Classifying Offensive Language and Hate Speech in Tweets

NTU-AI6127: Project Final Report

CHAN CHEONG **CHIA BOON YANG** **DENG XINHONG**

KRONBORG MAXIMILIAN **PENG WEI-LUN**

School of Computer Science and Engineering
Nanyang Technological University
{CHAN0987, BCHIA012, XDENG003, KRON0001, PE0001UN}@e.ntu.edu.sg

Abstract

The prevalence of offensive language and hate speech in online discourse has a number of negative effects, that can facilitate or drive racism, violence, bullying, and extremism. We aim to use various techniques to improve upon the state of the art in identifying tweets that fall into such categories, in order to better censor and void such language. We use a number of techniques to produce effective classification on the **hate speech and offensive language** dataset, consisting of a selection of 25,000 tweets. These include pre-processing, contrastive learning, ensemble modelling, two-stage classification, as well as one-versus-rest classification. We find the state of the art difficult to comprehensively beat, though we do confirm various techniques that can improve upon this, achieving an F1 score of 93.20% up from 92% in the state of the art. In addition, we proposed a CLS-Last-Attention model that can spot out the hate and offensive word without using token-level supervision.

1 Introduction

The prevalence of offensive language and hate speech on social media has profound negative effects. They discourage civilised debates on important topics, and instead encourage disagreements, physical and mental abuse, and increase occurrences of real life racism and violence. Furthermore, the constant exposure to offensive and racist language may lead to people taking up more extreme viewpoints, and this can be especially harmful to younger people who are still forming their perspectives and opinions of the world, and are therefore more impressionable.

For this reason, it is becoming increasingly important to be able to automatically identify such language, such that it may be either censored, or removed entirely. Doing so can be extremely difficult, in particular with hate speech. While it is somewhat easy to identify most offensive language based on certain words, most often swearwords, hate speech is much more complex than that.

Hate speech is defined as: *language that is used to expresses hatred towards a targeted group or is intended to be derogatory, to humiliate, or to insult the members of the group* [1].

As is evident from this definition, the key to identifying such language lies in the ability to understand the language itself. While racial slurs is an easy way to identify hate speech, it is also extremely easy to create hate speech without any racial slurs or swearwords. This means that in order to carry out this task successfully, a model will need to be able to learn such intricacies, and this is where the real challenge lies in this problem.

2 Related Work

We base some of our work on the original paper by [1], which is where the dataset we use originated. We aim primarily to take inspiration from their analysis of the dataset, and less so from their modelling approach, which is not based on Deep Learning. They make use of logistic regression to first reduce the dimensionality of the data. After this, they employ a number of methods highlighted in their research on preceding work in the space. They found best performance in the use of a Logistic Regression classifier, and the use of Linear SVM. In the end, their results are based on Logistic Regression using L2 loss, on a one-versus-rest basis. They find that their best performing model achieves an overall precision of 0.91, a recall of 0.93, and an F1 score of 0.90.

In addition to this, we base a lot of our deep learning approach on the work done by [2]. They make use of a transfer learning approach through the use of a pre-trained BERT model, trained on English Wikipedia and BookCorpus to carry out this task. They modify this architecture through the addition of nonlinear layers, Bi-LSTM layers, and CNN layers. The result of this work is an improvement of the previous work by achieving an F1 score of 0.93.

3 Method

3.1 Pre-processing

As is commonplace in NLP tasks, a text based dataset will usually need some amount of pre-processing. To some extent, our model choice eliminates the final steps required, since the BERT model will convert the data from text into vector embeddings, and into the format required for the transformer-type architecture.

This leaves us with the task of ensuring the input is as rich in information as possible, and as devoid if unnecessary or distracting information as possible. In addition to this, we wish to make the input as understandable to the model as possible.

For this reason, we approach the pre-processing in two stages, and evaluate three possible combinations using our baseline models, using no pre-processing, using the first stage of pre-processing, and finally, using both stages of pre-processing.

This will allow us to verify if there is in fact a benefit to performing this pre-processing, in particular since we explore the use of models that have been pre-trained on twitter data, in which case we might expect them to have no problem understanding the format and type of language used on twitter.

3.1.1 Regular Pre-processing

The first stage of pre-processing we apply is based on regular approaches taken in pre-processing text-based data of an irregular format such as that which occurs on twitter.

Tweets, or indeed any piece of text generated by users without any particular formatting or spelling rules or conventions, can be quite difficult to standardise. Text will include misspellings, emojis, usernames, URLs, various types of slang, and acronyms.

Correcting these these is quite difficult, but there are a number of steps that can be taken:

- Remove hashtags by removing the '#' character, leaving behind a combined string of multiple words, which can later be split into constituent words
- Remove words that start with '@' to eliminate usernames, and any words with 'https' or 'www' to remove URLs
- Remove words such as 'RT', standing for retweet, which has no importance for the classification task, but which is a twitter convention
- Remove exclamation marks, question marks, ambersands, full stops, commas, emojis, and numbers etc.

Carrying out these corrections will leave behind simply the pure text, though as discussed, when this is user generated without any particular rules, it can still be quite difficult to standardise.

As a step in the above pre-processing, we use the **contractions** library for python to fix contractions. This needs to be done before apostrophes are removed, reducing the number of words that are broken into 'shouldn' and 't' for example, and instead ensuring it can become 'should' and 'not'. This serves to reduce the number of meaningless words the model sees, ensuring each word has more value in determining the meaning of the sentence.

3.1.2 Spell Correction

The next step of pre-processing is to correct spelling mistakes as much as that is possible. This is an extremely difficult task to perfect, since some misspelled words may end up being spelled the same as a correct word. Additionally, it may also be the case that a misspelled word could map to multiple potential corrections, which even after receiving context can be hard to determine. In a sentence about animals, it would be hard to know if the word 'bat' should really have been 'cat', and if it was instead misspelled as 'nat', knowing which of the two to correct it to might still be quite difficult.

An additional benefit to using this approach is that the hashtags that we strip of the '#' character in the previous stage will now be split into its constituent words, depending on the quality of the spell correction. However, this means that '#goodtimes' will be converted into 'good' and 'times'.

After testing a number of solutions, some of which would correct far too many correctly spelled words, we found that the **enchant** python library provided reasonably good performance.

3.2 CLS-Last-Attention

When classifying a sentence to be one of the classes, one would also want to know which word causes the model to make such a decision. When doing classification using BERT, only the classification token ([token]) is used to do sentence-level classification [3]. The ideal situation would be to feed the rest of the tokens to MLP layers to classify on each token. However, this is not possible, as the data set does not contain such a token-level label. In other words, we cannot explicitly teach the model which word is considered to be 'hate' or 'offensive'. In this project, we proposed a model (CLS-Last-Attention, Fig. 1) that can locate the 'hate' and 'offensive' words without having token-level supervision. The proposed model uses a pre-trained BERT as the backbone and it contains two extra components, layer-wise attention, and final CLS attention. For the layer-wise attention,

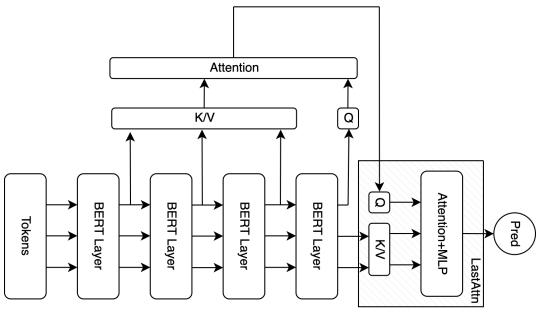

Figure 1: CLS-Last-Attention

consider that there are l transformer layers in the pre-train BERTed. The [CLS] token from the last layers will be used as *Query* and the [CLS] tokens from layer 1 to layer $l-1$ will be used as *Key* and *Value*. Multi-head Attention layer is applied to *Query*, *Key*, and *Value*; it will output the a $(N, 1, E)$ tensor, denoted as *Final Query*. Here, N is the batch size and E is the embedding dimension. This layer-wise attention block helps the model to look through layer-dimension and gather information from all the previous transformer layers. In this work, we only use the first 5 layers of BERT (F1:90.2) as we found that it can already surpass the performance of the 12 layers full BERT (F1:89.8).

For the final CLS attention block, consider that the output dimension of BERT is (N, T, E) where T is the sequence length that includes the [CLS] and [pad] tokens. All the $T-1$ (except [CLS]) tokens will be used as *Final Key* and *Final Value*. The trick of the CLS-Last-Attention is to perform Multi-Head Attention on the *Final Query*, *Final Key*, and *Final Value* before the classification layer. Note that all the tokens that correspond to the input [pad] token will be masked in the Multi-Head

Attention layer as we want the model to only focus on the useful tokens. The output of this Multi-Head Attention layer is a $(N, 1, E)$ tensor and will be flattened to (N, E); this output will then pass to an MLP layer with ReLU non-linearity and dropout, to perform classification. This final CLS attention block helps the model to look at the sequence-dimension and generate attention weight on all the tokens. This attention weight helps us to provide a location map to show which token is considered as 'hate' and 'offensive'.

3.3 Ensemble BERT

During the exploration of BERT backbone, we found there were several pre-trained BERT models on abusive language phenomena. HateBert [4] is obtained by re-training BERT-base-uncased with manually selected Reddit comments. While fBERT [5] uses SOLID dataset which covers a wider range of offensive languages, such as cyberbullying and profanity. Given this difference in the pre-trained model, we tried to ensemble these retrained BERT to obtain a diverse view of our dataset. We followed one of the ensemble BERT methods described in [6] for the implementation, which is to average the output from the retrained BERT models and feed the average output to the down stream model CLS-Last-Attention. Because our dataset is relatively small, we freeze the weights of the BERT models during training. Since the two retrained models are both based on BERT-base-uncased, they have the same vocabularies and the same architectures as the BERT-base-uncased. This justifies the correctness of averaging the BERT output.

3.4 Contrastive Learning

Contrastive learning is a method that contrasts the input sentence with other sentences. The goal of contrastive learning is to let those sentences belonging to the same class can be closer in the embedding space. Therefore, the model can learn the relationships between sentences. Generally, the state-of-the-art methods follow two tuning stages to fit the specific task. The first is to pre-train the model on an extensive dataset and finetune it in a particular task. Nonetheless, the cross-entropy loss can lead to suboptimal and unstable results. We follow the intuition from Gunel et al.[7] that good generalization requires capturing the similarity between examples in one class and contrasting them with examples in other classes. In addition, we follow the finetuning structure from Khosla et al.[8] that splits the training session into two stages.

$$L_{InfoNCE} = -log\frac{exp(A \cdot P/\tau)}{exp(A \cdot P/\tau) + \sum_{i=0} exp(A \cdot N_i/\tau)}$$

In stage one, for each input sentence, we sample one positive sample and several negative samples. After going through the model, we will get the embeddings and compute the InfoNCE loss. InfoNCE can calculate the similarity in the embedding space by inner product, where A is the input sentence, P is the positive sample, and N is the negative sample from the training set. τ is a temperature hyperparameter that controls the separation of classes. During this process, we select the pre-trained BERT model from the Huggingface library and freeze the parameters of BERT.

In stage two, all the trained parameters in stage one are frozen. In addition, we add the multilayer perceptron (MLP) at the end of the model structure and fine-tune it for the classification tasks. More details of the structure is shown in the appendix (Sec. 7.3) section.

3.5 Binary Classifiers

Using a single neural network model to perform multiclass classification with an unbalanced dataset might result in poor predictive performance for the minority classes. We propose the use of binary classification methods instead to attempt to improve the performance of the original multiclass classification problem. The illustrations of the proposed two methods can be referenced in figure 12 and 13.

3.5.1 Two Stage Binary Classification

In this method, a tweet is predicted by a first model to be class 0 or class 1, where class 0 constitutes tweets labelled neither hate speech nor offensive language, and class 1 constitutes tweets that are

labelled either hate speech or offensive language. If neither is predicted by the first model, it is the final prediction. If either is predicted by the first model, the same tweet is predicted by a second model to be class 0 or class 1, where class 0 constitutes tweets that are labelled as hate speech, and class 1 constitutes tweets that are labelled offensive language.

For the first model, the target class of a tweet is pre-processed by combining the hate speech class and offensive language class into a single class, and keeping the neither class intact. For the second model, within the partitioned training, validation and testing sets, tweets with the neither class are dropped. The overall two stage binary classification is evaluated with the original partitioned testing set without any modifications.

3.5.2 One-Versus-Rest Classification

In this method, a tweet is predicted by three binary classifiers, and the predictions are combined using majority voting to yield a final prediction.

Model one predicts a tweet to be either the hate speech or offensive language class, or neither class. Model two predicts a tweet to be either the hate speech class, or others class. Model three predicts a tweet to be either the offensive language class, or the others class. Prior to training of the three models, the same data pre-processing to the target class labels of the tweets (just like in model one of the two stage binary classification) are performed. The predictions from each of the three models are combined using majority voting, and randomness is used to settle ties in voting scores.

4 Experiments

In order to improve upon the performance of the state of the art on the selected dataset, we took multiple approaches, and assessed their efficacy, in order to understand how to best maximise performance in this space.

The **Hate Speech and Offensive Language** dataset was used in this project[1]. The dataset contains 24,763 tweets, each tweet is labelled as 'hate', 'offensive' or 'neither'. Each tweet is labeled by at least 3 people, and the majority vote of the three is what is used to classify each sample. 1,425 (6%) of them are labeled as 'hate', 19,181 (77%) are labeled as 'offensive', and 4,157 (17%) are labeled as 'neither'. As such, this is a very imbalanced dataset and in particular those data instances labelled as 'hate' will be a challenge to classify correctly, due to the deep understanding of the language required to identify it, as well as the lack of samples of this type in the dataset.

Due to the imbalanced nature of the dataset, and the fact that it is a multi-class classification problem, we determine the most appropriate evaluation metric to be the **weighted F1** score. This is also a primary metric used in the two papers that we base this work on, facilitating comparisons.

4.1 Implementation Details

The baseline model is the CLS-Last-Attention (Sec. 3.2). This model uses a pre-trained BERT model, hateBERT [4]. Only the first 5 transformer layers are used. The dropout probability of hidden BERT layers are set to 0.2, while the dropout probability of the MLP classifier is set to 0.1. Adam optimizer is used with a learning rate or $1e^{-4}$, betas of $(0.9, 0.999)$, and a weight decay of $1e^{-3}$. The data is split into train(80%), val(10%), and test set (10%). The model is trained for 20 epochs on the train set and evaluate on the val set. The best model obtained in the val set is used on the test set. The metric obtained in the test set is reported in this report.

4.2 Pre-process

We tested the pre-processing techniques used on the performance of the baseline CLS-Last-Attention model, using a number of pre-trained model types. The primary variation here is which dataset they were trained on.

- **BERT-base-uncased** - a BERT model trained on English language using masked language modelling [9].
- **twitter-roberta** - A roBERTa model trained on roughly 58 million tweets and finetuned for emotion recognition using TweetEval Benchmark [10]

- **bertweet** - RoBERTa pre-training based model trained on 850M english tweets [11]
- **hateBERT** - pre-trained BERT model, further trained on 1 million banned reddit posts [4]

Pre-trained model	No pre-process (%)	Pre-process (%)	Pre-process + spelling (%)
BERT-base-uncased	90.21	90.77	90.77
twitter-roberta	89.79	90.58	91.02
hateBERT	90.22	91.00	90.89
bertweet	**90.87**	**91.07**	**91.15**

Table 1: Pre-processing F1 scores

As shown in table 1 the results indicate that there is some improvement to be found in increased levels of pre-processing, though the difference is not particularly pronounced, most likely because the pre-trained models were trained on data of the same format, meaning they are not as affected by the irregularities as the models used in [1].

Next we perform a qualitative analysis of some of these results in order to illustrate the effects of the pre-processing.

These are shown in figures 2, 3, and 4, and consist of confusion matrices for the test-set predictions over the three different levels of pre-processing using the BERT-base-uncased pre-trained model. We are primarily interested in any improvement to the accuracy on the hate-speech, and potentially even more so in reducing the cases of hate speech that are classified as neutral. If an instance of hate speech is classified as offensive, this is a less severe mis-classification.

We only see small changes here, but we can identify 2 extra cases of hate speech being correctly classified using the full pre-processing pipeline. However, it seems that this comes at a cost, as both slightly fewer cases of offensive language and neutral are classified correctly.

4.2.1 Pre-processing Shortcomings

A number of challenges still remain in perfecting the spell correction. The primary one is that there are still a number of misspelled words that do not get corrected. For example such words as 'yess' are corrected to 'yetis' instead of 'yes'. This means that although the text should have been standardised to some extent, there might still be confounding errors in the text that hinders model performance. Additionally, there is the risk that some correct, or nearly correct words are converted into words that carry the wrong meaning in the context, potentially degrading model performance.

4.3 Ensemble BERT

We run experiments on ensemble BERTs and compare them with the baseline. The experiment configuration is the same as the baseline, except for the ensemble structure. There is no huge difference in the three metrics between the baseline and the ensemble BERTs method. It is also noticeable from figure 6 the ensemble BERT method overfits after epoch 3, which may be due to our small dataset size.

Model	F1-Score(%)	Precision(%)	Recall(%)
hateBERT + LastAttn (Baseline)	90.22	90.20	89.81
ensemble BERTs + LastAttn	90.08	89.80	90.60

Table 2: Comparison for Ensemble BERTs. Here LastAttn refers to CLS-Last-Attention

4.4 Contrastive Learning

We compare the two stage contrastive learning with directly tuning the models using the same structure. The best f1-score from the contrastive learning is 90.0%, while model with directly tuning can only reach 89.3% The f1-score only improves by around 0.5%. A possible reason is that there

is still a common problem with how to sample the hard negative samples for the model to learn for contrastive learning.

For this problem, some proposed a memory bank to store all the embeddings during training to find those hard negative tasks. However, this method takes a lot of disk memory and time to choose the proper samples. Another approach is to choose a big enough dataset and select those hard negatives within the sample set. Nevertheless, it requires a lot of GPU resources due to the large batch size. To solve the problems above, we came up with a method of sampling the top-K individual loss of the batch and let the average of top-K individual loss represent the whole batch loss. Hopefully, the loss will concentrate on those challenging tasks and ignore those easy tasks. However, due to the limited time, we have not tried much on this proposed method, and it can be a future work of this section.

4.5 Ensemble

In order to further boost the performance of the model, we tried to ensemble all the tested models. We reimplement the two models (hateBERT$_{base}$ + CNN and hateBERT$_{base}$ + LSTM) that are proposed in [2] instead of using their results directly. This is for a fair comparison as the experiment setting might be different between us and them.

Model	F1-Score(%)	Precision(%)	Recall(%)
pure hateBERT [4]	89.3	89.0	90.8
hateBERT$_{base}$ + CNN [2]	88.9	88.6	89.5
hateBERT$_{base}$ + LSTM [2]	90.0	89.8	90.4
hateBERT$_{base}$ + CLS-Last-Attention	90.2	90.2	89.8
hateBERT$_{base}$ + Constrastive Learning	90.0	89.7	90.4
ensemble*	**90.9**	**90.7**	**91.5**

Table 3: Comparison between different models.
$*$ensemble here uses the result of hateBERT$_{base}$ + LSTM, hateBERT$_{base}$ + CLS-Last-Attention, and hateBERT$_{base}$ + Constrastive Learning

4.6 Binary Classifiers

The dataset is partitioned into training, validation and testing sets with a 8-1-1 ratio respectively. The reported F1 scores are based on the testing set, which are evaluated with the respective models that had the best validation set F1 scores.

4.6.1 Two Stage Binary Classification

Two Stage Binary Classification	F1-Score(%)
Model One (Neither/Either)	96.00
Model Two (Hate Speech/Offensive Language)	91.82
Overall (Two Stage)	90.09

Table 4: F1 scores of models in Two Stage Binary Classification

The overall two stage F1 score compares to the baseline. However, we noticed that model one has a high F1 score which can be beneficial in a use case whereby there is a need to flag tweets before it is published to the public domain.

4.6.2 One-Versus-Rest Classification

The F1 score has some margin of error due to the randomness employed to settle ties in voting scores during majority voting. For this specific score, the majority voting resulted in 2,357 tweets (95.81%) with one majority class, 81 tweets (3.27%) with two majority classes and 40 tweets (1.61%) with three majority classes. If the randomness is assumed to be fair, the margin of error should be equal to the number of tweets with more than one majority class divided by the three classes, as a percentage of the total tweets - 1.63%.

One-Versus-Rest Classification	F1-Scores(%)
Model One (Neither/Either)	96.00
Model Two (Hate Speech/Others)	92.97
Model Three (Offensive Language/Others)	90.62
Overall (Majority Voting)	93.20

Table 5: F1 scores of models in One-Versus-Rest Classification

5 Locating Offensive Words

In this section, we will discuss the location map that is generated by our proposed model, CLS-Last-Attention. The location map contains information on which token did the [CLS] focuses on. Formally, the location map is the attention weight of the final Multi-Head Attention layer. It is shown that our location map can locate all the swear and insulting words on offensive and hate speech. We found that the model tends to classify racism sentence as 'hate' and sentence that contains swear words as 'offensive'. For sentences that are labeled as 'neither', the attention focus is spread to one or more keywords. For a sentence that is labeled as 'neither' but contains swear words, our model can still locate those words. However, there is one flaw on the location map, that is the attention focus is lag by exactly one token. Although this does not prevent us from locating offensive words, we will study this behaviour in the future. Some of the examples are shown in Appendix 7.4.

6 Conclusion

In this project, we proposed a model (CLS-Last-Attention) that can locate the 'hate' and 'offensive' words without having token-level supervision. The proposed model uses a pre-trained BERT as backbone and it contains two extra components, layer-wise attention and final CLS attention. We experimented with several approaches to try improve on the proposed LastAttn and determined that the one-versus-rest classification method had the best performance with a F1 score of 93.20%, which is slightly higher than the 92% in the paper[2] we benchmark against.

There exists several limitations in the dataset. During the experiments, we found that we couldn't agree with all the dataset labels. There may be human bias involved in the dataset collection. Besides, the boundary for hate and offensive speech is not very clear, which is also a common challenge in hate and offensive speech classification datasets.

For the pre-processing steps, further improving on the quality of spell correction may also aid in improving performance. Though we did explore the use of MOE (Misspellings-Oblivious-Embeddings) in doing so, we did not find it practical to integrate with our approach at the given time. Future work may explore the use of this dataset and a deep learning model to perform more accurate and effective spell correction.

For the future work, we may look at the lag exists in the current word attention. In addition, the current location map only tells us where the hate and offensive words are, but cannot distinguish them. The classification of hate and offensive words in attention level can be another future task.

References

[1] Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. Automated hate speech detection and the problem of offensive language. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 11, pages 512–515, 2017.

[2] Marzieh Mozafari, Reza Farahbakhsh, and Noel Crespi. A bert-based transfer learning approach for hate speech detection in online social media. In *International Conference on Complex Networks and Their Applications*, pages 928–940. Springer, 2019.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[4] Tommaso Caselli, Valerio Basile, Jelena Mitrović, and Michael Granitzer. HateBERT: Retraining BERT for abusive language detection in English. In *Proceedings of the 5th Workshop on Online Abuse and Harms (WOAH 2021)*, pages 17–25, Online, August 2021. Association for Computational Linguistics.

[5] Diptanu Sarkar, Marcos Zampieri, Tharindu Ranasinghe, and Alexander Ororbia. Fbert: A neural transformer for identifying offensive content. *arXiv preprint arXiv:2109.05074*, 2021.

[6] Yige Xu, Xipeng Qiu, Ligao Zhou, and Xuanjing Huang. Improving bert fine-tuning via self-ensemble and self-distillation. *arXiv preprint arXiv:2002.10345*, 2020.

[7] Beliz Gunel, Jingfei Du, Alexis Conneau, and Ves Stoyanov. Supervised contrastive learning for pre-trained language model fine-tuning. *arXiv preprint arXiv:2011.01403*, 2020.

[8] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *Advances in Neural Information Processing Systems*, 33:18661–18673, 2020.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[10] Francesco Barbieri, José Camacho-Collados, Leonardo Neves, and Luis Espinosa Anke. Tweeteval: Unified benchmark and comparative evaluation for tweet classification. *CoRR*, abs/2010.12421, 2020.

[11] Juan Manuel Pérez, Juan Carlos Giudici, and Franco M. Luque. pysentimiento: A python toolkit for sentiment analysis and socialnlp tasks. *CoRR*, abs/2106.09462, 2021.

7 Appendix

7.1 Pre-processing Confusion Matrices

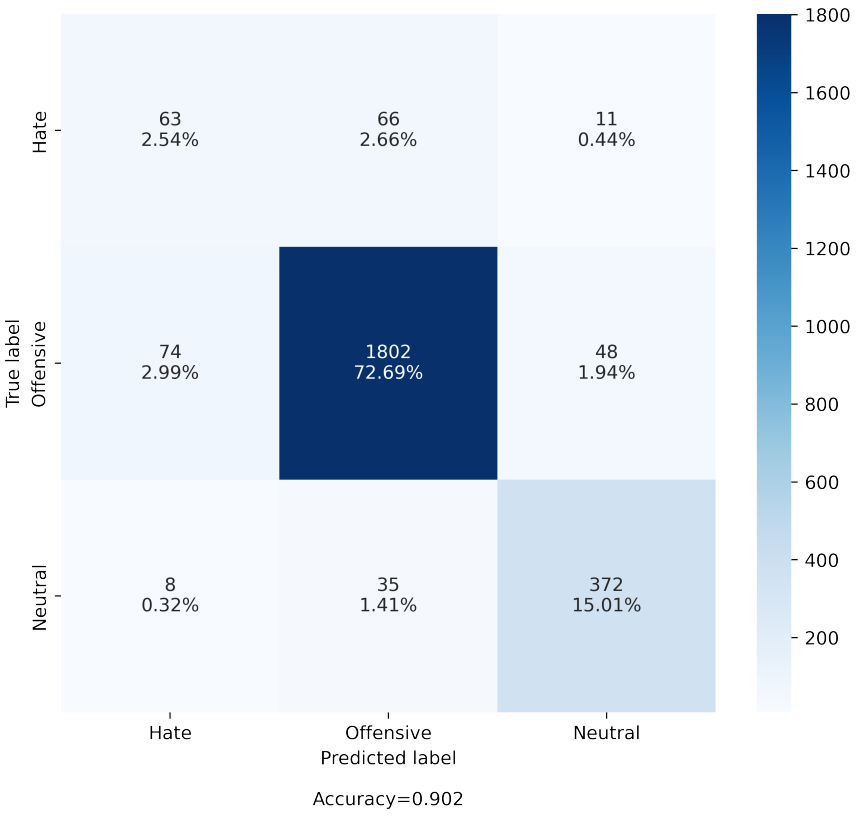

Figure 2: Confusion Matrix for BERT-base-uncased performance on unprocessed test set

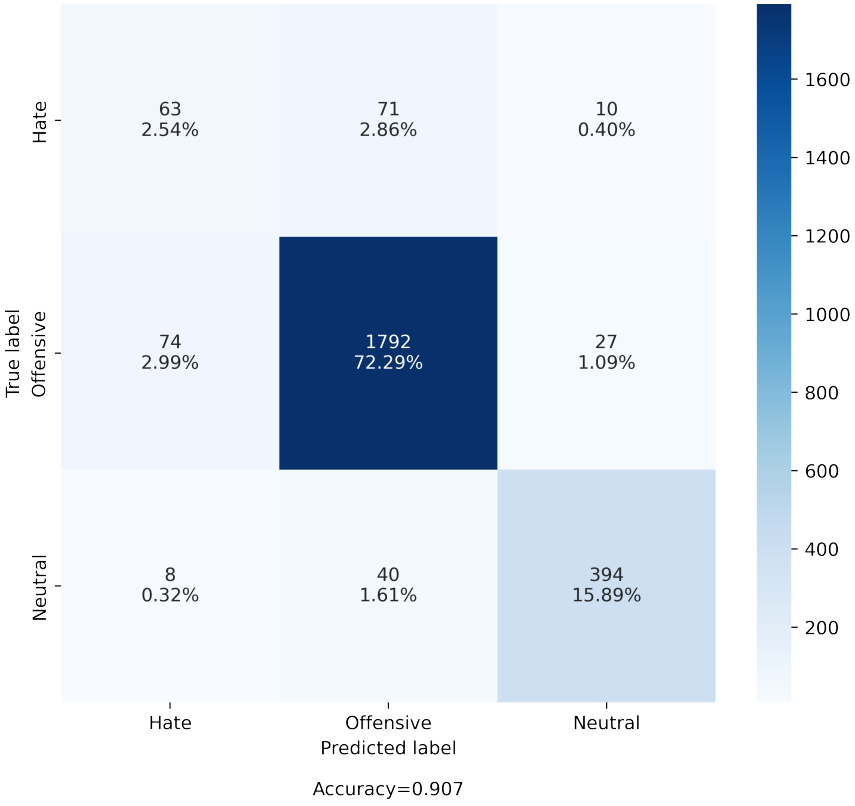

Figure 3: Confusion Matrix for BERT-base-uncased performance on pre-processed test set

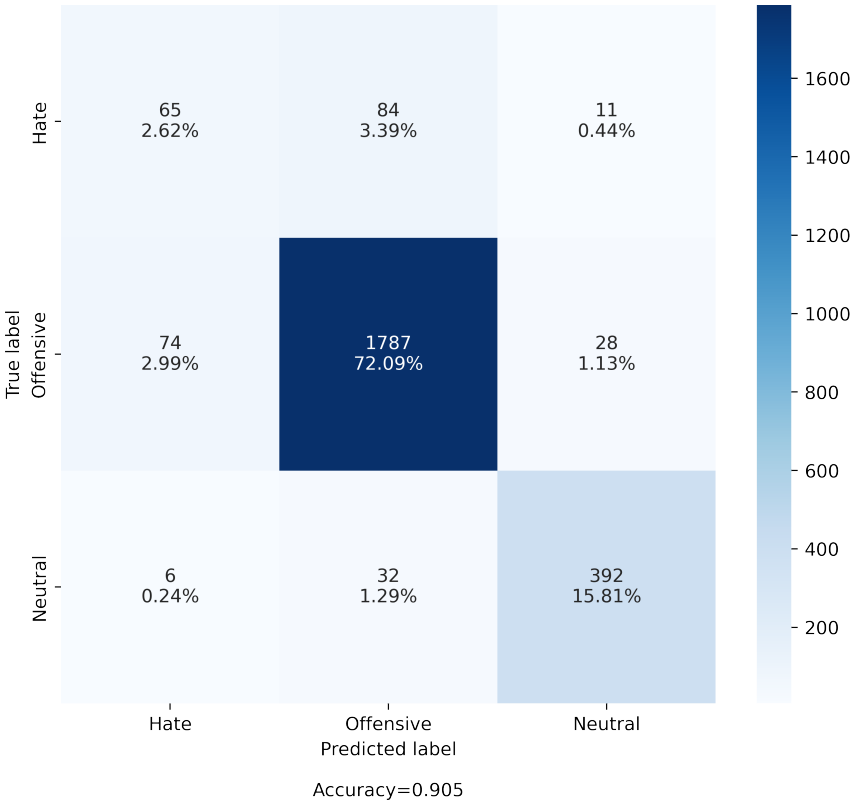

Figure 4: Confusion Matrix for BERT-base-uncased performance on pre-processed test set with spell correction

7.2 Ensemble BERTs

This section shows the structure and the loss for the ensemble BERTs method.

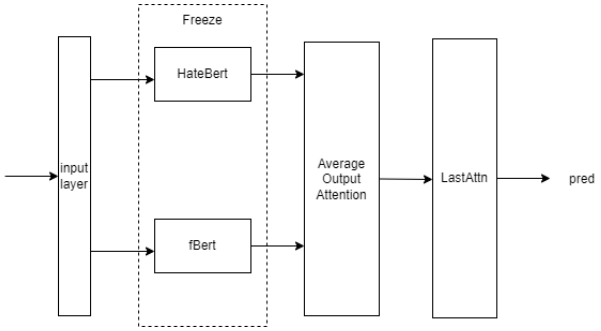

Figure 5: Ensemble BERT Models

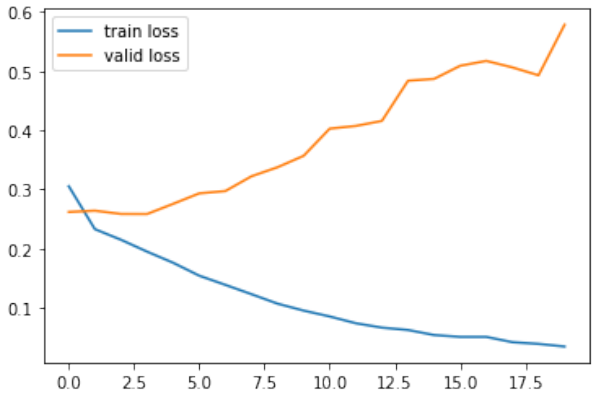

Figure 6: Ensemble BERTs Loss

7.3 Contrastive Learning Model Structure

This section shows the model structure of the contrastive learning.

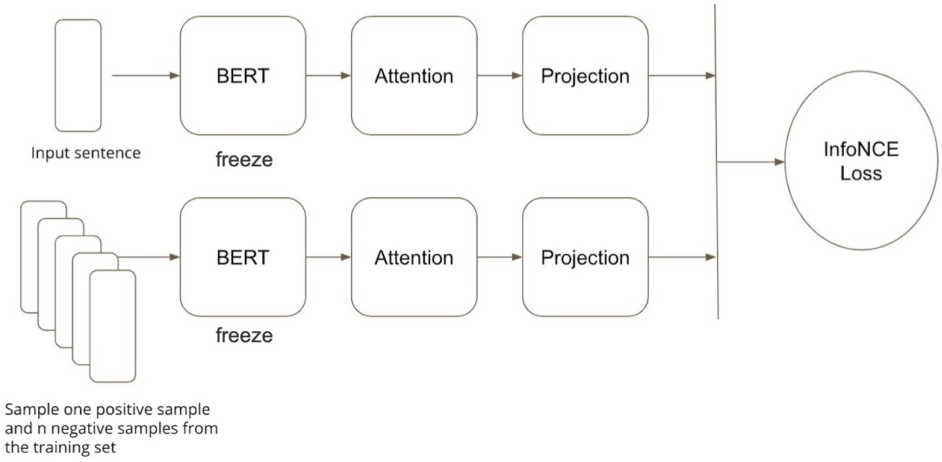

Figure 7: Stage 1 structure

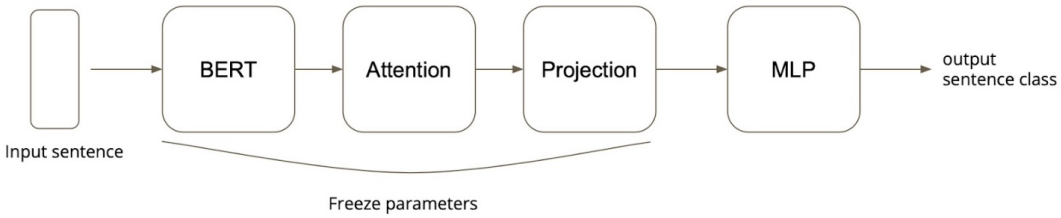

Figure 8: Stage 2 structure

7.4 Location Map

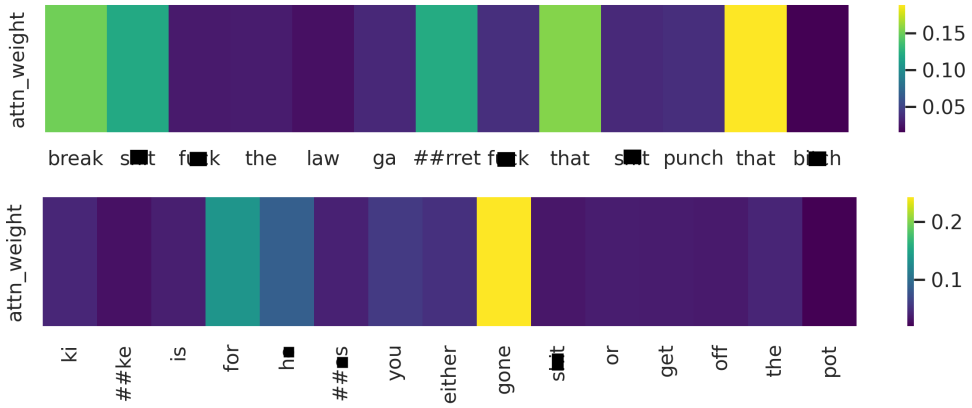

Figure 9: Location maps of 'offensive' word

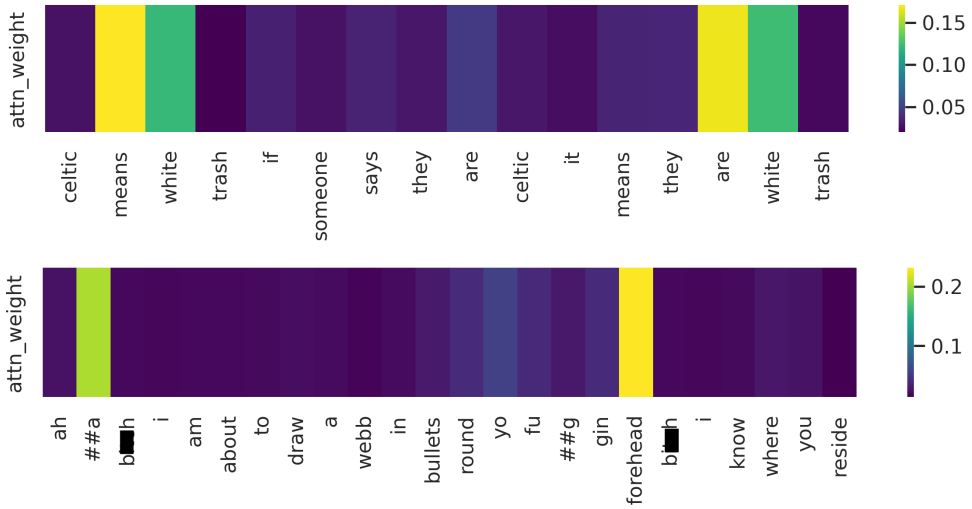

Figure 10: Location maps of 'hate' word

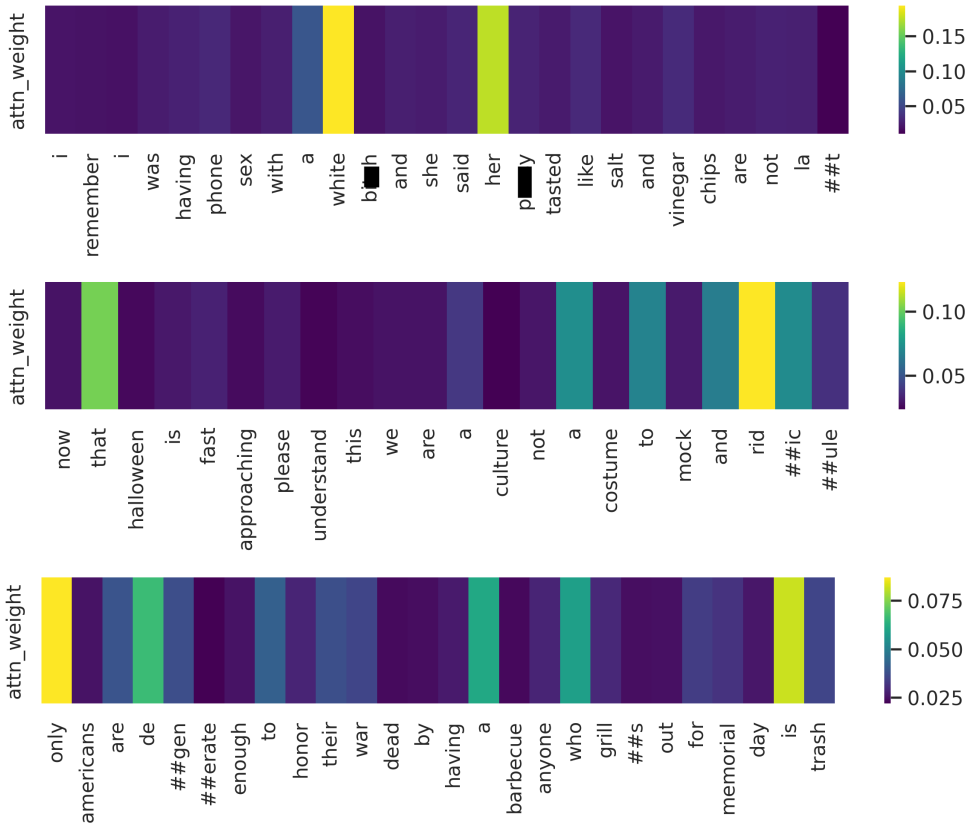

Figure 11: Location maps of ‘neither’ word

7.5 Binary Classifiers

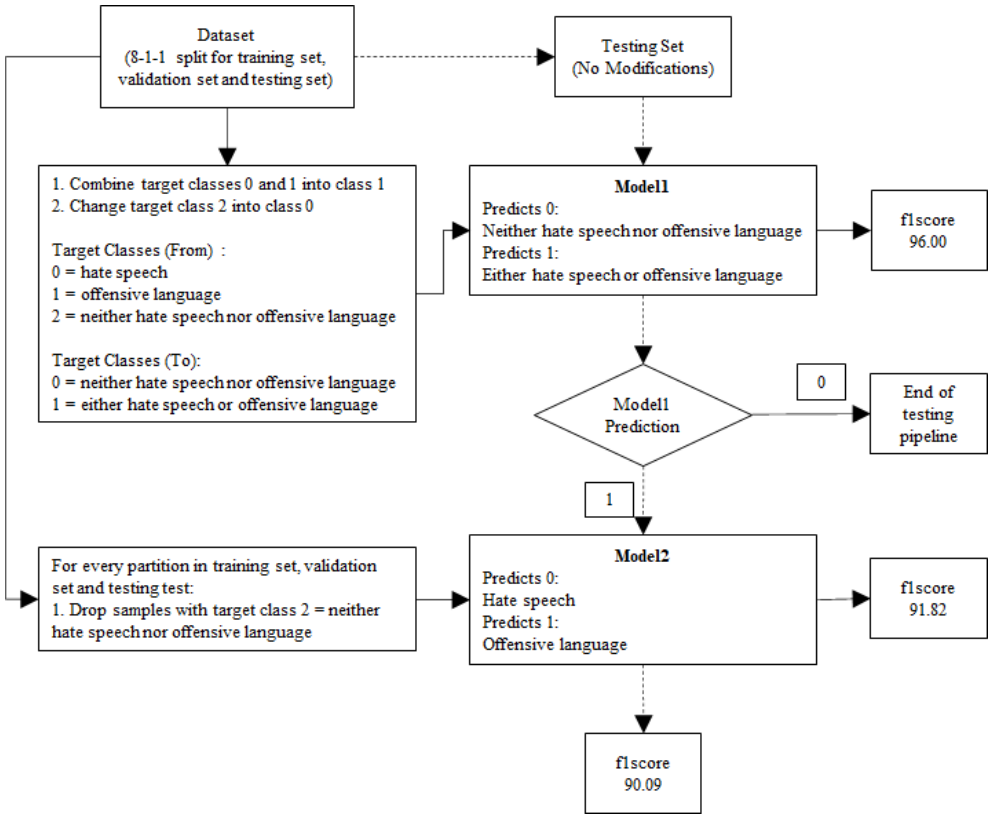

Figure 12: two stage binary classification

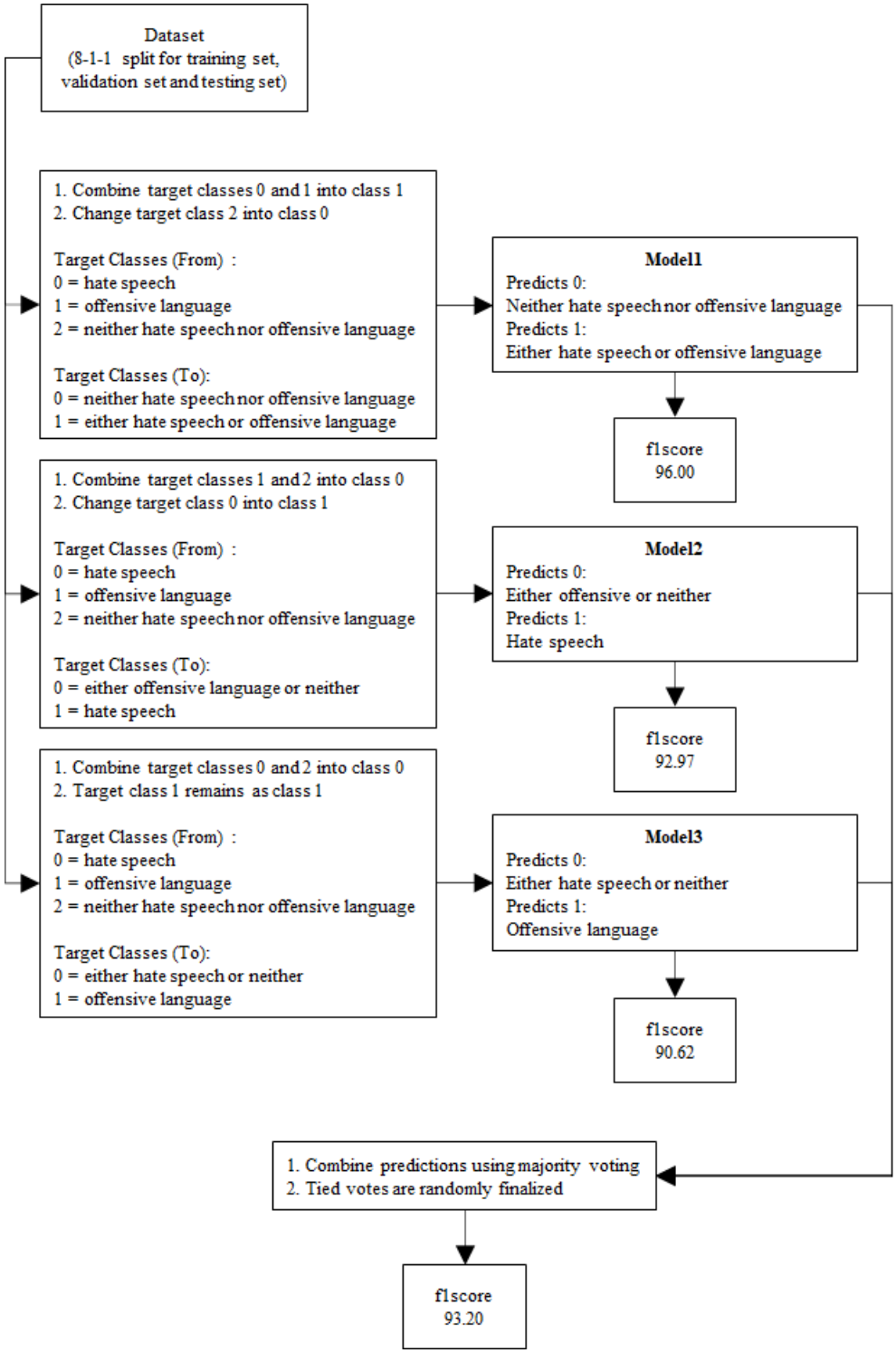

Figure 13: one versus rest classification