Writeup (*Candidate ID: 10443)*

This report provides a brief overview over my attempts at this task over the course of the competition.

I. Data Exploration

Exploration was performed only on the training data and not on the test data to avoid human bias[1] in the modelling process. The main aim of data exploration was to understand patterns within the data that would play important roles in being able to predict whether a particular comment was a personal attack or not.

A. The prevalence of class imbalance

As specified above, any comment could belong to one of the two classes - Personal attack or Not an attack. Out of the 15000 samples in the training data, the ratio of samples that were not attacks to those that were attacks was 3.65: 1, or *for every comment that was an attack in the training set, almost 4 comments existed that were not-attacks* (figure 1).

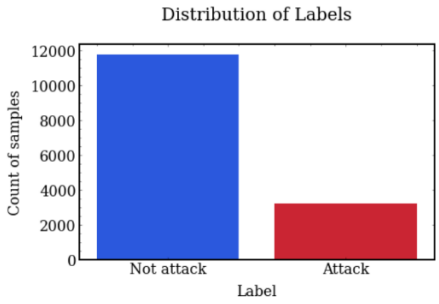

Figure 1. Class imbalance in training data

B. Understanding the most common used words

The simplest solution to identify whether a comment is an attack is to identify if it contains words that are generally used in comments that could be attributed to personally attacking language[2]. Wordclouds (Heimerl et al., 2014) are useful visualization techniques for the same. Figure 2 a) presents such an analysis. As expected, slurs and abusive language is very much prevalent in personal attacks. Comments that were not attacks were more inclined to having words that represented healthy discussion on Wikipedia Talks such as article, page, Wikipedia etc. The next step was to recreate the same word clouds as above, but this time with the exception of only using nouns and verbs from the comments (as in figure 2 b)). The level of granularity is more pronounced now and it were these word clouds that were used in the phase of feature engineering which is explained in the next section. The wordcloud analysis was followed by a manual inspection of the samples. An important find was that most comments that were attacks did not contain profanity. So, a dictionary approach with a list of profanities would not be helpful.

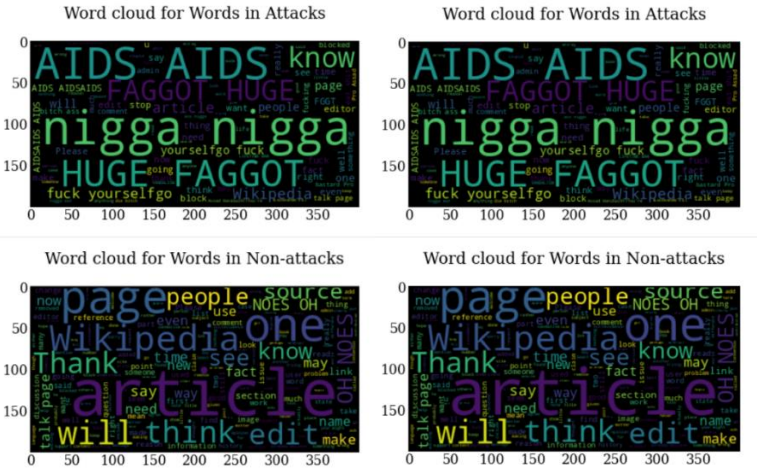

Figure 2. a) Most frequent words per class

[1] Exploring the test data could cause me to write code to include patterns in the test data, but real-world ML does not offer this luxury

[2] This is often how human beings learn to discern between good language and bad language when we first start learning

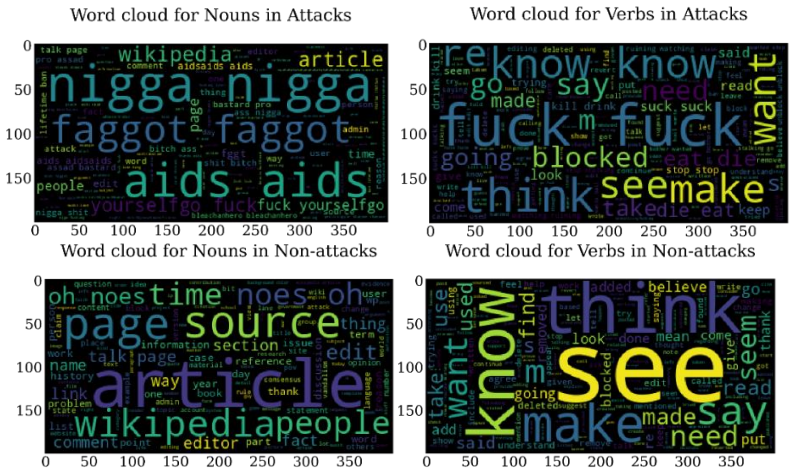

Figure 2. b) Most frequent nouns and verbs per class

II. Modelling

All models built during the course of the contest were built keeping in mind reproducibility and ease of updating. Both machine learning and deep learning approaches were used.

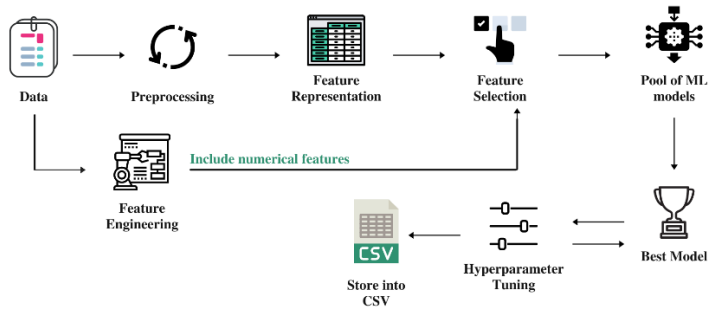

Figure 3. The machine learning approach

A. Machine learning

Figure 3 is a high-level representation of the machine learning approach. All 10 models created followed this. Scikit-learn (Pedregosa et al., 2011) was used to build end-to-end pipelines for the classification task. Pipelines were used in order to avoid data leakage from test data to train data and to also aid with reproducibility.

1. Preprocessing

Dealt with consideration of whether to convert the text into lowercase, how to tokenize text or whether to remove stopwords. Lowercase conversion was useful, but it was more useful to also keep record of capital words in the comment. Stopword removal was surprisingly ineffective as it dropped performance. The scikit-learn default tokenizer, NLTK tokenizer (Loper and Bird, 2002) (with stemming) and the fast.ai (Howard and Gugger, 2020) tokenizer[3] were used. The fast.ai tokenizer performed the best.

2. Feature engineering

Feature engineering is the process of creating new features out of the existing ones in the dataset. In this training data, the only real feature (or exploratory variables) is *text*. I computed 29 new features grouped into 3 broad categories from the text of each comment. These features were created based on general practices in quantitative text analysis, outputs of data exploration as well as on my personal hunches (see table I). The newly computed features are grouped into 3 categories

- Descriptive text features (DTF): These features follow from quantitative content analysis principles (Krippendorff, 2018)
- Specific word presence (SWP): These features take on value 1 if a specific word exists, or 0 if it does not

[3] This tokenizer incorporates a series of rules, including keeping track of capital characters, beginning of sentences etc.

- Intuition based features (IBF): These features were those formed on top of my hunches

Table I. Engineered Features

Name of feature	Type of feature	Rationale for construction	Category of feature
Character count	Interval	Quantitative content analysis	DTF
Word count	Interval	Quantitative content analysis	DTF
Syllable count	Interval	Quantitative content analysis	DTF
Sentence count	Interval	Quantitative content analysis	DTF
Mean words per sentence	Interval	Quantitative content analysis	DTF
Mean characters per word	Interval	Quantitative content analysis	DTF
Mean syllables per word	Interval	Quantitative content analysis	DTF
Flesch reading ease score	Interval	Hypothesizing that attackers don't have time to make their comments readable	DTF
Automated readability index	Interval	Hypothesizing that attackers don't have time to make their comments readable	DTF
Dale Chall readability score	Interval	Hypothesizing that attackers don't have time to make their comments readable	DTF
Count of nouns	Interval	Quantitative content analysis	DTF
Count of pronouns	Interval	Quantitative content analysis	DTF
Count of punctuations	Interval	Quantitative content analysis	DTF
Count of digits	Interval	Quantitative content analysis	DTF
Count of capital words	Interval	Hypothesizing that people use CAPS when they are angry more than when they are calm	DTF
Count of verbs	Interval	Quantitative content analysis	DTF
Presence of "fu**"	Binary	Wordcloud analysis output	SWP
Presence of "n***"	Binary	Wordcloud analysis output	SWP
Presence of "f*g"	Binary	Wordcloud analysis output	SWP
Presence of "aids"	Binary	Wordcloud analysis output	SWP
Presence of "article"	Binary	Wordcloud analysis output	SWP
Presence of "wiki"	Binary	Wordcloud analysis output	SWP
Presence of "page"	Binary	Wordcloud analysis output	SWP
Presence of "block"	Binary	Wordcloud analysis output	SWP
Presence of "source"	Binary	Wordcloud analysis output	SWP
Presence of "REDIRECT"	Binary	Wordcloud analysis of only CAPITAL WORDS	SWP

Sentiment score	Interval	Hypothesizing that personal attacks are often negative in their sentiment than non-attacks	IBF
Count of "you"	Interval	Hypothesizing that to be a personal attack, there has to be the use of "you" or its derivatives such as "yourself", "your" etc.	IBF
Count of "!"	Interval	Hypothesizing that when people are angry, they tend to use more exclamation marks than when they are calm	IBF

1. Quantitative content analysis concepts used for computing features were acquired from the material of MY459 at LSE
2. SWP's were computed considering both uppercase and lowercase counts
3. Features included in the final model are in blue

Statistical analysis was later used to create a much smaller set of features that could help most accurately classify a given comment as being an attack or not. This statistical analysis was conducted only on these newly computed features that helped *quantify the content of the comments*. T-test for equality of means was used across all the computed interval features and only statistically significant features were kept. The binary features were visually inspected (figure 4) and promising ones were added to the list of useful features. A simple naive bayes model was fit and the combination of features were tweaked till the best combination emerged.

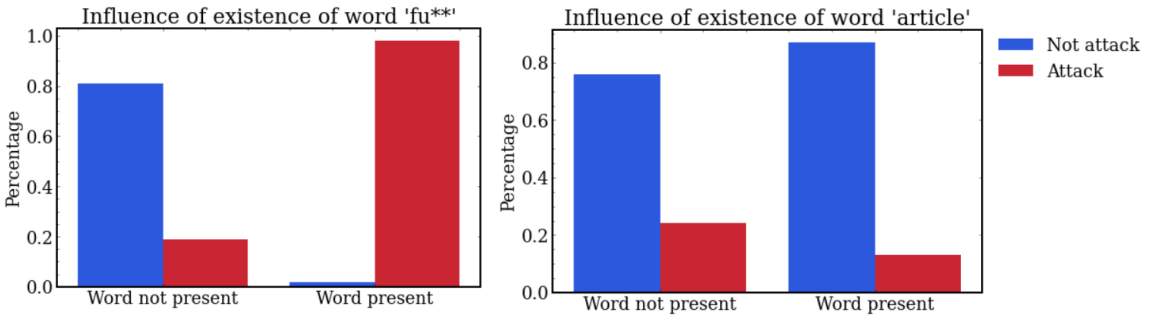

Figure 4. A couple of binary features that were examined

3. Feature Representation

Table II describes in brief the feature representations I used for the ML models.

Table II. Feature Representation

Representation	Brief Explanation
Bag of words (BOW)	Simple matrix representation of textual data where each row represents a single document (or comment in this case) and each column represents a unique word in the vocabulary, where the vocabulary contains all unique words used across all comments in the dataset. Each cell is given a value equal to the number of times a word j appears in a document i.
Bag of words with Tf-idf weighting (Tf-idf)	Similar to BOW, but weights each word on the basis of how rare it is across the corpus and how prevalent it is in individual comments

There were not very large differences between the use of both representations[4] if the rest of the modelling pipeline was kept unchanged. In theory though, Tf-idf is better than BOW as BOW tends to lay too much importance on words that are very frequent across all comments (Huilgol, 2020). In reality, words with such high frequencies can be treated as stopwords. This logic was used to mask high frequent words in the training set with a STOPWORD label[5]. This code is presented in the translate_text() function in the preprocess.py script in ../code/src. However, the masking didn't improve performance.

4. Feature selection

A challenge with textual data is that its feature representations consist of hundreds of thousands of features. Having such high-dimensional data can lead to curse of dimensionality problem (Köppen, 2000). Feature selection techniques were used to reduce the dimensionality of data and this resulted in better performance.

[4] Only unigrams seemed to improve performance in the initial runs, so only unigrams have been considered
[5] Based on the ambitious idea that all stopwords play the same role in speech

Feature selection of the text representation was performed by using univariate feature selection (UFS). UFS scores features on univariate statistical tests based on statistical significance of the association between an explanatory variable and the response(target) variable, and keeps only the top K features or features in top K percentile.

5. Pool of ML models

Classification models such as complement naïve bayes, multinomial naïve bayes, decision trees, random forests, logistic regression and support vector machines were used[6]. Their default implementations and default parameters from the scikit-learn library were used to compare the models against each other. Complement NB performed the best and tree-algorithms were the worst. This could be because of the curse of dimensionality problem as the feature representations of text data include very large sparse matrices[7], even after feature selection. Reducing the features by too much could be dangerous as doing so could blind side the model from certain words. I took a conscious decision to focus on the data processing part and only use a complement naïve bayes model from the ML approaches. Also, the complement naïve bayes runs very quickly and that was an added advantage in quickly experimenting with different data processing techniques.

6. Hyperparameter tuning

The best model out of the pool of models, i.e including the data processing choices was run through a grid search cross validation[8] process for fine tuning the model with respect to its hyperparameters. The best model is Model 8 in the my474_models.xlsx file in ../code/models. A complement NB model was used and therefore, the only hyperparameter that was searched was alpha. Alpha determines the extent of smoothing required and helps prevent the zero-probability problem in NB algorithms (Shimodaira, 2014). Tuning was also used to decide the optimum K for top K percentile of features to be selected in the feature selection step. The search space was alpha = [2, 1.5, 1, 1e-1, 3e-1, 5e-1, 1e-2, 3e-2, 5e-2, 1e-3, 3e-3, 5e-3, 0] and K = [1, 2, 5, 10, 15, 20, 50]. The results of the procedure are stored in ../code/models/hyp_opt_ml.csv .

B. Deep learning

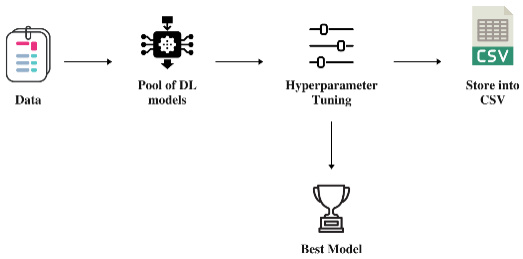

Figure 5. The deep learning approach

The deep learning models used in the competition were mainly based on transfer learning. Transfer learning is a subset of deep learning where models are used in tasks they were not trained explicitly to work with. In the context of NLP models (text), this pre-training step involves providing the model the ability to understand the "rules" of usage and context of language, also called self-supervised learning (Howard and Gugger, 2020). The knowledge obtained in pre-training is used in solving the problem at hand, in this case text classification.

1. Pool of DL models

I have used two deep learning techniques - The ULM-FiT(Universal Language Model Fine-tuning) approach and RoBERTa(Robustly Optimized BERT pre-training approach). The ULM-FiT approach (Howard and Ruder, 2018) involves fine-tuning a pretrained model (on Wikipedia English) with the training data of the contest. The output of this fine-tuning is a model that can predict the next token in a sentence. Naturally, if it were to make such a prediction, it is important that the model is able to discern patterns of language use. Therefore, the fine-tuned model is used to build a classifier using LSTM (Long Short-Term Memory), a Recurrent Neural Network approach that treats text as a sequence of tokens across a time frame. I used the fast.ai library to implement ULM-FiT (Howard and Gugger, 2020).

RoBERTa is a faster enhancement to the BERT (Bidirectional Encoder Representations from Transformers) architecture (Liu et al., 2019). A drawback of RNNs is that tokens that are far apart from each other are not easily considered to be related even if they do. BERT works around this by using a transformer that learns to

[6] For training, a 5-fold stratified cross validation scheme was setup
[7] Sparse matrices have lots of 0s
[8] Grid search is time-consuming but given the relatively fewer number of hyperparameters to tune, it was used

capture relations between tokens that are far apart in a text or sequence. Further, BERT uses a masked language model to apply a mask on 15% of tokens in a sequence and then predict the masked tokens. This is again analogous to learning the patterns and structure of language. BERT also captures context in both the forward as well as reverse order in a sequence. This pre-trained model is then used for a classification task. In the model I built, I used the Huggingface roberta-base pretrained model (trained on 160GB of English text) (roberta-base; Hugging Face, 2021). I used the simpletransformers library to implement RoBERTa (Rajapakse, 2021).

2. Hyperparameter tuning

It was performed on only the RoBERTa model and was used to identify the optimum number of epochs and learning rate for the neural network. The weights and biases was used to monitor the hyperparameter performance API (Weights & Biases – Developer tools for ML, 2021). Randomized search[9] was used across the search space of epochs = [1, 2, 3, 4, 5] and learning rate between 0 and 4e-5. The results of the procedure are stored in ../code/models/hyp_opt_transformer.csv

III. Models Submitted to Kaggle

Table III. Models submitted to the contest

Name of Model	Model specifics	Mean CV(5-fold) F1 on train data (%)	F1 on public leaderboard (%)
Model 1 (ML)	Preprocessing(lowercase, scikit-learn tokenizer), BOW, no feature selection, multinomial NB	62.77	63.36
Model 2 (ML)	Preprocessing(lowercase, lemmatization tokenizer), BOW, no feature selection, complement NB	65.10	64.28
Model 3 (ML)	Preprocessing(lowercase, snowball stemming tokenizer), Tf-idf, univariate feature selection with chi2, complement NB	66.37	66.75
Model 4 (ML)	Preprocessing(lowercase, snowball stemming tokenizer), Tf-idf, include numeric features, univariate feature selection with chi2, complement NB	67.72	68.41
Model 5 (ML)	Preprocessing(lowercase, fast.ai tokenizer), Tf-idf, include numeric features, univariate feature selection with chi2, complement NB	68.05	69.08
Model 6 (DL)	ULM-FiT transfer learning approach, tokenized using fast.ai tokenizer	71.34	73.00
Model 7 (DL)	RoBERTa transformer approach, tokenized using the default RoBERTa tokenizer that treats spaces like part of the text	76.15	78.28

1. Only those models that have some substantial differences are presented in this table
2. ../code/models contains all model performances.
3. In the case of DL models, the CV F1 is basically a hold-out F1 score
5. The models selected for scoring on the private leaderboard are in red

IV. Reflections

The creation of models in machine learning is a hard task. From the initial stage of data processing to the final stage of hyperparameter tuning, a modelling pipeline contains several components that are instrumental for the development of a "good" model.

Text classification is a very challenging problem. It's technical difficulties include the large number of features leading to curse of dimensionality, implicit problems with text such as spelling errors and in this case, the problem of class imbalance (Collins, Rozanov and Zhang, 2018). The practical difficulty of text classification, especially a problem such as the one in this competition is "how accurate are human labellers"? For example, a personal attack is best understood by the person who it was intended for. Also, not all attacks are of the same malicious level. Some need to be dealt with more seriousness than others.

[9] Randomized search used here as training transformer-based models are very time consuming

I built both ML and DL models for the competition and was able to understand the differences between the use of the two kinds in a practical setting. While the DL models performed much better than the ML models in terms of attained F1, it was the ML models that were quicker to train and easier to interpret. The difference in the F1 score between the best DL and ML model is about 9% while the difference in their accuracies is under 5%. But the inference (prediction) time of DL is 5 times the inference time for ML for the 100000 test samples.

The engineering question now is whether the finally deployed model is to be faster or more accurate by 5%. Considering how people don't prefer the long wait more often than not, the deployment of the ML model would seem like a better choice.

V. References

1. Collins, E., Rozanov, N. and Zhang, B., 2018. Evolutionary data measures: Understanding the difficulty of text classification tasks. *arXiv preprint arXiv:1811.01910*.
2. Heimerl, F., Lohmann, S., Lange, S. and Ertl, T., 2014, January. Word cloud explorer: Text analytics based on word clouds. In *2014 47th Hawaii International Conference on System Sciences* (pp. 1833-1842). IEEE.
3. Howard, J. and Gugger, S., 2020. *Deep Learning for Coders with fastai and PyTorch*. O'Reilly Media.
4. Howard, J. and Gugger, S., 2020. Fastai: A layered API for deep learning. *Information*, *11*(2), p.108.
5. Howard, J. and Ruder, S., 2018. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*.
6. Huggingface.co. 2021. *roberta-base · Hugging Face*. [online] Available at: https://huggingface.co/roberta-base [Accessed 16 May 2021].
7. Huilgol, P., 2020. *BoW Model and TF-IDF For Creating Feature From Text*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2020/02/quick-introduction-bag-of-words-bow-tf-idf/ [Accessed 13 May 2021].
8. Köppen, M., 2000, September. The curse of dimensionality. In *5th Online World Conference on Soft Computing in Industrial Applications (WSC5)* (Vol. 1, pp. 4-8).
9. Krippendorff, K., 2018. *Content analysis: An introduction to its methodology*. Sage publications.
10. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L. and Stoyanov, V., 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.
11. Loper, E. and Bird, S., 2002. Nltk: The natural language toolkit. *arXiv preprint cs/0205028*.
12. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J., 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, *12*, pp.2825-2830.
13. Rajapakse, T., 2021. *Simple Transformers*. [online] Simple Transformers. Available at: https://simpletransformers.ai/ [Accessed 16 May 2021].
14. Shimodaira, H., 2014. Text classification using naive bayes. *Learning and Data Note*, *7*, pp.1-9.
15. Wandb.ai. 2021. *Weights & Biases – Developer tools for ML*. [online] Available at: https://wandb.ai/site [Accessed 17 May 2021].