CS3244-22 Wiki Detox: Ready the Banhammer

Ong Cheng Geng, Christian James Welly, Bryan Lim Yan Jie , Sun Xiuqi, Sim Sheng Xue, Nham Quoc Hung

Introduction and Motivation

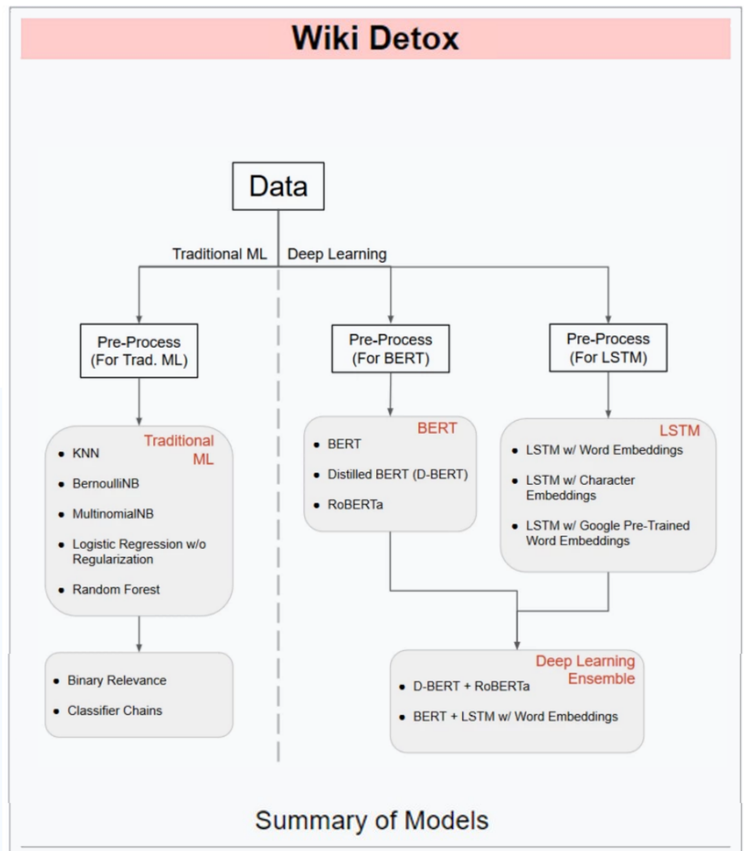
Being a large crowdsourced online repository of knowledge, **Wikipedia** boasts many articles with hundred thousands active editors on a daily basis. In order to coordinate editing efforts, each article has a discussion page where editors may discuss the information presented.

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.

Abstract

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)



Dataset [edit]

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 spammy
- '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

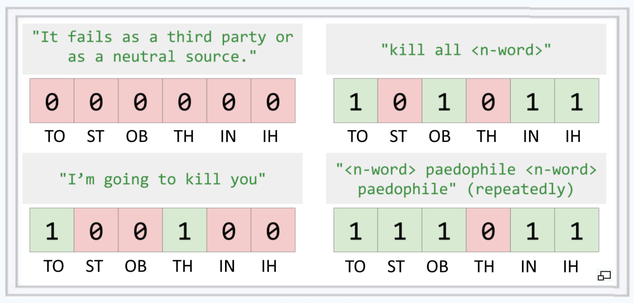


Figure 1: Labels of comments

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

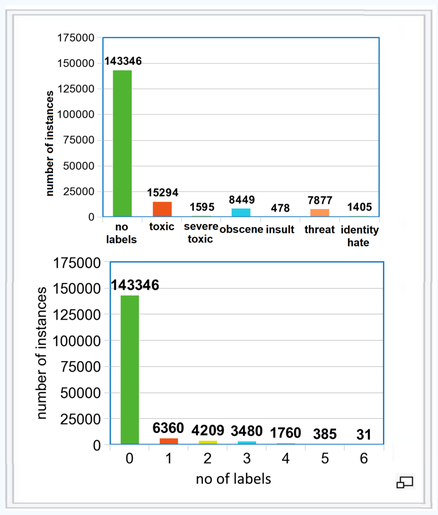


Figure 2: Distribution of labels

This 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**.

Moreover, we found that the **5 other labels were powerful predictors** for whether a comment was toxic. During our exploratory data analysis, we found that all 'severe_toxic' comments were also toxic comments. For each of the 'obscene', 'threat', 'insult' and 'identity_hate' comments, the vast majority of them were also labelled as 'toxic'.

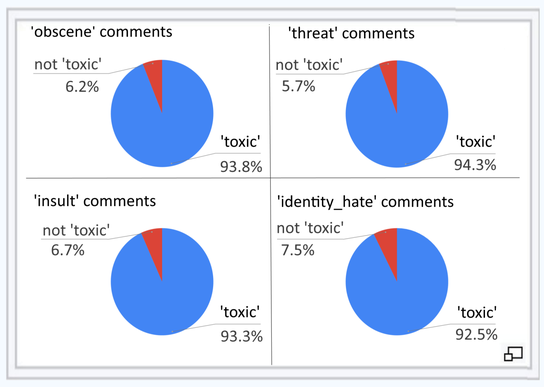


Figure 3: Relatedness of the labels to toxicitiy

Summary of Models

We began by training **5 Traditional Machine Learning models** to find a baseline model for the problem.

Then, we trained some **multi-output classifiers**, namely the Binary Relevance and Classifier Chains models to see how they would perform in comparison. We **tweaked the hyperparameters** to see how they could perform.

Finally, we trained a **few variations of Deep-Learning models** (LSTM and BERT) in order to find out how well they would perform.

We compared the performance between the different models to see which models worked best and why.

Traditional Machine Learning [edit]

Pre-Processing Pipeline (For Traditional ML)

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

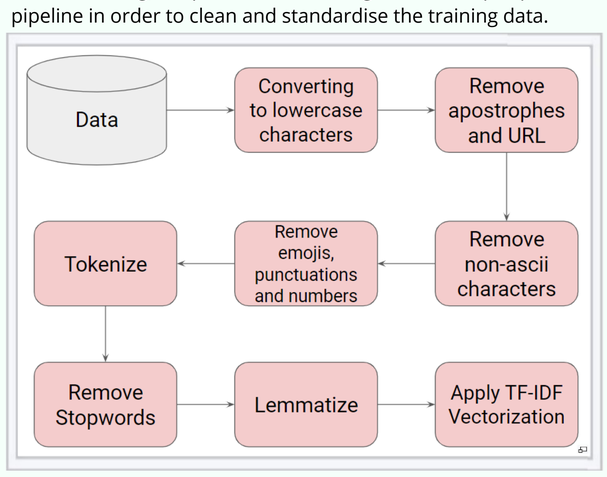


Figure 4: Pre-processing pipeline

Baseline Models

For our baseline model we attempted to create a model that **consists of 6 binary classifiers** (1 for each label) and merged their predictions to obtain a prediction for a single comment. This table shows the results of using various algorithms for each set of 6 classifiers.

Model used	AUC-ROC	Training Time
Log 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 5: AUC-ROC score and training time of Traditional Machine Learning models

We see that **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 the high dimensionality of the feature matrix which means the distance between neighbours was not meaningful for classification.

Logistic Regression (Varying Penalty & Regularization)

Seeing how well Logistic Regression performed, we decided to vary the loss function and regularization 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.

Model used	Loss Function	Regularization (C)	AUC-ROC	Training Time
LogReg	L1	0.01	0.70623	15.3
LogReg	L1	1	0.96783	371.4
LogReg	L1	100	0.94515	10120.6
LogReg	L2	0.01	0.95889	12.3
LogReg	L2	1	0.97024	14.7
LogReg	L2	100	0.95078	77.1

Figure 6: 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.

This is contrary to our initial hypothesis as we thought that due to the large number of features and typical keywords used in toxic comments (vulgarities, etc.), the inherent "feature selection" property of L1 would help the model perform well. It seems that there are **subtle toxic nuances** that can be expressed in comments than we initially thought.

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**.

SK-MultiLearn Library

We started by creating one Binary Relevance and one Classifier Chains models, each with **1000 features** generated from the TF-IDF vectorization.
We also made use of the popular sk-multilearn library, to create models based around Binary Relevance and Classifier Chains.
The initial results of this experiment are shown here:

Model Used	AUC-ROC	Training Time
Binary Relevance	0.78910	15.1s
Classifier Chains	0.94260	166.8s

Figure 7: AUC-ROC score and training time of Binary Relevance and Classifier Chains models

We attribute the disparity in performance to the **more accurate inductive bias of Classifier Chains** model as compared to the Binary Relevance model.

We attempted to tweak the hyperparameters of the Binary Relevance model, **retraining** it with a TF-IDF vectorization with **5000 max features**. While this lead to a vastly improved AUC-ROC score of 0.96, it **also greatly increased the training time** to 700s (~4500% increase). This increase in performance can be attributed to the greater expressive power of the model.

Deep Learning [edit]

LSTM

We then tried applying deep learning in order to get better results. We began by exploring another word representation known as **word embedding**. We decided upon a deep learning model suitable for NLP tasks. **Long short-term memory (LSTM)**, a special variant of Recurrent Neural Network (RNN), was chosen for experimentation in **3 variations** below.

LSTM with Self-trained Embedding:
Unlike RNN, LSTM has a cell state that transfers information throughout the processing of text sequence, hence it can preserve **long-term memory**. We built a baseline model which trains its own **Embedding layer** to convert words to **vector representations** and fed them into the LSTM layer where the main processing occurs.

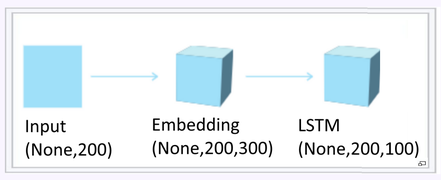


Figure 8: Embedding layer and LSTM layer

LSTM with Google NN Pretrained Embedding:
We explored **Word2Vec**, a powerful embedding technique which projects words onto a predefined dimension space such that **semantically similar words** will be **proximate** to each other. We replaced our own Embedding layer with a layer of **pretrained embedding weights** from the Google Negative News corpus.

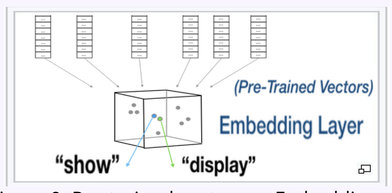


Figure 9: Pre-trained vectors as Embedding

LSTM with Character Embedding with CNN:
Wiki comments by nature tends to have a lot of **misspellings**. As such, we used **character vectors** in place of word vectors as most characters can still be **captured** in both the train and test sets despite the missing misspelt words. CNN is used to maintain a **reasonable dimensionality**.

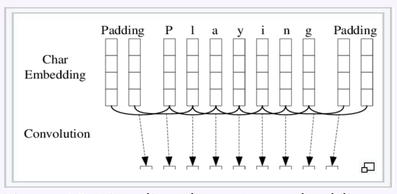


Figure 10: Visualize character embedding

Model	AUC-ROC	Training Time
LSTM w/ Word Embeddings	0.97388	3916s
LSTM w/ Google Pre-Trained Word Embeddings	0.97189	9148s
LSTM w/ Character Embeddings	0.97085	4480s

Figure 11: Comparison across models

Analysis: The **baseline** model performs best. We suspect the model with pre-trained embeddings did not improve performance due to a low number of **common word vectors** between Google's and our corpus.

Character embedding's poorer result indicates that our intuition on misspelt words might **not** be applicable to our dataset.

Ensemble methods / Deep Learning Ensembles

We performed an **ensemble averaging** of two DL models' outputs and in **both** cases shown above, we observed an **improvement** in result.

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

Figure 12 AUC-ROC score and training time of deep learning ensembles

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 performed better (BERT + LSTM). This shows that ensemble averaging can **average out** the **various errors** in each model.

BERT

The transformers used in BERT models allow an **entire comment** to be **processed at once**, instead of having the text **read sequentially**. This allows BERT-based models to preserve the **context** of each word based on **other words** in the sentence. For example, ambiguous sentence like "**I saw a man on a hill with a telescope**" has at least 5 valid interpretations:

- I saw the man. The man was on the hill. I was using a telescope.
- I saw the man. I was on the hill. I was using a telescope.
- I saw the man. The man was on the hill. The hill had a telescope.
- I saw the man. I was on the hill. The hill had a telescope.
- I saw the man. The man was on the hill. I saw him using a telescope.

By learning contextual relationships between words, BERT is able to outperform traditional machine learning models through its ability to **better understand ambiguous sentences.**

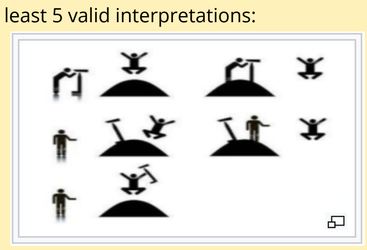


Figure 13: Different intepretrations of "I saw a man on a hill with a telescope"

Comparison between baseline BERT, DistilBERT and RoBERTa
DistilBERT and RoBERTa are both improvements of the original BERT model. DistilBERT applies **triple loss** combining language modelling, distillation and cosine-distance losses to better leverage the **inductive biases** learned by larger models.

RoBERTa leverages a robustly optimized BERT **pretraining approach**. Hence, both of these models are able to outperform BERT. Furthermore, through the use of **knowledge distillation** during the pre-training phase, the size of the DistilBERT model is reduced significantly, reducing the training time by a significant amount.

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

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

Conclusion/Analysis

In order to select a model for our task we decided to compare the models that gave the best AUC ROC score.

Comparing their AUC ROC scores, it is clear that the ensemble model performs the best at the expense of training time.

When we look at their F1 Scores we can see that the models generally perform poorly when trying to identify labels such as "severe toxic", "identity hate" and "threat". This is likely due to the **smaller training data** available for these labels.

However, we can see that Logistic Regression is comparatively better at identifying these labels.

Furthermore, Logistic Regression has the lowest training time among the models and the difference between the AUC ROC Scores is only ~0.008.

Finally, the Logistic Regression model is the easiest to interpret (as compared to deep learning models), explain and improve upon thus, we decided to select **Logistic Regression (w/ SAGA solver, L2 Loss, C = 1)** as our final model.

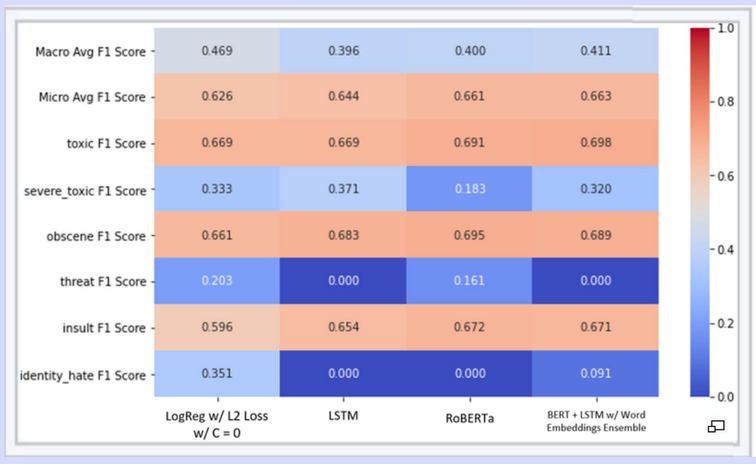


Figure 15: F1 scores of chosen models

References

Longtng. (2020, August 26). NLP Preprocessing & Feature Extraction Methods A-Z. Kaggle. https://www.kaggle.com/longtng/nlp-preprocessing-feature-extraction-methods-a-z.

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

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, É. (1970, January 1). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html.

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

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.

Szymański, P., & Kajdanowicz, T. (2017). A scikit-based Python environment for performing multi-label classification. arXiv preprint arXiv:1702.01460.

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.