Predicting Cyberbullying: Building Classification Models on an NLP Dataset

Jaeho Choi , **Sangwoo Lee** , **Charles Yoo**
University of Pennsylvania

Abstract

To aid in the effort to curb the risk of exposure for children to cyberbullying, which has been on the rise since the outbreak of COVID-19, we built binary classification models that can accurately and efficiently identify cyberbullying tweets. We compared performances of numerous encoder-model combinations. For the encoder, we chose from the one-hot, stemming, and pretrained BERT encoders. When using a non-BERT encoder, we trained the multinomial Naive Bayes, logistic regression, the random forest, and the xgboost. Of all the encoder-model combinations considered, the base cased BERT model fine-tuned for binary classification performed the best. We also found that the stemming encoder is a wonderful substitute for the one-hot encoder under limited computational space.

1 Motivation

In April 2020, UNICEF issued a technical note addressing the substantial impact that COVID-19 had had on children's learning experience [UNICEF, 2020]. According to the report, as of 3 April 2020, COVID-19 had led to school closures across at least 188 countries, impacting more than 90% of the world's student population. The massive school closures caused an abrupt shift in the way that children learn, moving their learning experience online. The report raised concerns about children's increased risk of exposure to online harms and recommended measures to mitigate it at the level of governments, corporations, schools, and parents. In particular, cyberbullying, i.e., repeated behavior aimed at scaring, angering, or shaming those who are targeted on social media, messaging platforms, gaming platforms and mobile phones, is cited as a major risk. We wish to aid in the global effort to lower children's risk of exposure to cyberbullying by building binary classification models that can identify potentially harmful tweets for children accurately and efficiently. Machine learning provides an ideal framework to build such models because there are machine learning techniques that allow one to encode text-based data such as tweets into a machine-friendly format, making it possible to leverage powerful binary classification models.

2 Related Work

One notable work in the area of cyberbullying detection is [Wang *et al.*, 2020]. This paper makes several important contributions. First, the authors address the lack of publicly available, balanced cyberbullying data that contain more than three classes of cyberbullying by building upon an existing dynamic query expansion (DQE) algorithm that can automate the collection of balanced data. Secondly, they make an innovative use of a graph convolutional network for the task of cyberbullying detection by training it on a graph of tweet embeddings designed to enable effective propagation of labels across tweets with similar main ideas. This classification method helps combat the terseness of tweets compared to much longer text-based data such as documents. Thirdly, they compare and contrast accuracy scores across numerous combinations of tweet embedding methods and classification models for two different sizes of data for a multi-class cyberbullying classification task. In doing so, they contextualize the performance of their novel graph neural network-based classification method and provide benchmarks for future research. Lastly, they make their dataset publicly available. Although they do not use non-cyberbullying tweets, they include such tweets in their dataset so that researchers interested in binary classification tasks can use their dataset. The authors argue that their dataset also contains a more representative sample of cyberbullying tweets than previously available datasets.

Despite these contributions, the paper has two identifiable shortcomings. First, although the authors demonstrate that their novel classifier can match the performance of top-of-the-line SVM models with statistical significance when trained on a dataset containing $4,000$ tweets, they omit without explanation their novel classifier when using a larger dataset containing $40,000$ tweets while keeping all the other classifiers in. Secondly, balanced as their dataset may be, it seems to suffer from mislabeling issues. Upon close inspection of the data using exploratory data analysis (EDA) techniques like word clouds for different cyberbullying classes and inspecting small batches of tweets for each cyberbullying class, we find that certain tweets are given incorrect labels, which will be further investigated in the Dataset section.

3 Dataset

We use a dataset from [Wang *et al.*, 2020], which is available on Kaggle[1]. It contains 46,017 tweets labelled as either not cyberbullying or one of the five classes of cyberbullying: age, ethnicity, gender, religion, and other. There are no missing data in the dataset, i.e., there are no empty tweet texts or missing labels. The tweets are evenly distributed across all labels as shown in Figure 1.

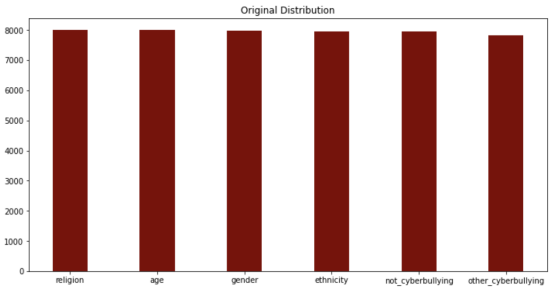

Figure 1: Distribution of labels in original data

Using standard EDA techniques, we reveal that the dataset may suffer from two kinds of labeling issues. The first kind of labeling issue occurs when a tweet that is naturally associated with more than one cyberbullying class is given just one label. In our dataset, for example, "girl" is one of the most important keywords in the word clouds for age and gender (see Figures 2 and 3), which suggests that there may be a number of tweets containing the word "girl" that can be labeled as either age or gender. It is not difficult to find tweets that suffer from this kind of labeling issue when we display random batches of tweets from age and gender. For example, the tweet "He's probably a white gay kid from some suburb. We don't want him to shoot up his school...so no one bully him" can be labeled as either age or gender, but it is assigned as age. The authors of [Wang *et al.*, 2020] also address this issue, calling it an "annotator error" on the part of the six older datasets from which the dataset is constructed, and remark that due to this issue, the accuracy score of a model trained on their dataset may underestimate its performance in real life.

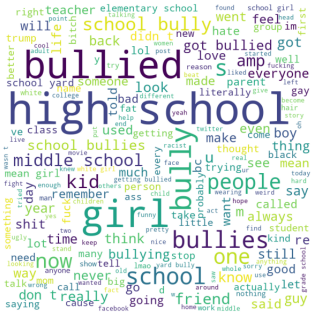

Figure 2: Word cloud for age

[1]https://www.kaggle.com/datasets/andrewmvd/cyberbullying-classification

Figure 3: Word cloud for gender

Figure 4: Word cloud for other cyberbullying

Figure 5: Word cloud for not cyberbullying

The second kind of labeling issue occurs when a tweet is incorrectly labelled as cyberbullying. In our dataset, we can easily find this kind of labeling issue for tweets in the other cyberbullying class. For example, the tweet "RT @BuzzFeedUK: When you accidentally open your front camera: http://t.co/gu35jqiPye" is given the other cyberbullying label, even though the text is neutral and the link directs one to innocent pictures of a cat and a chick. Another tweet "Getting OAPI's GitHub set up. Transferred emojiautoblocker, created private projects, time to start importing code" is labelled as other cyberbullying, albeit its neutral tone. This kind of labeling issue is exacerbated by tweets in the not cyberbullying class that carry tones that are reminiscient of cyberbullying. For example, the not cyberbullying class includes tweets such as "Because if Elliot Rodger had killed just ONE woman who had rejected him for a date, it would not be national news. #YesAllWomen"; "@BeRh00M What

you see as "normal," women see as threatening, and we keep telling you dudes this, but you don't actually want to listen."; and "@DrHaque @MaxBlumenthal @NYCJulieNYC @mehdirhasan @tnr You have to be dumber than a rock to think that ISIS wants a Jewish state." Although these tweets are not necessarily mislabeled, their uses of particular non-stopwords and underlying tones would make the classification process challenging for models. Indeed, we see that the word clouds for "other" cyberbullying and not cyberbullying are characterized by similar keywords, which suggests that it would be more difficult to correctly predict the label for the "other" cyberbullying class than for the other four cyberbullying classes (see Figures 4 and 5). This observation is later backed by results from preliminary experiments based on one of the simplest, out-of-the-box classification models trained on the dataset embedded by a rudimentary text encoder (see Section 6.3).

In view of this observation, we decide to merge all cyberbullying classes into one single class to make the classification task somewhat harder especially for more advanced classification models that we later employ. However, pooling five out of the six labels within the originally balanced dataset inevitably introduces a data imbalance issue. Although there are clever machine learning techniques that can handle such issues such as the synthetic minority oversampling technique (SMOTE), we instead decide to randomly sample $2,000$ tweets from each original cyberbullying class to create one representative cyberbullying class that matches the not cyberbullying class in size (see Figure 6). After random sampling, we end up with $10,000$ tweets in the cyberbullying class and $7,945$ tweets in the not cyberbullying class. Random sampling is an effective way to down-size data without compromising its quality, although it comes at the expense of discarding a large portion of the original dataset. This decision has been made for computational tractability given our time and computational resources.

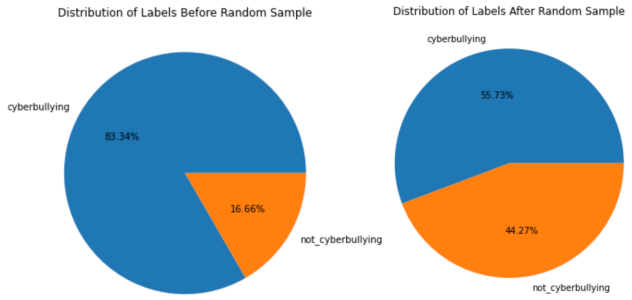

Figure 6: Distribution of labels before and after random sampling

4 Problem Formulation

4.1 Framework

Our aim is to build binary classification models that are able to detect tweets that can be deemed cyberbullying for children accurately and efficiently. There are two key challenges in formulating this classification problem into a machine learning task. The first challenge is that in order to leverage machine learning techniques, our text-based data must be encoded in a machine-friendly format that can be readily fed into binary classification models. The second challenge is that due to our limited time and computational resources relative to the size of the encoded data, we can train only a restricted set of classification models on the encoded data. Figure 7 presents a simple framework summarizing these challenges and elucidating the machine-learning process by which tweets can be classified.

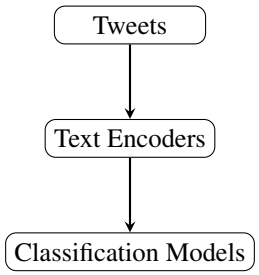

Figure 7: A framework for data preprocessing and model fitting

Tweets

In the ensuing discussion of this section, whenever we talk of the dataset, we are referring to the dataset re-balanced for binary classification via random sampling.

Encoding Tweets

The first step in this framework is encoding the tweets. We employ three different encoders: the one-hot encoder, the stemming encoder, and the pretrained BERT encoder. Each encoder embeds tweets as vectors that can be passed to classification models for training. However, each encoder achieves this essential task with a different objective.

For the one-hot encoder, all tweets are first tokenized to create a global dictionary of size d. The size d is less than or equal to the sum of the numbers of tokens in all tweets since some tweets may share one or more tokens. Once the dictionary has been made, each tweet is encoded as a vector of size d, whose ith entry contains the tf-idf score for the ith token of the dictionary for that tweet.

The tf-idf score for a token in a given text in a collection of texts is a measure of how important that token is to that text in the collection. It is proportional to the number of times that the token appears in the text and is inversely proportional to the number of texts that contain the token. We decide to store inside the vectors tf-idf scores instead of counts for respective tokens in the tweet, because the tf-idf scheme assigns relatively low scores to stopwords, which mimics the effect of dropping them. This is especially beneficial to us, because having already discarded a large portion of the initial dataset through random sampling, we do not wish to lose further information by getting rid of stopwords from the remaining tweets. In short, the one-hot encoder transforms the text-based data into a machine-friendly tabular form, in which the features are the tokens in the global dictionary extracted from all tweets.

For the stemming encoder, the way in which the tweets are encoded into vectors is identical as in the one-hot encoder,

i.e., every tweet is transformed into a vector of size equal to the dictionary size, whose entries store tf-idf values of respective tokens. The only difference is in that before vectorizing the tweets, they are pared down in such a way that balances between minimizing memory use to store tabularized data and minimizing the loss of information.

The objective of the stemming encoder is to reduce memory use without compromising on the classification accuracy, which is akin to the idea behind file compression. As with the case of file compression, there is considerable leeway in the manner in which we can pare down each tweet while maintaining its essential meaning. In our project, each tweet is cleaned up such that every character is lower-cased; all stopwords except for "not" and "can" are removed; some special characters and all punctuation except "?" are discarded; all trailing whitespaces are removed; "'t" is changed to " not"; and all tweet handles and hyperlinks are removed. Then, each tweet is stemmed such that each space-separated token in the tweet is reduced to its root token. For example, "singing" is reduced to "sing." Once pared down, each tweet is turned into a vector as in the one-hot encoder.

```
islam think merkel democraci
rt oh come not even tri hide hypocrisi anymor
click 4 detail gt road lead wildbul thursday host everyon 5 lt spread word
almost time
rt stuf instant couscou immedi elimin mkr
```

Figure 8: Tweets after cleaning and stemming

The clean-up process outlined above sometimes transform certain tweets into empty strings. We decide to keep these empty strings in for the following reason. Under the current clean-up scheme, a tweet can become an empty string only if a tweet is made up exclusively of stopwords, punctuation characters, special characters, tweet handles, or hyperlinks except notably for "not", "can", and "?". Sensibly, such a tweet has no discernible meaning with regards to detecting cyberbullying, on which its label can be justified. The fact that such tweets exist in the dataset, however, is further evidence for the earlier impression that the original dataset may suffer from mislabeling issues. Moreover, the distribution of these empty strings in Figure 9 reveals that such tweets are concentrated in the other cyberbullying and not cyberbullying classes, which supports the earlier claim that it is more challenging to correctly predict the label for the other cyberbullying class than for the other four cyberbullying classes. We decide to keep these vanishing tweets in, since they can provide regularizing effects against overfitting when trained on by classification models.

The pretrained BERT encoder is a state-of-the-art text encoding scheme for natural language processing (NLP). It is a neural network-based encoder with millions of parameters that, as its name implies, have been trained to develop a highly intricate text-embedding scheme to understand the nuances of the English language. Unlike the one-hot and stemming encoders, which are specifically designed for our binary classification task, the pretrained BERT encoder has

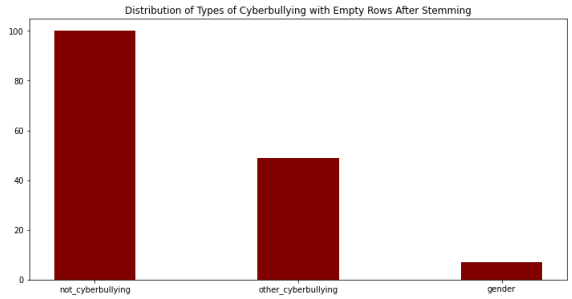

Figure 9: Distribution of empty strings after cleaning and stemming

no knowledge of our task. However, the pretrained BERT encoder can be fine-tuned for a wide range of tasks related to NLP, such as text classification and question answering. In our project, we will fine-tune the pretrained BERT encoder to solve our binary classification task. The use of the pretrained BERT encoder is an instance of transfer learning, which is a powerful machine learning technique.

Classification Models

The last step in our framework is the choice of classification models to use. Due to limited time and computational resources, we need to meticulously select a restricted set of binary classification models to train using the tabularized data encoded by the one-hot and stemming encoders. For the pretrained BERT encoder, there is no need to make such selections because we can simply fine-tune it for the task of binary classification.

As a baseline, we choose the multinomial naive Bayesian model, which is a non-regressive classification model whose predictions are based solely on computing probabilities under the so-called conditional independence assumption that greatly reduces probability computations. For this reason, it works even for enormous datasets with relative ease. Despite its relatively low computational cost, it performs decently when compared against other classification methods. Its reliable performance and relatively cheap computational cost makes it a natural choice for a baseline model. One caveat is that we need to determine for ourselves whether the conditional independence condition is sufficiently realistic in the context of our dataset. When our data are vectorized, their features will roughly correspond to words that appear in all the tweets. Other than grammatical reasons, there is little evidence that the tf-idf scores of words in the dictionary are correlated with one another, sufficiently justifying the use of the multinomial Bayesian model for our data.

For comparison, we include logistic regression and the xgboost because the former can provide a nice context to the performance of the latter beyond the baseline and, according to [Wang *et al.*, 2020], numerous earlier cyberbullying detection studies use them, which can provide some benchmark performances. Moreover, the random forest is included to compare against the xgboost, both of which are advanced tree-based ensemble methods. We expect that the multinomial naive Bayesian model would perform worst, while the ensemble methods would perform best for any encoder.

5 Methods

5.1 One-Hot Encoder

The one-hot encoder transforms words into mathematical vectors. Being one of the most intuitive and simplest-to-implement text encoders, it is a natural choice as our benchmark encoder.

Package keras.preprocessing.text.one_hot

5.2 Stemming Encoder

The stemming encoder functions very similarly to the one hot encoder. The difference is that it cleans up and shortens words, making it computationally lighter to use. The stemming encoder will be compared to the one hot encoder to understand the trade-off between performance and efficiency.

Package: keras.preprocessing.text.Tokenizer

5.3 BERT Encoder

The BERT encoder uses transformers instead of recurrent or convolutional neural networks. One advantage of using transformers over these neural nets is that the data can be processed in any order, which makes it easier to process larger datasets. BERT stands for Bidirectional Encoder Representations from Transformers. BERT's bidirectionality means that it is able to read text input in both directions at the same time. This allows us to pretrain the BERT model on two natural language processing tasks: masked language modelling and next sentence prediction.

Package: transformers.BertTokenizer

Masked Language Modelling

Masked language modelling is an unsupervised problem in which a model must predict the hidden words. To do this, a portion of the words in data are masked. The model will then look at the text samples and use the context provided by the available words to predict the hidden word. To prevent overfitting to this placeholder mask token, there is a parameter to replace this token with a token for a random word.

Next Sentence Prediction

Next sentence prediction is another unsupervised problem in which the model seeks to predict whether or not two sentences are connected. The model will predict with some probability whether or not the second sentence logically makes sense following the first one.

Structure of the BERT Encoder

The BERT encoder consists of multiple layers that correspond to a transformer encoder block. A transformer encoder block consists of 6 layers. Every layer has a multi-head self-attention sublayer and a fully connected feed-forward network sublayer, which are both followed by a normalization layer. Though the 6 layers share the same structure, they use different weights and bias parameters in their feed-forward network sublayers.

The BERT encoder takes in a sequence of tokens or a pair of sequences as input. The sequence of tokens is created using the word-piece tokenizer, which takes words and can tokenize some words as sub-words. This helps break down different forms of a word into known sub-words. BERT also takes in special tokens: [CLS] to indicate the beginning of the first sequence and [SEP] to indicate the end of a sequence. These sequences are used to create token, positional, and segment embeddings. Token embeddings are the tokenized representations of words in the sequence. Position embeddings are the position of each token within the sequence. Finally, segment embeddings are the sequence that the token is a part of.

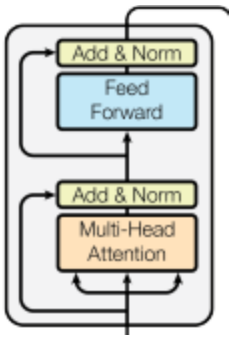

Figure 10: Transformer Layer [Vaswani *et al.*, 2017]

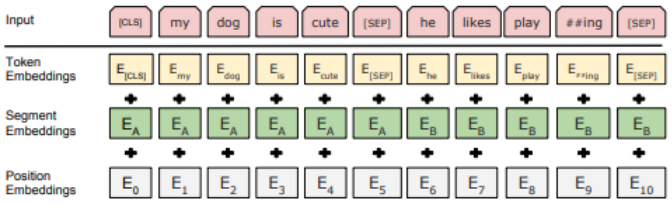

Figure 11: BERT Input Format [Devlin *et al.*, 2019]

The BERT encoder takes the combination of these embeddings and passes it through the chain of encoder blocks. Each block will identify relationships between the input sequences. As you pass the sequence through the chain of encoder blocks, the encoder is able to learn a more complex relationship between the input sequence and create a more accurate representation of the original input. There are two types of the model architecture. BERT-Base has 12-layers, 768 hidden nodes, 12 attention heads, and 110 million parameters. Bert-Large has 24 layers, 1024 hidden nodes, 16 attention heads, and 340 million parameters.

Transfer Learning for Binary Classification

What makes the pretrained BERT encoder useful is that it can be fine-tuned to specific problems. For binary classification, this can be done by connecting a single linear layer to the end of the original model. For our problem we added a layer using the simpletransformers package.

Package: simpletransformers.classification

5.4 Naive Bayesian Model (Baseline)

The baseline that we chose was the Naive Bayes model. The Naive Bayes model is a classification method that can be used for both binary and multi-class classification problems. Since we are using it in a binary classification problem, it is a relatively simple and quick model so it is a good way to gain

some understanding of the problem. One problem with the Naive Bayes model is that it assumes that features are conditionally independent of each other. Since this assumption may not necessarily true, the Naive Bayes model also serves as a good baseline to compare other models to.

Package: sklearn.naive_bayes.MultinomialNB

5.5 Logistic Regression

Logistic regression is another model that is useful for binary classification. This model uses a sigmoid function to determine whether or not a tweet should be considered cyberbullying. One potential advantage of logistic regression is that it does not need to make the assumption that all the features are independent. If logistic regression performs better than the Naive Bayes model, then this might suggest that the token features used in NLP problems are not independent.

Package: sklearn.linear_model.LogisticRegression

5.6 Random Forest

The Random Forest model is an ensemble method of multiple decision trees. Since it splits of random features, we can model the data with good accuracy and without a huge risk of overfitting (unlike regular decision trees). For binary classification, the Random Forest model will predict whether or not a tweet is cyberbullying based on the majority of the trees. For our problem, we tested using the entropy criterion for determining how good a split is.

Package: sklearn.ensemble.RandomForestClassifier

5.7 XGBoost

Extreme Gradient Boosting (XGBoost) is an implementation of gradient tree boosting. For this model, weak tree learners are added sequentially to minimize the error of the prior tree. These trees are then combined to make the final prediction of whether or not a tweet is cyberbullying. XGBoost is good for classification as it is both efficient and accurate. Pruning decision trees in the model and regularization will help improve the model's performance by helping to prevent overfitting. For our problem, we used 100 trees, a subsample ratio of 1, and a max depth of 3.

Package: xgboost.XGBRFClassifier

6 Experiments and Results

6.1 Data Preparation

The original data come in an ideal format for a multi-class classification task. However, its format is inadequate for a binary classification task. To make it suitable for binary classification, we shuffle and randomly sample 2,000 tweets from each of the five classes of cyberbullying and merge them into one representative class, leaving behind some 30,000 tweets for computational tractability. In so doing, we make the classification label binary and the dataset re-balanced for a binary classification task. For the one-hot encoder and the pretrained BERT encoder, we finally set aside 75% of the dataset for training and the remaining 25% for testing. For the stemming encoder, we first clean up and stem the tweets, and then set aside the same 75% of the dataset for training and the remaining 25% for testing.

6.2 Evaluation Metrics

We use the confusion matrix and the accuracy score to measure how well various combinations of encoders and models perform cyberbullying detection on test data.

The decision to report the confusion matrix is intentional. The act of flagging a tweet as cyberbullying has profound social repercussions. From the perspective of children's mental well-being, it is desirable to flag every suspicious tweet as cyberbullying. However, from the perspective of free speech, doing so may constitute an encroachment upon freedom of speech. In the former case, a high recall score is important albeit at the expense of precision. In the latter case, a high precision score is preferable even at the expense of recall. By reporting the confusion matrix from which both scores can be calculated, we defer the judgment of importance to the reader.

Since both train and test data were re-balanced for a binary classification task, it suffices to report the accuracy score without reporting the F1 score or the AUC score. Although the accuracy score can be calculated from the confusion matrix, we report it separately for convenience.

Finally, we calculate the difference in the time taken for each classification model to finish running based on the one-hot or stemming encoder to measure the efficiency of the stemming encoder over the one-hot encoder. More precisely, we subtract the stemming encoder time from the one-hot encoder time such that a positive (negative) number indicates higher (lower) efficiency for the stemming encoder.

6.3 Preliminary Experiments

Naive Bayesian	age	ethnicity	gender	religion	other
Test Accuracy (%)	88.91	91.60	85.07	91.32	67.45

Table 1: Table for test accuracy without hyperparameter tuning

In the Dataset section, we suggested that all but the "other" class of cyberbullying contain tweets that are relatively easy to classify even with bare eyes. We will now present some experimental evidence for this observation. Table 1 summarizes test accuracy scores obtained for the default naive Bayesian model trained on the binary data consisting of the non-cyberbullying class and just one of the five cyberbullying classes, each consisting of the $2,000$ randomly sampled tweets that are later combined into the single representative cyberbullying class. This binary data are embedded using the one-hot encoder. The table shows that one can achieve very high accuracy scores even with the baseline encoder-model combination for all but the "other" cyberbullying class. As stated in the Dataset section, this motivates our decision to merge all of the five cyberbullying classes into one single cyberbullying class to make the classification task more difficult, especially for more advanced classification models that we employ.

6.4 One-Hot Encoder

To implement the one-hot encoder, we use the Tokenizer class of the `keras` library. It can generate the global dictionary of tokens based on the input collection of tweets and vectorize

each tweet based on the tf-idf scores of its constituent tokens. Once training data are vectorized this way, we use it to train the following models for binary classification: the multinomial naive Bayesian, the logistic regression, the random forest, and the xgboost. Due to limited computational resources, we are unable to utilize the GridSearchCV function of the `sklearn` library to perform hyperparameter tuning. Instead, whenever possible, we tune the model by hand.

Naive Bayseian Model

For the multinomial naive Bayesian model, the MultinomialNB classifier of the `sklearn` library is used. Recall that our encoded data is in a tabularized format, where the rows correspond to the tweets, the columns correspond to the tokens in the dictionary, and the value at row i and column j corresponds to the tf-idf score of token j in tweet i. Although the MultinomialNB classifier is intended to be trained on tabularized text data containing integer token counts, we may still use it, because according to the documentation, fractional td-idf scores also work in practice. We manually tune the model by adjusting the Laplace smoothing hyperparameter $\alpha \in \{0.5, 1, 2, 3, 4\}$, starting from the default value of $\alpha = 1$. At the optimal value of $\alpha = 3$, the test accuracy is 77.42%. In Figure 12, we report the confusion matrix.

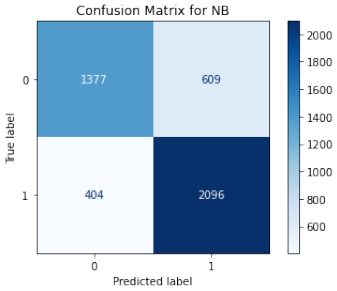

Figure 12: Confusion matrix for Naive Bayesian after tuning

Logistic Regression

For logistic regression, the LogisticRegression classifier of the `sklearn` library is used. We manually tune the model by adjusting the inverse of regularization strength hyperparameter $C \in \{0.001, 0.01, 0.1, 1, 10\}$, starting from the default value of $C = 1$. At the optimal value of $C = 0.01$, the test accuracy is 82.41%. In Figure 13, we report the confusion matrix.

Random Forest

For the random forest, the RandomForestClassifier of the `sklearn` library is used. We manually tune the model by adjusting the number of features considered when looking for the best split, max_features $\in \{\text{sqrt}, \text{log2}, 500\}$, starting from the default value of sqrt. At the optimal value of max_features $=$ sqrt, the test accuracy is 82.52%. In Figure 14, we report the confusion matrix.

XGBoost

For the XGBoost, the XGBClassifier of the `xgboost` package is used. Due to limited computational resources, we are

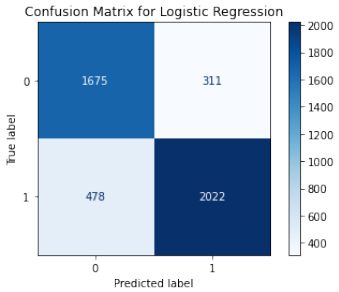

Figure 13: Confusion matrix for logistic regression after tuning

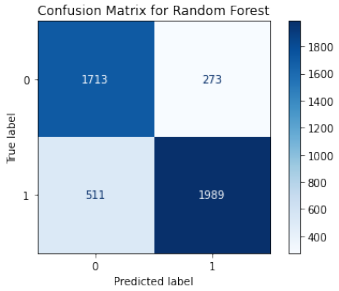

Figure 14: Confusion matrix for random forest after tuning

unable to perform hyperparameter tuning even by hand. With the default hyperparameters, the test accuracy is 84.82%. In Figure 15, we report the confusion matrix.

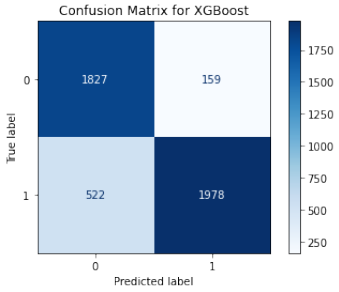

Figure 15: Confusion matrix for XGBoost

6.5 Stemming Encoder

To implement the stemming encoder, we apply the Tokenizer class of the `keras` library as in the one-hot encoder case, after using the `nltk` library for cleaning up and stemming the tweets. The set of models from the one-hot encoder case are re-trained on the data vectorized by the stemming encoder for direct comparison with the one-hot encoder. As in the one-hot encoder case, we are unable to utilize the GridSearchCV function of the `sklear` library to perform hyperparameter tuning due to limited computational resources. Instead, whenever possible, we tune the model by hand.

Native Bayesian Model

For the multinomial naive Bayesian model, the MultinomialNB classifier of the `sklearn` library is used. We manually

tune the model by adjusting the Laplace smoothing hyperparameter $\alpha \in \{0.5, 1, 2, 3, 4\}$, starting from the default value of $\alpha = 1$. At the optimal value of $\alpha = 3$, the test accuracy is 75.64%. In Figure 16, we report the confusion matrix.

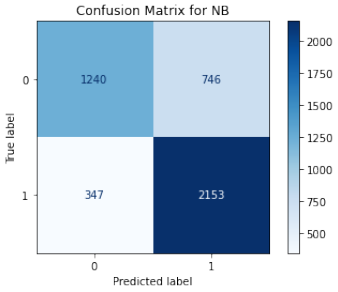

Figure 16: Confusion matrix for Naive Bayesian after tuning

Logistic Regression

For logistic regression, the LogisticRegression classifier of the `sklearn` library is used. We manually tune the model by adjusting the inverse of regularization strength hyperparameter $C \in \{0.001, 0.01, 0.1, 1, 10\}$, starting from the default value of $C = 1$. At the optimal value of $C = 0.01$, the test accuracy is 82.48%. In Figure 17, we report the confusion matrix.

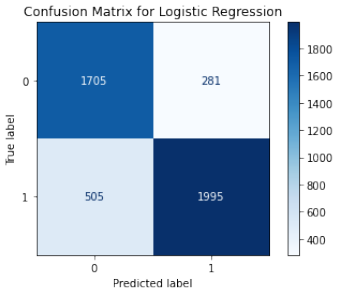

Figure 17: Confusion matrix for logistic regression after tuning

Random Forest

For the random forest, the RandomForestClassifier of the `sklearn` library is used. We manually tune the model by adjusting the number of features considered when looking for the best split, max_features, and the number of trees in the forest, n_estimators. Testing the pairs

$$(\text{max\_features}, \text{n\_estimators}) \in \{(\text{sqrt}, 100), (\text{log2}, 100), (10, 100), (10, 500), (10, 1000), (500, 100)\}$$

starting from the default pair of $(\text{sqrt}, 100)$, we find that the optimal pair is $(10, 500)$, with the test accuracy of 81.77%. In Figure 18, we report the confusion matrix.

XGBoost

For the XGBoost, the XGBClassifier of the `xgboost` package is used. Due to limited computational resources, we are unable to perform hyperparameter tuning even by hand. With the default hyperparameters, the test accuracy is 84.77%. In Figure 19, we report the confusion matrix.

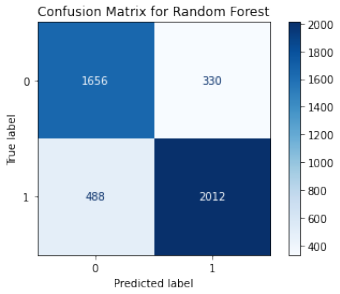

Figure 18: Confusion matrix for random forest after tuning

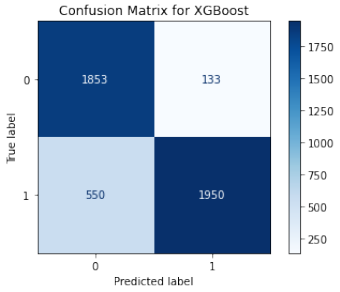

Figure 19: Confusion matrix for XGBoost

6.6 Pretrained BERT Encoder

To implement the pretrained BERT encoder, we use the `simpletransformers` library, which allows us to efficiently fine-tune pretrained BERT models for a binary classification task within the confines of our computational resources. Specifically, we use the ClassificationModel of the library. As it does not support features in which we can dictate which particular binary classification model to train on top of the BERT output layer, we do not use the set of models used for the one-hot and stemming encoders. In their place, we use four variants of the pretrained BERT encoder: the base cased, the base uncased, the large cased, and the large uncased models. Although we have no knowledge of the exact fine-tuning architecture atop the pretrained BERT encoder, we can still make valid comparisons across these variants without it and also draw interesting comparisons across all three of our main encoders.

Unlike the confusion matrices for the one-hot and stemming encoders which can be reproduced by running the same code, the confusion matrices for the pretrained BERT encoders can slightly change every time the same code runs. We have been unable to find any feature within the `simpletransformers` library to make the code output stable.

For each variant of the pretrained BERT encoder, we perform hyperparameter tuning whenever possible by adjusting the learning rate and the number of epochs that the binary

classification model will be trained for. We observe that whenever the number of training epochs is selected to be greater than 1, the performance of the pretrained BERT encoder fine-tuned for binary classification drops abruptly to a level which renders it unusable. Given that the authors of [Devlin *et al.*, 2019] recommend 2 to 4 epochs for fine-tuning, one training epoch seems too few at first glance. One sensible explanation for this phenomenon is the relatively small size of our dataset. [Devlin *et al.*, 2019] mentions that large datasets over the order of 100,000 observations are significantly less sensitive to hyperparameter choice than small datasets. Provided that our training data contain about 13,000 rows, an abrupt decrease in performance past 1 is perhaps no surprise.

Base Uncased Model

With 1 training epoch and the learning rate of 4e-5, the test accuracy is 82.61%. In Figure 20, we report the confusion matrix.

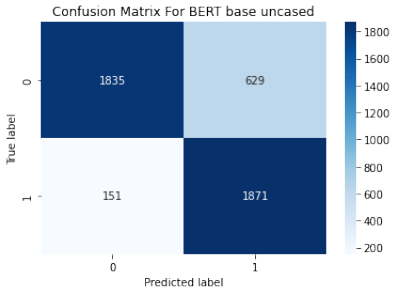

Figure 20: Confusion matrix for BERT base uncased model

Base Cased Model

With 1 training epoch and the learning rate of 4e-5, the test accuracy is 85.06%. In Figure 21, we report the confusion matrix.

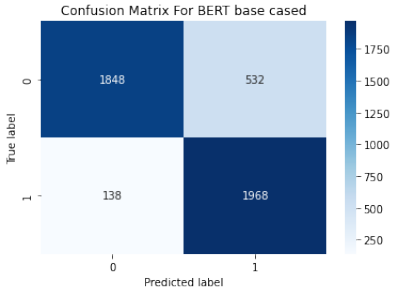

Figure 21: Confusion matrix for BERT base cased model

Large Uncased Model

With 1 training epoch and the learning rate of 2e-5, the test accuracy is 81.63%. In Figure 22, we report the confusion matrix.

Large Cased Model

With 1 training epoch and the learning rate of 4e-5, the test accuracy is 83.99%. In Figure 23, we report the confusion matrix.

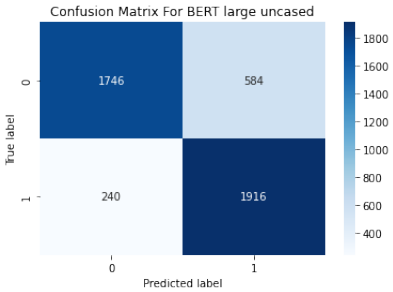

Figure 22: Confusion matrix for BERT large uncased model

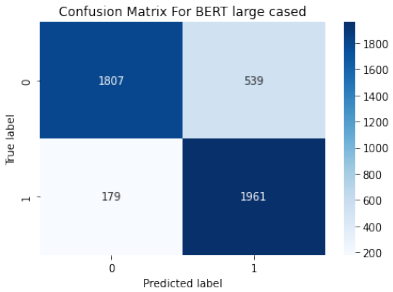

Figure 23: Confusion matrix for BERT large cased model

6.7 Summary

Below, we summarize our results in two tables. Table 2 summarizes the test accuracy scores for all combinations of encoders and binary classification models considered after hyperparameter tuning. Table 3 summarizes the efficiency scores for all binary classification models considered for the one-hot and stemming encoders after hyperparameter tuning. The efficiency scores reported are based on the CPU execution time, not the wall time.

Test Accuracy (%)	NB	LR	RF	XGB
One-Hot	77.42	82.41	82.52	84.82
Stemming	75.64	82.48	81.77	84.77
Base Uncased	82.61			
Base Cased	85.06			
Large Uncased	81.63			
Large Cased	83.99			

Table 2: Table for test accuracy after hyperparameter tuning

One-Hot vs. Stemming	NB	LR	RF	XGB
One-Hot (seconds)	1.54	15.94	157.90	2738
Stemming (seconds)	0.82	7.81	377.10	1499
Efficiency (seconds)	0.72	8.13	-219.2	1239

Table 3: Table for efficiency after hyperparameter tuning

7 Conclusion and Discussion

7.1 Conclusion

In Table 2, we observe that for one-hot and stemming encoders, the naive Bayesian model performs the worst and the xgboost performs the best. This outcome is not surprising since the multinomial naive Bayesian model is not designed for accuracy as much as it is for speed, and the xgboost is an ensemble model devised for performance.

One interesting phenomenon is that with respect to logistic regression, the marginal gain in performance for the random forest is insignificant relative to the marginal gain for the xgboost. One explanation is that while the random forest blindly averages performances of randomly configured decision trees trained on numerous subsets of data, the xgboost employs a more meticulous gradient boosting technique that deploys weak learners with the specific goal of reducing residual errors of the existing learners.

As for the pretrained BERT encoders, we observe that the base cased model performs the best. For both the base and large models, we see that the test accuracy is higher for the cased models. One reason is that unlike uncased models that do not distinguish between upper- and lower-cased letters, cased models do make such distinctions, which would have led to a better contextual learning of the English language. It comes as a surprise, however, that the accuracy scores for large models are lower than for base models. This difference in performance may be statistically insignificant. In a different compilation cycle, we in fact see higher accuracy scores for large models compared to base models. We see that the base cased model fine-tuned for binary classification beats the xgboost combined with either of the one-hot and stemming encoders. On one hand, this is not surprising as the BERT encoder is considered state-of the-art for many NLP tasks. One the other hand, this difference in performance may not be meaningful enough to draw any conclusion provided that the xgboost is not hyperparameter-tuned and the accuracy scores for fine-tuned BERT models are unstable.

Another interesting observation is that although the one-hot encoder tends to be slightly better than the stemming encoder, the difference in performance is negligible. However, a substantial difference is observed on the efficiency front, where except for the random forest, all the other models trained on data preprocessed with the stemming encoder take less CPU execution time than those trained on data preprocessed with the one-hot encoder. The reason for such a low efficiency for the random forest is that it is trained with about 189 features considered for the best split and 100 trees in the forest for the one-hot encoder, compared to 10 features considered for the best split and 500 trees in the forest for the stemming encoder. Although the number of features considered is greater for the one-hot encoder, the bottleneck must have occurred with the number of trees in the forest, causing the stemming encoder to take much more CPU time to execute.

Just for fun, we lastly compare our results to analogous results in [Wang *et al.*, 2020]. Any conclusion that one draws from these comparisons would be devoid of meaning since the settings are quite different (e.g., binary versus multi-class classification), and even if they were the same, speaking of percentage-point differences in accuracy measures have no meaning until shown statistical significance. With this in mind, the paper reports two separate tables for test accuracy for numerous combinations of encoders and multi-class classification models, depending on the size of the dataset. In either table, we only look at their accuracy scores reported for the "TF-IDF" encoder, which is essentially the same encoder as our one-hot and stemming encoders, and "NB", "LR", and "XGB", which are three of the four models that we use. For the smaller dataset, they report test accuracy scores of 78.90%, 83.30%, and 90.30% for NB, LR, and XGB, respectively. Their scores for NB and LR are comparable to ours, but their XGB score is higher than our highest XGB score of 84.82%. As for the BERT encoder, their best score of 86.00% is obtained when paired with multi-layer perceptron (MLP), which is comparable to our score of 85.06% for the based cased model.

7.2 Limitation and Future Work

As mentioned in the Dataset section, our biggest challenge was the lack of computing resources. Until we found the `simpletransformers` library, we were unable to make an end-to-end BERT model with fine-tuning because the GPU RAM of Google Colaboratory reached the limit with just 36% of our data.

[Wang *et al.*, 2020] reports that the SBERT encoder provides the best overall performance across models. In this light, our project can be expanded to include more sophisticated pretrained encoders such as SBERT, RoBERTa, and T5.

References

[Devlin *et al.*, 2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of NAACL-HLT*, pages 4171–4186, June 2019.

[UNICEF, 2020] UNICEF. *COVID-19 and its implications for protecting children online*. UNICEF, 2020.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[Wang *et al.*, 2020] Jason Wang, Kaiqun Fu, and Chang-Tien Lu. *SOSNet: A Graph Convolutional Network Approach to Fine-Grained Cyberbullying Detection*. IEEE, 2020.