基於 Spark 環境上的電影評論語意分析－以 meta-critic 網站之評論為例

指導教授:劉敦仁老師

組　　員:0453406 郭晉晏

0453433 陳楷仁

0453435 簡大鈞

目錄

一、動機

在現在時代網路的發展越來越蓬勃，我們每天所製造以及所獲得的資訊量與日俱增，日益龐大的資訊量漸漸無法為人工所負荷，龐大的資料無法被人們有效率的紀錄並吸收，因此許多人開始嘗試以電腦去處理複雜的問題，而其中一個複雜的問題就是文字探勘的語意分析，電影評論的語意分析就是其中一個相當泛用的例子，現在有許多已經發展的相當有規模的電影評論網站像是爛番茄，IMDB，以及 meta-critic，這些網站上都有相當程度的網友在上面留下自己的影評，若想要歸納這些影片使用傳統的方法運用人力的方式去標籤每個評論，將會曠日費時，因此若能用機器學習來運用演算法產生出近似一般人加入感情所產生的結果，將能有效率的幫助我們歸納這些影評。這學期在資料探勘研究與實務中，在實作作業中我們有學到決策樹的實作，課程中也學習到貝式分類的概念，因此在這次的資料探勘期末專題我們選擇運用 meta-critic 的影評資料來實作電影評論的語意分析，透過 Spark 裡的 MLlib 中的決策樹和 Naïve Bayes 的函式庫來幫助我們建立一個能處理龐大影評資料的機器學習模型，並用此模型能夠正確分類出正反評論，作為是否觀看電影時之參考依據。

二、資料集與分析工具

（一）資料集

在資料探勘的過程中通常會建立預測模型，之後以此模型針對目標進行預測。因此資料集在這方面的應用是個不可或缺的角色。

meta-critics 是一個外國相當熱門之影評網站，上面的影評基本分為兩種：user-critics、meta-critics。網站中 user-critics 是指一般使用者的評論，而 meta-critic 則是專業的影評或影評團隊留下的電影評論，而兩者評論也皆會有對該電影之評分。在此我們使用 python 撰寫爬蟲程式，協助我們抓取一般使用者的 user-critics 以及相對應評分。

（二）分析工具

隨著大數據概念的興起，相關應用在硬體方面有著嚴苛的需求，因此近年來，分散式系統的相關應用也不斷地被發展來協助計算任務的完成，這裡我們使用了以下工具及語言：

Hadoop

是一款支援資料密集型分布式應用的開源軟體框架，隨著巨量資料應用帶來的龐大硬體資源需求，其也成為相當熱門之工具。Hadoop 提供了分布式檔案系統，用以存儲所有計算節點的資料，這為整個集群帶來了非常高的帶寬。MapReduce 和分布式檔案系統的設計，使得整個框架能夠自動處理節點故障。

Spark

是一個開源叢集運算框架，最初是由加州大學柏克萊分校 AMPLab 所開發。相對於 Hadoop 的 MapReduce 會在執行完工作後將中介資料存放到磁碟中，Spark 運用了記憶體內運算速度較快的概念，在資料尚未寫入硬碟時即在記憶體內分析運算。因此 Spark 在運算速度的表現上能做到比 Hadoop MapReduce 快上數倍，非常適合用於機器學習演算法。

Python

是一種物件導向、直譯式的電腦程式語言。它包含了一組功能完備的標準庫，能夠輕鬆完成很多常見的任務，另外也提供了豐富的擴充套件，可以幫助使用者透過 pip install 指令輕鬆安裝所需之功能，這邊我們用了以下的函式庫協助我們在實驗的進行：

Python 函式庫	說明
NumPy	一個經常數學運算用途之函式庫，支援較高維度的陣列及矩陣運算。
Pandas	一個可用於資料格式處理之函式庫，在資料分析的前處理步驟中經常被使用到。

NLTK(Natural Language Toolkit)	一個經常被用於自然語言分析的函式庫，這裡我們取它在情感分析上的處理加以應用。
MLlib	一個建立於 Spark 之上的有關於機器學習函式庫，可被 JAVA、Scala、Python、R 等語言實作，內有許多分類、分群及回歸等機器學習相關之方法。
Splearn	一個 spark 上的擴充套件，可以讓 scikit-learn 的一些方法在 pySpark 上實現

我們將使用 Hadoop 環境和 spark 作為分析工具軟體，而使用的工具主要為 python 的 library 包括 NumPy、Pandas、NLTK 以及 Spark 上的 library MLlib，其中 NumPy 和 Pandas 主要是作為資料處理用，NLTK 有機器學習演算法功能也可以做資料處理， 而最主要的部份我們將會使用到 pySpark 上的分散是機器學習 library MLlib 來運作 decision tree。

三、實作流程

此次分析主要利用兩種方式作分析

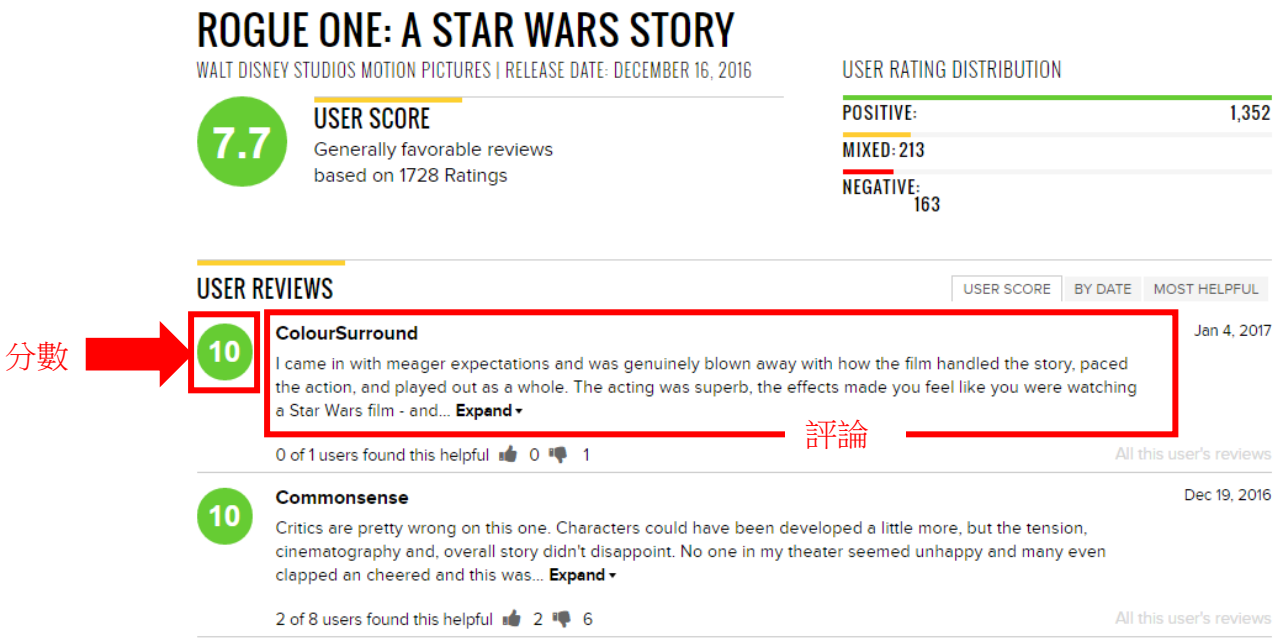
(1) 利用 scikit-learn 將資料作 TF-IDF，以建立出 frequency matrix，然後利用 Naïve Bayes 將資料作分析。

(2) 利用 nltk 中的”sentiment”套件先將資料作語意分析會分析出’compound’(代表總和情緒的高低，介於-1~1，負值表示為負情緒，正值表示為正情緒)、’neg’(負面情緒字詞比例)、’neu’(中間情緒字詞比例)、’pos’(正面情緒字詞比例)。將分析出來的 4 項數據使用 decision tree 作分析。

以上分析皆將資料的 75%設為 training data，剩下 25%為 test data；(1)的部分均在 Spark 上進行，(2)僅有 decision 的部分在 spark 上進行。

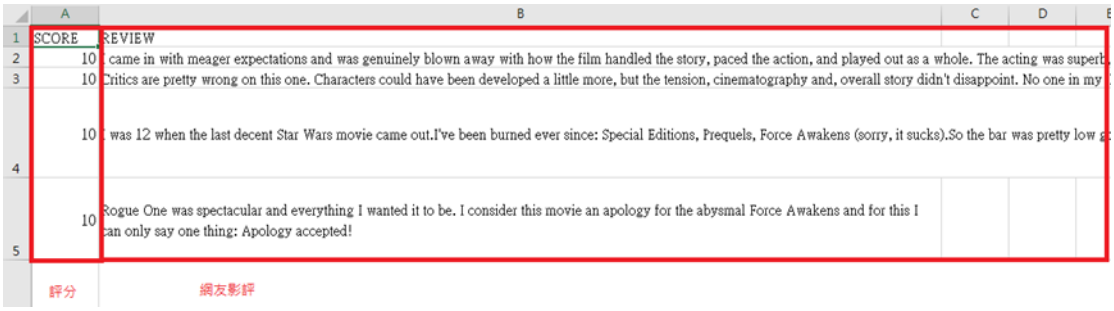
（一）資料擷取

1. 至 metacritic(http://www.metacritic.com/)抓取資料，抓取的部分為評論以及分數，總共抓取資料為 858 筆。



圖（一）metacritic 上我們資料擷取之目標，有評論內容和對電影之分數

2. 利用 python 中的”BeautifulSoup”套件 parser html 中的資料，在將 parse 出的資料轉成 csv 格式並輸出，其輸出檔案命名為 review.csv，如圖(二)。



圖（二）利用爬蟲程式所儲存之格式 CSV 格式

（二）分類方法實作

Mllib 是 spark 官方所提供的有關於機器學習的套件，其中有相當多的機器學習方法可以用於分析像是分類的分析方法如 Naïve Bayes、Decision Tree，SVM，其他也有分群、回歸方法可供利用，另外亦提供範例程式(scala、JAVA、python）等供大家參考，這裡的方法我們選擇實作學期初所學到的決策樹分類法，我們除了使用 Mllib 外也使用 splearn 來作貝式分類器(Naïve Bayes)。

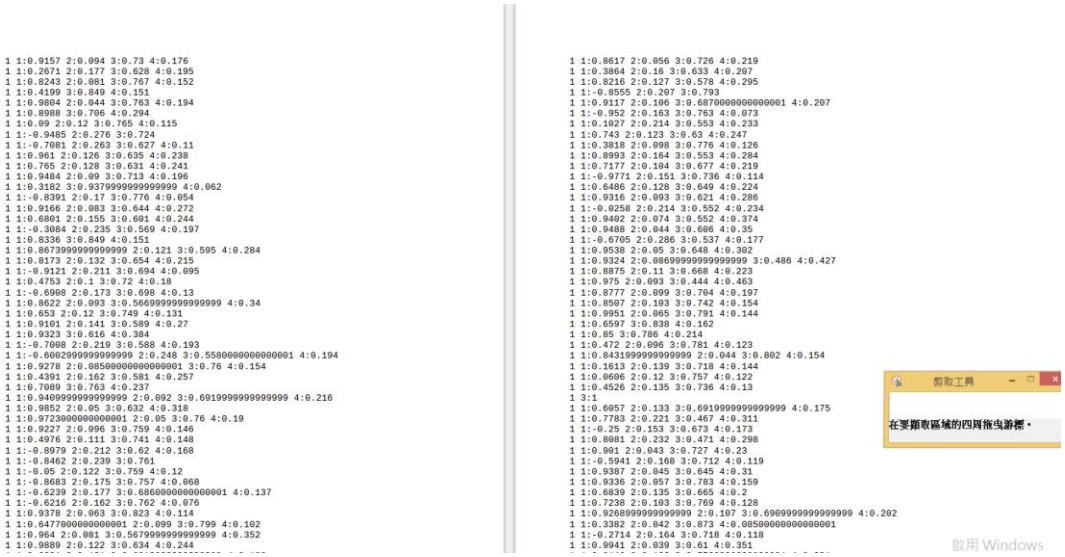
資料前處理

將資料作處理，這次主要的目的為判斷語意與分數高低之關係，因此須先將分數區分出為高或低，故將 0~5 分設為”0”表示為低分，6~10 分設為”1”表示為高分，程式部分如下。

```
df = pd.read_csv("review.csv",header=None,encoding='latin1')
df[0] =df[0].apply(lambda death: 0 if death <= 5 else 1)
df=df.dropna()
```

(a)決策樹

我們運用 nltk 的 SentimentIntensityAnalyzer 將 User Review 產生一個-1~+1 之間的值，-1 代表負面情緒的最極端，+1 代表正面情緒的最極端。將 review 轉換成 SVM 檔案格式的 odm 檔，轉換格式是為了符合 mllib decision tree 的 input 格式，而 data.odm 格式如下：



圖（三）將資料轉為 SVM 格式

每筆的第一值為 Label，後面則為屬性，而每筆皆有四個屬性，每個值皆介於-1 到 1 之間：

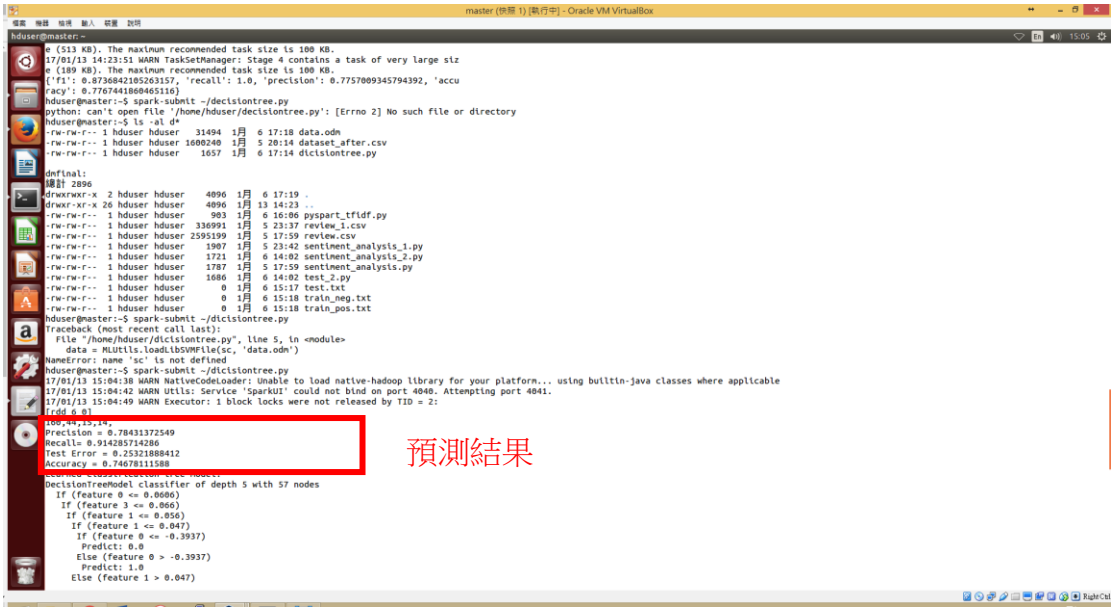
屬性	說明
Compound	代表總和情緒的高低。
Pos	正面情緒字詞比例。
Neu	中間情緒字詞比例。
Neg	負情緒字詞比例。

使用決策樹進行分類

將 data.odm 檔以及我們的程式檔 decisiontree.py 上傳至 hadoop 後，我們便開啟 Spark 進行程式。程式碼如圖（四）所示，我們將利用 predict 方法取得 TP、FP、PN、TN 等數值，再根據此四項數值計算出 Precision Rate、Recall Rate、Accuracy Rate、F1-Score 四個衡量指標。

```
from pyspark.mllib.tree import DecisionTree, DecisionTreeModel
from pyspark.mllib.util import MLUtils

# Load and parse the data file into an RDD of LabeledPoint.
data = MLUtils.loadLibSVMFile(sc, 'data.odm')
# Split the data into training and test sets (25% held out for testing)
(trainingData, testData) = data.randomSplit([0.75, 0.25])

# Train a DecisionTree model.
#  Empty categoricalFeaturesInfo indicates all features are continuous.
model = DecisionTree.trainClassifier(trainingData, numClasses=2, categoricalFeaturesInfo={},
                                     impurity='gini', maxDepth=5, maxBins=32)

# predict test
predictions = model.predict(testData.map(lambda x: x.features))
#map true value and predict value
labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)

testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() / float(testData.count())
acc = labelsAndPredictions.filter(lambda (v, p): v == p).count() / float(testData.count())
#calculate TP FP FN TN
TP=labelsAndPredictions.filter(lambda (v, p): v ==1 and p == 1).count()
FP=labelsAndPredictions.filter(lambda (v, p): v ==0 and p == 1).count()
FN=labelsAndPredictions.filter(lambda (v, p): v ==1 and p == 0).count()
TN=labelsAndPredictions.filter(lambda (v, p): v ==0 and p == 0).count()
print(str(TP)+","+str(FP)+","+str(FN)+","+str(TN)+",")

precision=float(TP)/(TP+FP)
recall=float(TP)/(TP+FN)
print('Precision = ' + str(precision))
print('Recall= ' + str(recall))
print('Test Error = ' + str(testErr))
print('Accuracy = ' + str(acc))
print('Learned classification tree model:')
print(model.toDebugString())
```

圖（四）decisiontree.py 程式碼



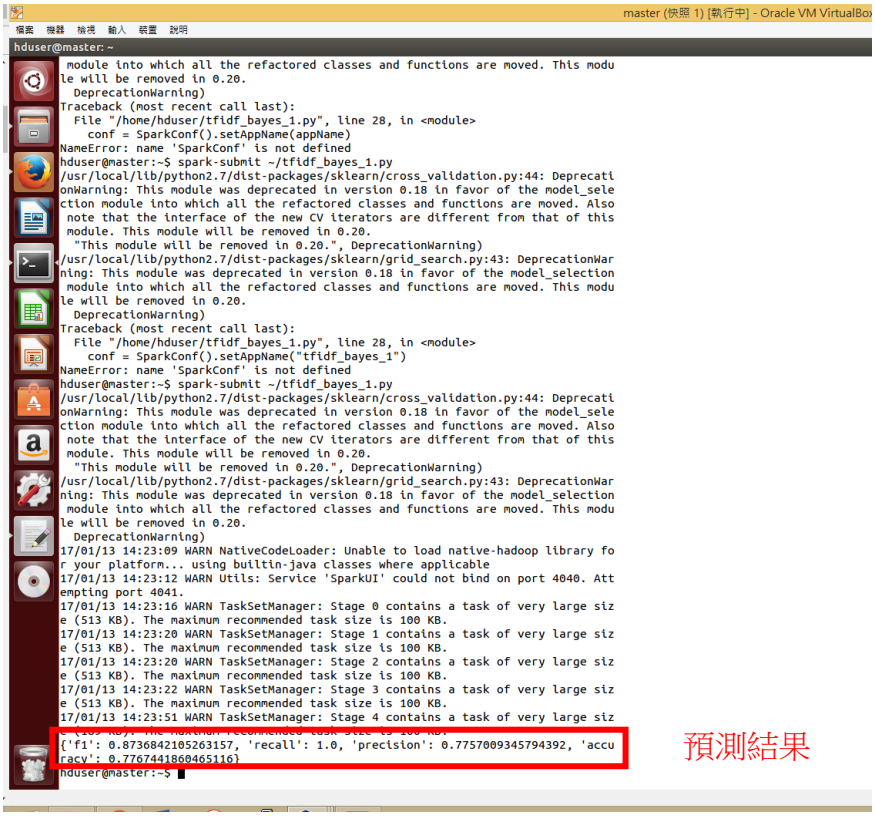
圖（五）將實驗結果輸出

(b)Naïve Bayes

我們將處理過後的資料以及程式碼上傳至 hadoop，並使用 Spark 進行程式的運作，之後同決策樹，我們利用 predict 方法取得 TP、FP、PN、TN 等數值，根據此四項數值計算出 Precision Rate、Recall Rate、Accuracy Rate、F1-Score 四個衡量指標。

```
import pandas as pd

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import BernoulliNB
from sklearn import cross_validation
from sklearn.metrics import classification_report
import numpy as np
from sklearn.metrics import accuracy_score

#preprocess data
def load_file():
    df = pd.read_csv("review.csv",header=None,encoding='latin1')
    df[0] =df[0].apply(lambda death: 0 if death <= 5 else 1)
    df=df.dropna()
    data =df[1]
    target = df[0]
    print(data)
    print(target)
    return data,target

# preprocess creates the term frequency matrix for the review data set using TF-IDF
def preprocess():
    data,target = load_file()
    count_vectorizer = CountVectorizer(binary='true')
    data = count_vectorizer.fit_transform(data)
    tfidf_data = TfidfTransformer(use_idf=False).fit_transform(data)
    return tfidf_data

def learn_model(data,target):
    # preparing data for split validation. 75% training, 25% test
    data_train,data_test,target_train,target_test = cross_validation.train_test_split(data,target,test_size=0.25,random_state=43)
    classifier = BernoulliNB().fit(data_train,target_train)
    predicted = classifier.predict(data_test)
    evaluate_model(target_test,predicted)

# read more about model evaluation metrics here
def evaluate_model(target_true,target_predicted):
    print(classification_report(target_true,target_predicted))
    print ("The accuracy score is {:.2%}".format(accuracy_score(target_true,target_predicted)))

def main():
    data,target = load_file()
    tf_idf = preprocess()
    learn_model(tf_idf,target)


main()
```

圖（六）setiment_analyze.py 程式碼



圖（七）setiment_analyze.py 實驗結果

（三）實驗結果

實驗中，我們的正評筆數共有 166 筆，負評共有 49 筆，兩分類結果如下表：

分類方法	Precision rate	Recall rate	Accuracy
決策樹	78.4%	91.4%	74.6%
Naïve Bayes	77%	100%	77.67%

Decision tree		Actual	
		Pos	Neg
Predicted	Pos	160	44
	neg	15	14

Naïve Bayes		Actual	
		Pos	Neg
Predicted	Pos	166	48
	neg	0	1

四、分析結果與結論

從實驗結果發現其實兩種分類法在準確率上相差不遠，大約皆在 76-77%，而再進一步比較 Recall Rate、Precision Rate 發現，雖然 Precision Rate 兩者相差不遠，但在 Recall Rate 方面卻相差非常多，來到了 100%，導致 f-score 方面也有很大的差距，我們認為這種現象的產生有可能是兩個原因所造成的，第一是在資料前處理上，擷取屬性上所造成的差距，決策樹的 NLTK 僅產生四個屬性之值，然而 Naïve Bayes，由於是做 TFIDF，因此會產生非常多的屬性，導致屬性的矩陣相對稀疏，再來有可能是因為我們在正評和負評的筆數相差過多，也間接導致這樣的結果產生。

根據此問題，我們這裡提出了幾個想法或許可以改善問題：

- 可以找尋更多的訓練資料集來解決 TFIDF 屬性矩陣過於稀疏的問題。
- 在測試資料的正負評筆數上，可以試著更為均衡（正、負評測試資料比至少$>\frac{1}{2}$）。
- 去除明顯正、負面之資料，以較為中立的影評用於訓練以及測試資料。

在未來的相關實作中，我們將會特別注意這方面問題，以期在各項數據能夠避免異常情況，使研究成果更具參考性。