Item- based Recommendation System for Massive & Sparse Dataset

Github

Nifflerbot

ShiuanShr

改良式Item- Based 推薦系統- 針對稀疏巨量資料之算法設計

專案價值:

1. 完全使用MapReduce方式，並完全使用使用底層資料結構RDD自行搭建算法，在不使用外部Package前提，撰寫推薦系統，以達技術上創新與客製化地解決問題。

2. 建立item- based recommendation，並進行演算法設計- Baseline Redefinition

解決問題:

(3-1) 本專案為Mining of Massive Data 技術應用之一，可解決實務上巨量數據、稀疏(Sparse)之情境下，針對data streaming 去做推薦系統與預測使用者評分。

(3-2) 因為指使用底層架構與資料結構，並未使用外來package與model- based算法，該算法設計與其他算法具有高度的擴充性、延展性。

(3-3) 為item- based recommendation 設計，此外，達成技術之創新與自行定義算法的附加優勢。

Data Type & Difficulty

- **Sparse Matrix & Massive Distributed Dataset　稀疏矩陣與巨量資料**

巨量數據測資提供檔ml-lastest，含有rating紀錄之約20萬部不同電影與600多位不同user構成之評分紀錄，該數據為稀疏矩陣，並附有電影相關標籤(tags)與(categories) csv file，若不使用mapreduce計算任兩位使用者之推薦電影分數預測，計算空間複雜度為C(200000,2)*C(600,2)，試改良之。

此外，額外提供小型測資於檔案addition-testing-data folder中，附有3類測資，可達不同測試目標，詳見Github資料夾。

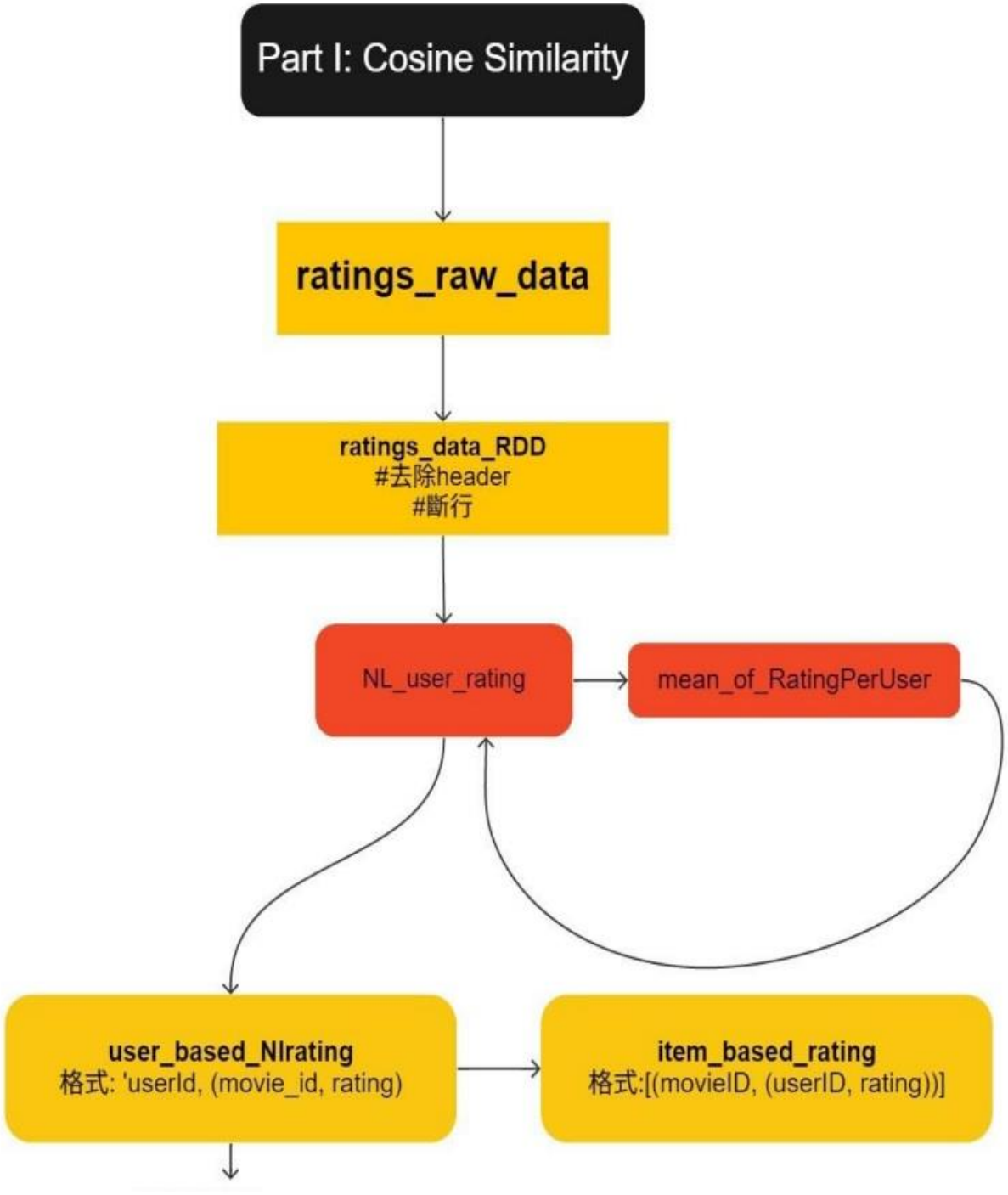
- **Result**

該專案受到教授與助教好評，屬於完成度高之專案，期末專案未列全班最高(滿分)，後續仍能以此為基礎進行開發，附上評分證明如下。

Term Project	○	-	01-06 00:00	✓ 100

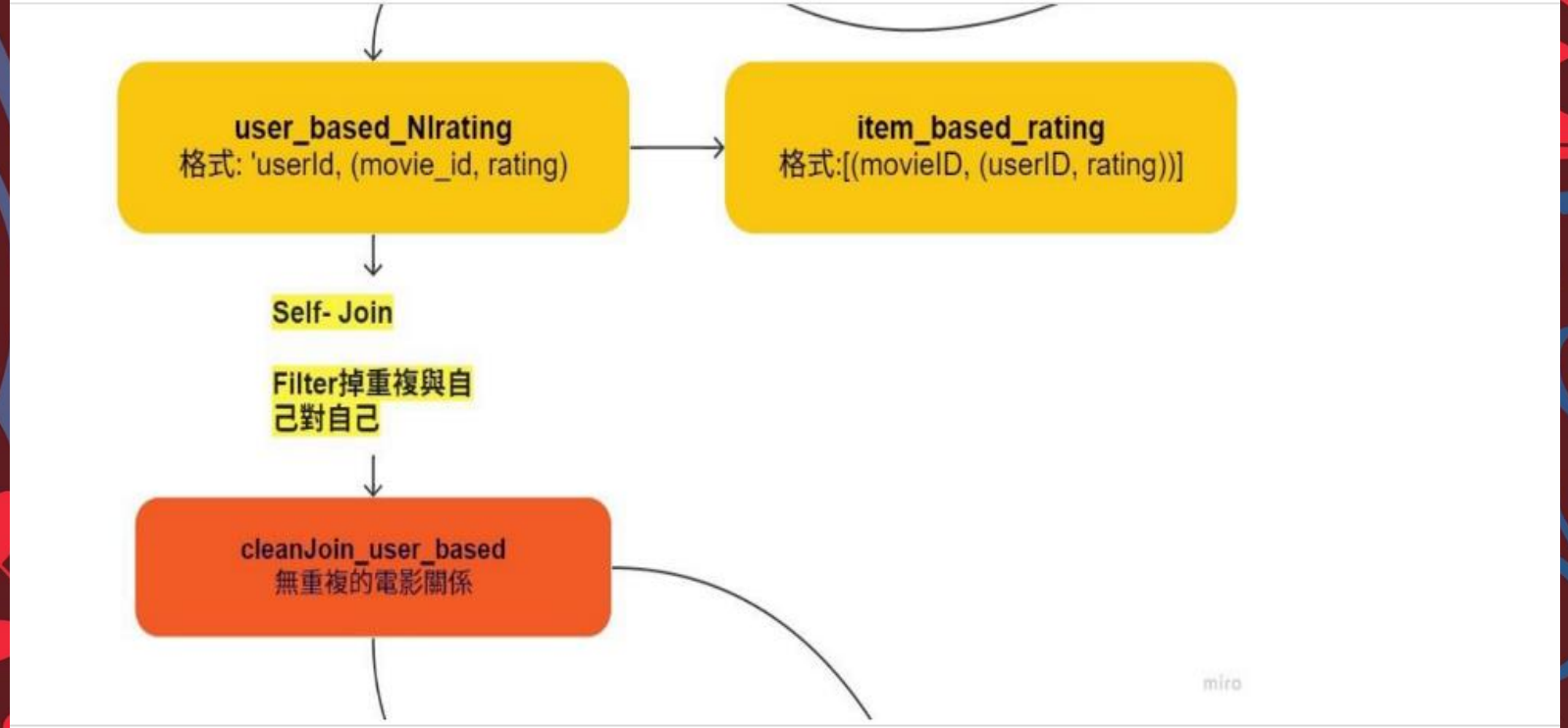
Cosine Similarity On Sparse Dataset

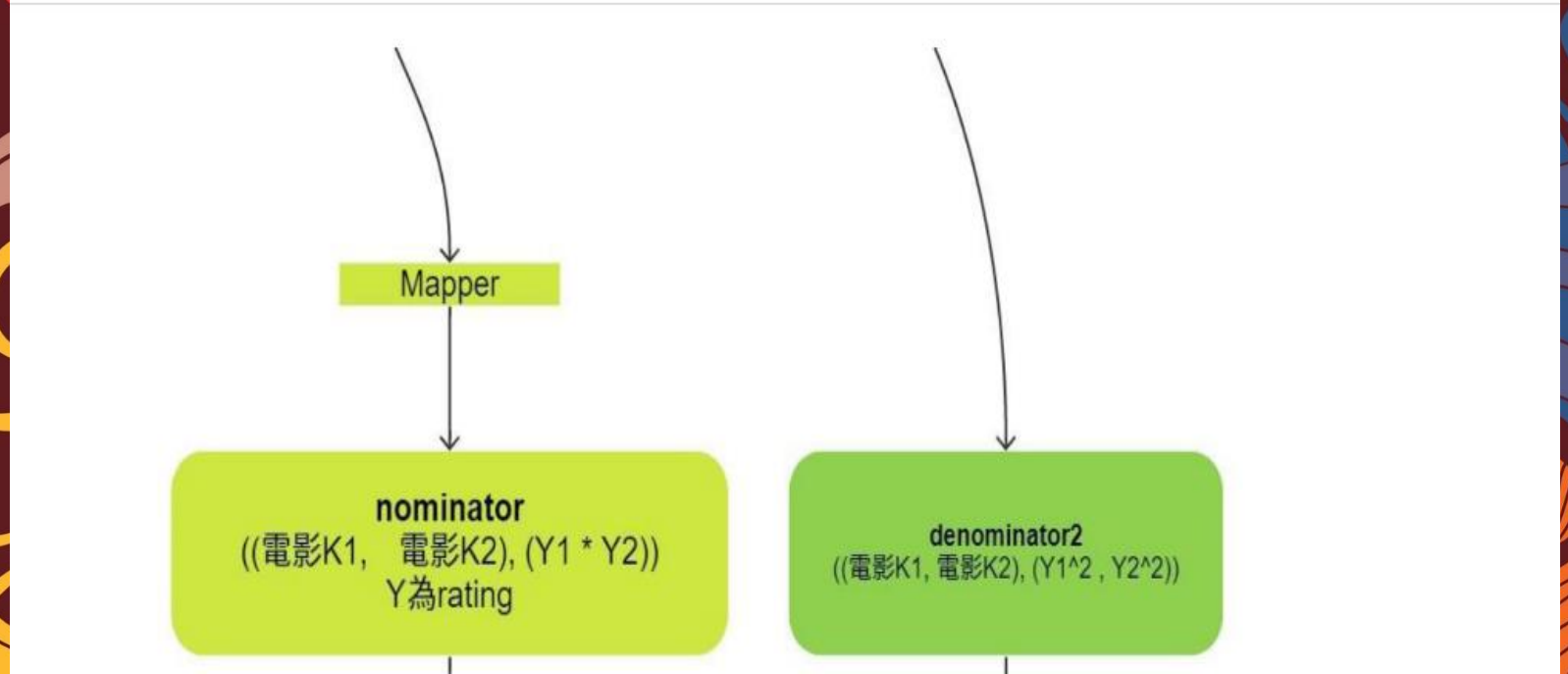
Algorithm Framework PART – (1-1)



Cosine Similarity On Sparse Dataset

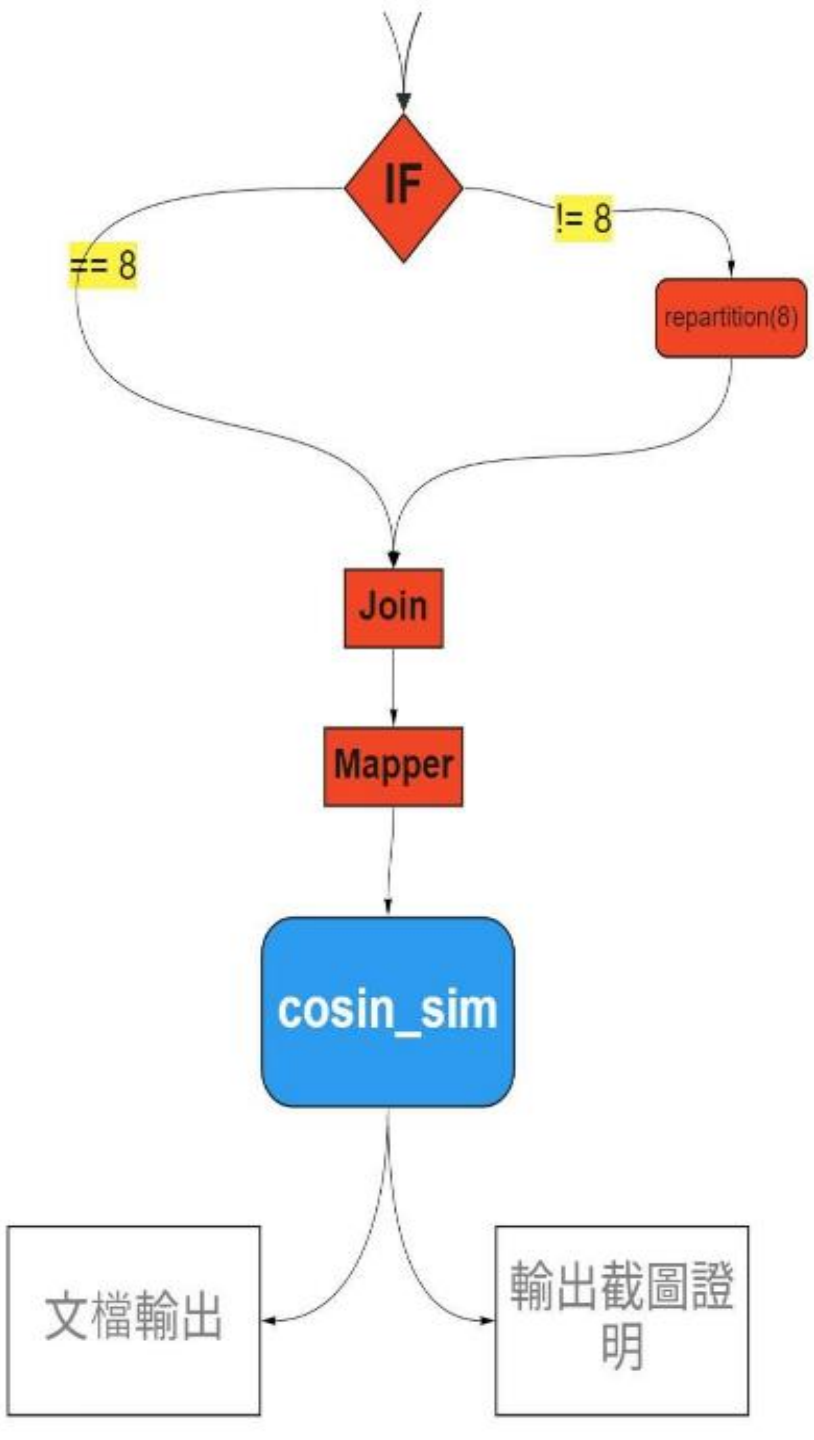
Algorithm Framework PART – (1-2)





Cosine Similarity On Sparse Dataset

Algorithm Framework PART – (1-3)



Cosine Similarity On Sparse Dataset

Algorithm Framework PART – (1-4)

Result Proof

Part I: (2)cosine similarity 輸出證明

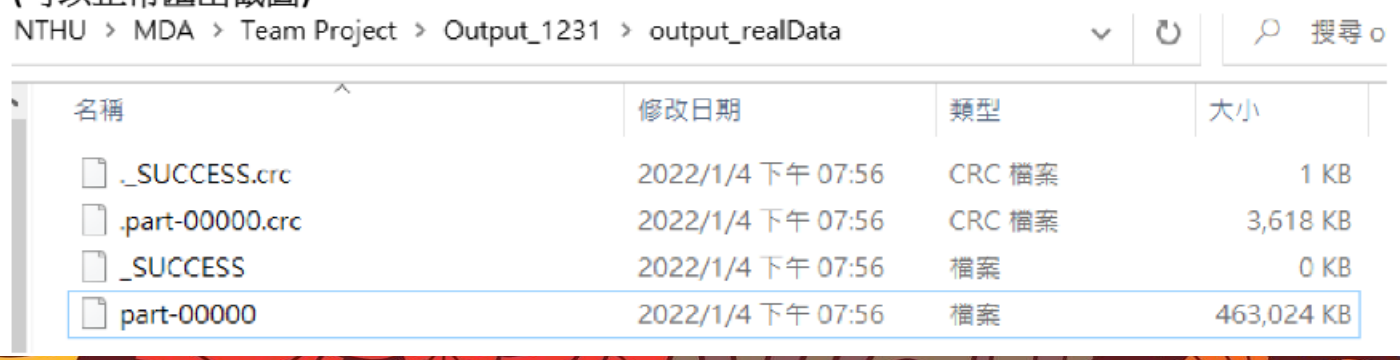
```
In [117]: cosin_sim = item_based_numerator_sum.join(denominator).map(lambda x:(x[0], x[1][0]/x[1][1]))

          cosin_sim.take(10)
```

```
Out[117]: [((3418, 7360), 0.4390684007697293),
           ((454, 1036), 0.22868583899781905),
           ((832, 8622), -0.666734404907775),
           ((1320, 1974), 0.05920895815149847),
           ((1393, 1981), 0.8971914258902179),
           ((4643, 36519), 0.2933442257114883),
           ((838, 6620), -0.7964850190885081),
           ((2916, 5410), -0.9138291406518673),
           ((3928, 5462), -0.999999999999994),
           ((3979, 89343), 0.9999999999999982)]
```

PS: 若是需要匯出，**ipynb** 檔案內有提供程式碼匯出，可用 **FireFox** 打開

(可以正常匯出截圖)



Part II Recommendation Sys Design

```
def recommendation_sys(specificed_user_id,num_recomd =500, top_i_in_cosine = 10):
    watched_movie_record  = user_based_Nlrating.filter(lambda x :(x[0] == specificed_user_id))
    #從user_based_Nlrating filter出該user的評分紀錄
    print(f'stage 1 completed')
    """
    目標是從cosin_sim中只取出會用到的的關係，加速運算速度,因此只針對 ((k1, k2), cos-sim),
    K1或是k2其中一部電影是看過，另一部是沒有看過的篩選出來
    """
    #2.  從cosine similarity (cosin_sim)結果中取出key中有該user看過電影的cos - sim
    print(f'stage 2 completed')
    ##2-1. 將該user有看過電影裝在list中
    watched_movie_list = watched_movie_record.map(lambda x : (x[1][0])).collect()

    ##2-2.   從cosine similarity 結果中取出key中有該user看過電影的cos，放在cosine sim 中
    #取出cosin_sim中key pair任一值為該user看過的電影，且另一部電影為沒看過的cosine 關係
    cosine_index = cosin_sim.filter(lambda x : ((x[0][1]) in watched_movie_list and x[0][0] not in watched_movie_list) or ((x[0][
    cosine_index.take(1)
    cosine_index.cache()
    print(f'stage 3 completed')

    """
    select out the required cosine sim relatonship =: Default top 10
    """
    #目標是將key pair整理好，
    #.map(lambda x : (x[0][1],(x[0][0], x[1]))) 部分則是將其mapping 成以有看過的電影為single key, value= (沒看過的電影ID, cos - sim)
    cs_watched_unwatch   =cosine_index.map(lambda x : (order_rddkey(x[0]),x[1])).map(lambda x : (x[0][1],(x[0][0], x[1])))
    print(f'stage 4 completed')

    #cs_watched_unwatch format原本為(看過的電影ID (沒看過movie_ID，cosine))
    #透過此mapper將其轉化成 ((沒看過movie_ID,看過的電影ID),cosine )
    tt1 = cs_watched_unwatch.map(lambda x : (int(x[1][0]), (int(x[0]), x[1][1])))
    tt1.cache()
    print(f'stage 5 completed')

    #先以未看過電影ID為key, group起來, group起來部分會是以tuple形式組組成的RDD，因此要轉list
    stupid_python = tt1.groupByKey().map(lambda x : (x[0], list(x[1])))
    stupid_python.take(1)
    print(f'stage 6 completed')

    #找出cosine互相為前十個的關係
    top_10__call = stupid_python.map(lambda x : (x[0],top10(x[1],top_i_in_cosine)))
    print(f'stage 7 completed')
    top_10__call.take(1)
```

For Code Details:

Part II- Revised Baseline (2)

Add the basedline

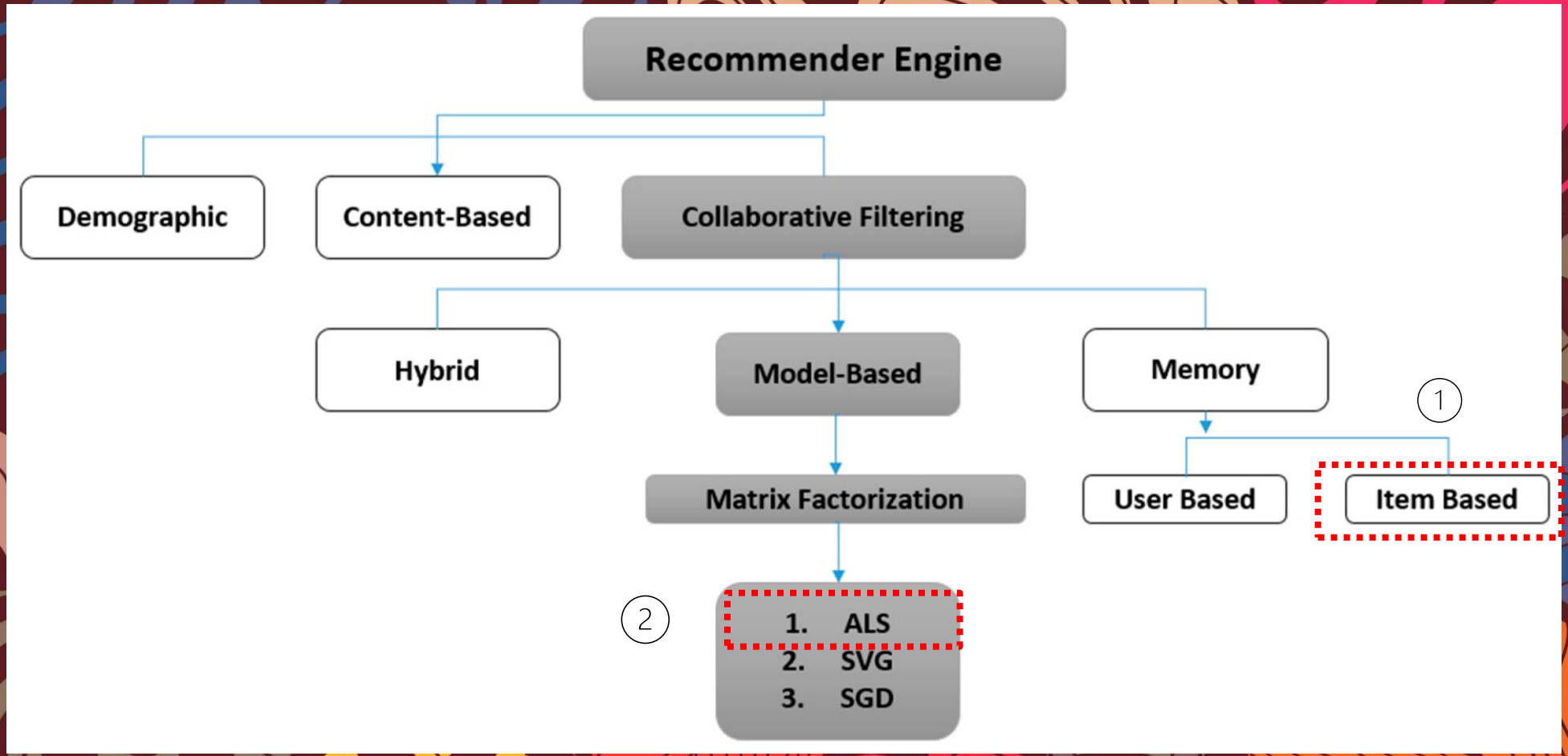
Which basedline?

- movie basedline (採用): 大眾普遍對某電影評分越低，我們預估該user對其評分越低 (Normailized 後的評分會有負值表示不受大眾看好)
- user basedline : 該user討厭特定類型電影，我們預估期其會給同類型電影分數也越低 (baseline為該user平均評分)

  缺點：若是只有評分太少(例如都是平價兩個3分) 則我們容易讓其預測評分 太高

並且確保加上based後不會低於0分或是高於5分

Part II-
ALS Model Improvement



Part II-
ALS Model Improvement

利用ALS找出擁有最小EMSE的 Rank 值

Rank值表示latent factor數量

```
for rank in ranks:

    #先將training set用來fit model, 通常iterations <20、 seed隨便、lambda 是 正則化的參數(具體我不知)
    #模型會丟出model參數 : DataType為<pyspark.mllib.recommendation.MatrixFactorizationModel的matrix

    ALS_model = ALS.train(training_set, rank, seed=seed, iterations=iterations, lambda_=lr)
    print('completed model')

    """
    validation_set
    """
    #使用fit好的模型對validation set作預測，並map成以user, movieID作key
    #(Foramt: (user id, movie_id) )
    predictions = ALS_model.predictAll(validation_set).map(lambda x: ((x[0], x[1]), x[2]))
    #predictions format= [Rating(user id, movie_id, Predicted rating),....]  ,
    print('predictions completed')

    #用map整理好，以 (user id, movie_id)為key，等等跟真正的rating做inner join
    temp = validation_set.map(lambda x: ((int(x[0]), int(x[1])), float(x[2])))
    predictions_result = predictions.join(temp)
    print('Join completed')

    #計算與實際rating的RSME，目標是取出最小RSME的Rank當作latent factor數量
```

Final Part
Item-Based +ALS

Testing Set

使用test_set驗證，得到predictions_result: 0.9089997788892856

```
ALS_model = ALS.train(training_set, Desired_rank, seed=seed, iterations=iterations, lambda_=lr)
# 模型會丟出model參數 : DataType為<pyspark.mllib.recommendation.MatrixFactorizationModel的matrix

##對testing set進行預測
#input=test_set(Foramt: (user id, movie_id) )
predictions = ALS_model.predictAll(test_set).map(lambda x: ((x[0], x[1]),x[2]))
# 模型會丟出DataType為<pyspark.mllib.recommendation.MatrixFactorizationModel的matrix
#[Rating(user id, movie_id, Predicted rating),....]  ，用map整理好，以 (user id, movie_id)為key，等等跟真正的rating做inner join

#將真正的評分與預測分數做inner join
temp = test_set.map(lambda r: ((r[0], r[1]), float(r[2])))
predictions_result = temp.join(predictions)

#format: ((使用者, 電影), (真實分數, 預測分數))

#計算預測分數與實際分數的Vavg(差的平方)
error = math.pow(predictions_result.map(lambda r: (r[1][0] - r[1][1]) ** 2).mean(), 0.5)
print(f'test set ERROR: {error}')
```

將預測分數與我先前計算出來預測的分數作平均

*將先前**PartII**指定推薦推薦之結果**(predcited rating)**與此模型萬出之預測**rating**做平均值，作為優化。*

IMPROVED ITEM- BASED RECOMMENDATION SYSTEM FOR MASSIVE & SPARSE DATASET

改良式ITEM- BASED 推薦系統- 針對稀疏巨量資料之算法設計

END