# Term Project Report Group63

## Project

1. Recommend System
2. dataSet: comes from https://grouplens.org/datasets/movielens/
   (https://grouplens.org/datasets/movielens/)
   ml-latest-small.zip
3. output: (userId, movieId) predicate rate

## implementation

1. 讀取資料，並且使用movies計算總共有幾個movie在計算最後的結果時才知道有幾個 movies

```
movies = set()
def readData(inputFile):·
    global movies
    with open(inputFile) as fp:
        data = fp.readlines()

    data = data[1:]
    for i in range(len(data)):
        data[i] = data[i].strip().split(',')[:3]
        data[i][0] = int(data[i][0])
        data[i][1] = int(data[i][1])
        movies.add(data[i][1])
        data[i][2] = float(data[i][2])
    return data

data = readData('ml-latest-small/ratings.csv')
```

2. Systme setting

```
import pyspark
from pyspark import SparkContext, SparkConf
import math
import os


conf = SparkConf().setMaster("local[*]").setAppName("Project").set("spark.
sc = SparkContext(conf=conf).getOrCreate()
```

3. 需要幾個最相似的item來計算推薦值

```
topK = 10
```

4. input data是[userId, movieId, score]
透過第一個map去掉userId,只剩下movieId,再將相同的movieId聚集起來
再透過mapValue取的movies的平均分數

```
inputMat = sc.parallelize(data)
point_for_each_item = inputMat.map(lambda s:(s[1], s[2])).groupByKey() # u
avg_point_for_each_item = point_for_each_item.mapValues(lambda s:sum(s)/le
```

5. 將input 轉換為 (movieId, (userId, score))的形式
再將上述結果與前面計算的movieId平均值相減
就可以得到扣除平均值的結果

```
point_minus_avg = inputMat.map(lambda s:(s[1], (s[0], s[2]))) # type (movi
point_minus_avg = point_minus_avg.join(avg_point_for_each_item) # type(mov
point_minus_avg = point_minus_avg.map(lambda s:((s[0], s[1][0][0]), s[1][0
```

6. 計算cosim similarity的方法
首先將item1與item2有的value平方相加開根號
再根據暴力法找看過相同的movie並且相乘在相加

```
# calculate cosine similarity
def cosim(_input):
    item1_L2_nor = 0
    for ele in list(_input[0][1]):
        item1_L2_nor += (ele[1]**2)
    item1_L2_nor = item1_L2_nor ** 0.5

    item2_L2_nor = 0
    for ele in list(_input[1][1]):
        item2_L2_nor += (ele[1]**2)
    item2_L2_nor = item2_L2_nor ** 0.5

    rxi_ryi = 0
    for ele1 in list(_input[0][1]):
        for ele2 in list(_input[1][1]):
            if(ele1[0] == ele2[0]):
                rxi_ryi += (ele1[1] * ele2[1])
                break
    if item1_L2_nor * item2_L2_nor != 0:
        return ((_input[0][0], _input[1][0]), rxi_ryi/(item1_L2_nor * iten
    else:
        return ((_input[0][0], _input[1][0]), 0.0)
```

7. point_minus_avg.map將資料轉換為(movieId, (userId, score))
   透過item_score.cartesian(item_score)得到所有的pair
   利用filter刪除 (2, 1)這種case，防止(2, 1),(1, 2)重複計算
   all_pair_for_item.map(cosim)得到所有的movie pair的similarity

```
item_score = point_minus_avg.map(lambda s:(s[0][0], (s[0][1],s[1]))).group
all_pair_for_item = item_score.cartesian(item_score).filter(lambda s: s[0]
all_pair_score_for_item = all_pair_for_item.map(cosim) # type((movieId1, m
```

8. all_pair_score_for_item.map(...)將所有資料轉為(similarity, ((movie1, movie2), (movie2, movie1)))
   透過all_pair_score_for_item.flatMapValues 將資料轉為(similarity, (movie1, movie2))，如此一來也能得到(2, 1)的similarity
   最後在map成 ((movie1, movie2), score)的形式

```
# type (score, ((movie1, movie2), (movie2, movie1)))
all_pair_score_for_item = all_pair_score_for_item.map(lambda s:(s[1], ((s
# type ((movie1, movie2), score)
all_pair_score_for_item = all_pair_score_for_item.flatMapValues(lambda s:
```

9. 將movie依照similarity排序
   all_pair_score_for_item.map將item轉為(movieId, sorted similarity)
   sorted similarity是list每個element為(item, score)並且根據score的大小進行排序
   ele當中為(movie, similarity)

```
def sort_similarity(_input):
    ele = list(_input[1])
    ele = sorted(ele, key=lambda s:s[1], reverse=True)
    return (_input[0], ele)


similarity_score_for_each_item = all_pair_score_for_item.map(lambda s:((s
```

10. 得到user沒看過的movieId
    user對movie的pred

```
def get_unRating(_input):
    global movies
    movie_list = list(movies)
    for ele in list(_input):
        movie_list.pop(movie_list.index(ele[0]))

    return movie_list

def get_rate(_input):
    label = dict()
    for ele in list(_input[1][0][1]): ## user have seen movie
        label[ele[0]] = ele[1]
    count = 0
    pred = 0
    weight_list = []
    for j in range(len(_input[1][1])):
        if _input[1][1][j][0] in label.keys() and _input[1][1][j][1]>0:
            pred += (_input[1][1][j][1] * label[_input[1][1][j][0]])
            weight_list.append(_input[1][1][j][1])
            count += 1
            if(count == topK):
                break
    weight_sum = sum(weight_list)
    if weight_sum == 0:
        return ((_input[1][0][0], _input[0]), 0)
    else:
        return ((_input[1][0][0], _input[0]), pred/sum(weight_list))
```

11. inputMat.map(...)得到user看過的所有電影 type為type(user ,list of (movie, score))
    透過get_unRating得到user沒看有過的movies，透過flatMapValues將type轉為 (user, unseen movie)
    user_unrating.join(user_rating)得到user沒看過move和看過的movie list與rating
    user_rating_and_unrating.map(...)將unseen movie當成key
    user_rating_and_unrating.join(...) 得到了unseen movie的similarity score
    最後透過total_info.map(get_rate)得到((userId, movieId), pred)

```
user_rating = inputMat.map(lambda s:(s[0],(s[1], s[2]))).groupByKey() # ty
user_unseen = user_rating.mapValues(get_unRating).flatMapValues(lambda s:s
user_rating_and_unrating = user_unseen.join(user_rating) # type(userId, (u
user_rating_and_unrating = user_rating_and_unrating.map(lambda s:(s[1][0],
total_info = user_rating_and_unrating.join(similarity_score_for_each_item)
user_unseen_rate = total_info.map(get_rate) # type(userId, (movieId, pred)
res = user_unseen_rate.sortByKey().collect()
```

12. 將檔案寫成file

```
with open("Outputfile.txt", 'w') as fp:
    for i in range(len(res)):
        fp.write("({0},{1}),{2}\n".format(res[i][0][0], res[i][0][1], res
```