作業 2 – 情緒分析

資管三_109403502_楊珮綾

- **colab 連結：**

  BERT 模型：https://colab.research.google.com/drive/18CAjCF20UEV-17r1Du5PMYzf3SPZFD0N?usp=share_link

  DISTILBERT 模型：https://colab.research.google.com/drive/1Oe0udXG6OXwxmbG3RaLtfQpuYz_gNoPn?usp=share_link

  ALBERT 模型：https://colab.research.google.com/drive/1TyYxsUoMdrX15gr1whHNqZiExlqZqF3I?usp=share_link

- **Test accuracy:**

  BERT 模型：0.9220

  ```
  test accuracy = 0.9220
  ```

  DISTILBERT 模型：0.8360

  ```
  test accuracy = 0.8360
  ```

  ALBERT 模型：0.8940

  ```
  test accuracy = 0.8940
  ```

- **撰寫過程與截圖:**

  一、安裝與載入所需套件

```
!pip install datasets transformers
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting datasets
  Downloading datasets-2.11.0-py3-none-any.whl (468 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 468.7/468.7 kB 18.8 MB/s eta 0:00:00
Collecting transformers
  Downloading transformers-4.28.1-py3-none-any.whl (7.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.0/7.0 MB 96.6 MB/s eta 0:00:00
Collecting xxhash
  Downloading xxhash-3.2.0-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (212 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 212.2/212.2 kB 24.3 MB/s eta 0:00:00
Requirement already satisfied: pyarrow>=8.0.0 in /usr/local/lib/python3.9/dist-packages (from datasets) (9.0.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.9/dist-packages (from datasets) (1.22.4)
Collecting aiohttp
  Downloading aiohttp-3.8.4-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 62.5 MB/s eta 0:00:00
Collecting responses<0.19
  Downloading responses-0.18.0-py3-none-any.whl (38 kB)
Collecting dill<0.3.7,>=0.3.0
  Downloading dill-0.3.6-py3-none-any.whl (110 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 110.5/110.5 kB 12.8 MB/s eta 0:00:00
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.9/dist-packages (from datasets) (2.27.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.9/dist-packages (from datasets) (6.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.9/dist-packages (from datasets) (1.5.3)
Collecting huggingface-hub<1.0.0,>=0.11.0
  Downloading huggingface_hub-0.13.4-py3-none-any.whl (200 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 200.1/200.1 kB 24.2 MB/s eta 0:00:00
Requirement already satisfied: packaging in /usr/local/lib/python3.9/dist-packages (from datasets) (23.1)
Collecting multiprocess
```

```python
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 #可以將警告訊息在運行時忽略
```

二、模型會用到的小函數（TODO1、TODO2)

- TODO1: 完成 get_pred()
  1. 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()
  1. 先將 tensor 轉換為 numpy 陣列
  2. 使用從 scikit-learn 函數庫中引入的 accuracy_score()、f1_score()、recall_score() 和 precision_score()，計算 accuracy、f1_score、recall、precision
  3. 計算 confusion matrix 混淆矩陣

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

from sklearn.metrics import accuracy_score, f1_score, recall_score, precision_score, confusion_matrix

# calculate confusion metrics
def cal_metrics(pred, ans):
    ##########
    # todo #
    ##########
    # 將 tensor 轉換為 numpy 陣列
    pred = pred.cpu().numpy()
    ans = ans.cpu().numpy()

    # 計算 accuracy、f1_score、recall、precision
    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')

    # 計算 confusion matrix
    cm = confusion_matrix(ans, pred)

    return acc, f1, recall, precision, cm
```

```python
# save model to path
def save_checkpoint(save_path, model):
    if save_path == None:
            return
    torch.save(model.state_dict(), save_path) #將模型的狀態字典存儲到指定的路徑上
    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) #載入模型的狀態字典
    print(f'Model loaded from <== {load_path}')

    model.load_state_dict(state_dict) #並將其放入 model 中。
    return model
```

三、載入資料 (TODO3)

```
from datasets import load_dataset

dataset = load_dataset("imdb")
```

```
Downloading builder script: 100% 4.31k/4.31k [00:00<00:00, 255kB/s]
Downloading metadata: 100% 2.17k/2.17k [00:00<00:00, 135kB/s]
Downloading readme: 100% 7.59k/7.59k [00:00<00:00, 322kB/s]
Downloading and preparing dataset imdb/plain_text to /root/.cache/huggingface/datasets/imdb/plain_text/1.0.0/d613c88cf8fa3bab83
Downloading data: 100% 84.1M/84.1M [00:07<00:00, 17.1MB/s]
Generating train split: 100% 25000/25000 [00:29<00:00, 3673.29 examples/s]
Generating test split: 100% 25000/25000 [00:22<00:00, 4475.62 examples/s]
Generating unsupervised split: 100% 50000/50000 [00:28<00:00, 7431.05 examples/s]
Dataset imdb downloaded and prepared to /root/.cache/huggingface/datasets/imdb/plain_text/1.0.0/d613c88cf8fa3bab83b4ded3713f1f7
100% 3/3 [00:00<00:00, 103.58it/s]
```

```
dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 25000
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 25000
    })
    unsupervised: Dataset({
        features: ['text', 'label'],
        num_rows: 50000
    })
})
```

```
dataset['train'][0]
```

```
{'text': 'I rented I AM CURIOUS-YELLOW from my video store because of all the controversy that surrounded it when it was first re
customs if it ever tried to enter this country, therefore being a fan of films considered "controversial" I really had to see thi
Swedish drama student named Lena who wants to learn everything she can about life. In particular she wants to focus her attention
thought about certain political issues such as the Vietnam War and race issues in the United States. In between asking politician
politics, she has sex with her drama teacher, classmates, and married men.<br /><br />What kills me about I AM CURIOUS-YELLOW is
sex and nudity scenes are few and far between, even then it\'s not shot like some cheaply made porno. While my countrymen mind fi
Swedish cinema. Even Ingmar Bergman, arguably their answer to good old boy John Ford, had sex scenes in his films.<br /><br />I d
film is shown for artistic purposes rather than just to shock people and make money to be shown in pornographic theaters in Ameri
the meat and potatoes (no pun intended) of Swedish cinema. But really, this film doesn\'t have much of a plot.',
 'label': 0}
```

- TODO3:
  1. 把資料拿出來後，轉換為 Pandas DataFrame 格式
  2. 將 train 及 test 合併，重新切割為 8:1:1，再個別儲存下來

```
import pandas as pd

all_df = [] # a list to save all data

##########
# todo #
##########
# 將資料轉換為 Pandas DataFrame
train_df = pd.DataFrame(dataset['train'])
test_df = pd.DataFrame(dataset['test'])

# 合併 train 和 test 資料集
all_df = pd.concat([train_df, test_df], axis=0)

# 將 all_data 重新切割為 train 和 test 和 val
train_data, temp_data = train_test_split(all_df, random_state=1111, train_size=0.8)
val_data, test_data = train_test_split(temp_data, random_state=1111, train_size=0.5)

# 儲存 train 和 test 和 val 資料集
train_data.to_csv('train.csv', index=False)
test_data.to_csv('test.csv', index=False)
val_data.to_csv('val.csv', index=False)

pd.read_csv("./train.csv").head()   #前五筆資料
```

	text	label
0	What a weekend. Two days ago I watched the fir...	0
1	wow, the Naked Brothers Band. What should i sa...	0
2	Just to clarify, Matthew Poncelet wasn't a rea...	1
3	I participate in a Filmmaker's Symposium, and ...	1
4	What the hell is this? "Kooky drama"? "Lawyers...	0

```
all_df.label.value_counts() / len(all_df)
```

```
0    0.5
1    0.5
Name: label, dtype: float64
```

```
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)
```

```
# of train_df: 40000
# of dev_df: 5000
# of test_df data: 5000
```

四、自定義 Dataset，將 tokenize 的步驟放進去（TODO4)

- TODO4: 完成 tokenize()
  1. 將輸入的字串編碼為一系列的 token
  2. 創建一個空的字典 data，它將用於存儲已編碼的 token。
  3. 使用 tokenizer.encode_plus 方法將輸入文本編碼為 token。
     add_special_tokens=True 指定在輸入句子的開頭和結尾添加特殊標記，
     max_length=self.max_len 指定了最大的輸入序列長度，
     padding="max_length"將序列填充到指定長度，
     truncation=True 表示將超過最大長度的序列截斷，
     return_token_type_ids=True 指定返回 token type IDs，
     return_attention_mask=True 指定返回 attention mask，
  4. 將已編碼的 input_ids、attention_mask 和 token_type_ids 轉換為 PyTorch tensor 對象，並分別存儲到 data 字典的相應鍵值對中。
  5. 返回已編碼的 token 數據。

```python
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): #CustomDataset 是一個繼承 Dataset 的自定義類別
    def __init__(self, mode, df, specify, args):  # self像java的this # mode代表目前是在訓練、驗證還是測試模式 # df代表包含資料的 DataFrame # specify代表 DataFrame 中要被用來做預
        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"]) #預先訓練好的 tokenizer，讓它可以將原始文本轉換成 BERT 模型所需的 input IDs、attention mask、token type IDs 等
                                                                    #args["config"] 是一個設定檔，代表使用哪一種預訓練的 BERT 模型。
        self.max_len = args["max_len"]
        self.num_class = args["num_class"]

    def __len__(self):
        return len(self.df) #計算資料集的樣本數

    # 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)
```

```python
    # transform text to its number
    def tokenize(self,input_text): #將輸入的字串編碼為一系列的 token
        ##########
        # todo #
        ##########
        data = {}
        encoded_dict = self.tokenizer.encode_plus(
                            input_text,
                            add_special_tokens=True,  #在輸入句子的開頭和結尾添加特殊標記
                            max_length=self.max_len,  #指定了最大的輸入序列長度
                            padding="max_length",      #將序列填充到指定長度
                            truncation=True,          #將超過最大長度的序列截斷
                            return_token_type_ids=True,
                            return_attention_mask=True
                        )
        data["input_ids"] = torch.tensor(encoded_dict.input_ids, dtype=torch.long)
        data["attention_mask"] = torch.tensor(encoded_dict.attention_mask, dtype=torch.long)
        data["token_type_ids"] = torch.tensor(encoded_dict.token_type_ids, dtype=torch.long)
        return data
```

```python
    # get single data
    def __getitem__(self, index): #用於從資料集中取得單一的資料

        sentence = str(self.df[self.specify][index])
        data = self.tokenize(sentence)
        ids = data["input_ids"]
        mask = data["attention_mask"]
        token_type_ids = data["token_type_ids"]

        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)

        #在方法中，會根據 self.mode 的值來判斷是否為測試模式，如果是，則只回傳經過 tokenize() 方法處理後的資料，否則還會回傳標籤。
        #如果 num_class 大於 2，則會回傳標籤的 one-hot 編碼。最後，將所有的資料轉換成 PyTorch 張量型別並回傳。
```

五、建立模型（TODO5)

- TODO5: 完成 BertClassifier

  __init__() :

  1. 創建 BertClassifier 類別，繼承自 BertPreTrainedModel 類別
  2. 在初始化函數中，呼叫父類別的初始化函數
  3. 設 self.bert 為一個 BertModel 的實例，使用 config 參數設置模型
  4. 創建一個 dropout 層 self.dropout
  5. 創建一個一個線性層 self.classifier（其維度為類別數量）

  forward ():

  1. 定義 forward 函數，參數為 input_ids、attention_mask 和 token_type_ids
  2. 使用 self.bert 對輸入參數進行處理，獲得輸出 outputs
  3. 從 outputs 中取出 pooler_output，存儲在 pooled_output 中
  4. 將 pooled_output 輸入至 self.dropout 層進行 dropout 操作，得到輸

出 pooled_output

5. 將 pooled_output 輸入至 self.classifier 線性層進行線性變換，得到輸出 logits
6. 返回 logits

```
# 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"])   # add dropout layer
        self.classifier = torch.nn.Linear(config.hidden_size, args["num_class"])   # add linear layer for classification

    # forward function, data in model will do this
    def forward(self, input_ids, attention_mask, token_type_ids):
        ##########
        # todo #
        ##########
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask, token_type_ids=token_type_ids)

        pooled_output = outputs.pooler_output   # take only the sentence representation

        pooled_output = self.dropout(pooled_output)   # pass through dropout layer
        logits = self.classifier(pooled_output)   # pass through linear layer

        return logits
```

```
# evaluate dataloader
def evaluate(model, data_loader, device):
    val_loss, val_acc, val_f1, val_rec, val_prec = 0.0, 0.0, 0.0, 0.0, 0.0
    step_count = 0
    loss_fct = torch.nn.CrossEntropyLoss()
    model.eval()
    with torch.no_grad():
        for data in data_loader:
            ids, masks, token_type_ids, labels = [t.to(device) for t in data]

            logits = model(input_ids = ids,
                           attention_mask = masks,
                           token_type_ids = token_type_ids
                           )
            acc, f1, rec, prec ,cm= cal_metrics(get_pred(logits), labels)
            loss = loss_fct(logits, labels)

            val_loss += loss.item()
            val_acc += acc
            val_f1 += f1
            val_rec += rec
            val_prec += prec
            step_count+=1

        val_loss = val_loss / step_count
        val_acc = val_acc / step_count
        val_f1 = val_f1 / step_count
        val_rec = val_rec / step_count
        val_prec = val_prec / step_count

    return val_loss, val_acc, val_f1, val_rec, val_prec
```

六、開始訓練（TODO6)

```python
from datetime import datetime
parameters = {
        "num_class": 2,
        "time": str(datetime.now()).replace(" ", "_"),
        # Hyperparameters
        "model_name": 'BERT',
        "config": 'bert-base-uncased',
        "learning_rate": 3e-5,
        "epochs": 4,
        "max_len": 256,
        "batch_size": 32,
        "dropout": 0.1,
}
```

```python
transformers.logging.set_verbosity_error() # close the warning message

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = BertClassifier.from_pretrained(parameters['config'], parameters).to(device)
loss_fct = torch.nn.CrossEntropyLoss() # we use cross entrophy loss

## You can custom your optimizer (e.g. SGD .etc) ##
# we use Adam here
optimizer = torch.optim.Adam(model.parameters(), lr=parameters['learning_rate'], betas=(0.9, 0.999), eps=1e-9)

## You also can add your custom scheduler ##
# num_train_steps = len(train_loader) * parameters['epochs]
# scheduler = get_cosine_schedule_with_warmup(optimizer, num_warmup_steps=int(0.1 * num_train_steps), num_training_steps=num_train_steps, num_cycles=1)
```

```
Downloading pytorch_model.bin: 100% 440M/440M [00:01<00:00, 285MB/s]
```

```python
import transformers
import pandas as pd

# load training data
train_df = pd.read_csv('./train.tsv', sep = '\t').sample(4000).reset_index(drop=True)
train_dataset = CustomDataset('train', train_df, 'text', parameters)
train_loader = DataLoader(train_dataset, batch_size=parameters['batch_size'], shuffle=True)

# load validation data
val_df = pd.read_csv('./val.tsv', sep = '\t').sample(500).reset_index(drop=True)
val_dataset = CustomDataset('val', val_df, 'text', parameters)
val_loader = DataLoader(val_dataset, batch_size=parameters['batch_size'], shuffle=True)
```

```
Downloading (…)okenizer_config.json: 100% 28.0/28.0 [00:00<00:00, 1.08kB/s]
Downloading (…)lve/main/config.json: 100% 570/570 [00:00<00:00, 16.5kB/s]
Downloading (…)solve/main/vocab.txt: 100% 232k/232k [00:00<00:00, 6.87MB/s]
Downloading (…)/main/tokenizer.json: 100% 466k/466k [00:00<00:00, 10.9MB/s]
```

- TODO6: 完成訓練

1. 使用 train_loader 進行訓練
2. t.to(device)將資料傳送到 GPU 進行運算
3. 將 optimizer 的梯度歸零
4. 模型進行預測，得到 logits
5. 計算 loss，即預測值和實際值的誤差
6. 計算模型的 acc、f1、rec、prec、cm
7. 將每一個 batch 的 acc、f1、rec、prec、loss

8. 將 step_count 加 1，用於計算每個 epoch 的平均值

```
# 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 #
        ##########
        # train the model
        model.train()

        for data in train_loader:
                ids, masks, token_type_ids, labels = [t.to(device) for t in data]
                optimizer.zero_grad()
                logits = model(input_ids = ids,
                                attention_mask = masks,
                                token_type_ids = token_type_ids
                        )
                loss = loss_fct(logits, labels)
                loss.backward()
                optimizer.step()

                acc, f1, rec, prec ,cm= 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: 181.6932 s
         loss      acc      f1       rec      prec
train | 0.4077, 0.8093, 0.7978, 0.8125, 0.8293
val   | 0.3282, 0.8766, 0.8739, 0.8779, 0.8830

[epoch 2] cost time: 181.2674 s
         loss      acc      f1       rec      prec
train | 0.1883, 0.9315, 0.9290, 0.9317, 0.9332
val   | 0.3182, 0.8805, 0.8776, 0.8830, 0.8870

[epoch 3] cost time: 181.7819 s
         loss      acc      f1       rec      prec
train | 0.0817, 0.9742, 0.9736, 0.9744, 0.9745
val   | 0.3188, 0.8859, 0.8826, 0.8881, 0.8834

[epoch 4] cost time: 181.3835 s
         loss      acc      f1       rec      prec
train | 0.0467, 0.9862, 0.9857, 0.9860, 0.9867
val   | 0.4069, 0.8902, 0.8888, 0.8935, 0.8933
```

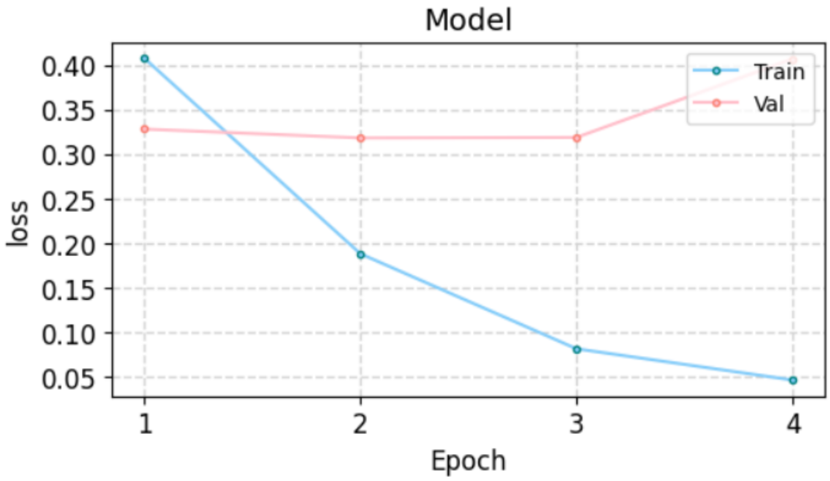
```
# save model
save_checkpoint('./bert.pt' , model)
```

```
Model saved to ==> ./bert.pt
```

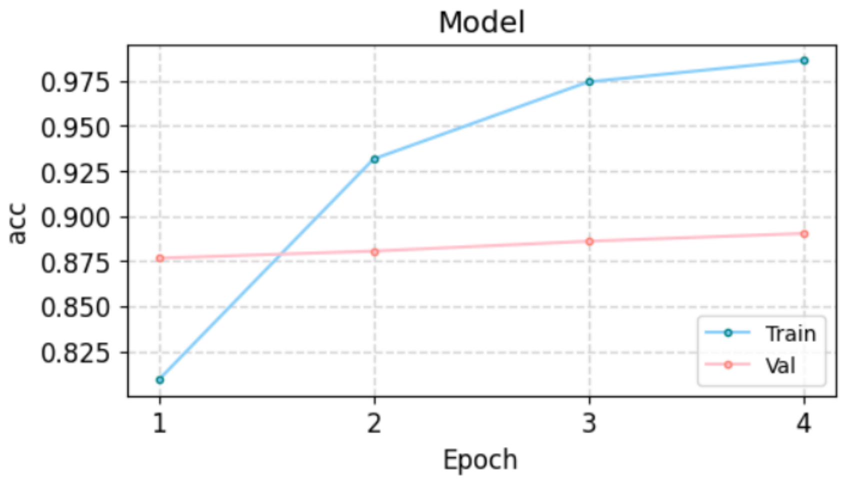
七、畫圖

```
# draw learning curve
import matplotlib.pyplot as plt
def draw_pics(record, name, img_save=False, show=False):
    x_ticks = range(1, parameters["epochs"]+1)

    plt.figure(figsize=(6, 3))

    plt.plot(x_ticks, record['train_'+name], '-o', color='lightskyblue',
             markeredgecolor="teal", markersize=3, markeredgewidth=1, label = 'Train')
    plt.plot(x_ticks, record['val_'+name], '-o', color='pink',
             markeredgecolor="salmon", markersize=3, markeredgewidth=1, label = 'Val')
    plt.grid(color='lightgray', linestyle='--', linewidth=1)

    plt.title('Model', fontsize=14)
    plt.ylabel(name, fontsize=12)
    plt.xlabel('Epoch', fontsize=12)
    plt.xticks(x_ticks, fontsize=12)
    plt.yticks(fontsize=12)
    plt.legend(loc='lower right' if not name.lower().endswith('loss') else 'upper right')

    if img_save:
        plt.savefig(name+'.png', transparent=False, dpi=300)
    if show:
        plt.show()

    plt.close()
```

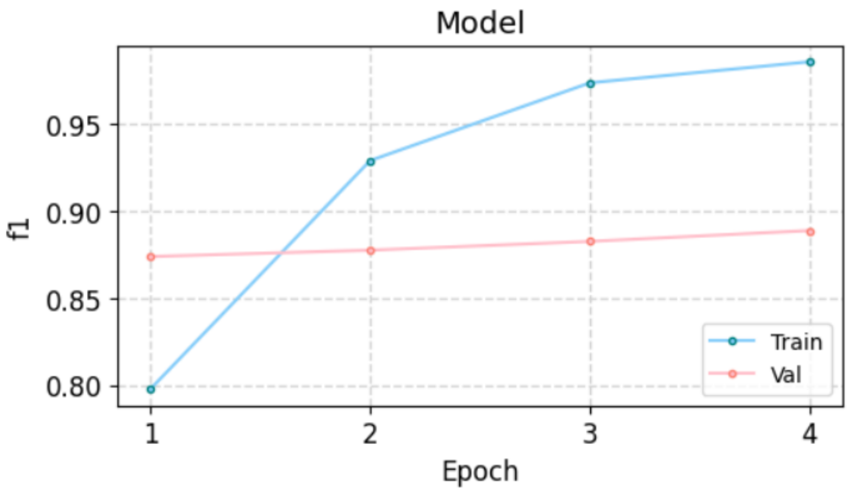
```
draw_pics(record, 'loss', img_save=False, show=True)
```



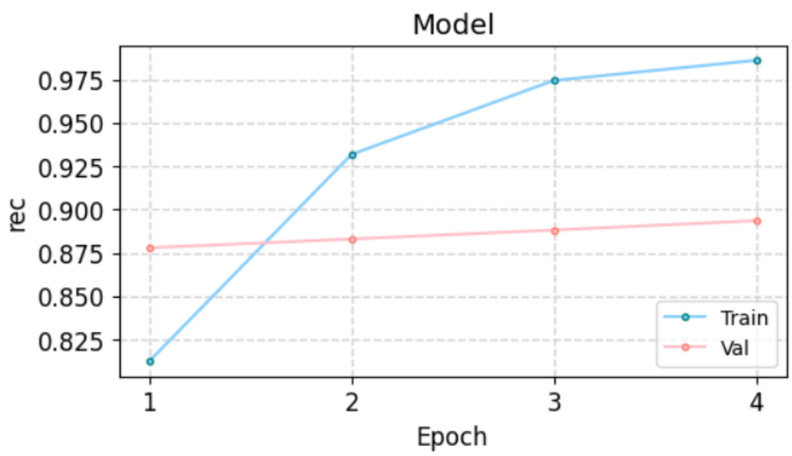

```
draw_pics(record, 'acc', img_save=False, show=True)
```



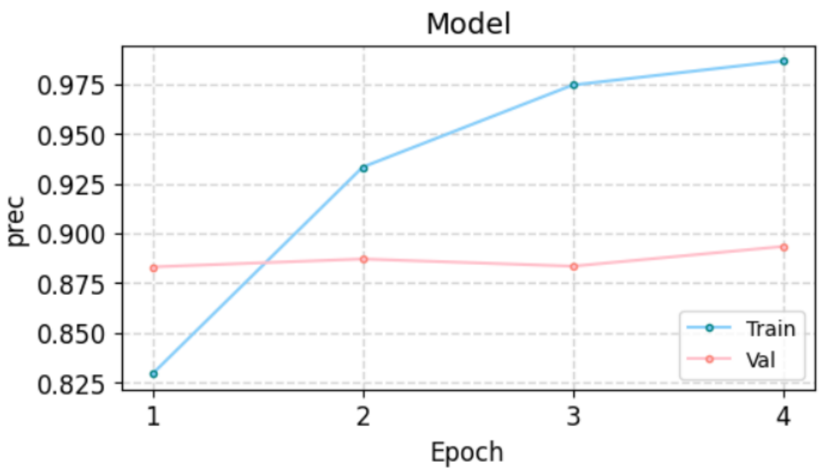

```
draw_pics(record, 'f1', img_save=False, show=True)
```



```
draw_pics(record, 'rec', img_save=False, show=True)
```




```
draw_pics(record, 'prec', img_save=False, show=True)
```



八、預測結果

```
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.item()]
```

- TODO7: 完成 predict_one()
  1. 從參數中載入 tokenizer 和設備
  2. 將模型設為評估模式 model.eval()
  3. 使用 tokenizer 將輸入的文本編碼成 token
  4. query_ids、attention_mask 和 token_type_ids 轉換成 PyTorch 張量
  5. 使用編碼後的文本在模型上進行預測，得到輸出 outputs

6. 將輸出 outputs 通過 Softmax 函數得到預測機率 probs
7. 將預測機率 probs 通過 get_pred() 函數得到預測類別 pred
8. 返回預測機率 probs 和預測類別 pred

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

    ##########
    # todo #
    ##########
    tokenizer = AutoTokenizer.from_pretrained(parameters['config'])
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.eval()
    with torch.no_grad():
        inputs = tokenizer.encode_plus(
                    query,
                    add_special_tokens=True,
                    max_length = parameters["max_len"],
                    truncation = True,
                    padding = 'max_length',
                    return_token_type_ids=True
               )
        query_ids = torch.tensor([inputs.input_ids], dtype=torch.long).to(device)
        attention_mask = torch.tensor([inputs.attention_mask], dtype=torch.long).to(device)
        token_type_ids = torch.tensor([inputs.token_type_ids], dtype=torch.long).to(device)
        outputs = model(input_ids=query_ids,
                        attention_mask=attention_mask,
                        token_type_ids=token_type_ids
                        )
        probs = Softmax(outputs)
        pred = get_pred(outputs)


    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
```

```
Model loaded from <== ./bert.pt
```

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

```
negative
CPU times: user 70.5 ms, sys: 11 ms, total: 81.5 ms
Wall time: 233 ms
```

```python
# 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
```

```python
# load testing data
test_df = pd.read_csv('./test.tsv', sep = '\t').sample(500).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)
```

```python
res.head(5)
```

	text	label	pred
0	"Throw Momma From the Train" is a simple dark ...	1	1
1	A modern scare film? Yep it is.. Th...	0	0
2	In my knowledge, Largo winch was a famous Belg...	0	1
3	1956's The Man Who Knew Too Much is exceptiona...	1	1
4	I have never really been interested in canniba...	0	0

```python
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
```

- **心得:**

  這次的作業我選擇了三個模型去做訓練，分別是 BERT、DISTILBERT、ALBERT。據我查資料所了解的，BERT 是一個較大的模型，需要更多的計算資源和時間來訓練，DISTILBERT 將模型縮小了 40%，因此比 BERT 更快且更容易訓練，然而 ALBERT 使用更輕量級的模型架構和訓練方式，比 BERT 具有更小的參數量，同時也具有更快的速度。
  但在準確率方面，我其實也不確定哪個模型是比較好或不好，因為可能是我沒調整到該模型最好的參數。
  這次做下來感覺比上次的難，蠻多時候都不知道該怎麼做，花的時間也比上次多很多，希望之後作業可以更得心應手一點。