Tweet Emotion Recognition

Luke Wu, Xiao Qi

May 3, 2023

Overview

Problem Statement

Data

Data Preprocessing

Model Selection

Model Comparison

Interpretation

Summary

Problem Statement

We want to classify the emotions from user's tweets into sadness, anger, fear, surprise, love and joy categories to gain insights into user's sentiment.

Data

Content:

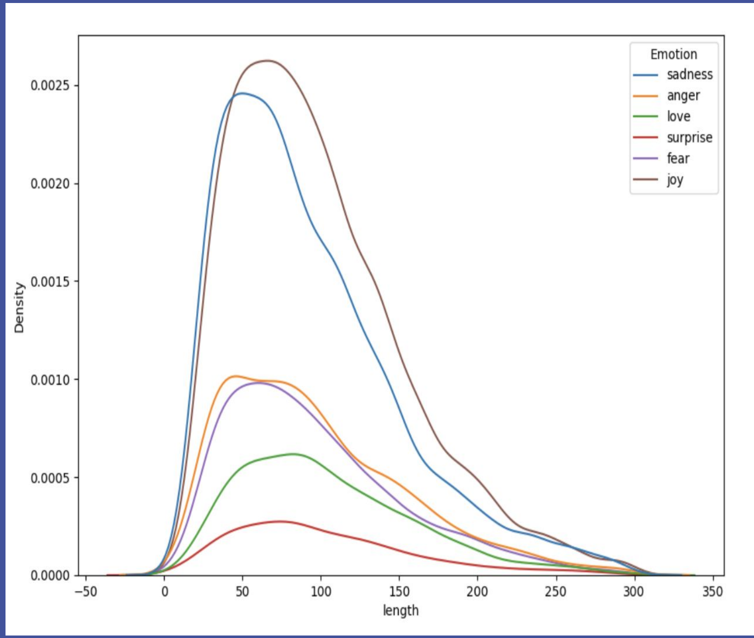
A collection of textual data that consists of over 13,000 tweets labeled with one of six emotions: anger, fear, joy, love, sadness, and surprise. Dataset is split into train, test & validation for building the machine learning model,

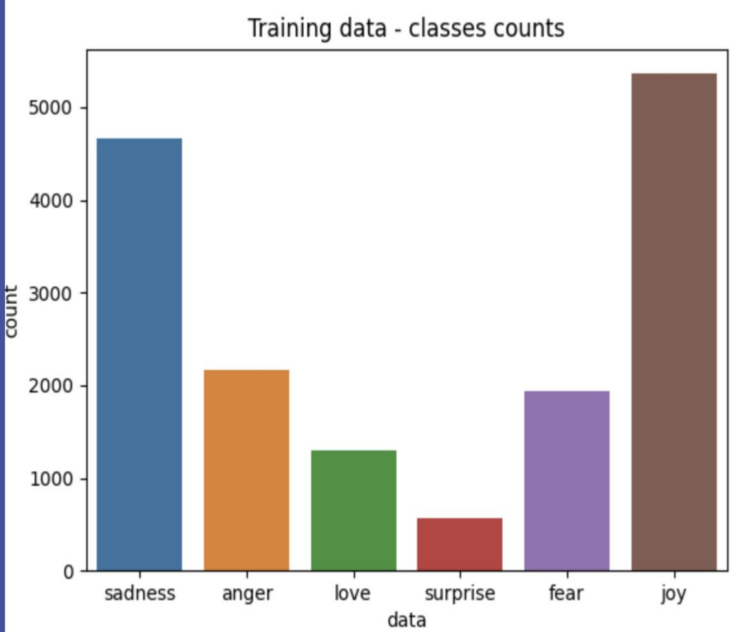
Example:
i feel like I am still looking at a blank canvas blank pieces of paper;sadness.

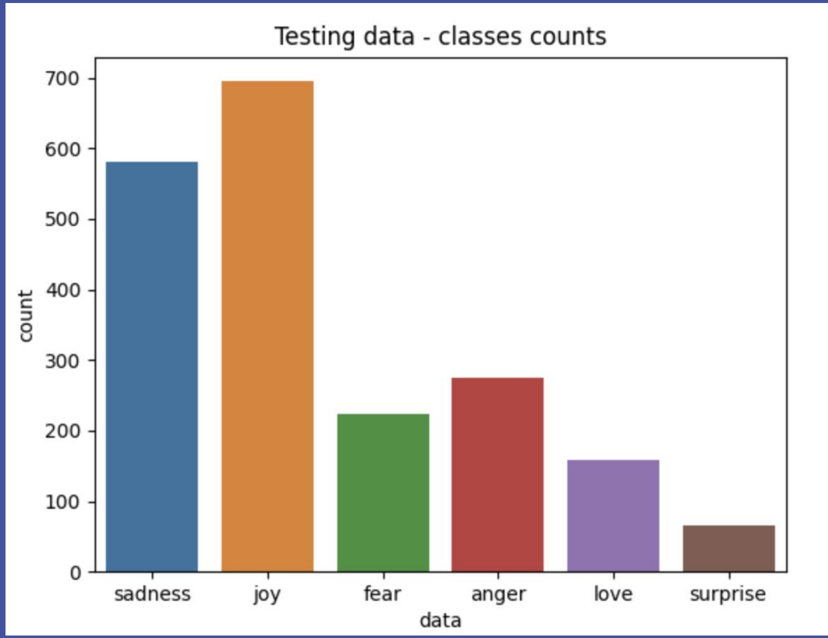
Source:

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

EDA







300

Maxlen

6 classes

Unbalanced

Data Preprocessing

Encoding Labels: Converting the sentiment labels to numerical values.

Remove URLs from the text.

Remove mentions (words starting with '@' that refer to Twitter users).

Remove numbers from the text.

Remove all punctuation except for spaces.

Convert all text to lowercase.

Tokenize the text into individual words.

Remove common stop words like 'the', 'and', 'a' from the text.

Lemmatize the words (i.e., convert them to their base form).

Join the words back into a string and return the cleaned text.

```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 Selection

1

We used a deep learning model called a Long Short-Term Memory (LSTM) network to classify the emotions. LSTMs are a type of Recurrent Neural Network (RNN) that are effective for modeling sequential data like text.

2

We also used the transformer model to classify the emotions. First, we used a pre-trained Bert model directly. Then,we fine-tuned the pre-trained Bert model by adding a LSTM layer.

Training and Evaluation

- Split the dataset into training, validation, and testing sets.
- Train the LSTM model on the training set, optimizing for accuracy.
- Validate the model on the validation set to tune hyperparameters and prevent overfitting.
- Evaluate the model on the testing set to measure its performance on unseen data.

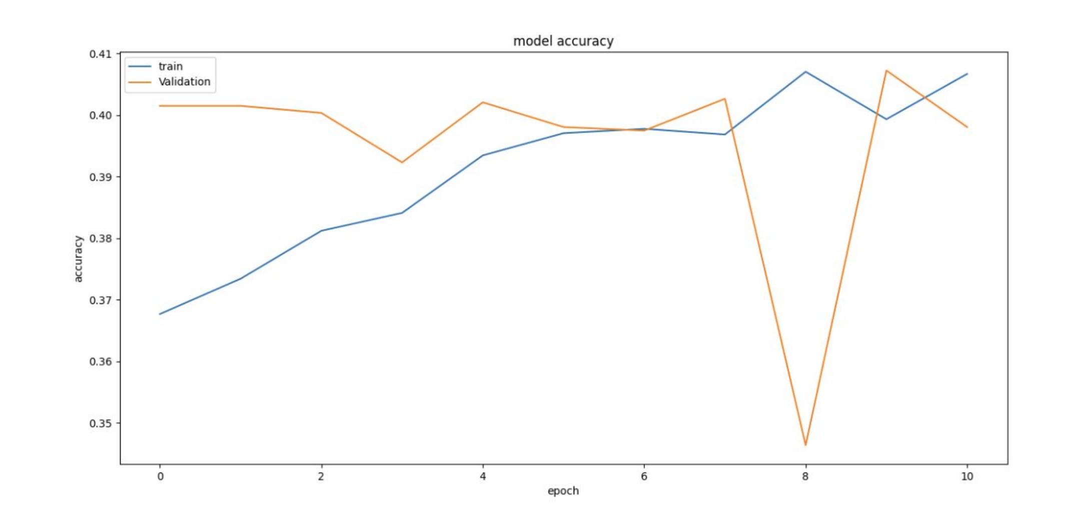
max_features = 14324
maxlen = 256
embedding_size = 200
kernel_size = 5
filters = 128
pool_size = 4
lstm_output_size = 128
input_length=256

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)	256-5+1
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)	

Methodology and results

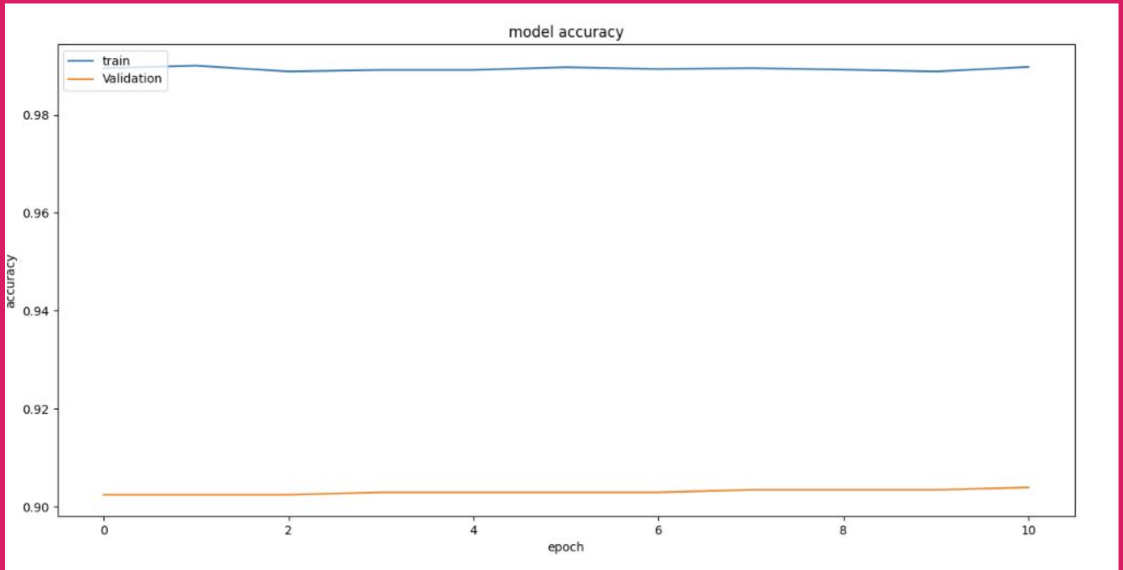
1. Use pre-trained word embeddings, such as Word2Vec, to initialize the embedding layer. This can help the model learn better representations of words and reduce the risk of overfitting.

```python
# create an embedding matrix
embedding_matrix = np.zeros((vocabSize, embedding_dim))
for word, i in tokenizer.word_index.items():
    if word in model_emb.wv.key_to_index:
        embedding_matrix[i] = model_emb.wv[word]
```

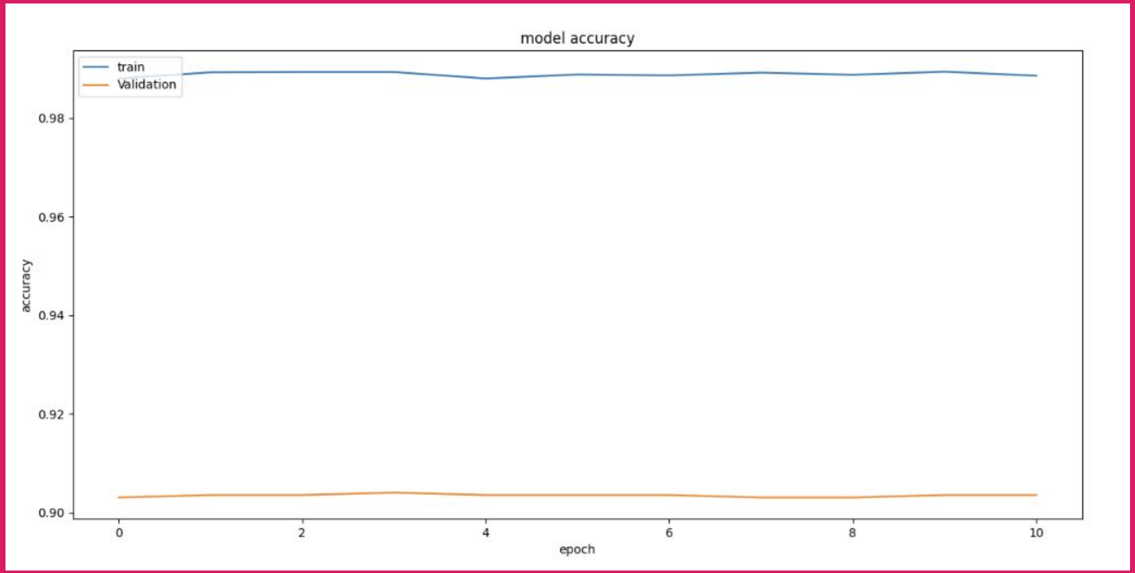


Methodology and results

3. Experiment with different hyperparameters (Optimizer)



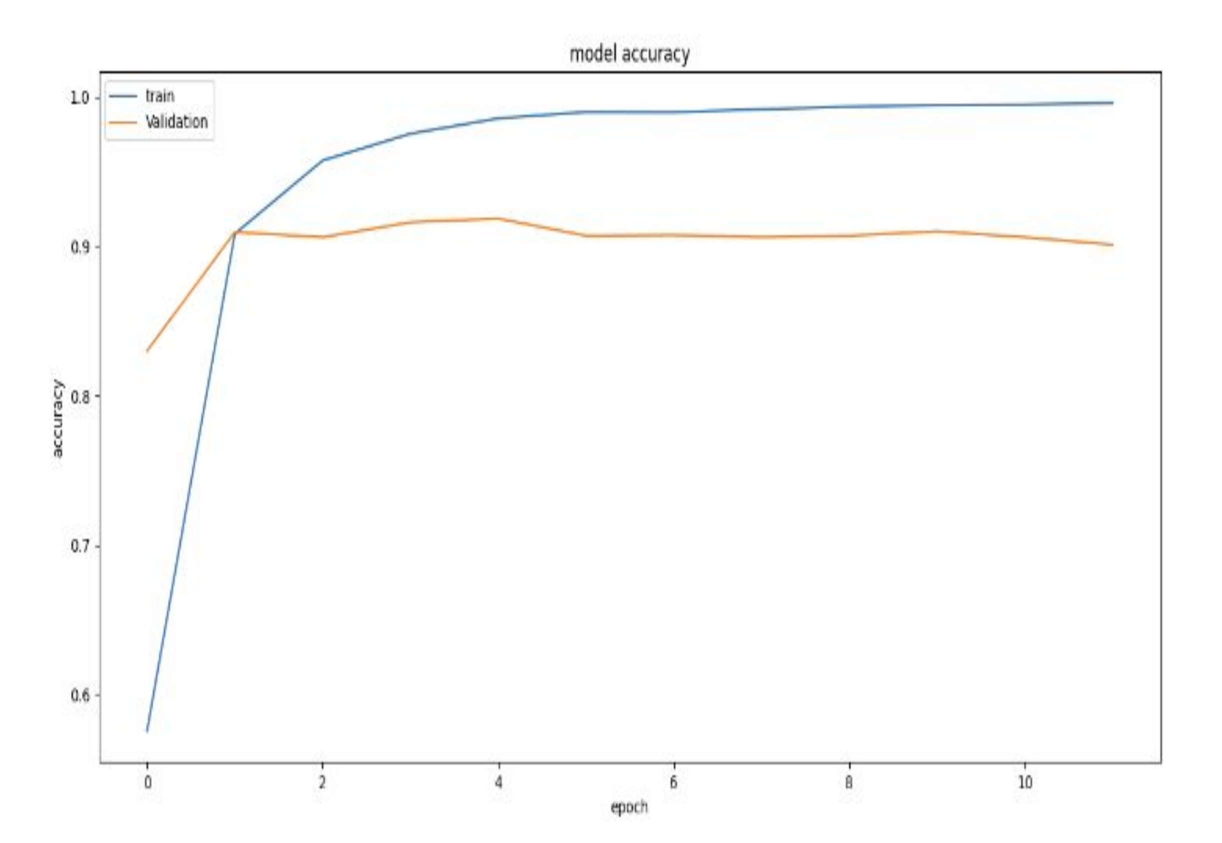
SGD



Adagrad

Methodology and results

3. Experiment with different hyperparameters (Optimizer)

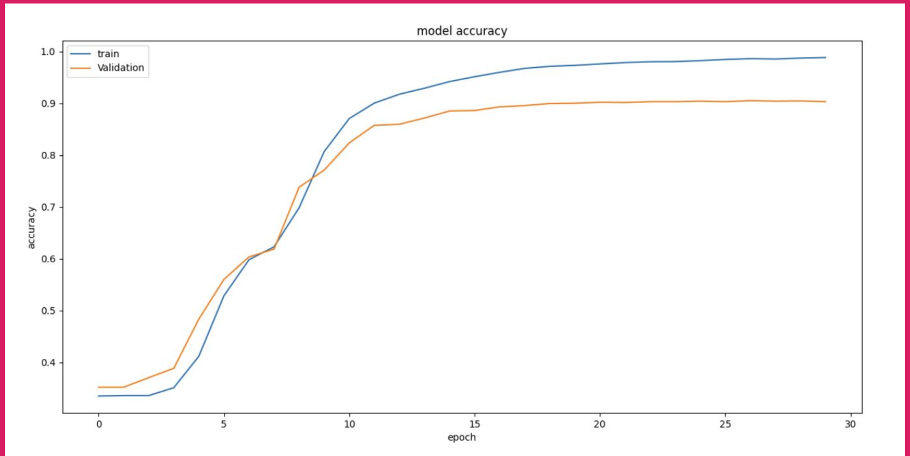



```
>>> model.evaluate(X_test, y_test, verbose=1)
63/63 [==============================] - 0s 7ms/step - loss: 0.2432 - accuracy: 0.8970
[0.24321456253528595, 0.8970000147819519]
```

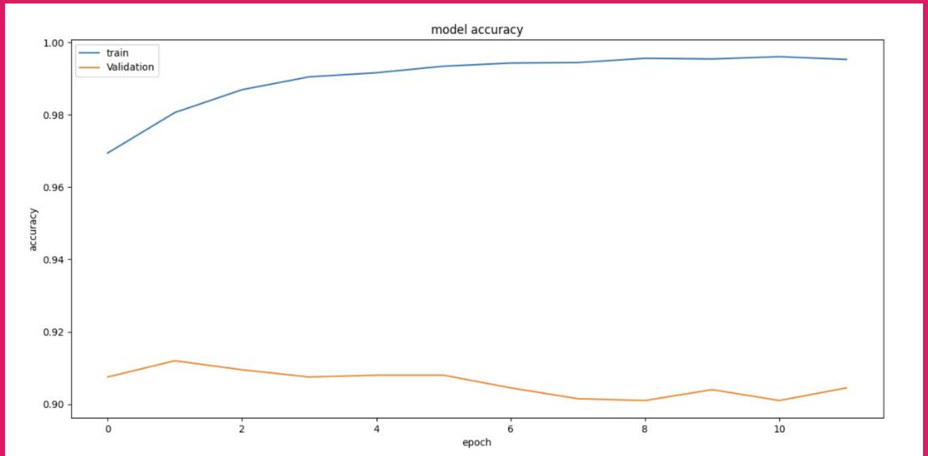
Adam

Methodology and results

2. Experiment with different hyperparameters (Learning Rate)



LR: 0.0001



LR: 0.001

```
>>> model.evaluate(X_test, y_test, verbose=1)
63/63 [==============================] - 0s 7ms/step - loss: 0.3108 - accuracy: 0.8925
[0.3107646703720093, 0.8924999833106995]
```

```
>>> model.evaluate(X_test, y_test, verbose=1)
63/63 [==============================] - 0s 7ms/step - loss: 0.2744 - accuracy: 0.9045
[0.27435314655303955, 0.9045000076293945]
```

Model Architecture (Transformer)

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

- distilBERT: a distilled form of the BERT model
- Reduced 40% knowledge in pre-training
- Retaining 97% of its language understanding abilities and being 60% faster

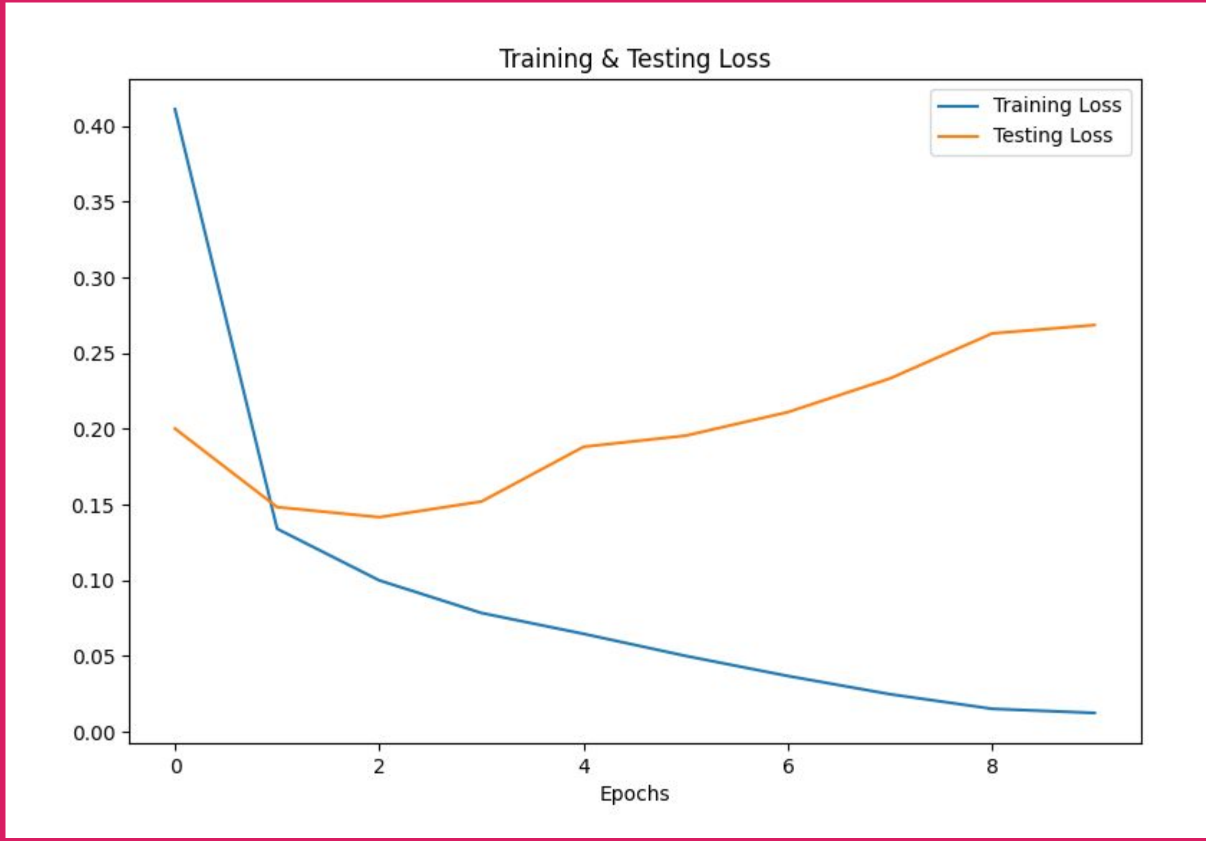
Model Architecture (Transformer)

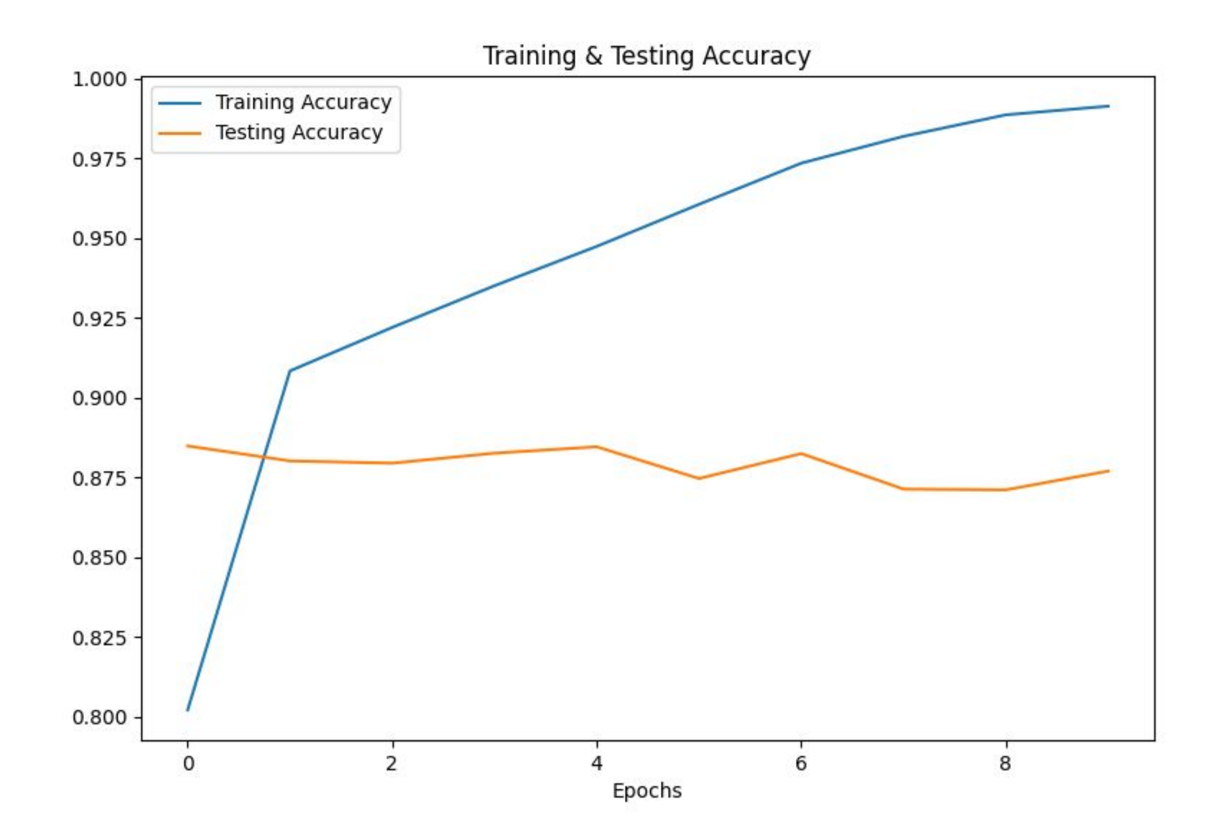
```python
checkpoint = "distilbert-base-uncased"
model = DistilBertForSequenceClassification.from_pretrained(checkpoint, num_labels=6)
```

```python
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
```

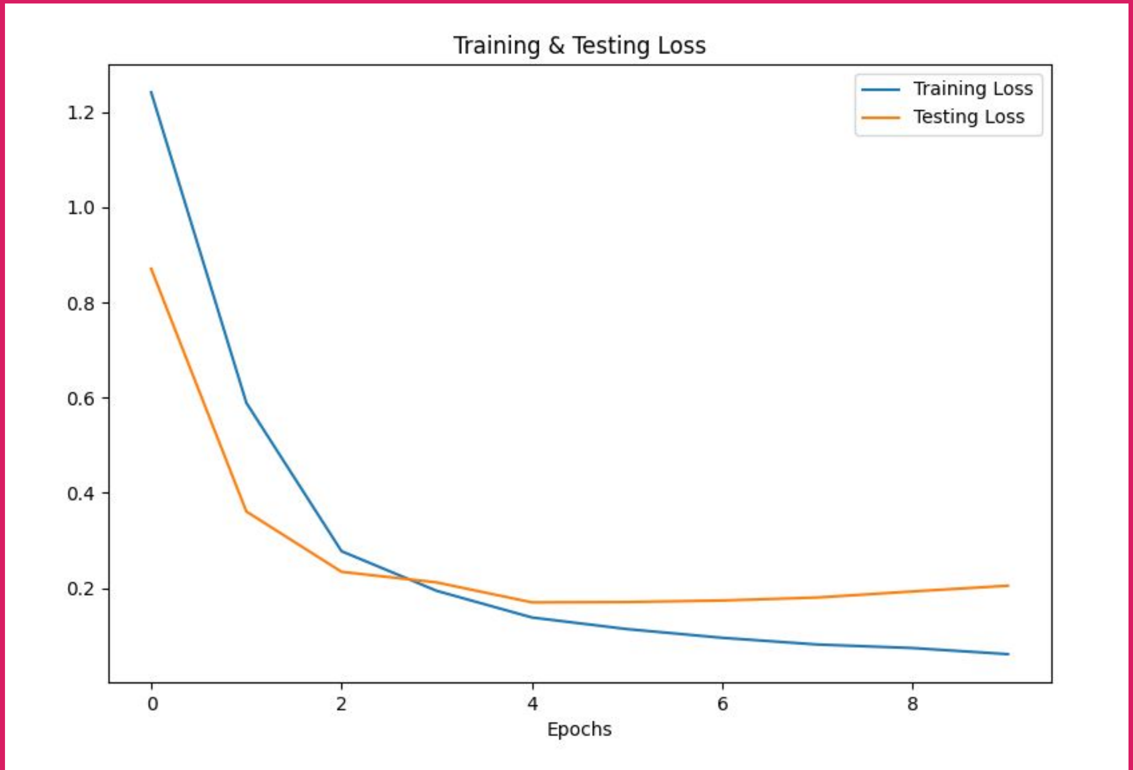
Model Result (Transformer)





The loss and f1 score (macro) for Distil Bert model

Model Result (Transformer)

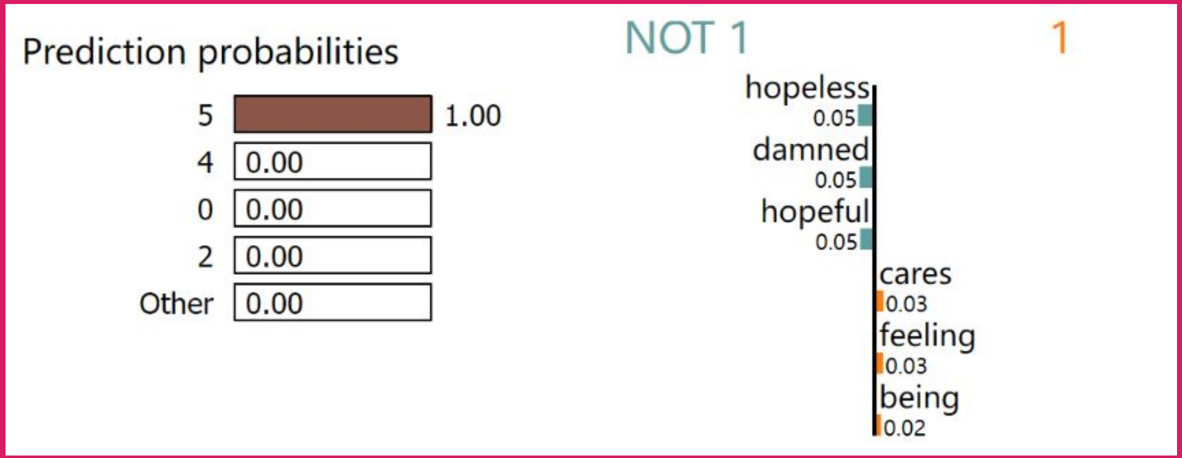


The loss and f1 score (macro) for Bert with LSTM model

Model Comparison

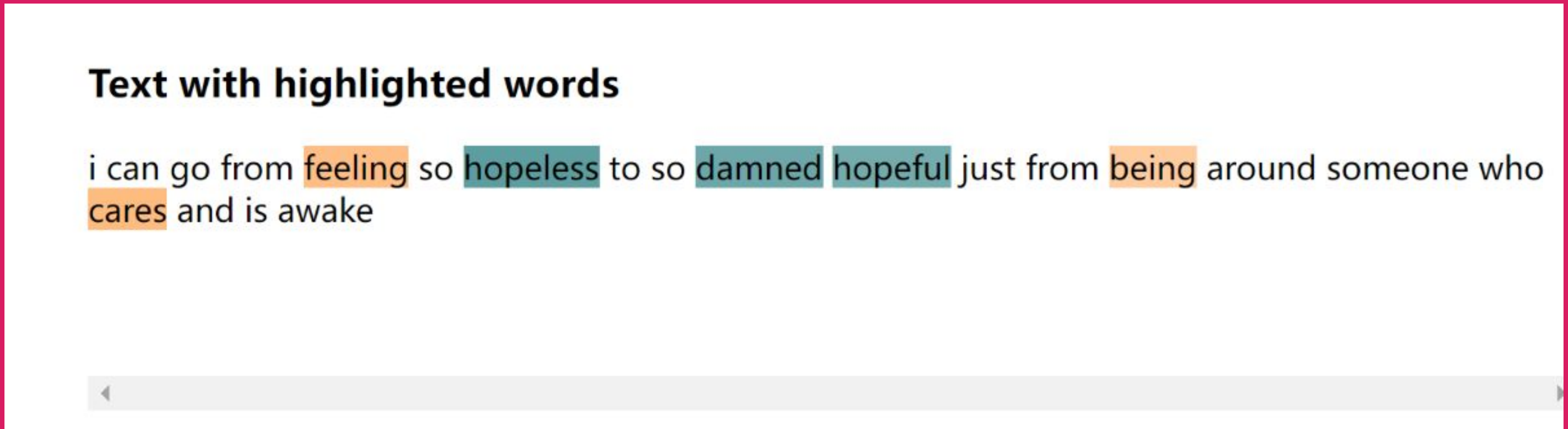
	f1_score (micro)	f1_score (macro)
LSTM+CNN	0.912	0.867
Transform	0.940	0.915
Transform+LSTM	0.936	0.907

Model Interpreter (Lime)



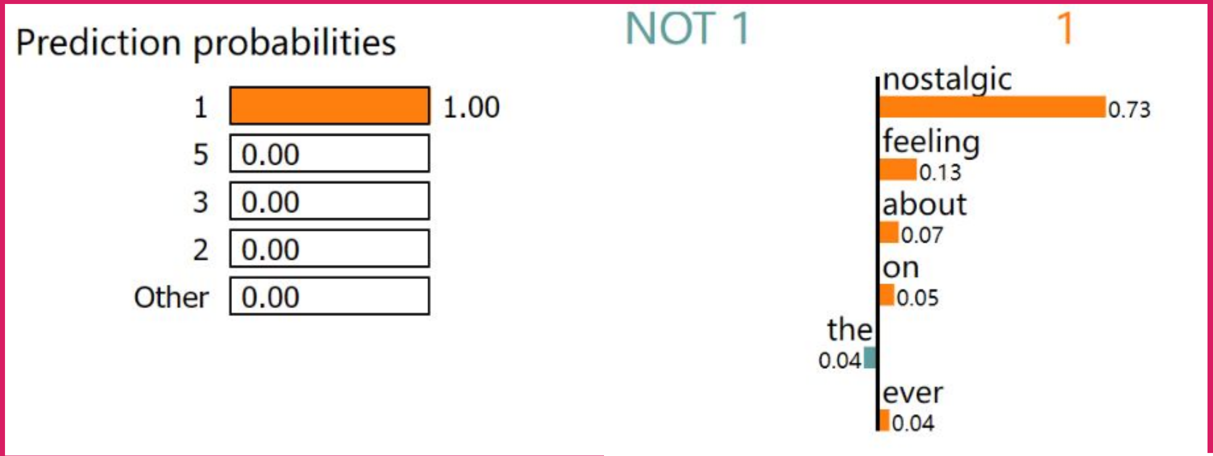
1: love

5: sadness



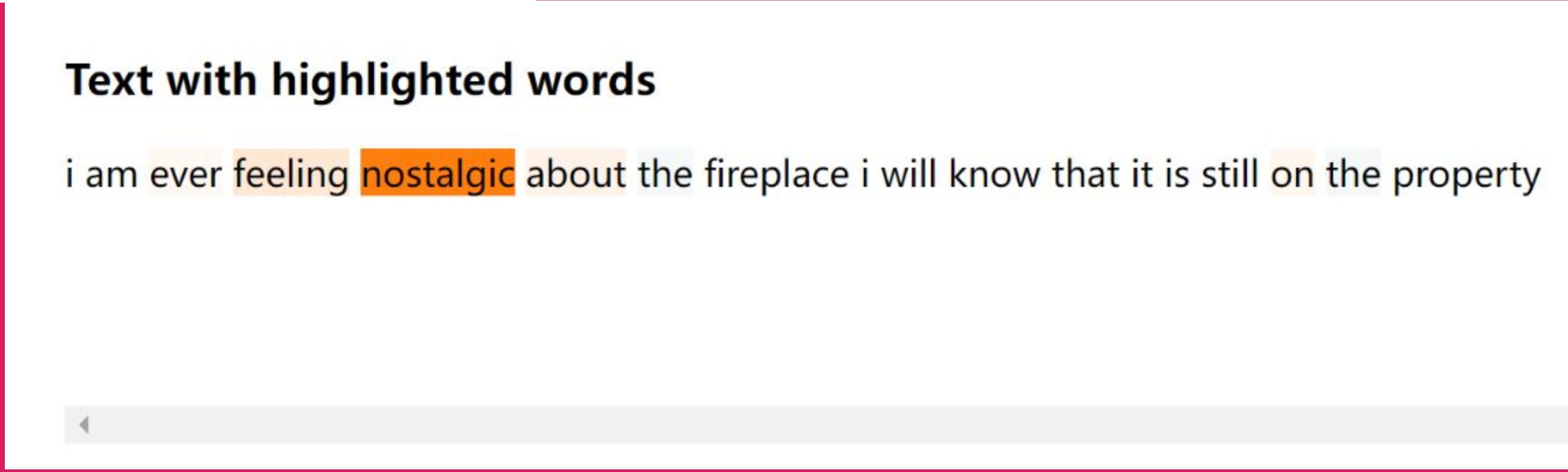
We used Lime library to explain the prediction of the model for this specific sample

Model Interpreter (Lime)



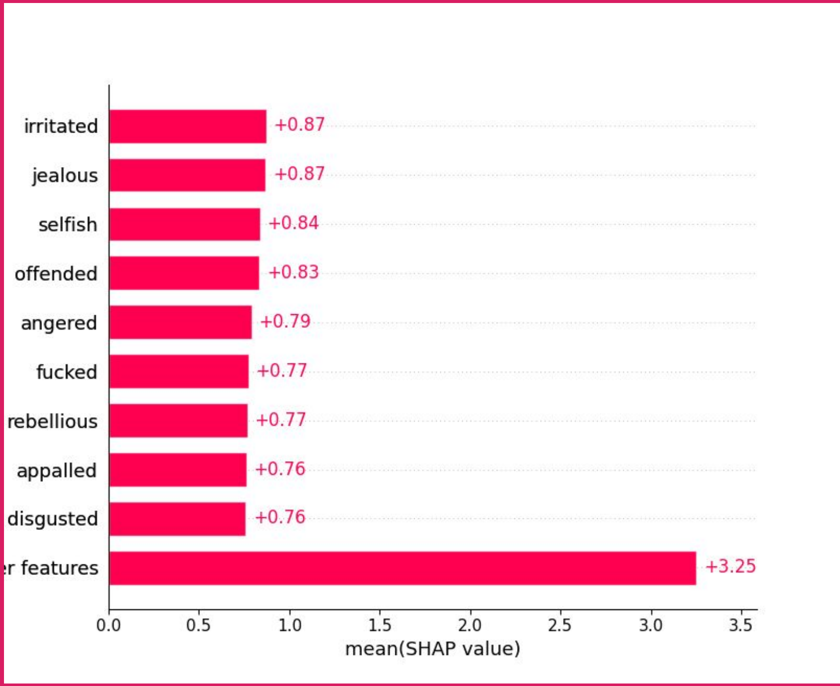
1: love

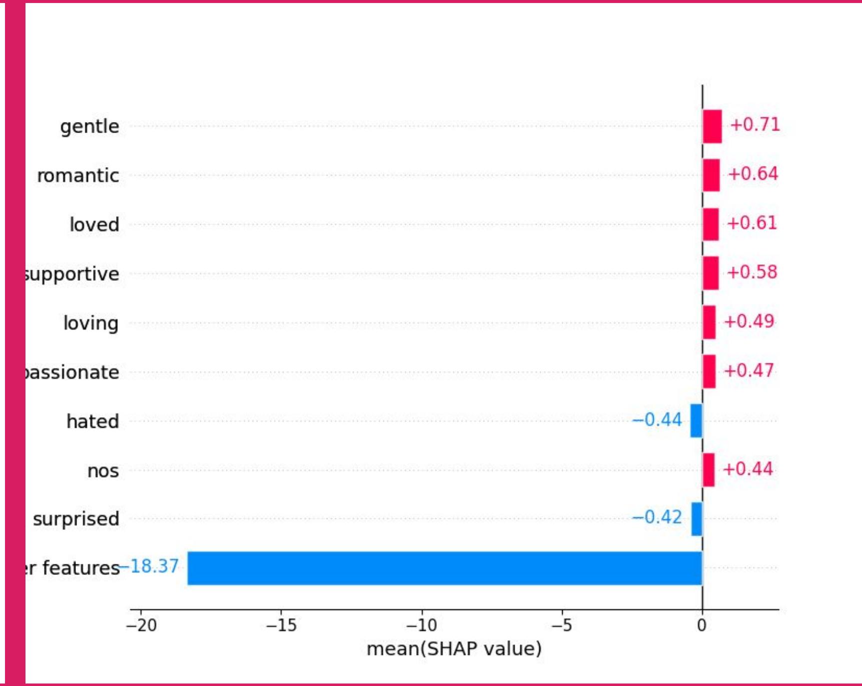
5: sadness

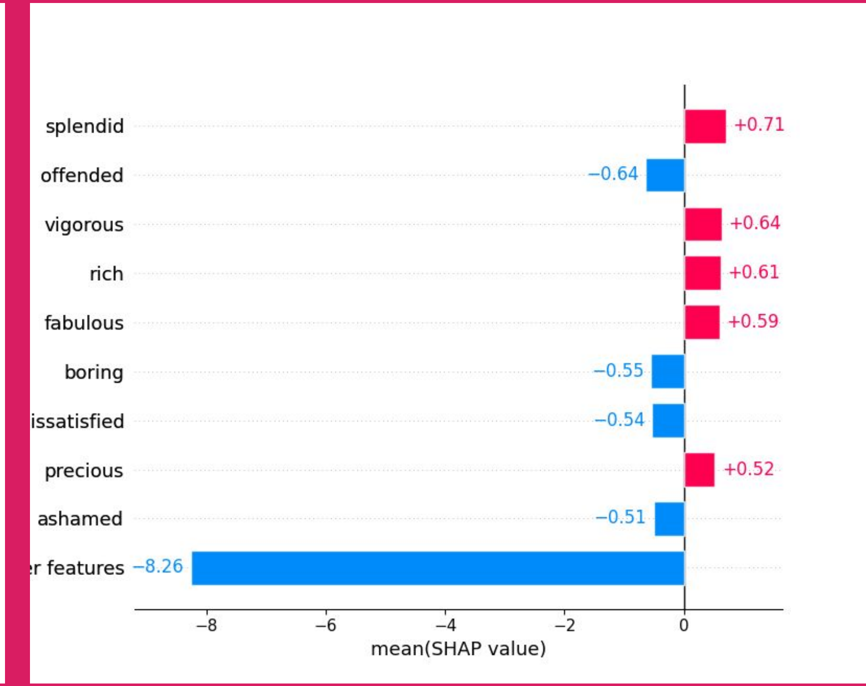


We used Lime library to explain the prediction of the model for this specific sample

Model Interpreter (Shap)







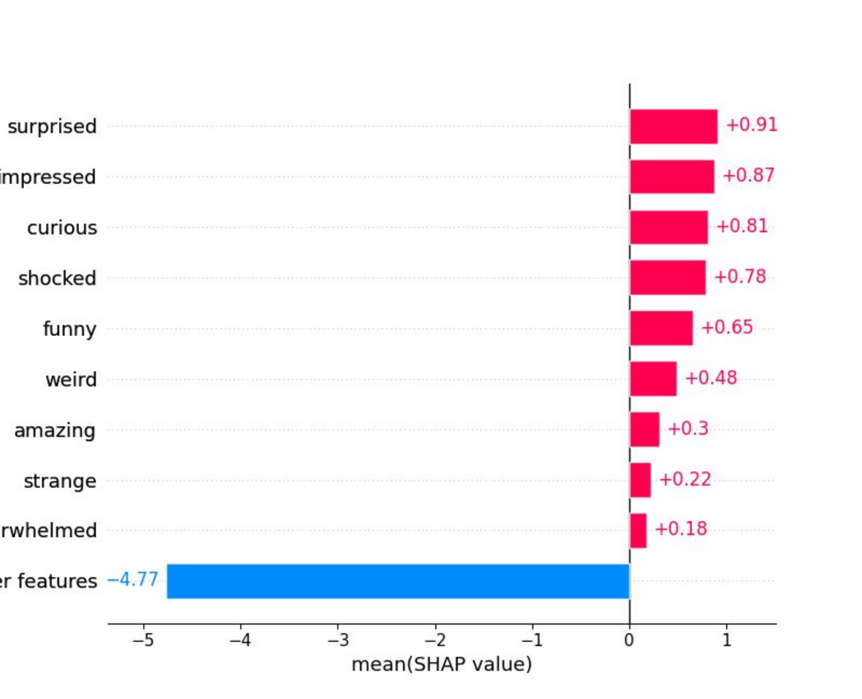
anger

love

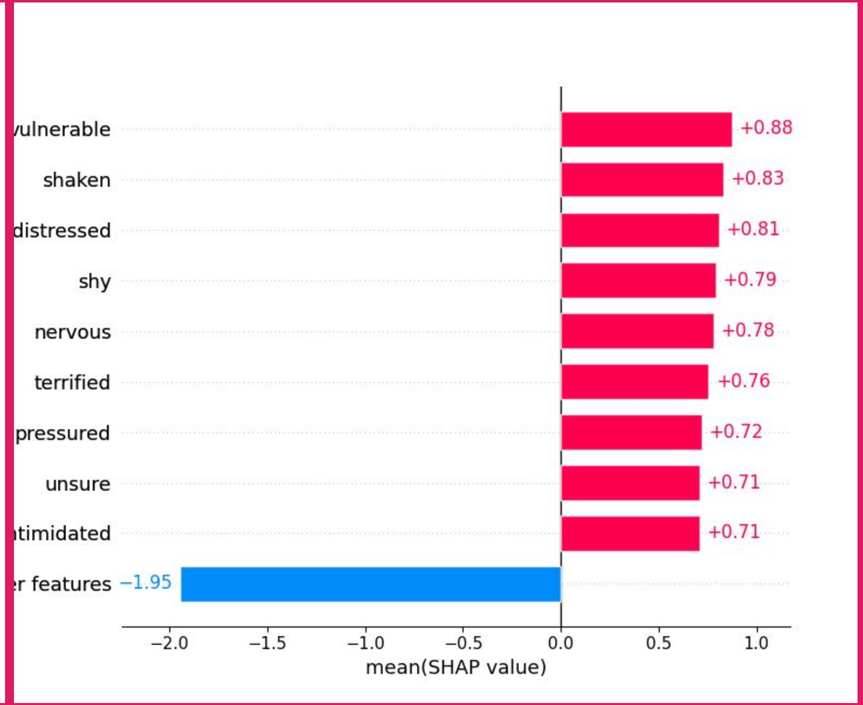
joy

We used Shap library to try to show the top words impacting a specific class

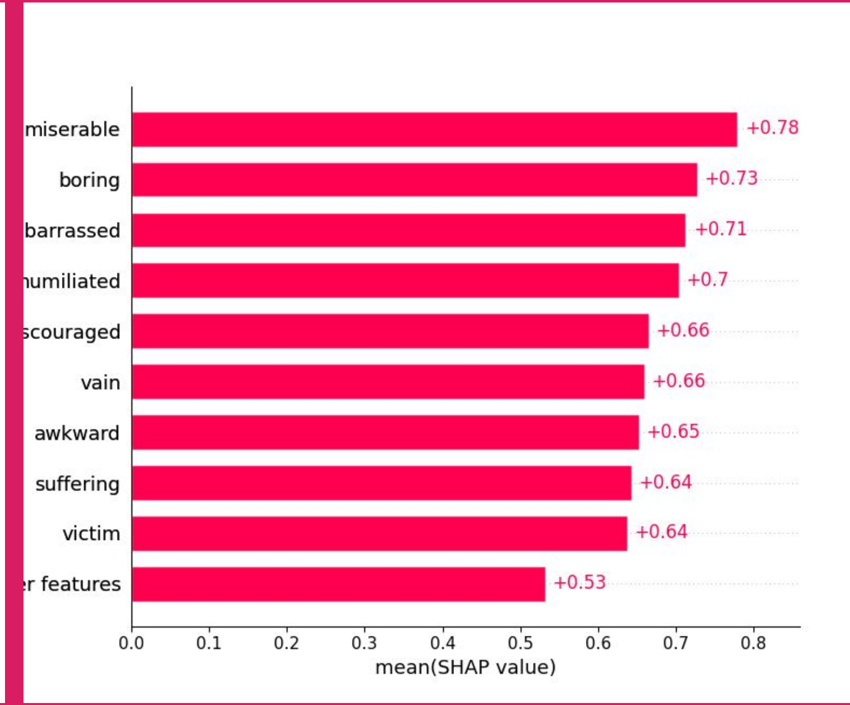
Model Interpreter (Shap)



surprise



fear



sadness

All the value of anger and sadness are positive

Summary

We built three models and compared performance of different models to classify tweet text into different emotion classes

The pre-trained transformer models have the high f1 score which means they are suitable for this task compared with LSTM model

We interpreted the results based on and Lime and Shap

Thank you

Have a nice summer break!