Project Text Sentiment Classification

Nicolas Filimonov, Thai Nam Hoang, Ilias Marwane Merigh
CS-433 Machine Learning, EPFL, Switzerland
21 December 2023

***Abstract*—In this paper, we detail the development of a machine learning model for sentiment analysis of tweets. Our approach integrates traditional and advanced machine learning techniques, alongside comprehensive data preprocessing and vectorization methods. We report that the BERT Large model achieved the highest accuracy of 90.0% on the test data. The paper also discusses the ethical implications of sentiment analysis..**

I. Introduction

This project represents a contribution to the *AIcrowd - Text Classifier* online competition, a platform dedicated to advancing the field of Machine Learning through collaborative and competitive problem-solving. The specific challenge addressed in this competition revolves around sentiment analysis, a critical and increasingly relevant area in Natural Language Processing (NLP). Our task was to develop an ML model capable of predicting the original sentiment of a tweet, indicated by the presence of either a positive ':)' or a negative ':(' smiley. This prediction had to be made based solely on the remaining text of the tweet, following the removal of the smileys.

This task posed various challenges, primarily due to the unique vocabulary used on *Twitter* and the spelling mistakes. Sentiment analysis in such a context demands a nuanced understanding of language, including slang, idioms, or abbreviations. Our approach combined standard NLP techniques with various machine learning algorithms, focusing on understanding context, and deciphering the underlying emotional tone.

Our paper details the methodologies employed, and the ethical considerations in deploying such a model. The results section presents a comprehensive analysis of the model's performance. We conclude with potential future enhancements to improve the model's efficacy in real-world applications.

II. Methodology

We started by preprocessing the dataset to eliminate noise and normalize the text, essential for effective analysis. This was followed by vectorizing the processed data using various vectorizers to convert tweets into numerical feature vectors.
We then tested two categories of models: traditional machine learning models, including *Stochastic Gradient Descent (SGD)*[1] and *Logistic Regression*[2], and advanced deep learning models, namely *Gated Recurrent Unit (GRU)*[3], *Bidirectional Encoder Representations from Transformers (BERT)*[4] and *Robustly Optimized BERT Pretraining Approach (RoBERTa)* [5]. The choice of these models was guided by their suitability for text classification and their varying approaches to processing language data.
To evaluate the models, we used accuracy as our primary metric, dividing the dataset into an 80-20% train-validation split.

III. Data Pre-processing

A. Methods used

In the preprocessing phase of our study, we employed a series of techniques to refine the tweet dataset, ensuring its optimal readiness for analysis.

Here are a complete explanation of each techniques and the results associated to them.

1) Elimination of Duplicate Entries: The initial step in our preprocessing was the elimination of duplicate entries within the training dataset. By identifying and removing exact replicates of tweets, we aimed to prevent the model from overfitting, which occurs when a model learns the details and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This procedure ensured that our training process would generalize better to unseen data.

2) Special Character Treatment: Special characters often pose challenges for model accuracy and training efficiency. Typically, the presence of such characters can detract from a model's ability to generalize and increase the computational resources required for training. To address this, we conducted an analytical review to discern the frequency of each special character within our dataset, focusing on their distribution across the positive and negative categories.

Our strategy involved a visual assessment, plotting the occurrence of every character to determine their relevance to either sentiment category. As depicted in Figure 1, we elected to retain only a select few characters: period '.', semicolon ';', dash '-', exclamation mark '!', and parentheses '()'. These characters demonstrated a skewed distribution, suggesting a potential influence on sentiment classification.

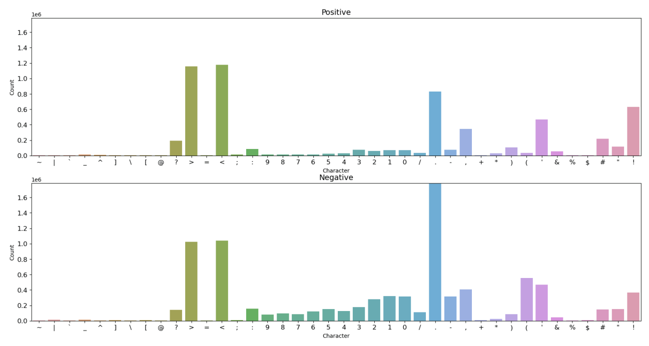

Figure 1: Frequency of special characters in the dataset

An intriguing observation was the prevalence of parentheses in the dataset. Figure 2 illustrates the count of open and closed parentheses across both sentiment categories.

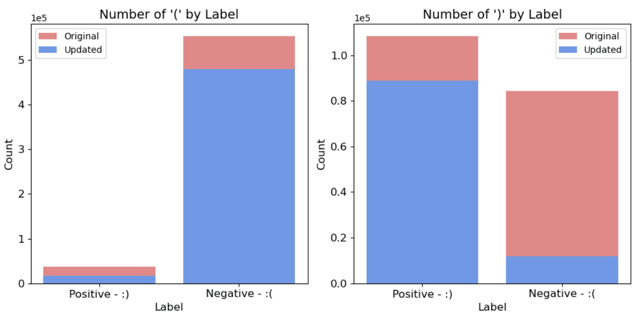

Figure 2: Initial count of parentheses in positive and negative datasets

It is important to note the initial data cleaning phase prior to the competition involved removing smiley faces represented by ':)' and ':('. However, variants with multiple parentheses like ':))' or ':((' resulted in solitary parentheses remaining in the data. To handle this, we implemented a function to eliminate pairs of matching parentheses, while singular parentheses were preserved due to their potential significance highlighted by the disparity in their occurrence, as shown in Figure 2.

The marked discrepancy in the appearance of opening and closing parentheses between classes underscored their value in sentiment analysis. Consequently, we opted against removing parentheses in our special character filtration process. Similarly, periods, dashes, and exclamation marks were also preserved due to their disproportionately high occurrence in one of the sentiment classes, thus contributing valuable discriminatory features for our model.

3) Emoji Reconstruction: Given the prior removal of standard smiley faces such as ':)' and ':(' from our dataset, we observed instances where single parentheses remained, as explained in the previous paragraph. Our preprocessing included a procedure to identify and reconstruct these solitary parentheses back into their original emoticon form. For instance, a lone closing parenthesis, when determined to be part of an emoticon sequence, was transformed back to a standard smiley face ':)'.

4) Normalization of Expressive Elements: Certain expressions hold predictive value regarding the sentiment class of tweets. However, the raw data often contains superfluous information that may obscure these valuable signals.

- Numerical figures appeared with higher frequency in negative tweets. To distill this insight while minimizing noise, we replaced actual numbers with a generic tag denoting their presence.
- Hashtags were uniformly substituted with a specific tag.
- Sequences of repeated characters were tagged to reflect their emphasis without retaining the repetitive characters themselves.
- Emoticons were systematically converted into tags corresponding to their conveyed sentiment, utilizing an emoji lexicon based on Wikipedia sources[6].
- Repetitive symbols, which are indicative of the intensity of emotion, were replaced by one instance of the symbol followed by a tag
- Word elongation is prevalent in informal communication. Elongated words were standardized to their base form and tagged to indicate the elongation.

5) Segmentation of Compound Words: Within platforms like Twitter, we encounter hashtags where multiple words are concatenated without spaces. These conglomerations of words need to be individually recognized and analyzed for accurate sentiment assessment. To tackle this, we implemented a word segmentation process. The goal was to dissect these concatenated strings into recognizable, individual words that could be analyzed independently.

6) Space Optimization Techniques: We implemented a series of space optimization techniques to enhance the integrity and readability of textual data. We began by removing trailing spaces at the end of tweets. Next, we addressed the presence of redundant spaces within the text, often resulting from other preprocessing activities. Finally, we fine-tuned the spacing around emojis to ensure their correct interpretation and association with adjacent words.

B. Importance of Sequential Order in Preprocessing Methods

The sequencing of preprocessing methods is a critical factor in optimizing model performance. The order in which these methods are applied significantly influences the quality of the processed data. Given the vast array of possible combinations of these methods, determining the most effective order was beyond the scope of this project. Therefore, while we implemented a logical sequence based on standard practices and theoretical understanding, we acknowledge that there may exist an arrangement of steps that could further enhance the model's performance, a consideration for future explorations in this domain.

C. Sampling of the dataset

To further enhance the efficiency of our development process, we initially utilized a subset of the full dataset, which comprised 250,000 tweets. This subset, considerably smaller than the full dataset of 2.5 million tweets, was instrumental in accelerating the training time during the preliminary stages of our model development. It allowed us to iterate quickly through grid search, and refine our code. Additionally, the use of cross-validation helped ensure the robustness and generalizability of our models. Once we finalized our models and code, we proceeded to train them on the entire dataset to ensure comprehensive learning and robust performance. This staged approach to dataset utilization was necessary in balancing speed and effectiveness throughout our model development process.

IV. Text Vectorization

Our initial dataset of tweets consists of text samples. However, machine learning models can only process numerical values. Consequently, we needed to convert these text strings into numerical vectors. This process is known as Vectorization. We concentrated on the following vectorization techniques:

A. Count Vectorizer [7]

- Converts text to vectors where each element is the count of a word's occurrence.
- Focuses on word frequency but ignores context or rarity across documents.

B. TF-IDF Vectorizer (Term Frequency-Inverse Document Frequency) [8]

- Similar to Count Vectorizer, but adjusts word counts based on their frequency across all documents.
- Useful for down-weighting common words and highlighting unique terms.

C. Word2Vec [9]

- Google's pre-trained model, which has been trained on a large Google News dataset and contains word vectors for 3 million words and phrases.
- Learns word embeddings from text corpus, reflecting word meanings based on surrounding words.

The selection among *Count Vectorizer*, *TF-IDF Vectorizer*, and *Word2Vec* depends on the specific model used. Therefore, we conducted a grid search for each model to identify the most suitable one, and its optimal hyperparameters. Interestingly, the only hyperparameter to consider turned out to be the n-gram range. This term refers to the number of consecutive words analyzed together. For instance, an n-gram range of `(1, 3)` implies considering individual words, pairs of words, and groups of three words. It's important to note that the size of the resulting vectorized feature matrix and the time taken for fitting the model increase exponentially with the n-gram range.
Regarding *GRU* and *BERT* models, each come with its respective tokenizer. Since they are specifically tailored for their neural networks, we did not need to manage the vectorization process for these models. Our focus was solely on preprocessing the data.

V. Models

We investigated five classification methods: *SGD*, *Logistic Regression*, *GRU*, *BERT*, and *RoBERTa*.
The cleaned training dataset is balanced, with 50.2% negative and 49.8% positive samples. Consequently, a random model would achieve an accuracy of `50%`. The subsequent subsection will detail each model and its performance on the validation set. Table I presents the results of each model on the test data following a submission to AIcrowd, along with a comparative analysis of their performances.

A. SGD

The *SGD (Stochastic Gradient Descent)* model iteratively updates its parameters by considering only a subset of the data in each iteration. This approach is faster and more scalable for large datasets, making it an ideal first choice for trial. We conducted a grid search across various vectorization techniques and their hyperparameters to identify the most effective combination for the model. We discovered that the *Count Vectorizer* with an n-gram range of `(1, 3)` was the most suitable.
After selecting the optimal vectorizer, we determined the ideal decision boundary threshold for the *SGD* model to classify a sample as positive or negative. The best threshold was found to be approximately `0.12`, which aligns statistically with the fact that our training data is balanced. Consequently, we maintained the regular decision boundary of the *SGD* model, which is `0`. This resulted in an accuracy of `81.68%` on the validation set.
Further, we implemented regularization methods to assess whether the model was overfitting the data. This was also done using grid search, focusing on the following hyperparameters:

- Loss : `Hinge`, `Log loss`, `Modified huber`
- Penality : `L1`, `L2`
- Alpha : `0.001`, `0.01`, `0.1`

We noted that the optimal regularization combination (`Modified Huber, L2, 0.001`) led to a slightly reduced validation accuracy of `81.04%`. Based on this observation, we concluded that the *SGD* model was not overfitting the data.

B. Logistic Regression

Similar to the *SGD* model, our grid search revealed that the most effective vectorizer for *Logistic Regression* was again the *Count Vectorizer* with an n-gram range of `(1, 3)`. The optimal decision boundary threshold remained the standard one. However, *Logistic Regression* introduces two critical learning hyperparameters: the learning rate γ and the maximum number of iterations `max_iter`. We conducted a grid search to identify the optimal values for these parameters and found that a γ `= 1` and `max_iter = 500` delivered a validation accuracy of `81.55%`.

In terms of regularization, we initially planned a grid search focusing on three parameters:

- `penalty`
- `ratio`
- `solver`

However, we found that the *Logistic Regression* model does not support the combination of these particular parameters. Additionally, we encountered solver incompatibilities, which resulted in issues with the gradient descent not converging. In light of these challenges, we shifted our focus exclusively to testing the `L2` regularization, using the model's default solver `lbfgs`. Ultimately, this approach of regularization resulted in the same validation accuracy of `81.55%`. Based on this outcome, we inferred that the model was not overfitting the data.

C. GRU

GRU (Gated recurrent units) is a variant of Recurrent Neural Networks (RNNs) specifically engineered for processing sequential data. It excels in capturing temporal dependencies within sequences, which makes it particularly apt for NLP tasks like sentiment analysis. Its proficiency in retaining information from earlier segments of text enables it to comprehend context and the progression of sentiment within a sentence or tweet.

Typically, the maximum length of a string entry is best set to a power of 2. However, in our case, we are analyzing tweets, with the longest tweet being around 273 characters. Therefore, we have chosen to cap `max_length` at `256` for our analysis. This decision is based on the observation that a cap of `256` yields a `1.0%` higher accuracy compared to a cap of `128`. As for the word embedding size, it is fixed at `100`, as we are utilizing weights from a pre-trained *GloVe* model.

This approach resulted in a validation accuracy of `85.8%`.

D. BERT

BERT (Bidirectional Encoder Representations from Transformers) utilizes the transformer architecture to process words in relation to all other words in a sentence, rather than sequentially. It is pre-trained on a vast corpus of text, enabling it to understand nuanced language patterns and contexts. *BERT* comes in two primary variants: *BERT Base* and *BERT Large*. These variants differ primarily in their size, complexity, and computational requirements. Here are the key differences :

- *BERT Base*: 12 transformer blocks, 768 hidden units, and 12 attention heads. This results in a total of about 110 million parameters
- *BERT Large*: 24 transformer blocks, 1024 hidden units, and 16 attention heads. This increases the total parameters to about 340 million.

We experimented with both variants, finding that `3 epochs` were sufficient for the loss to converge in each case. As for the `batch_size`, we settled on `32`, which is the maximum capacity our GPU can handle.
This approach resulted in a validation accuracy of `89.4%` for the base variant, and `89.7%` for the large variant.

E. RoBERTa

Subsequently, we've tested an optimized version of *BERT* : *RoBERTa (Robustly Optimized BERT Pretraining Approach)*. Essentially, *RoBERTa* is trained on a substantially larger dataset and employs dynamic masking instead of a fixed step approach. Moreover, it uses a byte-level BPE as a tokenizer.
Employing a setup identical to *BERT*'s, *RoBERTa Base* attained a validation accuracy of `90.7%`, surpassing *BERT* by *1%*. However, as illustrated in Table I, this did not translate into improved accuracy for the *AIcrowd* submission.

VI. Results

Model	Encoding	Accuracy	Total Time
SGD	Count Vectorizer	83.6%	∼5 min[1]
Logistic Reg.	Count Vectorizer	85.6%	∼13 min[2]
GRU	Keras Tokenizer	86.5%	∼40 min[3]
BERT Base	WordPiece Tokenizer	89.6%	∼11h[4]
BERT Large	WordPiece Tokenizer	90.0%	∼18h[5]
RoBERTa Base	Byte-Pair Encoding (BPE)	89.9%	∼11h[6]

Table I: Comparative Performance analysis of Models and their *AIcrowd* Prediction Results

These results clearly indicate that the *BERT* model surpasses the others in accuracies. However, the *BERT Large* variant only offers a marginal increase in accuracy (`+0.4%`) compared to the *Base variant*, yet requires `1.6x` times more time. For the purposes of this challenge, we have decided to preserve and submit the optimal weights of the Large model. This approach enables the model to make predictions without the necessity of retraining it entirely.

Finally, we didn't have enough resources to train *RoBERTa Large*, but we might expect it to achieve extra decimal points in accuracy compared to the base version.

[1]CPU: Quad-Core Intel Core i7, 1.2GHz, 16GB. Vectorization : 3 min. Training : 2 min
[2]CPU: Quad-Core Intel Core i7, 1.2GHz, 16GB. Vectorization : 3 min. Training : 10 min
[3]GPU: A100, 16Gb. Training : 4 min * 10 epochs
[4]GPU: A100, 40Gb. Tokenizing : 20 min. Training : 3h30 * 3 epochs. Validating : 8 min * 3 epochs
[5]GPU: A100, 40Gb. Tokenizing : 20 min. Training : 5h30 * 3 epochs. Validating : 20 min * 3 epochs
[6]GPU: A100, 40Gb. Tokenizing : 4 min. Training : 3h30 * 3 epochs. Validating : 8 min * 3 epochs

VII. Ethical Risks

Sentiment analysis, a subset of NLP, has shown remarkable benefits in various domains, from customer service to market analysis. By analyzing the sentiment behind texts, businesses can gauge public opinion, improve products, and tailor services to specific needs. However, with these benefits come significant ethical risks :

1) **Censorship in Journalism and Public Discourse:**
Journalists, social media users, and the general public are the groups most affected when sentiment analysis is used to filter and control public posts. This approach can lead to problems like limiting free speech. The risk here is significant because it threatens core values like democracy and freedom of speech. The chance of this happening can change depending on laws and how ethical the platforms are.
2) **Manipulation of Public Opinion:**
Voters and the general public are most affected when sentiment analysis is used to recommend content to users. This practice can sway public opinion, especially in important areas like politics, leading to unfair decisions and harming democratic processes. The risk is serious because it affects how individuals and groups make choices. The chance of this happening is getting higher as language processing tools become more advanced and are used more on social platforms. To check this risk, it helps to look at whether a wide range of content is being shown to users.
3) **Risks of Misinformation and Misinterpretation in NLP-Generated Texts:**
Businesses, researchers, and the general public, who use NLP services, are impacted when these services are relied upon to create and analyze texts. This can cause problems like spreading wrong information and misunderstandings, especially if these systems aren't properly fine-tuned or don't fully grasp the context and subtle meanings. The seriousness of this issue can range from small mix-ups to spreading major false information. The likelihood of these problems is quite high because automated systems for text writing and analysis are used more and more. To check how big this risk is, it's good to regularly test how accurate these systems are and compare them to human standards.

References

[1] "Stochastic gradient descent (sgd) documentation in scikit-learn." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

[2] "Logistic regression documentation in scikit-learn." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression

[3] "Gated recurrent unit (gru) documentation in tensorflow." [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/GRU

[4] "Bidirectional encoder representations from transformers (bert) documentation." [Online]. Available: https://arxiv.org/abs/1810.04805

[5] "Robustly optimized bert pretraining approach (roberta) - documentation." [Online]. Available: https://arxiv.org/abs/1907.11692

[6] W. contributors, "List of emoticons," *Wikipedia, The Free Encyclopedia*, 2023.

[7] "Count vectorizer documentation in scikit-learn." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

[8] "Tf-idf vectorizer in scikit-learn." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[9] ka-ka shi, "Googlenews-vectors-negative300 (word2vec)," 2020. [Online]. Available: https://www.kaggle.com/datasets/adarshsng/googlenewsvectors/

[10] "Library symspellpy." [Online]. Available: https://github.com/mammothb/symspellpy

[11] "Vadersentiment." [Online]. Available: https://ojs.aaai.org/index.php/ICWSM/article/view/14550

[12] "Keras." [Online]. Available: https://keras.io/

[13] "Tensorflow." [Online]. Available: https://www.tensorflow.org/

[14] "Glove embedding paper." [Online]. Available: https://nlp.stanford.edu/pubs/glove.pdf

[15] "Nltk library." [Online]. Available: https://www.nltk.org/