1. Introduction

這次作業目標是透過 user-item 的紀錄去推薦指定 user 的 items，過程中 Typical RecSys Methods 有嘗試 UCF-s、UCF-p、ICF-s、ICF-p、MF、FM，NN-based RecSys Methods 嘗試 FNN，最後依照 FNN 的模型架構下去修改成我自己的 NN model，主要使用 MovieLens 下去做實驗，後面有試圖嘗試 Yelp，而最後在 MovieLens 這個資料集上，我的模型在 recall 和 ndcg 上有跑出較好的結果。

2. Methodology

我提出的方法主要是從 FNN 下去做改良，步驟分成：將資料匯入、資料預處理、分割資料集、訓練模型、預測出推薦項目。

1. 資料匯入+預處理:

MovieLens

互動資料:

	userId	movieId	rating	timestamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596
...	...	...	...	...
99995	880	476	3	880175444
99996	716	204	5	879795543
99997	276	1090	1	874795795
99998	13	225	2	882399156
99999	12	203	3	879959583

100000 rows × 4 columns

Yelp

	userId	businessId	rating
2	2	186	5
3	2	205	5
4	2	209	4
5	2	461	3
6	2	499	2
...	...	...	...
198392	16238	13256	5
198393	16239	2580	4
198394	16239	10525	1
198395	16239	12714	4
198396	16239	13503	1

188456 rows × 3 columns

這兩個資料集都有做資料上的檢查，在 MovieLens 上，user-movies 的互動都有大於 3，所以沒做刪減，和原始資料及一樣為 100000 筆資料。而 Yelp 資料集經過檢查發現有部分 user 的互動小於三筆，所以將原本的 198397 筆資料刪減為 188456 筆資料。

建立 user-item 的關聯矩陣，如下圖(MovieLens):

movieId	1	2	3	4	5	6	7	8	9	10	...	1673	1674	1675	1676	1677	1678	1679	1680	1681	1682
userId																					
1	5.0	3.0	4.0	3.0	3.0	5.0	4.0	1.0	5.0	3.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	4.0	3.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
939	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
940	NaN	NaN	NaN	2.0	NaN	NaN	4.0	5.0	3.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
941	5.0	NaN	NaN	NaN	NaN	NaN	4.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
942	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
943	NaN	5.0	NaN	NaN	NaN	NaN	NaN	NaN	3.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

943 rows × 1682 columns

轉為矩陣型式然後作資料集的分割(training:80% testing: 20%)，如下圖(MovieLens):

```
Train data:

array([[4., 0., 0., ..., 0., 0., 0.],
       [4., 0., 0., ..., 0., 0., 0.],
       [4., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

```
Test data:

array([[0., 5., 0., ..., 0., 0., 0.],
       [4., 0., 0., ..., 0., 0., 0.],
       [4., 0., 3., ..., 0., 0., 0.],
       ...,
       [5., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [5., 0., 0., ..., 0., 0., 0.]])
```

資料集的部分做到後面有發現兩個問題:

(1) 原始資料集 Nan 的地方應該也是要使用方法預測出來的，但是後來沒有解決這個問題，替代方案是直接把 NaN 的地方設定成 0.。

(2) 如 1.提到，我把值設定成 0.了，在 MovieLens 上 user 少的形況下還可以應付，但 Yelp 直接變 10 倍以上的 user 造成資料過度稀疏的問題，必須把 user. item 各自的資訊(其他提供的.dat 檔案)一起考慮進去預測出來(也就是解決 1.的問題)，但最後時間來不及，所以後面主要針對 MovieLens 繼續說明，Yelp 資料集成果沒有做出來。

2. 訓練模型

參數設定(皆和 FNN 原論文方法設定相同)
epochs=30, batch_size=16, optimizer='adam'
另外加入的參數設定
Dropout rate=0.3
K-Fold Cross Evaluation
K=5

NN 實際結構如下圖:

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_30 (Dense)            (None, 400)               673200

 dropout_20 (Dropout)        (None, 400)               0

 dense_31 (Dense)            (None, 200)               80200

 dropout_21 (Dropout)        (None, 200)               0

 dense_32 (Dense)            (None, 1682)              338082

=================================================================
Total params: 1,091,482
Trainable params: 1,091,482
Non-trainable params: 0
_________________________________________________________________
```

我的模型和 FNN 原論文一樣使用三層的模型，修改的地方為一把 acitvation function 前兩層改為”relu”、最後一層改為”linear”，然後在中間加入 dropout rate。

後來有想到還可以修改隱藏層的 unit 數量，所以有做微調，新的 NN 模型架構如下:

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_45 (Dense)            (None, 16)                26928

 dropout_30 (Dropout)        (None, 16)                0

 dense_46 (Dense)            (None, 8)                 136

 dropout_31 (Dropout)        (None, 8)                 0

 dense_47 (Dense)            (None, 1682)              15138

=================================================================
Total params: 42,202
Trainable params: 42,202
Non-trainable params: 0
_________________________________________________________________
```

可以發現用到的變數減少很多，整體的運算速度也有明顯加快，rmse 的分數雖然提高了，但是 recall 和 ndcg 都有比較高的分數。

後面會針對有嘗試的方法做說明。

A. FNN

參數設定
epochs=30, batch_size=16, optimizer='adam'

因為我設計的模型和 FNN 是差不多的，所以就不再說明，呈現模型架構如下圖:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 400)               673200

 dense_1 (Dense)             (None, 200)               80200

 dense_2 (Dense)             (None, 1682)              338082

=================================================================
Total params: 1,091,482
Trainable params: 1,091,482
Non-trainable params: 0
_________________________________________________________________
```

B. UCF/ICF(Pearson correlation as similarity)

作業題目有要求用 5-Fold Cross Evaluation，但是我一直找不到要怎麼把它做使用，因為 user-based 和 item-based 都是用數學公式下去做計算的，不需要使用到 parameters。最後我決定把資料集分成 train 和 test 之後，把 test 的資料當成驗證下去做比較。

資料處理後如下圖:

```
data_table_train = pd.pivot_table(train_data,values='rating',columns='movieId',index='userId')
data_table_train.head()
```

movieId	1	2	3	4	5	6	7	8	9	10	...	1670	1671	1672	1673	1674	1675	1678	1679	1680	1681
userId																					
1	5.0	3.0	4.0	3.0	3.0	5.0	4.0	1.0	5.0	3.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	4.0	3.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 1656 columns

```
data_table_test = pd.pivot_table(test_data,values='rating',columns='movieId',index='userId')
data_table_test.head()
```

movieId	1	2	3	4	5	6	7	8	9	10	...	1656	1657	1659	1662	1666	1670	1673	1676	1677	1682
userId																					
1	NaN	NaN	NaN	3.0	NaN	NaN	4.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 1449 columns

然後和 FNN 不一樣的是，原始資料 NaN 的部分，在這邊我沒有把它設定成 0.，而是直接下去算，我覺得這也是 RMSE 成果會比 FNN 還要好的原因。

```
print("here are a list of 50 movies to recommend to a user who has liked 'userId=1'(pred)")
print(data_table_train.corr()[1].sort_values(ascending=False).iloc[:50])
# print(data_table_train.corr()[1])
print("here are a list of 50 movies to recommend to a user who has liked 'userId=1'(act)")
print(data_table_test.corr()[1].sort_values(ascending=False).iloc[:50])
# print(data_table_test.corr()[1])
```

而 correlation 的計算方式我是使用.corr()這個方法下去做計算，然後把

train 和 test 合併(如下表左圖)，可以得到指定 user 對其他 user 計算出來的 similarity 分數，之後再把計算為有 NaN 的值刪掉(如下表右圖):

userId	1_x	1_y
1	1.000000e+00	1.000000
2	-1.000000e+00	0.267822
3	NaN	0.112780
4	NaN	0.437048
5	4.575378e-01	0.442559
...	...	...
939	NaN	0.534390
940	8.685990e-01	0.299813
941	NaN	0.205616
942	-2.533726e-17	-0.190052
943	2.605512e-01	0.047938

943 rows × 2 columns

userId	1_x	1_y
1	1.000000e+00	1.000000
2	-1.000000e+00	0.267822
5	4.575378e-01	0.442559
6	5.339556e-01	0.266420
7	3.801751e-01	0.330458
...	...	...
937	-1.000000e+00	0.044499
938	9.859525e-02	0.300847
940	8.685990e-01	0.299813
942	-2.533726e-17	-0.190052
943	2.605512e-01	0.047938

[618 rows x 2 columns]

然後在計算 NDCG@10，是使用 ndcg_score(_,_, k=10)這個 function，我在寫報告的時候才注意到這個也要把數值轉成 binary 的型式，但是已經來不及了，所以我是用計算出來的數值下去計算的，之後會再把它轉成 binary 的型式下去算看看。

C. UCF(Cosine as similarity)

C 選項這個方法是我 HW3 第一個開始做的方法，也是 movie 有最完整推薦跑出來的，因為是第一個開始做也是花最久時間的一個模型，實際推薦結果部分截圖如下圖所示: (呈現內容為推薦某一 user 前 50 個相關的電影)

```
user: 6
The most similar users: [686 571 235 512 138 932 469 165  10 574 785  60 361 553 415 767 392 855
 522 312 412 321 613 645 884   8 272 267 409 583 766 383 379  21 172 794
 846 481 871 748 641 607 836 237 877 882  25 543 338 315]
user: 7
The most similar users: [415  29 512 573 185 132 333 350 469 768 118 835 474   2 647 138 332 354
 204 516  71 322  89 821 602 539  60 583 383  85 908  46 119 867 903 247
 123 674 645 514 665 375 552 540 710 634 235 210 694  10]
user: 8
The most similar users: [213 645  55 215 934 694 271 371 275 294  18  97 764 826 643 339 276 288
  99 903 538 251 862 892 210 622 117 378 177 272 344 600  59   1  94 676
 198   6 741 893 144 889 343 506 942 831 795  44 542 468]
user: 9
The most similar users: [308 474 303 650 450 429 269 533 655 407 592 758 406  43 796 870   7 537
 234  90 452 268 276  59  10 194 716  13   6 201 311 385 604 938 154 632
 866  64 675 331 904 448 545 826 144 771 143 466 102 921]
```

C 選項的方法為定義一個可以找出最相近使用者的 function(def find_the_most_similar_users)，首先找出指定 user 和其他 user 的關聯性，依照觀看過相同 movie 數是否大於 10 下去做判斷，其中找出看過相同 movie 的 function 定義為 find_common_movies()，如果觀看相同電影數大於 10 再進一步下去算他們之間的 movie similarity 分數(def cal_similarity_for_movie_ratings)，主

要 function 定義如下圖:

```
def find_the_most_similar_users(user, num, df,method):
    # Calculate the similarity between the user and other users
    similarities = []
    user_ids = []
    for other_user in df.userId.unique():
        if other_user == user:
            continue

        common_movies = find_common_movies(user,other_user,df)
        if len(common_movies)<10:
            sim = 0
        else:
            sim = cal_similarity_for_movie_ratings(user,other_user,common_movies,df,method)

        similarities.append(sim)
        user_ids.append(other_user)
#     print(similarities)

    # Find top n similar users
    similarities,user_ids = np.array(similarities),np.array(user_ids)
    sorted_index = (np.argsort(similarities)[::-1][:num]).tolist()
    most_similar_users = user_ids[sorted_index]
    most_similar_users_rat = similarities[sorted_index]
#     print(most_similar_users_rat)
    return most_similar_users,similarities
```

D. ICF(Cosine as similarity)

D 選項的計算方式和 C 選項大同小異所以這邊就沒有多做說明了，差別就是把原本 user 的地方改成 movies，把原本 movies 的地方改成 user。

另外本來有想做以下兩個模型出來，因為 FNN 是根據 FM 做出來的，而 MF 又是基礎，只是在做的時候如果要使用 Factorization machines，必須先下載 libFM 才可以做後續的動作，但是中間下載、使用各種方法(pytorch、在 cmd 下載 github 的資料、換到 ubumtu 上面跑等等)，下載下來之後 pylibfm 還是沒辦法 import 成功，考慮到時間因素，所以跳過下面兩個方法先做 FNN。

E. FM(沒有做出來)

```
NameError: name 'pylibfm' is not defined
```

F. MF(沒有做出來)

3. Experimental analysis

最終實驗結果呈現如下表(空白部分為結果很怪或沒有做出來):

Typical RecSys Methods: 從上表可以看到不論是使用哪一種 similarity 計算方法，ICF 的效果確實是比 UCF 還要好的；另外單從 user-based 方法來觀察，cosine similarity 表現比 Pearson correlation 還要好，但其他地方沒有做

	MovieLens			Yelp		
	RMSE	Recall@10	NDCG@10	RMSE	Recall@10	NDCG@10
UCF-s	0.55253738	0.61		0.27870381		
UCF-p		0.20	0.51357714			
ICF-s	0.46453985					
ICF-p		0.50	0.68688753			
FNN	0.86672402	0.1267	0.5048	0.15464597	0.0463	0.0369
My own NN model	0.88480755	0.1632	0.6411			

出來所以沒辦法很肯定。時間關係 Yelp 只有做出 rmse。

NN-based Methods: 我提出的方法整體上和 FNN 沒有很大的差異，在 RMSE 表現較差，Recall 和 NDCG 好一點點，但我覺得我的方法改善很多的地方是過程中減少了很多 parameters，有效加快計算速度，可以證實 parameters 確實會影響到模型效率的表現，算是有比較看的到成果的部分。

從上表可以看到不論是使用哪一種 similarity 計算方法，ICF 的效果確實是比 UCF 還要好的，但是 NN 的表現並不符合期待，我覺得最大的原因是因為在資料集的地方我把空值都設為 0.，但事實上這是不太好的處理方法，應該要利用到其他的資料並打它 concat 起來，但是我沒有找到適當的方式，另一個點是我這次沒有利用到其他背景資訊，就單純使用 user-item 的紀錄下去做推薦，我想這也是很大一部分原因，但光是要成功做出方法就花了我非常多時間，希望之後可以把背景資訊加進去。

4. Conclusions

在這次作業裡我嘗試了 UCF-s、UCF-p、ICF-s、ICF-p、MF、FM、FNN，雖然沒有每個都做出來，但至少有做到一定的比較，從中我另外學習到的是，本來在思考要做更多模型方法還是先做其他資料集，選擇做 Yelp 資料集之後發現了我原本設計模型的缺點，因為 MovieLens 算是很小的資料集，所以不會有空值設為 0.跑不出來的問題，但是 Yelp 的 user 數量直接翻倍，這樣的方法是行不通的，必須先預測出一定的 ratings，更不用說第三個資料集，user 數量更大。

未來會嘗試把像 user-user 等資料使用進去，然後實際去改善(把 rating 預測出來，可能利用像 MF 的方式)數據稀疏的問題，這次光是研究要怎麼把模型建出來就花了非常多時間，另外在 evluation 的地方也有遇到一些問題，好像有很多方式可以做出 recall 和 NDCG 這之後也可以做嘗試。

5. Citations

https://arxiv.org/pdf/1601.02376.pdf
https://medium.com/qiubingcheng/%E4%BB%A5python%E5%AF%A6%E7%8F%BE%E6%8E%A8%E8%96%A6%E7%B3%BB%E7%B5%B1%E7%9A%84%E5%8D%94%E5%90%8C%E9%81%8E%E6%BF%BE%E7%AE%97%E6%B3%95-d35cc1a1ec8a

*程式碼檔案名稱說明:

	MovieLens			Yelp		
	RMSE	Recall@10	NDCG@10	RMSE	Recall@10	NDCG@10
UCF-s	R76114026_UCF-cos.ipynb			R76114026_FNN-yelp.ipynb		
UCF-p	R76114026_CF-PCC.ipynb			X		
ICF-s	R76114026_ICF-cos.ipynb					
ICF-p	R76114026_CF-PCC.ipynb					
FNN	R76114026_FNN.ipynb					
My own NN model	R76114026_My own nn model.ipynb					