Data Mining Technology for Business and Society

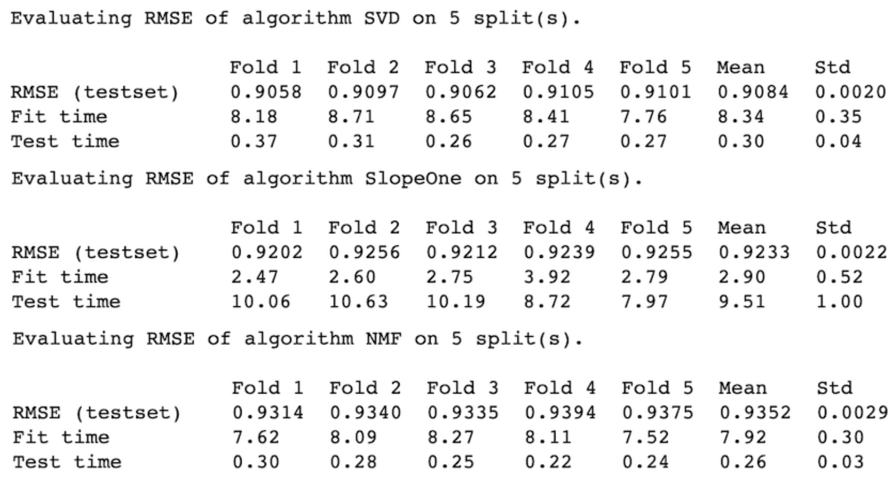
Alice Schirinà, Maria Luisa Croci

1 Recommendation-System

1.1 Non - trivial algorithms

The main goal of the first part of the homework is to optimize the performance of a system recommendation applying to the dataset "rating.csv" all the algorithms from "Surprise" library with the default configurations.

The following screens show the tables in output from the execution of all algorithms.

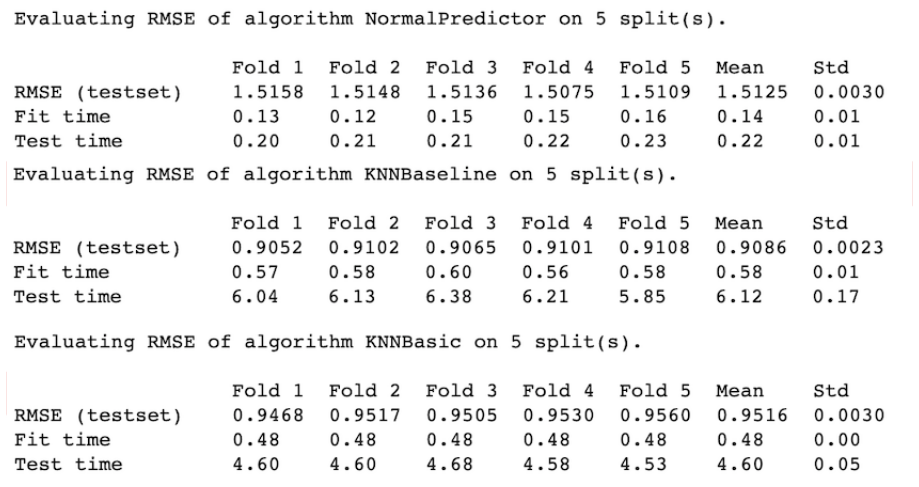


```
Evaluating RMSE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9058  0.9097  0.9062  0.9105  0.9101  0.9084  0.0020
Fit time          8.18    8.71    8.65    8.41    7.76    8.34    0.35
Test time         0.37    0.31    0.26    0.27    0.27    0.30    0.04
Evaluating RMSE of algorithm SlopeOne on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9202  0.9256  0.9212  0.9239  0.9255  0.9233  0.0022
Fit time          2.47    2.60    2.75    3.92    2.79    2.90    0.52
Test time         10.06   10.63   10.19   8.72    7.97    9.51    1.00
Evaluating RMSE of algorithm NMF on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9314  0.9340  0.9335  0.9394  0.9375  0.9352  0.0029
Fit time          7.62    8.09    8.27    8.11    7.52    7.92    0.30
Test time         0.30    0.28    0.25    0.22    0.24    0.26    0.03
```



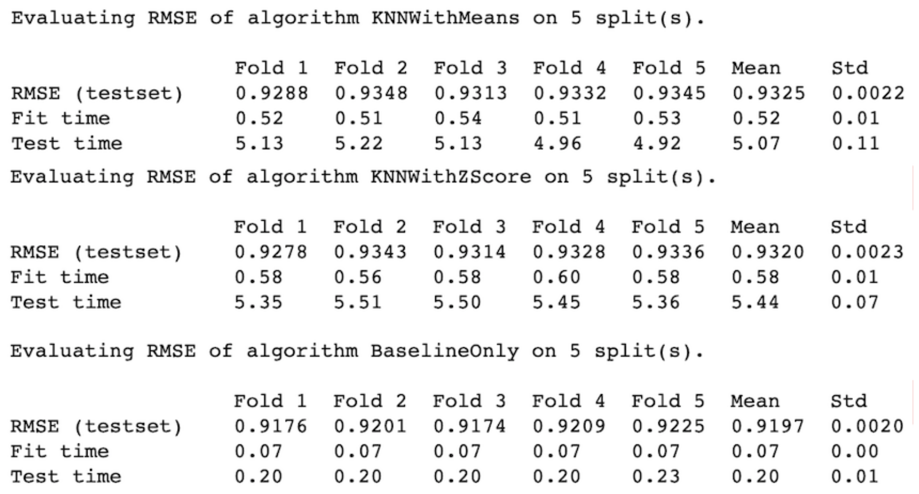
```
Evaluating RMSE of algorithm NormalPredictor on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.5158  1.5148  1.5136  1.5075  1.5109  1.5125  0.0030
Fit time          0.13    0.12    0.15    0.15    0.16    0.14    0.01
Test time         0.20    0.21    0.21    0.22    0.23    0.22    0.01
Evaluating RMSE of algorithm KNNBaseline on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9052  0.9102  0.9065  0.9101  0.9108  0.9086  0.0023
Fit time          0.57    0.58    0.60    0.56    0.58    0.58    0.01
Test time         6.04    6.13    6.38    6.21    5.85    6.12    0.17

Evaluating RMSE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9468  0.9517  0.9505  0.9530  0.9560  0.9516  0.0030
Fit time          0.48    0.48    0.48    0.48    0.48    0.48    0.00
Test time         4.60    4.60    4.68    4.58    4.53    4.60    0.05
```



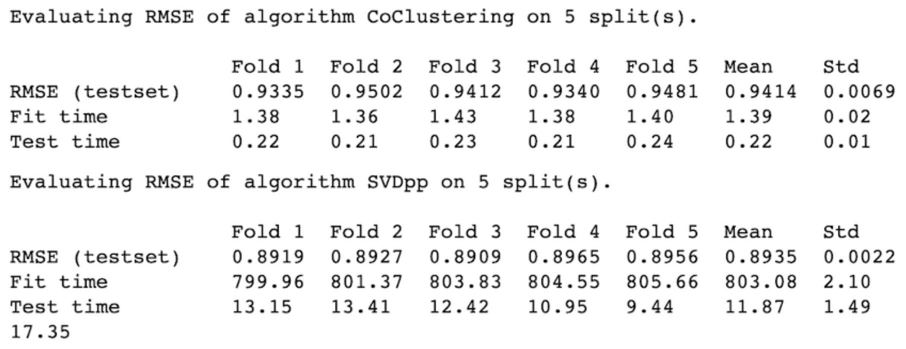
```
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9288  0.9348  0.9313  0.9332  0.9345  0.9325  0.0022
Fit time          0.52    0.51    0.54    0.51    0.53    0.52    0.01
Test time         5.13    5.22    5.13    4.96    4.92    5.07    0.11
Evaluating RMSE of algorithm KNNWithZScore on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9278  0.9343  0.9314  0.9328  0.9336  0.9320  0.0023
Fit time          0.58    0.56    0.58    0.60    0.58    0.58    0.01
Test time         5.35    5.51    5.50    5.45    5.36    5.44    0.07

Evaluating RMSE of algorithm BaselineOnly on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9176  0.9201  0.9174  0.9209  0.9225  0.9197  0.0020
Fit time          0.07    0.07    0.07    0.07    0.07    0.07    0.00
Test time         0.20    0.20    0.20    0.20    0.23    0.20    0.01
```



```
Evaluating RMSE of algorithm CoClustering on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9335  0.9502  0.9412  0.9340  0.9481  0.9414  0.0069
Fit time          1.38    1.36    1.43    1.38    1.40    1.39    0.02
Test time         0.22    0.21    0.23    0.21    0.24    0.22    0.01
Evaluating RMSE of algorithm SVDpp on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8919  0.8927  0.8909  0.8965  0.8956  0.8935  0.0022
Fit time          799.96  801.37  803.83  804.55  805.66  803.08  2.10
Test time         13.15   13.41   12.42   10.95   9.44    11.87   1.49
17.35
```

Looking above it can been observed that the three algorithms with the best average RMSE (on 5 folds) are KNNBaseline (0.9086), SVD (0.9084) and SVDpp (0.8935). For running those simulations we have exploited all the CPU-cores available using the parameter `n_jobs` equal to -1 in the ”cross validate” function.

1.2 Hyper - parameters tuning

In this part we focus on two algorithms, SVD and KNNBaseline, trying to optimize the performance using hyper-parameters tuning. The following codes contain the values for the parameters in the Grid Search.

SVD grid of parameter for the GridSearch:

```
grid_of_parameters = {"n_factors": 110, "n_epochs": 25, "lr_all": 0.009,
"reg_all": 0.06}
```

As best configurations we found:

```
{"n_factors": 110, "n_epochs": 25, "lr_all": 0.009, "reg_all": 0.06}
```

KNNBaseline grid of parameter for the GridSearch:

```
grid_of_parameters = {"bsl_options" : "method" : ["als"], "n_epochs" : [20],
"sim_options" : {"name" : ["cosine", "pearson", "pearson_baseline"],
"min_support" : [1,5], "user_based" : [False, True], "k": [20, 40, 100]}}
```

As best configurations we found:

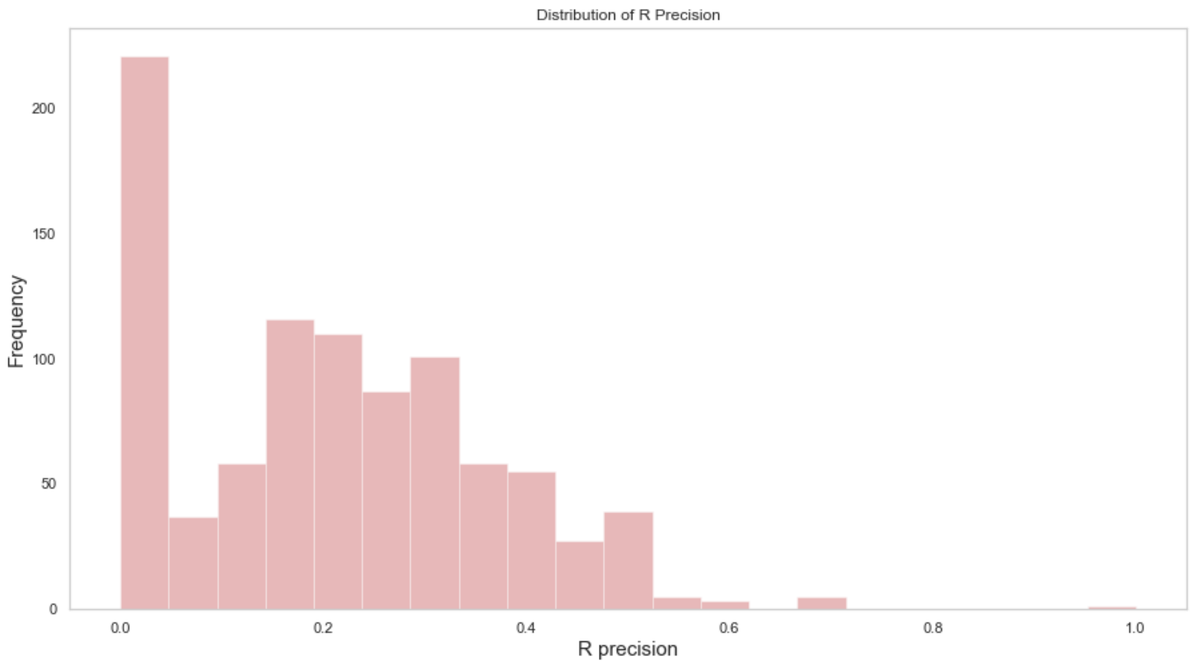
```
{"bsl_options": "method": "als", "n_epochs": 20, "sim_options": {"name":
"pearson_baseline", "min_support": 1, "user_based": False, "k": 20}}
```

The SVD simulation has run in 7.67 minutes (specifying again to use all CPU cores available, i.e. 8 CPU cores, by using `n_jobs` equal to -1) and it has returned the average RMSE equal to 0.887 and as best parameters; the KNNBaseline simulation took about 15 minutes and gives an average RMSE of 0.886.

2 Recommendation Prediction with a PageRank-Like method

2.1 Topic Specific Page Rank

In this part we implement a recommendation/prediction system using the Topic-Sensitive-PageRank method. Using the "Networkx" library we read the edges in the file "User_Item_BIPARTITE_GRAPH___UserID__ItemID.tsv" into order to create a bipartite graph. From this we worked on the Projected-Item-Item-Graph associated to the original bipartite graph, where two items are linked if there is at least one user between them and the weight is equal to the number of users that interact with those two items. We used the Networkx function "pagerank" giving as input the projected graph, the damping factor equal to 0.1 and a personalization vector defined as a dictionary where for each topic/user there is another dictionary that maps if an item (the key) is linked with the topic (so value equal to 1) or if it isn't (value equal to 0). Finally, in order to test the quality of our method, we obtained an average R precision equal to 0.205 and its distribution as follow:



The execution of the entire script has been of 4.5 hours.

2.2 Personalized Page Rank

In the last part we have to recommend again some items to the users, but this time working only with a part of the bipartite graph and the Personalized-PageRank vector associated to each item in the complete Projected-Item-Item graph. Since the personalized PageRank vector for any user can be expressed as a linear combination of the topic specific PageRanks, for each user we calculated for each item the probability vector by averaging over the probabilities given in the Personalized vectors. Again we rated the quality of this method with an average R precision of 0.318. The total execution time was 30 minutes.