Sentiment Analysis Report Based on BERT(Draft)

Authors: Teng-Yu Hsiao, YiTao Shi, Yue Yu
NetID: th331,ys386,yy373

Abstract

This report delves into the application of BERT (Bidirectional Encoder Representations from Transformers) for sentiment analysis and emotion classification tasks. Leveraging a labeled dataset, the study explores data preprocessing techniques, tokenization, and model fine-tuning to classify text into six distinct emotion categories: anger, fear, joy, love, sadness, and surprise. Key insights into the performance of the model are presented, including accuracy metrics and challenges posed by class imbalance. The findings highlight the effectiveness of BERT in achieving high accuracy for text-based emotion recognition, while emphasizing areas for improvement, such as handling imbalanced datasets and exploring advanced architectures for further optimization. This report contributes to the understanding and application of pre-trained NLP models in real-world sentiment analysis scenarios.

I. Introduction

Sentiment analysis, a key task in Natural Language Processing (NLP), focuses on identifying, extracting, and analyzing subjective information from text data. It has diverse applications, including social media monitoring, market research, and customer feedback analysis. The advent of deep learning, particularly pre-trained models like BERT (Bidirectional Encoder Representations from Transformers), has greatly enhanced the accuracy and efficiency of sentiment analysis methods.

This report presents an in-depth exploration and implementation of a sentiment analysis task using the provided dataset and code. It covers data introduction, preprocessing steps, an overview of the BERT model, a detailed explanation of the technical approach and framework used in the code, GPU acceleration techniques, and a thorough analysis of the results.

Emotion recognition in text is another critical NLP task, enabling systems to understand human emotions for applications such as customer service, mental health assessment, and social media analysis. This report also details the development of a BERT-based classifier for classifying text into various emotion categories using labeled datasets.

II. Data Introduction

1. Dataset Overview

The provided dataset consists of three text files: `train.txt`, `test.txt`, and `val.txt`.

`train.txt`: Large with diverse examples for robust model training.

`val.txt`.: Moderate-sized for tuning hyperparameters.

`test.txt`: Small but sufficient for evaluating generalization.

Each file contains lines with the following format:

```
im feeling quite sad and sorry for myself but ill snap out of it soon;sadness
i feel like i am still looking at a blank canvas blank pieces of paper;sadness
i feel like a faithful servant;love
```

Each line comprises a text sentence and its corresponding emotion label, separated by a semicolon (`;`).

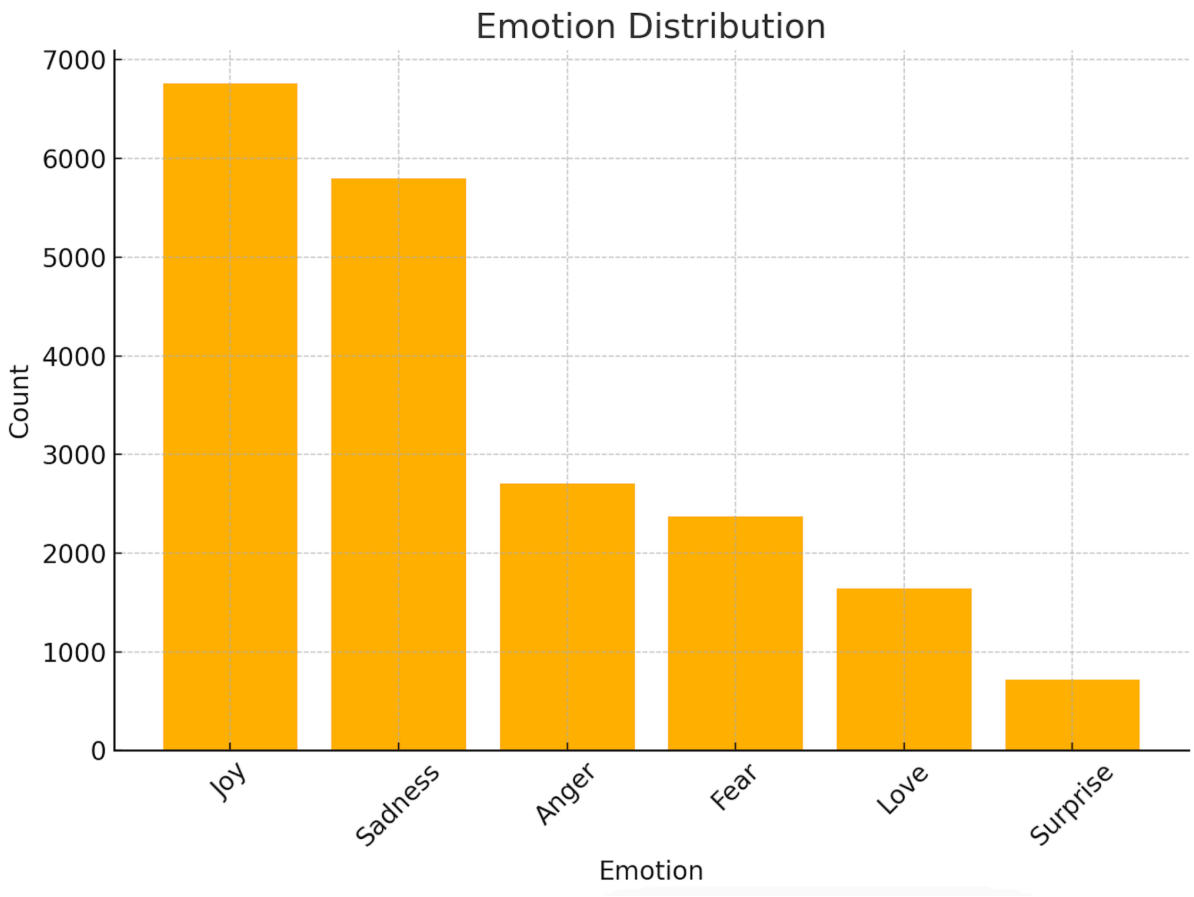
2. Emotion Labels

The dataset includes six different emotion labels:

- `anger`
- `fear`
- `joy`
- `love`
- `sadness`
- `surprise`

3. Data Distribution

After combining the three datasets into one, we analyze the distribution of samples across the emotion labels:



Note: It is evident that the dataset is imbalanced, with `joy` and `sadness` having significantly more samples, while `surprise` has the fewest.

III. Data Preprocessing

1. Data Merging and Label Encoding

- **Data Loading and Merging**: Use **pandas** to read the three datasets and merge them into a single DataFrame.
- **Label Encoding**: Utilize **`LabelEncoder`** from **`scikit-learn`** to convert the textual emotion labels into numerical labels for model processing.

```
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
df['label_enc'] = labelencoder.fit_transform(df['label'])
```

The label encoding results are as follows:

Emotion Label (label_desc)	Encoded Label (label)
sadness	4
anger	0
love	3
surprise	5
fear	1
joy	2

2. Text Tokenization and Encoding

- **Tokenization**: Use **`BertTokenizer`** to tokenize and encode the text sentences. Special tokens **`[CLS]`** and **`[SEP]`** are added, and all sentences are padded or truncated to a maximum length **`MAX_LEN`**.

```
MAX_LEN = 256
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)
input_ids = [tokenizer.encode(sent, add_special_tokens=True, max_length=MAX_LEN, padding='max_length') for sent in sentences]
```

- **Example**:

```
Actual sentence: im grabbing a minute to post i feel greedy wrong
Encoded input: [101, 10047, 9775, 1037, 3371, 2000, 2695, 1045, 2514, 20505, 3308, 102, 0, 0, ..., 0]
```

3. Attention Masks

- **Creating Attention Masks**: Generate attention masks to differentiate actual tokens from padding tokens, where actual tokens are marked with `1` and padding tokens with `0`.

```
attention_masks = [[float(i > 0) for i in seq] for seq in input_ids]
```

- **Example Mask**:

```
[1.0, 1.0, ..., 1.0, 0.0, ..., 0.0]
```

4. Dataset Splitting

- **Splitting Data**: Divide the dataset into training and validation sets with a ratio of 9:1.

```
from sklearn.model_selection import train_test_split
train_inputs, validation_inputs, train_labels, validation_labels =
train_test_split(input_ids, labels, random_state=41, test_size=0.1)
train_masks, validation_masks, _, _ =
train_test_split(attention_masks, input_ids, random_state=41,
test_size=0.1)
```

- **Tensor Conversion**: Convert the data into **`torch.tensor`** format for PyTorch processing.

IV. Introduction to BERT

1. Model Overview

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained language model introduced by Google in 2018. It is based on the encoder structure of the Transformer architecture and is capable of learning text representations from both left and right contexts simultaneously. BERT has achieved state-of-the-art performance in various NLP tasks.

2. Model Features

- **Bidirectional Training**: Considers context from both directions in the text.
- **Pre-training and Fine-tuning**: Initially pre-trained on large corpora in an unsupervised manner and then fine-tuned on specific tasks.
- **Versatility**: Excels in tasks like question answering, text classification, and named entity recognition.

V. Code Technical Approach and Framework

1. Technical Approach

- **Fine-tuning a Pre-trained Model**: Utilize the pre-trained **`bert-base-uncased`** model and add a linear classification layer on top for emotion classification.
- **GPU Acceleration**: Detect the availability of a GPU using **`torch.device`** and move the model and data to the GPU to accelerate training.

```
device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
model.to(device)
```

2. Framework and Tools

- **PyTorch**: The main deep learning framework used for tensor computations and automatic differentiation.
- **Transformers**: A library by Hugging Face providing pre-trained Transformer models and tokenizers.
- **scikit-learn**: Used for data preprocessing, label encoding, and model evaluation.

3. Model Training Workflow

1. **Data Loading and Preprocessing**: Read the dataset, tokenize and encode text, generate attention masks, and split the data into training and validation sets.
2. **Creating Data Loaders**: Use `TensorDataset` and `DataLoader` to create iterable data loaders, specifying the batch size.
3. **Model Initialization**: Load the pre-trained BERT model and specify the number of output labels (6 emotion classes).
4. **Optimizer and Learning Rate Scheduler**: Use the `AdamW` optimizer with specified learning rate and epsilon parameters; employ `get_linear_schedule_with_warmup` for learning rate scheduling.
5. **Training Loop**: Iterate over multiple epochs, training the model on batches of data, computing loss, performing backpropagation, and updating model parameters.
6. **Validation Evaluation**: After each epoch, evaluate the model on the validation set, calculating metrics like accuracy and Matthews correlation coefficient (MCC).

4. GPU Acceleration Explanation

- **Device Selection**: Check for GPU availability using `torch.cuda.is_available()`.
- **Data and Model Migration**: Move the model and data to the GPU to leverage parallel computing capabilities and accelerate training.

```
model.to(device)
batch = tuple(t.to(device) for t in batch)
```

- **Multi-GPU Support**: The code detects the number of available GPUs (`n_gpu = torch.cuda.device_count()`), although the actual training uses a single GPU.

VI. Analysis of the Results

1. Training Process

- **Training Loss Reduction**: Over three epochs, the training loss decreases significantly, indicating that the model is learning and optimizing effectively.

```
Average Training loss:

Epoch 1: 0.5699966197658072
Epoch 2: 0.14640079047960985
Epoch 3: 0.09851478539460189
```

- **Learning Rate Adjustment**: A linear learning rate scheduler reduces the learning rate during training, which helps the model converge more smoothly.

```
Current Learning rate:
Epoch 1: 1.3333333333333333e-05
Epoch 2: 6.666666666666667e-06
Epoch 3: 0.0
```

2. Validation Results

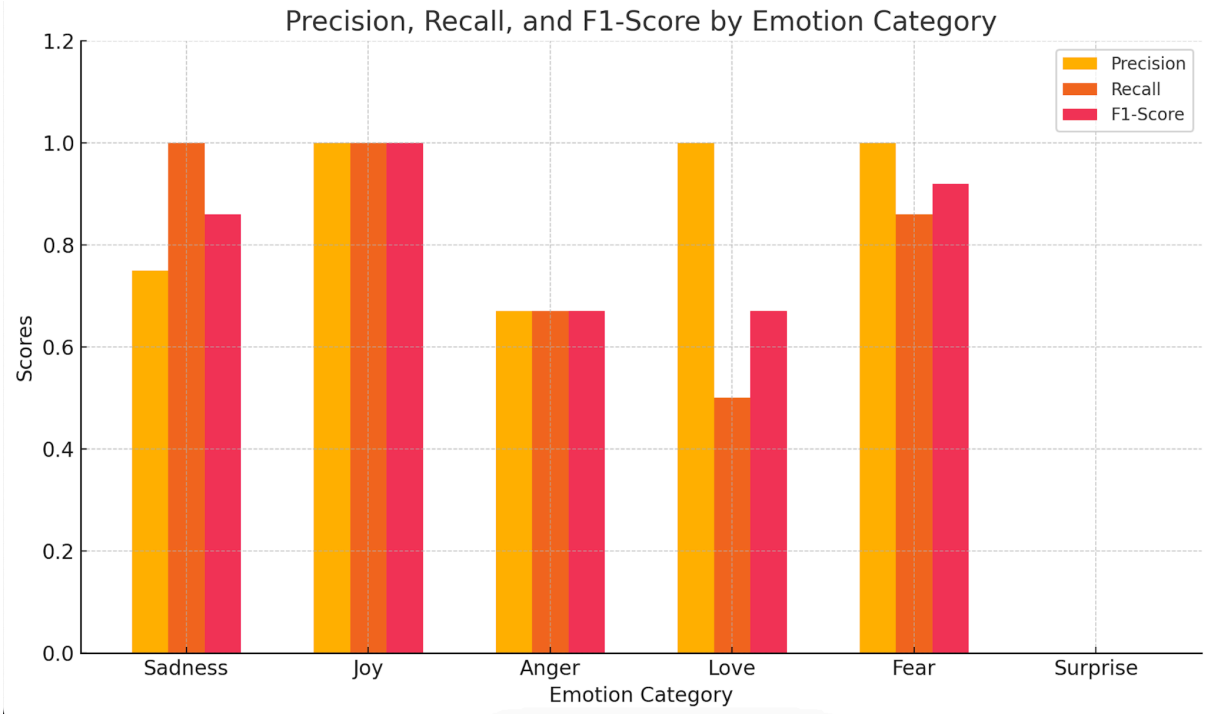
- **Accuracy and MCC**: The validation accuracy and MCC improve over the epochs but show a slight decrease in the third epoch, possibly due to overfitting or the learning rate being too low.

```
Validation Accuracy:
Epoch 1: 0.9201388888888888
Epoch 2: 0.9320436507936508
Epoch 3: 0.9295634920634921

Validation MCC Accuracy:
Epoch 1: 0.8960070016947914
Epoch 2: 0.9101417789668996
Epoch 3: 0.908430575977424
```

3. Confusion Matrix and Classification Report

- **Classification Report**: A classification report is generated for the test set, providing precision, recall, and F1-score for each class.

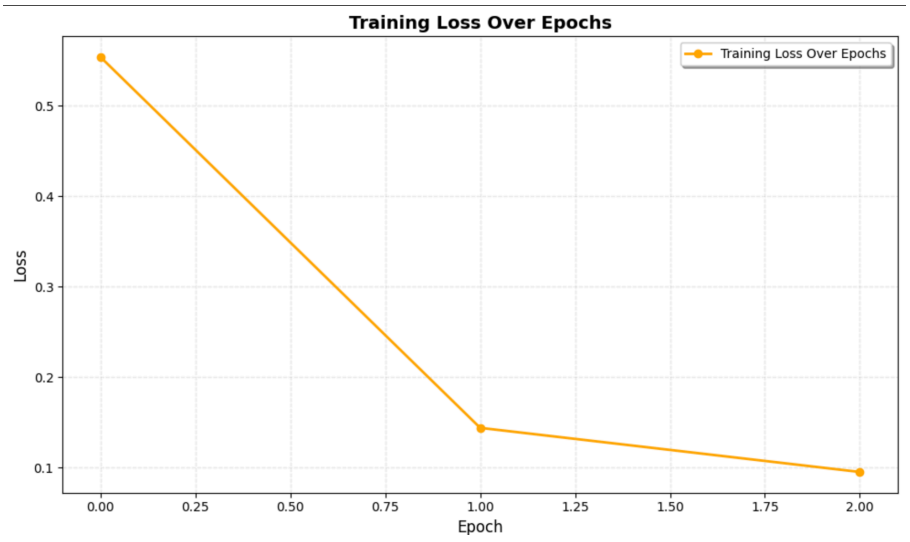


Category	Precision	Recall	F1-Score	Support
Sadness	0.75	1.0	0.86	3
Joy	1.0	1.0	1.0	1
Anger	0.67	0.67	0.67	3
Love	1.0	0.5	0.67	2
Fear	1.0	0.86	0.92	7
Surprise	0.0	0.0	0.0	0

- **Note**: There are warnings about undefined metrics for the `surprise` class due to zero samples in the validation set, highlighting the issue of data imbalance.

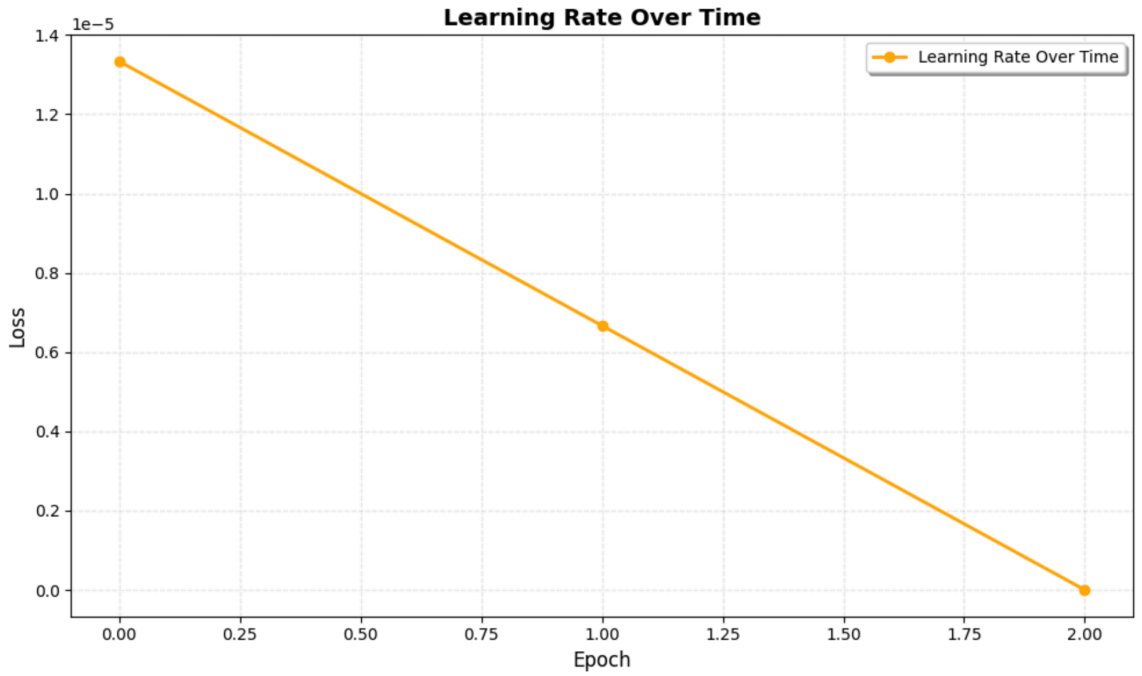
4. Loss and Learning Rate Curves

- **Training Loss Curve**:



The curve shows a clear downward trend, indicating effective model training and convergence.

- **Learning Rate Curve**:



The learning rate decreases over time according to the linear scheduler, facilitating smoother convergence.

5. Model Saving and Prediction

- **Model Saving**: The trained model and tokenizer are saved to a specified directory for future use.

```
output_dir = '/working/bert_emotion_model/'
model_to_save.save_pretrained(output_dir)
tokenizer.save_pretrained(output_dir)
```

- **Emotion Prediction Function**: A `predict_emotion` function is defined to input a text string and output the predicted emotion class and probabilities.

```
def predict_emotion(text, model, tokenizer, max_len=256):
    # Preprocessing, encoding, model inference, and probability
computation
    return predicted_class, probabilities
```

- **Example Prediction**:

```
text = "I'm feeling so blue today!"
predicted_class, probs = predict_emotion(text, model, tokenizer)
```

- **Prediction Result**:

```
Predicted Emotion: sadness
Probabilities: [0.00563996 0.00172148 0.00275413 0.00293407
0.9859531  0.00099723]
```

The model successfully predicts the emotion as `sadness`, with the highest probability, demonstrating its effective predictive capability.

VII. Conclusion

In this experiment, we utilized a pre-trained BERT model for an emotion classification task. Through data preprocessing, model construction, and training, we achieved accurate predictions of emotions from text data. The results indicate that using BERT for sentiment analysis yields high accuracy. However, issues like data imbalance and insufficient validation samples for certain classes need to be addressed in practical applications.

Future work could focus on:

- **Data Augmentation**: Increase the number of samples for minority classes to mitigate data imbalance.
- **Hyperparameter Tuning**: Adjust learning rates, batch sizes, and the number of epochs to further improve model performance.
- **Model Optimization**: Explore more advanced pre-trained models like RoBERTa or ALBERT, or incorporate sentiment lexicons and attention mechanisms to enhance classification effectiveness.

This experiment deepens our understanding of applying BERT models to sentiment analysis tasks and equips us with practical experience in data preprocessing, model training, and result evaluation, laying a solid foundation for future research and applications.