

# Term\_Project\_Group23

## Topic: Recommendation System: Item-item Collaborative Filtering

107062103 王依婷

Final Project 要實作的是 recommendation system，以下是基本分的部分，也就是計算電影與電影之間的 similarity。

- **Datasets**

使用的是助教 pdf 中給的網址中的 MovieLens Latest Datasets


( small )，共有 9724 部電影，610 位使用者，取用資料是其中的 rating.csv，內容包含使用者 userID，這位 user 所評分的電影 movieID 和評分 rating，還有並不會用到的 timestamp。

- **Code explanation**

- 讀檔案 ( readFile )

首先把下載下來的 rating 檔案以 `sc.textFile` 讀入，因為檔案的格式是用逗號將資料分隔的 csv，，因此讀入後，用第一個 mapper -- `readFile` 以逗號為斷點，把資料分開並存到 list 中。接著將分好的資料，按照我想要的資料結構 ( `movieID, [userID, rating]` ) 的格式回傳。以 `movieID` 當作 key 是因為要做的是 item-item 版本，所以用電影來分類資料對於後面的操作比較方便。要注意的是檔案中第一列是用來標明該行的內容為何，會是字串，並不是需要使用的資料，因此在回傳前會先用 `isdigit()` 判斷，如果不是數字的話會回傳 `None`。

ratings



userId	movieId	rating	timestamp
1	1	4.0	964982703

```
# mapper1
def readFile(line):
    result = line.split(',')
    if (not result[0].isdigit()):
        return
    else:
        return int(result[1]), [[int(result[0]), result[2]]]
        # movieID, [userID, rating]
```

#### - 篩選並合併資料

經過第一個 mapper 後，用 filter 把回傳 None 的那列篩掉，再將同一部電影由不同使用者所給出的所有 rating 使用 reduceByKey 合在一起。用到的 reducer 就是簡單地將相同 key 的 value 結合在一起。(雖然 reducer 很簡單，但因為後面還會重複使用，還是把它寫成了一個 function )

```
def reducer1(a, b):  
    return a+b
```

所以資料會變成( movieID, [ [userID, rating], [userID, rating], ...]) 的樣子。為了維持資料的順序性，在做完 reduce 之後會用 key 做 sort。

```
m1 = file.map(readFile).filter(lambda x: x != None).reduceByKey(reducer1).sortByKey(True)
```

#### - 計算 mean rating 及相關計算 ( Calculate )

接著，對於每一部電影（每一列資料）所收到的所有 rating，計算平均值，並把每位使用者所給的 rating 扣掉平均值，存下來，也順便計算 similarity 會用到的分母的值 -- subMeanSqr，也就是該電影中每位使用者所給的 rating 扣到 mean 平方後相加，最後開根號。根據公式，之後的計算會用每位使用者對於同電影的評分算 similarity，所以這次把 userID 當作 key，方便後面計算使用，value 則是包含很多 [ movieID, rating to this movie – mean, subMeanSqr ]的資料組。計算完之後一樣用 reduceByKey 把相同使用者的 rating 資料結合在一起。

```
def Calculate(line):  
    # line = (movieID, [userID, rating])  
    total = 0  
    subMean = []  
    subMeanSqr = 0  
    merge = []  
    for person in line[1]:  
        total += float(person[1])  
    mean = total / len(line[1])  
  
    for person in line[1]:  
        subMean.append(float(person[1]) - mean)  
        subMeanSqr += pow(float(person[1]) - mean, 2)  
    subMeanSqr = pow(subMeanSqr, 0.5)  
  
    for i in range(len(line[1])):  
        merge.append((line[1][i][0], [[line[0], subMean[i], subMeanSqr]]))  
        # userID, [movieID, rating to this movie - mean, meanSquare]  
    return merge
```

## - 計算 similarity ( 部分 )

最後，計算 similarity。對於同一使用者，將他有評分電影分兩個組（C 所有有評分的電影 取 2）來依照公式計算。這裡要注意的是，如果這位使用者在其中一部電影發生他所給的 rating 和該部電影 mean rating 相同的狀況，就會讓計算時的分母變成 0，產生錯誤，因此在計算之前會先判斷如果分母等於 0 的話，就直接將該處的 similarity 設成 0。計算完之後，這時不需要 user 的資料了，只回傳 ((movieID1, movieID2), similarity) 這樣的資料。

```
def Similarity(line):
    # line = userID, [ [movieID, rating-mean, meanSquare], [movieID, rating-mean, meanSquare], ... ]
    tmp = []
    for i in range(len(line[1])):
        for j in range(i + 1, len(line[1])):
            length = line[1][i][2] * line[1][j][2]
            if length == 0:
                tmp.append((line[1][i][0], line[1][j][0]), [0])
            else:
                tmp.append((line[1][i][0], line[1][j][0]), [line[1][i][1] * line[1][j][1] / length])
    # (movieID1, movieID2), {sim}
    return tmp
```

## - 計算 similarity ( 加總 )

再將所有相同兩部電影之間的 similarity reduce 成同一筆資料，簡單用 mapper 加總在一起，最後用 sortByKey 將資料以 movieID 大小順序排好，把資料轉成 list 後寫到 outputFile 中儲存。

```
m3 = m3.map(Add).sortByKey(True)
m3 = m3.collect()
outputfile = open("OutputFile.txt", "w")
for i in range(len(m3)):
    outputfile.write("(" + str(m3[i][0][0]) + ', ' + str(m3[i][0][1]) + ') : ' + str(m3[i][1]))
    outputfile.write('\n')
outputfile.close()
```

最後的資料格式 (item, item) : similarity

```
(1, 2) : 0.13964884936019545
(1, 3) : 0.11384956862543402
(1, 4) : 0.032658300302978296
(1, 5) : 0.07622958231949363
(1, 6) : 0.037290999300725255
(1, 7) : 0.06426817569923113
(1, 8) : 0.08598649986291004
(1, 9) : 0.04138877803276812
(1, 10) : -0.00818400806640569
(1, 11) : 0.0598170707122273
(1, 12) : 0.03739629135898846
```

接著是 advance 的部分。

#### - 讀資料

首先因為要預測 rating 是以使用者為主，所以換個方式將資料讀取進來，結構改為 ( userID, [movieID, rating] )，並且和基本部分做一樣的篩選與合併。

```
def readAgain(line):
    result = line.split(',')
    if (not result[0].isdigit()):
        return
    else:
        return int(result[0]), [[int(result[1]), result[2]]]
        # userID, [movieID, rating]
```

#### - 找到要預測的電影

如果那部電影該使用者已經評分了，就不需要再預測，因此要篩掉符合這種情形的電影。在 mapper function 中把 user 所有給 rating 過的電影和 rating 都放到一個 list 中，也另外把那些 movieID 放到另一個 list 中方便後續判斷使用，接著把所有電影中，沒有被該 user 評過分的那些放到再另一個 list 中。把三個 list 放在 value 的部分，userID 為 key 回傳。

```
def findPredict(line):
    predict = []
    haveRating = []
    haveRating_id = []
    flag = False
    for item in line[1]:
        haveRating_id.append(item[0])
        haveRating.append(item)
    for i in range(1, movie+1): # movie = number of movie
        if i not in haveRating_id:
            predict.append(i)

    return line[0], [predict, haveRating_id, haveRating]
    # userID, [[movies that need to be predicted], [predicted]]
```

#### - 資料處理

接下來的部分我用 collect()把 RDD 的結構轉為 list 做處理，之後會放到 mapper function 中做計算。對於每位 user，在所有需要被預測的電影中，從上面做基本部分所算出的 (( item, item ), similarity ) pair 中，找到符合那部電影和 user 有評過的電影之間的 similarity，並且挑選 similarity 最大的十個來做計算。接著，因為在計算預測值時需要

user 對這十部電影的 rating，所以也另外找出來，並且和 similarity 放在一起（計算時不需要 movieID，所以就沒有保留）。

```
m5 = m5.collect()

array = []
for line in m5:
    data = []

    for movie in line[1][0]:
        # m3 = ((movieID1, movieID2), sim)
        m6 = m3.filter(lambda x: (x[0][0] == movie and x[0][1] in line[1][1]) or \
                                (x[0][1] == movie and x[0][0] in line[1][1])).sortBy(lambda x: -x[1])
        m6 = m6.take(10)

        tmp = []
        for pair in m6:
            for r in line[1][2]:
                rating = []
                if r[0] == pair[0][0] or r[0] == pair[0][1]:
                    tmp.append((float(r[1]), pair[1]))
                    break
            data.append((movie, tmp))
        array.append((line[0], data))
```

最後把處理好的資料轉換成 RDD。

```
m7 = sc.parallelize(array)
# (userID, [(to be predicted, [(rating, sim), (rating, sim), ...]), (...)])
```

## - 計算 Rating Predictions

按照公式計算出預測值，要注意的是如果在 top10 中有取到負值的 similarity，就將它捨棄不做計算，因為這樣算出來會是負數，最後也要避免掉除以零的狀況（比如如果該 user 對所有電影都以評分的話）。

```
def Predict(line):
    tmp = []
    n = 0
    d = 0
    for predict in line[1]:
        movie = predict[0]
        for rating in predict[1]:
            if rating[1] > 0:
                n += rating[0] * rating[1]
                d += rating[1]
        if d > 0:
            tmp.append((line[0], movie), n / d)
    return tmp
```