3244-2010-0022 - Wiki Detox: Ready the Banhammer

Authors: Bryan Lim Yan Jie, Christian James Welly, Nham Quoc Hung, Ong Cheng Geng, Sim Sheng Xue, Sun Xiuqi
Emails: e0309849@u.nus.edu, e0324077@u.nus.edu, e0323236@u.nus.edu, e0310581@u.nus.edu, e0324776@u.nus.edu, e0325334@u.nus.edu

Abstract

This paper presents a comparative study on the effectiveness of multiple machine learning models for a textual multi-labelling problem. We examine the performance of classical techniques (KNN, BernoulliNB, MultinomialNB, Logistic Regression without Regularization and Random Forest) and state-of-the-art techniques (BERT, Distilled BERT, RoBERTa, LSTM) in labelling comments.

Our models were trained on the Wikipedia toxic comments dataset from Kaggle's Toxic Comment Classification Challenge in 2017.

We found that either an LSTM model, or an ensemble with BERT & LSTM with Word Embeddings is the most suitable depending on the desired use case.

1 Introduction

As a large crowdsourced online repository of knowledge, Wikipedia boasts many articles with over a hundred thousand active editors on a daily basis. In order to coordinate editing efforts, each article has a discussion page where editors may discuss the information presented.

However, discussion threads can be misused, attracting abuse and insults that detract from productive discussion. Our project aims to build and explore different models that can be used to help discussion moderators identify hateful or toxic comments.

We model our toxic comment labelling problem as a multi-label classification problem. Given that there are many well-known and popular ways to perform text-based multi-label classification, we performed a comparative study on the following groups of machine learning models:

- Traditional ML models
- Long Short-Term Memory (LSTM)
- Bi-directional Encoder Representations (BERT)

2 Related Work

This project is based on the Kaggle Competition 'Toxic Comment Classification Challenge' by Jigsaw/Conversation AI. The reference of our work comes from the various notebooks published and discussions made in their forums.

3 Exploratory Data Analysis

Our data comes from a Kaggle competition for classifying toxic comments. Each input consists of one comment and 6 labels, the meanings of which are specified below:

'toxic' (TO): Comment is harmful, malicious or spam
'severe_toxic' (ST): Comment is particularly harmful or malicious
'obscene' (OB): Comment makes lewd remarks
'threat' (TH): Comment makes a declaration of violent intent
'insult' (IN): Comment is intentionally demeaning towards other users
'identity_hate'(IH): Comment contains ethnic, religious etc. slurs

Figure 1 shows examples of the labelled comments.

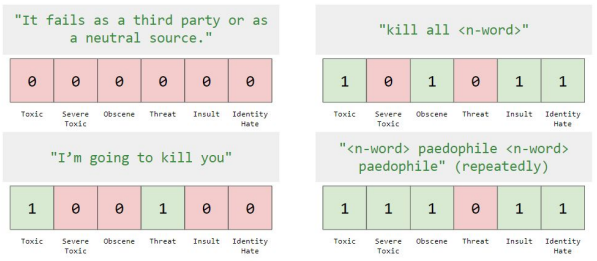

Figure 1: Labels of comments

Based on exploratory data analysis, we discovered the following distribution of labels in our dataset:

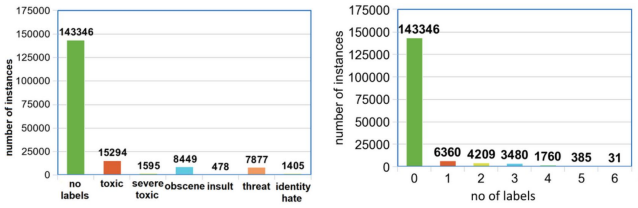

Figure 2: Distribution of labels and the number of instances with the number of labels

Figure 2 demonstrates a vast class imbalance in our dataset. The number of unlabeled inputs greatly outnumbers the labelled data. As we might expect, the number of instances decreases with the number of labels.

4 Method

4.1 Traditional Machine Learning Methods

4.1.1 Pre-Processing Pipeline (For Traditional ML)

Before training, we put the data through the same preprocessing pipeline in order to clean and standardise the training data.

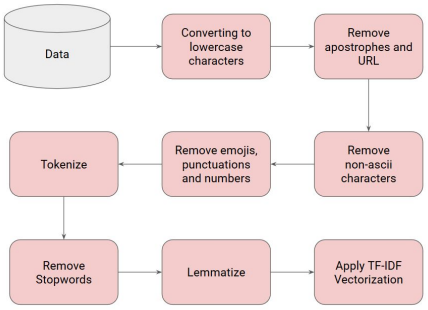

Figure 3: Pre-processing pipeline

4.1.2 Baseline Models

For our baseline model we attempted to create a model that consists of 6 binary classifiers (1 for each label) using the same algorithm and merged their predictions to obtain a prediction for a single comment.

We did this using Logistic Regression (lbfgs solver with L2 Loss), Random Forest, Multinomial Naive Bayes, Bernoulli Naive Bayes and kNN. We also recorded the training times taken for these.

4.1.3 Logistic Regression (Varying Penalty & Regularization)

Seeing how well Logistic Regression performed, we decided to vary the loss function (L1, L2) and regularization (0.01, 1, 100) applied to

further improve the model. We also changed the solver used from L-BFGS-B to SAGA in order to test L1 loss function.

4.1.4 Binary Relevance

In an attempt to further improve upon the basic Logistic Regression model, we attempted to make use of the popular sk-multilearn library. We trained multiple models using the offered Binary Relevance model.

4.1.5 Classifier Chains

We also made use of the Classifier Chains model offered by the sk-multilearn library. We expected this model to have better performance than the Binary Relevance model.

Due to RAM constraints, we were regrettably unable to train the Classifier Chains models on more than 3000 features.

4.2 LSTM Models

4.2.1 Word Embeddings

We attempted to use word embedding in order to overcome the shortcomings of TF-IDF in comment vectorization.

As a Bag-of-Words approach, TF-IDF is unable to capture the interdependency in meaning between different words. With word embedding, the probability of each word appearing before or after another is taken into account during vectorization. As a result, words that appear within the corpus under similar contexts would end up near each other in the vector space.

Moreover, TF-IDF also faced an issue in terms of huge dimensionality. Due to the sheer size of the vocabulary, the generated feature matrix was extremely sparse, with most vector components set to 0, even after pre-processing (word cleaning, lemmatization, removal of stopwords, limiting max features). In contrast, word embedding tends to project words into a dense vector with no zero values.

	a	an	apple	ate	banana	eat	i	today	will
Doc 1		0.0811	0.0811	0.0811			0		
Doc 2		0.0676	0.0676			0.1831	0	0.1831	0.1831
Doc 3	0.2197			0.0811	0.2197		0		

Figure 4: Sparse TF-IDF matrix with high dimentionality

We explored one particular type of embedding known as Word2Vec. Word2Vec utilizes a simple neural network with a single hidden layer to perform self-supervised learning on an unlabelled corpus. It does so by preprocessing the text input to construct data pairs so that when a word is inputted, it will try to predict the closest words to the input word and thus forms the labels by itself to allow for adjusting of weights during training to minimise loss.

In a way, it resembles both a neural network as well as an auto-encoder for unsupervised feature learning. Word2Vec takes in a large input vector representing the sentence, compresses it down to a smaller dense vector similar to auto-encoders but instead of decompressing it back to the original input vector, Word2Vec output probabilities of target words for its output so that backpropagation can be performed for loss minimization, similar to neural networks. The hidden layer is a fully-connected dense layer with each set of weights representing each word embedding. This is also known as the Skip-Gram model.

4.2.2 LSTM

It is possible to model our problem as a text sequence classification problem, where each comment is a sequence of words that vary in length, with the goal of assigning toxicity labels to each comment. We chose the Long-Short-Term Memory (LSTM) model for both for its ability to handle the large vocabulary of input words, as well as to capture and preserve the dependencies between distant words in a long input sequence. The LSTM is a variant of a Recurrent Neural Network (RNN) structurally designed to overcome the vanishing gradient problem (short-term memory) in RNN.

Structurally speaking, even though LSTM also processes data passing on information as it propagates forward similar to RNN, the key differences lie in operations within the LSTM's cells. LSTM is constructed with a cell state and various gates. The cell state acts as a conveyor belt that transfers relevant information throughout the sequence chain. This can be seen as the memory of the network. As the cell state traverses through the network, information is either removed or updated by a combination of gates including the Forget Gate, Input Gate, Cell Gate and Output Gate.

With this understanding, we conducted 3 main experiments with LSTM:

a. **LSTM with its own self-trained Embedding**: In this model, we want to project the input words onto a vector space of 300 dimensions. As such, an Embedding hidden layer is put right after the input layer and the resultant weight matrix with 20000 rows (our predetermined vocabulary size) and 300 columns (number of features for each neuron) represent our word vectors. The end goal was thus to fine-tune this hidden layer weight matrix with binary cross-entropy loss function computed from the output layer. An LSTM layer follows the Embedding layer to process each sequence of text. We identify this as a many-to-one sequence problem. Thus, when the last word vector is inputted into the last LSTM cell, the cell has encapsulated the entire sentence representation.

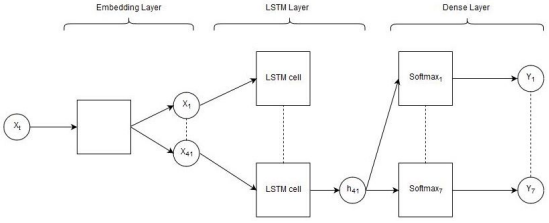

Figure 5: LSTM model with self-trained Embedding Layer

b. **LSTM with Google News Pretrained Embedding:** Since Word2Vec is obtained from self-supervised training on an unlabelled corpus, there are many pre-trained embeddings available on different datasets. One of them is the weight vectors trained on the Google News corpus which could share many similarities with our dataset. To use it, we loaded in the Google News embeddings' dictionary of words and its corresponding weights, identified common words with our dataset to assign them the corresponding pretrained weights. For words that are not shared between two datasets, we randomly initialized weights with the same mean and standard deviation in order to train them subsequently. We also began to explore another variant of LSTM known as Bi-directional LSTM (Bi-LSTM). Bi-LSTM will process the input sequences in both forward and backward directions so that at any point along the network, there are two hidden states combined to preserve information from both the past and the future. Hence, Bi-LSTM can capture the overall context better than unidirectional LSTM.

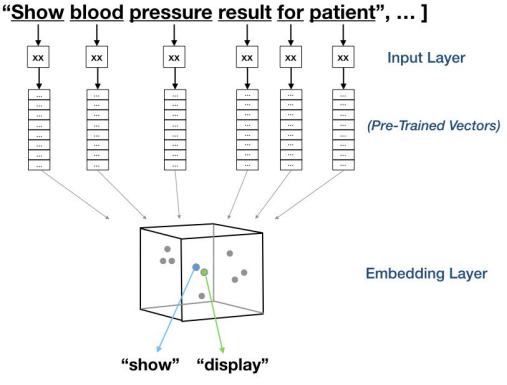

Figure 6: Embedding Layer with pre-trained weight vectors

c. **LSTM with Character Embedding:** Due to the nature of Wikipedia comments, there are a lot of misspellings. This raised concerns over true words that were missing from the train set's vocabulary but appeared in the test set. If we splitted the input sentence into characters instead of words, the characters are more likely to be captured in both train set and test set. With tf.keras.Tokenizer, we are capable of creating a character-level vocabulary and convert each sentence into a sequence of character indexes as input. The only concern, however, is the exploding dimensionality of each sentence with character representation. To resolve this, we made use of Convolutional Neural Network (CNN) with its Convolutional and Max Pooling layers, both of which are capable of performing information-preserving dimensionality reduction.

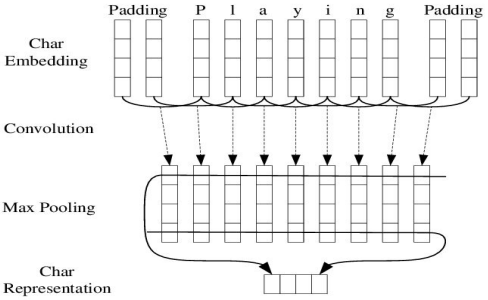

Figure 7: Character Embedding with CNN Layers

4.3 BERT Model Introduction

4.3.1 Baseline BERT model

In line with our framing of the problem as a text sequence classification problem, we explored ways in which we could process text sequence data without losing contextual information in the process. BERT is one model which is able to retain the contextual information provided by the entire sequence by encoding the entire sentence at once with its transformer network.

To enhance its context learning capabilities, BERT utilises two training strategies which help avoid the directional 'next word prediction' method which limits the context learning abilities of other models.

a. Masked Learning Model (MLM)

BERT replaces 15% of the words in each sequence before they are fed into BERT with a [MASK] token. The model then attempts to predict the value of the masked words based on the context provided by the other non-masked words in the sequence.

We add a classification layer on top of the encoder output, and then multiply the output vectors by the embedding matrix, transforming them by the vocabulary dimension. Then the probability of each word in the vocabulary is calculated with softmax.

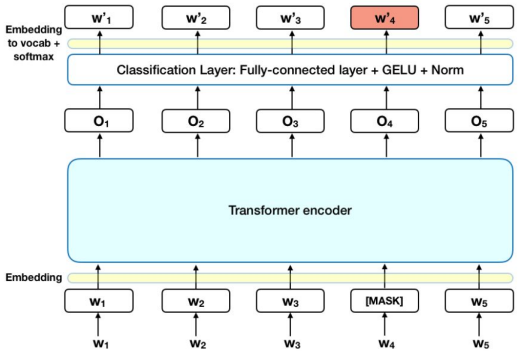

Figure 8: BERT word embeddings

b. Next Sentence Prediction (NSP)

In the BERT training process, the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the sequence. During training, 50% of the inputs are a pair in which the second sentence is the subsequent sentence in the original document, while in the other 50% a random sentence from the corpus is chosen as the second sentence.

A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence. A sentence embedding indicating the sentences are added to each token. Sentence embeddings are similar in concept to token embeddings with a vocabulary of 2. A positional embedding is added to each token to indicate its position in the sequence.

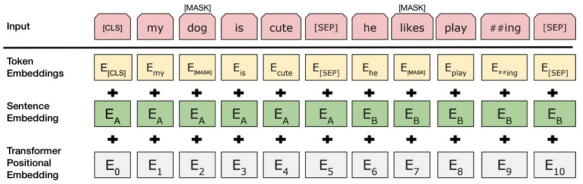

Figure 9: BERT architecture

The entire input sequence enters the Transformer model. The output of the [CLS] token is transformed into a 2×1 shaped vector, using a simple classification layer (learned matrices of weights and biases).
We then calculate the probability of the second sentence being the subsequent one of the first with softmax.

4.3.2 Improved BERT Model using FastBERT and LAMB

In the improvement of the BERT model, we applied the pytorch-learning-rate finder, which allows us to find the optimal learning rate. The learning rate (independent variable) is increased linearly or exponentially between the two given boundaries, which will affect the loss(dependent variable). As seen in the graph below, an optimal static learning rate can be found at about 0.002.

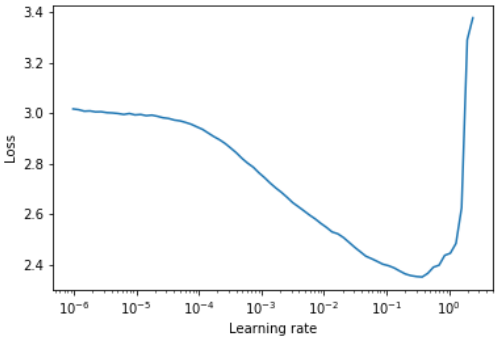

Figure 10: Variation of loss with learning rate

Gradient descent is used to find the weights corresponding to minimum loss. A learning rate that is too small may lead to the model getting stuck, as the step size is too small to escape local minima. On the other hand, a learning rate that is too high corresponds to a huge step size, that may cause the model to diverge as it misses the global minima.

Optimizer	MLP	CNN	Note
Adam	93.05	97.92	Just ordinary Adam
AdamW	93.67	98.00	Used in BERT
LAMB	97.68	98.82	New optimizer for large batch

Figure 11: Performance of different optimisers on batch size of 1024

We used LAMB to replace the ADAM optimizer, which not only provides better results but it is also much faster. LAMB is a new layer-wise adaptive large batch optimization technique. Not only

does LAMB provide better results, it is also much faster than ADAM. As seen in this table, LAMB consistently outperforms Adam and AdamW in most of the times, and shows consistent results among different batch sizes.

4.3.3 Alternative BERT Models

DistilBERT or D-BERT is a lightweight version of BERT which allows access to the performance offered by larger models through the use of 'teacher-student' learning, in which a smaller 'student' network is trained to emulate the outputs of a larger 'teacher' network.

We also looked into how we could further improve our accuracy scores through using an improved version of BERT. RoBERTa leverages a robustly optimized BERT pretraining approach to improve performance. It has been pretrained on 160GB of data, thus offering a more fine-tuned model to work with.

4.4 Ensemble of deep learning models

We also attempted to perform ensemble averaging on BERT and LSTM models in order to reap the benefits of both models.

4.5 Note on validation

Validation was not performed in training our models, as we could not find a good way to split the data to obtain representative validation sets.

Due to the vast class imbalance in our data, attempting a k-fold random split resulted in some training sets having no instances of the lesser-frequency labels (such as 'insult').

Even when we performed stratified sampling, the distribution could not be captured well by the training and validation splits.

Hence, we decided to train each model on the entire training data set.

5 Evaluation

5.1 Baseline Models Results

This table shows the results of using various algorithms for each set of 6 classifiers.

Model used	AUC-ROC	Training Time
Logistic Regression	0.97023	34.2s
Random Forest	0.95633	3722.9s
Bernoulli NB	0.90810	0.7s
Multinomial NB	0.84383	0.4s
KNN	0.66134	0.2s

Figure 12: AUC-ROC score and training time of Traditional Machine Learning models

From the table, we can see that the Logistic Regression model has the best result.

We believe that Naive Bayes models have comparatively poorer performance as they are not complex enough to capture the complexity of the target function.

While the KNN model performs poorly due to the high dimensionality of the feature matrix which means the distance between neighbours was not meaningful for classification.

5.2 Logistic Regression (Varying Penalty & Regularization) Results

Loss	Regularisation (C)	AUC-ROC	Training Time
L1	0.01	0.70623	15.3s
	1	0.96783	371.4s
	100	0.94515	10120.6s
L2	0.01	0.95889	12.3s
	1	0.97024	14.7s
	100	0.95078	77.1s

Figure 13: AUC-ROC score and training time of Logistic Regression models

From the table we see that L2 penalty results in faster training times and slightly better performance.

Firstly, as all the features were text features, it was unsurprising that L2 outperformed L1 across the board. Our data is homogenous, hence L2 is more appropriate than L1 which tends to encourage sparse solutions for feature selection.

Secondly, we see that the optimal regularization budget is C = 1. This is surprising as both increasing and decreasing reduced the performance of the model. Models with lower C values likely had insufficient expressive power to represent the target function while models with higher C will likely begin to fit the noise.

5.3 Binary Relevance and Classifier Chains results

Features	AUC-ROC	Training Time
1000	0.7891	15.774s
2000	0.8202	30.722s
5000	0.9631	720.12s

Figure 14: AUC-ROC score and training time of Binary Relevance models

Features	Regularisation (C)	AUC-ROC	Training Time
1000	1.0	0.94260	167s
2000	1.0	0.95408	328s
2000	10.0	0.94984	828s
3000	1.0	0.95496	503s

Figure 15: AUC-ROC score and training time of Classifier Chain models

As expected, the increased expressiveness of the Classifier Chains model allowed it to perform better than the Binary Relevance model. However, while the AUC-ROC score is rather high, the performance of the model was still lower than the "pure" Logistic Regression model. We attribute this to the lack of expressiveness of

the Classifier Chains model due to the restrictions imposed on the number of features.

We expect this to be the case as the original TF-IDF vectorization generated more than 15000 features. Moreover, we see that the AUC-ROC score continues to increase as we raise the number of features used.

Given more time and resources, we would have liked to train the model on a higher number of features, as we feel that we have not fully explored the use of Classifier Chains in this problem area.

5.4 LSTM results

Model	AUC-ROC	Training Time
LSTM with self-trained Embedding	0.97382	9341s
LSTM with pre-trained Embedding	0.96889	9297s
LSTM with Char-Embedding	0.97008	5612s

Figure 16: AUC-ROC score and training time of LSTM models

The baseline LSTM model with its self-trained Embedding layer performed the best. This is likely because the model could fully capture the vocabulary as well as context of our dataset. Meanwhile, the Embedding weights pre-trained on Google News corpus did not share many common word vectors with our dataset, making their effect on the Embedding layer's weights not sufficient for an improvement. For the model with Char-Embedding, performance was good but did not improve, perhaps because our intuition on misspelt words was not particularly applicable to our dataset.

5.5 BERT results

Model	AUC-ROC	Training Time
BERT	0.96545	2395s
D-BERT	0.96645	1372s
RoBERTa	0.96983	2433s

Figure 17: AUC-ROC score and training time of BERT models

The DistilBERT has a shorter training time as compared to BERT, as it has a smaller network structure compared to BERT. The network size in DistilBERT has been reduced due to the utilization of 'teacher student' learning or knowledge distillation which allows a smaller neural network to emulate the outputs of a larger network.

RoBERTa has a higher AUC_ROC score as compared to the other two models as it has been pre trained on a much larger set of data.

5.6 Ensemble results

Model	AUC-ROC	Training Time
BERT + LSTM w/ Word Embeddings	0.97857	6311s
D-BERT + RoBERTa	0.97669	3805s

Figure 18: AUC-ROC score and training time of ensemble models

We performed an ensemble averaging of two Deep Learning models' outputs and in both cases above, we observed an improvement in result. The AUC-ROC score of LSTM with Word Embeddings and BERT are 0.97388 and 0.96545 respectively, while the score of the ensemble model improved to 0.97857.

This corresponds with the theory of ensemble averaging, which states that combining a group of Artificial Neural Networks can reduce variance at no cost to bias. Furthermore, we observed that an ensemble of two very different models (BERT+LSTM) performed better as compared to an ensemble of the two variations of BERT (D-Bert + RoBERTa). This shows that ensemble averaging can average out the various errors in each model, and an ensemble works best when the individual models are as different as possible.

However, a limitation of our ensemble model is the training time required. Due to the ensemble averaging approach, the multiclass probabilities of the comments have to be generated separately by the individual models. The models can be run in parallel, although more computational power will be required.

6 Discussion

Originally, we performed a comparative analysis based on the F-1 Score. Under this analysis, we found Logistic Regression to be comparatively better at identifying the"severe toxic", "identity hate" and "threat" labels.

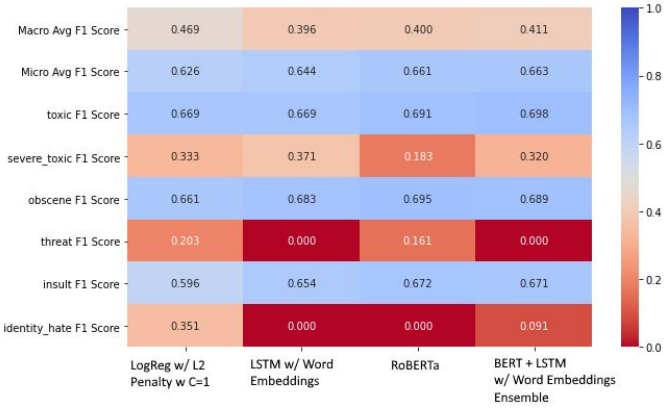

Figure 19: F1 scores of chosen models

Moreover, given the low training time and small improvements to AUC-ROC score (~0.008) brought about by the use of deep learning, we came to the conclusion that the Logistic Regression model was sufficient. However, after further evaluation, we came to the conclusion that the Logistic Regression model was insufficient for the task.

We made further evaluations by using a generalised F Score, the F-Beta Score. It is defined in the following manner:

$$F_\beta = (1+\beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

From this, we managed to discover the following information:

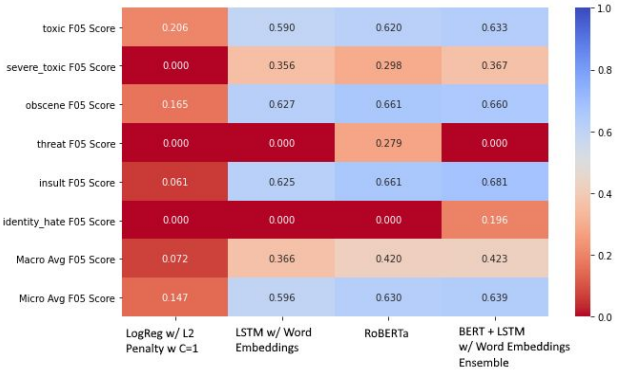

Figure 20: F-0.5 score (favours recall)
Recall = true positives / (true positives + false negatives)

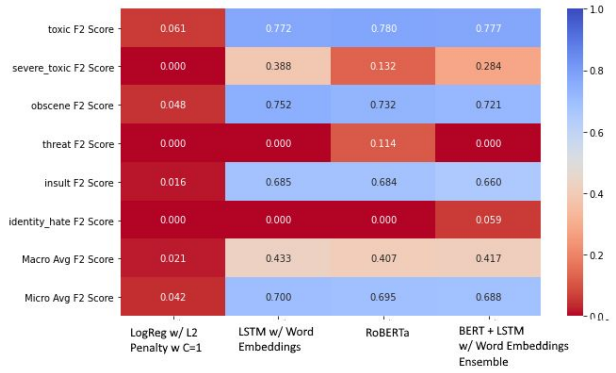

Figure 21: F-2 score (favours precision)
Precision = true positives / (true positives + false positives)

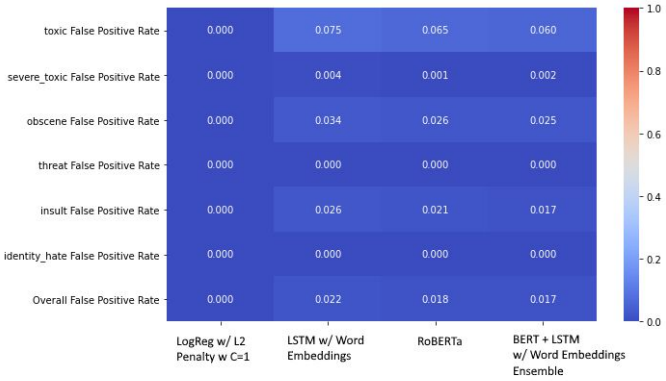

Figure 22: False Positive Rate

While all models perform well here, the ensemble of BERT and LSTM manage to edge out the other 2 models. We speculate that this is due to the theory of ensemble averaging.

While Logistic Regression has no false positives, this is actually emblematic of how the Logistic Regression model is performing no-skill classification and labelling most if not all instances as negative instances.

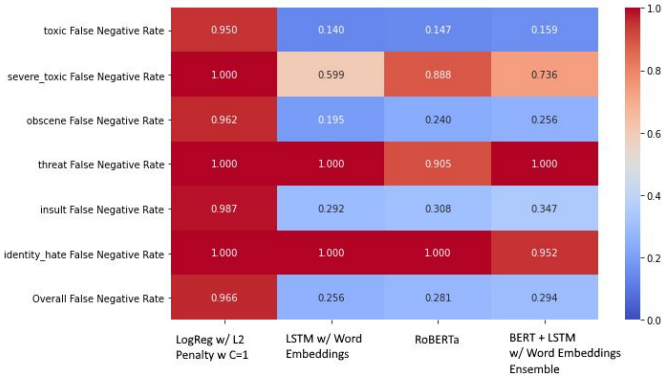

Figure 23: False Negative Rate

Across all the models, we see that LogReg performs sub-optimally compared to the other models. It achieves near-zero F-0.5 and F2 scores.

The high false negative rates for Logistic Regression further supports our theory that the Logistic Regression is performing no skill classification by labelling most instances as negative. The results demonstrate that it is totally unable or is barely able to distinguish between positive and negative instances.

Across the board, our models perform poorly at identifying threat and identity_hate instances. We attribute this to the low number of training instances for these labels.

The LSTM model performs the best at identifying severe_toxic comments. However, this should not be considered in isolation. LSTM on average achieves a low false negative rate compared to the other two deep-learning models. We speculate that this is because RoBERTa uses pre-trained word-embeddings, but LSTM's word-embeddings are derived from training on the dataset.

Therefore, LSTM is able to achieve a more representative word-embedding for the problem. While we note that LSTM has slightly worse false positive rates than the other models, the degree to which it is worse is outweighed by the degree to which it is better than the other models for the false negative rates. This is why we believe that LSTM is still a better classifier as compared to RoBERTa for this classification task.

7 Conclusion

Initially we concluded that among the final models we chose to compare between (Logistic Regression w/ L2 Penalty w/ C=1 vs LSTM w/ Word Embeddings vs RoBERTa vs BERT + LSTM w/ Word Embeddings Ensemble), Logistic Regression would be a good model to use as it had the lowest training time among the models and the difference between the AUC ROC Scores is only ~0.008. Also the F1 Scores that Logistic Regression has shown that it is comparatively better at identifying "severe toxic", "identity hate" and "threat" labels.

However, after scrutinising the metrics such as the False Positive & False Negative Rate, we realised that Logistic Regression was only producing good results due to its almost no skill classification and the imbalance in the test data set. Thus, if it is implemented in the real world, it would not be of much value in identifying true positives.

The best model to use depends on the use case.

If identified toxic comments are flagged for review by a human moderator, we will choose LSTM as the best model. It achieves the lowest false negative rate, while achieving comparatively good scores in other metrics (AUC-ROC and F-beta scores). Hence, it will be likely to let a lesser number of actual toxic comments slip by.

Although it has a slightly higher false positive rate than the ensemble, this penalty is tolerable. The ensemble model only has a marginally lower false positive rate (at 0.005 points less), which would only translate to a tolerable increase in the number of comments to be reviewed by hand.

On the other hand, if there is little or no human intervention and the AI's labelling is immediately used to censor the comments, minimizing the false positive rate might be more crucial as we would not want to stifle legitimate discussions. Such a use case might be present, when manpower is tight and humans cannot moderate all of the flagged comments by the model. Thus, the ensemble model, with its lower false positive rate, might be more suitable.

Acknowledgements

We would like to thank project teaching assistant Shreyas Kuthanoor Prakash, Professor Kan Min-Yen, and peers for giving us valuable feedback on the project, and pointing us towards useful information.

References

[1] Abu-Mostafa, Yaser M., Magdon-Ismail, Malik and Lin, Hsuan-Tien. (2012) *Learning From Data*, AMLBook.

[2] Bishop, Christopher M. (2006) *Pattern Recognition and Machine Learning*. Springer.

[3] Karani, D. (2020, September 02). Introduction to Word Embedding and Word2Vec. Retrieved November 12, 2020, from https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa

[4] Gilyadov, J. (n.d.). Word2Vec Explained. Retrieved November 12, 2020, from https://israelg99.github.io/2017-03-23-Word2Vec-Explained/

[5] Phi, M. (2020, June 28). Illustrated Guide to LSTM's and GRU's: A step by step explanation. Retrieved November 12, 2020, from https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

[6] Sbongo. (2018, January 24). [For Beginners] Tackling Toxic Using Keras. Retrieved November 12, 2020, from https://www.kaggle.com/sbongo/for-beginners-tackling-toxic-using-keras

[7] Horev, R. (2018, November 17). BERT Explained: State of the art language model for NLP. Retrieved November 12, 2020, from https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

[8]Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108.

[9]Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692.

[10] Davidtvs. (n.d.). Davidtvs/pytorch-lr-finder. Retrieved November 12, 2020, from https://github.com/davidtvs/pytorch-lr-finder

[11] Mann, B. (2019, May 02). An intuitive understanding of the LAMB optimizer. Retrieved November 12, 2020, from https://towardsdatascience.com/an-intuitive-understanding-of-the-lamb-optimizer-46f8c0ae4866

[12] Ymcui. (n.d.). Ymcui/LAMB_Optimizer_TF. Retrieved November 12, 2020, from https://github.com/ymcui/LAMB_Optimizer_TF

[13] Brownlee, J. (2020, February 24) A Gentle Introduction to the Fbeta-Measure for Machine Learning. Retrieved November 12, 2020, from https://machinelearningmastery.com/fbeta-measure-for-machine-learning/

[14] Brownlee, J. (2020, January 14). Tour of Evaluation Metrics for Imbalanced Classification. Retrieved November 12, 2020, from https://machinelearningmastery.com/tour-of-evaluation-metrics-for-imbalanced-classification/

[15] Long Ng. (2020, Aug). NLP Preprocessing and Feature Extraction Methods A-Z. Retrieved November 12, 2020, from https://www.kaggle.com/longtng/nlp-preprocessing-feature-extraction-methods-a-z

[16] Mangiavacchi, J. (2018, April 25). CoreML with GloVe Word Embedding and Recursive Neural Network - part 2. Retrieved November 13, 2020, from https://medium.com/@JMangia/coreml-with-glove-word-embedding-and-recursive-neural-network-part-2-ab238ca90970

[17] Ma, X., & Hovy, E. (2016). End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. doi:10.18653/v1/p16-1101

Appendix

Colab Notebooks

Main copy of project Colab Notebook https://colab.research.google.com/drive/1Sil2BGE2hQtNz6YRzAqAEHD1aU3LanWy?usp=sharing

Binary Relevance: https://colab.research.google.com/drive/1LqqlC4idgOCXRksn0h_mHYZFyUh4QY6y?usp=sharing

Classifier Chains: https://colab.research.google.com/drive/1EAtrqw9KftWVOegueuV_6aHgicjOMnfy?usp=sharing

LSTM: https://colab.research.google.com/drive/1jzEx0dgVUBL7JRFuYKYfDAsj6MB4wL_c?usp=sharing

https://colab.research.google.com/drive/1Wwwm6VM8MXVHJU4RYIz2BpwueHgQ3nfz?usp=sharing

https://colab.research.google.com/drive/1giESGrjU6z8d9lE7e7BNouJwmO9aiBMc?usp=sharing

BERT: https://colab.research.google.com/drive/1wD8nja_CwbSBh8-MQzgHsGtDZtISkLT9?usp=sharing