Introduction to Massive Data Analysis

Homework 4 Report

Student ID: 105062635
Name: 吳浩寧

I. MEANING OF EACH FILE

※表示實際上沒有用到的程式碼，有沒有都沒差

1.) **Main.java**

假設 M 是我們每個 user 對每個電影的 utility matrix，每一條 row 代表一個使用者，每個 column 代表一部電影，其維度為 n×m，每一個元素代表使用者對該電影的評分，範圍是 1~5 或是空的。若我們要將 M 分解成兩個矩陣 U 和 V，這兩個矩陣的大小分別會是 n×d 和 d×m

程式中可以指定的相關變數如下：

SOLUTION	使用哪種計算方式	**DIMENSIONS**	d
MOVIES	電影數量 m	**USERS**	使用者數量 n
REDUCERS	在每個 job 使用 setNumReduceTasks(REDUCERS) 來設定使用的 reducers 數量		
REPLICATION_FACTOR	為了避免 reducer 記憶體不夠大，把工作再切成這個數值的份數。		

2.) **Preprocessing.java**

將 M 進行 normalization，也就是針對每個使用者，算出其給出所有分數的平均，並將原始分數減去平均，藉此得到新的分數，如此就能讓分數平均分佈在正和負的方向，在計算彼此距離時差距會更顯著。

Map Function:

純粹將輸入檔每一行讀進來傳給 reducer<userID, movieID, grade, date-of-grade>，並以 userID 作為 key。

Reduce Function:

為了讓每次 reduce 可以產生多個輸出檔，必須宣告 MultipleOutputs 物件，並 override setup()，並在其中利用 context 初始化，如此就能執行物件的 write()取代原本的 context.write()；也必須 override cleanup()進行資源的釋放。

先透過一個迴圈將每筆資料透過 Map<Integer, String>存起來，如此可以讓輸出檔依照 movieID 排序。在 put 資料的同時，也會判斷 Hashmap 中是否已經有同樣的元素，若有則代表使用者已對該電影做出評分，但由於輸入檔中不同日期的項目前後順序不固定，這種寫法可能造成新評分不會把舊的覆蓋掉；在此迴圈也會計算將使用者評過的電影數，和評價累加起來。

跑完迴圈，即可算出每位使用者給出的平均分數，再遍歷過前面產生的 Hashmap，使用 MultipleOutputs 的 write(String namedOutput, K key, V value, String baseOutputPath)，namedOutput 為自己定義給這個 output 的名稱，要執行 job 時即可將該名稱傳入 addNamedOutput()來使用；baseOutputPath 則是產生檔案的基底名稱，若有多個檔案就會以 baseOutputPath-r-#####的格式來名命，最後會輸出格式為<M, (userID, movieID, grade-mean)>和<userID, (sum, count)>。

input	output	
1,1,3,2005-09-06	M--r-#####	sum--r-#####
1,2,5,2005-05-13	M,1,3,-1	1,12,3
1,5,4,2005-10-19	M,2,5,1	2,9,3
2,2,4,2005-09-05	M,5,4,0	
2,3,3,2005-04-19	M,2,4,1	
2,4,2,2005-04-22	M,3,3,0	
	M,4,4,-1	

※getMeanFromDFS()是在 MapReduce 執行完之後，使用 FileStatus[]來記錄輸出資料夾裡面所有檔案，找出檔名有"sum-"者，將資料一行一行讀出來，針對每個使用者的 sum 和 count 進行加總，最後對 M 中非空元素計算平均，開根號後回傳給呼叫 Preprocessing.run()者，此 function 在 IterationV.java、IterationVSol2.java 也有出現。

3.) **UVGen.java**

初始化 U、V 矩陣，為了讓學習的效果更好，每個元素最好是隨機分佈的，如此進行 gradient descent 時比較不容易收斂到某個局部最小值上，因此不同於單純地將所有元素初始為 1，或課本建議的 sqrt(a/d)+salt，直接使用 Random.nextGaussian()產生平均為 0，標準差為 1 的常態分佈亂數，再乘上 0.25 使標準差變為 0.25，作為每個元素初始值，程式碼中的 means 並沒有使用到，最後分別以<U, i, j, value>、<V, i, j, value>的格式產生初始矩陣存到/U_0、/V_0 底下。

4.) **IterationU.java**

根據課本使用的演算法，每一輪對 U 的一條 row 進行更新，使得整體的 RMSE 降低。假設我們每次把元素 u_{rs} 更新為 x，而 P 為更新完後的 UxV 矩陣，則 x 只會影響到 P 的 row r，$p_{rj} = \sum_{k \neq s} u_{rk} v_{kj} + x v_{sj}$，假設 m_{rj} 如為 M 當中非空的元素，便能計算出更新 u_{rs} 後造成 RMSE 的改變 $\sum_j (m_{rj} - \sum_{k \neq s} u_{rk} v_{kj} - x v_{sj})^2$，將前式進行微分，當微分等於 0 的時候可以求得 x 的最小值：$\frac{\sum_j v_{sj}(m_{rj} - \sum_{k \neq s} u_{rk} v_{kj})}{\sum_i v_{sj}^2}$

Map Function:

純粹將 U、V、M 三個矩陣檔案讀入，並將資料以<name-of-matrix, (i, j, value)>的型式送出，不過 V 矩陣必須送到每個 reduce function，而 U 和 M 會根據 user ID，也就是 row index 以 round-robin 的方式平均地分配給總量 REDUCERS*REPLICATION_FACTOR 的 reduce function。

Reduce Function:

每次 reduce 會用到 V 中所有的值，和 M、U 的部分 rows，透過一個迴圈將大小已知的 V 存到 Double[][]，M、U 則存到以 user ID 作為 key 的 HashMap<Integer, Double[]>，這裡必須判斷 M 中是否已存在 user，若無則用 put()新增一個項目，並配置 Double[]用來存其中一個 row，若有則用 get()將值存入，還須注意原本陣列檔案內 index 是從(1, 1)開始編號，不過陣列第一個元素 index 為 0，因此使用到檔案內 index 時都要先減 1。

主要計算部分需要 M、U 的各一個 row，和整個 V 來進行，首先判斷 M 中是否有值，有才須要納入 RMSE 的計算。上頁的算式與程式碼的對應關係如下$\text{lineCol} = \sum_{k \neq s} u_{rk} v_{kj}$， $\text{sum}_\text{j} = \sum_j v_{sj}(m_{rj} - \text{lineCol})$，$\text{sum}_\text{s} = \sum_i v_{sj}^2$，最後即可算出 row r 每個元素的更新值$\text{u}'_{rs} = \text{sum}_\text{j}/\text{sum}_\text{s}$。最後將 index 再加 1 回來，寫到輸出檔。

5.) **IterationV.java**

基本上演算法和 IterationU.java 一模一樣，不過這次要將 V 傳給每個 reduce function，v_{rs} 更新的 y 值則可用$\frac{\sum_i v_{ir}(m_{is} - \sum_{k \neq r} u_{ik} v_{ks})}{\sum_i u_{ir}^2}$表示，主要差別在一次處理一個 column，而非一個 row，此外，這個階段用到的 U 矩陣是由剛剛 IterationU 產生的，因此存取的資料夾編號為 iteration+1。

※算完每個 column 後， comment out 的程式碼用來計算 M 中每個非空元素，與更新後的 P 中元素的 Squared Error：$(m_{rs} - \sum_{d \neq r} u_{rd} v_{ds})^2$，並將結果存到 RMSE-i-r-#####。

6.) **IterationUSol2.java、IterationVSol2.java**

主要計算部分基本上沒變，以 IterationUSol2.java 為例，和 Solution 1 的差異在每個 reduce function 都會用到的 V，不是透過 mapper 讀取 HDFS 上輸入檔的 splits，而是用 getVMatrix()，讓 reducer 直接將資料從 HDFS 讀到記憶體中，如此可以減輕 mapper 的負擔。這種方法每個 reduce function 只負責 1 筆 user 的資料，因此用 Double[]而不像 Solution 1 需先用 HashMap 處理資料。

2. FLOW CHART

藍線為程式運行流程與輸出檔案，綠線為輸入檔案，i 初始為 0。