Social Media Emotion Classification

Jialong Li, Yuan Zhang, QiXi Huang, Daniel Wang, Wei Wang

1. Research Problems

Firstly, Natural Language Processing, is a field in computer science and AI focused on enabling computers to understand human language. It encompasses techniques that allow computers to interpret and analyze human languages, supporting applications like translation, sentiment analysis, for example, emotion classification, the goal in the project.

Emotion classification in social media involves analyzing user's posts into basic emotions such as happiness, sadness or anger, etc. This approach helps create personalized experience and it can even support mental health by identifying potential issues at an early stage. By understanding the emotional context of posts, social platforms can collect content that aligns with the user's feelings, which can not only know what kind of post contains the most of aggressive emotion but also provide targeted support so that it maintains a healthier online community.

The team is using a dataset of 20,000 English Twitter messages labeled with six emotions (anger, fear, joy, love, sadness, surprise) for sentiment analysis. Models include Logistic Regression, Random Forest Classifier, and BERT (Bidirectional Encoder Representations from Transformers), which is adept at understanding language nuances, making it ideal for processing with varied-length Twitter texts, each associated with with an emotion represented by integers 0 to 5. Additionally, few-shot learning techniques enable the team’s model to learn from a small number of examples per emotion, which might further improve the system's ability to classify emotions in Twitter messages.

2. Dataset Introduction

2.1 General info

The dataset is a collection of English Twitter messages (about 20k records), each labeled with one of six basic emotions: anger, fear, joy, love, sadness, and surprise. The team intends to use this dataset for sentiment analysis or emotion recognition tasks.

Each entry in the dataset consists of two fields:
Text: This field contains the raw text of a Twitter message. The text is likely to be informal, with the use of internet slang, abbreviations, and possibly emojis, given the nature of Twitter as a social media platform. The messages might vary in length, with most tweets being on the shorter side, but with some extending to longer lengths.

Label: The label field corresponds to the emotion associated with the text of the Twitter message. Note that each text can belong to only one kind of emotion. The six emotions are encoded as integers ranging from 0 to 5, where each number corresponds to a specific emotion (sadness: 0; joy: 1; love: 2; anger: 3; fear: 4; surprise: 5).

2.2 EDA

Emotions are not evenly spread out across the datasets. Around 70% of the recorded texts are associated with emotions such as "joy," "sadness," and "anger," while emotions like "surprise" make up only 4% of data. In other words, the team has an imbalanced dataset to work with. In this case, it would be easy to achieve high accuracy if one class is very common (simply labeling everything as that class).

In addition to accuracy, the team has decided to use other evaluation metrics to gain a more holistic view of the team's models in the later stages. Precision and Recall are particularly useful when there are more than two labels. Macro-average (average precision/recall of all classes) is useful if performance on all classes is equally important. Micro-average (average precision/recall of all items) is useful if performance on all items is equally important. The F1 score can also be a good metric.

Furthermore, The length of the text exhibits a right-skewed distribution, suggesting that while most of the Twitter messages are relatively short, there are a few messages that are much longer.

2.3 Encoders

A. CountVectorizer & N-gram

In order to build machine learning/NLP models that effectively predict emotional scores, the team needs a way to preprocess and extract numerical data (a matrix) from the text data. CountVectorizer plays a crucial role in satisfying the aforementioned task as it performs vectorization, dimensionality reduction, and preprocessing all at once.

CountVectorizer converts text data into numerical vectors, where each feature (column) represents a unique word or phrase, and the values in the vector indicate the frequency of occurrence of each feature in the text. This process is known as vectorization or feature extraction, essentially enabling machine learning models to work with text data effectively. Each text is now a vector of ones and zeros.

CountVectorizer also allows for dimensionality reduction by limiting the vocabulary size through parameters such as min_df and max_df, which specify the minimum and maximum document frequency thresholds for word inclusion. The team decided to set min_df to 0.01, which tells CountVectorizer to ignore terms that appear in less than 1% of the documents/texts. Additionally, CountVectorizer provides options for preprocessing text data, such as removing stop words (uninformative words: 'and', 'him', 'a', etc.), contributing to a more efficient feature space.

Given a text like 'I didn't feel humiliated and sad,' the extracted vectors will have 1 in the columns 'didn't,' 'feel,' 'humiliated,' and 'sad,' and 0 in other columns.

Another important concept in vectorization is n-gram. An n-gram is a contiguous sequence of n items (words in this case) from a given sample of text. N-grams consider single words as well as sequences of up to three consecutive words as features during the vectorization process. This helps capture more contextual information from the text than single words alone (unigrams). For example, phrases like 'machine learning' or 'natural language processing' may convey specific meanings that are not captured when considering individual words separately.

In the context of Twitter messages, there's a clear distinction between unigrams and bigrams. For example, the bigram 'damned hopeful,' which conveys a positive connotation overall, carries a completely different contextual message than separate words like 'damned' and 'hopeful.' On the other hand, the bigram 'greedy wrong,' which conveys a negative connotation overall, still carries a much more abundant connotation than just 'greedy' and 'wrong'.

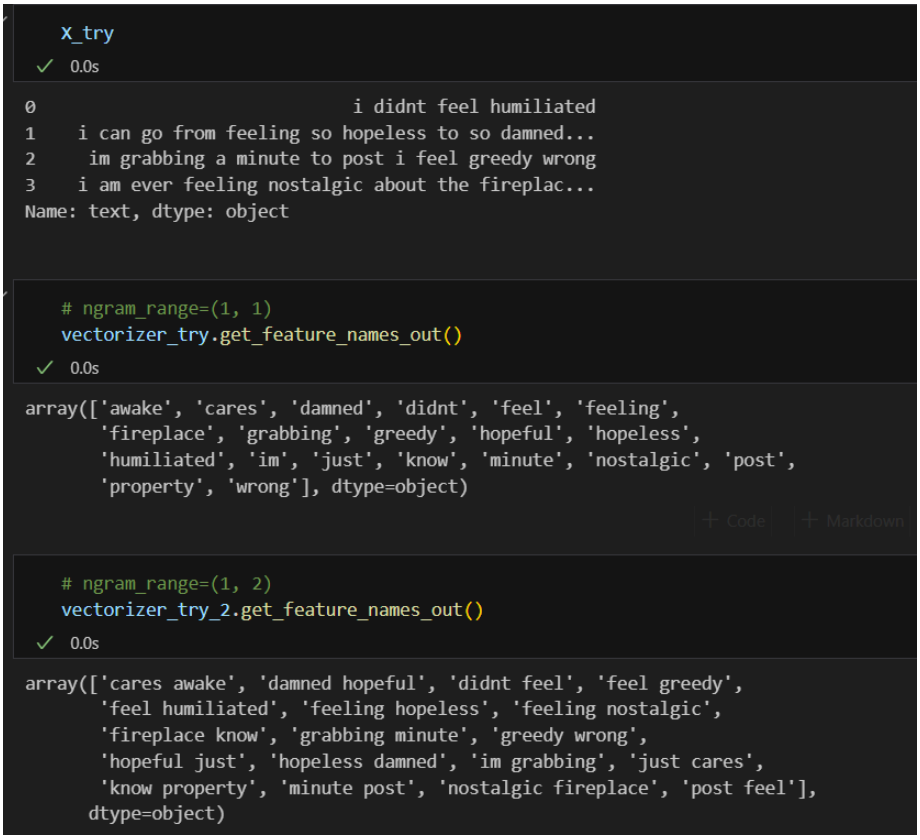
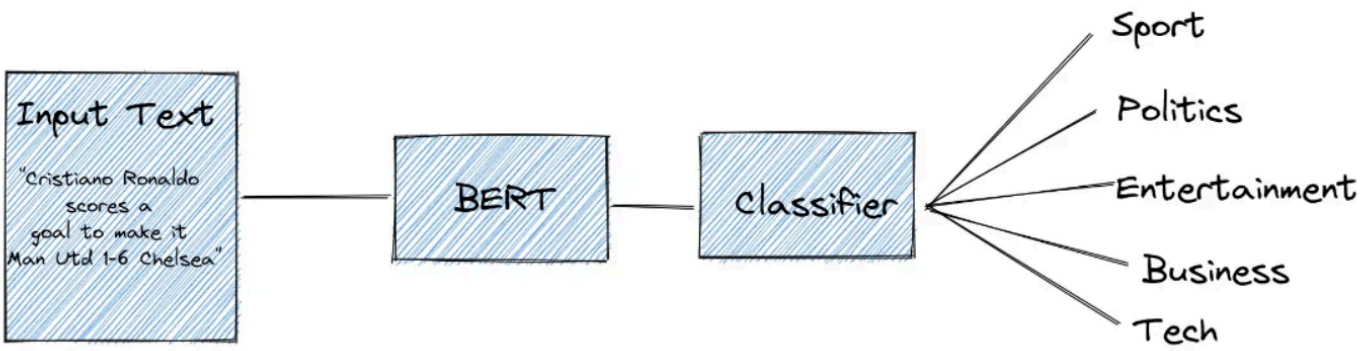


Figure 1. Unigram vs Bigram output comparison

The team utilizes CountVectorizer with appropriate n-gram values to convert the text into a matrix, enabling model-building in later stages.

B. BERT tokenizer

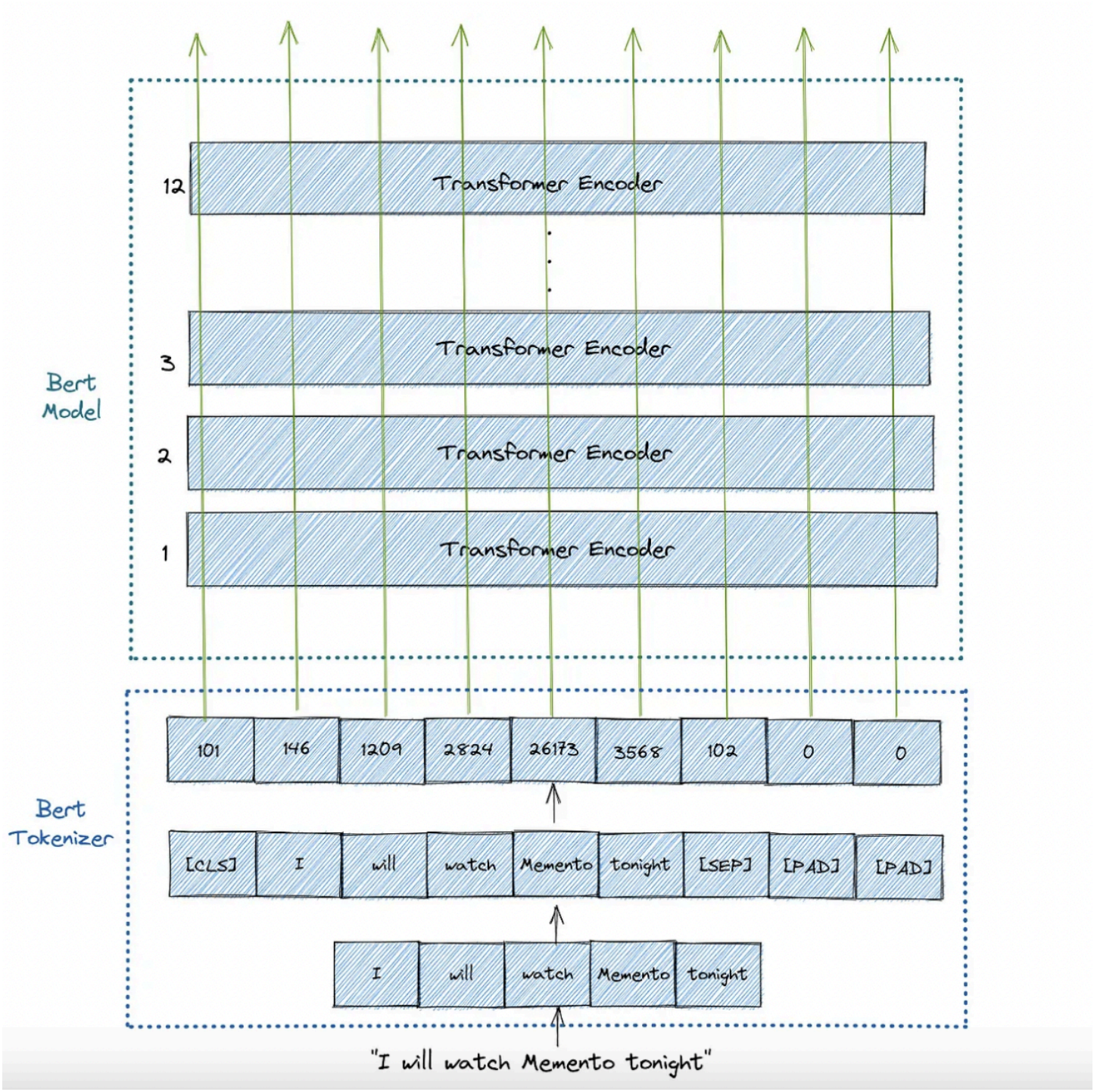


Originally from [1] BERT for classification tasks

The team decided to use DistilBERT as the final model, a simplified version of the original BERT model, for classifying emotions in text. Think of DistilBERT as BERT's smaller sibling, so it means it's designed to do much of the same work but in a way that's quicker and uses fewer resources.

DistillBERT has been specially trained to retain most of the effectiveness of the original BERT model despite being more compact. In fact, it has 40% fewer parameters, which means it's less complex. This reduction in complexity doesn't compromise its ability to understand language: it still holds onto over 95% of BERT's language understanding capabilities as well as 60% faster.

By using DistilBERT, the team is able to process language, understand emotions, and make predictions more quickly and efficiently. This model is a smart choice for the team's project, where the team needs to accurately grasp the nuances of human emotions in text while also considering the practical aspects like speed and computational cost.



Originally from [1] How BERT works for a specific input

2.4 Class Balancing

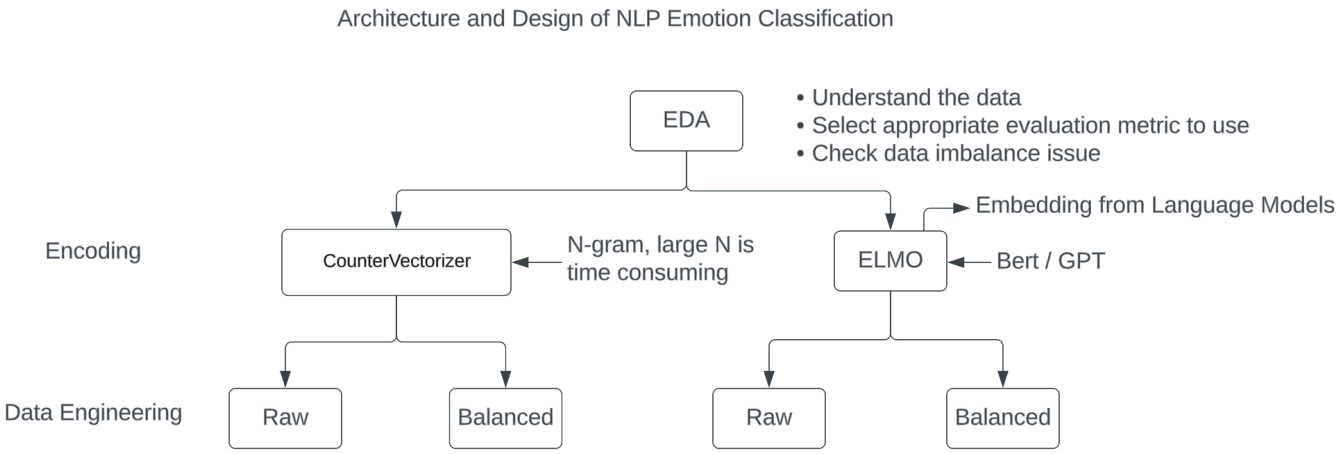
From the 2.2 EDA part, it is noticeable that there are minority classes in the dataset (e.g., class “surprise” only takes up 4% out of 6 classes. Whereas, class “joy” takes up 34% by itself). Therefore, class imbalance can

potentially cause problems in the following modeling part. One of the most important problems is the biasness. Having too many classes with positive messages, models are more likely to over-represent features that are in the positive classes. As a result, the predicting power of negative classes will drop significantly, causing downward bias toward classes like “fear” and “anger” (11% and 13% respectively). Furthermore, models will be over-confident about predicting negative classes being positive classes with high probabilities. On the other hand, joy and sadness are large classes that represent mild emotions while other smaller classes represents more intense emotion, but it’s dangerous to misclassify the intense emotion classes since those emotions are usually related with unusual and important behaviors that requires attention, there is higher misclassification costs for minority groups, having many instances of mild emotions would weaken the team’s ability to correctly classify strong emotion classes with higher costs.

To remedy the problem of class imbalance, the team applied both oversampling techniques including random oversampling and SMOTE, undersampling techniques such as random undersampling to upsample the minority classes or downsample the majority class to have the same number of observations between classes in the dataset. Through test set weight F1 score, the baseline model (logistic regression without any resampling techniques) has the same performance as the logistic regression model used with random oversampling, which both outperformed other techniques. Even though random oversampling has the same weighted test set F1 score as the baseline model, it is more friendly to generate more accurate classification towards minority classes, so the team decided to apply random oversampling technique for following steps.

Dataset	**Weighted F1 Score**
Dataset without Oversampling (Baseline)	0.91
Dataset with Random Oversampling	0.91
Dataset with SMOTE	0.90
Dataset with Random Undersampling	0.80

3. Modeling and Benchmark



Logistic Regression

In scikit-learn, a multinomial logistic model uses the one-vs-rest (OvR) scheme. In a trained logistic model for the team's dataset, there are 6 classes along with 6 sets of coefficients. To predict a class, the model will select the class with the highest probabilities among the 6 classes. Each class will be compared with the probability of belonging to the rest of classes.

Based on the coefficients and CounterVector, it is possible to check the most important words that the model is used to predict the corresponding class. For example, the top 3 most important words with the most positive coefficients for predicting the class "sadness" are "disturbed", "heartbroken", and "doomed". The top 3 most important words for all classes are summarized in the table below.

Class	Top 1	Top 2	Top 3
sadness	disturbed	heartbroken	doomed
joy	superior	sociable	divine
love	sympathetic	loyal	longing
anger	impatient	fucked	cranky
fear	terrified	shaken	paranoid
surprise	amazed	dazed	curious

As the table shows, most of the top words are closely matched with the corresponding emotion class. Therefore, the logistic model has the advantage of interpretability to see which of the words are helpful for predicting the

emotion of the message. Those coefficients are easily understandable by the general public as well. In conclusion, the logistic model is easy to fit and explain. It is even possible to evaluate the model performance for each class based on the word's semantic meaning because of its transparency.

Random Forest Classifier

Random Forest's ensemble learning approach, which builds multiple decision trees and aggregates their predictions, is suited for dealing with the complex and potentially non-linear relationships present in textual data. More specifically, each tree in the forest works with a random subset of feature, making the model robust against overfitting and be able to capturing a wide variety of pattern in the data so that this characteristic is particularly beneficial for text classification, where the importance of and relevance of specific words or phrase can vary significantly across documents.

The optimal parameters after GridSearch on cross-validation identified were a maximum depth of "None" and minimum sample split of 2 along with 50 estimators. These settings reflect a model that is allowed to grow trees to their full depth, ensuring that the model captures as much information as possible from the dataset. The high accuracy and F1 scores are indicative of the model's balanced performance across different classes, making it a reliable choice for text classification challenges.

Feature	**Importance**
curious	0.016001
amazed	0.015885
impressed	0.014937
surprised	0.012138
funny	0.011345
overwhelmed	0.011345
weird	0.009212
shocked	0.008924
amazing	0.008875
strange	0.008464

Bert-base-uncased-emotion

In this section, the team utilizes a pre-trained model that has been trained using the same emotion dataset and specifically tailored for the team's sentiment analysis task, aiming to explore the upper limits of the model's performance. More specifically, the team split the dataset into training(60%), validation (20%) and test (20%) subsets to evaluate the performance of the team's model effectively, especially the one comprising 20,000 textual samples, each labeled with one of the six emotions.

Initially, the uncased BERT-based model is a pre-trained model on the English language utilizing a masked language modeling (MLM) objective. This involves taking a sentence, randomly masking a chosen percentage of the words, then running the entire masked sentence through the model, which must predict the masked words. This approach, distinct from RNNs and autoregressive models, allows the model to learn a bidirectional representation of the sentence. Additionally, "uncased" signifies that the model does not differentiate between lowercase and uppercase English letters.

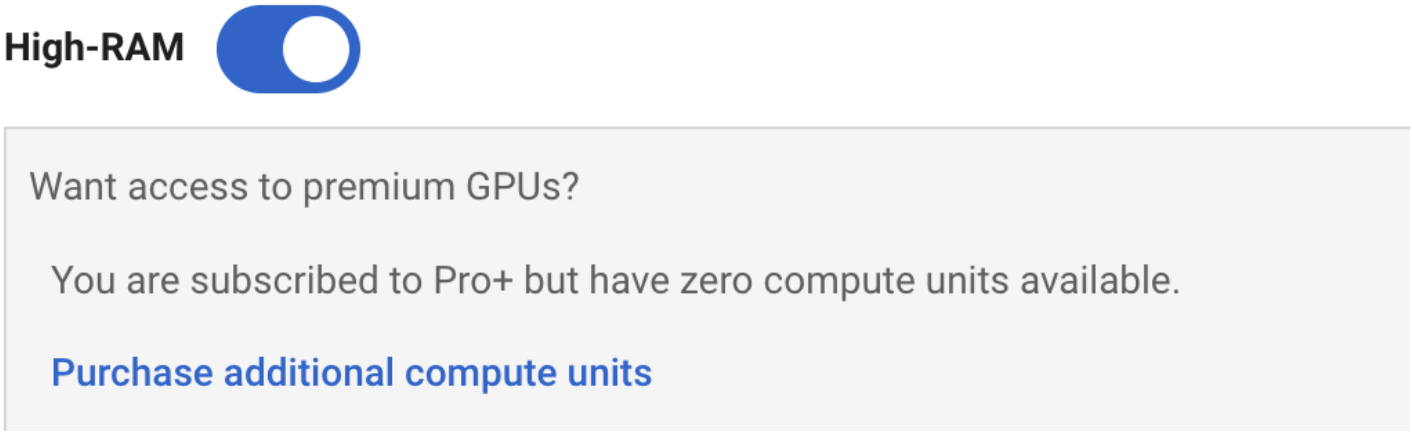
The targeted training process begins with the same emotion dataset. The dataset is then preprocessed, with texts being tokenized using AutoTokenizer for compatibility with BERT's input format. A model, AutoModelForSequenceClassification, is instantiated from the bert-base-uncased model and adapted for sentiment analysis by adding a classification layer. Custom metrics for accuracy and F1 score are defined to evaluate the model's performance and assist in hyperparameter tuning. The pre-trained model implements a hyperparameter learning rate of 5e-7, a batch size of 4, and num_train_epochs=20 as their optimal configuration.

The best fine-tuned BERT model from huggingface can give us a f1 score of 0.95.

Desired Outcome Table (F1 Score):

Model (Embeddings)	N-gram Raw	N-gram Balanced	ELMO Raw	ELMO Balanced
Logistic Regression	0.9083	0.9110	NA.	NA.
Random Forest	0.9010	0.8871	NA.	NA.
BERT-base	0.9060	0.8942	NA.	NA.
BERT-huggingface	NA.	NA.	0.95	NA.

EMLo is impossible to run because the team used up all compute units (It cost $60 my lord)



4. Conclusion

In conclusion, the integration of four datasets with distinct characteristics has equipped models to effectively classify emotions in social media content, in particular, commend the use of PyTorch's torch.cuda for its ability to parallelize computations across GPUs, significantly boosting processing efficiency for the team's emotion classification models. Moreover, the transition of raw data to PyTorch tensors has been pivotal in enhancing memory management, facilitating faster and more scalable model training.

However, the study acknowledges the limitation of its generalization capacity, primarily because it focuses solely on Twitter data, which may not be representative of language usage on other platforms or longer text formats. Future work should aim to broaden the model's applicability to diverse text sources and invest in refining the natural language understanding through advanced techniques like TF-IDF coupled with Word2Vec, as well as more extensive model fine-tuning.

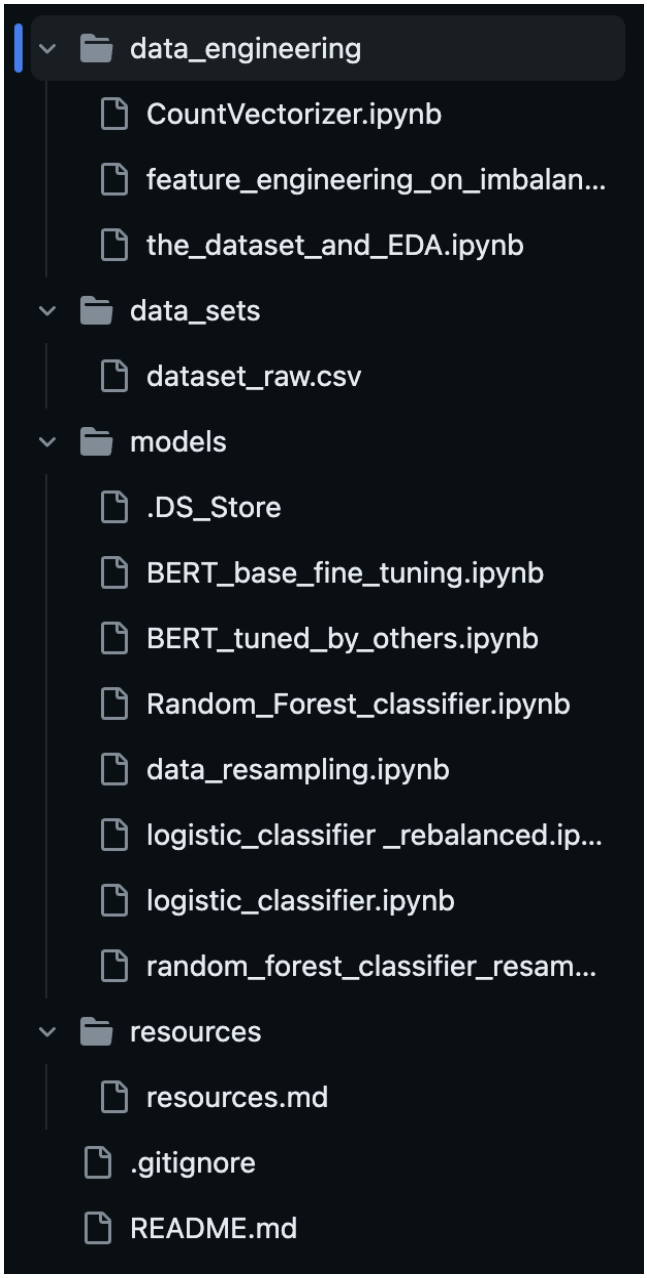
Reference:

[1] Winastwan, R. (2021, November 10). Text Classification with BERT in PyTorch. Towards Data Science. https://towardsdatascience.com/text-classification-with-bert-in-pytorch-887965e5820f

Code Appendix:

GitHub Link: https://github.com/Marklong7/NLP_emotion_classification/tree/main

There are many code documents created but the team has documented each step and the code we implemented in Github Repository, the structure is shown below:


data_engineering
CountVectorizer.ipynb
feature_engineering_on_imbalan...
the_dataset_and_EDA.ipynb
data_sets
dataset_raw.csv
models
.DS_Store
BERT_base_fine_tuning.ipynb
BERT_tuned_by_others.ipynb
Random_Forest_classifier.ipynb
data_resampling.ipynb
logistic_classifier _rebalanced.ip...
logistic_classifier.ipynb
random_forest_classifier_resam...
resources
resources.md
.gitignore
README.md