Sapienza Università di Roma

Data Mining Technology for Business and Society

Homework 1

Tran Luong Bang - *tran.1956419@studenti.uniroma1.it*
Juan Mata Naranjo - *matanaranjo.1939671@studenti.uniroma1.it*

Part 1.1

Applying all algorithms for recommendation made available by ***'Surprise'*** libraries, according to their default configuration to two datasets *ratings_1.csv* and *ratings_2.csv*

Evaluation: Using Cross-Validation method with number of folds is 5.

*Table 1: Ranking all algorithms to **ratings_1.csv** dataset*

Algorithms	**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Mean**	**Std**
SVD++							
RMSE	0.8934	0.8947	0.89	0.8967	0.896	**0.8942**	0.0024
Fit time	833.42	838.21	833.72	835.04	802.1	828.5	13.31
Test time	13.86	13.79	13.58	13.07	13.15	13.49	0.32
SVD							
RMSE	0.9066	0.9078	0.9036	0.9102	0.9087	**0.9074**	0.0022
Fit Time	12.55	13.06	12.91	11.99	11.74	12.45	0.51
Test time	0.63	0.41	0.4	0.41	0.41	0.45	0.09
KNNBaseLine							
RMSE	0.9051	0.9101	0.9065	0.91	0.9107	**0.9085**	0.0022
Fit Time	1.48	1.27	1.25	1.28	1.33	1.32	0.08
Test time	9.39	9.49	9.71	9.79	9.59	9.59	0.15
BaseLineOnly							
RMSE	0.9176	0.9201	0.9174	0.9209	0.9225	**0.9197**	0.002
Fit Time	0.09	0.09	0.09	0.09	0.09	0.09	0
Test time	0.27	0.27	0.27	0.27	0.28	0.27	0
SlopeOne							
RMSE	0.9199	0.9253	0.9209	0.9236	0.9253	**0.923**	0.0022
Fit Time	3.17	3.59	3.54	3.54	3.13	3.4	0.2
Test time	13.11	12.62	12.16	11.77	12.05	12.34	0.47
KNNWithZScore							
RMSE	0.9272	0.9338	0.9307	0.9322	0.933	**0.9314**	0.0023
Fit Time	1.38	1.43	1.42	1.46	1.3	1.4	0.05
Test time	8.93	8.94	8.74	8.42	8.54	8.72	0.21
KNNWithMeans							
RMSE	0.9287	0.9347	0.9312	0.9331	0.9344	**0.9324**	0.0022
Fit Time	1.21	1.19	1.27	1.29	1.22	1.24	0.03
Test time	7.76	7.96	8.08	7.92	7.82	7.91	0.11
NMF							
RMSE	0.9332	0.9425	0.9341	0.934	0.9386	**0.9365**	0.0036
Fit time	14.19	13.57	13.33	12.58	12.19	13.17	0.71
Test time	0.34	0.44	0.33	0.34	0.32	0.36	0.04
CoClustering							
RMSE	0.9398	0.9469	0.9424	0.9316	0.9383	**0.9398**	0.0051
Fit time	3.67	3.42	3.28	3.34	3.23	3.39	0.15
Test time	0.34	0.32	0.33	0.31	0.3	0.32	0.01
KNNBasic							
RMSE	0.9468	0.9517	0.9505	0.953	0.956	**0.9516**	0.003
Fit time	1.25	1.18	1.17	1.34	1.33	1.26	0.07
Test time	7.8	7.57	7.75	7.66	7.19	7.6	0.22
NormalPredictor							
RMSE	1.5047	1.5031	1.5026	1.5016	1.5018	**1.5028**	0.0011
Fit time	0.25	0.25	0.25	0.3	0.25	0.26	0.02
Test time	0.35	0.34	0.34	0.39	0.34	0.35	0.02

*Table 2: Ranking all algorithms to **ratings_2.csv** dataset*

Algorithms	**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Mean**	**Std**
KNNBaseLine							
RMSE	1.8519	1.904	1.8733	1.8511	1.8565	**1.8674**	0.02
Fit Time	0.14	0.14	0.14	0.14	0.14	0.14	0
Test time	0.95	0.93	0.91	0.91	0.9	0.92	0.02
SVD							
RMSE	1.8599	1.9056	1.8562	1.862	1.8736	**1.8715**	0.018
Fit Time	0.93	0.93	0.93	0.93	0.93	0.93	0
Test time	0.03	0.03	0.03	0.03	0.03	0.03	0
KNNBasic							
RMSE	1.8605	1.9067	1.877	1.8616	1.8684	**1.8748**	0.017
Fit Time	0.13	0.13	0.13	0.13	0.13	0.13	0
Test time	0.73	0.72	0.72	0.72	0.73	0.73	0.01
SVD++							
RMSE	1.8706	1.9123	1.8903	1.8975	1.8993	**1.894**	0.0137
Fit Time	9.15	9.18	9.11	9.33	9.01	9.16	0.11
Test time	0.18	0.18	0.18	0.18	0.18	0.18	0
KNNWithMeans							
RMSE	1.8869	1.9516	1.9123	1.897	1.8863	**1.9068**	0.0243
Fit Time	0.14	0.14	0.14	0.16	0.14	0.14	0.01
Test time	0.79	0.79	0.78	0.77	0.77	0.78	0.01
KNNWithScore							
RMSE	1.9043	1.9714	1.929	1.9142	1.9087	**1.9255**	0.0244
Fit Time	0.16	0.16	0.16	0.15	0.16	0.16	0
Test time	0.83	0.83	0.83	0.82	0.82	0.83	0.01
BaseLineOnly							
RMSE	1.9661	2.0052	1.9827	1.9678	1.9668	**1.9777**	0.015
Fit Time	0.01	0.01	0.01	0.01	0.01	0.01	0
Test time	0.02	0.02	0.02	0.02	0.02	0.02	0
SlopeOne							
RMSE	1.9741	2.0103	1.9811	1.9709	1.9749	**1.9823**	0.0144
Fit time	0.03	0.03	0.03	0.03	0.03	0.03	0
Test time	0.14	0.14	0.14	0.14	0.14	0.14	0
NMF							
RMSE	1.9996	2.0291	2.0262	2.0069	2.0081	**2.014**	0.0116
Fit time	0.98	0.98	0.99	0.97	0.97	0.98	0.01
Test time	0.02	0.03	0.03	0.02	0.03	0.03	0
CoClustering							
RMSE	1.9901	2.0714	2.0268	2.0066	2.0162	**2.0222**	0.0274
Fit time	0.3	0.3	0.3	0.3	0.3	0.3	0
Test time	0.02	0.02	0.02	0.02	0.02	0.02	0
NormalPredictor							
RMSE	3.3075	3.225	3.2375	3.2308	3.2545	**3.251**	0.0299
Fit time	0.02	0.02	0.02	0.02	0.02	0.02	0
Test time	0.03	0.03	0.03	0.03	0.03	0.03	0

We have exploited all **4 CPUS-cores** in our computer running this part by assigning **n_jobs =4** as a parameter of **cross_validate()** function.

Part 1.2

Improving the performance of both KNNBaseline and SVD algorithms, by performing hyperparameters tuning over 5-folds

Grid-of-Parameters:

- **SVD:** *Grid-Search-Cross-Validation* approach for tuning the hyper-parameter of SVD algorithms.

```
param_grid = {
    'n_factors': [50, 100],
    'n_epochs': [10, 30],
    'lr_all': [0.005, 0.01],
    'reg_all': [0.05, 0.1]
}
gs = GridSearchCV(SVD, param_grid, measures=['rmse'], cv=5, n_jobs=4)
```

- **KNNBaseline:** *Random-Search-Cross-Validation* process for tuning the hyper-parameter of the KNNBaseline algorithm

```
param_grid = {
    'k': [30, 40, 50],
    'min_k': [1, 5, 10],
    'sim_options': {
        'name': ['cosine', "msd", 'pearson', 'pearson_baseline'],
        'user_based': [True, False],
        'min_support': [3, 5],
        },
    'bsl_options': {
        'method': ['als'],
        'reg_i': [5, 10, 20],
        'reg_u': [5, 15, 20],
        'n_epochs': [10, 50, 100],
         }
}
gs = RandomizedSearchCV(KNNBaseline, param_grid, n_iter=20,
measures=['rmse'], random_state = 1, cv=5, n_jobs=4)
```

The Best Configurations:

	SVD	KNNBaseline
ratings_1.csv	{'n_factors': 100, 'n_epochs': 30, 'lr_all': 0.01, 'reg_all': 0.1}	{'k': 50, 'min_k': 5, 'sim_options': {'name': 'pearson_baseline', 'user_based': False, 'min_support': 3}, 'bsl_options': {'method': 'als', 'reg_i': 10, 'reg_u': 5, 'n_epochs': 50}}
ratings_2.csv	{'n_factors': 100, 'n_epochs': 30, 'lr_all': 0.005, 'reg_all': 0.1}	{'k': 30, 'min_k': 1, 'sim_options': {'name': 'pearson_baseline', 'user_based': True, 'min_support': 5}, 'bsl_options': {'method': 'als', 'reg_i': 20, 'reg_u': 20, 'n_epochs': 100}}

As a result:

ratings_1.csv	***RMSE***	***Time***
SVD	0.8861	6m 30s
KNNBaseline	0.8865	10m 31s
ratings_2.csv		
SVD	1.8349	31s
KNNBaseline	1.8408	40s

We have exploited all **4 CPUS-cores** in our computer running this part by assigning n_jobs =4 as a parameter of **GridSearchCV()** and **RandomizedSearchCV()** functions.

Part 2.1

In this section we will be using Topic Speci_c PageRank to _nd optimal teams using a damping factor of 0,33. In bold are the pokemons around which the ideal teams will formed.

Best team of 6 Pokemon using Set A as input:

{Gengar, Sirfetch-d, Dragonite, **Pikachu**, Kingdra, Lucario}

Best team of 6 Pokemon using Set B as input:

{**Venusaur**, Dusclops, Torkoal, **Blastoise**, **Charizard**, Urshifu}

Best team of 6 Pokemon using Set C as input:

{**Milotic**, **Dracovish**, Tyranitar, Cinderace, **Excadrill**, **Whimsicott**}

Best team of 6 Pokemon using Charizard as input:

{Venusaur, Grimmsnarl, Torkoal, Clefairy, Groudon, **Charizard**}

Best team of 6 Pokemon using Venusaur as input:

{**Venusaur**, Dusclops, Porygon2, Torkoal, Charizard, Stakataka}

Best team of 6 Pokemon using Kingdra as input:

{Tornadus, Politoed, Tsareena, **Kingdra**, Togedemaru, Kyogre}

Best team of 6 Pokemon using Charizard and Venusaur as input:

{**Venusaur**, Dusclops, Torkoal, **Charizard**, Groudon, Stakataka}

Best team of 6 Pokemon using Charizard and Kingdra as input:

{Politoed, **Charizard**, Raichu, **Kingdra**, Bronzong, Sableye}

Best team of 6 Pokemon using Venusaur and Kingdra as input:

{**Venusaur**, Politoed, Tapu Koko, **Kingdra**, Bronzong, Corviknight}

Number of team members inside the Team(Charizard, Venusaur) that are neither in Team(Charizard) nor in Team(Venusaur): ***0***

Number of team members inside the Team(Charizard, Kingdra) that are neither in Team(Charizard) nor in Team(Kingdra): ***3***

Number of team members inside the Team(Venusaur, Kingdra) that are neither in Team(Venusaur) nor in Team(Kingdra): ***3***

The reason for which the number of team members inside the Team(Charizard, Venusaur) that are neither in Team(Charizard) nor in Team(Venusaur) is 0 is because Charizard and Venusaur have a high affinity.

Part 2.2

In this section we will be looking for local pokemon communities based on their battle affinity. For this purpose, we will look for the ideal community for each pokemon (best community is that which minimizes the conductance) using different damping factors. When doing this we will also take into consideration two constraints: (i) Communities with a conductance value of 0 or 1 are not considered as valid communities and (ii) Communities with more than 140 nodes (Pokemons) are not considered as valid communities (this is the reason for which in the output tsv file we don't have communities greater than 140 pokemons). Based on this analysis we will report the Pokemons that appear in the most number of communities (we will interpret these as the pokemons that have highest affinity overall) and those which appear in least communities (these pokemons will not have very high affinity with most of the other pokemons, better not keep these pokemons 😉).

Five most frequent Pokemon in local communities:

Table 3 Most frequent Pokemons in local communities

Pokemon	Frequency
Kartana	178
Incineroar	177
Primarina	172
Mimikyu	170
Volcarona	169

Five least frequent Pokemon in local communities:

Table 4 Least frequent Pokemons in local communities

Pokemon	Frequency
Roserade	43
Miltank	42
Magnezone	37
Hitmonlee	37
Armaldo	31

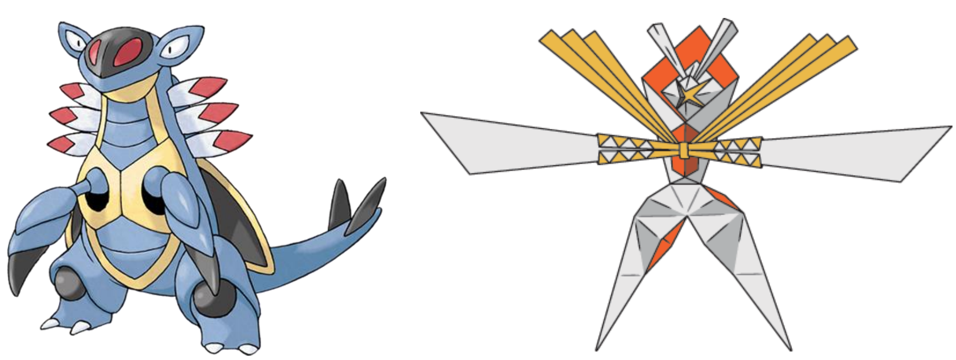

Table 5 Best (right) and worst (left) Pokemon based on Affinity Communities