人工智慧作業二

-

情緒分析 心得報告

資管三 B　109403048 林子恩

目錄

Colab 連結(1.bert-base-uncased 2.bert-large-uncased 3. nghuyong/ernie-2.0-large-en)

1.https://colab.research.google.com/drive/1pR86kzSr_rUK6RXyjzBzqW36NGiF6prs?usp=sharing

2.https://colab.research.google.com/drive/1ceaMyIsbmV9njQQwsTqlqBye0mzJs9Ly?usp=sharing

3. https://colab.research.google.com/drive/12LjHScdd_Ad-EkiCzwkfcK0mgcYP1F2-?usp=sharing

ToDo1~7

- TODO1: 完成get_pred()

  > 從logits的dimension=1去取得結果中數值最高者當做預測結果

- TODO2: 完成cal_metrics()

  > 透過將tensor轉為numpy，可使用sklearn的套件算出acc, f1_score, recall及precision

```
# get predict result
def get_pred(logits):
    ##########
    # todo #
    pred = torch.argmax(logits, dim=1)
    return pred
    ##########
```

```
# calculate confusion metrics
def cal_metrics(pred, ans, average='macro'):
    ##########
    # todo #
    pred = pred.cpu().numpy()
    ans = ans.cpu().numpy()
    acc = accuracy_score(ans, pred)
    f1 = f1_score(ans, pred, average=average)
    recall = recall_score(ans, pred, average=average)
    precision = precision_score(ans, pred, average=average)
    metrics = [acc, f1, recall, precision]
    return metrics
    ##########
```

在 cal_metrics，先將 tensor 格式的 pred,ans 轉為 numpy，因為 scikit-learn 函式庫只接受 numpy 格式使用 average 指定為 macro ，他可以使計算 acc,f1,recall,precision 的計算方式為每個類別的平均，最後將他們指定為 metrics array 格式回傳

TODO3:把資料拿出來後，將train及test合併，重新切割後，儲存下來。

```
import pandas as pd

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

TODO4: 完成tokenize()

```
    def tokenize(self,input_text):
        ###########
        # todo #
        encoding = self.tokenizer.encode_plus(
                input_text,
                add_special_tokens=True,     #更好理解文本
                max_length=self.max_len,
                padding="max_length",
                return_attention_mask=True,
                return_token_type_ids=True,
                truncation=True,
        )
        return encoding["input_ids"], encoding["attention_mask"], encoding["token_type_ids"]
        ###########
```

使用 transformer 中的 encode_plus 函式，之後存在 encoding 字典中，

add_special_token 設為 true，將會有 token 用來表示一段文本輸入的開始跟結束。

max_len 採用如果文本不夠用 padding 補齊的方式

後面的預設都為 true 但還是寫上去比較有提醒的感覺。

TODO5: 完成BertClassifier

> 在初始化的地方加上dropout, linear layer（等於一層NN），其維度為類別數量；在forward function中把輸入值放進對應層數（bert -> dropout -> classifier）；請注意我們只取用bert輸出的sentence representation去做分類

```
# 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 #
        outputs = self.bert(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)
        # get only sentence representation
        pooled_output = outputs.pooler_output

        # put pooled_output into dropout and classifier
        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)
        return logits
        ##########
```

使用 pretrained model，將前面 cal_metrics 傳出的參數導入模型中，獲得 ouputs，之後使用最後一層的輸出(pooled_ouput)，導入 dropout 層，最後將特徵轉為預測結果，傳回 logits

TODO6: 完成訓練，可參照evaluate()並稍作調整以完成訓練

```
##########
#  todo  #
model.train()  #  set  model  to  training  mode

for  data  in  train_loader:
        ids,  masks,  token_type_ids,  labels  =  [t.to(device)  for  t  in  data]

        optimizer.zero_grad()  #  zero  out  the  gradients  from  the  previous  iteration

        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

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

用梯度下降法，每次訓練周期結束，將紀錄那些指標

TODO7: 完成predict_one()

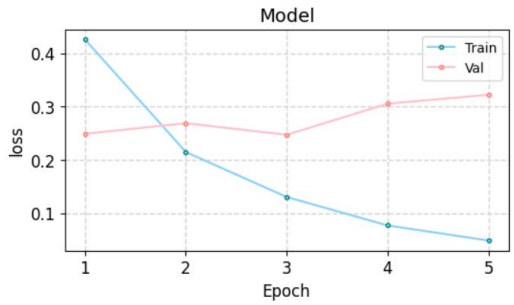
```
def  predict_one(query,  model):

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


       encoded_query  =  tokenizer.encode_plus(query,  max_length=parameters['max_len'],  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
```

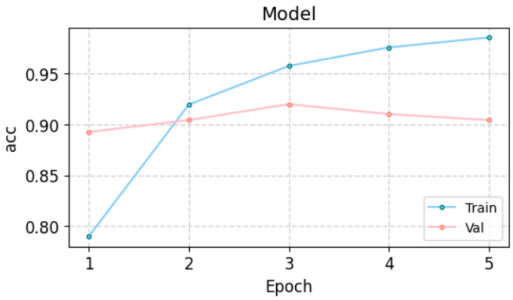
用 Softmax 取得每個 num_classes 的機率分布，再用 torch.argmx 預測最有可能的類別

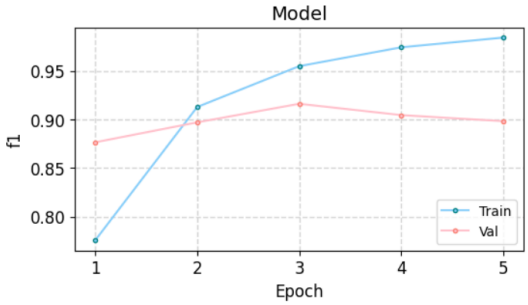
Bert-base-uncased 執行結果

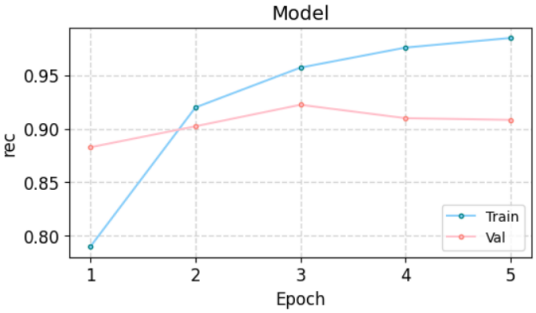
```
from  datetime  import  datetime
parameters  =  {
        "num_class":  2,
        "time":  str(datetime.now()).replace("  ",  "_"),
        #  Hyperparameters
        "model_name":  'BERT',
        "config":  'bert-base-uncased',
        "learning_rate":  1e-5,
        "epochs":  5,
        "max_len":  256,#發現最長到13000多
        "batch_size":  16,
        "dropout":  0.5,
}
```

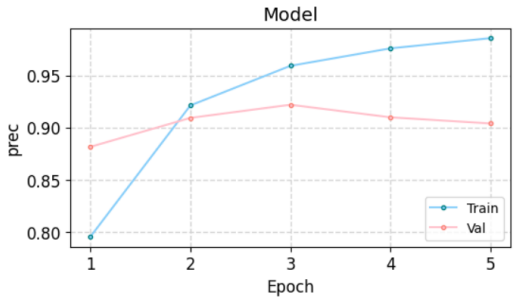
```
[epoch 1] cost time: 185.8135 s
         loss     acc     f1      rec     prec
train | 0.4262, 0.7897, 0.7758, 0.7897, 0.7954
val   | 0.2488, 0.8926, 0.8765, 0.8827, 0.8816

[epoch 2] cost time: 183.5180 s
         loss     acc     f1      rec     prec
train | 0.2146, 0.9195, 0.9128, 0.9199, 0.9212
val   | 0.2688, 0.9043, 0.8971, 0.9023, 0.9093

[epoch 3] cost time: 184.1551 s
         loss     acc     f1      rec     prec
train | 0.1301, 0.9575, 0.9548, 0.9572, 0.9591
val   | 0.2470, 0.9199, 0.9161, 0.9224, 0.9219

[epoch 4] cost time: 183.6639 s
         loss     acc     f1      rec     prec
train | 0.0767, 0.9758, 0.9742, 0.9760, 0.9758
val   | 0.3054, 0.9102, 0.9045, 0.9099, 0.9098

[epoch 5] cost time: 183.4397 s
         loss     acc     f1      rec     prec
train | 0.0483, 0.9855, 0.9842, 0.9850, 0.9856
val   | 0.3222, 0.9043, 0.8983, 0.9083, 0.9039
```












```
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 = 0.9000
```

最後是 90%整

learning_rate	epochs	max_len	batch_size	dropout	acc
1e-4	3	256	16	0.5	83%
1e-4	3	512	16	0.3	49%
1e-4	3	256	16	0.5	85%
1e-5	3	256	16	0.5	89%
1e-5	5	256	16	0.5	90%
1e-5	5	512	16	0.6	88%
1e-5	5	256	16	0.5	90%

最後就大概 90 上下了，為尋求更高的準確率，使用兩個另外的預訓練模型

加分題

挑了兩個分別是 bert 底下的 bert-large-uncased，與 bert-base-uncased 差在使用更多的層數、hidden units、參數，他可以處理更複雜的文本，與之相對的他的耗時也更多，但是也可更好的學習；另外的是 nghuyong/ernie-2.0-large-en，根據我查到的資料 ernie 是基於 bert 的架構作優化，使用了英中混雜方式訓練，之後分割出英文版本，也就是這次我使用的預訓練模型。

加分題訓練過程

其實都不太複雜，原本挑 roberta 但之後太多參數，方法要調整，跟同學討論聽到 roberta 結果也不如預期，之後就放棄改這兩個。

Bert-large-uncased

```
from datetime import datetime
parameters = {
        "num_class": 2,
        "time": str(datetime.now()).replace(" ", "_"),
        # Hyperparameters
        "model_name": 'BERT',
        "config": 'bert-large-uncased',
        "learning_rate": 1e-5,
        "weight_decay": 1e-5,
        "epochs": 3,
        "max_len": 256,#發現最長到13000多
        "batch_size": 16,
        "dropout": 0.5,
}
```

```
[epoch 1] cost time: 581.8849 s
          loss     acc      f1      rec     prec
train | 0.3990, 0.8157, 0.8022, 0.8150, 0.8186
val   | 0.2577, 0.9043, 0.8950, 0.9032, 0.9065

[epoch 2] cost time: 581.9261 s
          loss     acc      f1      rec     prec
train | 0.1905, 0.9320, 0.9268, 0.9319, 0.9338
val   | 0.2475, 0.9062, 0.9026, 0.9112, 0.9126

[epoch 3] cost time: 581.3829 s
          loss     acc      f1      rec     prec
train | 0.0988, 0.9710, 0.9690, 0.9711, 0.9712
val   | 0.2771, 0.9102, 0.8966, 0.9057, 0.8979
```

```
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 = 0.9118
```

最後最高差不多是 91 上下，確實是有高了一點，但時間也花了 3 倍左右

nghuyong/ernie-2.0-large-en

```
from datetime import datetime
parameters = {
        "num_class": 2,
        "time": str(datetime.now()).replace(" ", "_"),
        # Hyperparameters
        "model_name": 'ERNIE',
        "config": 'nghuyong/ernie-2.0-large-en',
        "learning_rate": 1e-6,
        "epochs": 3,
        "max_len": 256,#發現最長到13000多
        "batch_size": 16,
        "dropout": 0.3,
}
```

```
[epoch 1] cost time: 582.0013 s
         loss     acc     f1      rec     prec
train | 0.4418, 0.7775, 0.7589, 0.7766, 0.7747
val   | 0.1953, 0.9336, 0.9290, 0.9371, 0.9360

[epoch 2] cost time: 582.2762 s
         loss     acc     f1      rec     prec
train | 0.1886, 0.9363, 0.9313, 0.9351, 0.9363
val   | 0.1410, 0.9453, 0.9418, 0.9477, 0.9436

[epoch 3] cost time: 582.2693 s
         loss     acc     f1      rec     prec
train | 0.1434, 0.9545, 0.9513, 0.9532, 0.9564
val   | 0.1383, 0.9492, 0.9462, 0.9503, 0.9499
```

```
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 = 0.9414
```

最後最佳為 94%
這個模型是了非常多次，但都在 93%上下，有時候會 overfitting，最佳的這次訓練也沒有到非常收斂，但提高 epoch，又會 overfitting，增高 dropout 也不如預期，減少學習率，反而收斂太慢。

結論心得

個人覺得這次作業相較上一次，難度上升很多，除了時間不用跑那麼久以外，限制蠻多，像我用 colab 寫，發現只要 max_len 設 512，非常容易爆掉，batch_size 更是無法設超過 16，模型也有很多選擇。

最後方便助教算分，我 todo7 個都有寫出來，做了加分題，最高的準確率為 94.14%