Emotion Classification

Vinit Patel

I. Abstract

In this project, I endeavored to leverage Natural Language Processing (NLP) models for the purpose of emotion classification. The main objective is to classify text as displaying one of six distinct emotions: joy, anger, sadness, love, fear and surprise. Towards this end, I selected a classic Logistic Regression model, a deep-learning Long Short-Term Memory (LSTM) network, and a lightweight version of a Bidirectional Encoder Representation from Transformers (BERT). The ultimate result was that the models showed accuracies of 0.87, 0.92, and 0.93% respectively. While these results are impressive, I believe they are limited by hardware capabilities and can be improved upon with further tuning and enhanced processing speed.

II. Introduction

In NLP, sentiment analysis is a foundational task; it aims to discern the sentiment or opinion conveyed in textual data. However, this approach offers a simplified view of human expression, categorizing text into positive, negative, or neutral sentiment. Emotion classification in my eyes is an evolution beyond sentiment analysis, delving deeper into the complexities that words can convey. By identifying and categorizing a broader spectrum of emotions expressed within text, emotion classification offers a more nuanced understanding, encompassing feelings of joy, sadness, anger and so on as aforementioned.

In this study, I explore the efficacy of three distinct models – logistic regression, Long Short-Term Memory (LSTM) networks, and Bidirectional Encoder Representations from Transformers (BERT) – in accurately identifying and categorizing emotions within text. The aim is to uncover the capabilities of each approach and identify which is best suited for further tuning towards this purpose.

III. Dataset

A. Data

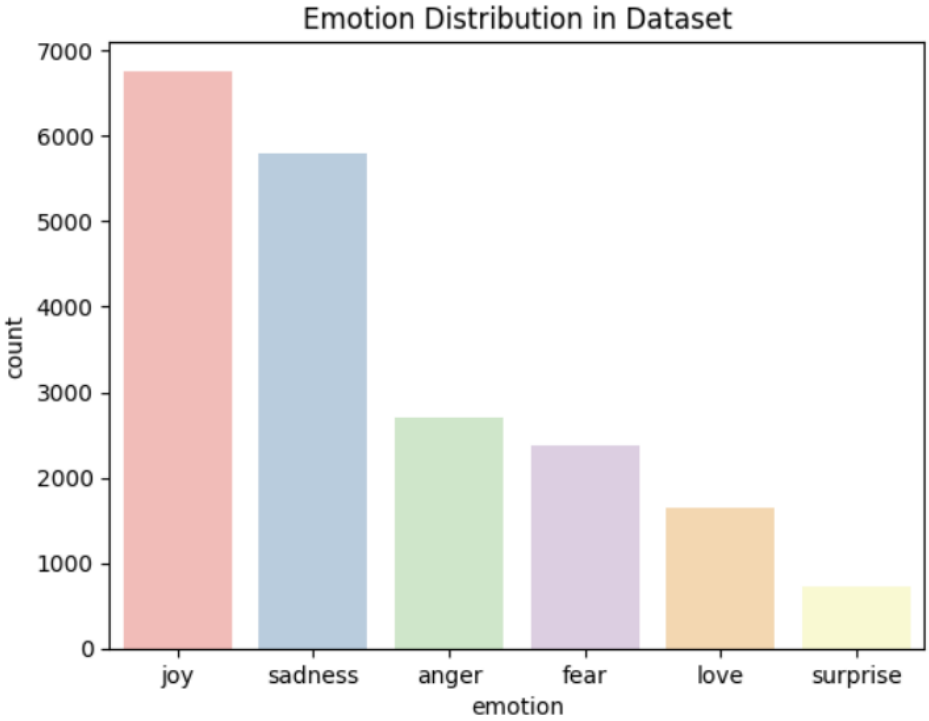
The data used for this project is provided by a data card from Kaggle that was worked on by the HuggingFace team. The data was designed specifically for emotion classification; combining the train, test, and val files gives us 20,000 sentences and their corresponding labels – strings such as “joy”, “anger”, “sadness”, and so forth that label which emotion the text corresponds to. Here are some concise examples:

im feeling insecure at the moment	fear
i am feeling so happy	joy
i feel gloomy and tired	sadness

(the text and labels are split by a “;” in the file)

B. Visualization

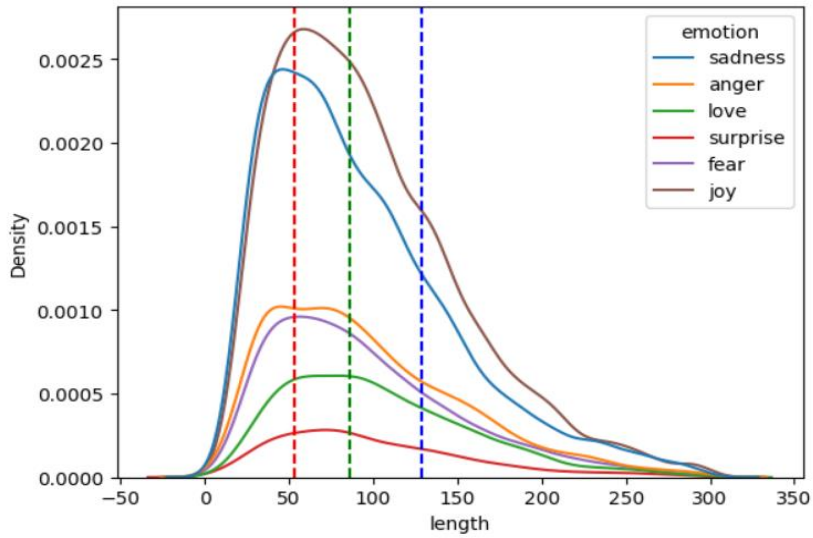
To give a better understanding of the dataset, below are visualizations from further analysis of the data. First is a simple plot of the amount of data we have for each unique emotion within our dataset.



Imbalanced datasets usually encourage underfitting/overfitting, but due to such a huge disparity in the amount of data we have for ‘joy’ vs. ‘surprise’, I chose to work with the imbalance in exchange for not losing a large amount of our data by under-sampling and avoiding overfitting via over-sampling.

Following this, we can start to look at the data itself, beginning with simple word clouds that correspond to each emotion:

Lastly, this kde plot shows the distribution of the length of the inputs colored by their respective emotions and includes lines colored to represent the 25th (red), 50th (green), and 75th (blue) percentile.



The difference in length between the texts is concerning, and along with the fact that 75% of the data falls at or below a length of 129, it became clear that some preprocessing was necessary to prevent a large reduction in efficiency due to padding each sequence to the max length of 300 within our data and to remove the clear abundance of generally insignificant words in our word clouds.

C. Preprocessing

The approach to preprocessing was to define **stopwords** using NLTK - the Natural Language Toolkit - and remove them, getting rid of words such as “the”, “is”, and other tokens that would be generally insignificant in identifying an emotion.

Furthermore, the tokens were **lemmatized**, i.e. reduced to root/base words. For example, “running” would be reduced to “run”, making it far easier to recognize tokens that held significance in our text.

IV. Models

A. Logistic Regression

The first model implemented was logistic regression. Our data was split into training and testing data (80% and 20% respectively) using **train_test_split**. I used scikit-learn libraries for my implementation and used their **TF-IDF Vectorizer** to convert the text data into TF-IDF vectors.

TF-IDF (Term Frequency-Inverse Document Frequency) was chosen over one-hot encoding vectorization due to its ability to capture the importance of each word in the context of the dataset, considering a words frequency and its rarity across all documents and the fact that it assigned each text an index value for the corresponding emotion instead of an entire vector, making TF-IDF vectors are more compact than one-hot vectors.

This choice helped make up for logistic regressions lack of ability to learn patterns/sequences compared to the other models by having the TF-IDF vectors capture the importance of words, providing weights based on frequency as opposed to binary values based on their presence, which in turn helped the model in distinguishing emotions Ultimately, this led to the best implementation of the logistic regression model, and this configuration was therefore used for the final result.

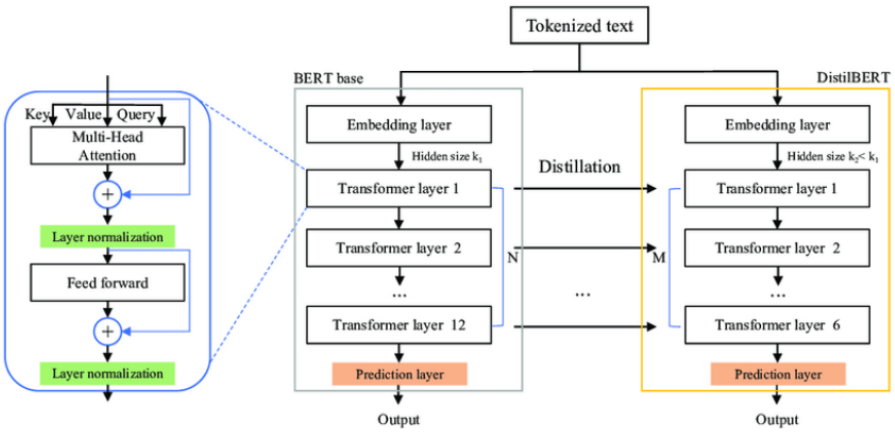
B. Long Short-Term Memory (RNN)

The second model implemented was Long Short-Term Memory (LSTM). I once again split the data into training and testing sets (80-20) and used TensorFlow and Keras for building and training the LSTM model. First, the data was tokenized using a Tokenizer provided by TensorFlow which was fitted on the processed text data, converting the text into sequences of integers. The data was then padded to ensure uniform length, with the maximum length being set equal to the longest sequence in the dataset.

Unlike logistic regression which used the TF-IDF vectors for feature representation, a Label Encoder from scikit-learn was used for the RNN, once again providing a more compact and thereby computationally efficient alternative to one-hot encoding. Afterwards, I tokenized the text and used an embedding layer to convert each word into dense vectors of equal size to the vocab of the tokenizer. This embedding layer was followed by a dropout layer to prevent overfitting and an LSTM layer with 100 units. Finally, a dense layer with a softmax activation function was added to output the probability distribution over different emotion classes.

The model was compiled using sparse categorical cross-entropy loss function and the AdamW optimizer. It was trained for 5 epochs with a batch size of 64, with early stopping to prevent overfitting on the data. Among different combinations of layers and hyperparameters, this configuration provided the best results.

C. DistilBERT (Transformer)



The third and final model implemented for this project was **DistilBERT**, a lightweight version of BERT that offers efficient processing of text data. To begin, I loaded a pre-trained DistilBERT tokenizer and model. The data was split into

training, validation, and testing sets, a 60-20-20 split respectively. A Label Encoder was used for this model similar to the RNN for the increased computational efficiency. Tokenization and encoding were facilitated by the DistilBERT tokenizer, converting input text into sequences compatible with the model, and padding was applied to ensure uniform sequence lengths.

Hyperparameter tuning was conducted on the validation set (using val_accuracy as the benchmark for optimal parameters) to optimize model performance, exploring combinations of learning rates, batch sizes, and epochs. The best performing hyperparameters that were selected for training the model were a learning rate of 5e-5, a batch size of 32, and a training length of 2 epochs.

During training, the AdamW optimizer was employed to update model parameters, while the training loop iterated over epochs to minimize cross-entropy loss between predicted and actual emotion labels. Upon completion, the model was evaluated on test data that it had not been exposed to.

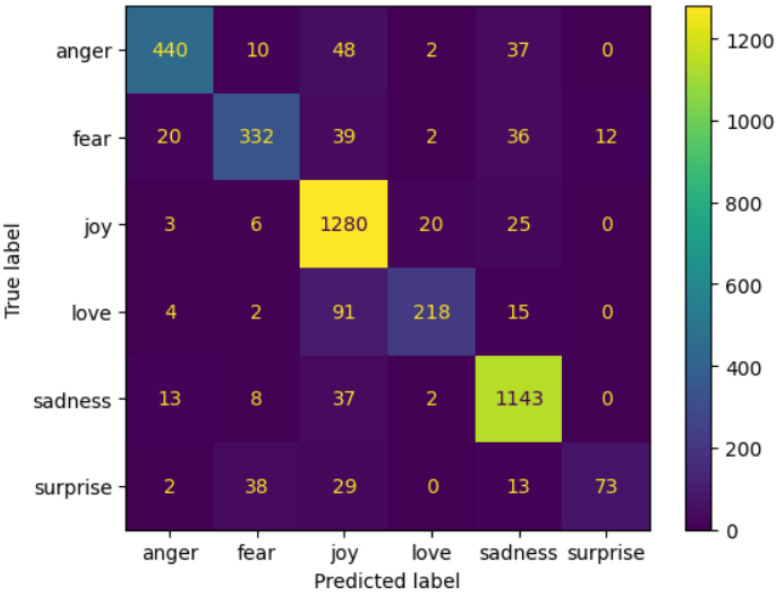
V. Results & Conclusions

Here are the classification reports and confusion matrices for each model:

Linear Regression

Classification Report:

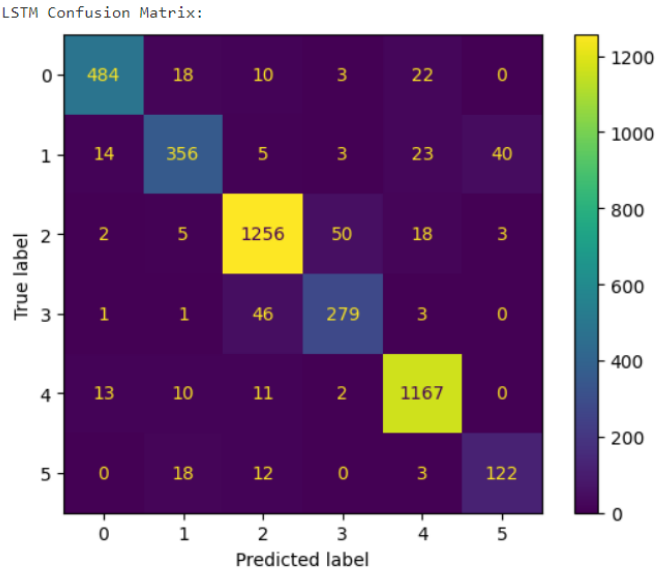
	precision	recall	f1-score
anger	0.91	0.82	0.86
fear	0.84	0.75	0.79
joy	0.84	0.96	0.90
love	0.89	0.66	0.76
sadness	0.90	0.95	0.92
surprise	0.86	0.47	0.61
accuracy			0.87
macro avg	0.87	0.77	0.81
weighted avg	0.87	0.87	0.87



LSTM

LSTM Classification Report:

	precision	recall	f1-score
0	0.94	0.90	0.92
1	0.87	0.81	0.84
2	0.94	0.94	0.94
3	0.83	0.85	0.84
4	0.94	0.97	0.96
5	0.74	0.79	0.76
accuracy			0.92
macro avg	0.88	0.88	0.88
weighted avg	0.92	0.92	0.92

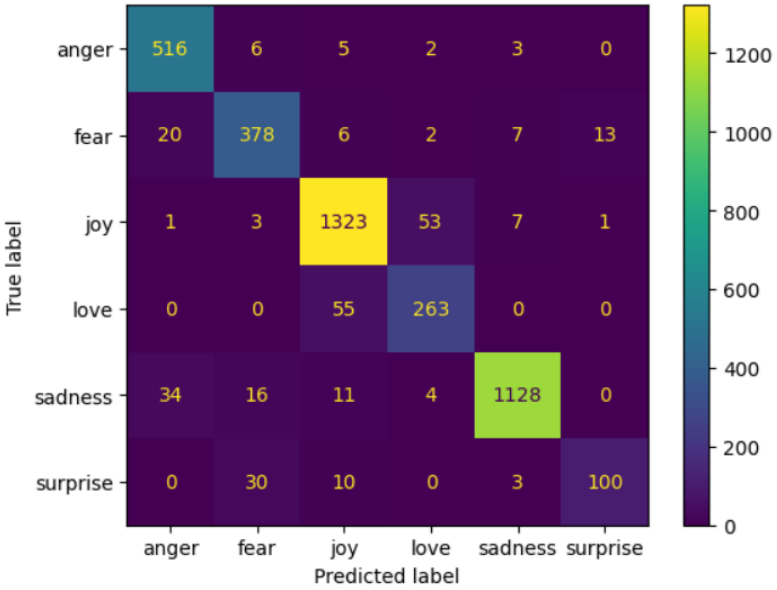


0 = anger | 1 = fear | 2 = joy | 3 = love | 4 = sadness | 5 = surprise

DistilBERT

Classification Report:

	precision	recall	f1-score
0	0.90	0.97	0.94
1	0.87	0.89	0.88
2	0.94	0.95	0.95
3	0.81	0.83	0.82
4	0.98	0.95	0.96
5	0.88	0.70	0.78
accuracy			0.93
macro avg	0.90	0.88	0.89
weighted avg	0.93	0.93	0.93



VI. Future Plan

As aforementioned, I believe there is still room to improve with these models, particularly the DistilBERT implementation. With enhanced processing capability and further tuning of hyperparameters (specifically the learning rate and batch size as the professor suggested), I believe the model could achieve increased accuracy.