Machine Learning for Sentiment Analysis

Avery E. Peiffer
ECE, University of Pittsburgh
Pittsburgh, PA, USA
aep65@pitt.edu

Daniel C. Stumpp
ECE, University of Pittsburgh
Pittsburgh, PA, USA
dcs98@pitt.edu

***Abstract*—Emotional sentiment analysis can be used in a wide range of of applications, from gauging public opinion to studying the impact of social media on mental health. Despite the ease with which humans can perform sentiment analysis, the task poses a more substantial challenge for computers. We perform an investigation of a state-of-the-art transformer-based machine learning method called RoBERTa [1] to address this challenging task. We also evaluate this model's performance against other alternative solutions based on more traditional machine learning techniques. Results and findings are reported in detail in this report. [1] We show that state-of-the-art transformer methods achieve a classification accuracy about 5% higher than traditional decision tree based methods, however, the low complexity of decision tree based methods makes their further investigation worthwhile.**

***Index Terms*—Machine learning, sentiment analysis, natural language processing, transformers, random forest classifiers, gradient boosting, XGBoost, BERT**

I. Problem Description

Humans possess an innate ability to determine emotion from conversation without explicit cues. We automatically collect samples of a speaker's speech patterns, body language, choice of vocabulary, and the surrounding context, for example, as data points from which we construct an overall picture of their emotion. This subconscious analysis then informs our decisions and actions as we interact with those around us. While this task is incredibly simple for humans, determining emotional sentiment is much harder for a computer.

Sentiment analysis is a branch of computer science that aims to use machine learning techniques to automatically determine emotion conveyed through text. This work explores sentiment analysis using Twitter status updates (tweets) as samples for evaluation. The related work, solutions, and experiments will be outlined in the following sections.

II. Related Work

The following section provides a detailed review of the work in the space of NLP and sentiment analysis. Some methods not used in this study are discussed to provide a thorough overview of the state-of-the-art.

A. Methods for text feature extraction

In order for text data to be processed with machine learning algorithms, it must be distilled from a list of words into a set of workable features. This section provides a survey of various methods for extracting features from text, including traditional methods and newer, graph-based methods.

1) Traditional methods: Waykole and Thakare [2] conducted a review of three traditional methods for extracting features from text. The simplest method is the bag of words (BoW) method, in which each word in a block of text is compared to a lexicon of known words, and its frequency in the text data is counted. However, this method fails to identify words that are unique and important to a specific document, as their frequency counts are dwarfed by common words that make up the basis of a language. To mitigate this issue, each word can instead by assigned a Term Frequency - Inverse Document Frequency (TF-IDF) score. The term frequency is the frequency of the word in the existing document, and the inverse document frequency is the score of the word among all documents. This allows words that are important and unique to specific documents to be scored correctly.

The most complicated traditional method is Word2Vec, a series of two-layer neural networks that represent semantic contexts of words. It maps an entire corpus of text to a high-dimensional vector space, in which words with common contexts occupy similar locations in the vector space.

2) Graph methods: The authors of [3] introduced a graph-based algorithm to create an emotion-rich representation of text for emotion recognition tasks. An emotion graph was created using objective and subjective tweets scraped from the Twitter API. Network metrics were calculated for this graph, such as the eigenvector centrality and clustering coefficient of each vertex (which represent the tokens from the overall corpus). Vertices with high values in either of these metrics were used as candidates for a bootstrapping process to generate patterns of opinionated words. The authors then trained a neural network on pre-trained word embeddings from [4] to compare to the bootstrapped patterns. This process resulted in only emotion-rich patterns; that is, patterns that are present throughout the graph dataset and the pre-trained word embeddings. These patterns were then evaluated on several emotion recognition tasks and outperformed existing state-of-the-art techniques.

B. Basic classification techniques for sentiment analysis

Neethu and Rajasree [5] performed a general survey of classification techniques using a dataset of tweets about electronic products. The dataset was preprocessed to remove misspellings and slang words. For each sample in the dataset,

[1]Code available at https://github.com/danielstumpp/sentiment-analysis

a feature vector was created by extracting the positive and negative keywords from the tweet, including emoticons and hashtags. The authors then used three different classification techniques on their dataset for sentiment classification: naive Bayes, SVM, and maximum entropy. Finally, an ensemble classifier was created using the three previous techniques as base classifiers. All four methods had similar accuracies, ranging around 90%, when evaluated on the test dataset.

Rathi et al. [6] performed a similar evaluation of sentiment analysis using three classifiers: SVM, decision tree, and Adaboost decision tree. This work used three large and robust datasets, including the Stanford Sentiment140 dataset, containing over 1.6 million tweets [7]. The authors observed experimental accuracies of 82%, 67%, and 84%, respectively.

C. CNN and LSTM architectures for sentiment analysis

Sosa [8] introduced a combined LSTM-CNN architecture for sentiment analysis on Twitter data. They trained and validated their models on a large dataset compiled from a University of Michigan Kaggle competition [9] and Neik Sanders' Twitter Sentiment corpus [10], containing over 1.5 million total tweets labeled as positive or negative. The LSTM-CNN architecture used is shown in Figure 1. The combined LSTM-CNN architecture outperformed both a LSTM and a CNN architecture individually, achieving an average accuracy rate of 75.2%, compared to 66.7% for the CNN architecture and 72.5% for the LSTM architecture.

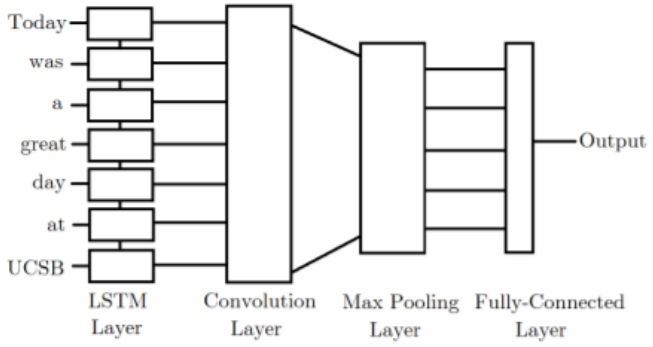


Fig. 1: Sosa's LSTM-CNN architecture for sentiment analysis

Chen and Wang [11] built on Sosa's implementation by adding an encoder/decoder framework to the existing LSTM-CNN architecture. The CNN is regarded as the encoder, corresponding to a two-layer deconvolution layer as a decoder. The architecture is shown below in Figure 2. The authors state that this structure allows the CNN to learn features more intrinsically and effectively. On the same datasets, the authors were able to achieve an accuracy of 78.6%.

D. Transformers

The architectures discussed in previous sections rely heavily on recurrent or convolutional neural networks that use an encoder and a decoder. The complexity of these models results in a high training cost, in the range of 10^{20} floating point operations per second (FLOPs). Vaswani et al. [12] introduced a much simpler architecture called the transformer, which does not need recurrence or convolution. The transformer architecture is shown below in Figure 3. This architecture relies solely on attention mechanisms [13], which are designed to focus on critical data while ignoring the rest. The transformer architecture improved on existing results for the WMT 2014 English-to-German translation task at a fraction of the training cost. The authors also showed that this architecture can be applied to English constituency parsing, which breaks down sentences into their syntactic structures, proving its worth in general NLP tasks.

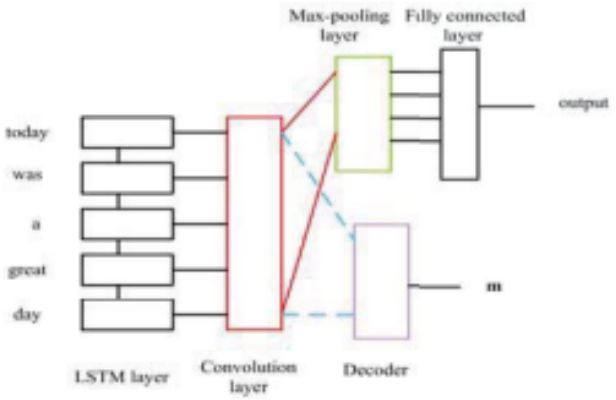


Fig. 2: Chen's and Wang's LSTM-CNN-encoder-decoder architecture, improving upon Sosa's sentiment analysis accuracy

In 2018, Devlin et al. [14] introduced a language representation model called BERT, which stands for Bidirectional Encoder Representations from Transformers. BERT makes use of transformers to pretrain bidirectional representations from unlabeled text, meaning that it learns representations by reading a sentence from left to right and right to left [15]. This bidirectional pretraining allows the model to understand syntax and semantics of a sentence at a level greater than can be achieved with unidirectional pretraining. The text below shows an example of BERT's pretraining, adapted from [15].

The letter of recommendation was sent to your ***address****.*

Leaders across the globe have ***addressed*** *their people on the pandemic.*

The word *address* is used in different contexts in each sentence. In order to understand the meaning of the word in each sentence, it is necessary to read the entire sentence before making a judgment. The BERT model is able to extract this meaning correctly by reading the sentence in both directions. After the pretraining process is complete, the BERT model can then be adapted to perform NLP tasks with an additional output layer.

III. Dataset

This section outlines a preliminary analysis of the tweet dataset being used [3]. Table I outlines the basic information

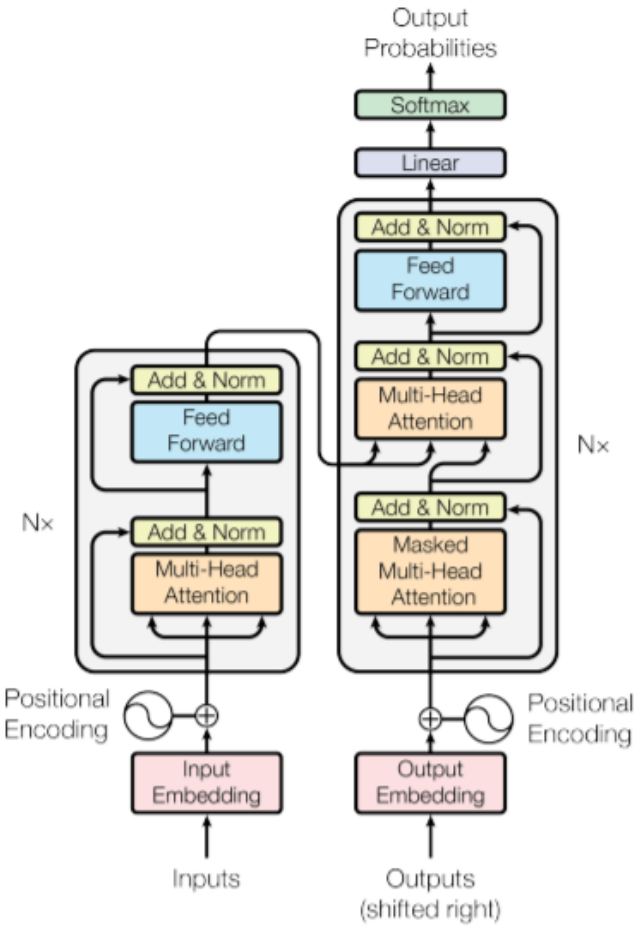


Fig. 3: The transformer architecture presented by Vaswani et al.

regarding the dataset. The dataset includes train, test, and validation splits, which represent 80%, 10%, and 10% of the samples, respectively. For models or methods that do not require cross-validation, the validation set will be combined with the training set to represent a 90-10 train-test split. The total number of samples in the dataset is 20,000.

TABLE I: Tweet dataset information and statistics.

statistic	value
classes	{sadness, joy, fear, anger, love, surprise}
splits	{train, val, test}
train samples	16000
val samples	2000
test samples	2000

To effectively use the dataset, it is important to understand the distribution of classes. Fig 4 shows that the distribution of class labels is not balanced. The imbalanced nature of the dataset is important when selecting and training models for classification. Joy is the most prevalent class label with nearly 7000 instances. Sadness is the second most common class at just under 6000 instances while all other classes occur less than 3000 times.

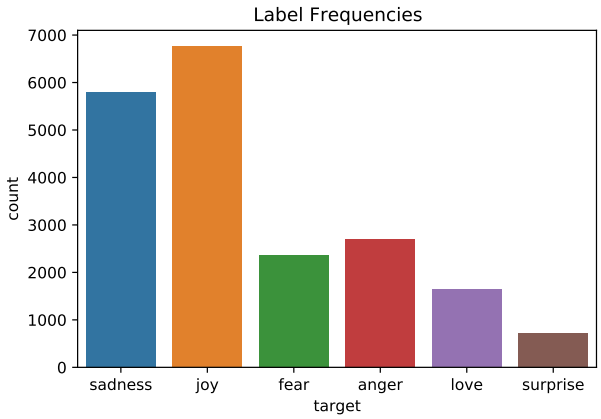
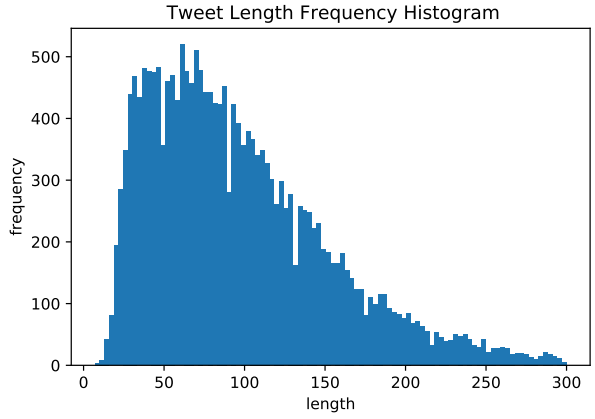


Fig. 4: Dataset label frequencies. The plot shows the imbalanced nature of the dataset being used.

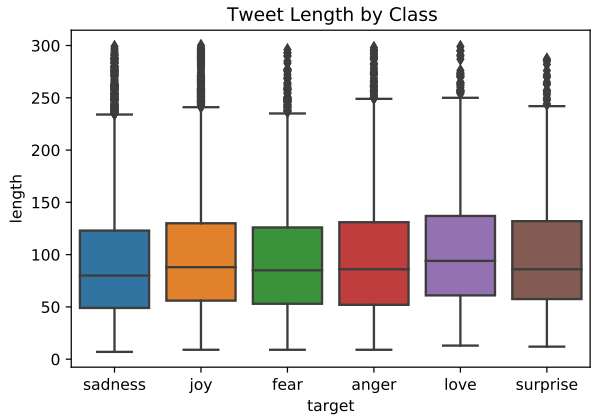
Another characteristic of the dataset considered is the character length of the tweets. Fig. 5(a) shows a histogram distribution of the tweet character lengths in the dataset. The mean tweet length is 96.7 characters. The plot in Fig. 5(b) shows box plots for tweet length for each different class label. From this data we can see that the tweet length appears to have the same characteristics regardless of the tweet class. This indicates that tweet length may not be a salient feature for classification of tweets.

The last characteristic of the dataset considered is the word distribution in each class. Specifically, we are interested in seeing what words have high frequency in one class but not in the others. To get a sense for the what key words are in each of the classes, the top 15 unique words from each class were plotted in a histogram as shown in Fig. 6. These unique words histograms were found by iteratively determining the 15 most frequent words in each class and then adding each word that appeared in more than one class to an exclusion list. This process was repeated until no new words were added to the exclusion list; thus, all the words are unique across all classes. This gives a good representation of the most significant words for the classification of each class. This process resulted in exclusion of 85 words. These words tend to be frequently used words that have no significant relationship to one specific sentiment class. Examples of these words are ‘and’, ‘the’, ‘like’, ‘but’, ‘really’, ‘can’, ‘your’, ‘need’, and ‘know’. Intuitively, these words provide no insight as to what the sentiment of the tweet would be. Likewise, these features likely have little significance for machine learning methods. These words and ones like them should be excluded during training through the use of a stop list.

The words that represent the top 15 unique words for each class are more representative of the sentiment, and therefore useful for classification. This is not the case for all words shown in Fig. 6, however it is for the majority. Some examples



(a)



(b)

Fig. 5: (a) Distribution of tweet character lengths and (b) box plots for tweet lengths by class.

of descriptive words are ‘bad’ and ‘alone’ for sadness, ‘good’ and ‘happy’ for joy, ‘anxious’ and ‘terrified’ for fear, ‘angry’ and ‘irritable’ for anger, ‘sweet’ and ‘caring’ for love, and ‘amazed’ and ‘impressed’ for surprise. These words intuitively have high correlation with the sentiment class that they belong to, meaning that these words in a tweet will often be a good indication of the sentiment of that tweet. This will be considered when selecting and tuning a method to extract features, and when considering what types of features to use for training of a machine learning model to classify tweet sentiment.

A. Dataset Limitations

It is important to note a few limitations of the dataset being used. One is the size of the dataset. The dataset includes 16000 training samples, which is significant and larger in size than some other datasets used in literature; however, this size still makes overfitting a significant risk on more complex models. Another problem, as already introduced, is the imbalanced nature of the dataset. While this may be in part due to the natural prevalence of different types of sentiments in tweets, it adds additional challenges to training the model. Special care must be taken to ensure that the model developed can effectively classify all classes.

Finally, the subjectivity of the problem makes the accuracy of the ground truth labels questionable at times. Some tweets could easily demonstrate characteristics of two or more of the classes, making the ground truth difficult to assign and ultimately inaccurate at times. There are also tweets that could simply be misinterpreted based on lack of context or misunderstanding when the ground truth values were being assigned, leading to errors in the dataset. Additionally, there are some tweets and phrases that may carry a neutral sentiment. This may occur with objective statements of facts, questions being asked, or opinions being stated. Forcing tweets of these types into one of the defined classes could lead to misleading training data and ultimately errors in the trained model. It is for these reasons that overfitting should be avoided as much as possible so that the model can perform well on unseen data.

IV. Proposed Solutions

This section outlines proposed solutions to the problem of sentiment analysis of tweets. The implementation and results of these solutions will be outlined later in Section V.

A. Text Feature Extraction

A text-to-feature extraction method must be chosen for the decision tree based methods being explored. The two methods considered were bag-of-words (BoW) and TF-IDF. Preliminary analysis of these methods using random forest and bagging classifiers showed that they perform nearly the same on the dataset. Although TF-IDF typically has some advantages, those are not seen in this case, likely due to the small length of the tweets. We therefore chose the BoW feature extractor to be used for the decision tree methods explored in this study.

We make use of the *CountVectorizer* method of the *scikit-learn* Python machine learning package to build the BoW and to transform test data into the feature space [16]. This method provides a stop words list for common English words that don’t have significant value in discriminating between classes. We utilize that stop list to remove those words and help to reduce the overall feature space. We configure the BoW to only contain single words instead of using any higher order N-grams. For a word to be included in the BoW, it must appear in at least 5 training samples and also not appear in more than 50% of training samples. This ensures that words that are so rare or so common, such that they would provide no benefit to classification, are not included as features.

When the BoW is trained using the training data and the parameters outlined above, the result is a feature vector with 3397 features. This large number of features means that overfitting is possible for the decision tree methods. Careful tuning must be performed to ensure that overfitting is reduced and the model is generalizable to test data. However, reducing the feature space further would limit the accuracy of the model

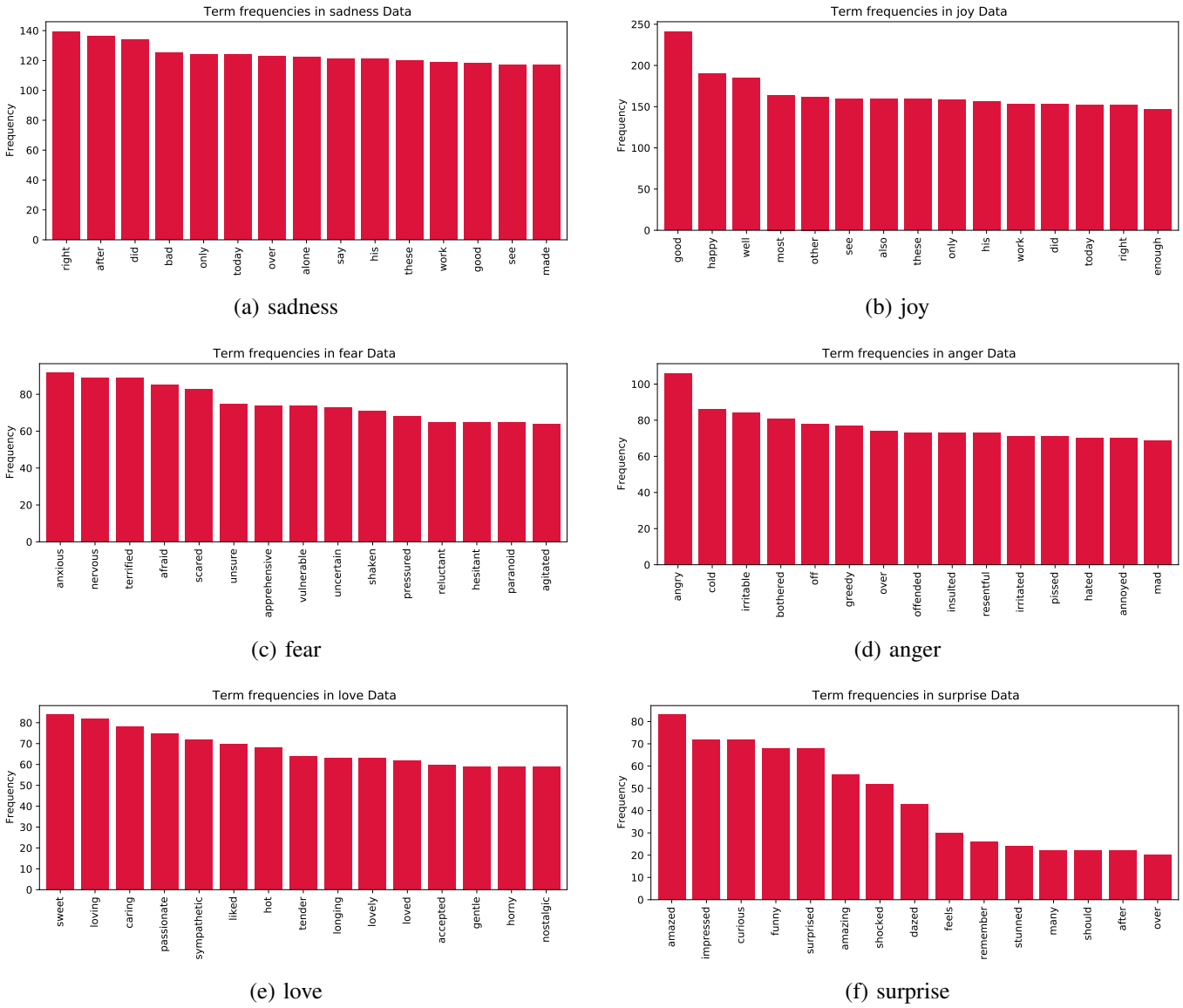


Fig. 6: Top 15 unique words for each class.

due to the removal of important features from the feature space.

As stated previously, BERT [14] and other transformer-based architectures rely on word representations that preserve context within a sentence. To accomplish this, BERT creates a set of pre-trained word representations by reading a sentence from left to right and from right to left. To accomplish this, some percentage of the input tokens in each sentence are masked so as to not be visible. These tokens are then predicted and used to train a deep bidirectional transformer. This masked language model (MLM) process allows the transformers to infer greater context about masked tokens than in other feature extraction methods.

B. Traditional Methods

In this research, we have selected tree based methods to be the traditional, non-deep-learning methods to be evaluated. We choose bagging, random forest, AdaBoost, and XGBoost gradient boosting as four types of decision tree based classifiers to investigate in this work. Initial investigation showed bagging and AdaBoost to perform the worst out of the four methods. For that reason, we focus our study on the use of random forest classifiers and gradient boosting classifiers.

1) Random Forest: To implement random forest classifiers for this work, we used the *RandomForestClassifier* provided by *scikit-learn* [16]. Random forest classifiers have relatively few hyperparameters and often require less tuning than other types of classifiers. The three hyperparameters that we focus on, and that have the largest impact on the performance of

TABLE II: Original results for the RoBERTa model with a learning rate of 1×10^{-4}

	Precision	Recall	F1-Score	Support
Sadness	0.966783	0.951807	0.959327	581
Joy	0.964018	0.925180	0.944200	695
Love	0.782609	0.905660	0.839650	159
Anger	0.920863	0.930909	0.925859	275
Fear	0.908654	0.843750	0.875000	224
Surprise	0.681319	0.939394	0.789809	66
Accuracy			0.923500	2000
Macro avg	0.870708	0.916117	0.888959	2000
Weighted avg	0.928936	0.923500	0.924889	2000

the classifier, are the number of estimators (i.e. number of trees), the maximum number of features considered for each split, and the maximum depth of a tree in the classifier. The selection and tuning of these parameters will be discussed in further detail in Section V. The split quality criterion is also a parameter that can be set. We keep this at the 'gini' score as is typically used.

2) Gradient Boosting: For the implementation of gradient boosting we will use the XGBoost python package [17]. XGBoost provides a large number of hyperparameters along with different tree growing techniques and methods. It also provides many configuration parameters that can be set based on the desired configuration of the training algorithm. We propose and implement extensive cross-validation in order to tune the XGBoost classifier. To achieve this we utilize the *GridSearchCV* method provided by *scikit-learn* along with the built in cross-validation functionality of the XGBoost package. Further details about the parameters tuned and the results of that tuning is outlined in Section V.

C. Transformer Methods

As discussed in our progress report, we have been working with an implementation [18] of RoBERTa [1], which is an extension of BERT that improves its pretraining. For our progress report, we updated the out-of-date parts of the RoBERTa implementation, and we reported the results of the RoBERTa code on a large tweet dataset. Our initial experiments showed that the RoBERTa model could achieve a superior performance to other methods, and its results are shown in Table II below.

V. Experiments and Results

This section outlines the experiments performed in this research. The results collected from this study are also discussed in detail. Analysis is performed as results are discussed and in further sections.

A. Traditional Methods

1) Random Forest: Initial testing of the random forest classifier produced an accuracy of 88% with a weighted F1-Score of 0.88. This initial test used only 10 estimators to achieve these results. Evaluation of the impact of the two hyperparameters considered (n_estimators and max_features) showed that the results of classification with random forest is not sensitive to hyperparameter tuning with the exception of the selection of extreme values. The results of these tests can be seen in Fig. 7 and Fig. 8. To maximize accuracy, the max depth of the trees is left at the default of none, implying no defined maximum.

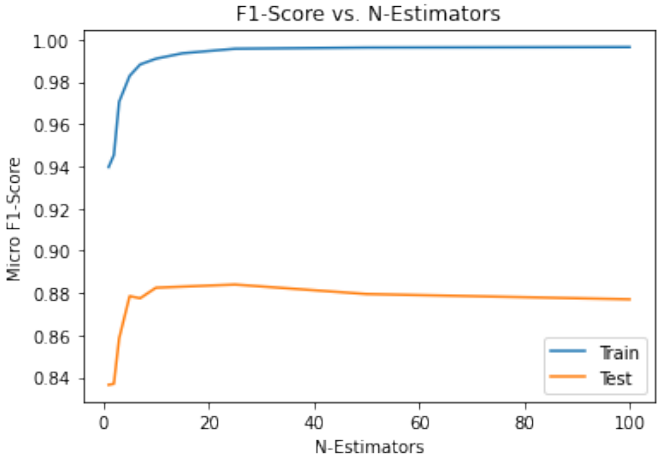


Fig. 7: Impact of number of random forest estimators on test and train F1-score. All other parameters are set to the default value.

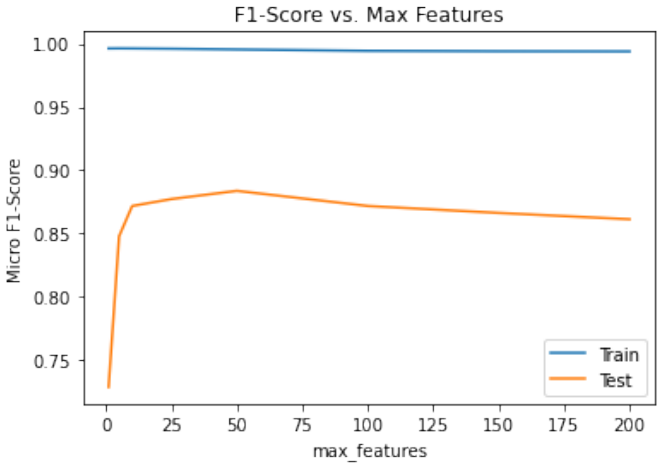


Fig. 8: Impact of max number of features on test and train F1-score. The number of estimators is set at 25.

These plots show that, except for very few estimators or very low values of the maximum number of features considered, the test micro F1-score remains consistently above 0.85. These plots also show evidence that indicate overfitting may be occuring. The training F1-score consistently reaches near 0.99, about 0.1 higher than the best test score. This slight overfitting is likely primarily due to the large number of features used in the bag-of-words feature vector along with no limit being placed on the depth of the trees. Despite this, the random forest still performs well at the sentiment classification task, indicating that the model still generalizes well to unseen data. Fig. 9 shows the resulting confusion matrix of the best random

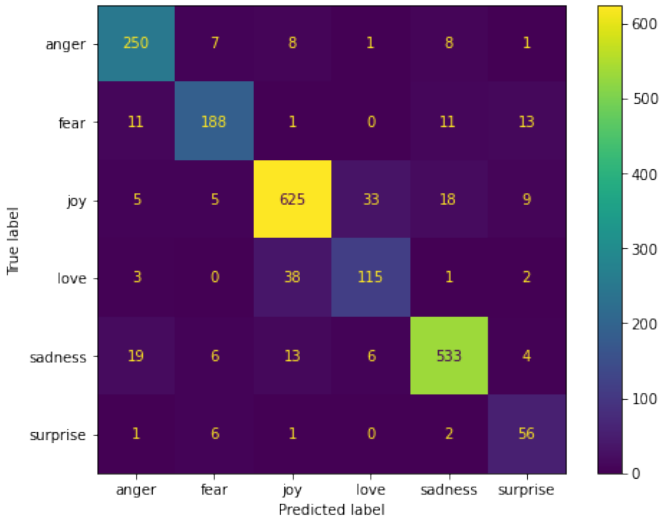


Fig. 9: Confusion matrix for best random forest classifier. Results predicted on test set. Number of estimators is 25 and max features is set to 50.

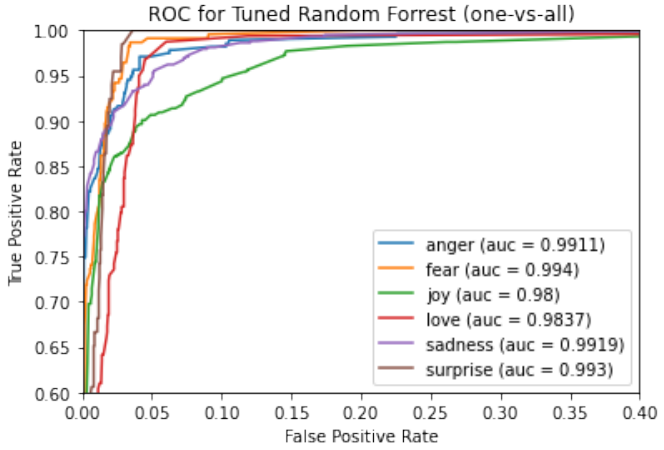


Fig. 10: ROC plot for best random forest classifier. Results predicted on test set. Number of estimators is 25 and max features is set to 50.

forest classifier's prediction on the test data. The confusion matrix shows that, although the dataset is imbalanced, the classifier is not biased to one or two of the classes. Fig. 10 shows the one-vs-rest ROC curves for the classifier. The area under curve metric is also shown in the legend. This helps to visualize the tradeoff between false positive rate and true positive rate. For example, the threshold for the 'joy' class to reach a true positive rate of 95% corresponds to a false positive rate of about 11%. This is nearly twice as high as any other class and shows that the joy class is more prone to false positives, likely because it is the majority class.

2) Gradient Boosting: The XGBoost configuration parameters for training the gradient boosted decision tree models is shown in Table III. These configuration parameters remained the same throughout all experiments.

TABLE III: XGBoost training configuration parameters.

parameter	description	value
objective	learning task objective	'multi:softprob'
n_class	number of classes	6
tree_method	tree construction algorithm	'hist'
eval_metric	metric for validation	'mlogloss'
nthread	parallel threads (-1 is all available)	-1

XGBoost has many more hyperparameters that can be tuned than do random forest classifiers. Because of this, it is important to outline a methodical tuning approach that will effectively tune the many parameters of the model to achieve improvement in classification performance. The tuning methodology used for this research is outlined in Algorithm 1. The approach makes extensive use of cross-validation to enable the selection of hyperparameters without biasing the test results. The ranges of hyperparameters used for cross-evaluation were determined based on initial testing, knowledge of the default values, and computing constraints.

Algorithm 1 Pseudocode for XGBoost tuning.

1: Load Data
2: Set initial model parameters
3: Use early-stopping cross-validation to determine initial $n\_estimators$
4: Coarse tuning of max_depth and min_child_weight using grid search cross-validation
5: Fine tuning of max_depth and min_child_weight based on coarse tuning results
6: Coarse tuning of $gamma$ using cross-validation
7: Fine tuning of $gamma$ based on coarse result
8: Use early-stopping cross-validation to re-calculate the value of $n\_estimators$
9: Tune $colsample\_bytree$ and $subsample$ using grid search cross-validation
10: Tune reg_lambda using cross-validation
11: Halve learning rate and update $n\_estimators$ using early-stopping cross-validation
12: Evaluate tuned model

The initial values for the hyperparameters were set using estimated best values based on knowledge of the problem and examples of other similar design. These initial un-tuned hyperparameter values are shown in Table IV. Multiclass log loss is used as the evaluation and scoring metric for cross-validation due to its consideration of probability instead of just discrete predictions. This means it will consider not only whether the correct class was chosen, but the confidence of the model as well. The initial un-tuned model has a test log loss of 0.272 and a micro F1-score of 0.89. Micro F1-score is used because it considers precision and recall and also considers the support of each class.

After tuning is performed as outlined in Algorithm 1, the

TABLE IV: Un-tuned and tuned hyperparameters for XGBoost models.

hyperparameter	Un-tuned Value	Tuned Value
learning_rate	0.3	0.15
n_estimators	519	1791
min_child_weight	2	0
gamma	0	0.2
subsample	0.9	1
colsample_bytree	0.9	0.7

new values for the hyperparameters are found as shown in Table IV. This tuned model has a test log loss of 0.245 and a micro F1-score of 0.90, equating to an approximate improvement of 9.9% and 1.1% respectively for the metrics. This improvement shows that tuning had a positive effect on the performance of the model. The larger improvement in log loss indicates that the model became more confident in its predictions while the F1-score represents an improvement in finding the correct class for a given tweet. The standard accuracy of the model also improved from 89% to 90% due to tuning of the model.

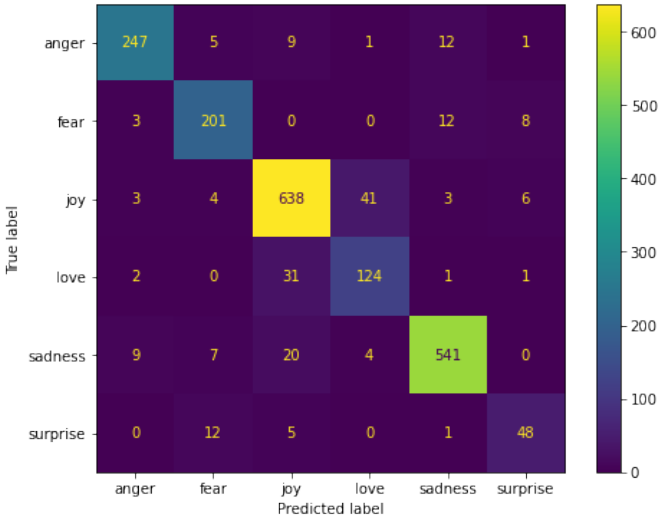


Fig. 11: Confusion matrix for tuned XGBoost classifier. Results predicted on test set.

The tuned model's confusion matrix and ROC curve are shown in Fig. 11 and Fig. 12 respectively. The confusion matrix shows good performance for all classes. One notable observation is that 'fear' instead of 'surprise' and 'joy' instead of 'love' are two of the more common mispredictions. This is likely due to the similarity and crossover between these two classes. Many tweets that are labeled with the 'fear' emotion may also exhibit characteristics of surprise. This means some of this error is likely due to subjectivity and class overlap in labeling the tweets. The ROC curve for the tuned XGBoost model shows similar characteristics to the one shown for the random forest classifier; however, in general the curves are shifted slightly up and to the left, indicating the higher overall

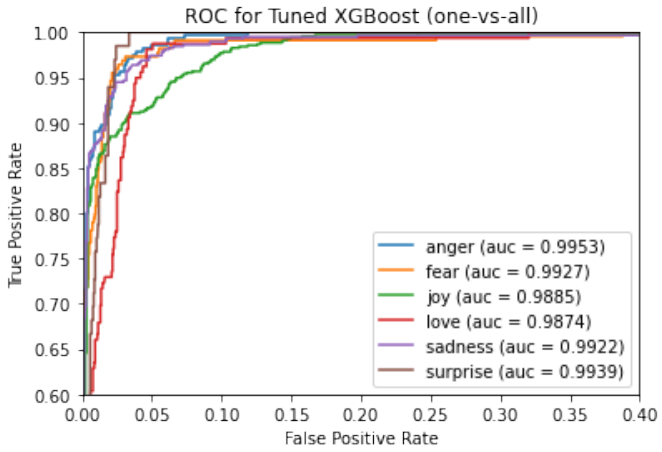


Fig. 12: ROC plot for best XGBoost classifier. Results predicted on test set.

performance of the XGBoost model.

B. *Transformer Methods*

We have used our additional time to try and assess the performance of the RoBERTa model in detail, in order to have a standard against which we can compare our other methods. The tutorial code includes an interface that makes use of LRFinder [19], a module which estimates the optimal learning rate of a model through a range test detailed in [20]. There were several issues with this section of the tutorial notebook that caused it to not work at first; as a result, we decided to use a tame learning rate estimate of 1×10^{-4} for our progress report experiments. We were able to dedicate time to fix this code and eventually got the LRFinder interface to work correctly. Running the LRFinder code resulted in a suggested learning rate of 9.18×10^{-5}, which was used for our additional experiments. The results with this suggested learning rate are shown below in Table V. Interestingly, the suggested learning rate does not seem to offer a significant performance upgrade relative to the learning rate estimate we used for our first round of experiments. This could mean that the LRFinder code was incorrect, and the learning rate of 9.18×10^{-5} was unnecessarily small for the model.

Because the learning rate tends to be the most important hyperparameter in dictating a model's performance, and experimenting with our model's learning rate did not seem to result in any significant improvements, we decided to employ cross validation on the train and test sets to get a more accurate measure of the model's performance with respect to other hyperparameters. We tested models that varied in the number of epochs and the batch size. For the number of epochs, we tried 1, 2, and 3 (to not stress the memory while training); for the batch size, we tried 8, 16, and 32. All of the results were comparable to the original experiment, with the best results on the train and test sets coming with 2 epochs and a batch size of 32. This is not to say that the best model has these hyperparameters, as the test set cannot be included in cross

TABLE V: Results for the RoBERTa model with the suggested learning rate of 9.18×10^{-5}

	Precision	Recall	F1-Score	Support
Sadness	0.962199	0.963855	0.963027	581
Joy	0.954142	0.928058	0.940919	695
Love	0.784091	0.867925	0.823881	159
Anger	0.953125	0.887273	0.919021	275
Fear	0.887892	0.883929	0.885906	224
Surprise	0.666667	0.878788	0.758170	66
Accuracy			0.921500	2000
Macro avg	0.868019	0.901638	0.881820	2000
Weighted avg	0.925917	0.921500	0.922834	2000

TABLE VI: Sample results for the RoBERTa model after varying hyperparameters (learning rate = 9.18e-5, epochs = 2, batch size = 32)

	Precision	Recall	F1-Score	Support
Sadness	0.966044	0.979346	0.972650	581
Joy	0.960352	0.941007	0.950581	695
Love	0.828221	0.849057	0.838509	159
Anger	0.944238	0.923636	0.933824	275
Fear	0.873418	0.924107	0.898048	224
Surprise	0.819672	0.757576	0.787402	66
Accuracy			0.934500	2000
Macro avg	0.898658	0.895788	0.896836	2000
Weighted avg	0.934907	0.934500	0.934510	2000

validation that aims to optimize hyperparameters. Instead, these results act as a realistic upper bound for the model's performance in different situations. These results are reported below in Table VI. Overall, this exploration of hyperparameter values for the RoBERTa model shows that it is accurate, capable of consistently achieving an accuracy of 92-93%.

VI. Discussion and Analysis

In the previous section, we outlined the experiments performed and results for each model type. These results are summarized in Table VII. We observe that RoBERTa performs the best at the classification task, followed by XGBoost and then random forest. It is not surprising that RoBERTa achieved the highest performance, as it is the most complex model and is heavily optimized for NLP tasks. RoBERTa also utilizes a more robust text-to-feature extraction method than the bag-of-words method utilized for the decision tree models, which likely contributes significantly to the difference in performance. Despite the feature extraction and complexity advantage of the transformer model, the decision tree methods still show strong performance, achieving accuracy within about 5% of that achieved by RoBERTa.

Another critical takeaway from this work is the dependence on good ground truth labels for the dataset, as well as the effect of class overlap in the dataset. In an effort to minimize the effect of class overlap on our scoring metrics, we evaluated the top-2 accuracy for each model. These scores show substantially higher performance with a gain of about 9% for both decision tree methods as opposed to the standard accuracy metric. This indicates that our hypothesis, that class overlap was creating most of the errors, was correct. It also indicates that, with better defined classes and reduced class overlap, the models will perform with extremely high accuracy. Interestingly, on the metric of top-2 accuracy the XGBoost model actually performs slightly better than the RoBERTa model, again emphasising the high performance potential of gradient boosting as a method for sentiment analysis.

TABLE VII: Summary of best sentiment analysis results.

	Random Forest	XGBoost	RoBERTa
Accuracy	0.884	0.900	0.935
Macro F1-Score	0.842	0.858	0.897
Top-2 Accuracy	0.981	0.989	0.987

The results observed show that multiclass sentiment analysis can be performed effectively using decision tree models, specifically random forest and XGBoost. Both these models are significantly less complex than BERT based models and therefore have faster training and prediction times. These advantages may make XGBoost or random forest better options than RoBERTa in cases where compute resources are limited or the application needs to be deployed for an edge application, and where a small trade-off in accuracy is acceptable. That even relatively simple models can achieve accuracy in the realm of 90% serves as evidence that computers can learn to classify sentiment with ease at the scale of a tweet.

Future investigations of the same topic could apply the sentiment analysis models to data of longer length and compare the performance to that achieved with the Twitter datasets. For example, Facebook status updates do not have a maximum length, and therefore afford users the space to convey much more nuanced emotions than on Twitter. In a dataset with the same set of six labels, we would expect the models' performances to improve, albeit with a higher training cost, as the longer status updates give more information about the various emotions.

VII. Conclusion

In this research we have explored machine learning for sentiment analysis. We have utilized a dataset consisting of tweets classified as one of six sentiments for training and evaluation of our models. We evaluate both traditional and state-of-the-art machine learning methods for the task. For the traditional methods, we utilized two decision tree based methods: random forest and gradient boosting. For the state-of-the-art transformer based natural language processing method we use RoBERTa, a version of BERT. This work goes beyond some existing literature by expanding decision tree methods to a multiclass sentiment analysis problem as opposed to a simple binary classification, which is most prevalent in literature. Cross-validation is used to tune models and select hyperparameters for optimal performance for the classification task. We've outlined in this report the steps and methods taken to achieve optimal tuned models for each of the three methods considered.

The results of this research show that RoBERTa achieves an accuracy of 93.5% followed by XGBoost gradient boosting and random forest at 89.7% and 88.4% respectively. These results show that the state-of-the-art transformer based method performs best as expected; however, the decision tree based methods only incur a small reduction in accuracy while providing a more compute-efficient solution to the sentiment analysis task. This study indicates that consideration of decision tree based methods is warranted in cases where large BERT-like NLP models are not feasible based on resource or time constraints.

We were able to apply concepts learned throughout ECE 2195 to this project, especially what we learned about assessing the performance of a model. We made considerable use of accuracy, precision, recall, and confusion matrices to establish a method of directly comparing our various models. The review of related work in sentiment analysis provided context to some of the more complicated machine learning architectures that we touched on in the latter part of the class, such as LSTM architectures. Overall, we found this project to be an incredibly interesting foray into the field of sentiment analysis.

REFERENCES

[1] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019.

[2] R. Waykole and A. Thakare, "A review of feature extraction methods for text classification," *International Journal of Advance Engineering and Research Development*, vol. 5, 2018.

[3] E. Saravia, H.-C. T. Liu, Y.-H. Huang, J. Wu, and Y.-S. Chen, "CARER: Contextualized affect representations for emotion recognition," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 3687–3697. [Online]. Available: https://www.aclweb.org/anthology/D18-1404

[4] J. Deriu, A. Lucchi, V. D. Luca, A. Severyn, S. Müller, M. Cieliebak, T. Hofmann, and M. Jaggi, "Leveraging large amounts of weakly supervised data for multi-language sentiment classification," 2017.

[5] M. Neethu and R. Rajasree, "Sentiment analysis in twitter using machine learning techniques," in *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*. IEEE, 2013, pp. 1–5.

[6] M. Rathi, A. Malik, D. Varshney, R. Sharma, and S. Mendiratta, "Sentiment analysis of tweets using machine learning approach," in *2018 Eleventh international conference on contemporary computing (IC3)*. IEEE, 2018, pp. 1–3.

[7] A. Go, R. Bhayani, and L. Huang, "Twitter sentiment classification using distant supervision," *Processing*, vol. 150, 01 2009.

[8] P. M. Sosa, "Twitter sentiment analysis using combined lstm-cnn models," *Eprint Arxiv*, pp. 1–9, 2017.

[9] U. of Michigan, "Umich si650 - sentiment classification dataset," 2011. [Online]. Available: https://www.kaggle.com/c/si650winter11

[10] N. Sanders, "Twitter sentiment corpus," 2011. [Online]. Available: https://github.com/zfz/twitter_corpus

[11] N. Chen and P. Wang, "Advanced combined lstm-cnn model for twitter sentiment analysis," in *2018 5th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS)*, 2018, pp. 684–687.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[13] T. P. Galassi A., Lipp M., "Attention in natural language processing," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, 2021.

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[15] R. Khandelwal, "Intuitive explanation of bert- bidirectional transformers for nlp," Apr 2020. [Online]. Available: https://towardsdatascience.com/intuitive-explanation-of-bert-bidirectional-transformers-for-nlp-cdc1efc69c1e

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[17] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 785–794. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939785

[18] M. Zabłocki, "Custom classifier on top of bert-like language model - guide," Mar 2020. [Online]. Available: https://zablo.net/blog/post/custom-classifier-on-bert-model-guide-polemo2-sentiment-analysis/

[19] D. Silva, "Pytorch learning rate finder," https://github.com/davidtvs/pytorch-lr-finder, 2020.

[20] L. N. Smith, "Cyclical learning rates for training neural networks," 2017.