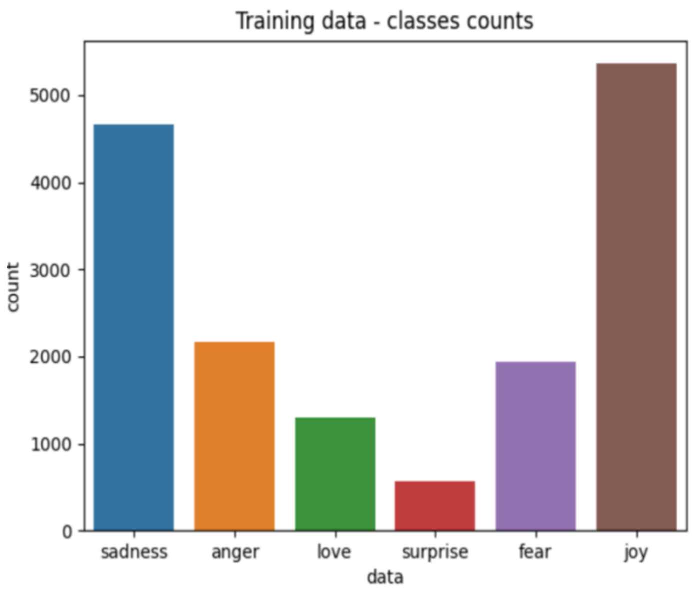
Introduction

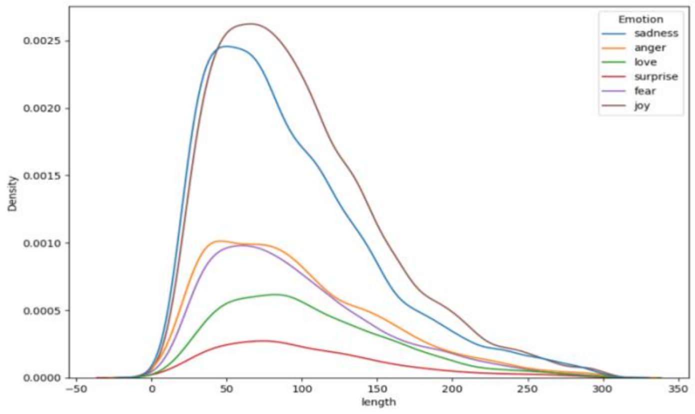
Emotions are an integral part of human life, and the ability to identify and express them effectively is crucial for maintaining mental health and well-being. In recent years, the growing availability of social media platforms has resulted in an explosion of data containing textual expressions of emotions. Multiclass emotion classification is the task of automatically identifying emotions expressed in text and assigning them to predefined categories. This project aims to explore the effectiveness of machine learning models for multiclass emotion classification on a textual dataset. The dataset contains labeled text samples belonging to six distinct emotions (sadness, fear, angry, joy, love, surprise), and the goal is to develop a model that accurately classifies these emotions. The project involves various stages such as data preprocessing, model selection, hyperparameter tunning and evaluation. The results of this project can have practical applications in various fields, including mental health, marketing, and social media analysis.

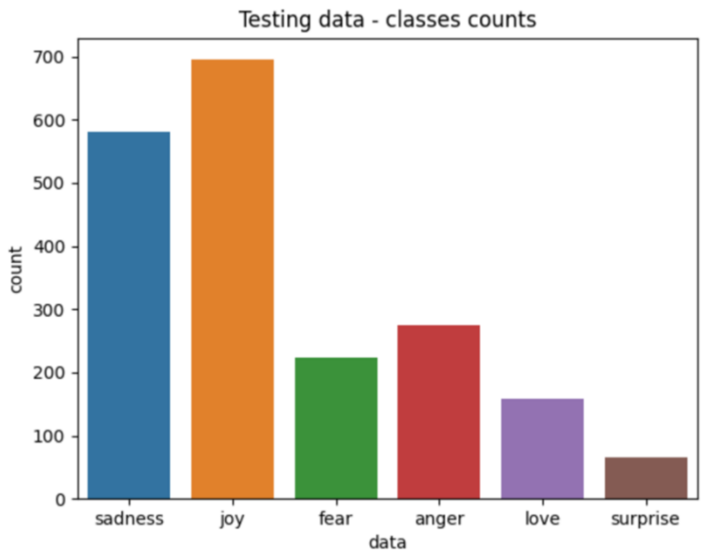
Work and results in detail

- **EDA and Data Pre-Processing**

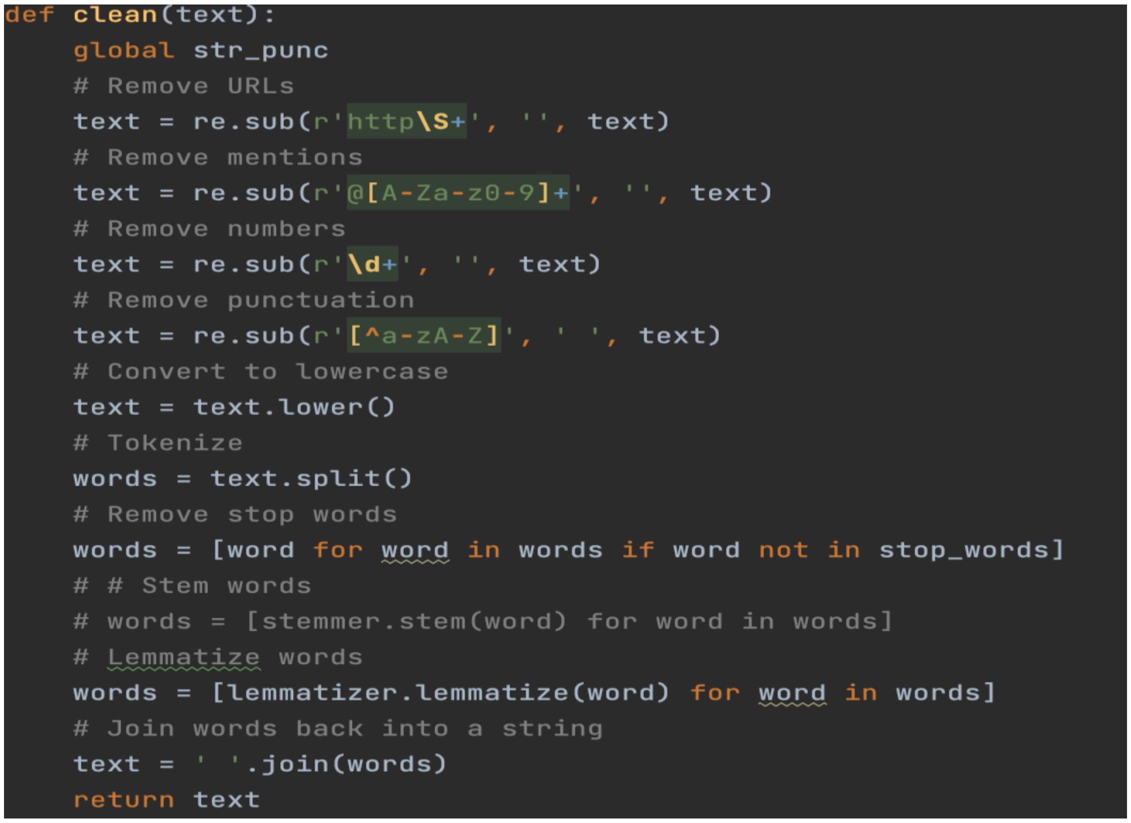
We generated some visualizations for the data exploration and analysis in the project. We used the seaborn library to plot the counts of each class in the training, testing, and validation datasets. This helps to identify if there is a class imbalance issue in the dataset. Then, a kernel density plot is generated using seaborn to visualize the distribution of the length of texts for each class in the training data. This plot can be used to identify if there are any patterns or differences in the length of texts for each class.







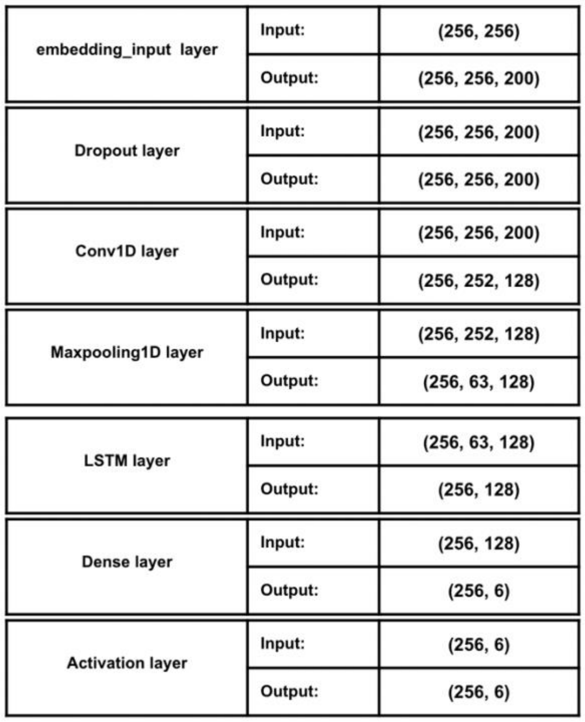
We defined a function for data pre-processing named clean. It takes a raw text input and performs a series of operations to remove unwanted elements and clean up the text. Specifically, the function removes URLs, mentions, numbers, and punctuation. It then converts the text to lowercase, tokenizes it into words, removes stop words, and lemmatizes the remaining words. Finally, the function joins the words back into a cleaned text string. These operations help to standardize the text data and remove any irrelevant or noisy elements, making it easier for the model to learn from the text and make accurate predictions.



```python
def clean(text):
    global str_punc
    # Remove URLs
    text = re.sub(r'http\S+', '', text)
    # Remove mentions
    text = re.sub(r'@[A-Za-z0-9]+', '', text)
    # Remove numbers
    text = re.sub(r'\d+', '', text)
    # Remove punctuation
    text = re.sub(r'[^a-zA-Z]', ' ', text)
    # Convert to lowercase
    text = text.lower()
    # Tokenize
    words = text.split()
    # Remove stop words
    words = [word for word in words if word not in stop_words]
    # # Stem words
    # words = [stemmer.stem(word) for word in words]
    # Lemmatize words
    words = [lemmatizer.lemmatize(word) for word in words]
    # Join words back into a string
    text = ' '.join(words)
    return text
```

- **Build LSTM+CNN model**

We used a combination of convolutional neural network (CNN) and recurrent neural network (LSTM) layers. In our CNN architecture for text classification, we used an embedding layer with an input shape of (256, 256) and an output shape of (256, 256, 200). This layer converts the input sequences of words into dense vectors of fixed size, which are then fed to the next layer. A dropout layer with an input shape of (256, 256, 200) and an output shape of (256, 256, 200) was added to reduce overfitting. The following layer is a Conv1D layer with an input shape of (256, 256, 200) and an output shape of (256, 252, 128). A MaxPooling1D layer was applied with an input shape of (256, 252, 128) and an output shape of (256, 63, 128), where the output length was divided by the pool size (4) resulting in 252 // 4 = 63. An LSTM layer with an input shape of (256, 63, 128) and an output shape of (256, 128) was used to capture the sequential dependencies in the text. The output was passed through a Dense layer with an input shape of (256, 128) and an output shape of (256, 6), assuming that there are 6 classes in the output. Finally, an activation layer was added with an input shape of (256, 6) and an output shape of (256, 6) to generate the predicted probabilities for each class. The SoftMax activation function is used to output probability scores for each class. The Adam optimizer is used to minimize the categorical cross-entropy loss function. The model is trained for 30 epochs with early stopping based on validation loss to prevent overfitting. Finally, the model is evaluated on the test set and the accuracy is reported.



embedding_input layer	Input:	(256, 256)
	Output:	(256, 256, 200)
Dropout layer	Input:	(256, 256, 200)
	Output:	(256, 256, 200)
Conv1D layer	Input:	(256, 256, 200)
	Output:	(256, 252, 128)
Maxpooling1D layer	Input:	(256, 252, 128)
	Output:	(256, 63, 128)
LSTM layer	Input:	(256, 63, 128)
	Output:	(256, 128)
Dense layer	Input:	(256, 128)
	Output:	(256, 6)
Activation layer	Input:	(256, 6)
	Output:	(256, 6)

- **Hyperparameter tuning**

We conducted experiments to optimize the learning rate and optimizer of our model. Specifically, we tested three different learning rates (0.0001, 0.0005, and 0.001) and two optimizers (Adam and SGD) on our model. The results showed that the model's accuracy on the test dataset was slightly better with a learning rate of 0.001 compared to the other learning rates. Additionally, we found that the model could be trained effectively over multiple epochs when the optimizer was set to Adam. These findings suggest that optimizing the learning rate and optimizer can lead to improved model performance for text classification tasks.

Summary

The project focuses on Tweet Text Emotion Recognition tasks. To classify the six emotions, we used the LSTM and Transform model individually and together to build three different models and compared performance of different models.

The pre-trained transformer model has the highest macro average f1 score, and the transform model with LSTM head was followed, which means that the transform model is suitable for this emotion classification task compared with the LSTM model.

Model interpretations were completed. First, we used the LIME to explain the prediction of the model for some specific samples. Second, we also used the SHAP to find out the top words impacting each specific class.

Calculate the percentage of the code that you found or copied from the internet
The total number of lines of the py. file is 150, and I modified 20 of lines and added another 15 lines of my own code. The percentage would be (430-20)/(430+15)= 92.13%

References

https://www.kaggle.com/datasets/praveengovi/emotions-dataset-for-nlp

https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f

https://discuss.huggingface.co/t/what-is-the-classification-head-doing-exactly/10138

https://discuss.huggingface.co/t/how-do-i-change-the-classification-head-of-a-model/4720

https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452 https://towardsdatascience.com/transformers-explained-visually-part-2-how-it-works-step-by-step-b49fa4a64f34

https://towardsdatascience.com/transformers-explained-visually-part-3-multi-head-attention-deep-dive-1c1ff1024853

https://stackoverflow.com/questions/65079318/attributeerror-str-object-has-no-attribute-dim-in-pytorch

https://github.com/huggingface/notebooks/blob/main/examples/text_classification.ipynb

https://towardsdatascience.com/explain-nlp-models-with-lime-shap-5c5a9f84d59b

https://shap.readthedocs.io/en/latest/example_notebooks/text_examples/sentiment_analysis/Emotion%20classification%20multiclass%20example.html

https://medium.com/nlplanet/two-minutes-nlp-explain-predictions-with-lime-aec46c7c25a2
.