SAS Recommendation Project Documentation

Introduction

The project *SAS Recommendation* is **eClub Summer Camp** result organized by **Czech Technical University in Prague,** created by **Adam Fessl** and sermonized by **Jan Motl.**

These days there are too many things to see, to buy or to do on the internet or somewhere else. It is important to reduce insignificant items for particular customers and to recommend the best products for them. Also there is SAS software used for data analysis, datamining or just for preparation of data to the next processing like Recommendation engines. The ability to do predictions or recommendations directly in SAS can save money and time.

SAS Recommendation Engine [SAS RecoE] is based on **collaborative filtering** and is developed for **MovieLens** dataset (Netflix dataset can be used too). K-Nearest Neighbour [**K-NN**] approach is selected as basic method that is step by step upgraded. Root Mean Square Error [**RMSE**] evaluation is chosen as the most comparable method appended by **percentage success** of predicted recommendations (How many people like predicted recommendations, how many are chilled and how many hate us).

The best result with advanced k-NN approach is RMSE around 0,92 with 43,5% success and less than 10 % of displeased users.

Running SAS RecoE

Software

SAS® Studio or Base SAS® are required for running RecoE scripts. If needed, SAS® University Edition can be used for running scripts. It can be downloaded here.

SAS® University Edition limitations:

Available only as a Virtual Machine (RedHead Linux)
Only 2 CPU allowed
Only 1 GB RAM can be used
Only 10 MB input data files can be uploaded*

**it is possible to cut data to 10 MB chunks and join after upload*

Dataset

MovieLens dataset is used for script testing. Dataset is available at GroupLens website here or at *dataset* folder.
Any dataset that contains User, Item and Rating columns can be used.

```
User  Item  Rating
196   242   3
186   302   3
22    377   1
244   51    2
166   346   1
…     …     …
```

Settings

All scripts are all-in-one. Each contains settings part, dataset & data preparation part, computation part and eventually evaluation part.

Script parameters:

```
reco              – specifies directory path to dataset
InDS              – Input dataset name
RandomSeed        – used for calculate pseudo-random numbers used
                    to divide dataset 80 | 20 %
k                 – number of nearest neighbour
DistanceMethod    – Method to calculate distances of similarities.
ord               – desc/ascendent order inside k-NN computation.
```

Scripts

01_AVG

Baseline script. Item average is used for recommendation computation. Correction based on user average and global average is applied. Predicted ratings less than one and more than five are bounded back to the rating limits. Global average is used if no one has rated item.

```
PredictedRating = ItemAVG + UserAVG - GlobalAVG
MIN PredictedRating   = 1
MAX PredictedRating   = 5
```

22s	**0,932754**	RMSE 1M	2,5s	**0,958709**	RMSE 100K
42,19%	84345	Success	41,15%	8286	Success
47,32%	94595	Difference 1	47,41%	9546	Difference 1
9,34%	18672	Difference 2	10,01%	2016	Difference 2
1,09%	2183	Difference 3	1,35%	271	Difference 3
0,06%	117	Difference 4	0,08%	16	Difference 4

02_Median

Instead of using average rating median is used.

```
PredictedRating = ItemMedian + UserMedian - GlobalMedian
MIN PredictedRating   = 1
MAX PredictedRating   = 5
```

21s	**1,056985**	RMSE 1M	2,2s	**1,101382**	RMSE 100K
38,41%	76679	Success	36,00%	7240	Success
47,81%	95448	Difference 1	47,92%	9636	Difference 1
11,94%	23843	Difference 2	13,68%	2751	Difference 2
1,67%	3333	Difference 3	2,19%	440	Difference 3
0,17%	331	Difference 4	0,21%	42	Difference 4

03_k-NN

Basic k-NN script. Predicted rating is counted from average item ratings of the top to the nearest neighbours. Euclidian distance is used for selecting the nearest neighbours. Item average is used if rating is missing.

For more info about k-NN see: http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

```
PredictedRating = Item AVG of k Nearest Neighbours ratings.
```

1h 18m	**0,97934**	RMSE 1M	66s	**1,012069**	RMSE 100K
37,79%	75808	Success	36,69%	7357	Success
50,65%	101600	Difference 1	50,21%	10067	Difference 1
10,19%	20432	Difference 2	11,46%	2298	Difference 2
1,36%	2735	Difference 3	1,62%	325	Difference 3
0,01%	20	Difference 4	0,02%	4	Difference 4

04_k-NN

Build on the previous script [No 03]. The results are improved by adding correction based on user average and global average– user bias is removed.

```
PredictedRating = Item AVG of k-NN + UserAVG - GlobalAVG
```

RMSE is improved by 5,5% [RMSE -0,05] and Success is improved by 4%

65s	**0,959549**	RMSE 100K
40,97%	8214	Success
47,76%	9577	Difference 1
9,85%	1975	Difference 2
1,37%	274	Difference 3
0,05%	11	Difference 4

05_k-NN

Build on previous script [No 04]. Predicted rating, less than one and more than five, are bounded back to the rating limits.

```
PredictedRating = Item AVG of k-NN + UserAVG - GlobalAVG
MIN PredictedRating   = 1
MAX PredictedRating   = 5
```

RMSE is improved by 0,25% [RMSE -0,002] and Success is improved by 0,17%

1h 20m	**0,934431**	RMSE 1M	65s	**0,957188**	RMSE 100K
42,20%	84653	Success	41,14%	8248	Success
47,20%	94680	Difference 1	47,64%	9552	Difference 1
9,45%	18947	Difference 2	9,81%	1967	Difference 2
1,10%	2199	Difference 3	1,37%	274	Difference 3
0,06%	116	Difference 4	0,05%	10	Difference 4

06_k-NN

Build on previous script [No 05]. Dimensionality reduction is based on implemented SVD.

For more information about SVD see: http://en.wikipedia.org/wiki/Singular_value_decomposition

```
Input dimensionality reduced.
PredictedRating = Item AVG of k-NN + UserAVG - GlobalAVG
MIN PredictedRating   = 1
MAX PredictedRating   = 5
```

RMSE is improved by 0,8% [RMSE -0,008] and Success is improved by 0,16%

50 min	**0,934935**	RMSE 1M	21s	**0,949498**	RMSE 100K
41,89%	84027	Success	41,29%	8280	Success
47,49%	95258	Difference 1	47,76%	9576	Difference 1
9,50%	19064	Difference 2	9,62%	1928	Difference 2
1,06%	2135	Difference 3	1,29%	259	Difference 3
0,06%	111	Difference 4	0,04%	8	Difference 4

07_k-NN

Build on script No 05 – no SVD used. Cosine similarity used instead of Euclidian distance.

```
Cosine similarity used for finding k-NN.
PredictedRating = Item AVG of k-NN + UserAVG - GlobalAVG
MIN PredictedRating   = 1
MAX PredictedRating   = 5
```

RMSE & Success not affected. Time reduction is improvement due to better algorithm not due to Cosine similarity.

16 min	**0,934435**	RMSE 1M	21s	**0,9571**	RMSE 100K
42,19%	84636	Success	41,11%	8242	Success
47,21%	94701	Difference 1	47,66%	9556	Difference 1
9,44%	18942	Difference 2	9,82%	1970	Difference 2
1,10%	2199	Difference 3	1,36%	273	Difference 3
0,06%	117	Difference 4	0,05%	10	Difference 4

08_k-NN

Build on previous script [No 07]. User similarities computed from transformed ratings to zero-one interval. Ridit scoring is used for rating transformation.

For more info about ridit see: http://en.wikipedia.org/wiki/Ridit_scoring

```
Cosine similarity based on ridit used for finding k-NN.
PredictedRating = Item AVG of k-NN + UserAVG - GlobalAVG
MIN PredictedRating   = 1
MAX PredictedRating   = 5
```

RMSE is improved by 1 % [RMSE -0,01] and Success is improved by 0,5%.

17 min	**0,923398**	RMSE 1M	20,5s	**0,947766**	RMSE 100K
42,86%	85965	Success	41,61%	8344	Success
46,92%	94119	Difference 1	47,49%	9522	Difference 1
9,12%	18286	Difference 2	9,54%	1913	Difference 2
1,06%	2117	Difference 3	1,32%	264	Difference 3
0,05%	108	Difference 4	0,04%	8	Difference 4

09_k-NN

Build on previous script [No 08]. Rating prediction computation inside k-NN is completely rewritten. Ratings are weighted by distances.

For more information see file: *How does Recommendation engine works*

```
Cosine similarity based on ridit used for finding k-NN.
PredictedRating = weighted Item AVG of k-NN + UserAVG - GlobalAVG
MIN PredictedRating   = 1
MAX PredictedRating   = 5
```

28 min	**0,923158**	RMSE 1M	40s	**0,947233**	RMSE 100K
42,88%	86011	Success	41,59%	8340	Success
46,90%	94073	Difference 1	47,50%	9524	Difference 1
9,12%	18289	Difference 2	9,55%	1915	Difference 2
1,05%	2113	Difference 3	1,32%	265	Difference 3
0,05%	109	Difference 4	0,03%	7	Difference 4

10_k-NN

Build on previous script [No 09]. There are no more fake ratings [item AVG instead of missing ratings]. Ratings are normalized by user average.

```
Cosine similarity based on ridit used for finding k-NN.
PredictedRating = UserAVG + weighted Item AVG of k-NN
MIN PredictedRating   = 1
MAX PredictedRating   = 5
```

RMSE is improved by 0,33 % [RMSE -0,003] and Success is improved by 0,8%.

26 min	**0,9201**	RMSE 1M	44s	**0,949793**	RMSE 100K
43,58%	87410	Success	42,45%	8511	Success
46,44%	93155	Difference 1	46,43%	9309	Difference 1
8,63%	17321	Difference 2	9,46%	1896	Difference 2
1,29%	2584	Difference 3	1,63%	327	Difference 3
0,06%	125	Difference 4	0,04%	8	Difference 4

11_k-NN

Build on previous script [No 10]. Instead of using normalized ratings as input ridit score with linear denormalization is used.

```
Cosine similarity based on ridit used for finding k-NN.
Ratings transformed to ridit score [interval zero – one]
PredictedRating = weighted Item AVG of k-NN
Linear denormalization [to interval one t – five]
MIN PredictedRating   = 1
MAX PredictedRating   = 5
```

Branch which is not further developed for its insufficient results.

47s	**1,064918**	RMSE 100K
33,82%	6781	Success
50,38%	10101	Difference 1
14,27%	2862	Difference 2
1,45%	291	Difference 3
0,08%	16	Difference 4

12_SVD

SVD [proc princomp] is used to compute predictions.

Branch which is not further developed for its insufficient results.

30s	**1,120631**	RMSE 100K
35,98%	7183	Success
47,43%	9467	Difference 1
13,41%	2676	Difference 2
2,53%	505	Difference 3
0,66%	131	Difference 4

13_k-NN

Build on script no 10. K-means like clustering [proc fastclust] are used to reduce compute time.

```
Cosine similarity based on ridit used for finding k-NN.
K-means like clustering
PredictedRating = UserAVG + weighted Item AVG of k-NN
MIN PredictedRating   = 1
MAX PredictedRating   = 5
```

With 500 cluster is this approach less accurate [RMSE 16% worst] than the best k-NN script [No 10], but is almost 4x faster. However compared to AVG script this script is useless.

7 min	**1,096295**	RMSE 1M	53s	**1,028258**	RMSE 100K
35,21%	70623	Success	38,98%	7815	Success
48,67%	97635	Difference 1	47,08%	9440	Difference 1
13,17%	26412	Difference 2	11,67%	2339	Difference 2
2,85%	5713	Difference 3	2,17%	436	Difference 3
0,11%	212	Difference 4	0,10%	21	Difference 4

Testing

RMSE

„Root mean square error is a frequently used measure of the differences between values predicted by a model or an estimator and the values actually observed“ [wiki]

RMSE = $AVG\sqrt{(Rating\ -\ Predicted\ Rating)^2}$

Success

This measurement is nearer to man. It simply shows how many users like predicted values and how many are displeased.

Difference = $ROUND\sqrt{(Rating\ -\ Predicted\ Rating)^2}$

Differences zero to four are counted and transformed to percentage view.

HW & SW

Scripts are tested on SAS® University Edition virtual machine.
2 CPU (Core 2 Quad 2.00 GHz without VT-d)
1024 MB RAM

RMSE Comparsion

For **100k MovieLens** dataset script No. 9 has the best RMSE – **0,947233**. Below, there are results for comparison:

0,937 – UserKNNCosine (http://mymedialite.net/examples/datasets.html)
0,934 – UserKNNCosine Rapid Miner (http://predictorfactory.com/doku.php/recommendation:approach)

For **1M MovieLens** dataset script No. 10 has the best RMSE – **0,9201**. Below, there are results for comparison:

0.871 – ItemKNNPearson (http://mymedialite.net/examples/datasets.html)

Conclusion

Thirteen different scripts to predict rating in MovieLens dataset is presented. The most advanced script – number ten – has also the best results RMSE **0,9201** computed in **26 minutes** on virtual machine.
The base line AVG script [number one] with RMSE **0,9328** computed in **22 seconds** can be used in cases, where elapsed time is the most important measurement.