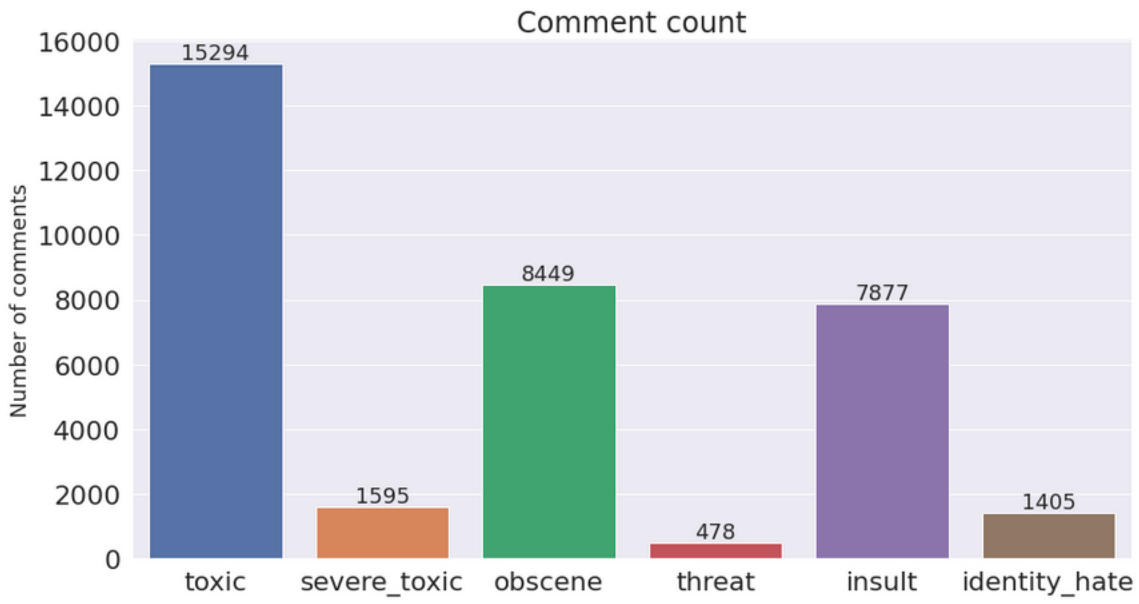


Figure 1: comment count over all labels.

From the visualization above we can observe that most of the comments have a label of toxic comments. Comments with threat label are very small compared to others. This could cause our model to bias towards the most frequent labels and we need to take care when evaluating and choosing our model. There are many techniques to deal with imbalanced class labels in a classification problem. One mechanism is to use appropriate evaluation metrics when evaluating our model. We added F1-score metrics which can be used to evaluate models with class imbalance. The other option is to try out to over sample minority class or under sample majority class or in most cases we can generate synthetic classes. But sampling could reduce our amount of total samples. So if we couldn't get a good result and our models biased towards these labels we will try these methods. For now we will just leave it as it is.

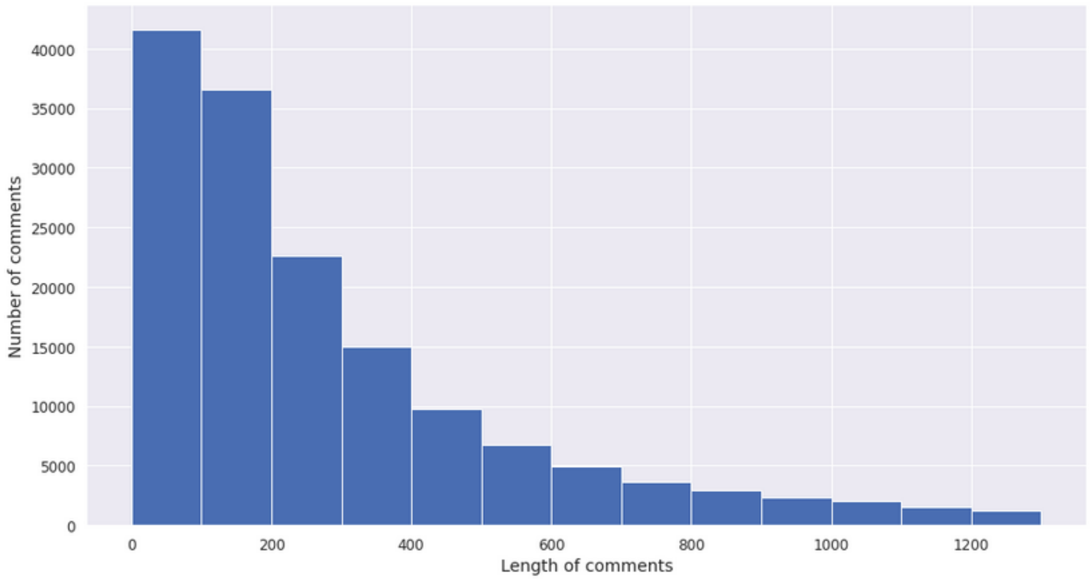


Figure 2: comment length count.

The above visualization shows the number of comments with respect to the length of comments. We want to observe this information because fitting our model on the memory was a problem for us. So we chose to remove lengthy comments. More lengthy comments add more words to our dictionary which causes longer training time and decrease training accuracy. From the visualization we can observe that most of the comments have a length between 0 and 400. So we set a threshold of length 400 and we removed comments having length more than 400.

II. Label Correlation

We also did a correlation analysis to seek for relevant correlations between the label of classes. We found some correlation between the labels.

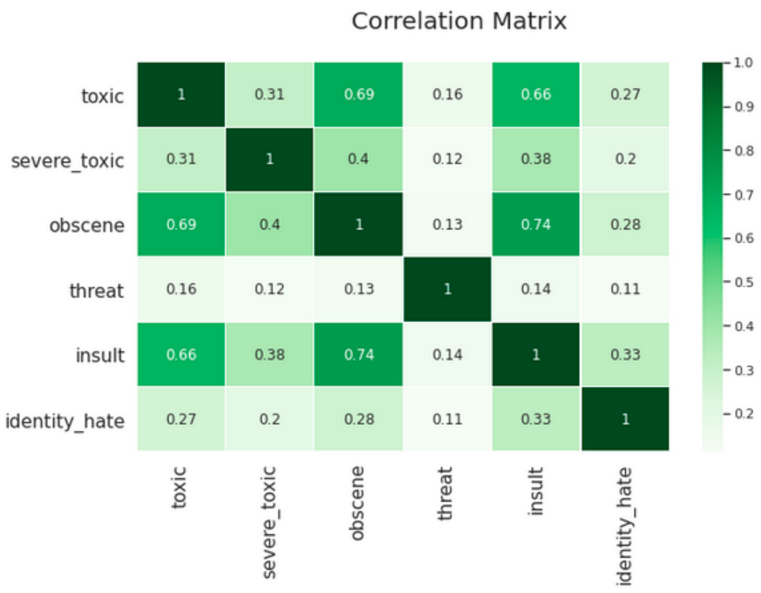
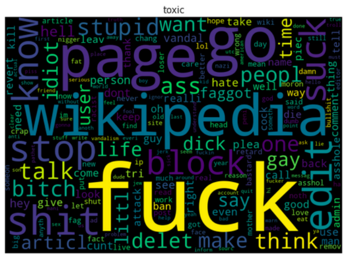


Figure 3: Comment Label Correlation.

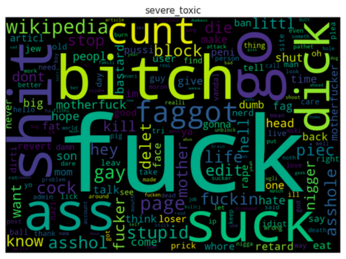
From the correlation heatmap we can observe that toxic comments have high positive correlation with insult and obscene comments. Obscene comments and insult comments are also highly correlated. There is very small correlation between threat and the other class labels. We can explore these correlation in our models. We used ClassifierChain model which will enable us to consider the correlation between class. We will see it later in detail.

III. Word cloud representation

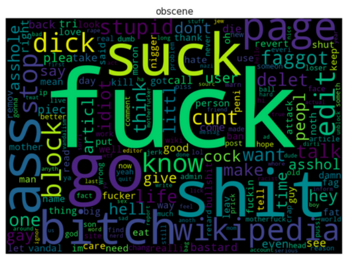
Word cloud representation will enable us to see which word contribute more to the labels.



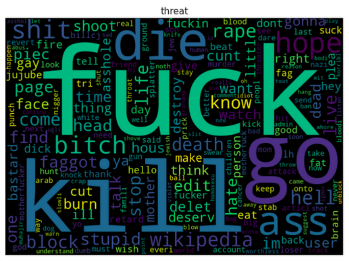
(a) Toxic



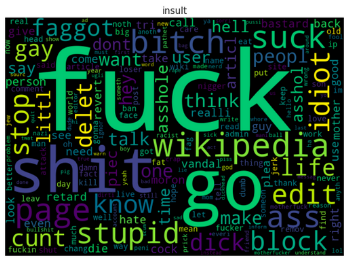
(b) Severe Toxic



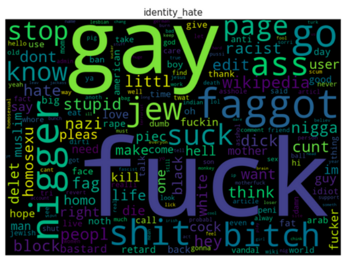
(c) Obscene



(d) Threat



(e) Insult



(f) Identity Hate

Figure 4: Word Cloud Representation

IV. Data Pre-processing

Data pre-processing is very important task in machine learning problem. Especially in natural language processing (NLP) projects we spend most of the time in pre-processing and preparing the data for training.Our pre-processing task is only focused on the comment_text column of our both training and testing data.

I. Removing lengthy comments

Like we stated above in the visualization section, Since we didn't have much computational power we removed lengthy comments from our data-set to reduce the number of our dictionary. We chose a good threshold of maximum of 400 length comment.

II. Cleaning comments

We then created a function to clean URLS, Months, Digits, Emails, new line, non-english characters and convert all the texts to a small letters. Since these contents in comments wont contribute in the toxicity of the comment it is safe to remove them. We applied this to all of the comments in a training and testing dataset.

III. Expanding contracted words

We expanded the contracted words in a comments so that we don't have duplicate words with the same meaning in our dictionary. We used a json file that contain words in contracted words with their expanded words.

IV. Removing stop words

It is a common activity to remove stop words from a text in NLP tasks. We used English stop words from nltk.corpus library.

V. Stemming

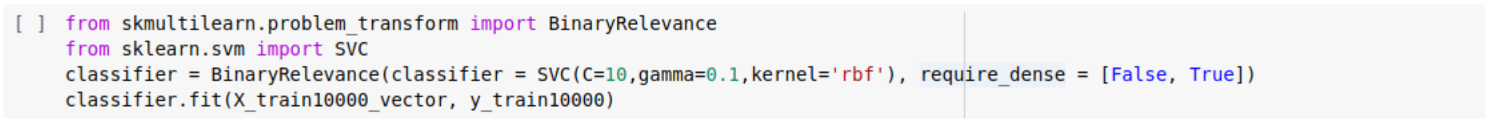
Stemming is also another important task in NLP. Stemming reduces words to their root or base form. Stemming will also help us in reducing our dictionary size. We used PorterStemmer from nltk.stem.snowball to do the stemming.

V. EXPERIMENTS AND MODELING

In this section we are going to discuss different model experiments and design choices. The first approach we tried is Binary Relevance model, then we experimented with classifier chain and Label Power Set model. Finally, We then experimented with deep learning models. We chose CNN, RNN and the combination of the two models. We put into consideration different architectures and word embedding during our experiment. We will discuss all the models and result in detail in the following subsections.

I. Binary Relevance Method with SVM classifier.

This model do not take into account the interdependence of labels. Each labelis solved separately like a single label classification problem. we used support-vector machines (SVM)classifier as a Base estimator with fine tuning. We used support-vector machines (SVM)classifier as a Base estimator with fine tuning. We used TfidfVectorizer to vectorize our text so that we can feed to our models. We set a minimum of 3 document frequency and a maximum of 1000 features. We applied this to both our training and testing data.Since we have large dataset, it would be impossible to fit the data into our memory to learn. It has been causing memory full error and we need to sample down the data.



```
[ ] from skmultilearn.problem_transform import BinaryRelevance
    from sklearn.svm import SVC
    classifier = BinaryRelevance(classifier = SVC(C=10,gamma=0.1,kernel='rbf'), require_dense = [False, True])
    classifier.fit(X_train10000_vector, y_train10000)
```

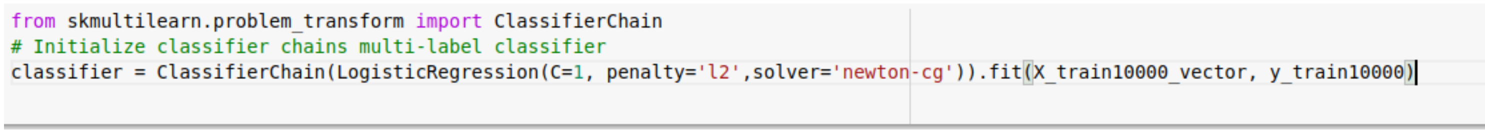
Figure 5: Binary Relevance Model

model hyper-parameters	
kernel	‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’
gamma	0.0001, 0.001, 0.01, 0.1
C	0.001, 0.01, 0.1, 1.0, 10.0, 100.0

Table 1: Binary relevance model hyper-parameters

II. Classifier chain with Logistic Regression classifier.

For this model, we trained the first classifier on input data and then each next classifier is trained on the input data and previous classifier, and so on. Hence this method takes into account some interdependence between labels and input data. Some classifiers may show dependence such as toxic and severe toxic. We used both Logistic Regression model and MultinomialNB to chain inside the classifier chain model. We selected Logistic Regression and MultinomialNB because it showed a better result compared to other model when we experimented with separate classification.



```
from skmultilearn.problem_transform import ClassifierChain
# Initialize classifier chains multi-label classifier
classifier = ClassifierChain(LogisticRegression(C=1, penalty='l2',solver='newton-cg')).fit(X_train10000_vector, y_train10000)
```

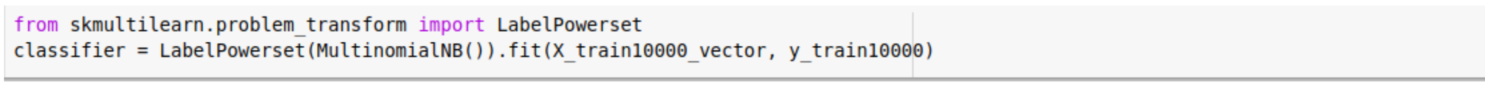
Figure 6: Classifier chain model

model hyper-parameters	
Solver	'newton-cg', 'lbfgs', 'liblinear','sag'
class weight	None, 'balanced'
Penalty	'l1','l2','elasticnet'
C	0.01, 0.1, 1, 10

Table 2: Classifier chain with Logistic regression model hyperparameters

III. Label Powerset with MultinomialNB classifier.

For this model, we considered all unique possible label combinations. Any one particular combination for this reason serves as a label, converting our multi-label problem to a multi class classification problem. Considering our dataset, many comments are such that they have all non-toxic labels together and many are such that obscene and insult are true together. Hence, this algorithm seems to be a good method to be applied.



```
from skmultilearn.problem_transform import LabelPowerset
classifier = LabelPowerset(MultinomialNB()).fit(X_train10000_vector, y_train10000)
```

Figure 7: Label Powerset Model

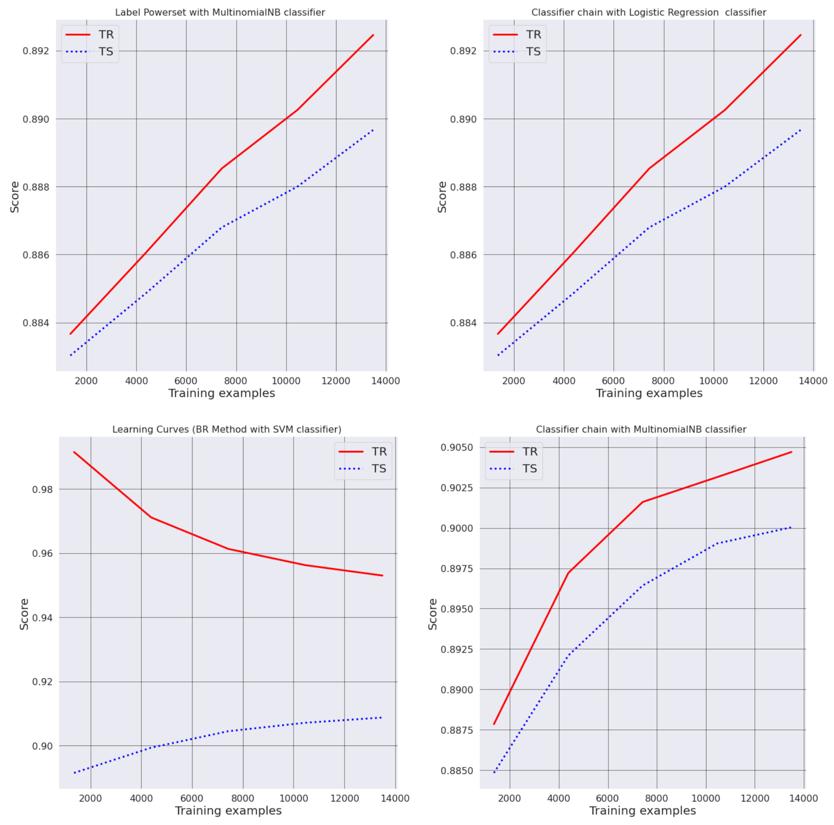


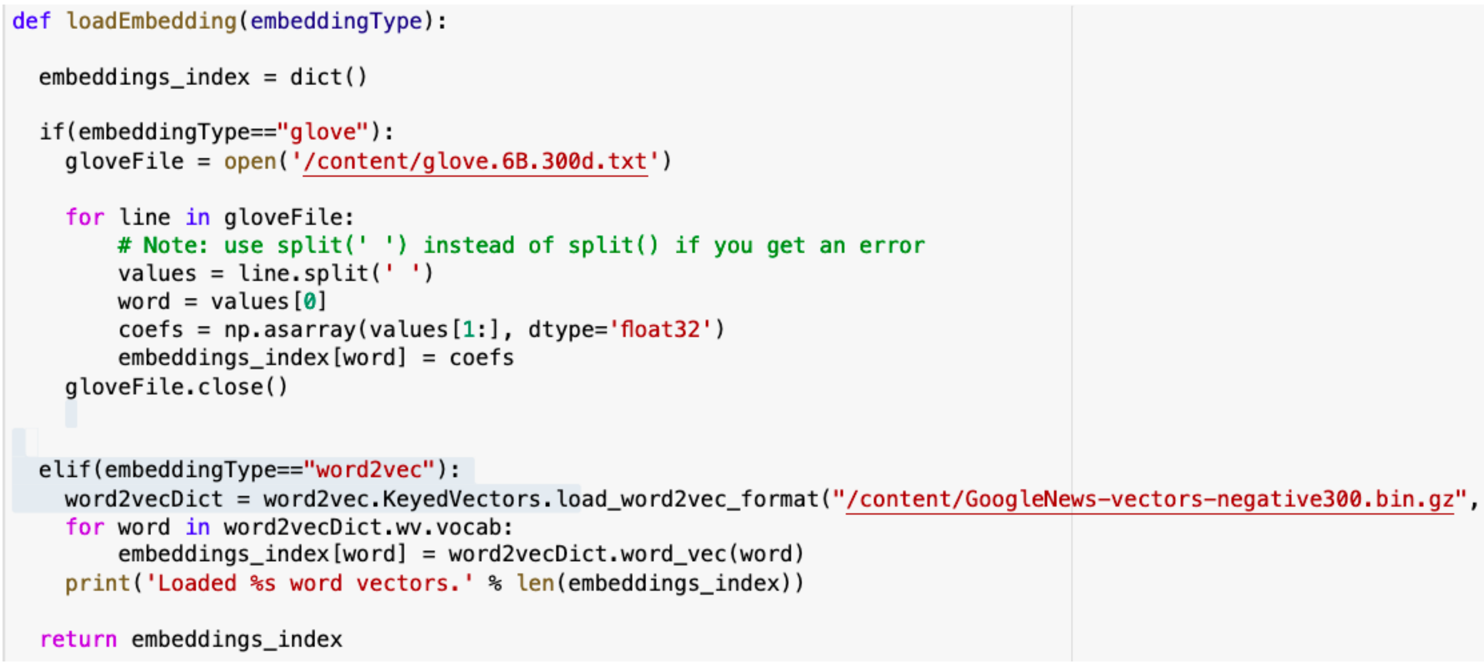
Figure 8: Learning curve

model result			
Model	Accuracy	Log Loss	Hamming Loss
Binary Relevance with SVM	84.39	5.30	0.61
Classifier chain with NB	84.12	6.60	0.49
Classifier chain with LG	85.42	5.19	0.61
Label Powerset with NB	86.36	5.17	0.60

Table 3: BR, CC and LP experiment results

IV. Glove and Word2Vec Embeddings

Embeddings are a break through techniques that enabled neural network models excel in natural language processing tasks. We chose the two most popular word-embeddings, Word2Vec[Kai et ak,2013] and Glove [Pennington et al., 2014]. We selected dimension of 300 for both embedding and we created a method that will make easy to load both of the embedding. We created a method to load the two embeddings easily.



```
def loadEmbedding(embeddingType):

  embeddings_index = dict()

  if(embeddingType=="glove"):
    gloveFile = open('/content/glove.6B.300d.txt')

    for line in gloveFile:
        # Note: use split(' ') instead of split() if you get an error
        values = line.split(' ')
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
    gloveFile.close()


  elif(embeddingType=="word2vec"):
    word2vecDict = word2vec.KeyedVectors.load_word2vec_format("/content/GoogleNews-vectors-negative300.bin.gz",
    for word in word2vecDict.wv.vocab:
        embeddings_index[word] = word2vecDict.word_vec(word)
    print('Loaded %s word vectors.' % len(embeddings_index))

  return embeddings_index
```

Figure 9: Loading embedding

V. CNN - Convolutional Neural Network

Convolutional neural networks(CNNs) are most commonly applied to the task of analyze visual analysis in imagery. For instance, CNN are used in image classification, face recognition, image segmentation and others. However in the recent years CNNs are now being used to problems other than visual analysis and natural language processing is one of the areas where CNN started to shine. For this reason we wanted to explore and learn how to apply CNN in this task. We selected the below hyper-parameters for our architecture.

Architecture hyper-parameters	
Activation	'relu','than','sigmoid'
Batch size	16,32,64
optimizers	'SGD', 'RMSProp', 'Adam'
Drop out	0.2,0.3,0.5

Table 4: CNN model hyper-parameters

We couldn't perform a full grid search on all parameters combination because of the resource constraint we have. We manually experimented with a fewer configuration and we selected a batch size of 32, relu activation for our hidden layers and Adam as our model optimizer optimizer.

In general, Our architecture consists of an embedding layer, a convolutional layer with a relu activation function, a maxpooling layer, a batch normalization layer, then a global max pooling layer. Then we added a drop out layer to control the complexity of our model(Hinton et al., 2012). Then one dense layer with 50 neurons and final one output layer with 6 neurons each for our output layers.

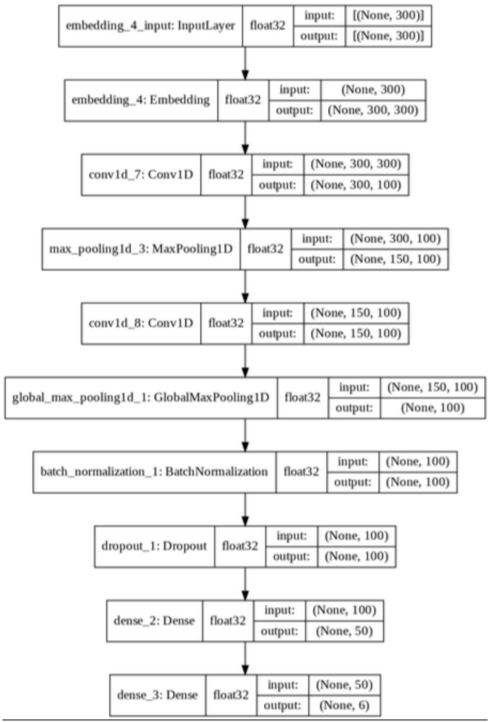

embedding_4_input: InputLayer | float32 | input: [(None, 300)] | output: [(None, 300)]
embedding_4: Embedding | float32 | input: (None, 300) | output: (None, 300, 300)
conv1d_7: Conv1D | float32 | input: (None, 300, 300) | output: (None, 300, 100)
max_pooling1d_3: MaxPooling1D | float32 | input: (None, 300, 100) | output: (None, 150, 100)
conv1d_8: Conv1D | float32 | input: (None, 150, 100) | output: (None, 150, 100)
global_max_pooling1d_1: GlobalMaxPooling1D | float32 | input: (None, 150, 100) | output: (None, 100)
batch_normalization_1: BatchNormalization | float32 | input: (None, 100) | output: (None, 100)
dropout_1: Dropout | float32 | input: (None, 100) | output: (None, 100)
dense_2: Dense | float32 | input: (None, 100) | output: (None, 50)
dense_3: Dense | float32 | input: (None, 50) | output: (None, 6)

Figure 10: CNN Architecture

The batch normalization layer can be usefull to speed up the training time and prevent us from phenomenon called internal covariate shift [Sergey and Christian, 2015].

V.1 Experiments

In this section, we will describe our experiment on CNN with different configurations. For all experiments we set a validation split of 0.2.

We started off by tokenizing our data. We set a parameter of 20,000 maximum number of vocabulary size for our tokenizer and a maximum of 300 words per comment. We experimented with GloVe and Word2Vec pre-trained embedding and self-embedding.

Convolutions neural network					
Embedding	Accuracy	Log Loss	Hamming Loss	F1-score Macro Avg	F1-score Micro Avg
Glove	85.0	2.16	3.58	0.34	0.63
Word2Vec	85.2	2.46	3.54	0.32	0.61
self-embedding	83.4	2.11	3.53	0.32	0.62

Table 5: CNN experiment results

As seen on the table Glove word embedding showed a better performance in F1-score compared to other word embeddings.

VI. RNN models

We have also considered RNN models for our task. They were first proposed by [Bengio et al, 2003] to be used in language modeling. Our RNN architecture consists of embedding layer, bidirectional LSTM layer, batch normalization layer like we used in CNN architecture, followed by batch normalization layer and global max pooling layer. We added a drop out layer and fully connected layer with one hidden layer and one output layer with 6 units.

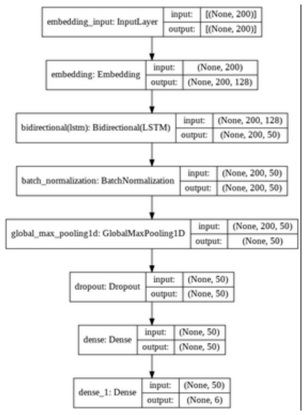


Figure 11: RNN Architecture

VI.1 Experiments

We run the baseline model with embedding layer LSTM layer with 25 units and drop out layer of 0.2. We tested on all the embeddings we used. The result of our experiment is summarised below.

RNN model					
Embedding	Accuracy	Log Loss	Hamming Loss	F1-score Macro Avg	F1-score Micro Avg
Glove	86.03	1.6	3.9	0.34	0.64
Word2Vec	87.63	2.07	3.1	0.37	0.63
self-embedding	85.91	1.84	3.54	0.3	0.62

Table 6: RNN experiment results

From this experiment Word2Vec showed a better performance comparaed to other embeddings.

VII. RNN + CNN model

This model combines RNN and CNN model together. This technique is used in Image classification and also many other applications like DNA protein binding site detection[Zhang et el, 2020] on DeepSite architecture. We hoped that by using the combination of the two models we can a better result. The below graph shows the model architecture.

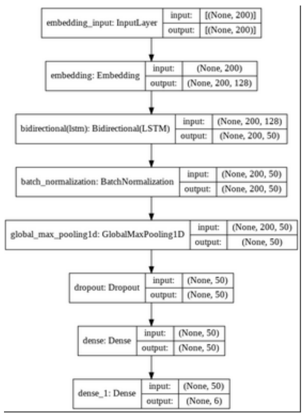


Figure 12: CNN Architecture

VII.1 Result

We run our experiment on LSTM of unit 25 and kernel size of 64 with relu activation for both LSTM unit and CNN unit. We run on all the embeddings to see the performance as well. We have summarized the result we get below.

RNN + CNN model					
Embedding	Accuracy	Log Loss	Hamming Loss	F1-score Macro Avg	F1-score Micro Avg
Glove	86.6	1.9	3.4	0.32	0.62
Word2Vec	88.14	1.9	3.16	0.36	0.62
self-embedding	87.1	2.28	3.54	0.32	0.63

Table 7: RNN + CNN experiment results

VII.2 Kernel Size effect

We further want to explore if the size of the kernel have an impact on the performance of our model. We selected RNN + CNN with word2vec embedding since it showed a better performance in our experiment. For this experiment, the result has been shown in the following table.

Different Kernels for RNN + CNN					
Kernel Size	Accuracy	Log Loss	Hamming Loss	F1-score Macro Avg	F1-score Micro Avg
13	85.03	2.6	3.9	0.34	0.59
32	86.53	2.19	3.1	0.32	0.62
64	88.14	1.9	3.16	0.36	0.63
128	87.03	2.24	3.17	0.32	0.62

Table 8: Experiment on different kernel sizes

From this experiment we can understand that the number of kernel do matter and have a performance impact on our model. Increasing the number of kernel showed an improvement but too much amount of kernel might not show a better result if not the same. For our experiment kernel size of 64 seems a good choice.

VI. DEPLOYMENT

We wanted to deploy our best performing model which is RNN+CNN with Word2Vec embedding in real life application. For this we chose to build a telegram bot that is powered by our selected model that can be used as a group discussion moderator. The full implementation of the bot can be found here[2]. When we developed our bot we considered the following functional requirements.

- The bot should be able to classify or assign each possible labels to a comments in a discussion.
- The bot should be able to remove comments that are toxic.
- The bot should warn the user.
- The bot should ban for 24hr the user if the he/she exceeds max threshold of warnings.
- The bot should keep the log of each toxic comments.

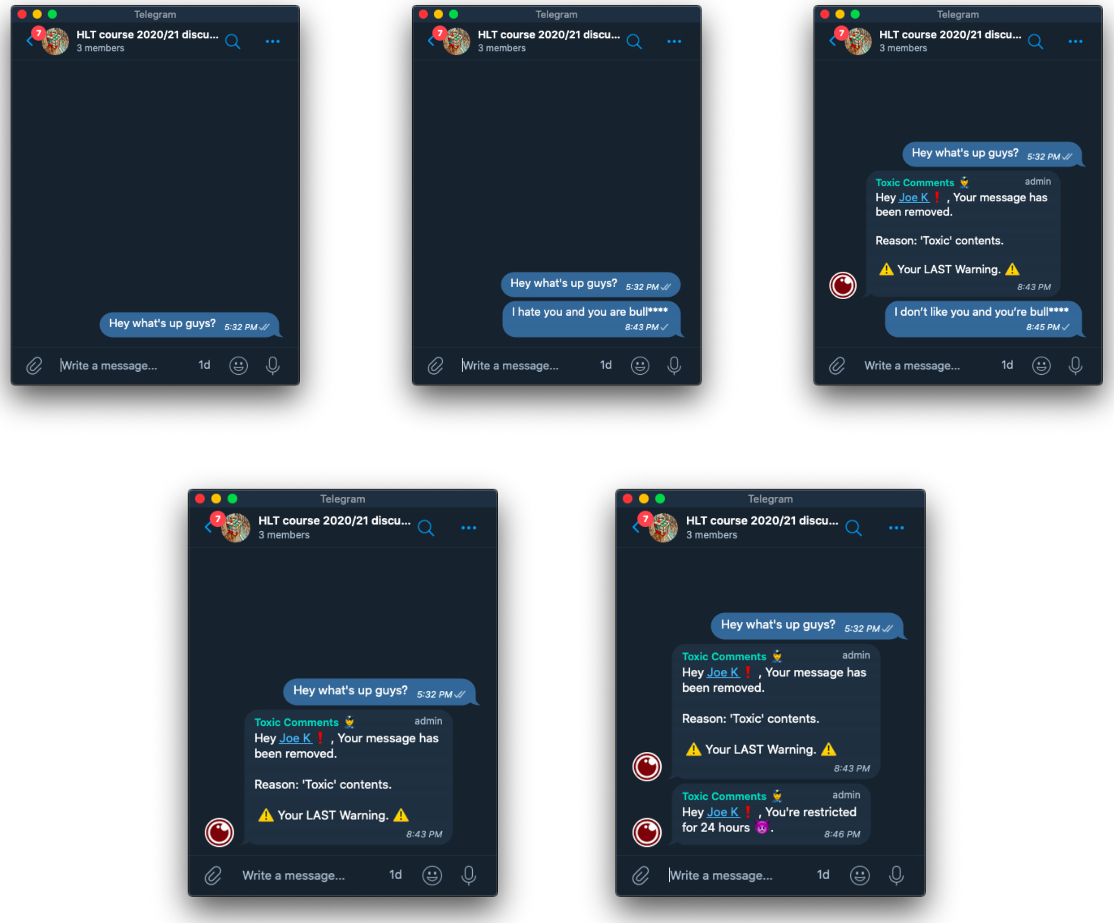
[2]https://github.com/JoeKifle/AI-powered-chat-moderator-bot-

We used pyTelegramBotAPI[3] API to develop our bot. The bot will analyse each message interaction between each group members and make a prediction whether the message contains toxic, severe toxic, threat and identity hate content. We also used a Firebase database[4] to store toxic comments and user information. We created two separate databases. One to store user information and one to store toxic comments. We used firebase-admin[5] to code the database interaction.

The `bot.py` is where our main function is defined. We created two python files `preprocess.py` and `db.py`. Inside `preprocess.py` is where all the preprocessing of a comment is implemented. We cleaned the comments like we did in the pre-processing, including stemming, expanding texts. We also implemented the method to load the model and tokenizer in this file. The `db.py` contains the operations relating to the database interactions.

I. Testing and Findings

We created a test group[6] to test our bot. In the below pictures you will see the bot in action.



Hey what's up guys? 5:32 PM

Hey what's up guys? 5:32 PM
I hate you and you are bull**** 8:43 PM

Hey what's up guys? 5:32 PM
Toxic Comments admin
Hey Joe K ❗ , Your message has been removed.
Reason: 'Toxic' contents.
⚠ Your LAST Warning. ⚠ 8:43 PM
I don't like you and you're bull**** 8:45 PM

Hey what's up guys? 5:32 PM
Toxic Comments admin
Hey Joe K ❗ , Your message has been removed.
Reason: 'Toxic' contents.
⚠ Your LAST Warning. ⚠ 8:43 PM

Hey what's up guys? 5:32 PM
Toxic Comments admin
Hey Joe K ❗ , Your message has been removed.
Reason: 'Toxic' contents.
⚠ Your LAST Warning. ⚠ 8:43 PM
Toxic Comments admin
Hey Joe K ❗ , You're restricted for 24 hours 👿. 8:46 PM

Figure 13: Telegram bot

[3]https://github.com/eternnoir/pyTelegramBotAPI
[4]https://console.firebase.google.com/?pli=1
[5]https://github.com/firebase/firebase-admin-python
[6]https://t.me/hltToxicc

VII. Discussion and result summary

This report confirmed that the combination of RNN and CNN architecture gives a better performance in text classification. In addition, the proper selection of embedding layer, in our exiperment Word2Vec and kernel size will also contribute to the increase in performance. We weren't able to do the full hyper parameter search and optimization but with proper model hyper parameter selection and tuning it can easily achieved a better performance. From the problem transformation models we experimented with the Power label set scored a better performance in this classification task. All the developed code is available on github[7].

References

[1] Maslej-Krešňáková, V.; Sarnovský, M.; Butka, P.; Machová, K. Comparison of Deep Learning Models and Various Text Pre-Processing Techniques for the Toxic Comments Classification. Appl. Sci. 2020, 10, 8631. https://doi.org/10.3390/app10238631

[2] [Hinton et al., 2012] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov,R. (2012). Improving neural networks by preventing co-adaptation of feature detectors.CoRR,abs/1207.0580.

[3] [Bengio et al., 2003]Y. Bengio, R. Ducharme, P. Vincent,and C. Janvin. A neural probabilistic language model.JMLR, 3:1137–1155, 2003

[4] [Zhang et el, 2020] ZHANG, Y., Qiao, S., Ji, S., Li, Y. (2020). DeepSite: bidirectional LSTM and CNN models for predicting DNA–protein binding. International Journal of Machine Learning and Cybernetics, 11.

[5] [Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectorsfor word representation. InEmpirical Methods in Natural Language Processing (EMNLP), pages1532–1543.

[6] Tomas Mikolov, et al. "Efficient Estimation of Word Representations in Vector Space." (2013).

[7] [Yon Kim, 2014], Yoon Kim (2014). Convolutional Neural Networks for Sentence Classification. CoRR, abs/1408.5882.

[8] [Minaee et al, 2021], Shervin Minaee and Nal Kalchbrenner and Erik Cambria and Narjes Nikzad and Meysam Chenaghlu and Jianfeng Gao (2020). Deep Learning Based Text Classification: A Comprehensive Review. CoRR, abs/2004.03705.

[7] https://github.com/JoeKifle/Toxic-Comment-Classification