Fianl Project Report: Book Review Recommend System

Motivation (Project Background) and Problem

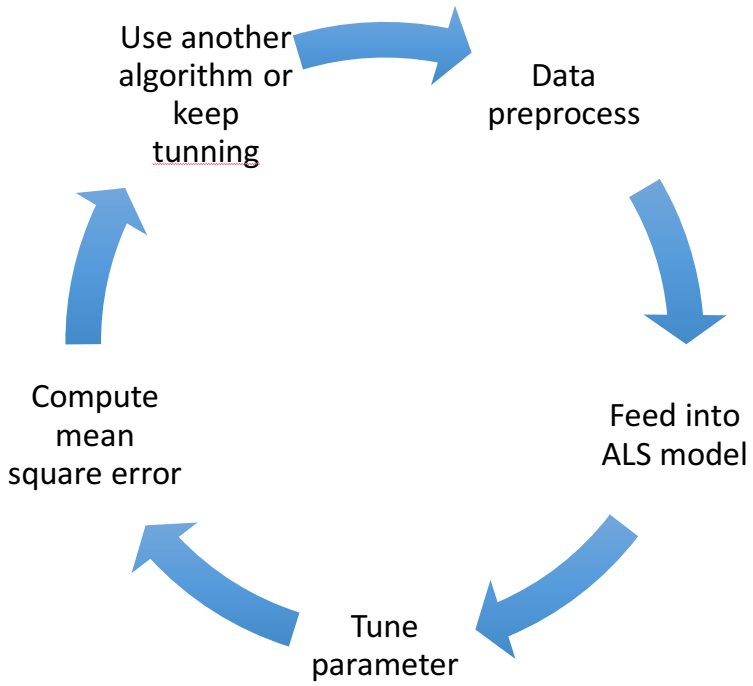
題目動機為隨著社會的進步，每年都有大量的書被出版，這些海量的知識已經不是我們有生之年所能吸收的完，因此我們必須慎選真正對自己有幫助的書，也就是藉由推薦系統來找出和自己相近閱讀興趣的人所喜愛的書，進而從茫茫書海中找出適合自己的書，而不是像以往只看封面就決定要看哪本的習慣。

技術動機為希望能實作老師上課教的 neighborhood method 在 spark 平台上並且和 spark 現有的推薦系統（Alternating Least Square）來比較。最後在這兩個實作的技術上進行參數調校，探討參數和準確率之間的關係。

Project Goal

1. 以 spark 平台來實作兩個推薦方法
2. 針對各個推薦方法進行參數調校

System Flow



在資料前處理的部份共分成以下步驟：

1.把原始的文字資料分割成所希望的屬性格式

2.處理遺失的資料

3.如果 ID 欄位不全是整數的話，要新建 hash table 來做對應

4.要把 User.csv 跟 Book.csv 的檔案整合進來的話，也各新建一個 hash table 才能對應使用

5.把資料轉換成 LabelPoint 的形式才能餵進 spark 的 model 裡

計算 MSE，這是用來計算誤差的，再加上根號就成了標準差。也就是 Rating 的分數作預測值和真實值的相減後平方再取 mean，我會以 Mean square error 來計算誤差找出評價誤差最小的，從而選出最適當的推薦系統。

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

Algorithm

Collaborative Filtering 是用來推薦物品的系統，簡單來說就是把每個使用者當作矩陣的列，每項物品當作矩陣的行，每一筆資料就是使用者對該物品的評分，而我們所希望找的就是填滿剩下的空格來預測使用者對其他物品的喜愛度。

另外需要注意的是在這裡用的是 Explicit feedback，並不是上課所教到的 implicit feedback。這兩者的差異就是對 explicit 來說，餵進去的資料是使用者"直接"對物品的評分，而 implicit feedback 來說，就是還包含了像是點擊數瀏覽數這些比較間接的資訊。但是這次可用的資料集並沒有機會使用到 implicit feedback。

而 Alternating Least Square 是先把整個矩陣分成兩個矩陣相乘，一個是 user factor matrix，另一個是 item factor matrix。

$$Q = X * Y$$

Q: rating matrix

X: user factor matrix

Y: item factor matrix

$$Q_{ui} = \begin{cases} r & \text{if user u rate item i} \\ 0 & \text{if user u did not rate item i} \end{cases}$$

接下來你必須最小化 cost function 但是因為有兩個未知矩陣，所以這裡採取的方法是先固定一個矩陣去預

估另一個，反之繼續互相預估。

Cost function

$$J(x_u) = (q_u - x_u Y) W_u (q_u - x_u Y)^T + \lambda x_u x_u^T$$

$$J(y_i) = (q_i - X y_i) W_i (q_i - X y_i)^T + \lambda y_i y_i^T$$

Solution for factor

$$x_u = (Y W_u Y^T + \lambda I)^{-1} Y W_u q_u$$

$$y_i = (X^T W i X + \lambda I)^{-1} X^T W_i q_i$$

Experiment Platform

- Apache Spark
- Scala
- Spark.mllib

Input Data

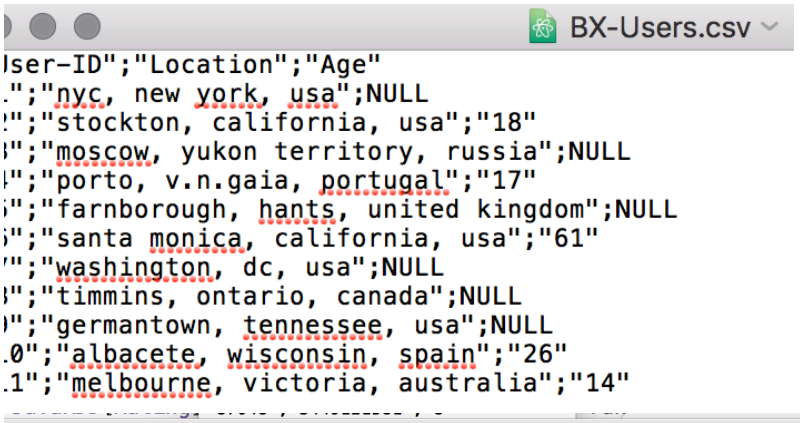
Book-Crossing Dataset

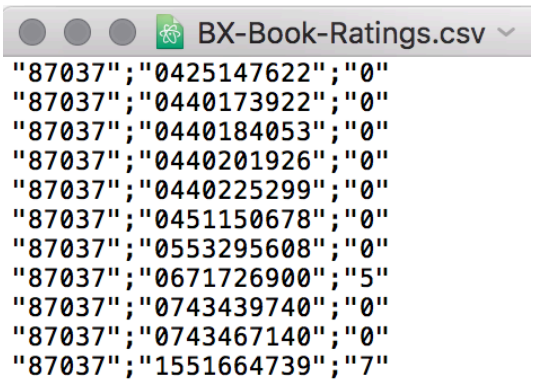
- BX-Users
  Contains the users. Note that user IDs (`*User-ID*`) have been anonymized and map to integers. Demographic data is provided (`*Location*`, `*Age*`) if available. Otherwise, these fields contain *NULL*-values.

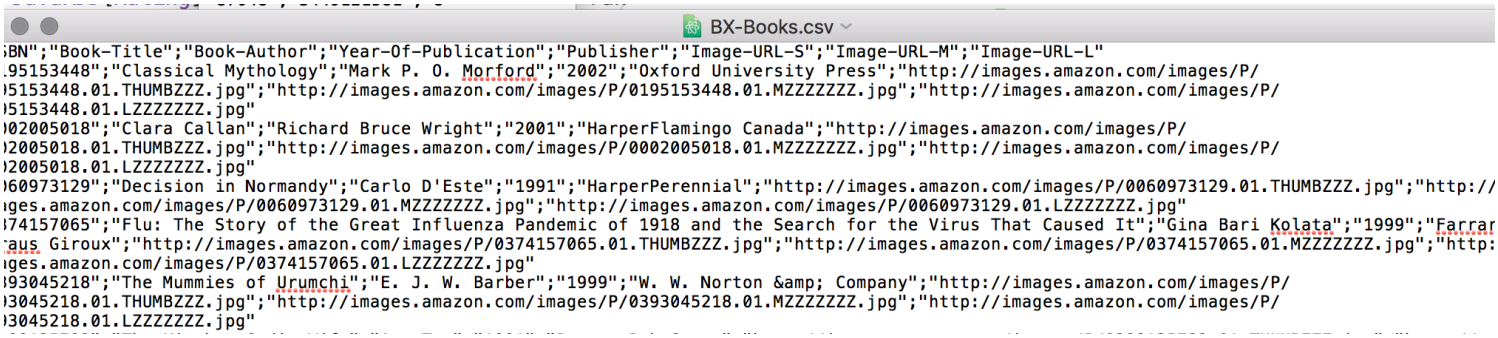
- BX-Books
  Books are identified by their respective ISBN. Invalid ISBNs have already been removed from the dataset. Moreover, some content-based information is given (`*Book-Title*`, `*Book-Author*`, `*Year-Of-Publication*`, `*Publisher*`), obtained from Amazon Web Services. Note that in case of several authors, only the first is provided. URLs linking to cover images are also given, appearing in three different flavours (`*Image-URL-S*`, `*Image-URL-M*`, `*Image-URL-L*`), i.e., small, medium, large. These URLs point to the Amazon web site.

- BX-Book-Ratings
  Contains the book rating information. Ratings (`*Book-Rating*`) are either explicit, expressed on a scale from 1-10 (higher values denoting higher appreciation), or implicit, expressed by 0.

以下是原始資料的截圖。







Experiment Result

先說實驗結論：

0.實作出基於 ALS 的推薦系統

1.Neighborhood method 不適合用在大量資料且分散式的平台上

2.ALS 的方法沒問題，且進行了參數調校找出了參數和準確度之間的

關係

3.通常 model 預設的參數值效果都相當好，這也應證了在大部分的情況下都是不錯才會當作預設值

實驗情形：

在實作 neighborhood method 的過程中，發現沒辦法把演算法寫成平行化的過程，也就是藉著 RDD 使用 map 轉換的過程，雖然可以用普通的寫法硬寫出來，但是會失去了 RDD 分散式儲存並平行計算的本意了，接著又遭遇到 Exception in thread "main" java.lang.OutOfMemoryError: Java heap space 問題，解決辦法是要增加 JVM 裡的 memory 上限，但最後僅管我再怎麼增加還是沒辦法解決。

在這邊可以發現，同樣大小的資料，使用 ALS 的話反而不會出現記憶體的問題又可以平行運算。

接下來就是關於參數調校，在 ALS 中一共有三個參數可以調校，iteration, lambda, block。首先 Iteration 就是前面提到 ALS 互相預估的過程要跑幾次的參數，照理說，如果跑越多次的話效果會越好，根據圖中的表現可以看出越高效果的確是越好的，計算過後準確率提高了 69.11%，但是，難不成就調更高讓他一直跑下去不就好了？

((11,1.0E-4,20),1.3252719578428072)
((11,1.0E-6,20),1.440682594902883)
((11,1.0E-4,10),1.6331643708272123)
((11,1.0E-6,10),1.7820553760731273)
((11,1.0E-4,1),4.290268352234962)

```
((rank, lambda, iter), mse,
```

但是經過實驗之後反而發現，調更高的時候反而會出現

StackOverflowError，目前還不知道解法，因此理想的狀況是調高到

快接近極限就會有好的 model 了

```
java.lang.StackOverflowError
    at java.io.ObjectInputStream$PeekInputStream.read(ObjectInputStream.java:2320)
    at java.io.ObjectInputStream$PeekInputStream.readFully(ObjectInputStream.java:2333)
    at java.io.ObjectInputStream$BlockDataInputStream.readInt(ObjectInputStream.java:2828)
    at java.io.ObjectInputStream.readHandle(ObjectInputStream.java:1453)
    at java.io.ObjectInputStream.readClassDesc(ObjectInputStream.java:1512)
    at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1774)
    at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
    at java.io.ObjectInputStream.readObject(ObjectInputStream.java:371)
    at scala.collection.immutable.$colon$colon.readObject(List.scala:362) <3 internal calls>
```

再來是 lambda 的部分，這就只是很單純的 regularization 而已，所以

太小會 overfiiting，太大又會不準確，

((11,1.0,20),5.53831748751736,)
((11,0.01,20),1.3192921338906438)
((11,1.0E-4,20),1.3252719578428072)
((11,1.0E-6,20),1.440682594902883)

```
((rank, lambda, iter), mse,
```

從上圖可以看得出來只要到一定程度準確率就會趨緩了，這部分不

出預期。

最後一個參數是 block，這是我們的 user factor matrix 還有 item

factor matrix 要分割成多少個 block 來執行。通常設定得當的話可以加速運算。

```
block 20:    115691070910
block 10:     95523212344
block 5 :    107714440456
block -1:     63925225620
        [2]:         71880645939
        [8]:         77448430999
block 4:      62951545432
block 2:      73836873879
```

單位是 nano time，簡單來說就是要分割成幾個最好是跟你開多少個 threads 來平行運算 spark 有關，如果你是 local[4]，那 block 就設 4 的效果就會最好了。

另外就是原本要選用的資料集，是 amazon 的資料，更大而且還有很多 implicit 的欄位可以做 implicit feedback，但是寫完了才遇到一個問題是 *Exception in thread thread_name: java.lang.OutOfMemoryError: GC Overhead limit exceeded* 也就是你整個執行的過程中，有 98%以上的時間都在跑 garbage collection，真正執行程式碼的時間不到 2%才會跳出這個 error，網路上的解決辦法是在跑 JVM 前設一個 flag 來關掉跳出 error，但是這沒辦法解決問題，資料還是跑不動，最後只好放棄，重新找資料集以及重新寫程式碼。

最後就是推薦結果的呈現截圖，給使用者 ID 為 87351 的推薦 5 本書。

Don't Shoot the Dog! : The New Art of Teaching and Training
Variation on a Theme
Yogi: It Ain't over
Lost (Mira)
Stranger in a Strange Land

以及這個 model 最好的參數及最佳的 MSE 值。

((11,1.0E-4,20),1.3252719578428072)

Reference:

1. http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html
2. http://www2.informatik.uni-freiburg.de/~cziegler/BX/
3. https://en.wikipedia.org/wiki/Mean_squared_error
4. 上課講義 How does Netflix recommend movies_0519.pdf
5.