109502009 吳尚明

爬蟲取得電影評論資料集並用於自然語言機器學習

題目發想:

回首學期後半段學習的主題，其中包含如何用 Beautifulsoup 爬蟲，以及各種有關機器學習的理論以及程式實作，因此最初的構想為利用爬蟲收集資料集，並套入到某種機器學習模型中。

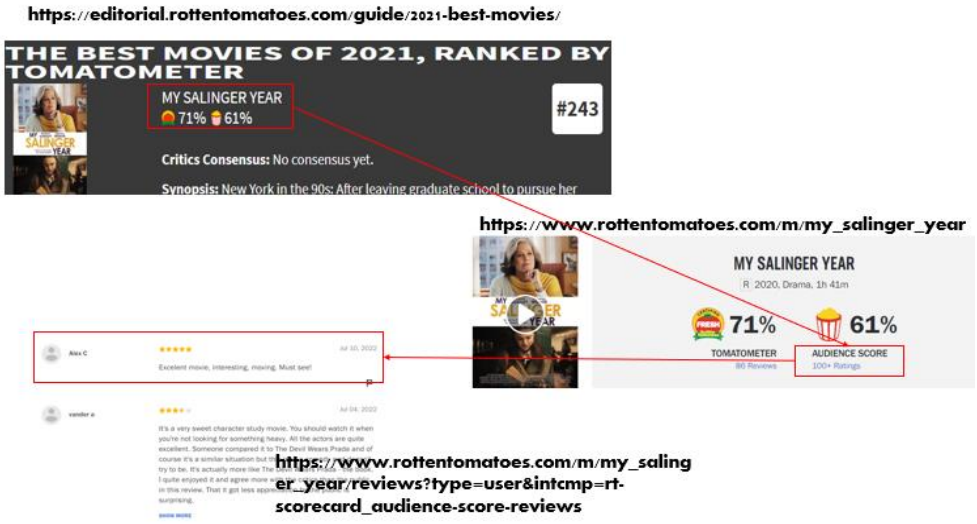
在廣泛搜索 AI 相關應用後，我把目標鎖定在了自然語言處理，我認為人的語言是富含很深層的含意的，但機器竟然也能判斷出語意，這讓我感到不可思議，且我自己又是電影的愛好者，每當觀影完我總想上網查詢影評，其中一個我最常看的評論網站便是爛番茄，論壇上的用戶可以對電影留下評論並附上星級評分，但我觀察到有時不同人給出相似的評論但卻在星級評分上有很不一樣的表現。

因此我開始思考以往人們看的都是這個電影平均獲得幾顆星來判斷受歡迎程度，且某些評論區上的評論可能是沒有附上星級評分的，若從另一個角度出發，是否可以從電影的評論來判斷受歡迎程度呢，將 3.5 顆星以上的評論視為正面評論，以下的視為負面評論，爬蟲抓取爛番茄上的評論以及對應的星級評分讓機器訓練判斷一條評論是正面評論還是負面評論，如此當我爬取例如:2022 年總年度所有電影的評論或者是單一電影的評論時，我可以利用自然語言模組做分類，取得其中的正負評論比率或其他數據運用，即可提供另一個指標給大眾。

功能說明:

1. 首先是爬取爛番茄上的評論當作訓練集:

在第一頁裡有 1~243 編號的電影

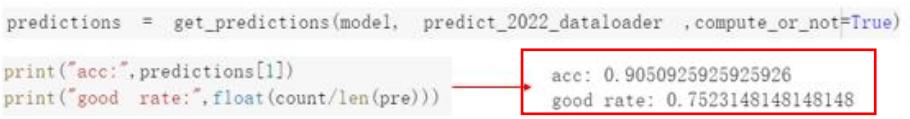


為了使電影評論多元我從 2018 找到 2021 把總共接近 800 多個電影中各取 10 則評論，總共接近 8000 多則評論下載成 csv 檔當作訓練模型的資料集:

```
collect("https://editorial.rottentomatoes.com/guide/2021-best-movies/")
collect("https://editorial.rottentomatoes.com/guide/the-best-movies-of-2020/")
collect("https://editorial.rottentomatoes.com/guide/best-movies-of-2019/")
collect("https://editorial.rottentomatoes.com/guide/best-movies-of-2018/")
```

2. 對文本做預處理並丟到模型裡面訊練(使用 colab 的 GPU 運行):

3. 爬取 2022 年所有電影評論，讓模型計算出 2022 年電影正向評論的比率



4. 輸入特定電影名稱爬取爛番茄上的評論並放入模型預測:

實作技術: 1. 爬蟲部份我用課程學的工具:Bs4,pandas,urllib.request

2. 集前處理運用的工具有:Sklearn, torch, simpletransformers, bert

3. 創建模型:我有總結自己的理解將幾個值得注意的地方做備註

1. 利用 define_model(nn.Module) 建構BERT分類模型

```python
class define_model(nn.Module):
    def __init__(self, freeze_bert=False, hidden_size=768):
        super().__init__()
        config = BertConfig.from_pretrained('bert-base-uncased')
        config.update({'output_hidden_states':True})
        self.bert = BertModel.from_pretrained("bert-base-uncased",config=config)
        self.fc = nn.Linear(hidden_size*4, 2)
        #是否要停止使用bert，讓參數不能參與更新
        #freeze_bert預設為False
        if freeze_bert:
            for p in self.bert.parameters():
                p.requires_grad = False
    def forward(self, input_ids, attention_mask):
        outputs = self.bert(input_ids, attention_mask)
        hidden_states = torch.stack(outputs[2]) #將輸出的tuple以矩陣儲存
        #取得最後四層輸出並進行拼接
        concat_last_four_layers = torch.cat((hidden_states[-1],hidden_states[-2],hidden_states[-3],hidden_states[-4]), dim=-1)
        take_cls = concat_last_four_layers[:,0,:]  #取 [CLS] 這個token對應最後四層concat後的輸出
        logits = self.fc(take_cls)
        return logits
```

2.
1. 透過torch.device回傳使用的是cuda還是cpu
2. 創建model
3. 定義損失函數
4. 優化模型

```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")#回傳使用的是cuda還是cpu
print('已成功取得',device)
model = define_model().to(device)#創建model
criterion = nn.CrossEntropyLoss().to(device)#定義損失函數
#優化模型
#p.requires_grad會回傳參數是否被凍結，若被凍結就過濾掉
#反向傳播需要更新的參數
optim = AdamW(filter(lambda p: p.requires_grad, model.parameters()), lr=2e-5)
total_steps = len(train_loader) * 1
scheduler = get_linear_schedule_with_warmup(optim,
            num_warmup_steps = 0, # Default value in run_glue.py
            num_training_steps = total_steps)
```

4. 訓練模型:

```python
for epoch in range(3):
    print("------------Epoch: %d ----------------" % epoch)
    train()
    validation()
```

```python
def train():
    model.train()
    total_train_loss = 0
    iter_num = 0
    total_iter = len(train_loader)
    for batch in train_loader:
        optim.zero_grad()#先將梯度歸零
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        outputs = model(input_ids, attention_mask=attention_mask)
        loss = criterion(outputs, labels)
        total_train_loss += loss.item()

        loss.backward()#反向傳播計算得到每個參數的梯度值
        nn.utils.clip_grad_norm_(model.parameters(), 1.0) #梯度的裁剪

        # 通過梯度下降執行一部參數更新
        optim.step()
        scheduler.step()
```

```python
def flat_accuracy(preds, labels):
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    return np.sum(pred_flat == labels_flat) / len(labels_flat)
def validation():
    model.eval()
    total_eval_accuracy = 0
    total_eval_loss = 0
    with torch.no_grad():
        for batch in test_dataloader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)
            outputs = model(input_ids, attention_mask=attention_mask)
            loss = criterion(outputs, labels)
            logits = outputs
            total_eval_loss += loss.item()
            logits = logits.detach().cpu().numpy()
            label_ids = labels.to('cpu').numpy()
            total_eval_accuracy += flat_accuracy(logits, label_ids)

    avg_val_accuracy = total_eval_accuracy / len(test_dataloader)
```

1. 把資料放入 model
2. Outputs[:2]為 tuple(outputs.loss,output.logits)
3. Loss = criterion(outputs,labels)

5. 取得預測結果:

```python
把2022年的電影評論拿去預測，計算出正評以及複評的比率，get_predictions會回傳預測的解果
predictions = get_predictions(model, predict_2022_dataloader ,compute_or_not=True)
def get_predictions(model, dataloader, compute_or_not=False):
    predictions = None
    correct = 0
    total = 0
    strl = "strng"
    with torch.no_grad():
        for batch in dataloader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)
            outputs = model(input_ids, attention_mask=attention_mask)
            _, pred = torch.max(outputs.data, 1)
            # 用來計算訓練集的分類準確率
            if compute_or_not:
                labels = batch['labels'].to(device)
                total += labels.size(0)
                correct += (pred == labels).sum().item()
            # 將最後 batch 記錄下來
            if predictions is None:
                predictions = pred
            else:
                predictions = torch.cat((predictions, pred))#依照batch size把分類後的結果整合
    if compute_or_not:#若compute_or_not設為true，及回傳分
        acc = correct / total
        return predictions, acc
    return predictions
```

Torch.max會回傳兩個值一個為value本身另一個為value的index，而我們只關心最大的值是在哪個index，故寫為_, pred = ，因此pred為文本的分類結果

自我評估:

這次的專題是在不斷的構想以及推翻構想中前行，爬蟲不難但如何運用資料相當難，為了想出方向我花費了大量時間在爬有關資料科學與爬蟲的教材，若沒有如此恐怕想不出這個主題，爬蟲以及機器學習我都是第一次完整的應用，花了一段時間弄熟 bs4 提供的工具，花了一段時間在尋找模型，選定了 bert 後，花了超長一段時間去爬文自學，最後實現出了我最初想要的效果，很感謝網路上的學習資源，更感謝的是我自己有靜下心來慢慢克服障礙，除了專題本身的技術外，在設想專題時所蒐尋的知識以及構想專案的練習，都是很充實的自我提升。