人工智慧與機器學習作業二 情緒分析

資管三B 109403533 林采璇

一、 專案連結：

BERT_ bert-base-uncased （ACC = 93.00%）

BERT_bert-large-uncased （ACC = 90.24%）

RoBERTa （ACC = 92.20%）

ALBERT （ACC = 90.80%）

ERNIE （ACC = 96.80%）［最高］

二、 基本專案過程（BERT_ bert-base-uncased）

(一) 安裝並載入所需套件

```
[ ] !pip install datasets transformers
```

```
[ ] from torch.utils.data import Dataset, DataLoader
    from transformers import AutoTokenizer
    from transformers.models.bert.modeling_bert import BertPreTrainedModel, BertModel
    from sklearn.model_selection import train_test_split
    import torch
    import torch.nn.functional as Fun
    import transformers
    import matplotlib.pyplot as plt
    import pandas as pd
    import time
    import warnings
    warnings.filterwarnings('ignore') # setting ignore as a parameter
```

(二) 一些模型會用到的小函數（todo1、todo2）

- get_pred：從 logits 的 dimension=1 去取得結果中數值最高者當做預測結果。
- cal_metrics：透過 detach() 和 cpu()將 tensor 轉為 NumPy，再透過使用 sklearn 的套件算出 acc, f1, recall 及 precision，以矩陣形式回傳。
- save_checkpoints、load_checkoints：分別用於儲存和載入訓練好的模型。

```
[ ] # get predict result
    def get_pred(logits):
      # todo #
      return logits.argmax(dim=1)

    # calculate confusion metrics
    from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
    def cal_metrics(pred, ans):
      # todo #
      pred = pred.detach().cpu().numpy()
      ans = ans.detach().cpu().numpy()

      acc = accuracy_score(ans, pred)
      f1 = f1_score(ans, pred, average='macro')
      recall = recall_score(ans, pred, average='macro')
      precision = precision_score(ans, pred, average='macro')
      metrics = [acc, f1, recall, precision]
      return metrics
```

```
[ ] # save model to path
    def save_checkpoint(save_path, model):
      if save_path == None:
          return
      torch.save(model.state_dict(), save_path)
      print(f'Model saved to ==> {save_path}')

    # load model from path
    def load_checkpoint(load_path, model, device):
      if load_path==None:
          return
      state_dict = torch.load(load_path, map_location=device)
      print(f'Model loaded from <== {load_path}')

      model.load_state_dict(state_dict)
      return model
```

(三) 載入並處理資料(todo3)

載入資料集， 並只使用 train_data 和 test_data (完整資料集有 train, test, unsupervised)。重新切割完資料成 train, val, test， 儲存資料。

```
[ ] from datasets import load_dataset

    dataset = load_dataset("imdb")
```

```
[ ] import pandas as pd
    # todo #
    all_data = [] # a list to save all data
    train_data = pd.DataFrame(dataset['train'])
    test_data = pd.DataFrame(dataset['test'])
    all_df = pd.concat([train_data, test_data], ignore_index=True)
    all_df.head()
```

```
[ ] from sklearn.model_selection import train_test_split

    train_df, temp_data = train_test_split(all_df, random_state=1111, train_size=0.8)
    dev_df, test_df = train_test_split(temp_data, random_state=1111, train_size=0.5)
    print('# of train_df:', len(train_df))
    print('# of dev_df:', len(dev_df))
    print('# of test_df data:', len(test_df))

    # save data
    train_df.to_csv('./train.tsv', sep='\t', index=False)
    dev_df.to_csv('./val.tsv', sep='\t', index=False)
    test_df.to_csv('./test.tsv', sep='\t', index=False)
```

(四) Tokenize(todo4)

- __init__ ： 初始 CustomDataset class 的建構方法。
- __len__：回傳長度。
- one_hot_label：將 label 轉換成 one-hot encoding。
- tokenize：將傳入的文句， 用 AutoTokenizer 去 tokenize， 並回傳 input ids, attention mask, token type ids 三個 tensor。

```
[ ] import torch
    from torch.utils.data import Dataset, DataLoader
    from transformers import AutoTokenizer
    import torch
    import torch.nn.functional as Fun

    # Using Dataset to build DataLoader
    class CustomDataset(Dataset):
        def __init__(self, mode, df, specify, args):
            assert mode in ["train", "val", "test"]  # 一般會切三份
            self.mode = mode
            self.df = df
            self.specify = specify # specify column of data (the column U use for predict)
            if self.mode != 'test':
                self.label = df['label']
            self.tokenizer = AutoTokenizer.from_pretrained(args["config"])
            self.max_len = args["max_len"]
            self.num_class = args["num_class"]

        def __len__(self):
            return len(self.df)
```

```
[ ]     # transform label to one_hot label (if num_class > 2)
        def one_hot_label(self, label):
            return Fun.one_hot(torch.tensor(label), num_classes = self.num_class)

        # transform text to its number
        def tokenize(self, input_text):
            # todo #
            # 使用tokenizer將文本轉為Bert的輸入格式
            encoded_dict = self.tokenizer.encode_plus(
                    input_text,                              # 需要轉換的文本
                    add_special_tokens=True,                 # 添加Special Token
                    max_length=self.max_len,                 # 設定最大長度
                    # pad_to_max_length=True,                # 不足最大長度的用padding補齊
                    padding="max_length",
                    # return_attention_mask=True,          # 創建Attention Mask
                    return_token_type_ids=True,            # 創建Token Type Ids
                    truncation=True,
                    )
            input_ids = encoded_dict['input_ids']
            attention_mask = encoded_dict['attention_mask']
            token_type_ids = encoded_dict['token_type_ids']

            return input_ids, attention_mask, token_type_ids
```

- 先透過 index 獲取文句並 tokenize， 再依照使用之用途(testing, training or validation)回傳不同的內容。

```
[ ]   def __getitem__(self, index):

          sentence = str(self.df[self.specify][index])
          ids, mask, token_type_ids = self.tokenize(sentence)


          if self.mode == "test":
                  return torch.tensor(ids, dtype=torch.long), torch.tensor(mask, dtype=torch.long), \
                      torch.tensor(token_type_ids, dtype=torch.long)
          else:
                  if self.num_class > 2:
                      return torch.tensor(ids, dtype=torch.long), torch.tensor(mask, dtype=torch.long), \
                          torch.tensor(token_type_ids, dtype=torch.long), self.one_hot_label(self.label[index])
                  else:
                      return torch.tensor(ids, dtype=torch.long), torch.tensor(mask, dtype=torch.long), \
                          torch.tensor(token_type_ids, dtype=torch.long), torch.tensor(self.label[index], dtype=torch.long)
```

(五) 建立模型(todo5)

- __init__：繼承 BertPreTrainedModel，並加上 dropout, linear layer。
- forward：把 tensors (input_ids, attention_mask, and token_type_ids)放進對應層數（bert -> dropout -> classifier)，以取得 logits 並回傳。

```
[ ] # BERT Model
    class BertClassifier(BertPreTrainedModel):
      def __init__(self, config, args):
        super(BertClassifier, self).__init__(config)
        self.bert = BertModel(config)
        # todo #
        self.dropout = torch.nn.Dropout(args["dropout"])
        self.classifier = torch.nn.Linear(config.hidden_size, args["num_class"])
        self.init_weights()

      # forward function, data in model will do this
      def forward(self, input_ids=None, attention_mask=None, token_type_ids=None, position_ids=None,
                  head_mask=None, inputs_embeds=None, labels=None, output_attentions=None,
                  output_hidden_states=None, return_dict=None):
        # todo #
        bert_output = self.bert(input_ids=input_ids, attention_mask=attention_mask, token_type_ids=token_type_ids,
                                position_ids=position_ids, head_mask=head_mask, inputs_embeds=inputs_embeds,
                                output_attentions=output_attentions, output_hidden_states=output_hidden_states,
                                return_dict=return_dict)
        pooled_output = bert_output.pooler_output  # (batch_size, hidden_size)
        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)
        return logits
```

(六) 訓練模型

1. 設定訓練參數：說實話，沒有甚麼訣竅，只能看訓練狀況慢慢調整。如果說這之中有甚麼轉機的話，就是把 learning rate 從 le-4 改為 le-5，不然原本的訓練狀況，acc 只有在 50%左右浮動。

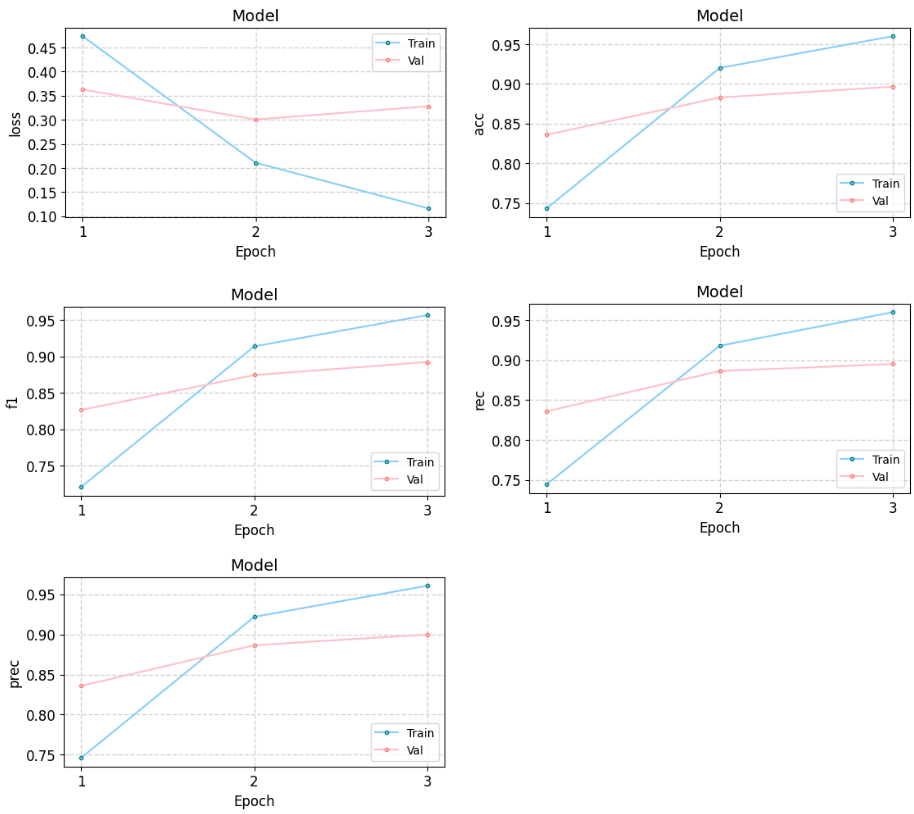
```
[ ] from datetime import datetime
    parameters = {
        "num_class": 2,
        "time": str(datetime.now()).replace(" ", "_"),
        # Hyperparameters
        "model_name": 'BERT',
        "config": 'bert-base-uncased',
        "learning_rate": 1e-5,
        "epochs": 3,
        "max_len": 256,
        "batch_size": 16,
        "dropout": 0.4,
    }
```

2. 開始訓練(todo6)：從多次輸出的訓練狀況來看，通常大於 epoch3 就會 overfitting 了。

```
[ ] # Start training
    import time

    metrics = ['loss', 'acc', 'f1', 'rec', 'prec']
    mode = ['train_', 'val_']
    record = {s+m :[] for s in mode for m in metrics}

    for epoch in range(parameters["epochs"]):

        st_time = time.time()
        train_loss, train_acc, train_f1, train_rec, train_prec = 0.0, 0.0, 0.0, 0.0, 0.0
        step_count = 0

        # todo #
        model.train()
        for data in train_loader:
          ids, masks, token_type_ids, labels = [target.to(device) for target in data]

          optimizer.zero_grad()
          logits = model(input_ids = ids,
                  token_type_ids = token_type_ids,
                  attention_mask = masks)
          loss = loss_fct(logits, labels)
          loss.backward() # compute gradients
          optimizer.step() # update model parameters

          # update metrics
          acc, f1, rec, prec = cal_metrics(get_pred(logits), labels)
          train_loss += loss.item()
          train_acc += acc
          train_f1 += f1
          train_rec += rec
          train_prec += prec
          step_count += 1


        # evaluate the model performace on val data after finishing an epoch training
        val_loss, val_acc, val_f1, val_rec, val_prec = evaluate(model, val_loader, device)

        train_loss = train_loss / step_count
        train_acc = train_acc / step_count
        train_f1 = train_f1 / step_count
        train_rec = train_rec / step_count
        train_prec = train_prec / step_count
        print('[epoch %d] cost time: %.4f s'%(epoch + 1, time.time() - st_time))
        print('         loss     acc     f1      rec     prec')
        print('train | %.4f, %.4f, %.4f, %.4f, %.4f'%(train_loss, train_acc, train_f1, train_rec, train_prec))
        print('val   | %.4f, %.4f, %.4f, %.4f, %.4f\n'%(val_loss, val_acc, val_f1, val_rec, val_prec))

        # record training metrics of each training epoch
        record['train_loss'].append(train_loss)
        record['train_acc'].append(train_acc)
        record['train_f1'].append(train_f1)
        record['train_rec'].append(train_rec)
        record['train_prec'].append(train_prec)

        record['val_loss'].append(val_loss)
        record['val_acc'].append(val_acc)
        record['val_f1'].append(val_f1)
        record['val_rec'].append(val_rec)
        record['val_prec'].append(val_prec)
```

```
[epoch 1] cost time: 177.4896 s
         loss     acc     f1      rec     prec
train | 0.4736, 0.7432, 0.7211, 0.7445, 0.7460
val   | 0.3631, 0.8359, 0.8267, 0.8359, 0.8358

[epoch 2] cost time: 176.2199 s
         loss     acc     f1      rec     prec
train | 0.2109, 0.9200, 0.9137, 0.9180, 0.9219
val   | 0.3006, 0.8828, 0.8744, 0.8864, 0.8865

[epoch 3] cost time: 176.3526 s
         loss     acc     f1      rec     prec
train | 0.1161, 0.9600, 0.9566, 0.9600, 0.9608
val   | 0.3282, 0.8965, 0.8923, 0.8952, 0.8999
```

- 繪圖



(七) 預測

```
[ ] def Softmax(x):
      return torch.exp(x) / torch.exp(x).sum()
    # label to class
    def label2class(label):
      l2c = {0:'negative', 1:'positive'}
      return l2c[label]
```

1. 預測單筆(todo7)：完成預測單筆的程式，結果也如預期是 negative。

```
[ ] # predict single sentence, return each-class's probability and predicted class
    def predict_one(query, model):

        # todo #
        tokenizer = AutoTokenizer.from_pretrained(parameters['config'])

        encoded_query = tokenizer.encode_plus(query, max_length=256, padding='max_length', truncation=True, return_tensors='pt')
        input_ids = encoded_query['input_ids'].to(device)
        attention_mask = encoded_query['attention_mask'].to(device)
        token_type_ids = encoded_query['token_type_ids'].to(device)

        # forward pass
        with torch.no_grad():
            logits = model(input_ids, attention_mask, token_type_ids)
            probs = Softmax(logits) # get each class-probs
            label_index = torch.argmax(probs[0], dim=0)
            pred = label_index.item()

        return probs, pred
```

```
[ ] # you can load model from existing result
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    init_model = BertClassifier.from_pretrained(parameters['config'], parameters) # build an initial model
    model = load_checkpoint('./bert.pt', init_model, device).to(device) # and load the weight of model from specify file
```

```
[ ]  %%time
     probs, pred = predict_one("This movie doesn't attract me", model)
     print(label2class(pred))
```

```
negative
CPU times: user 60.6 ms, sys: 4.02 ms, total: 64.6 ms
Wall time: 292 ms
```

2. 所有測試集

```
[ ]  # predict dataloader
     def predict(data_loader, model):

       tokenizer = AutoTokenizer.from_pretrained(parameters['config'])
       device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

       total_probs, total_pred = [], []
       model.eval()
       with torch.no_grad():
         for data in data_loader:
           input_ids, attention_mask, \
           token_type_ids = [t.to(device) for t in data]

           # forward pass
           logits = model(input_ids, attention_mask, token_type_ids)
           probs = Softmax(logits) # get each class-probs
           label_index = torch.argmax(probs[0], dim=0)
           pred = label_index.item()

           total_probs.append(probs)
           total_pred.append(pred)

       return total_probs, total_pred
```

```
[ ]  # load testing data
     test_df = pd.read_csv('./test.tsv', sep = '\t').sample(5000).reset_index(drop=True)
     test_dataset = CustomDataset('test', test_df, 'text', parameters)
     test_loader = DataLoader(test_dataset, batch_size=1, shuffle=False)

     total_probs, total_pred = predict(test_loader, model)
     res = test_df.copy()
     # add predict class of origin file
     res['pred'] = total_pred

     # save result
     res.to_csv('./result.tsv', sep='\t', index=False)
```

(八) 結果(acc = 93%)

```
[ ]  correct = 0
     for idx, pred in enumerate(res['pred']):
       if pred == res['label'][idx]:
         correct += 1
     print('test accuracy = %.4f'%(correct/len(test_df)))
```

```
test accuracy = 0.9300
```

三、　替換模型

替換模型後的實作流程都與最初版本的 BERT_ bert-base-uncased 大同小異，主要是將 import 的 pre-trained 模型更改後，調整模型建構的參數值以及訓練用的參數設定。

1. BERT_bert-large-uncased (acc = 90.24%)：

其實從 BERT 的官方說明文件中，就會發現，BERT based 底下的模型很多，所以將 config 從 bert-base-uncased 改為 bert-large-uncased。

- bert-base-uncased：12 layers, 110 million parameters
- bert-large-uncased：24 layers, 340 million parameters

Model	#params	Language
bert-base-uncased	110M	English
bert-large-uncased	340M	English
bert-base-cased	110M	English
bert-large-cased	340M	English
bert-base-chinese	110M	Chinese
bert-base-multilingual-cased	110M	Multiple
bert-large-uncased-whole-word-masking	340M	English
bert-large-cased-whole-word-masking	340M	English

我使用了與 bert-base-uncased 相同的訓練參數值，結果好像有點 overfitting 訓練結果比較差一點。

```
[ ] from datetime import datetime
    parameters = {
            "num_class": 2,
            "time": str(datetime.now()).replace(" ", "_"),
            # Hyperparameters
            "model_name": 'BERT',
            "config": 'bert-base-uncased',
            "learning_rate": 1e-5,
            "epochs": 3,
            "max_len": 256,
            "batch_size": 16,
            "dropout": 0.4,
    }
```

```
[ ] correct = 0
    for idx, pred in enumerate(res['pred']):
      if pred == res['label'][idx]:
        correct += 1
    print('test accuracy = %.4f'%(correct/len(test_df)))

    test accuracy = 0.9024
```

2. RoBERTa (acc = 92.20%)：

RoBERTa (Robustly optimized BERT approach)，相關資料說是 BERT 的改良版，使用更多樣化的資料集進行訓練，並使用更多的資料擴充，在情感分析等 NLP 中有不錯的表現。但我自己可能參數沒有設的很好，導致訓練狀況跟原本的 BERT 差不多。

```
[ ] correct = 0
    for idx, pred in enumerate(res['pred']):
      if pred == res['label'][idx]:
        correct += 1
    print('test accuracy = %.4f'%(correct/len(test_df)))

    test accuracy = 0.9220
```

3. ALBERT (acc = 90.80%)

ALBERT (A Lite BERT)，是輕量版的 BERT，透過參數共享、矩陣分解等技術減少模型參數，因此訓練速度極快準確率也高。

```
[ ] correct = 0
    for idx, pred in enumerate(res['pred']):
      if pred == res['label'][idx]:
        correct += 1
    print('test accuracy = %.4f'%(correct/len(test_df)))

    test accuracy = 0.9080
```

4. ERNIE (acc=96.80%) (最高)

前面訓練太多 BERT 的衍伸版本，結果訓練狀況差不多，就換著找其他可能還不錯的模型。ERNIE 在網路上有看到報導說是超越 BERT 的模型，但還算是有意外之喜，畢竟官方文件說它特別在中文的訓練上優秀，沒想到英文的也不錯。

Model Name	Language	Description
ernie-1.0-base-zh	Chinese	Layer:12, Heads:12, Hidden:768
ernie-2.0-base-en	English	Layer:12, Heads:12, Hidden:768
ernie-2.0-large-en	English	Layer:24, Heads:16, Hidden:1024
ernie-3.0-base-zh	Chinese	Layer:12, Heads:12, Hidden:768
ernie-3.0-medium-zh	Chinese	Layer:6, Heads:12, Hidden:768
ernie-3.0-mini-zh	Chinese	Layer:6, Heads:12, Hidden:384
ernie-3.0-micro-zh	Chinese	Layer:4, Heads:12, Hidden:384
ernie-3.0-nano-zh	Chinese	Layer:4, Heads:12, Hidden:312
ernie-health-zh	Chinese	Layer:12, Heads:12, Hidden:768
ernie-gram-zh	Chinese	Layer:12, Heads:12, Hidden:768

我選了 ernie-2.0-base-en，是唯二英文版本中，層數比較多的。ERNIE 相對於其他 pre-trained 模型訓練的時長特別長，兩個 epoch 跑 40 分鐘。

```
[ ] correct = 0
    for idx, pred in enumerate(res['pred']):
      if pred == res['label'][idx]:
        correct += 1
    print('test accuracy = %.4f'%(correct/len(test_df)))

    test accuracy = 0.9680
```

四、 綜合結果：以我自己的訓練狀況來看，ERNIE 的訓練狀況是最好的，但也不排除其他模型沒有抓到訓練訣竅，導致錯失訓練得更好的可能性。

```
[ ] correct = 0
    for idx, pred in enumerate(res['pred']):
      if pred == res['label'][idx]:
        correct += 1
    print('test accuracy = %.4f'%(correct/len(test_df)))
```

```
test accuracy = 0.9680
```

五、 心得：

這次的專案，嘗試了不同的 Transformer Pretrained Model 深刻感受到每個 Model 都非常有各自的特性需要做參數上的設定與嘗試，還有程式碼的除錯。但同時因為是用 Pre-Trained Model，相對於上次作業自己從頭 Train 而言，所需要的 Epoch 數大大減少，整體的準確度也提升，感受到其使用上的便利性。

六、 References：

https://huggingface.co/docs/transformers/index
https://huggingface.co/docs/transformers/model_doc/bert
https://huggingface.co/docs/transformers/model_doc/roberta
https://huggingface.co/docs/transformers/model_doc/albert
https://huggingface.co/docs/transformers/model_doc/ernie

[論文整理] RoBERTa
[論文整理] ALBERT
BERT vs ERNIE: The Natural Language Processing Revolution
Baidu’s ERNIE 2.0 Beats BERT and XLNet on NLP Benchmarks