

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。

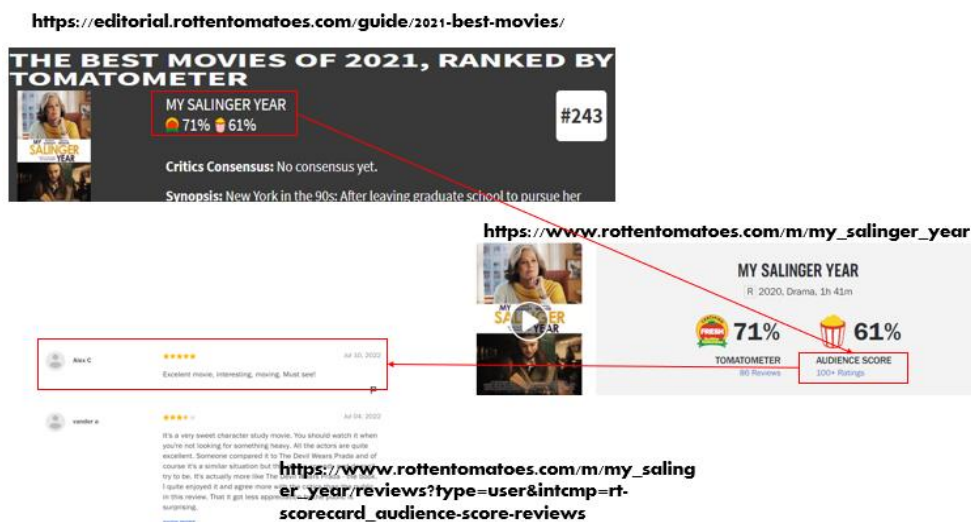
在廣泛搜索 AI 相關應用後，我把目標鎖定在了自然語言處理，我認為人的語言是富含很深層的含意的，但機器竟然也能判斷出語意，這讓我感到不可思議，且我自己又是電影的愛好者，每當觀影完我總想上網查詢影評，其中一個我最常看的評論網站便是爛番茄，論壇上的用戶可以對電影留下評論並附上星級評分，但我觀察到有時不同人給出相似的評論但卻在星級評分上有很不一樣的表現。

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訓練(使用 colab 的 GPU 運行):

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

Train()執行的是training data的訓練
Validation()執行的是testing data的訓練

右圖為執行結果準確率86.6%

```
-----Epoch: 0-----
epoch: 0, iter_num: 100, loss: 0.3992, 25.13%
epoch: 0, iter_num: 200, loss: 0.3385, 50.25%
epoch: 0, iter_num: 300, loss: 0.5334, 75.38%
Epoch: 0, Average training loss: 0.3667
Accuracy: 0.8662
Average testing loss: 0.3194
-----Epoch: 1-----
epoch: 1, iter_num: 100, loss: 0.1160, 25.13%
epoch: 1, iter_num: 200, loss: 0.1754, 50.25%
epoch: 1, iter_num: 300, loss: 0.3252, 75.38%
Epoch: 1, Average training loss: 0.2487
Accuracy: 0.8662
Average testing loss: 0.3193
-----Epoch: 2-----
epoch: 2, iter_num: 100, loss: 0.3353, 25.13%
epoch: 2, iter_num: 200, loss: 0.1995, 50.25%
epoch: 2, iter_num: 300, loss: 0.4637, 75.38%
Epoch: 2, Average training loss: 0.2493
Accuracy: 0.8652
Average testing loss: 0.3214
```

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

```
predictions = get_predictions(model, predict_2022_data_loader, compute_or_not=True)

print("acc:", predictions[1])
print("good rate:", float(count/len(pre)))
```

acc: 0.9050925925925926
good rate: 0.7523148148148148

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:

輸入特定的電影名稱

```
please enter movie name : Avatar: The Way of Water
comment
0 OMG! seriously! visually stunning! awesome sto...
1 I wish the 3D was sharper and less dark but th...
2 Beautiful Animation and Colors and 3D Effects....
3 Better than the first, more emotional. Not a J...
4 Seating was awesome & movie was great.
5 There's nothing else left to see, after this t...
6 It was just "OK." The script was all over the ...
7 appreciate the story but felt it was a little ...
8 Awesome movie cant wait till the next one come...
9 Amazing effects and animation... but foind it ...
(tensor([1, 1, 1, 1, 1, 1, 0, 0, 1, 1], device='cuda:0'), 0.9)
```

正負評論預測結果(只有編號第5則評論預測結果與真實給分不同，但也可能代表機器認為這則更像負評。)

NEW & UPCOMING MOVIES



因為我爬取的是爛番茄上評論所以依舊有評分系統，故順便拿來當對照

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

```
for audience in audiences:
    star=0;
    #取得評論
    p=audience.p.getText()
    #取得星數
    s=audience.span.span.select('span')
    for i in range(5):
        if (s[i].get('class')[0]=='star-display__filled'):
            star+=1;
        if (s[i].get('class')[0]=='star-display__half'):
            star+=0.5;
```

過程中比較大的問題是計算評分的部分，後來找到了星星圖的規律計算出了評分

利用Pandas將爬取的資料下載成csv

```
DF_tr = pd.DataFrame({'comment':p_array,
                      'goodorbad':star_array})
DF_2022 = pd.DataFrame({'comment':p_2022_array,
                        'goodorbad':star_2022_array})
DF_tr.to_csv('commenttr.csv', encoding = 'utf-8-sig')
files.download('commenttr.csv')
DF_2022.to_csv('comment2022.csv', encoding = 'utf-8-sig')
files.download('comment2022.csv')
```

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

1.

利用bert-base-uncased這個模型裡的tokenizer對文本做預處理，會回傳一個字典分別有下列三個鍵值:

1. input_ids: 文句裡各個token在字典中的編碼
2. token_type_ids: 文句裡各個token是在文本中的第幾行
3. attention_mask: token是否為mask

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
train_encoding = tokenizer(x_train, truncation=True, padding=True, max_length=64)
test_encoding = tokenizer(x_test, truncation=True, padding=True, max_length=64)
predict_2022_encoding = tokenizer(x_2022, truncation=True, padding=True, max_length=64)
```

2.

Pytorch提供兩種數據集:此為其中一種Map式的數據集，必須要重寫以下兩個函數:

```
1. getitem(self, idx): 提供idx給dataloader使其能找到對應的data使其能找到對應的data
2. len(self):回傳數據集的長度
class NewsDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels
    #因為之後的DataLoader需要依照batchsize一批批的將傳輸的資料，封裝成訓練模型可用的輸入，所以需要idx使其能找到對應的資料
    def __getitem__(self, idx):
        item = {'key': torch.tensor(val[idx]) for key, val in self.encodi
        item['labels'] = torch.tensor(int(self.labels[idx]))
        return item
    #回傳數據集的長度
    def __len__(self):
        return len(self.labels)
```

3.

將數據集透過NewsDataset以及DataLoader將數據集封裝成訓練模型可用的輸入，其中因為2022的電影評論是我們用來predict的資料，所以不須隨機打亂資料(shuffle=False)

```
train_dataset = NewsDataset(train_encoding, y_train)
test_dataset = NewsDataset(test_encoding, y_test)
predict_2022_dataset = NewsDataset(predict_2022_encoding, y_2022)
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=True)
predict_2022_loader = DataLoader(predict_2022_dataset, batch_size=16, shuffle=False)
```

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

```
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

```
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
optimizer = optim.Adam(model.parameters())#反向傳播需要更新的參數
optimizer = optim.Adam(model.parameters())
total_steps = len(train_loader) * 1
scheduler = get_linear_schedule_with_warmup(optimizer,
                                             num_warmup_steps = 0, # Default value in run_glue.py
                                             num_training_steps = total_steps)
```

4. 訓練模型:

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

```
def train():
    model.train()
    total_train_loss = 0
    iter_num = 0
    total_iter = len(train_loader)
    for batch in train_loader:
        optimizer.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) #梯度的截斷
    # 通過梯度下降執行一部分參數更新
    optimizer.step()
    scheduler.step()
```

```
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:

```
把2022年的電影評論拿去預測, 計算出正評以及極評的比率, 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
    str1 = "string"
    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)
            # 用來計算預測結果的分數並歸零
            if compute_or_not:
                labels = batch['labels'].to(device)
                total += labels.size(0)
                correct += (pred == labels).sum().item()
            # 將預測 batch 記錄下來
            if predictions is None:
                predictions = pred
            else:
                predictions = torch.cat((predictions, pred))#依照batch size把分類的結果整合
    if compute_or_not:#若compute_or_not為true, 返回得分
        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。