Data Mining Lab2 Competition Report

113062572 張仲濡

1. Load the data

2. Covert emotion types into index

```
[ ] emo_to_id = {
            'anger' : 0,
            'anticipation' : 1,
            'disgust' : 2,
            'fear' : 3,
            'sadness' : 4,
            'surprise' : 5,
            'trust' : 6,
            'joy' : 7
    }
    id_to_emo ={
            0 : 'anger',
            1 : 'anticipation',
            2 : 'disgust',
            3 : 'fear',
            4 : 'sadness',
            5 : 'surprise',
            6 : 'trust',
            7 : 'joy'
    }

    train_data['emo_id'] = train_data['emotion'].apply(lambda x: emo_to_id[x])
```

Each emotion got an unique id.

3. Define and build Dataset

```
[ ] class Dataset(torch.utils.data.Dataset):
        def __init__(self, sequences):
            self.sequences = sequences

        def __len__(self):
            return len(self.sequences)

        def __getitem__(self, index):
            x = self.sequences.iloc[index][0]
            y = self.sequences.iloc[index][1]

            return x, y
    class TestDataset(torch.utils.data.Dataset):
        def __init__(self, sequences):
            self.sequences = sequences

        def __len__(self):
            return len(self.sequences)

        def __getitem__(self, index):
            x = self.sequences.iloc[index]
            return x
```

```
[ ] train_data_sample = train_data.sample(frac = 0.8)
    train_data_sample, test_data_sample = train_test_split(train_data_sample, test_size=0.05, random_state=42)
    ds_train = Dataset(train_data_sample[['text', 'emo_id']])
    ds_test = Dataset(train_data_sample[['text', 'emo_id']])
    ds_real_test = TestDataset(test_data['text'])
```

I take 80% of training data for training to reduce training time because I start the competition too late.

Dataset() is for training and Testdataset() is for testing.

(ds_test represent validation)

4. Collate functon and DataLoader

```
[ ] def collate_fn(batch):
        text = [data[0] for data in batch]

        text = tokenizer(text,padding = True,truncation = True,return_tensors = "pt")
        emo = torch.tensor([data[1] for data in batch],dtype = torch.long)

        return text, emo

    def collate_fn_test(batch):
        text = [data for data in batch]

        text = tokenizer(text,padding = True,truncation = True,return_tensors = "pt")

        return text
```

```
[ ] dl_train = DataLoader(dataset = ds_train, batch_size = 64, shuffle = True, collate_fn = collate_fn)
    dl_test = DataLoader(dataset = ds_test, batch_size = 64, shuffle = False, collate_fn = collate_fn)
    dl_real_test = DataLoader(dataset=ds_real_test, batch_size=64, shuffle=False, collate_fn=collate_fn_test)
```

collate_fn() is for training and collate_fn_test() is for testing.

The test data only got text.

Set batch size = 64, and shuffle the train data.

5. Build the model and layer

```
class MultiLabelModel(torch.nn.Module):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.bert = BertModel.from_pretrained("bert-base-uncased")

        self.classification = torch.nn.Linear(self.bert.config.hidden_size, 8)

        self.softmax = torch.nn.Softmax(dim=1)

    def forward(self, **kwargs):
        input_ids = kwargs.get('input_ids')
        attention_mask = kwargs.get('attention_mask')

        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)

        x = outputs.pooler_output

        emo = self.classification(x)
        emo = self.softmax(emo)

        return emo
```

```
[ ] model = MultiLabelModel().to(device)
    tokenizer = T.BertTokenizer.from_pretrained("google-bert/bert-base-uncased", cache_dir="./cache/")
```

```
[ ] epochs = 1
    optimizer = torch.optim.Adam(model.parameters(), lr=2e-5, eps=1e-8)
    classification_loss_fn = torch.nn.CrossEntropyLoss()
```

Select BERT tokenizer.

Set epoch = 1 to decrease run time.

Compute Cross Entropy Loss to solve classify problem.

6. Training

```
for ep in range(epochs):
    pbar = tqdm(dl_train)
    pbar.set_description(f"Training epoch [{ep+1}/{epochs}]")
    model.train()
    for text,emotion in pbar:
        optimizer.zero_grad()

        input_ids = text['input_ids'].to(device)
        attention_mask = text['attention_mask'].to(device)
        emo = emotion.to(device)

        emo_pred = model(input_ids=input_ids, attention_mask=attention_mask)

        loss = classification_loss_fn(emo_pred, emo)
        loss.backward()

        optimizer.step()
```

```
Training epoch [1/1]:   0%|          | 0/17210 [00:00<?, ?it/s]<ipython-input-12-3fbaa9f4b8bf
  x = self.sequences.iloc[index][0]
<ipython-input-12-3fbaa9f4b8bf>:10: FutureWarning: Series.__getitem__ treating keys as positi
  y = self.sequences.iloc[index][1]
Training epoch [1/1]: 100%|██████████| 17210/17210 [2:35:44<00:00,  1.84it/s]
```

Compute the loss between truth and predictions.

7. Turn the result back to text & Output

```
predictions = []

model.eval()
pbar = tqdm(dl_real_test)
with torch.no_grad():
    for text in pbar:
        input_ids = text['input_ids'].to(device)
        attention_mask = text['attention_mask'].to(device)


        emo_pred = model(input_ids=input_ids, attention_mask=attention_mask)

        predicted_emotion = torch.argmax(emo_pred, dim=1).tolist()
        predictions.extend([id_to_emo[x] for x in predicted_emotion])
```

```
100%|██████████| 6438/6438 [21:16<00:00, 5.04it/s]
```

```
submission = pd.DataFrame({
    'id': test_data['tweet_id'],
    'emotion': predictions[:len(test_data)]
})
```

```
submission.to_csv('/content/submission_colab2.csv', index=False)
```

8. State

final_submission.csv Complete · 3d ago	0.45953	0.47025
submission_colab2.csv Complete · 3d ago	0.46651	0.48528

I've tried (frac = 0.5,epoch = 1) , (frac = 0.8,epoch = 1), (frac = 0.5,epoch = 2), and the voting result from three of them.

(frac = 0.8,epoch = 1) has the best score among them. The voting result is the final_submission, I thought it could have the best performance at first.

I think I could get a better score by making frac = 1.

Extra. Validation & Evaluation

```
[ ] from torchmetrics import Accuracy, F1Score

    accuracy = Accuracy(task="multiclass",num_classes=8).to(device)
    f1_score = F1Score(task="multiclass",average="macro", num_classes=8).to(device)
```

Example from (frac = 0.5,epoch = 1) version:

```
for ep in range(epochs):
    pbar = tqdm(dl_test)
    pbar.set_description(f"Validation epoch [{ep+1}/{epochs}]")
    model.eval()
    with torch.no_grad():
        for text,emotion in pbar:

            input_ids = text['input_ids'].to(device)
            attention_mask = text['attention_mask'].to(device)
            emo = emotion.to(device)

            emo_pred = model(input_ids=input_ids, attention_mask=attention_mask)

            loss = classification_loss_fn(emo_pred, emo)

            accuracy.update(emo_pred, emo)
            f1_score.update(emo_pred, emo)
```

```
Validation epoch [1/1]:   0%|          | 0/2265 [00:00<?, ?it/s]<ipython-input-16-b649a17b
  x = self.sequences.iloc[index][0]
<ipython-input-16-b649a17b9bd4>:10: FutureWarning: Series.__getitem__ treating keys as pos
  y = self.sequences.iloc[index][1]
Validation epoch [1/1]: 100%|██████████| 2265/2265 [08:03<00:00,  4.68it/s]
```

```
[ ] accuracy_score = accuracy.compute()
    f1_score_val = f1_score.compute()
    print(f"Accuracy: {accuracy_score:.4f}")
    print(f"F1 Score: {f1_score_val:.4f}")

    accuracy.reset()
    f1_score.reset()
```

```
Accuracy: 0.5021
F1 Score: 0.2638
```

Predictions Visualization:

```python
pbar = tqdm(dl_test)
with torch.no_grad():
    for i,(text,emotion) in enumerate(pbar):

        input_ids = text['input_ids'].to(device)
        attention_mask = text['attention_mask'].to(device)
        emo = emotion.to(device)

        original_emotion = emo.view(-1).tolist()

        emo_pred = model(input_ids=input_ids, attention_mask=attention_mask)

        predicted_emotion = torch.argmax(emo_pred, dim=1).tolist()

        print("Batch:",i)
        print("Original emotion:",[id_to_emo[x] for x in original_emotion])
        print("Predicted emotion:",[id_to_emo[x] for x in predicted_emotion])
        print("\n")
```

```
  0%|          | 0/2265 [00:00<?, ?it/s]<ipython-input-16-b649a17b9bd4>:9: FutureW
  x = self.sequences.iloc[index][0]
<ipython-input-16-b649a17b9bd4>:10: FutureWarning: Series.__getitem__ treating key
  y = self.sequences.iloc[index][1]
  0%|          | 1/2265 [00:00<11:31,  3.28it/s]Batch: 0
Original emotion: ['anger', 'disgust', 'joy', 'joy', 'joy', 'sadness', 'joy', 'joy
Predicted emotion: ['sadness', 'sadness', 'sadness', 'joy', 'sadness', 'sadness',


  0%|          | 2/2265 [00:00<09:46,  3.86it/s]Batch: 1
Original emotion: ['joy', 'joy', 'sadness', 'anticipation', 'anticipation', 'joy',
Predicted emotion: ['anticipation', 'anticipation', 'sadness', 'anticipation', 'an


  0%|          | 3/2265 [00:00<09:18,  4.05it/s]Batch: 2
Original emotion: ['anticipation', 'joy', 'anger', 'joy', 'anticipation', 'joy', '
Predicted emotion: ['anticipation', 'joy', 'sadness', 'joy', 'joy', 'joy', 'joy',


  0%|          | 4/2265 [00:00<08:53,  4.24it/s]Batch: 3
Original emotion: ['trust', 'anticipation', 'anticipation', 'surprise', 'joy', 'sa
Predicted emotion: ['anticipation', 'anticipation', 'anticipation', 'sadness', 'jo


  0%|          | 5/2265 [00:01<08:46,  4.30it/s]Batch: 4
Original emotion: ['joy', 'sadness', 'sadness', 'joy', 'anger', 'anticipation', 'a
Predicted emotion: ['joy', 'sadness', 'sadness', 'anticipation', 'disgust', 'sadne
```