Lab 2 Homework

109062129 林奕廷

Implementation

Data Preparation

For data preparation, I extract tweet_id, identification, text, and emotion columns for the training dataset and tweet_id, identification, and text for the testing dataset.

Below is the emotion distribution in the training data:

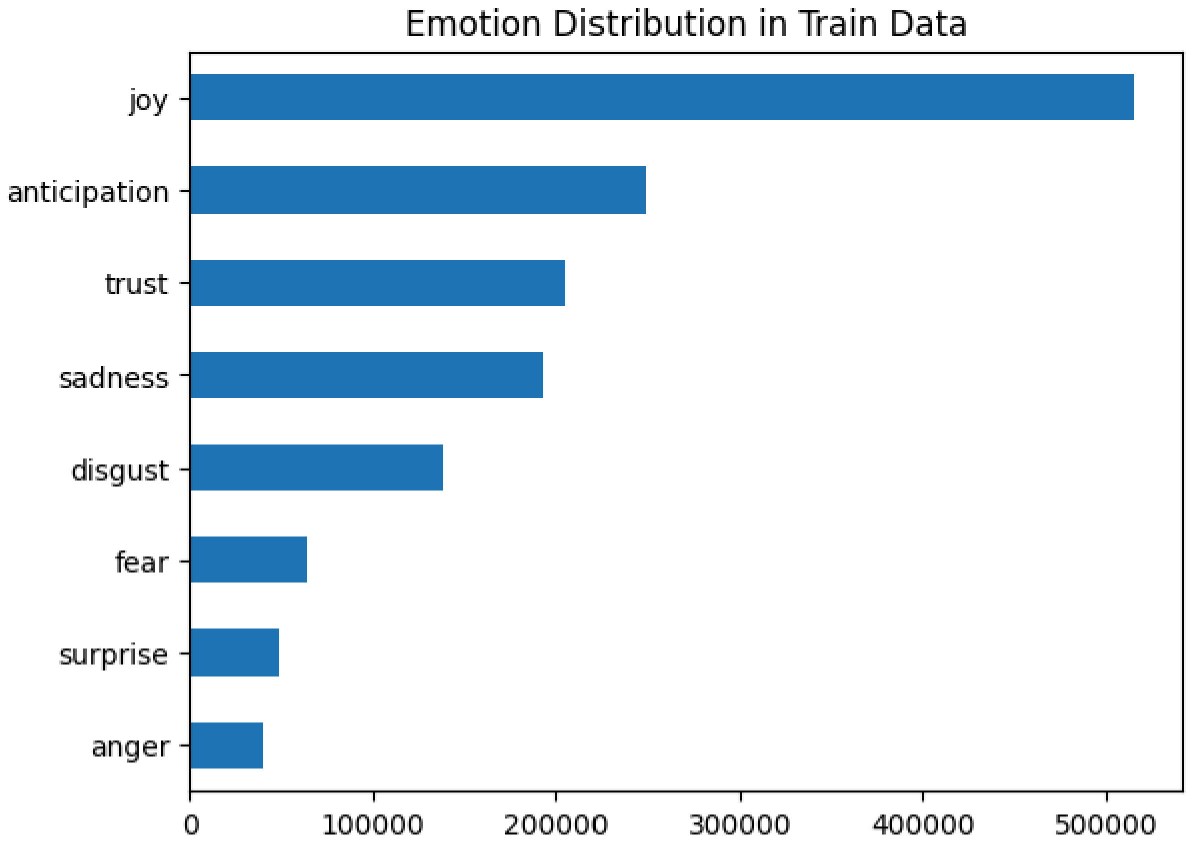


Figure 1. Emotion distribution in train data

Preprocessing

Here I use AutoTokenizer in Hugging Face Transformers to select a tokenizer that matches the pretrained model. Then I use LabelEncoder to transform the “emotion” column into integer labels.

```
tokenizer = AutoTokenizer.from_pretrained("microsoft/deberta-v3-base")
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

encoder = LabelEncoder().fit(train_df['emotion'])
num_labels = len(encoder.classes_)

# Tokenize the dataset
def tokenize(batch):
    tokenized_text = tokenizer(batch['text'], truncation=True)
    batch.update(tokenized_text)
    batch['label'] = encoder.transform(batch['emotion'])
    return batch
```

Training

I utilize AutoModelForSequenceClassification to apply a pre-trained model on Hugging Face. The class has a classification head on top of the model outputs for classification task training. With this AutoModel, we only need to set up training arguments such as epochs, batch size, fp16, etc.

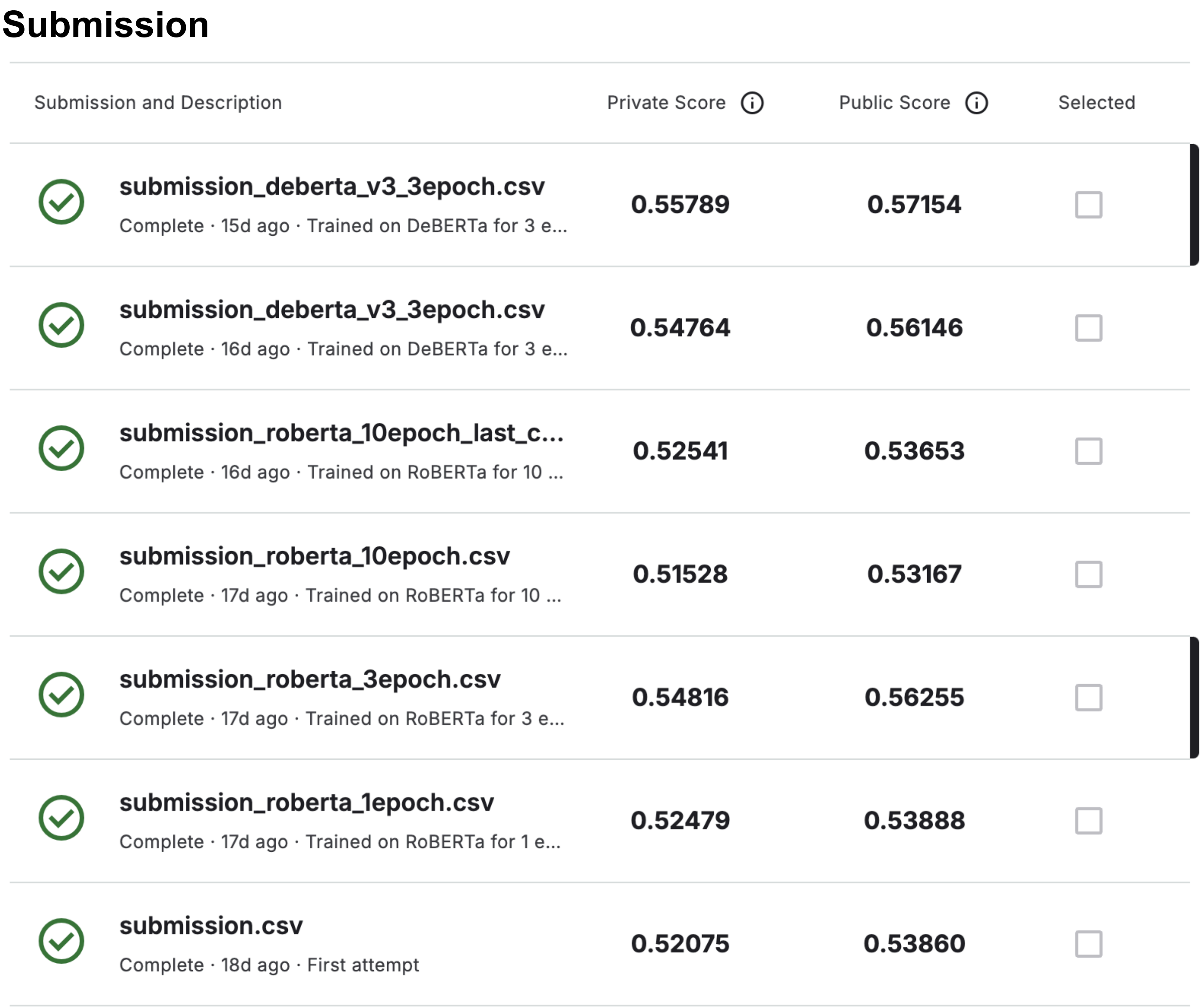
```
training_args = TrainingArguments(
    output_dir='results_deberta_v3',
    num_train_epochs=3,
    per_device_train_batch_size=32, #32
    per_device_eval_batch_size=32, #32
    eval_strategy='steps',
    eval_steps=10000, #10000
    save_strategy='steps',
    save_steps=10000, #10000
    logging_steps=1000,
    save_total_limit=5,
    load_best_model_at_end=False, #True
    fp16=True,
    torch_compile=True,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_val_dataset['train'],
    eval_dataset=train_val_dataset['test'],
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
```

Results

I train several pre-trained Transformer models available on Huggingface including BERT, RoBERTa, and DeBERTa v3 with various batch sizes and epoch settings. By utilizing the powerful BERT models and tokenizers, I obtain fairly good results without feature engineering steps, significantly reducing the efforts needed.

However, Transformer-based models require some GPU memory and take time to train. Results also show that the models tend to overfit the training dataset when train epochs > 3.

To further enhance the performance, one can search for feature engineering techniques that suit BERT models.



Submission

	Submission and Description	Private Score ⓘ	Public Score ⓘ	Selected
✓	**submission_deberta_v3_3epoch.csv** Complete · 15d ago · Trained on DeBERTa for 3 e...	**0.55789**	**0.57154**	□
✓	**submission_deberta_v3_3epoch.csv** Complete · 16d ago · Trained on DeBERTa for 3 e...	**0.54764**	**0.56146**	□
✓	**submission_roberta_10epoch_last_c...** Complete · 16d ago · Trained on RoBERTa for 10 ...	**0.52541**	**0.53653**	□
✓	**submission_roberta_10epoch.csv** Complete · 17d ago · Trained on RoBERTa for 10 ...	**0.51528**	**0.53167**	□
✓	**submission_roberta_3epoch.csv** Complete · 17d ago · Trained on RoBERTa for 3 e...	**0.54816**	**0.56255**	□
✓	**submission_roberta_1epoch.csv** Complete · 17d ago · Trained on RoBERTa for 1 e...	**0.52479**	**0.53888**	□
✓	**submission.csv** Complete · 18d ago · First attempt	**0.52075**	**0.53860**	□