Final Project Report

Tweet Emotion Recognition

NLP for Data Science

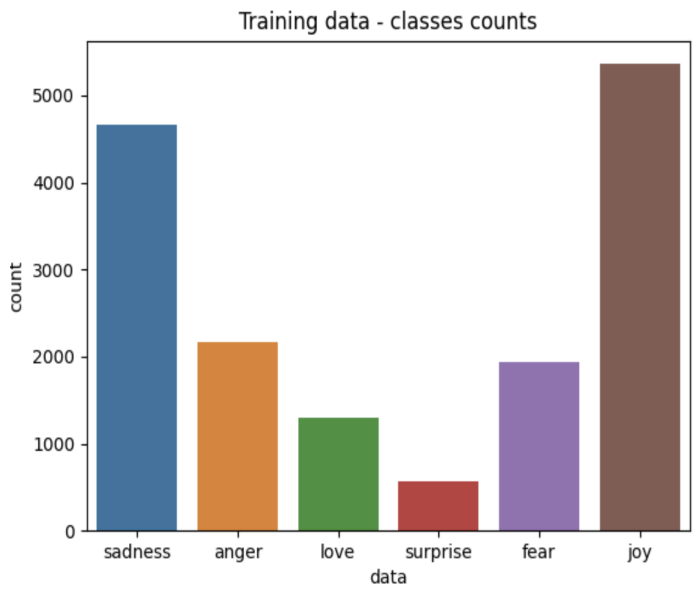
Luke Wu, Xiao Qi

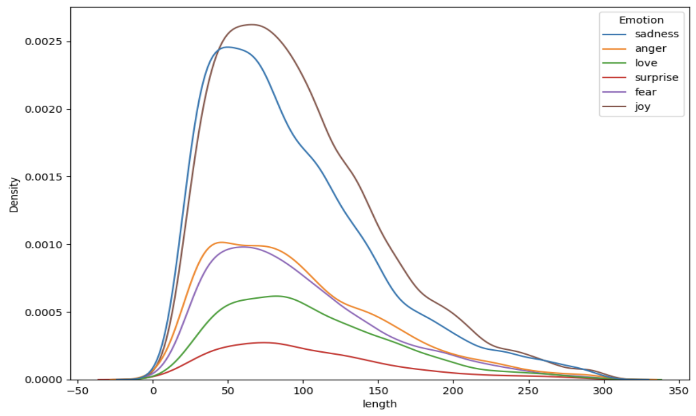
Introduction

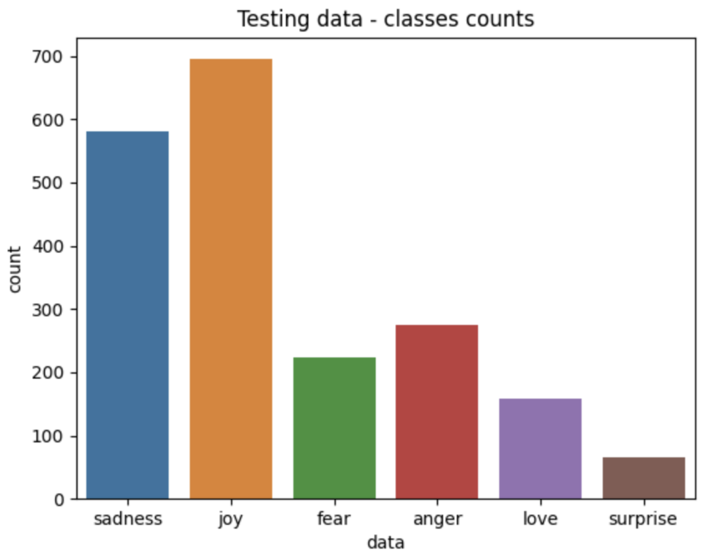
Emotions are an integral part of human life, and the ability to identify and express them effectively is crucial for maintaining mental health and well-being. In recent years, the growing availability of social media platforms has resulted in an explosion of data containing textual expressions of emotions. 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.

EDA and Data Pre-Processing

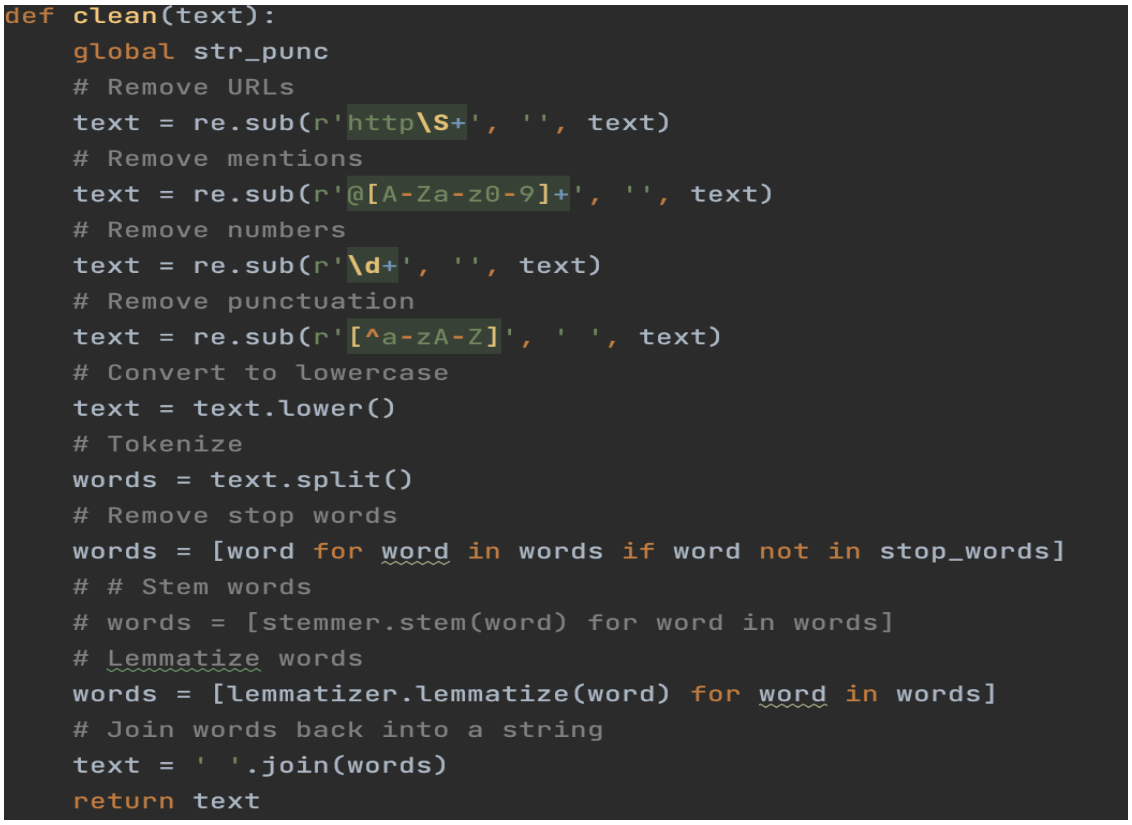
We generated some visualizations for the data exploration and analysis in the project. We used the seaborn library to plot the counts of each class in the training, testing, and validation datasets. This helps to identify if there is a class imbalance issue in the dataset. Then, a kernel density plot 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.


Training data - classes counts
count
0
1000
2000
3000
4000
5000
sadness
anger
love
surprise
fear
joy
data


Emotion
sadness
anger
love
surprise
fear
joy
Density
0.0000
0.0005
0.0010
0.0015
0.0020
0.0025
−50
0
50
100
150
200
250
300
350
length


Testing data - classes counts
count
0
100
200
300
400
500
600
700
sadness
joy
fear
anger
love
surprise
data

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
```

Model Built

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.

Transform

We also built two transformer models after the LSTM model. First, we used Bert for sequence classification directly. Here, DistilBERT model was used, which is a distilled form of the BERT model. it was reduced by 40% knowledge in pre-training but still retaining 97% of its language understanding abilities and being 60% faster.

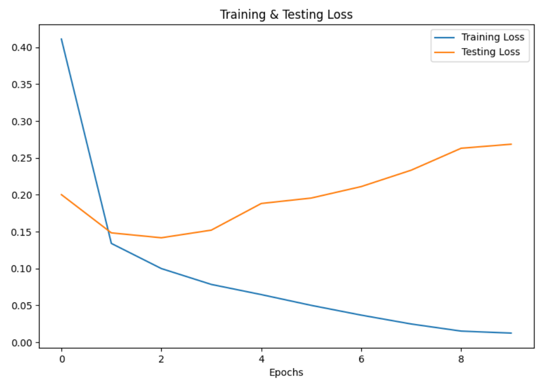
Because the model is already pre-trained and encapsulated, what we need to do is give the number of labels to the model. The code for the model is as follows:

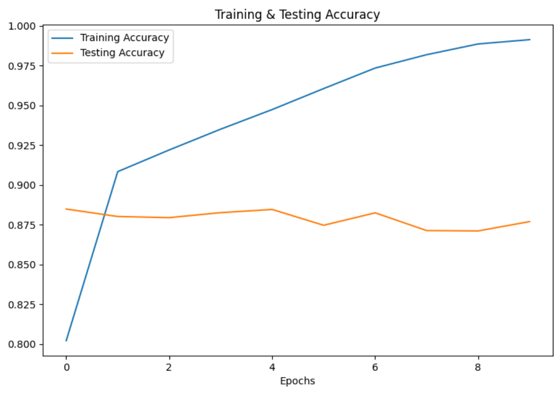
```
checkpoint = "distilbert-base-uncased"
model = DistilBertForSequenceClassification.from_pretrained(checkpoint, num_labels=6)
```

After model building, we train the first model. We set the parameters as follows to train the model:

parameters	value
epoch	10
optimizer	AdamW
Learning rate	5e-5

And the model results are as follows:





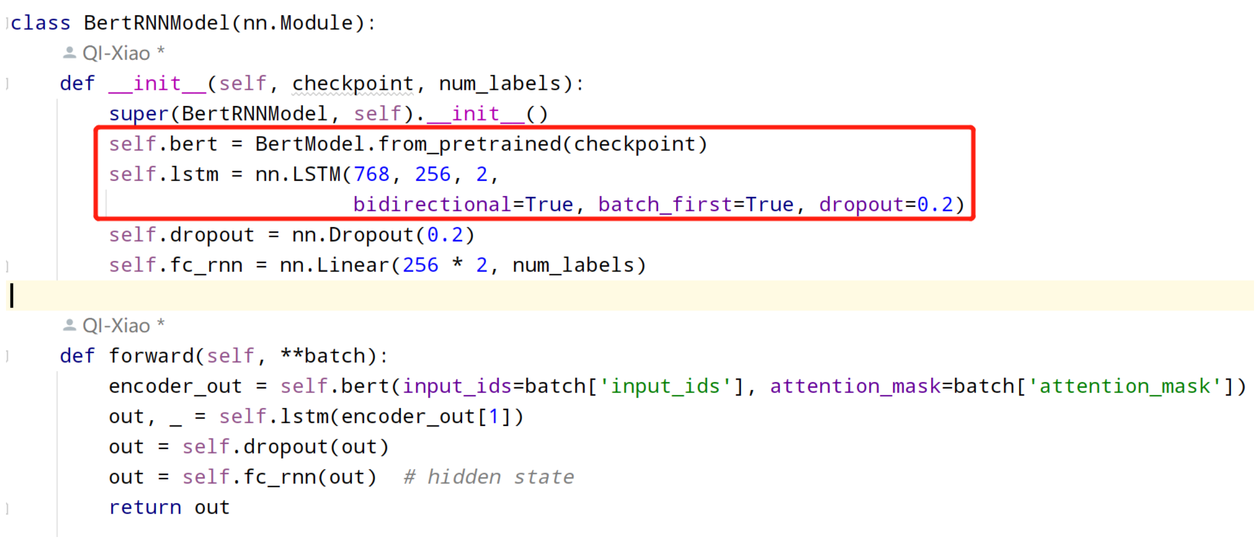
Here, we used macro average f1 score here. Because we are working with an imbalanced dataset where all classes are equally important, using the macro average would be a good choice as it treats all classes equally.

We can find from the pictures above that the Distilled Bert model can be trained in two loops. After that, an overfitting happens. It may be because that model is pre-trained and we didn't change anything in the model, which makes the model easy to train.

Transform with LSTM head

After the DistilBERT model, we change the head of the DistilBERT by a LSTM head to see if the performance changes.

The architecture of the model is as follows:

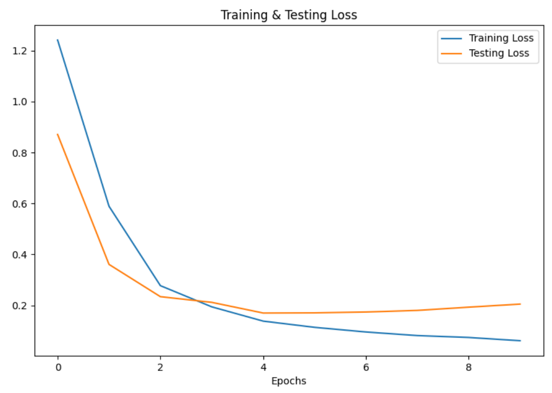


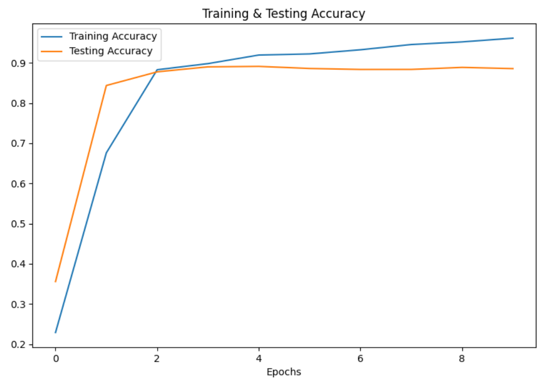
```
class BertRNNModel(nn.Module):
    QI-Xiao *
    def __init__(self, checkpoint, num_labels):
        super(BertRNNModel, self).__init__()
        self.bert = BertModel.from_pretrained(checkpoint)
        self.lstm = nn.LSTM(768, 256, 2,
                            bidirectional=True, batch_first=True, dropout=0.2)
        self.dropout = nn.Dropout(0.2)
        self.fc_rnn = nn.Linear(256 * 2, num_labels)

    QI-Xiao *
    def forward(self, **batch):
        encoder_out = self.bert(input_ids=batch['input_ids'], attention_mask=batch['attention_mask'])
        out, _ = self.lstm(encoder_out[1])
        out = self.dropout(out)
        out = self.fc_rnn(out)  # hidden state
        return out
```

The training parameters of the model are the same as the model before.

And the model results are as follows:





We can find that the model also has a high macro average f1 score. And after ten loops, the model didn't overfit.

Performance Comparison

Here, we compared three models based on their micro average f1 score and macro average f1 score individually.

	f1_score (micro)	f1_score (macro)

LSTM+CNN	0.912	0.867
Transform	0.940	0.915
Transform+LSTM	0.936	0.907

Compared with the transform model, The LSTM model has a lower f1 score. This is because that transform model has been pre-trained with a large amount of data. Meanwhile, the LSTMi was challenging to deal with long-range dependencies between words that were spread far apart in a long sentence.

Compared with two transform models, we can find that changing the head has a low influence on the model performance. Specifically, using the head of LSTM causes the model to have a little micro and macro f1 score. Meanwhile, it increased the training time. For the DistilBERT model, a loop of training takes only one minute and a half. However, the time of training a transform model with LSTM head took three minutes.

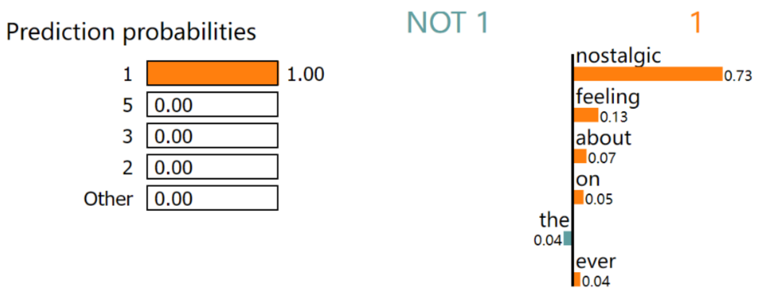
Model Interpretation

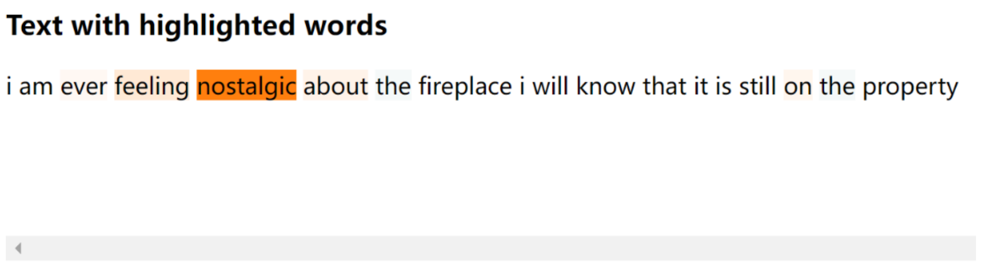
LIME

LIME, which stands for Local Interpretable Model-Agnostic Explanations, is a Python library that provides a way to explain the predictions made by any machine learning model. It was developed to help human beings understand and trust the decisions made by machine learning algorithms.

In our project, we used the explain_instance method in LIME to explain the prediction of the model for this specific sample.

We picked two samples here. One is from class love, and the other is from class sadness. We got the report from LIME for the first sentence as follows:




Text with highlighted words

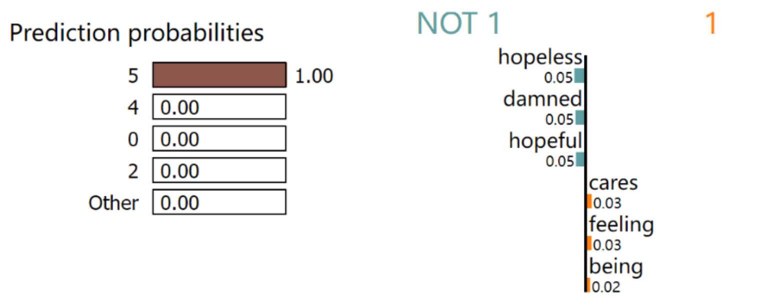
i am ever feeling nostalgic about the fireplace i will know that it is still on the property

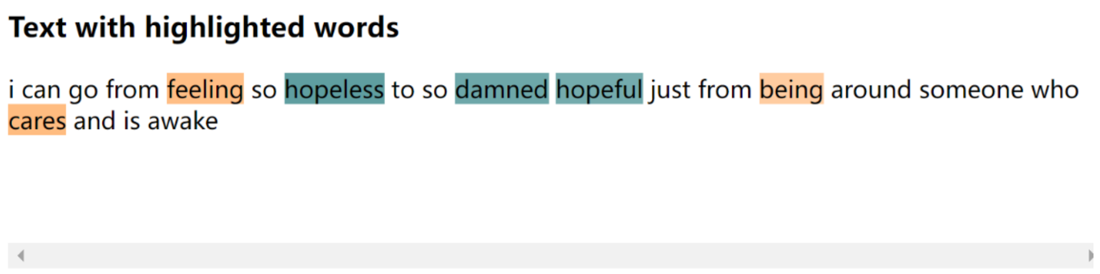
For this sentence, the left upper part is the probabilities predicted by the model for the sample: in this case, the probability of class 1 (love) is 1.00, which is correct.

The right upper part is about the coefficients of the learned sparse linear model fitted on the input perturbations. In this case, the word "nostalgic" has a coefficient of 0.73, which means that a large positive perturbation of its score to classify this sentence as class 1.

The bottom part is full sample text with highlighted words, which are the ones with higher coefficients from the learned sparse linear model.

Let's look at the other sentence:




Text with highlighted words

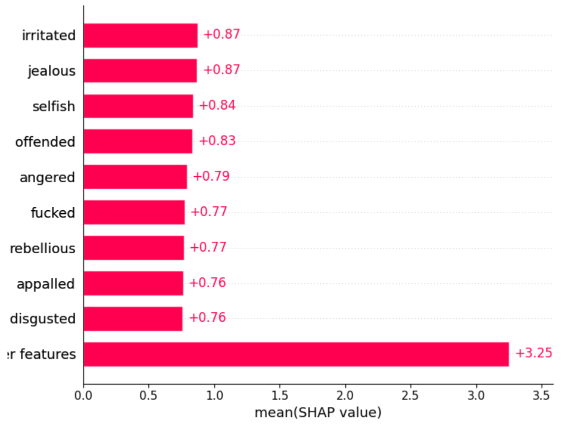
i can go from feeling so hopeless to so damned hopeful just from being around someone who cares and is awake

We can find that this sentence belongs to class 5, which is sadness. Obviously, this sentence didn't belong to class 1. So we can find that all the words in this class have low effect in being or not being class 1 for this sentence. Even though words "hopeless", "damned", and "hopeful" have a negative effect, while words "cares", "feeling" have a positive effect, they all have minimum values.

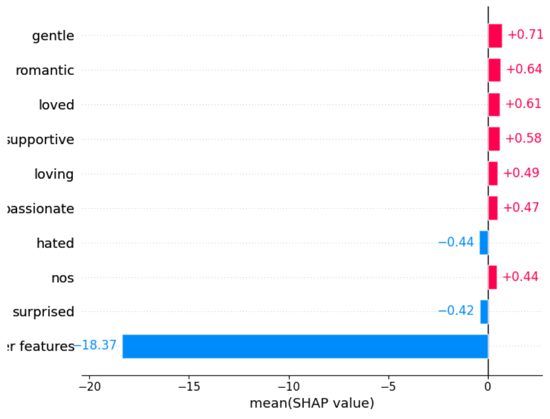
SHAP

SHAP (SHapley Additive exPlanations) is a unified measure of feature importance that allocates the contribution of each feature towards making a particular prediction in a fair and consistent manner. The SHAP library in Python implements a suite of methods to compute the SHAP values for machine learning models.

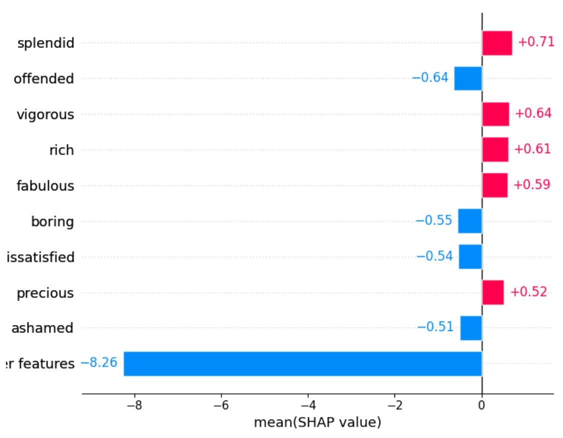
Here, in order to get a better summary, we used a larger portion of the dataset of 200 sentences from the training dataset. We put them into the SHAP to see the top words impacting each specific class.



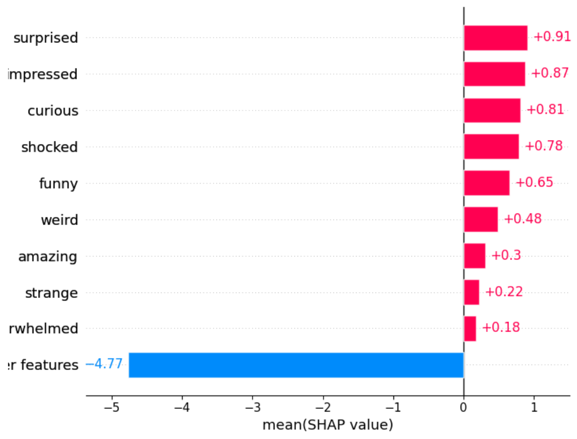
anger



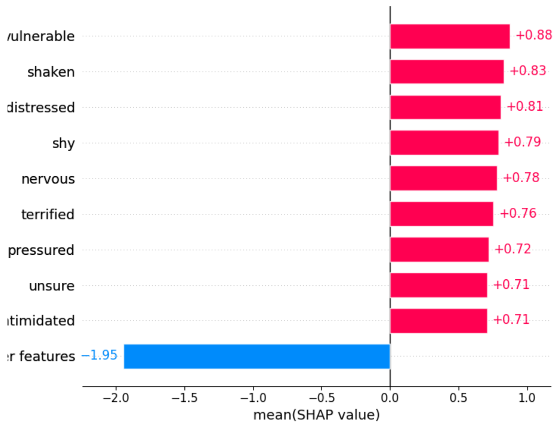
love



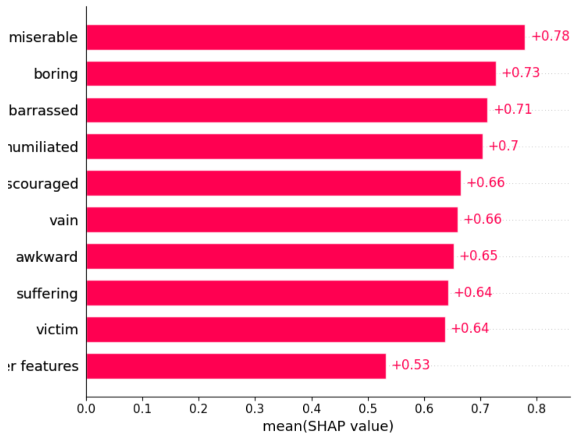
joy



surprise



fear



sadness

We made six plots for all classes. And from each image, we can find the most influential words in each class. For example, in “sadness” class, the top five influential words, in order, are “miserable”, “boring”, “barrassed”, “humiliated”, and “scourged”.

Summary

(1) The project focuses on Tweet Text Emotion Recognition tasks. In order 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.

(2) 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.

(3) 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.

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