Individual Final Report

Xiao Qi

1. Introduction

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 tuning and model evaluation. We built the LSTM model and the fine-tuned transform model at the same time and compared the results of different models. After that, we did some model interpretation work, in order to understand the models better.

The results of this project can have practical applications in various fields, including mental health, marketing, and social media analysis.

2. Description of my individual work

(1) Models Built

I built the DistilBERT model and Bert model with a LSTM head, trained the model, and evaluated the result. I also tried different methods, like changing batch size, max length of input, learning rate, optimizers, epoches, to improve the results.

(2) Model Interpretation

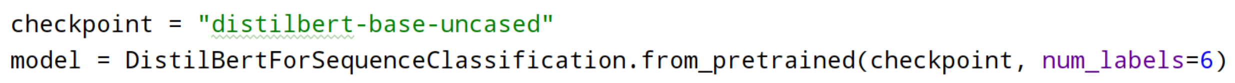
I used the explain_instance method in LIME to explain the prediction of the model for this specific sample. Meanwhile, I used a larger portion of the dataset of 200 sentences from the training dataset, putting them into the SHAP to see the top words impacting each specific class.

(3) Report and slide writing

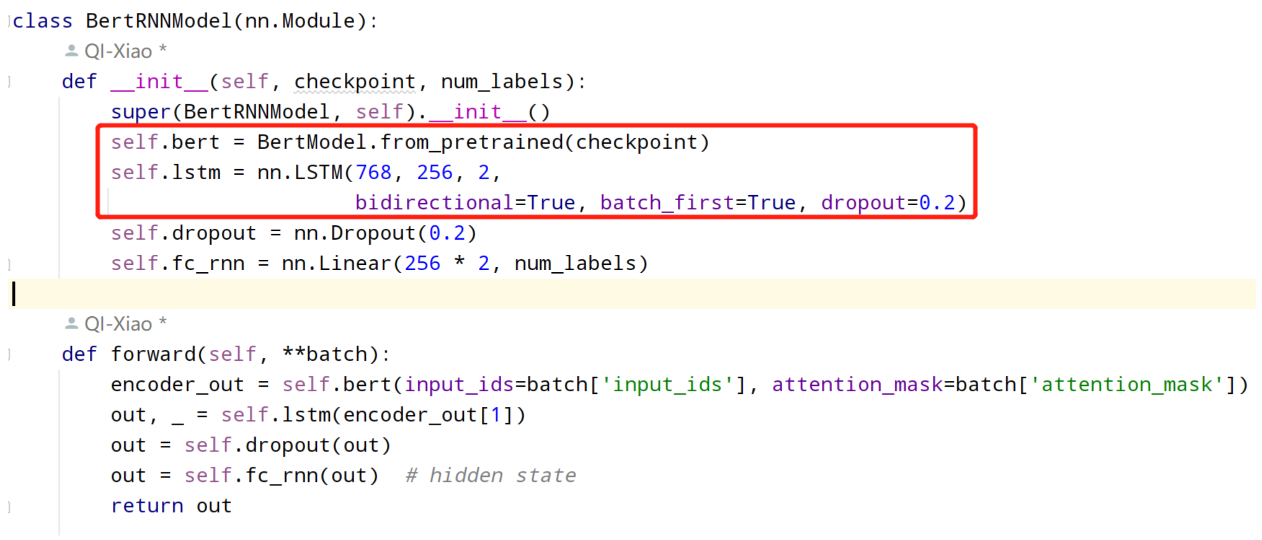
Finished the report and slides with team members.

3. Describe the portion of the work

(1) Model built



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



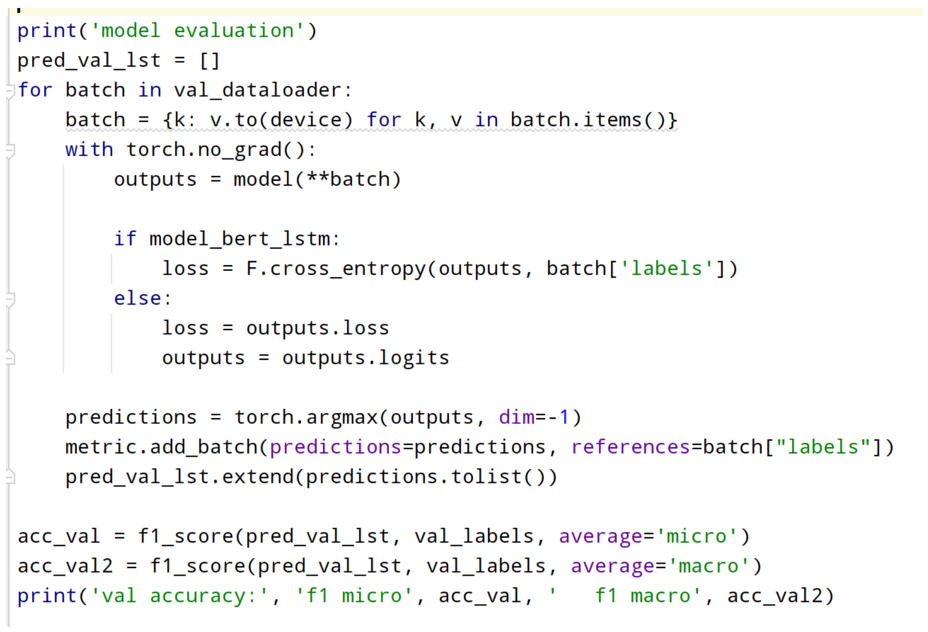
```
class BertRNNModel(nn.Module):
    QI-Xiao *
    def __init__(self, checkpoint, num_labels):
        super(BertRNNModel, self).__init__()
        self.bert = BertModel.from_pretrained(checkpoint)
        self.lstm = nn.LSTM(768, 256, 2,
                            bidirectional=True, batch_first=True,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

(2) Train and test the model

```
for epoch in range(num_epochs):

    train_loss, steps_train = 0, 0
    pred_train_lst = []
    origin_train_lst = []

    model.train()
    with tqdm(total=len(train_dataloader), desc="Training Epoch {}".format(epoch)) as pbar:
        for batch in train_dataloader:
            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            if model_bert_lstm:
                loss = F.cross_entropy(outputs, batch['labels'])
            else:
                loss = outputs.loss
                outputs = outputs.logits

            loss.backward()

            optimizer.step()
            lr_scheduler.step()
            optimizer.zero_grad()
            pbar.update(1)

            train_loss += loss
            steps_train += 1

            predictions = torch.argmax(outputs, dim=-1)
            metric.add_batch(predictions=predictions, references=batch["labels"])
            pred_train_lst.extend(predictions.tolist())
            origin_train_lst.extend(batch['labels'].tolist())
```

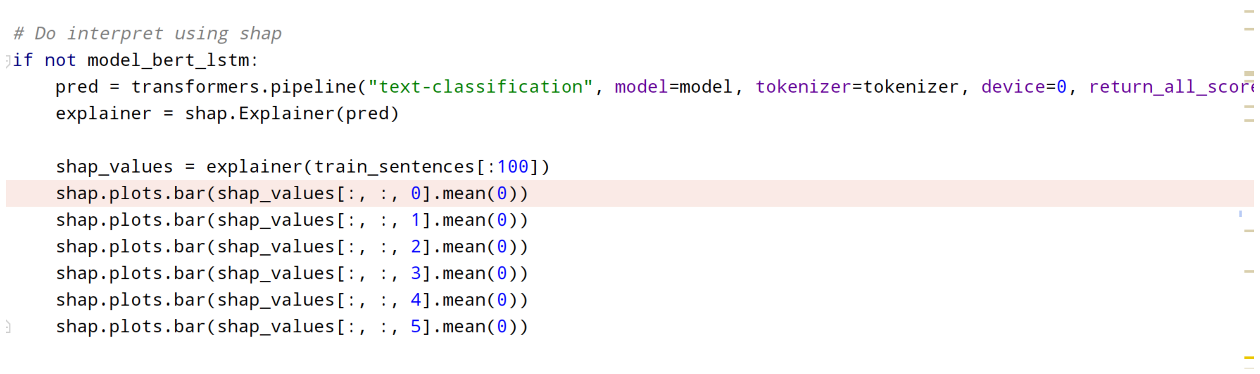
```
    model.eval()
    pred_test_lst = []
    test_loss, steps_test = 0, 0

    with tqdm(total=len(test_dataloader), desc="Test Epoch {}".format(epoch)) as pbar:
        for batch in test_dataloader:
            batch = {k: v.to(device) for k, v in batch.items()}
            with torch.no_grad():
                outputs = model(**batch)

                if model_bert_lstm:
                    loss = F.cross_entropy(outputs, batch['labels'])
                else:
                    loss = outputs.loss
                    outputs = outputs.logits

            pbar.update(1)

            test_loss += loss
            steps_test += 1

            predictions = torch.argmax(outputs, dim=-1)
            metric.add_batch(predictions=predictions, references=batch["labels"])
            pred_test_lst.extend(predictions.tolist())

    acc_test = f1_score(pred_test_lst, test_labels, average=f1_type)
    avg_test_loss = (test_loss / steps_test).item()

    print('epoch' + str(epoch), 'train loss', avg_train_loss, 'train f1 '+f1_type, acc_train)
    print('test loss', avg_test_loss, 'test f1 '+f1_type,  acc_test, '\n')

    train_loss_lst.append(avg_train_loss)
```

(3) Model evaluation



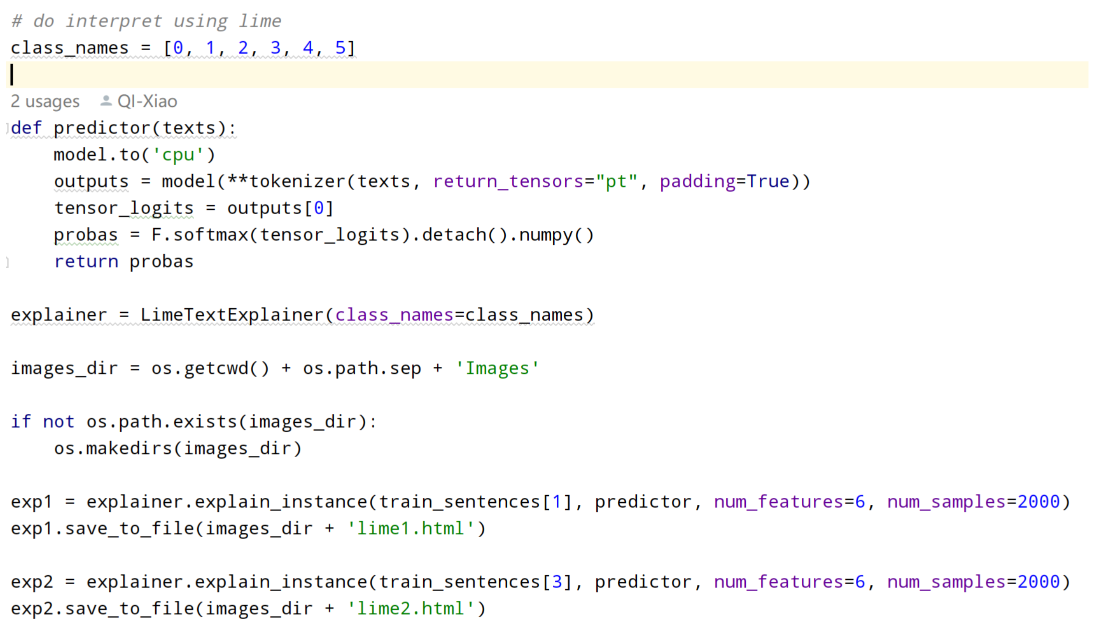
```
print('model evaluation')
pred_val_lst = []
for batch in val_dataloader:
    batch = {k: v.to(device) for k, v in batch.items()}
    with torch.no_grad():
        outputs = model(**batch)

        if model_bert_lstm:
            loss = F.cross_entropy(outputs, batch['labels'])
        else:
            loss = outputs.loss
            outputs = outputs.logits

    predictions = torch.argmax(outputs, dim=-1)
    metric.add_batch(predictions=predictions, references=batch["labels"])
    pred_val_lst.extend(predictions.tolist())

acc_val = f1_score(pred_val_lst, val_labels, average='micro')
acc_val2 = f1_score(pred_val_lst, val_labels, average='macro')
print('val accuracy:', 'f1 micro', acc_val, '   f1 macro', acc_val2)
```

(4) Model interpretation using SHAP



```
# Do interpret using shap
if not model_bert_lstm:
    pred = transformers.pipeline("text-classification", model=model, tokenizer=tokenizer, device=0, return_all_scor
    explainer = shap.Explainer(pred)

    shap_values = explainer(train_sentences[:100])
    shap.plots.bar(shap_values[:, :, 0].mean(0))
    shap.plots.bar(shap_values[:, :, 1].mean(0))
    shap.plots.bar(shap_values[:, :, 2].mean(0))
    shap.plots.bar(shap_values[:, :, 3].mean(0))
    shap.plots.bar(shap_values[:, :, 4].mean(0))
    shap.plots.bar(shap_values[:, :, 5].mean(0))
```

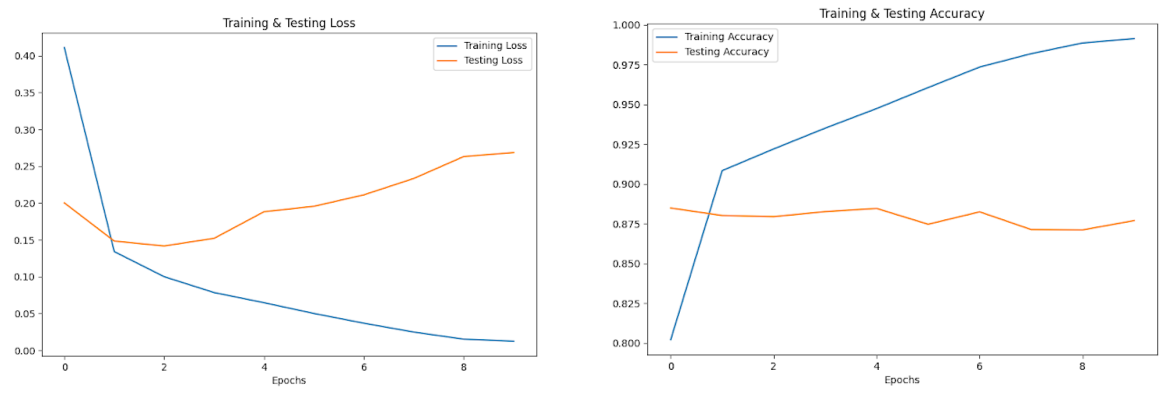
(5) Model interpretation using LIME



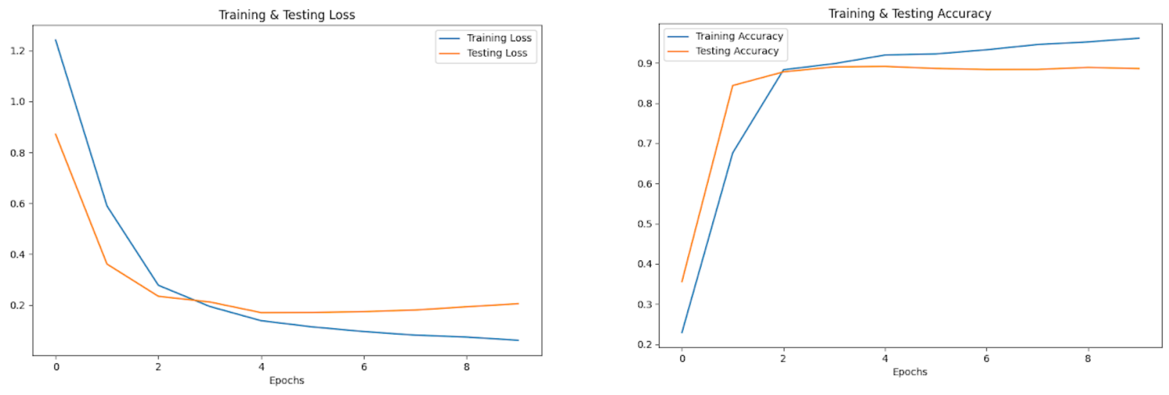
```
# do interpret using lime
class_names = [0, 1, 2, 3, 4, 5]

2 usages  QI-Xiao
def predictor(texts):
    model.to('cpu')
    outputs = model(**tokenizer(texts, return_tensors="pt", padding=True))
    tensor_logits = outputs[0]
    probas = F.softmax(tensor_logits).detach().numpy()
    return probas

explainer = LimeTextExplainer(class_names=class_names)

images_dir = os.getcwd() + os.path.sep + 'Images'

if not os.path.exists(images_dir):
    os.makedirs(images_dir)

exp1 = explainer.explain_instance(train_sentences[1], predictor, num_features=6, num_samples=2000)
exp1.save_to_file(images_dir + 'lime1.html')

exp2 = explainer.explain_instance(train_sentences[3], predictor, num_features=6, num_samples=2000)
exp2.save_to_file(images_dir + 'lime2.html')
```

4. Results

(1)Train and test results from different models

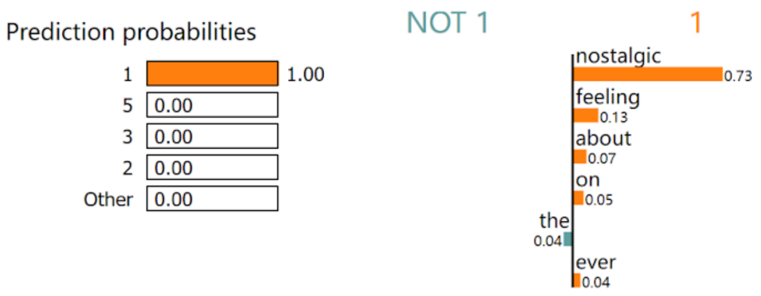


DistilBERT model



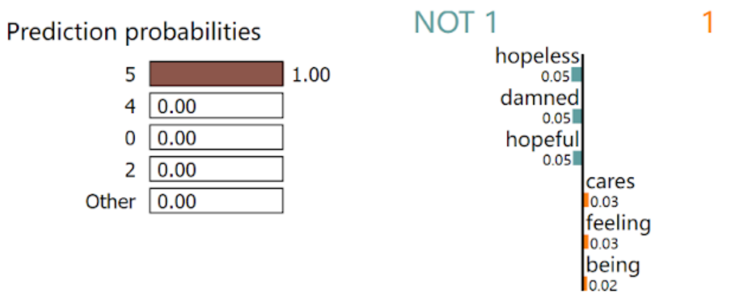
Transform model with LSTM head

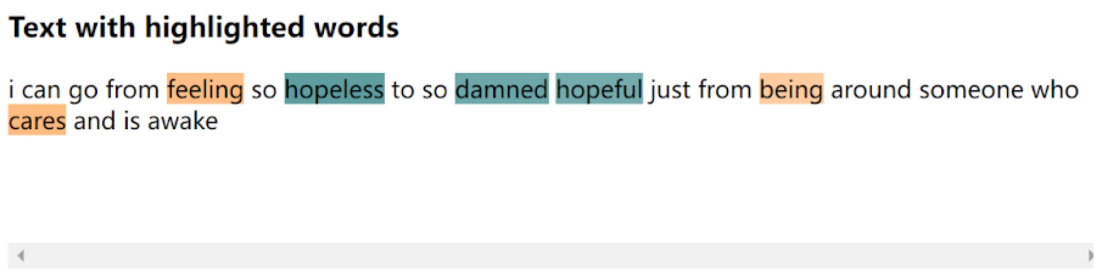
(2)LIME interpretation



Text with highlighted words

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

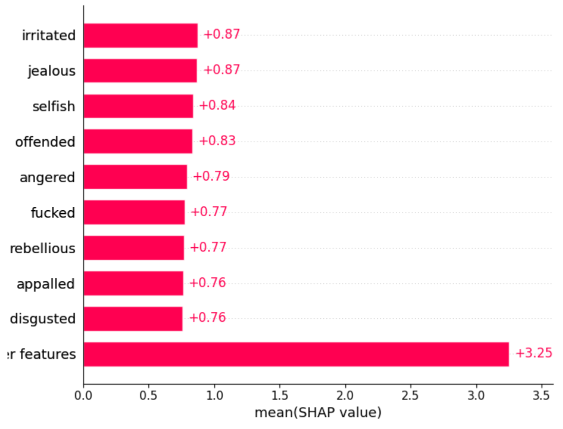




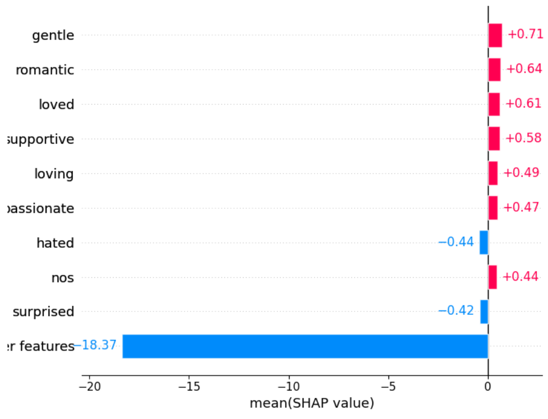
Text with highlighted words

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

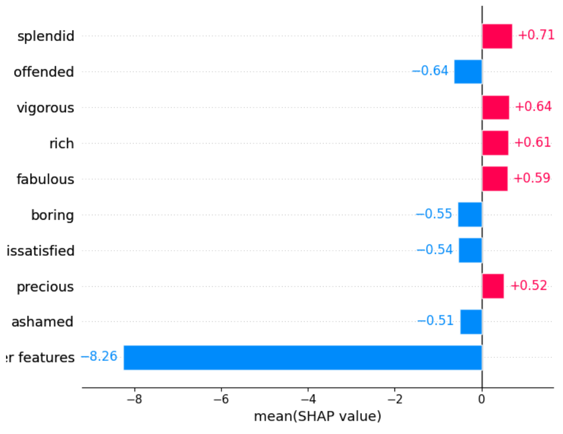
(3)SHAP interpretation



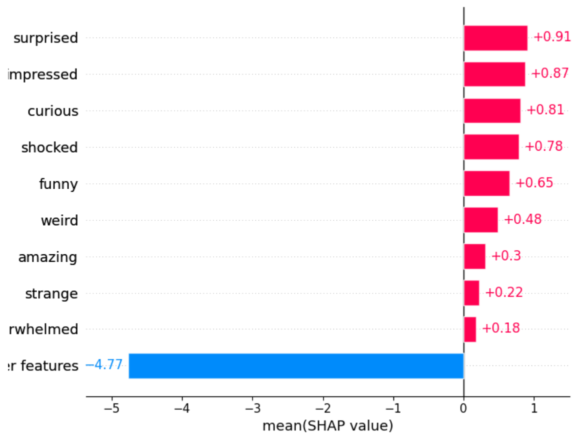
anger



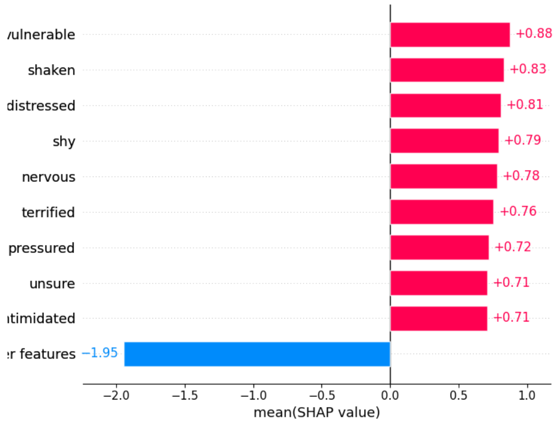
love



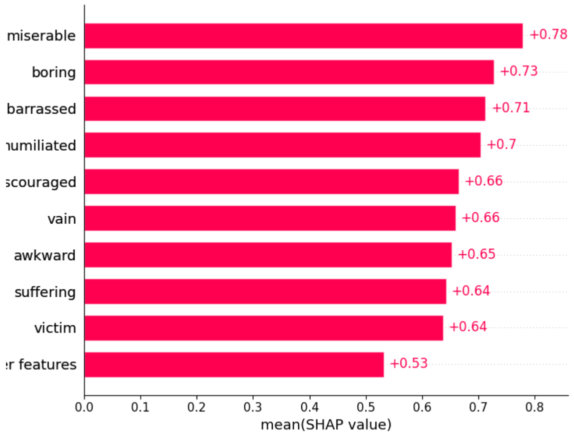
joy



surprise



fear



sadness

5. Summary and conclusions

(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.

6. 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