Hate Speech Detection for Twitter Data

Fadwa Maatug
University of Stavanger, Norway
f.maatug@stud.uis.no

Javeria Habib
University of Stavanger, Norway
j.habib@stud.uis.no

Khoa Le Nguyen
University of Stavanger, Norway
k.lenguyen@stud.uis.no

ABSTRACT

In recent years, the emergence of hate speech on social media has become an urgent problem that needs to be handled. In fact, there are several effective researches and experiments that gain good results on hate speech detection, especially the methods using deep neural networks. In this project, we focus on implementing and comparing some of recent methods and also conduct experiments on data augmentation to improve the efficiency of the models.

KEYWORDS

Recurrent neural network RNN, Convolutional neural network CNN, Long-short term memory LSTM, Gated recurrent unit GRU, GloVe Embedding, W2V Embedding, Textgenrnn, LeakGAN, SMOTE.

1 INTRODUCTION

Hate speech is defined as "speech that targets disadvantaged social groups in a manner that is potentially harmful to them" [2]. In order to classify hate speech, this project mentions several combinations of preprocessing techniques with word embeddings (including Word2vec, Global Vector and an average of both) and deep neural networks, which Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM and Bi-LSTM) and Gated Recurrent Units (GRU) are mainly looked at. Badjatiya et al [13] were the first paper applying deep learning models for hate speech classification and showed high performance.

All of the comparisons and experimental results for those combinations have been recorded and give us the insight information on which model is the best one for the hate speech detection problem. Beside of that, two data augmentation techniques using Recurrent Neural Network and Generative Adversarial Network are brought in and prove the effectiveness by improving the precision, recall and F1 score across all combinations.

2 DATASET

In this project, dataset from Cornell University [2] is used. This dataset consists of 24k tweets labeled by the members of CrowdFlower [3] as one of the following classes:

- `0` - hate speech.
- `1` - offensive language (but no hate speech).
- `2` - neither.

Analysis of dataset shows that it has imbalanced class distribution. Table 1 presents that there is only 5.8% of samples which are labeled as hate speech (class 0). Moreover, class 1 takes a 77.4 % share in the records. Having such an imbalanced dataset, machine learning models would not get sufficient data from minority classes and majority classes tend to overfit. Therefore, optimized results can not be achieved. This leads to the need for data augmentation for hate speech detection [10] which will be presented later in this report.

Table 1: Imbalanced classes of the dataset

Class	Number of Sample	Percentage
Hate speech	1430	5.8%
Offensive language	19190	77.4%
Neither	4163	16.8%

3 PREPROCESSING

The preprocessing phase constitutes of the following steps:

(1) Convert the text to **lower case**
(2) Remove all tweet mentions (any words starting with "@")
(3) Replace the following contraction words with their proper form:
   - n't with not
   - 'm with am
   - 've with have
   - 's with is
   - 're with are
(4) Remove all the remaining symbols.
(5) Replace the words **"amp"** and **"rt"** as these are specific context symbols which are not relevant to speech detection in tweets
(6) (Optional) Remove all stop words. This step is avoided because removing stop words can change the class tweet belongs to originally.

Example:

- Raw Tweet: "!!! RT @mayasolovely: As a woman you shouldn't complain about cleaning up your house. & as a man you should always take the trash out..."
- Cleaned Tweet: "as a woman you should not complain about cleaning up your house amp as a man you should always take the trash out"

4 EMBEDDINGS

Word embedding are vector based representations of individual words with a predefined vector dimension [4]. These vectors have similar values for words with the same meanings. Each vector is learned in a way similar to the neural networks. Therefore, word-embedding learning is also categorized as a deep learning technique.

4.1 Word2Vec

Word2Vec is a type of word embedding which takes the text corpus as input and returns vector embedding for every word in the corpus [14], [15]. Word2Vec is trained on a corpus represented by a list of lists with every item representing a list of words in the corresponding tweet. **Example:** Suppose our data has two tweets:

- as a woman you should not complain about cleaning up your house as a man you should always take the trash out
- me downloads flappy bird sierra welp there goes school

The input to Word2Vec is a list of two lists: [['as', 'a', 'woman', 'you', 'should', 'not', 'complain', 'about', 'cleaning', 'up', 'your', 'house', 'as', 'a', 'man', 'you', 'should', 'always', 'take', 'the', 'trash', 'out'], ['me', 'downloads', 'flappy', 'bird', 'sierra', 'welp', 'there', 'goes', 'school']]

As the output, every word has a vector representation of dimension 100.

4.2 Global Vector

Global Vector (GloVe) is an extension of Word2Vec which was developed as an open-source project at Stanford [5]. For a small dataset, Word2Vec does not have enough corpus to learn meaningful representations. Therefore, GloVe has chances of giving better word embedding as compared to Word2Vec. GloVe data for Twitter having vector dimensions = 100 is being used for this project.

4.3 Combining Word2Vec and GloVe:

GloVe gives more generalized word embedding as compared to Word2Vec which has its scope limited to the text corpus. Therefore, if a word exists in both Word2Vec and Glove, we can have a resulting vector by taking an average of both.

$$Average\_vector = \frac{GloVe\_Vector + Word2Vec\_Vector}{2} \quad (1)$$

4.4 Embedding Layer

Embedding layer can be formed by any of the methods mentioned above. Figure 1 shows the word embedding layer. The dimension of this layer are according to the tweets in the training model. As the maximum length of a tweet is 34 words and every word is represented by an embedding vector of dimension 100, the dimensions of embedding layer are 34×100.

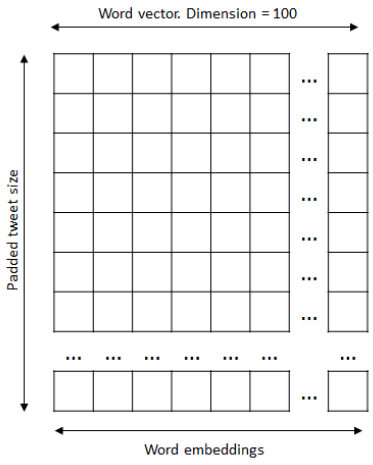

Figure 1: Word Embedding Layer

5 DEEP NETWORK MODELS

Neural networks have been widely used to improve model performance of various natural language processing (NLP) tasks. In this work, two major deep learning techniques have been used.

5.1 Bidirectional Long Short-Term Memory (Bi-LSTM)

Recurrent Neural Networks (RNN) are a type of artificial neural networks (ANN) specifically designed to identify patterns in a sequence of data, e.g. time series data, genomes, videos and text etc. RNN is basically a Long-Term memory based network. One of the drawbacks of Long-Term dependency is that as the sequence grows larger, RNN is not being able to predict the next element in the sequence (exploding and vanishing gradient problem).

Long Short Term Memory (LSTM), is a special type of RNN, which can capture potential long-distance dependencies [8]. As compared to RNN (a simple structure with a single *tanh* layer), LSTMs have four neural network layers [12]. Generally, LSTMs are uni-directional and receive information from past states only.

Bi-LSTM learns information from both past and future states. It works on the principle of running two hidden layers of a uni-directional LSTM in opposite directions. This way, Bi-LSTMS are trained to predict both next and previous value of the current state [1]. The parameters set for **Bi-LSTM** are as follows:

- $units = 100$: the dimension of the output space. As we are using a bidirectional LSTM, the dimensionality would be $100 * 2 = 200$.
- $return\_sequences = True$: full sequence will be returned rather than returning the last value in the output sequence.
- $dropout = 0.2$: fraction of the units dropped while transforming input to the output. For 200 units, $200 * 0.2 = 40$ random units are dropped.

5.2 Convolutional neural network CNN

While RNN-like architectures have been dominant in most of the NLP tasks; Yoon Kim [9] proposed the usage of convolutional neural network (CNN) in the context of sentence classification, which was previously developed for computer vision.

Even though CNNs may lose some vital context information, they still are a powerful tool for feature representation. Moreover, CNN has been utilized not only in the text classification tasks but also in machine translation tasks.[17].

5.2.1 ***The CNN + LSTM Model:*** **CNN** and **RNN** (specifically **LSTM**) are one of the most popular network architectures. In the literature, CNN is well known as an effective network to act as *feature extractors*, whereas LSTM is useful for modeling orderly sequence learning problems. In the context of hate speech classification, intuitively, CNN extracts word or character combinations (e.g., phrases, n-grams), and LSTM learns word or character dependencies (order information) in tweets[19].

The idea of this architecture is similar to the model proposed in [19]. However, in this project, we were interested in evaluating the performance of **LSTM** on top of CNN.

The first layer is a word embedding layer, which passes an input feature space to a drop-out layer with a rate of *0.2* in order to regularise learning and avoid overfitting.

Afterward the output feeds into a 1D-convolutional layer with *62* filters and a window size of *15*. The output features are then passed into a max-pooling-1D layer, where the best model had a pool size of *4*.

The output of the max-pooling-1D is then passed to an LSTM layer which treats each of the feature dimensions as time steps of the recurrence and outputs 100 hidden units per time step.

Next, a global max pooling layer flattens the output space and passes it to the fully-connected layer with *Softmax* output layer. To train the model we have used *the sparse categorical cross entropy* loss function, *Adam* optimiser and *ReLU* activation function.

6 DATA AUGMENTATION

There are several approaches to solve class imbalance problem, such as undersampling - randomly delete samples in the class which has sufficient observations, or oversampling - copy the existing samples from the imbalanced class to increase the number of observations. Data Augmentation is another technique that can counter this problem and it is getting more attention in Natural Language Processing [11].

In this paper, we look at two Natural Language Generation models to generate new samples for hate speech class based on the training data: Recurrent Neural Network (RNN) and Generative Adversarial Networks (GANs).

6.1 RNN for text generation

The idea is to build vocabulary from the training data, then initialize a random word from the vocabulary and predict the next word in the sequence using RNN. *textgenrnn* [16] is selected to implement RNN for text generation. To train the RNN model, the hate speech samples from the original dataset are filtered to make a training data file.

textgenrnn [16]:

(1) Takes in an input of up to 40 words.
(2) Converts each word into a 100 dimensional word embedding vector.
(3) Feeds the embedding matrix containing vectors into two 128-cell LSTM recurrent layers.
(4) All outputs from the embedding layer and LSTM layers are concatenated before they are fed into an attention layer to weight the most important features and average them.
(5) The outputs are then fed into a fully-connected layer
(6) The output of this layer is mapped to probabilities where the next word is the one with the highest probability.

6.2 Text generation using LeakGAN

GAN was invented by Ian Goodfellow et al. [7]. A GAN model consists of one *generator network* that is trained to generate realistic samples and one *discriminator network* to detect if a given sample is from the training data or generated by the *generator*. Both *generator* and *discriminator* are continuously improved until a balance point is achieved. GANs have shown high effectiveness for image generation [7] and recently have been applied in text generation with promising results [6].

In order to find a GAN model to apply to the hate speech detection problem, the results from Texygen have been used. Texygen is a benchmarking platform to support research on open-domain text generation models [18]. LeakGAN [6] has the highest score in the evaluation results of this benchmark, so we decided to do experiments with LeakGAN to generate samples and solve the class unbalancing problem. LeakGAN is highly effective, especially in long text generation (more than 20 words).

7 EXPERIMENT AND RESULTS

7.1 Experiment Settings

After text preprocessing, the dataset is randomly split into 70% for training and the remaining 30% for testing. Moreover, 40% of the training set is used for validation. Precision, Recall and F-score is calculated for every evaluated method.

All models are implemented using ***Keras***. For loss function, ***sparse categorical crossentropy*** is used and *Adam* optimizer is selected.

7.1.1 ***Baseline Model:*** Support Vector Machine (SVM) with TF-IDF vectorization has been used as a baseline model. This model gives good results for tweet classification so it can be compared with other models for analysis.

7.1.2 ***Bi-LSTM:*** . Figure 2 shows the model using Bi-LSTM with parameters described in 5.1. Output of LSTM is flattened and *Dense*

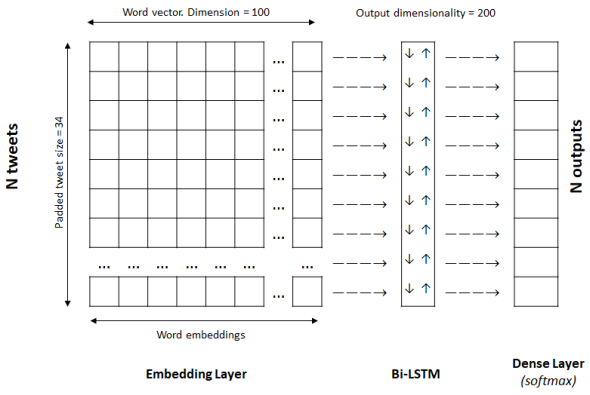

Figure 2: Word Embedding + Bi-LSTM

layer with activation *softmax* is used to generate output. In order to optimize results, 3 epochs have been used while fitting the model to train data.

7.1.3 ***CNN***. We used trial and error method when tuning the hyperparameters. The history train/validate plots were studied to avoid overfitting, especially in our case, where there were a high class imbalance. Moreover, all the hyperparameters are fixed apart from the one that are being optimized. Figure 3 shows the layers of CNN + LSTM model[19].

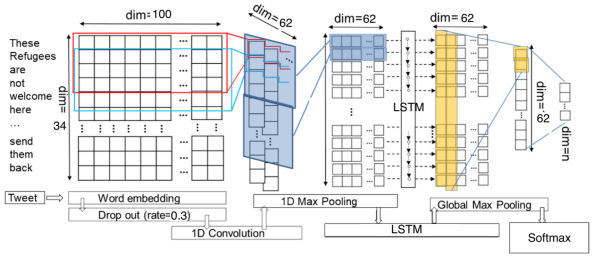

Figure 3: Word Embedding + (CNN + LSTM)

Table 2: SVM model as baseline

Precision	Recall	F1 score
0.89	0.89	0.92

Table 3: Bi-LSTM) with word embedding techniques Comparison

Models	Precision	Recall	F1 score
W2V + Bi-LSTM	0.83	0.86	0.84
GloVe + Bi-LSTM	**0.89**	**0.91**	**0.9**
Average + Bi-LSTM	0.88	0.90	0.88

Table 4: CNN + LSTM with word embedding techniques Comparison

Models	Precision	Recall	F1 score
W2V + CNN + LSTM	0.84	0.89	0.87
GloVe + CNN + LSTM	**0.89**	**0.9**	**0.89**
Average + CNN + LSTM	0.85	0.90	0.88

7.2 Comparison of word embedding techniques

In section (4), three word embedding techniques have been described. Table 3 and 4 show the comparison of these techniques implemented with Bi-LSTM and CNN + LSTM respectively.. Precision, recall and F1 score are being calculated from the ***weighted average*** score of every class. As discussed previously, GloVe embedding gives better results as compared to Word2Vec. Moreover, the embedding generated by using equation (1) also gives results comparable to GloVe. Moreover, the results are also similar to SVM results mentioned in table 2

It can be seen from the plots in Figure 4 that GloVe + Bi-LSTM and Average Model + Bi-LSTM have performance similar to SVM.

7.3 Deep network comparative models

After comparing Bi-LSTM with CNN+ LSTM it was noted that both have similar results, Bi-LSTM is bringing further improvement as noticed with a 1% percent positive increase on F1 score, the BI-LSTM ROC curve for every class is higher than CNN as Figure 5

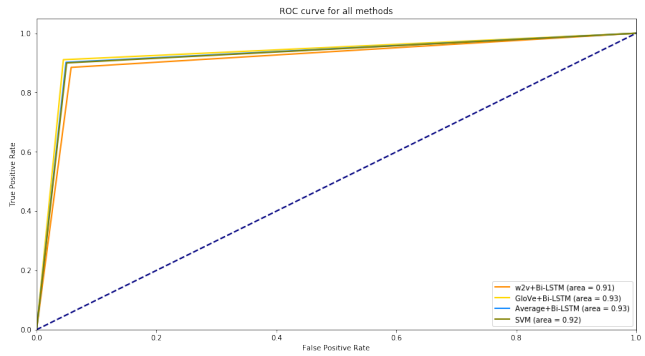

Figure 4: Bi-LSTM ROC curve for every word embedding (compared with SVM)

and 6 illustrate. Table 5 shows the comparison results indicating that F1 score for class zero is considerably low due to the imbalanced classes, this was a sufficient reasoning for applying data augmentation.

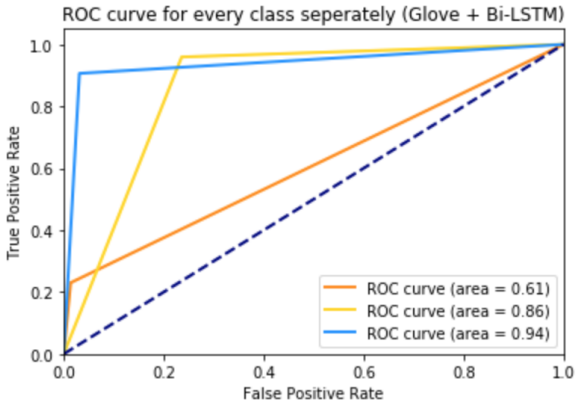

Figure 5: RNN (LSTM) ROC curve for every class separately

7.4 Data Augmentation Experimental Results

We make the training data from all hate speech samples in the original dataset to train *textgenrnn* and *LeakGAN* models . Then we generate 3000 samples using *textgenrnn* and 2944 samples using *LeakGAN*. The generated hate speech samples are appended into the original dataset, and then we shuffle the new dataset. Both data augmentation methods are tested on *GloVe + Bi-LSTM* model and *GloVe + CNN + LSTM* model.

Table 6 shows the experimental results for both methods on hate speech classification. We can see that model's precision, recall and F1 score are all improved with data augmentation. Also, the results of *LeakGAN* are a bit better than *textgenrnn*. Figure 7 and figure 8 shows how LeakGAN improve the ROC curves in comparison to figure 5 and figure 6. Those are promising results for applying GANs for text generation in the further work.

Table 5: Deep network models Comparison

Models	Precision	Recall	F1 Score	Hate Class F1 Score
RNN (LSTM)	0.89	0.91	0.90	0.32
CNN + LSTM	0.89	0.90	0.89	0.36

Table 6: Data augmentation results

Models	Precision	Recall	F1 score
GloVe + Bi-LSTM (without class weighted)	0.51	0.23	0.32
GloVe + CNN + LSTM (without class weighted)	0.40	0.33	0.36
GloVe + Bi-LSTM + textgenrnn	0.71	0.68	0.69
GloVe + CNN + LSTM + textgenrnn	0.66	0.60	0.63
GloVe + Bi-LSTM + LeakGAN	0.74	0.69	0.72
GloVe + CNN + LSTM + LeakGAN	0.74	0.56	0.64

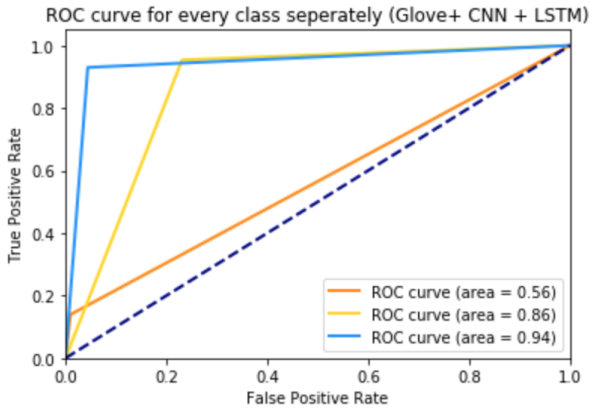

Figure 6: CNN + LSTM ROC curve for every class separately

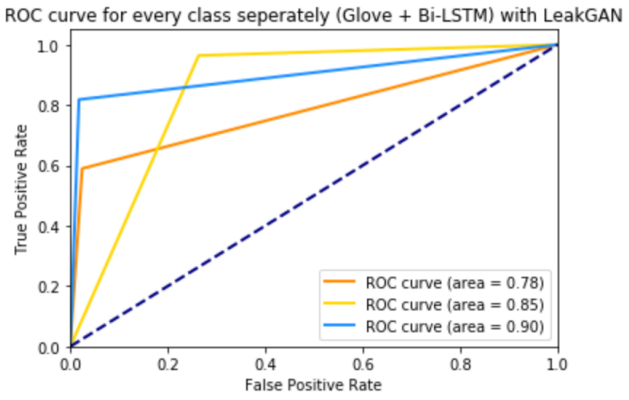

Figure 7: RNN (LSTM) + LeakGAN ROC curve for every class separately

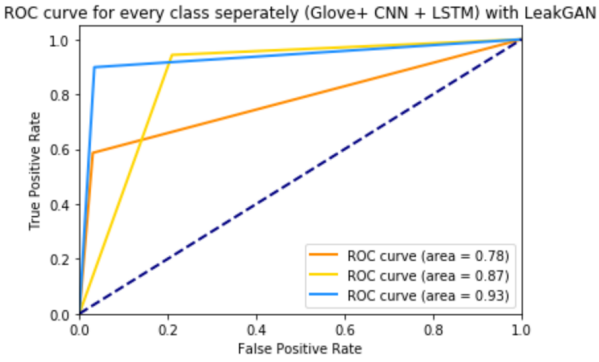

Figure 8: CNN + LSTM + LeakGAN ROC curve for every class separately

8 ADDITIONAL EXPERIMENTS

In this project, we have used CNN models with various architectures as it showed good results in ([9],[19],[8],[17]). Therefore, it was attractive and reasonable for us to experiment the impact on applying this model on our targeted dataset.

8.1 The simple CNN architecture:

A similar model to the static-CNN model presented in [9]. The difference here is that the present model feeds the output of a word embedding using GloVe embedding matrix as the layer weights, into a simple 1D-convolutional layer with (62) filters and a window size of (3). Which passes out convolved features into a max-pooling-1D layer with default pool size (2).

Following that there is a drop-out layer with a rate of (0.3) to avoid over-fitting and then another repeated 1D-convolutional and max-pooling-1D layer follows.

Afterwards, the outcome feature space is flattened into a feature vector and then passes to a fully-connected layers using *ReLU* activation function, follows with the *Softmax* layer output which is the probability of a sample belongs to each class.

Table 7: Overview of best hyperparameters

Hyperparameters	Simple CNN	CNN + LSTM/GRU	GRU + CNN
Optimizer	RMSprop	Adam	Adam
Filters	62	62	256
Kernel	3	15	3
Batch size	20	128	128
Drop out	0.3	0.2	0.2

Table 8: CNN Models Comparison

Models	Precision	Recall	F1 score
Simple CNN	0.89	0.86	0.87
GRU + CNN	**0.91**	**0.91**	**0.91**
CNN + LSTM	0.89	0.9	0.89
CNN + GRU	0.89	0.91	0.88

We have used *the sparse categorical cross entropy* loss function and the mean squared error *RMSprop* optimiser to train the model.

8.2 The CNN + GRU architecture:

This is a similar model to the previous CNN + LSTM model we used in our main experiment. However, we are interested in this step in evaluating the performance of GRU on top of CNN.

GRU (Gated Recurrent Units): It was proposed to make each recurrent unit to adaptively capture dependencies of different time scales. Similarly to the LSTM unit, the GRU has gating units that modulate the ïňĈow of information inside the unit, however, without having a separate memory cells [8].

The layers in this model were treated the same way as mentioned previously on the CNN + LSTM section with the difference of using GRU instead of LSTM.

The key difference between a GRU and an LSTM is that a GRU has two gates (reset and update gates) whereas an LSTM has three gates (namely input, output and forget gates). Thus GRU is a simpler structure with fewer parameters which leads to faster training [19]

8.3 The GRU + CNN architecture

This architecture is based on the model proposed in [17]. Starting with embedding layer with GloVe embedding matrix as weights of the layer. Thereafter, the output is passed into a bidirectional-GRU which helps the network to learn about words from previous as well as future time steps, and avoid some gradient vanishing problem in long sentence [17].

Next, the output feeds into 1D-convolutional layer with (256) filters and kernel with size of (3), and Global max-pooling-1D layer flattens the output and passes it to fully-connected layers with *Softmax* output. To avoid gradient vanishing we add dropout layer with rate of (0.2) between each of the dense layer. Also, we are using *the sparse categorical cross entropy* loss function and *Adam* optimiser to train the model.

8.4 Comparison of CNN Models

Tuning method used in this section is similar to CNN + LSTM, The best hyperparameters are summarised in Table 7

Table 8 compares our CNN models on Precision, Recall and F1 score from the *weighted average* score of each class. The highest figures are highlighted in **bold**. LSTM and GRU on top of CNN both gave close results on performance and speed, the reason of the speed when using LSTM is due to the small dataset. Moreover, GRU + CNN has 2-4 % higher F1 score result comparing to the other models while The simple CNN has the lowest results.

9 CONCLUSION

In summary, from all of our experiments, the pre-trained Word2Vec and GloVe models are very good for embedding layer, and deep learning models we tried out give very high performance. Data augmentation does really solve the problem of imbalance classes and help improve the accuracy. For further work, other state-of-the-art deep neural networks such as attention LSTM models should also be brought in. Given the same importance, a better, classes balanced and lager dataset for hate speech detection should be created to improve the machine learning models.

10 CONTRIBUTION:

Fadwa Maatug:

- The CNN + (GRU & LSTM) model implemetation
- The simple CNN implementation
- The GRU + CNN implementation
- Comparison of CNN + LSTM with all word embeddings
- comparison of all CNN model

Javeria Habib: Made the basic template for training and presenting results. Tasks include:

- Data Preprocessing
- Creation of Word Embedding and Embedding Layers for all models along with word padding.
- Bidirectional LSTM implementation
- SVM Implementations
- Calculation of Precision, Recall, F1score and ROC curves
- Comparison of BiLSTM for all word embeddings with SVM

Khoa Le Nguyen:

- Text generation using textgenrnn and LeakGAN
- Apply data augmentation to Bi-LSTM and CNN + LSTM models to prove the effectiveness
- Report writing: Abstract, Introduction, Dataset, Data Augmentation, Data Augmentation Experimental Results

REFERENCES

[1] BiLSTM [n. d.]. Build with AI. https://deepai.org/machine-learning-glossary-and-terms/bidirectional-recurrent-neural-networks

[2] Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. 2017. Automated Hate Speech Detection and the Problem of Offensive Language. In *Proceedings of the 11th International AAAI Conference on Weblogs and Social Media (ICWSM '17)*.

[3] Figure Eight. 2019. CrowdFlower. Retrieved Apr 24, 2019 from https://www.figure-eight.com/

[4] Embed 2017. What are Word Embeddings for Text. https://machinelearningmastery.com/what-are-word-embeddings/

[5] GloVe [n. d.]. GloVe: Global Vectors for Word Representation. https://nlp.stanford.edu/projects/glove/

[6] Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. 2017. Long Text Generation via Adversarial Training with Leaked Information. *arXiv preprint arXiv:1709.08624* (2017).

[7] M. Mirza B. Xu D. Warde-Farley S. Ozair A. Courville I. Goodfellow, J. Pouget-Abadie and Y. Bengio. 2014. Generative adversarial nets. *Advances in neural information processing systems* (2014), 2672–2680.

[8] K. Cho J. Chung, C. Gulcehre and Y. Bengio. 2014. *Empirical evaluation of gated recur- rent neural networks on sequence modeling.* https://www.researchgate.net/publication/269416998_Empirical_Evaluation_of_Gated_Recurrent_Neural_Networks_on_Sequence_Modeling.

[9] Yoon Kim. 2014. *Convolutional Neural Networks for Sentence Classification.* https://richliao.github.io/supervised/classification/2016/11/26/textclassifier-convolutional/.

[10] Bjorn Schuller Konstantin Hemker. 2018. *Data Augmentation and Deep Learning for Hate Speech Detection.* Master's thesis. Imperical College London.

[11] Emilio Lapiello. 2018. *Shuffling Paragraphs: Using Data Augmentation in NLP to Increase Accuracy.* https://medium.com/bcggamma/shuffling-paragraphs-using-data-augmentation-in-nlp-to-increase-accuracy-477388746bd9.

[12] LSTM [n. d.]. Understanding LSTM Networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[13] M. Gupta P. Badjatiya, S. Gupta and V. Varma. 2017. Deep learning for hate speech detection in tweets. *Proceedings of the 26th International Conference on World Wide Web Companion* (2017), 759–760.

[14] W2V 2017. A Beginner's Guide to Word2Vec and Neural Word Embeddings. https://skymind.ai/wiki/word2vec

[15] W2VWiki 2019. Word2vec. https://en.wikipedia.org/wiki/Word2vec

[16] Max Woolf. 2017. *RNN Text Generator in Python.* https://github.com/minimaxir/textgenrnn.

[17] Dat.nguyen Zhongsheng.wang, Shinsuke.sugaya. 2019. *Salary Prediction using Bidirectional-GRU-CNN Model.* http://www.anlp.jp/proceedings/annual_meeting/2019/pdf_dir/F3-1.pdf.

[18] Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. 2018. Texygen: A Benchmarking Platform for Text Generation Models. *SIGIR* (2018).

[19] J.Tepper Z.Zhang, D.Robinson. 2018. *Detecting hate speech on twitter using a convolution-gru based deep neural network.* https://www.researchgate.net/publication/323723283_Detecting_hate_speech_on_Twitter_using_a_convolution-GRU_based_deep_neural_network.