Abusive Language Detection using Recurrent Neural Networks

Md Rafiqul Islam Rabin
PeopleSoft ID: 1797648
mrabin@central.uh.edu

1 Introduction

Nowadays the social media has been widely used for any kind of discussion and feedback. People generally write their opinions or questions on popular platforms like Facebook or Twitter. However, aggression in comments and inappropriate behaviors are also easily spread out over social media. Because of the huge number of users, this is totally impossible for admins to detect and control those aggressive comments. Therefore, researchers are focusing on various machine learning approaches for automatic detection of abusive language.

The goal of this assignment is to learn how to detect and classify the abusive language in noisy and unstructured text, which is known as Abusive Language Detection. The abusive language detection is basically a subtask of sentiment analysis which mainly focuses on the interpretation of aggression (no, overtly and covertly) within text comment. The main task under this assignment is to implement an abusive language detector using Recurrent Neural Network (RNN).

This report presents the list of preprocessing steps that I have applied on the raw text to clean the data, the encoding techniques to convert the tokens into numerical vectors, the approach to train the RNN models on those vectors to implement the abusive language detector, and finally the metrics for evaluation. The model was trained on the training set and evaluated on the development set. The best weighted average F1-score for the training set and development set is 62% and 51%, respectively.

The structure of the paper is organized as follows: This section represents the introduction related to the assignment, Section 2 describes the dataset to be used for solving this assignment, Section 3 demonstrates the approach I followed to solve this assignment, which also describes the experimental setup (i.e. data preprocessing, encoding, models and other parameters) for solving the assignment. Section 4 provides an analysis of the results and findings. Section 5 discusses some of the challenges that occurred during solving the assignment. Finally, Section 6 concludes the assignment.

2 Dataset

The dataset given to us for this assignment comes from the `Facebook` corpus for the training set, development set, and test set. However, the test set comes without labels. The data for all three sets are given as `CSV`. The format of the data is same across all three sets. Each row includes the document_id, comment, and label. Table 1 summarizes the dataset in various aspect. The dataset contains a total of 12000 comments in the training set, 2000 in the development set, and 1001 in the test set. The 75% of the comments have around 30 tokens, however, the maximum number of tokens in a comment is almost 3000 in the training set, 725 in the development set, and 465 in the test set.

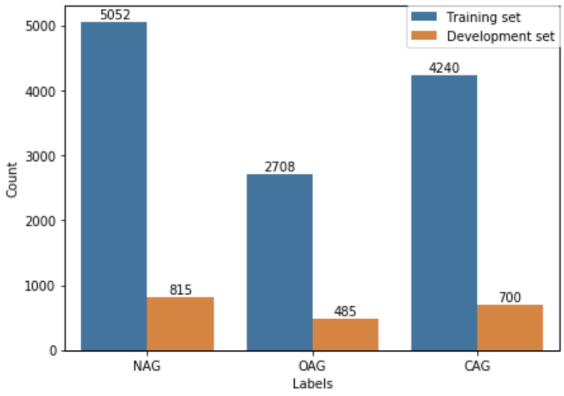

Figure 1: Distribution of labels in dataset.

Figure 1 is showing the distribution of labels in training set and development set. There are three different labels in the dataset:

Dataset	# Total Comments	# Total Tokens	# Unique Tokens	Distribution of Tokens						
				mean	std	min	25%	50%	75%	max
Training set	12000	345538	29726	28.79	51.96	1.0	12.0	18.0	31.0	2918.0
Development set	2000	55646	9963	27.82	35.87	1.0	12.0	18.0	30.0	725.0
Test set	1001	26464	6209	26.44	33.17	1.0	12.0	18.0	30.0	465.0

Table 1: Distribution of comments and tokens in dataset.

- **Non-aggressive (NAG):** There is no aggression in the comment. Example: '*Hope everyone is fine....*'.
- **Overtly aggressive (OAG):** The comment is somewhat containing either aggressive lexical items or certain syntactic structures. Example: '*Cheap publicity...*'.
- **Covertly aggressive (CAG):** The comment is containing an indirect attack where the aggression is generally hidden or has sarcastic negative emotions. Example: '*Haha... He makes his face like an innocent baby..*'.

3 Methodology

In this section, I will describe the experimental setting and the evaluation approach to solve the assignment. The approach can broadly be divided into four main steps which depict an overall view of the evaluation process: (1) Data preprocessing, (2) Token encoding, (3) Build the RNN-based classifier, (4) Training the model, (5) Evaluation and Classification task. I will discuss these in the following subsections.

3.1 Data Preprocessing

In this section, I will discuss different data preprocessing techniques. As the `comment` could not be directly fitted to the neural network as most of them expect numerical feature vectors instead of raw text, we need to extract tokens from the raw text and then we have to represent each token as numerical feature vectors.

The followings are preprocessing steps to extract tokens from raw text that improves the classification in my analysis.

1. **Whitespace and Lowercase:** Removing extra white spaces from comment and converts all uppercase characters to lowercase. Example: '*AAP. Wrong glasses ?*' becomes '*aap. wrong glasses ?*'.
2. **URL and Email:** Replacing URLs and email addresses with the word 'url' and 'email', respectively. I have used regular expressions [1] to detect URL and email in comment. Example: '*sent me text to something@gmail.com*' becomes '*sent me text to email*'.
3. **Mention and Tagging:** Removing mention/-tag (@ symbol and word after @ symbol) from comment using regular expression. Example: '*@Prateek you should first become a voter*' becomes '*you should first become a voter*'.
4. **Non-alphabetic Characters:** Stripping all non-alphabetic characters from comment. Example: '*—-you### nailed it.....*' becomes '*you nailed it*'.
5. **TweetTokenizer:** Using the `Python NLTK TweetTokenizer` (Loper and Bird, 2002), I have converted the comments into smaller tokens. Example: *'this will not solve the problem!'* becomes *['this', 'will', 'not', 'solve', 'the', 'problem', '!']*.
6. **Gazetteers:** Removing tokens that are found in the `gazetteer` corpus [2] such as 'USA', 'Bangladesh', 'Texas', etc.
7. **WordNetLemmatizer:** Using the `Python NLTK WordNetLemmatizer` (Loper and Bird, 2002) to represent different forms of words with similar meaning to one word. Example: 'works' token maps to 'work'.

NAG	0
OAG	1
CAG	2

Table 2: Label encoding on target labels in dataset.

Besides the comments, the labels are also text and we have to change those into a numerical

[1] http://www.noah.org/wiki/RegEx_Python

[2] https://www.geeksforgeeks.org/nlp-location-tags-extraction/

value. I have used the Label Encoder transformer of `Python scikit-learn Library` (Pedregosa et al., 2011) to normalize the target labels from text to value. This encoder tags the labels between 0 and n_classes-1. The label encoding on target labels in our dataset has been shown in Table 2.

3.2 Token Encoding

In this section, I will study the 'One-Hot Encoding' and the 'GloVe Embedding' to represent each token as numerical feature vectors. For encoding, we need a vocabulary with tokens of the training set and development set, but without including the test set. The vocabulary has been created with all different types of unique tokens. Our vocabulary contains 32416 unique tokens from the training set and development set. Later, two extra tokens `<PAD>` and `<UNK>` are also added into the vocabulary list. The `<PAD>` means padding that is used to fix the length of input tokens and filled as zero value. The `<UNK>` means an unknown word that is used for a word that doesn't exist the vocabulary set and calculated as the mean value of other tokens. The final size vocabulary of is 32418.

One-Hot Encoding: The one-hot representation of words encodes inputs (i.e. plain text) as an array of 0 and 1 where 1 means the presence of token and 0 means the absence of token. To encode a comment, this encoder initializes an array of vocabulary size with all zeros and then puts 1 at the index of tokens of the original vocabulary. Few examples of one-hot encoding have been shown in Table 3. I have used the one-hot encoding to encode the tokens of comment into numeric vectors.

Tokens of Comment	One-Hot Encoding	Output Shape
['hi', 'there']	[[0, 0, 0, ..., 0, 0, 0] ,[0, 0, 0, ..., 0, 0, 0]]	[2, 32418]
['luxury', 'changed', 'often']	[[0, 0, 1, ..., 0, 0, 0] ,[0, 0, 0, ..., 1, 0, 0] ,[1, 0, 0, ..., 0, 0, 0]]	[3, 32418]

Table 3: Example of One-Hot Encoding.

GloVe Embedding: `GloVe` stands for Global Vectors for Word Representation. This is an unsupervised learning algorithm implemented by Stanford NLP Group (Pennington et al., 2014) for obtaining vector representations for words that have been trained on 6B tokens and 400K vocab. Pre-trained word vectors are already made available publicly to use as 50d, 100d, 200d, and 300d vectors. Few examples using `GloVe.6B.50d` have been shown in Table 4. I have used the GloVe encoding to encode the tokens of comment into numeric vectors.

Tokens of Comment	GloVe Embedding	Output Shape
['hi', 'there']	[[-0.54313, 0.34427, ..., 0.9809] , [0.6849, 0.3239, ..., 0.0624]]	[2, 50]
['luxury', 'changed', 'often']	[[1.2959, 0.8718, ..., -0.0369] , [-0.4004, -0.1933, ..., 0.3664] , [0.0000, 0.0000, ..., 0.0000]]	[3, 50]

Table 4: Example of Glove (6B.50d) Embedding.

Note that I have adapted the demo code of encoder shared by TA[3] for token encoding.

3.3 Training the RNN-based Model

In this assignment, I have been practicing with three different versions of the recurrent networks: (1) Recurrent Neural Network (RNN), (2) Long Short-Term Memory (LSTM), and (3) Gated Recurrent Units (GRU). As GRU is outperforming other recurrent networks, I have also studied different versions of GRU such as Bidirectional GRU (BiGRU) and Multi-layer Bidirectional GRU (MulBiGRU).

Demo code for RNN and GRU with a link of LSTM as reference was given by TA[3] for a separate classification task. I have implemented my code on this shared demo in `Python3`. I have modified the number of classes to 3 which required the following changes as the decision boundary in the classification task is not binary:

- `Softmax` activation function instead of `sigmoid` activation function in classifier to perform the forward pass.
- `CrossEntropyLoss` loss function instead of `binary_cross_entropy` loss function in training to backpropogate the error.
- `argmax` instead of `round` for prediction as classification label is more than 2.

The following parameter settings are used to build the RNN models:

- **Input Size**: This indicates the size of each input sample. This is actually the input size of the first RNN layer. For example, if we

[3] https://colab.research.google.com/drive/12a2m4axuWJOWGdynL925zGObcceskW1v

use one-hot encoding, the input size is the length of vocabulary. Similarly, if we use GloVe.6B.100d, the input size is 100.

- **Number of Class**: This indicates the size of each output sample. This is actually the output size of the last RNN layer. For example, the output size is 3 in terms of our target labels showed in Table 2.
- **Mask**: This is used to handle the variable-length inputs. The model needs to be fixed the length based on the longest input sequence and initialize the additional value with 0 (known as padding). The mask contains 1 to the positions where the sample actually has values and 0 to the rest of the positions.
- **Epoch**: One forward pass and backward pass on all training examples and development examples. I have used 3000 as an epoch value with 20 as patience value. The model will continue training for 3000 epoch but it will stop early if there is no improvement on the development set over consecutive 20 epochs.
- **Batch Size**: It determines how many samples model should be processed before updating the parameters. I have used 64 batch sizes both for training and evaluation.
- **Number of Layers**: For multiple layers, RNNs are stacked on top of each other. The input is feed into the lowest layer of RNN and then the output of the lowest layer is forwarded to the next layer and so on. I have tuned the number of layers in section 4.3.
- **Hidden Dimension**: This indicates the output size of the lowest RNN layer and the input size of the rest of the RNN layers. I have tuned the hidden size in section 4.3.
- **Optimizer**: Optimizer is used to update the weight parameters to minimize the loss function so that the model could learn over training. I have used the stochastic gradient descent (SGD) optimizer for this assignment as it is one of the most popular and effectively used optimizer. At the start of each batch, the optimizer is set to zero out the gradients before doing the next backpropagation.
- **Learning Rate**: It takes a value in the range between 0.0 and 1.0. It controls how fast the gradient will step toward the minimum of a loss function. I have tuned the learning rate in section 4.3.
- **Momentum**: It also takes a value in the range between 0.0 and 1.0. However, the learning rate should be kept smaller if the momentum term is large. It is used to avoid the situation of stuck in local minima. I have tuned the momentum in section 4.3.
- **Dropout**: It is one kind of regularization technique used to reduce the overfitting in neural networks by randomly dropping out units from the neural network layer. I have used the dropout value as 0.3.
- **Bidirectional**: When the value of bidirectional is set as `true`, models use the information from both previous state and next state to make predictions about the current state.

After preprocessing the dataset using various preprocessing steps discussed in section 3.1, the token has been encoded using the word encoding techniques discussed in section 3.2. Then, different neural network layer has been created in terms of different RNN classifier (i.e. RNN layer, GRU layer, MulBiGRU layer, etc.) using different hyper-parameters. Using the RNN layer and embedding layer, the RNN classifier has been created. Next, a training loop has been implemented using the parameter configuration such as epoch, patience, batch size, etc. Each epoch performs several batch pass (forward pass and backward pass using batch samples). In each batch, the model performs a forward pass and calculates loss, and the optimizer initiates a backward pass to adjust the parameters for better learning. After each epoch, the model evaluates the performance on the development set and keeps track of the best epoch. Finally, the best model is saved after training and used for final predictions.

3.4 Classification Task

The classification task is to predict the aggressive label for new unlabeled input. In our context, given an unlabeled comment, we have to classify the aggressive type: NAG, OAG or CAG. To this purpose, first, the model encodes the comment into numeric vectors using token encoding techniques and predicts the probability of each label. Then `argmax` is applied to the probabilities to get the index of

the predicted label (i.e. 0, 1 or 2). After getting the prediction on the development set, I have used the Python `sklearn.metrics` Library (Pedregosa et al., 2011) to report the accuracy score, classification score, and confusion matrix. The classification reports output the Precision, Recall, and F1-score for each label including macro and weighted average. After prediction on the test set, I mapped the index of the predicted label to [NAG, OAG, CAG], and saved in a CSV file with document ID. This file later has been sent to TA for anonymous evaluation.

4 Analysis of Results

In this section, I will explain my findings from results based on model selection, token encoding, hyper-parameters tuning, data preprocessing steps and evaluation of model.

4.1 Selection of RNN-based Model

In section 3.3, I have explained different types of recurrent networks. It has already been confirmed by TA in the demo code that Multi-layer Bidirectional GRU (MulBiGRU) performs better. However, before tempting to use the MulBiGRU, I performed an experiment with different models using variable-length GloVe.6B.100d as word embedder and TweetTokenizer as preprocessing technique. The accuracy of the best epoch is showed in Table 5, and MulBiGRU performs better than others.

Model	Best Epoch	Training Accuracy	Validation Accuracy
RNN	23	0.5086	0.4355
GRU	37	0.5282	0.4420
BiGRU	17	0.5191	0.4773
MulBiGRU	13	0.5421	**0.4819**

Table 5: Accuracy of best epoch in various models (config: momentum=0.99, lr=1e-2, num_layers=2, hidden_dim=40, dropout=0.3).

4.2 Selection of Token Encoding

In section 3.2, I have explained the One-Hot encoding and GloVe embedding in detail. In this section, I will run an experiment to select the token encoding techniques using variable length. For simplicity, I used the `GloVe.6B.100d`. The accuracy of the best epoch is showed in Table 6, and GloVe performs better than others.

Encoding	Best Epoch	Training Accuracy	Validation Accuracy
One-Hot	16	0.6229	0.4781
GloVe (6B.100d)	26	0.6329	**0.5156**

Table 6: Accuracy of best epoch on different token encoding techniques (config: model=MulBiGRU, preprocessing=TweetTokenizer, momentum=0.99, lr=1e-2, num_layers=2, hidden_dim=32, dropout=0.3).

4.3 Hyper-Parameters Optimization

To find the optimum value of parameters, I have tuned the number of layer in [2,3,4], hidden dimension in [32, 64, 128], learning rate in [0.01, 0.1], and momentum in [0.99, 0.9]. After tuning, the best model is found with number of layer = 2, hidden dimension = 64, momentum = 0.99, learning rate = 0.01 under fixed length `Glove.6B.100d`. Note that the average length of tokens is around 30 for 75% comments, I used 40 as sequence length for fixed length encoding. The detail result for each configuration is shown in Table 7.

4.4 Selection of Preprocesing Steps

In section 3.1, I have explained various data preprocessing techniques in detail. In this section, I will explain my experiences with those preprocessing techniques. Note that `GloVe.6B` comes with four different dimensions (50d, 100d, 200d, 300d), however, we only used `100d` with `TweetTokenizer`. In this section, I will perform the tuning on those dimensions along with preprocessing. The configuration of the model would be the parameter settings of section 4.3. Table 8 shows the comparison result and we can see that adding other preprocessing steps with `TweetTokenizer` increases the accuracy in development set in most cases, also the best accuracy is found using dimension 50d of Glove. Note that '+' in Table 8 denotes the inclusion of all previous preprocessing steps.

Steps	Preprocessing	GloVe.6B			
		50d	100d	200d	300d
Step-1	TweetTokenizer	0.476	0.474	0.475	0.473
Step-2	(+) Gazetteers	0.495	0.496	0.500	0.490
Step-3	(+) URL and Email	0.504	0.496	0.500	0.508
Step-4	(+) Mention and Tagging	0.511	0.503	0.506	0.500
Step-5	(+) Non-alphabetic Charaxters	0.514	0.495	0.499	0.504
Step-6	(+) Lemmatize	**0.516**	0.500	0.509	0.508

Table 8: Accuracy in development set for different preprocessing steps and glove dimension (config: model=MulBiGRU, momentum=0.99, lr=1e-2, num_layers=2, hidden_dim=64, dropout=0.3).

Glove.6B.100d	Number of Layer	Hidden Dimension	Momentum	Learning Rate	Development Accuracy
fixed-length	2	32	0.99	0.01	0.495
variable-length	2	32	0.99	0.01	0.493
fixed-length	2	32	0.9	0.1	0.494
variable-length	2	32	0.9	0.1	0.498
fixed-length	2	64	0.99	0.01	**0.505**
variable-length	2	64	0.99	0.01	0.494
fixed-length	2	64	0.9	0.1	0.494
variable-length	2	64	0.9	0.1	0.498
fixed-length	2	128	0.99	0.01	0.504
variable-length	2	128	0.99	0.01	0.490
fixed-length	2	128	0.9	0.1	0.501
variable-length	2	128	0.9	0.1	0.498
fixed-length	3	32	0.99	0.01	0.499
variable-length	3	32	0.99	0.01	0.491
fixed-length	3	32	0.9	0.1	0.492
variable-length	3	32	0.9	0.1	0.490
fixed-length	3	64	0.99	0.01	0.500
variable-length	3	64	0.99	0.01	0.498
fixed-length	3	64	0.9	0.1	0.502
variable-length	3	64	0.9	0.1	0.492
fixed-length	3	128	0.99	0.01	0.501
variable-length	3	128	0.99	0.01	0.493
fixed-length	3	128	0.9	0.1	0.495
variable-length	3	128	0.9	0.1	0.489
fixed-length	4	32	0.99	0.01	0.498
variable-length	4	32	0.99	0.01	0.489
fixed-length	4	32	0.9	0.1	0.496
variable-length	4	32	0.9	0.1	0.481
fixed-length	4	64	0.99	0.01	0.501
variable-length	4	64	0.99	0.01	0.492
fixed-length	4	64	0.9	0.1	0.497
variable-length	4	64	0.9	0.1	0.489
fixed-length	4	128	0.99	0.01	0.492
variable-length	4	128	0.99	0.01	0.487
fixed-length	4	128	0.9	0.1	0.504
variable-length	4	128	0.9	0.1	0.487

Table 7: Hyper-parameters optimization for MulBiGRU.

4.5 Evaluation of Model

In this section, I will evaluate the best model found in section 4.4 on the training set and development set. The accuracy, classification report, confusion matrix and total misclassified number with other metrics on the training set and development set are shown in Figure 2 and Figure 3, respectively. The best macro average F1-score for the training set, development set, and test set are 60%, 48%, and 46.60%, respectively.

```
Accuracy Score:  0.6334166666666666

Classification Report:
               precision    recall  f1-score   support

          NAG       0.69      0.76      0.72      5052
          OAG       0.60      0.38      0.46      2708
          CAG       0.58      0.64      0.61      4240

     accuracy                           0.63     12000
    macro avg       0.62      0.59      0.60     12000
 weighted avg       0.63      0.63      0.62     12000


Confusion Matrix:
       NAG  OAG  CAG
 NAG [3839  268  945]
 OAG [ 662 1029 1017]
 CAG [1078  429 2733]


Miscalssified:   4399
```

Figure 2: Result of best model on the training set.

```
Accuracy Score:  0.516

Classification Report:
               precision    recall  f1-score   support

          NAG       0.59      0.67      0.62       815
          OAG       0.46      0.27      0.34       485
          CAG       0.45      0.51      0.48       700

     accuracy                           0.52      2000
    macro avg       0.50      0.48      0.48      2000
 weighted avg       0.51      0.52      0.51      2000

Confusion Matrix:
       NAG  OAG  CAG
 NAG [542   63  210]
 OAG [130  133  222]
 CAG [250   93  357]

Miscalssified:   968
```

Figure 3: Result of best model on the development set.

5 Discussion

In this section, I will mention the main challenges I have faced during solving this assignment. Besides, I will also talk about some lacking that could be improved for better results.

The dataset contains comments from social media that are noisy and unstructured. Having misspelling and unknown characters increase the size of the vocabulary. Therefore, dataset cleaning is challenging and important as it has a direct impact on the model's performance. If vocabulary contains unknown tokens, then GloVe will return nothing for them, and those will be filtered out from the candidate token list for prediction. As a result, the meaning of comment may change. Additionally, the dataset contains other languages but written using the English alphabet. It expands the vocabulary and makes the model more confused. Moreover, unknown comment like `Ã·> âù_ï¸è _Ÿ˜˜_Ÿ˜˜ âù_ï¸è <Ã·` also frequent in the dataset, and can negatively impact the performance of the model. There are many comments whose length is very small (i.e. 1 or 2 tokens in comment) or very large (i.e. 3000 tokens in comment). It hurts the samples in batch during training. Therefore, using the appropriate data preprocessing can significantly improve the accuracy score of the model.

The hyper-parameter optimization is also important, as I only used fixed list for parameter optimization, the global optimal point is still unknown. Training the model for larger epochs with floating-point steps can find an efficient configuration.

Furthermore, we can deploy the properties of tokens in features vectors, for example, the POS tag of a token. Or, we can study with n-gram tokens to add more context for models. It could be an interesting experiment to see how the model behaves with these additional features.

6 Conclusion

The social media platforms are useful for opinion sharing and discussion, but also create rooms for harassment. In this assignment, I build an RNN-based classifier to detect those abusive threats in online posts. The experiment shows that choosing the best model, encoding technique and preprocessing steps are important to build the best classifier. However, there are many challenges in the dataset and those left plenty of rooms for improvement. In the future, I would like to implement different sub-model for different kinds of data (i.e. one model for smaller comments, another for larger comments, next for non-English comments, etc.) and perform an ensemble learning for the final model.

References

Edward Loper and Steven Bird. 2002. NLTK: The Natural Language Toolkit. In *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for*

Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543. http://www.aclweb.org/anthology/D14-1162