1. **Introduction**

Text classification is an important part of the natural language processing research and it is invariable used in several research area such as natural language understanding in dialogue system\cite{ grau\_dialogue\_nodate}, sentiment analysis\cite {wilson\_recognizing\_nodate} and scammers identifying\cite{stabek\_case\_2009}. However, the performance of text classification tasks is always inextricably linked with the data size and data quality. Data augmentation can be treated as an excellent method to improve performance in computer vision tasks\cite{berard\_magic\_2003} and some researchers also apply it to the text classification tasks, such as translate back\cite{zhang\_text\_2016}, synonym insertion\cite{topkara\_hiding\_nodate} and GAN\cite{adar\_synthetic\_2018}. However, since all the methods above are costly and time consuming, it is essential to discover a new method in text classification task.

One famous study developed a new algorithm which called Easy Data augmentation(EDA)\cite{wei\_eda:\_2019} to augment data by inserting synonym, replacing synonym, swapping words and deleting words\cite{wei\_eda:\_2019}. But this method can work little if anything on large datasets and number of long sentences is not considered.

In this paper, we improved the EDA by considering a small number of long sentences and also reducing the probability of meaning changed. We evaluated the updated version on a classical text classification task which is the Kaggle toxic comments challenge\cite{van\_aken\_challenges\_2018} with three different models which is TextCNN\cite{xiong\_looking\_2018}, bidirectional Long Short-Term Memory Networks(LSTM)\cite{cheng\_long\_2016} and Bert Transformer\cite{devlin\_bert:\_2019}. The results shows that our updated version can also work well on a large dataset. Our code will be public available.

1. **Related Work**

Data augmentation has been a vital research area in data mining and machine learning. In the area of computer vision, some efficient data augmentation techniques such as flipping and rotation can always improve the performance dramatically with deep learning models. Because they always requirement huge amount of data to train themselves \cite{taylor\_improving\_2017}. However, only few studies have been done in the area of data augmentation with NLP tasks since most of the techniques for it demand a high cost for implementation. In this part, we examine some prominent techniques for data augmentation and text classification on NLP tasks.

* **Back Translation:**

Sergey and Myle\cite{edunov\_understanding\_2018} managed to combine beam search and unrestricted sampling together to augment the training language sentences. They discovered that neural machine translation can be improved by adding more noisy synthetic sentences than genuine bitext when translating them back. They evaluated the performance with BLEU score and it showed that beam with noise data can work best among greedy search and beam search when conduct back translation with more than 17 million sentences. However, most of the translation tools will not be free and invariably have words limitation. Furthermore, even some translation interfaces can only support limited language such as English and Chinese only. As a consequence, back translation can hardly be utilized in practice.

* **DAGAN:** Antreas and Amos\cite{antoniou\_data\_2018} trained a form of data augmentation generative adversarial network(DAGAN) on the vanilla classifier to generate huge amount of example images. The idea is that the effect of true and false can be achieved by iteration and enhancing the training phase in the way of contesting between generator and discriminator. However, the difficulty of this method is that high quality data can be achieved only when the GAN model is well trained, which always leads to relatively complicated problems and demanding an ocean of work.
* **EDA:**

Synonym insertion has been an old and useful technique for data augmentation in NLP tasks. In 2019, Jason and Kai\cite{wei\_eda:\_2019} from Dartmouth College proposed another three mechanism which is synonym replacement, random swap and random deletion combine with synonym insertion to generate the EDA algorithm. They performed all these four methods on five different dataset including Stanford sentiment treebank\cite{socher\_recursive\_nodate}, customer reviews\cite{hu\_mining\_nodate}, question type dataset\cite{li\_learning\_2006}, subjectivity/objectivity dataset\cite{pang\_sentimental\_2004} and Pro-Con dataset\cite{jindal\_mining\_nodate}. Then they tried two excellent deep learning modes which is RNN and CNN on four groups of experimental training set with various sample size ranging from 500 to 10000.Lastly, they revealed that average accuracy improved by about 5% on the small training dataset with 500 samples and less than 1% on the full training dataset. EDA can help avoid overfitting and boost the performance, but in their research we realized that they did not focus on the fewer longer sentences in some special case and the method can work little but anything on large dataset with only around 10000 samples.

1. **Methodology**

In this project, we modified the source code of EDA so that it can generate the number of augmentation by the length of sentences instead of passing the augmentation parameter manually. Then the short sentences will have a higher probability for dropping each of the three operations including synonym insertion, random swap and random deletion to reduce the risk for meaning changed. Correspondingly, we downloaded the dataset from Kaggle toxic comments challenge to be our training set and test set. Finally, we use three state-of-art models which is TextCNN\cite{xiong\_looking\_2018}, bidirectional Long Short-Term Memory Networks(LSTM)\cite{cheng\_long\_2016} and Bert Transformer\cite{devlin\_bert:\_2019} to train the dataset with and without updated EDA. Additionally, we apply glove\cite{pennington\_glove:\_2014},word2vec\cite{rong\_word2vec\_2016} and fasttext\cite{liao\_textboxes:\_2016} to be the word embedding files for TextCNN and LSTM. In this way, we can evaluation the performance of our updated data augmentation algorithm working on this large dataset for solving a text classification problem.

**3.1 Improved version of Easy Data Augmentation(EDA)**

In the original version\cite{wei\_eda:\_2019}, the number of augmented sentences (Naug) is a parameter passed by users manually and it ranges from 1 to 32 depending on the size of dataset. This mechanism only works on the whole dataset instead of taking number of long sentences and short sentences into consideration. As a consequence, we removed the number of augmented sentences and α from the parameter list, where α is a parameter which describe the percent of words will be changed within one sentence. Then we generated number of augmentation and α depending on the size of dataset automatically since Jason \cite{wei\_eda:\_2019} indicated that these two parameters can be inferred from length of sentence \cite{wei\_eda:\_2019}. Moreover, since only 28199 sentences from our training set have more than 100 words (we have 159571 comments in total) then we set 100 words in each sentence to be a threshold to identify whether the sentence is long or short. If the sentence is a short one, we generated less augmented sentences than Naug and more augmented sentences than Naug when it is a long one.

Another problem of the authentic version \cite{wei\_eda:\_2019} is that although most of the augmented sentences retained their original label, there are still some sentences will change their meaning that will hurt the model performance. Corresponding to the paper \cite{wei\_eda:\_2019}, each of the operation can work well alone when they have a little percent of changed words but will hurt the performance of models when the percent is large. Consequently, we added a new parameter which called the probability of operations dropping (pod) to drop each of the three operations including random insertion, random swap and random deletion. Because these three operations will result in the risk of label changed .The same thing as handling the Naug problem we just discussed above, if the comment is a long one then the probability for operations dropping should be low otherwise there will be high probability for operation dropping.

* 1. **Baseline model- 2D TextCNN**

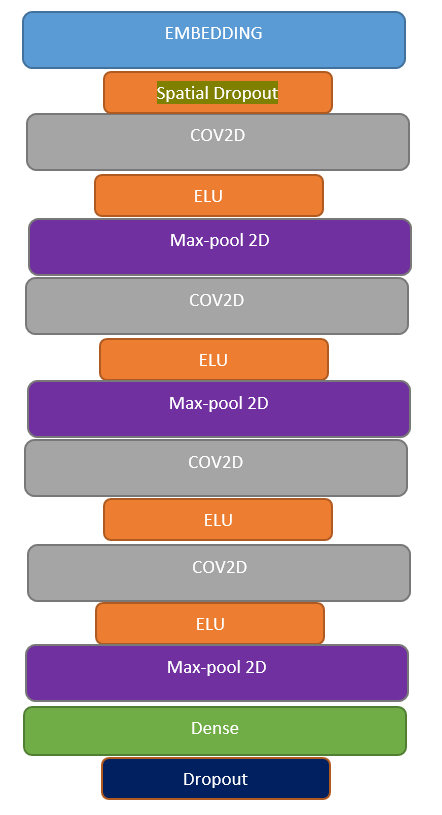
Yoon Kim\cite{kim\_convolutional\_2014} proposed a new Convolutional Neural Networks to solve sentence Classification problem and it is called TextCNN. The main idea is to use multiple kernels of different sizes to extract key information in sentences and it is very similar to multiple window N-gram\cite{abou-assaleh\_detection\_nodate}. In such manner the model can capture local correlation better. Peng and Zhengyu\cite{zhou\_text\_nodate} achieved greater performance when combining Bidirectional LSTM\cite{cheng\_long\_2016} with two-dimensional max-pooling since two dimensional max-pooling can sample more important information when solving sequential problem\cite{zhou\_text\_nodate}. In this project, we constructed a four layers TextCNN as shown in figure 3.1. Each layer assign with a two-dimensional convolutional layer and a two-dimensional max-pooling layer. And we use Elu\cite{wang\_elu\_2017} to be our activation function in the internal layers because this function can produce more accurate results with a faster converge speed to zero. The figure 3.1 shows our architecture of TextCNN.

Figure 3.1: Two-dimensional Convolutional layer and two-dimensional max-pooling layer with ELU activation function

In the convolutional layers, we use 32 filters in total and filter size of 1,2,3,5 for each layer respectively. In terms of the two-dimensional convolutional layers, each layer will assign a 2D filter with n words and k feature vectors. Then the feature Fi,j will be generated from a filter vector Hi:i+n-1,j:j+l-1 by following function:

Fi,j =f(Hi:i+n-1,j:j+l-1 +b) (1)

In the 2D max-pooling layers, we set the vertical pooling size to be the difference between max words length and filter size plus one and horizontal pooling size to be 1. Then if we denote the pooling size to be (P1, P2), then P1 and P2 should be equal to :

(P1, P2) = (maxlen-filter\_size+1, 1) (2)

Then max-pooling function can extract the maximum value from the window of matrix F and the pooling results should be shown as follow:

Pi,j =Max-Pooling(Fi:i+p1,j:j+p2 +b) (3)

For the output layer, we use several sigmoid functions for each of the output unit. The classifier takes the hidden state h as the input. For example, if the input text is S then the prediction label should be:

P(y | s) = Sigmoid (W\*h + b) (4)

Y= argmax (P(y | s)) (5)

Finally, the loss function we used is the binary cross-entropy loss. The loss function is defined as follow:

J(w)= (6)

* 1. **Bidirectional LSTM**

Bidirectional LSTM (Bi-LSTM) composed of forward LSTM and backward LSTM. It is often used to model context information in the area of speech recognition \cite{graves\_hybrid\_2013}, sequence tagging \cite{huang\_bidirectional\_2015} and named entity recognition \cite{chiu\_named\_2016}. Furthermore, Bi-LSTM compensate the disadvantage of plain LSTM when solving encoding problem from back to front information. Additionally, Bi-LSTM works well in fine-grained classification such as strong positive sense, weak positive sense, neutral sense, weak negative sense and strong negative sense \cite{li\_convergence\_nodate}. The LSTM \cite{cheng\_long\_2016} equations has been shown as follow:

Input Gates:

𝑖𝑡 = (𝑊𝑥𝑖𝑥𝑡 + 𝑊ℎ𝑖ℎ𝑡−1 + 𝑊𝑐𝑖𝑐𝑡−1 + 𝑏𝑖 ) (7)

Forget Gates:

f𝑡 = (𝑊𝑥f𝑥𝑡 + 𝑊ℎfℎ𝑡−1 + 𝑊𝑐f𝑐𝑡−1 + 𝑏f ) (8)

Cells:

C𝑡 = f𝑡 ct-1+ 𝑖𝑡 tanh(𝑊𝑥c𝑥𝑡 + 𝑊ℎcℎ𝑡−1 +𝑏c ) (8)

Output gates:

O𝑡 = (𝑊𝑥o𝑥𝑡 + 𝑊ℎoℎ𝑡−1 + 𝑊𝑐o𝑐𝑡 + 𝑏o ) (9)

Cell Output:

ht =O𝑡 \*tanh(C𝑡 )

𝜎 is the activation function and it is the Relu \cite{li\_convergence\_nodate} function in this case. I,f ,c and o represent the input gate, forget gate , cell and output gates respectively. In this project, we use 50 LSTM outputs units ,Relu \cite{li\_convergence\_nodate} as our activation function and binary cross-entropy as our loss function.

1. **Experiments**

In this project, we choose KAGGLE toxic comments classification challenge (TCCC) as our text classification task.

**4.1 Benchmark Dataset**

Conversation AI team and Google planned to construct a clean online conversation environment then they found several human raters to label more than 150k comments from Wikipedia talk page\cite{mohammad\_is\_nodate}. All the dataset contains comment ID and the content of the comment. Each comment has been labeled into one or more classification and there are 6 different classification including toxic, severe\_toxic, insult, threat, obscene and identity\_hate. All the training set and test set are stored as csv format and the size of training set and test set are both more than 150K comments.

**4.2 Data analysis and Cleaning**

In order to clean the dataset, we need to generate some new features from comments to check whether we should clean those contents regarding to these new features.

* **Leaky features:** In order to avoiddataleakage, we generate three new features which is common of IP address, Common Links and Common usernames. Then we check whether there are some comments coming from the same IP, Links or username which means there might be a data leakage problem. The data analysis results are shown in 4.1, 4.2 and 4.3.

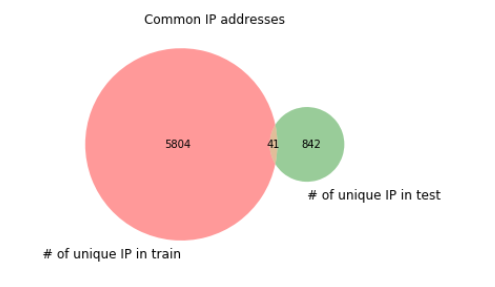


Figure 4.1 There are 41 comments from same IP address

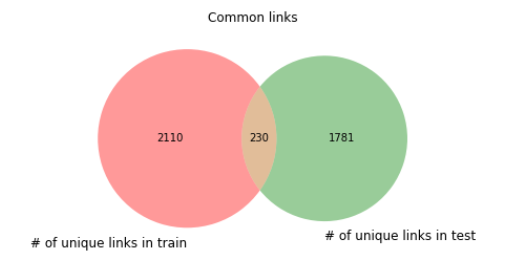


Figure 4.2 There are 290 comments containing same links

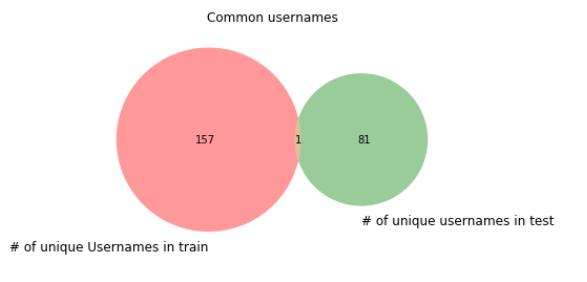


Figure 4.3 There are only one comment coming from the same username

From the analysis from the leaky features, we can discover that links and IP address can lead to data leakage since both training set and test set have many common information. Furthermore, although only one common username appear in both of the training set and the set, we still remove it on the safe side since only 157 and 81 comments containing username in training set and test set. Finally, we remove IP address and Link from each comment.

* **Indirect Features:**

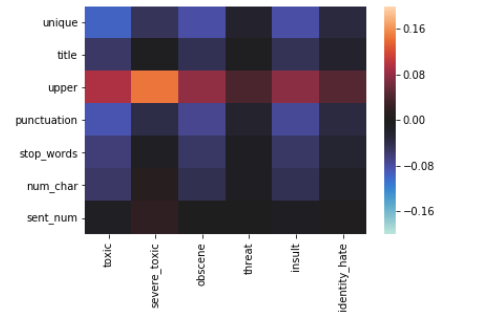
In this part, we defined some statistical features as signage to enrich our models. Our preference is to check whether a comment is toxic or not has some connections with these indirect features including number of unique words (unique), number of title words (title), number of upper words (upper), number of punctuation (punctuation) , number of stop words(stop\_words), number of char(num\_char) and number of sentence in one comment(sent\_num). Then we check the correlations between all these features and all six different types of toxicity. The results are shown in Figure 4.4:

Figure 4.5: number of unique words, number of upper words, number of stop words and number of char have different correlation value between the 6 types of toxic.

As a result, from correlation results from Figure 4.5 we decided to keep the stop words, unique words and upper words since these words might be some useful features to enhance our model.

* **Imbalanced Data:**

We count the number of comments in each of the 6 types of toxicity beside the clean comment. Then we found that most of the comments are clean with 143346 comments and only few of them are toxic one just as shown in Figure 4.6. Then we need treat this problem as an imbalanced classification problem. Whether balancing data will boosting or hurt the performance is an essential thing we need to confirm. So we need to use EDA to balancing the data to check the results.

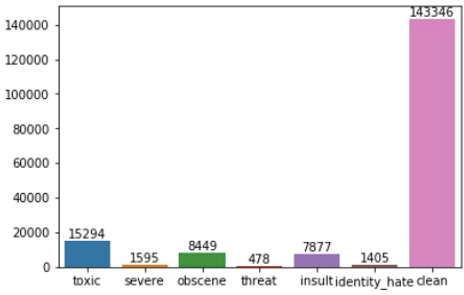


Figure 4.6: most of the comments are clean

* **Corpus Cleaning:**

Before we started to train our model, we need to clean our training set and test set. We performed the cleaning task from the following aspect:

* + Acronym: We developed an apostrophe lookup dictionary which including all the possible apostrophe replacement scenario. Then we use it to standardize our corpus.
  + Spelling Correction: We correct the spelling of some words and kept the case of the original words.
  + Special replacement: In this part, we remove emoji, non-English words and some symbols.
  + Punctuation abolishment : We decide to remove all the punctuation although the number of punctuation is a promising feature. We wish to use word vector to be our direct features then there is no need to keep punctuation. However, stop words are appropriate according to our analysis above then we decide to keep them.
  + Leaky Feature Removement: It is essential to remove all the leaky features including IP address, Links and username. In such way, this removement method can help avoid overfitting due to data leakage.

**4.3 Pre-trained Word Embedding**

Initializing pre-trained word embedding has been a solid solution to boosting performance on text classification problem \cite{abadi\_tensorflow:\_2016}. In this project, we use three various public word embedding files including word2vec\cite{pennington\_glove:\_2014} trained from Google News, fastText \cite{liao\_textboxes:\_2016} created by Facebook AI team and Glove embedding\cite{rong\_word2vec\_2016} from Stanford university.

The dimensional information of these three different word embedding in our experiments are shown in the following Table 4.1

|  |  |
| --- | --- |
| Embedding Files | Embedding size |
| Word2vec from Google News | 300 |
| Glove from Stanford university | 50 |
| fastText from Facebook AI team | 300 |

**4.4 Data augmentation by improved EDA**

Since we have removed α and number of augmentation, we just augmented all the training set and the size training set increased to 757929 samples. Then we keep the ratio between training set and validation set to be 95:5.

**4.5 Training model and metrics**

We run experiments on two-dimensional textCNN\cite{xiong\_looking\_2018} and Bidirectional LSTM\cite{cheng\_long\_2016} with Keras\cite{schoenauer-sebag\_stochastic\_2017} framework, which can help construct and evaluation various model without implementing complex algorithm by hand. For embedding layer, we converted our corpus into word embedding format with the help of our public embedding files.

For textCNN, we use four 2D convolutional layers and 2D max-pooling layers. Number of filter is 32 and filter size for each of the four layers is 1,2,3,5 respectively. Elu function to be the activation function within each layer. Then we use 6 sigmoid function in our output layer and use Adam optimizer to dynamically adjust the learning rate depending on the first moment estimation and second prediction of the loss function gradient \cite{wang\_semantic\_2016}. Our loss function for textCNN is binary cross-entropy.

For bidirectional LSTM, the output dimension in the LSTM layer has been defined as 60.Then we use relu as our activation function. The dropout rate for the first operation of our inputs is 0.1 and the same value as the dropout rate for the application of the recurrent kernel. Then the optimizer and loss function are the same as that of textCNN.

For the training phase, we set batch size to be 265 and number of epoch to be 3.Then the proportion between training size and validation set is keeping in 0.95.

Finally, we use average accuracy between 6 types of toxicity as our performance metrics.

1. **Results and analysis**

In this section, we show our evaluation results about this text classification task with or without improved EDA which are running on our models. Furthermore, we analyze the results to give some hints when handling data augmentation problem.

**5.1 Overall Performance**

This work implements six models, textCNN with Glove, textCNN with word2vec, textCNN with fastText, Bidirectional LSTM with Glove, Bidirectional LSTM with word2vec and Bidirectional LSTM with fastText. The table 5.1 shows the average accuracy as the performance (%) for these 6 models running with improved EDA, original EDA and original EDA with balanced data.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | TextCnn (word2vec) | TextCnn (Glove) | TextCnn (fastText) | Bi-LSTM (word2vec) | Bi-LSTM (Glove) | Bi-LSTM (fastText) |
| Original Data | 97.74 | 97.99 | 98.03 | 98.02 | 98.11 | 98.62 |
| Improved EDA | 98.29 | 98.32 | 98.51 | 98.48 | 98.70 | 99.01 |
| Original EDA with Balanced Data | 92.85 | 95.43 | 95.47 | 93.43 | 94.27 | 94.77 |

Table 5.1 overall average accuracy for all the benchmarks

* 1. **Effect of improved EDA**

Hopefully, we can discover that improved can work best on any model in our experiments. The average accuracy even increased to 99.01% using Bi-LSTM with fastText. Moreover, even though we have a large dataset with more than 700K records after data augmentation, the performance boosting will still keep in 0.5% in each of the model we tried in this experiments. As a result, we actually break the limitation that EDA can only work well in small dataset.

* 1. **Effect of Balanced Data**

The interesting thing is that the model work worst when we are trying to balance the data. Usually, researchers always intend to balance the dataset when they face to an imbalanced dataset. But not in this case, performance will be hurt since too many augmentation operation such as random swap or random deletion can change the original meaning of the sentences. Consequently, those four operations can work well within a reasonable augmentation number.

1. **Conclusion**

This paper introduce an improved version for EDA to reduce the risk of meaning changed and make it work well on a large dataset with more than 700K records. In our experiments, we tried 6 different models and apply them to with original data, with improved EDA and with improved EDA on a balanced data. We can discover that Bi-LSTM work better than textCNN and fastText work best among all the three embedding files. Then we can see improved EDA boost performance on any model we have tried in this project. Finally, we reveal that generate too many augmentation can hurt performance when balancing the whole dataset.