The BellKor solution to the Netflix Prize

Robert M. Bell, Yehuda Koren and Chris Volinsky
AT&T Labs – Research
BellKor@research.att.com

Our final solution (RMSE=0.8712) consists of blending 107 individual results. Since many of these results are close variants, we first describe the main approaches behind them. Then, we will move to describing each individual result.

The core components of the solution are published in our ICDM'2007 paper [1] (or, KDD-Cup'2007 paper [2]), and also in the earlier KDD'2007 paper [3]. We assume that the reader is familiar with these works and our terminology there.

Neighborhood-based model (k-NN)

A movie-oriented k-NN approach was thoroughly described in our KDD-Cup'2007 paper [kNN]. We apply it as a post-processor for most other models. Interestingly, it was most effective when applied on residuals of RBMs [5], thereby driving the Quiz RMSE from 0.9093 to 0.8888.

An earlier k-NN approach was described in the KDD'2007 paper ([3], Sec. 3) [Slow-kNN]. It appears that this earlier approach can achieve slightly more accurate results than the newer one, at the expense of a significant increase in running time. Consequently, we dropped the older approach, though some results involving it survive within the final blend.

We also tried more naïve k-NN models, where interpolation weights are based on pairwise similarities between movies (see [2], Sec. 2.2). Specifically, we based weights on $corr^2/(1-corr^2)$ [Corr-kNN], or on mse^{-10} [MSE-kNN]. Here, *corr* is the Pearson correlation coefficient between the two respective movies, and *mse* is the mean squared distance between two movies (see definition of s_{ij} in Sec. 4.1 of [2]). We also tried taking the interpolation weights as the "support-based similarities", which will be defined shortly [Supp-kNN].

Other variants that we tried for computing the interpolation coefficients are: (1) using our KDD-Cup'2007 [2] method on a binary user-movie matrix, which replaces every rating with "1", and sets non-rated user-movie pairs to "0" [Bin-kNN]. (2) Taking results of factorization, and regressing the factors associated with the target movie on the factors associated with its neighbors. Then, the resulting regression coefficients are used as interpolation weights [Fctr-kNN].

As explained in our papers, we also tried user-oriented k-NN approaches. Either in a profound way (see: [1], Sec. 4.3; [3], Sec. 5) [User-kNN], or by just taking weights as pairwise similarities among users [User-MSE-kNN], which is the user-oriented parallel of the aforementioned [MSE-kNN].

Prior to computing interpolation weights, one has to choose the set of neighbors. We find the most similar neighbors based on an appropriate similarity measure. In Sec. 4.1 of [2] we mention a correlation-based similarity measure and a distance-based similarity measure. Another kind of similarity is based solely on who-rated-what (we refer to this as "support-based" or "binary-based" similarity). The full details are given in an Appendix 1. Interestingly, when post-processing residuals of factorization or

RBMs, these seemingly inferior support-based similarities led to more accurate results.

A factorization model

The earlier factorization model that we employed is fully described in our KDD'2007 paper ([3], Sec. 4). Later we moved to a more powerful model, which is described at Section 5.1 of [1]. The essence of these models is alternating between computing all movie factors and all user factors, by optimally solving regularized least squares problems; also known as *alternating least squares*. The KDD'2007 paper [3] suggested computing each factor separately while performing simple shrinkage [IncFctr]. The ICDM'2007 paper [1] suggested computing all factors associated with a single user/movie simultaneously, while using Ridge regression [SimuFctr]. An additional accuracy boost was achieved when we required all factors to be non-negative [NNMF], by employing a non-negative least squares solver. Another alternative that we tried is replacing the Ridge regression (L2-norm penalty) by Lasso regression (L1-norm penalty) [LassoNNMF]. In general, the Lasso regularization was far less effective, but somewhat contributed to our overall blend.

All factorization models significantly benefit from recomputing user-factors in a neighborhood-aware fashion, in the spirit described in our KDD'2007 paper ([3], Sec. 4.4), modified according to the actually employed model. In this process, we adapt the computed user factor to the given query, by weighting all related ratings by movie-movie similarities. Here, choosing the most effective movie-movie similarities involves more art than science. Our recommended choice is taking $s_{ij}=MSE(i,j)^{-6}$, where MSE(i,j) is the mean squared distance between ratings of movie i and those of movie j [MseSim]. Other approaches for the movie-movie similarities that were used are: (1) $s_{ij}=corr(i,j)^2/(1-corr(i,j)^2)$, where corr(i,j) is the Pearson correlation coefficient [CorrSim]. (2) Compute all similarities simultaneously as we compute interpolation weights in the KDD-Cup'2007 paper ([2], Eq. (13)) [SimuSim]. (3) Basing similarities on the inverse of the edit distance of movie titles, accounting also for gap of release year [EditSim]; see Appendix 2. (4) Support-based similarities, which were mentioned earlier [SuppSim]; see Appendix 1.
In addition, no matter how similarities are computed, we also introduce a "date factor" into them. That is, when measuring the similarity between two ratings, we also account for the date gap between them. More specifically, if movie i was rated *x* days later than movie j, we multiply their similarity (s_{ij}) by $\exp(-x/600)$. The denominator 600 (days) was determined by cross validation, and reflects the fact that after two years, similarity decays by approximately a factor of 3.

Regularization based on a Gaussian prior:
While Ridge-regression was proved very effective when deriving factors, it offers a quite limited model by assuming that all factors belong to a normal distribution with zero mean, and a uniform diagonal covariance matrix. A richer model, will not impose these restrictions, but assume a general normal distribution for the factors [GaussFctr]; see, e.g., Zhang and J. Koren [6]. In practice, we have found a rich Gaussian prior being very effective when computing user factors, but less so when dealing with movie factors. Consequently, we mainly apply it on the users' side.

Restricted Boltzmann Machines

Restricted Boltzmann Machines (RBM) for collaborative filtering were recently described by one of the leading teams in this challenge [5]. We implemented their idea, and could verify most claims of the authors. (We used almost the same parameter setting as suggested in [5], except doubling the learning rate to 0.02.) We have found that RBMs lead to competitive results in terms of accuracy, with a relatively low sensitivity to parameter setting. While the basic approach is fully detailed in the paper, one modification that we tried is replacing the multinomial visible units with Gaussian ones. This way the RBMs can post-process the residuals of global effects or any other method.

Asymmetric factor models

The factorization model offered a symmetric view of users and movies, by directly parameterizing each of them. However, this practice involves a clear redundancy, as user parameters ("factors") are dependent on the movie-parameters, and vice-versa. An interesting family of models differs from the factorization model by parameterizing only the movies, which have higher support in the training data. Consequently, no explicit user factors are computed, but a user factor is implicitly derived by aggregating the movie factors associated with the movies liked by that user (taking the view that a user is "a bag of movies").
This aggregate is often a plain sum of the respective movie factors, transformed by a function that accounts for the deviations in the number of summed values. The approach was first publicly mentioned by Paterek (Section 3.2 of [4], [NSVD1,NSVD2]). Paterek suggested normalizing this sum by the square root of the support of the respective user. Initially, we followed this approach. Later, we found two more efficient related models that we outline in the followings.

A weighted scores model:
Let n be the number of movies, m the number of users, and k the number of factors. Let us denote by g the number of different scores (g=5 for the Netflix data).
We estimate r_{ui}, rating by user u of movie i, as follows:

$$r_{ui} = \sum_{f=1}^{k} \left(p_{if} \cdot \sigma \left(b_f + \sum_{j \ rated \ by \ u} w_{score(u,j)} \cdot q_{jf} \right) \right)$$

Here, the constant score(u,j) is the score given by user u to movie j (an integer between 1 to g). Whenever this score is unknown, but we know that the rating exists (e.g., (u,j) belongs to the Qualifying set), we set score(u,j)=0.

$\sigma(x)$ is the Sigmoid function, defined as: $\sigma(x) = \frac{1}{1+e^{-x}}$.

The following parameters should be learnt from the data:
(1) Two sets of "movie factors": an nxk matrix $P=\{p_{if}\}$, and an nxk matrix $Q=\{q_{if}\}$.
(2) A set of k intercepts - the b_f's.
(3) The g+1 weights: $w_0,w_1,...,w_g$.

All these 2kn+k+g+1 parameters are learnt by gradient descent that minimizes the related cost function:

$$\sum_{given\ rating\ r_{ui}} \left(r_{ui} - \sum_{f=1}^{k} \left(p_{if} \cdot \sigma \left(b_f + \sum_{j\ rated\ by\ u} w_{score(u,j)} \cdot q_{jf} \right) \right) \right)^2$$

Optimization process should be regularized by penalizing the magnitudes of the learnt parameters [SIGMOID1] (learning rate=0.002, regularization=0.01).
One can use a single set of movie factors, by equating the matrices P and Q [SIGMOID2] (learning rate=0.001, regularization=0.01).
Typically, we apply this method to residuals of global effects, so the above r_{ui} would symbolize a residual, rather than a raw score.

Weighting with residuals:
Let us denote by res_{ui} the residual of the rating by user u of movie i after removing global effects (one can alternatively just use double centering, which will lead to somewhat inferior results).
We estimate res_{ui}, the residual of the rating by user u of movie i, as follows:

$$res_{ui} = \sum_{f=1}^{k} \left(p_{if} \cdot \sigma \left(b_f + \sum_{j\ rated\ by\ u} q_{jf} + \sum_{known\ rating\ r_{uj}} res_{uj} \cdot p_{jf} \right) \right)$$

The following parameters should be learnt from the data:
(1) Two sets of "movie factors": an nxk matrix $P=\{p_{if}\}$, and an nxk matrix $Q=\{q_{if}\}$.
(2) A set of k intercepts - the b_f's.

All these 2kn+k parameters are learnt by gradient descent that minimizes the related quadratic error cost function. Optimization process should be regularized by penalizing the magnitudes of the learnt parameters [SIGMOID3] (learning rate=0.001, regularization=0.02).
The astute reader will find distinct relations between this residual-based model and RBMs with Gaussian visible units.

Regression models

Regression can be performed in two symmetric ways:

1. A user-centric approach: The sample is all movies rated by a specific user. The response variable is rating of a movie by this user. The predictor variables are different attributes associated with the movies.
2. A movie-centric approach: The sample is all users that rated a specific movie. The response variable is rating of the movie by a user. The predictor variables are different attributes associated with the users.

The central issue is how to create predictor variables. A few approaches were beneficial to us. The relatively low number of movies facilitates building movie-based predictor variables by concentrating on movie-movie relationships, including:

1. Estimating the movie-movie covariance matrix, and then taking its top k eigenvectors as the predictor variables. This is akin to Principal Components Analysis [PCA].
2. Embedding all movies in a k-D Euclidean space, by solving a multidimensional scaling (MDS) problem, based on minimizing the stress energy by majorization. Typical values of k lie between 40 and 100 [STRESS].
3. Build a user-movie binary matrix, where an entry value is "1" iff the corresponding user rated the corresponding movie. Interestingly, this matrix contains no missing value, so we can directly extract its SVD vectors and use them as predictor variables [BIN-SVD]. In some variants we considered all ratings smaller than 3 as zeros [BIN-SVD3]. When working with these binary-based factors, we suggest normalizing the vector of factors associated with each movie, to offset for movie popularity.

These movie-based predictors were used within a user-centric regression approach. In order to use a movie-centric regression, we need to derive user-based predictors, which are more challenging due to the huge number of users in the dataset. To overcome this, we first derive the movie-based predictors as above, and then infer from them the user-based predictors in one of the following two methods:

1. Regress user-based predictors from the movie-based predictors using a user-centric regression (within the binary representation). This way, [BIN-SVD] and [BIN-SVD3] become [BIN-SVD-USER] and [BIN-SVD3-USER], respectively. Once again, we suggest normalizing the vectors of factors associated with each user.
2. A barycentric assignment: A user predictor is taken as a weighted average of the movie predictors, for movies rated by this user. Here, we used those weights found in the "weighted scores model" mentioned above. This way, [PCA] and [STRESS] become [PCA-USER] and [STRESS-USER], respectively.

Unless stated otherwise, we apply the regression models on residuals of global effects.

Combining multiple results

Predictive accuracy is substantially improved when blending multiple predictors. *Our experience is that most efforts should be concentrated in deriving substantially different approaches, rather than refining a single technique.* Consequently, our solution is an ensemble of many methods.

We approach blending as a linear regression problem. We ought to regress a target ratings vector on multiple predictors. The target rating vector can be the true ratings of the Probe set, and the predictors are the respective estimates for the Probe set by an ensemble of methods. The solution is the coefficients, or the weights, that should be given to each of the predictors in the ensemble.

An extension would be to slice and dice the target vector based on some criteria. This allows overweighting certain methods on some user-movie pairs, while overweighting different methods on other user-movie pairs. The criterion we used is the support of the user-movie pair, which we define as the minimum between the support associated with the user and the support associated with the movie. (A support of a user refers to the number of ratings made by this user. Similarly, a support of a movie is the number of ratings given to it.) Consequently, we typically partition the Probe (or Qualifying) sets into 15 bins, based on the support level. Then, we solve a different regression

problem within each bin, thereby obtaining unique combination coefficients for each bin. For example, we have found that RBMs should be overweighted on low support pairs, while factorization should be overweighted on high support pairs.
Since the ratings of the Qualifying pairs are unknown, we cannot simply regress them. One can "borrow" the combination coefficients learnt for the Probe set. Accuracy can be improved by partitioning the Probe set into several disjoint sets. Then, learn combination coefficients for each portion of the Probe set separately, while including the rest of the Probe set within the training data. This better reflects the situation with the Qualifying set, where all Probe set is included within the training data. Finally, for the Qualifying coefficients, one can average the coefficients derived for the multiple portions of the Probe set. Alternatively, sufficient statistics that enable regressing directly the Quiz set can be obtained with the aid of the RMSEs reported for each predictor.

Our predictors

Below, we list all 107 results that were blended to deliver RMSE=0.8712, with their weights within the ensemble. The results are grouped based on the type of method that produced them.
We strongly believe that the success of an ensemble approach depends on the ability of its various predictors to expose different, complementing aspects of the data. Experience shows that this is very different from optimizing the accuracy of each individual predictor. Quite frequently we have found that the more accurate predictors are less useful within the full blend. Therefore, the RMSEs listed below should be considered just as a reference of our experience, and we would not encourage the readers to repeat the same RMSEs, but rather to concentrate on achieving the “spirit” of the listed methods.
Please note that before submitting a result, we translate it so its mean will coincide with the mean of the Quiz set, which is known to be 3.6744[1]. Also, before mixing the results, each of them is centered (to have zero mean). Later, the final blend is translated back so its mean coincides with the Quiz mean.

Asymmetric factor models

1. *rmse*=0.9194, *weight*=-0.0382
   SIGMOID3 with k=30
2. *rmse*=0. 9286, *weight*=-0.0481
   SIGMOID2 with k=40
3. *rmse*=0. 9383, *weight*=0.0514
   NSVD2 with k=40
4. *rmse*=0. 9245, *weight*=-0.0393
   SIGMOID1 with k=40
5. *rmse*=0. 9114, *weight*=0.0574
   40 neighbors kNN on residuals of NSVD1 with k=200.
6. *rmse*=0.9236., *weight*=0.0550
   NSVD1 with k=200
7. *rmse*=0.9259, *weight*=0.0832
   NSVD1 with k=150

[1] See Winsteps post at http://www.netflixprize.com//community/viewtopic.php?id=503

8. *rmse*=0. 9134, *weight*=0.0660
   30 neighbors kNN on residuals of non-regularized NSVD1 with k=200
9. *rmse*=0. 9260, *weight*=-0.0484
   NSVD1 with k=40

Regression models

10. *rmse*=0.9269, *weight*=-0.0370
    PCA-USER, based on top 40 PCs
11. *rmse*=0.9302, *weight*=-0.0499
    STRESS with 40 coordinates per movie
12. *rmse*=0.9335, *weight*=-0.1145
    BIN-SVD-USER based on 256 vectors
13. *rmse*=0. 8996, *weight*=0.1350
    50 neighbors kNN on residuals of BIN-SVD-USER (256 vectors)
14. *rmse*=0.9290, *weight*=-0.0626
    BIN-SVD3-USER based on 65 vectors
15. *rmse*=0.9394, *weight*=0.0311
    BIN-SVD3-USER based on 256 vectors
16. *rmse*=0.9241, *weight*=-0.0692
    PCA based on top 40 PCs
17. *rmse*=0.9212, *weight*=-0.0639
    PCA based on top 50 PCs
18. *rmse*=0. 9451, *weight*=-0.0484
    BIN-SVD-USER based on 60 vectors
19. *rmse*=0. 9610, *weight*=0.0401
    BIN-SVD-USER based on 100 vectors, but here we regressed residuals of double centering rather than the usual residuals of global effects
20. *rmse*=0. 9414, *weight*=0.0424
    BIN-SVD3-USER based on 40 vectors
21. *rmse*=0. 9067, *weight*=0.0689
    20 neighbors Corr-kNN on residuals of BIN-SVD-USER (60 vectors)
22. *rmse*=0. 9020, *weight*=0.0556
    50 neighbors kNN on residuals of BIN-SVD-USER (60 vectors)
23. *rmse*=0.9030, *weight*=-0.0491
    50 neighbors kNN on residuals of BIN-SVD-USER (100 vectors)
24. *rmse*=0. 9223, *weight*=0.0763
    BIN-SVD3 based on 40 vectors

Restricted Boltzmann Machines with Gaussian visible units
The default is to apply the machine in a "conditional RBM" mode on data normalized by applying all global effects that we describe in the KDD-Cup'2007 paper ([2], Sec. 3), except the last four, which interact movies/users with popularity.

25. *rmse*=0.9052, *weight*=0.0704
    800 hidden units
26. *rmse*=0.9044, *weight*=0.0739
    400 hidden units
27. *rmse*=0.9056, *weight*=0.0771
    256 hidden units

28. *rmse*=0.9068, *weight*=0.0433
    100 hidden units
29. *rmse*=0.9121, *weight*=-0.0268
    40 hidden units
30. *rmse*=0.9429, *weight*=0.0220
    100 hidden units, applied on raw data (no normalization/centering)
31. *rmse*=0.9489, *weight*=-0.0295
    50 hidden units, applied on raw data (no normalization/centering)
32. *rmse*=0.9267, *weight*=-0.0726
    100 hidden units, without conditional RBM, on residuals of full global effects

Restricted Boltzmann Machines

We use conditional RBMs as described in [5].

33. *rmse*=0.9029, *weight*=0.0785
    256-unit RBM
34. *rmse*=0.9029, *weight*=-0.0700
    200-unit RBM
35. *rmse*=0.9093, *weight*=-0.0977
    100-unit RBM
36. *rmse*=0.9206, *weight*=-0.0239
    40-unit RBM

Using RBM as a pre-processor:

37. *rmse*=0.8960, *weight*=0.0577
    Postprocessing residuals of 100-unit RBM with factorization
38. *rmse*=0.8905, *weight*=0.1839
    50 neighbors kNN on 200-unit RBM
39. *rmse*=0.8904, *weight*=0.0576
    40 neighbors kNN on 150-unit RBM
40. *rmse*=0.8888, *weight*=0.1387
    50 neighbors kNN on 100-unit RBM

Matrix factorization

41. *rmse*=0.9135, *weight*=-0.0302
    IncFctr (40 factors)
42. *rmse*=0. 8992, *weight*=0.0436
    IncFctr (80 factors), adaptive user factors by [MseSim]
43. *rmse*=0.9042, *weight*=0.0339
    IncFctr (40 factors), adaptive user factors by [SuppSim]
44. *rmse*=0.9083, *weight*=-0.0389
    IncFctr (40 factors), adaptive user by [EditSim]
45. *rmse*=0. 9002, *weight*=-0.0907
    SimuFctr (40 factors), adaptive user factors by [MseSim]
46. *rmse*=0. 9050, *weight*=0.1178
    SimuFctr (40 factors), adaptive user factors with $s_{ij}=MSE(i,j)^{-12}$
47. *rmse*=0. 9035, *weight*=-0.0546
    MSE-kNN applied to residuals of SimuFctr (60 factors)
48. *rmse*=0. 9515, *weight*=-0.0146
    NNMF (5 factors)
49. *rmse*=0.9347, *weight*=-0.0227
    NNMF (20 factors), training set excluded Probe set

50. *rmse*=0.9084, *weight*=-0.0342
    NNMF (20 factors), adaptive user factors by [SimuSim]
51. *rmse*=0. 9073, *weight*=-0.0353
    NNMF (20 factors), adaptive user factors by [EditSim]
52. *rmse*=0.9094, *weight*=-0.1412
    NNMF (40 factors)
53. *rmse*=0.9018, *weight*=0.2535
    NNMF (40 factors, adaptive user factors by [EditSim]
54. *rmse*=0.8986, *weight*=-0.1093
    NNMF (40 factors, adaptive user factors by [MseSim]
55. *rmse*=0.9026, *weight*=-0.1159
    Cor-kNN on residuals of NNMF (60 factors)
56. *rmse*=0.8963, *weight*=-0.1032
    NNMF (90 factors), adaptive user factors by [MseSim]
57. *rmse*=0.8986, *weight*=0.0714
    NNMF (90 factors), adaptive user factors by naive [SuppSim] (where $x_{ij}=n_i \cdot n_j/n$)
58. *rmse*=0.9807, *weight*=0.0228
    NNMF (90 factors), adaptive user factors by $s_{ij}=MSE(i,j)^{-12}$
59. *rmse*=0.8970, *weight*=0.1028
    NNMF (90 factors), adaptive user factors by [SuppSim]
60. *rmse*=0.8978, *weight*=0.1035
    NNMF (90 factors), adaptive user factors by [CorrSim]
61. *rmse*=0.8985, *weight*=-0.1121
    NNMF (90 factors), adaptive user factors by [EditSim]
62. *rmse*=1.1561, *weight*=-0.0111
    NNMF (128 factors), adaptive user factors by [MseSim], but adaptive user factors where computed with Lasso regularization, rather than Ridge regularization
63. *rmse*=0.9039, *weight*=-0.0928
    NNMF (128 factors)
64. *rmse*=0.8955, *weight*=0.1060
    NNMF (128 factors), adaptive user factors by [MseSim]
65. *rmse*=0.9426, *weight*=-0.0564
    LassoNNMF (30 factors)
66. *rmse*=0.9327, *weight*=0.0445
    LassoNNMF (30 factors), adaptive user factors by [SuppSim]
67. *rmse*=0.9016, *weight*=0.0543
    Start with NNMF 40 factors, then alternate between GaussFctr on user side and SimuFctr on movie side
68. *rmse*=0.8998, *weight*=0.0958
    Start with SimuFctr 60 factors, then a single GaussFctr iterations on movie side followed by many GaussFctr iterations on user side
69. *rmse*=0.9070, *weight*=-0.0954
    Start with NNMF 90 factors, followed by many GaussFctr iterations on user side
70. *rmse*=0.9098, *weight*=0.0720
    Start with SimuFctr 40 factors, followed by many GaussFctr iterations on user side

Neighborhood-based model (k-NN)
Some k-NN results were embedded in the previous sections. Here, we report the rest.

71. *rmse*=0. 8953, *weight*=0.0932
    30 neighbors kNN on residuals of NNMF (180 factors)
72. *rmse*=0. 9105, *weight*=-0.1386
    50 neighbors kNN on residuals of all global effects except the last 4
73. *rmse*=0. 9082, *weight*=-0.0456
    50 neighbors kNN on residuals of full global effects
74. *rmse*=0.9496, *weight*=-0.0272
    25 neighbors kNN on raw scores (no normalization)
75. *rmse*=0.8979, *weight*=-0.1099
    60 neighbors kNN on residuals of NNMF (60 factors)
76. *rmse*=0. 9247, *weight*=0.0525
    50 neighbors Bin-kNN on residuals of full global effects, neighbor selection by [SuppSim]
77. *rmse*=0.9215, *weight*=0.0783
    50 neighbors Bin-kNN on residuals of full global effects, neighbor selection by [CorrSim]
78. *rmse*=0.9309, *weight*=-0.0985
    50 neighbors Fctr-kNN on residuals of full global effects. Weights based on 10 factors computed on binary matrix
79. *rmse*=0.9097, *weight*=0.0681
    25 neighbors Fctr-kNN on residuals of NNMF (60 factors). Weights based on 10 NNMF factors
80. *rmse*=0.9290, *weight*=-0.0451
    50 neighbors Fctr-kNN on raw scores. Weights based on 10 factors computed on binary matrix
81. *rmse*=0.9097, *weight*=0.0408
    100 neighbors User-kNN on residuals of NNMF (60 factors)
82. *rmse*=0. 9248, *weight*=0.0333
    30 neighbors User-MSE-kNN on residuals of full global effects
83. *rmse*=0.9057, *weight*=0.0550
    50 neighbors Slow-kNN on residuals of full global effects
84. *rmse*=0.9170, *weight*=-0.0648
    Corr-kNN on residuals of full global effects
85. *rmse*=0.9237, *weight*=-0.0561
    MSE-kNN on residuals of full global effects
86. *rmse*=0.9110, *weight*=0.0439
    Supp-kNN on residuals of IncFctr (80 factors)
87. *rmse*=0.9440, *weight*=-0.0422
    Supp-kNN on residuals of full global effects. Here, we used the more naïve similarities where xij=ni*nj/n
88. *rmse*=0.9335, *weight*=0.0402
    Supp-kNN on residuals of full global effects

Combinations:
Each of the following results is based on mixing two individual results. Before mixing we split the user-movie pairs into 15 bins based on their support. For each bin we compute unique combination coefficients based on regression involving the Probe set.

89. *rmse*=0.8976, *weight*=0.0552
    Combination of #67 with #35
90. *rmse*=0.8876, *weight*=0.1471
    Combination of #36 with <NNMF (60 factors) adaptive user factors by MseSim>
91. *rmse*=0.8977, *weight*=0.1053
    Combination of #81 with #75
92. *rmse*=0.8909, *weight*=0.0588
    Combination of #45 with <User-kNN on all global effects but the last 4>
93. *rmse*=0.9003, *weight*=0.0757
    Combination of #50 with <NNMF (20 factors, adaptive user factors by [MseSim]>
94. *rmse*=0.8906, *weight*=-0.0634
    Combination of #45 with #73
95. *rmse*=0.9024, *weight*=0.0569
    Combination of #73 with <50 neighbors Slow-kNN on residuals of all global effects except last 4>
96. *rmse*=0.9078, *weight*=0.0372
    Combination of #84 with <User-kNN on raw scores>
97. *rmse*=0.9046, *weight*=0.0508
    Combination of #74 with *#66*

Imputation of Qualifying predictions:
We had predictions for the Qualifying set with RMSE of 0.8836. Then, we inserted the Qualifying set into the training set, while setting unknown scores to the RMSE= 0.8836 predictions. We tried some of our methods on this enhanced training set:

98. *rmse*=0.8952, *weight*=0.0937
    MSE-kNN on residuals of SimuFctr (20 factors)
99. *rmse*=0.9100, *weight*=-0.0314
    IncFctr (40 factors)
100. *rmse*=0.9039, *weight*=0.0735
    IncFctr (40 factors), adaptive user factors by [SuppSim]
101. *rmse*=0.9056, *weight*=-0.1866
    SimuFctr (20 factors), Probe set is excluded from training set
102. *rmse*=0.9093, *weight*=-0.0769
    IncFctr (40 factors), adaptive user factors by [SuppSim]. Probe set is excluded from training set
103. *rmse*=0.9005, *weight*=0.0503
    MSE-kNN on residuals of IncFctr (40 factors)
104. *rmse*=0.8975, *weight*=0.1155
    A combination (by 15 support-base bins) of #99 with <SimuFctr (20 factors)>

Specials:

105. *rmse*=1.1263, *weight*=-0.1345
Take binary matrix (rated=1, not-rated=0), and estimate it by 40 factors. Using this factors, construct predictions for the Probe and Qualifying set and center the predictions for each set. Consequently, using the probe set we learn how to regress centered true ratings on these predictions, and do the same on the Qualifying set.

106. *rmse*=0.9162, *weight*=-0.0702
This method fits a series of models, each using the residuals from the previous model. There were three stages of models. First, effects were fit for rating date, movie, and user. Second, interactions were fit between users and 11 movie factors. The first factor was movie effect estimated above, while the last ten factors were based on approximate principal components of the movies. The movie and user effects and all 11 interactions were shrunk using a form of empirical Bayes. Third, residuals of movies in the qualifying data set were predicted using linear combinations of residuals of correlation–based nearest neighbors. Predictions for the qualifying data equal the sums of the various models. Models from all three stages were fit using training data only (without including the Probe set).

107. *rmse*=0.9134, *weight*=0.1051
This method is similar to #106 with the following exceptions. The main effect for date of rating was excluded. After fitting main effects for movie and user, eight interactions were included for movie and user support, movie and user effects, and four versions of time (see [2], Sec. 3). For these eight interactions, the linear fits were replaced by quadratic fits, and empirical Bayes shrinkage was performed on both the linear and quadratic terms (after make them orthogonal to each other) analogously to the previous method (#106). Models for all three stages were fit using the training data plus a random 90 percent sample of the probe data.

How many results are really needed?

For completeness, we listed all 107 results that were blended in our RMSE=0.8712 submission. It is important to note that the major reason for using all these results was convenience, as we anyway accumulated them during the year. This is the nature of an ongoing competition. However, in hindsight, we would probably drop many of these results, and recompute some in a different way. We believe that far fewer results are really necessary. For example, based on just three results one can breach the RMSE=0.8800 barrier: A blend of #8, #38, and #92, with weights 0.1893, 0.4225, and 0.4441, respectively, would already achieve a solution with an RMSE of 0.8793. Similarly, combining #8, #38, and #64 yields RMSE=0.8798. Notice that these combinations touch the main approaches (k-NN, factorization, RBMs and asymmetric factor models). In addition, we have found that at most 11 results suffice for achieving a blend with above 8% improvement over Cinematch score.

Appendix 1 - Estimating similarity scores from binary data

Similarity scores among items are a key component within many collaborative filtering techniques. Here, we suggest a similarity measure for two items - i and j - based only on their "binary rating history". That is the identity of the users that rated them, rather than the actual ratings that they got. We have found, quite surprisingly, that in some occasions, this similarity score was more effective than a score which is based on the ratings themselves.

An obvious input to this score is n_{ij}, the number of users who viewed (or, "rated") both items.

However, $n_{ij} = 5$ means very different things depending on whether n_i and n_j, the number of viewings of each item, are on the order of 10 each or 200 each. Consequently, some rescaling seems necessary. One option is to use n_{ij}/x_{ij}, where $x_{ij} = n_i n_j / n$ and n is the total number of ratings. We believe that there are better choices for x_{ij}.

Consider two movies that have each been rated 10 times, but differ as follows. Movie j was always rated by someone who had rated only five other movies, while Movie k was always rated by active viewers who had each rated 100 other movies. That is, Movie j is part of 50 pairs of movies rated by the same user (including multiple occurrences), while Movie k is part of 1000 pairs. If $n_{ij} = 3$, that is much stronger evidence of similarity than if $n_{ik} = 3$.

Let N_i equal the number of pairs involving Movie i; that is, $N_i = \sum_{j \neq i} n_{ij}$, and let $N = \sum_i N_j$ equal twice the total number of pairs. We propose

$$x_{ij} = N_i N_j/(N-N_i) + N_i N_j/(N-N_i) .$$

This should approximately standardize n_{ij} in the sense that

$$\sum_j n_{ij} \approx \sum_j x_{ij} \quad \text{for all } i.$$

Appendix 2 – Deriving Similarity Scores from Movie Titles [EditSim]

Some similarity information between movies can be captured from the movie titles and release year provided in the Netflix Prize data. The resulting similarity values were proved useful within the neighborhood-aware factorization. However, we doubt their utility to the overall blend. For completeness, we list below the exact formula that we used for deriving these similarity weights.

The following measure is based on our prior experience with disambiguating a user's input. We have found that a combination of plain edit distance, with an emphasis on full words and prefixes could better uncover the user's intention.

Let us concentrate on two movies, with titles *ttl1* and *ttl2*, and release years *year1* and *year2*, respectively. We use the following notation:

- |*str*| is the length of string *str*.
- editD(*ttl1*,*ttl2*) is the edit distance between *ttl1* and *ttl2*.
- jointPrefix(*ttl1*,*ttl2*) is the longest prefix of *ttl1* and *ttl2*.
- prefixBonus(*ttl1*,*ttl2*) is min(5, |jointPrefix(*ttl1*,*ttl2*)|/3).
- #overlaps(*ttl1*,*ttl2*) is the number of full words appearing in both *ttl1* and *ttl2*.

Our adjusted edit distance is defined as:

$$dist = \frac{\text{editD}(ttl1, ttl2)}{\max(|ttl1|, |ttl2|) \cdot \text{prefixBonus}(ttl1, ttl2)}$$

And the derived similarity score between the two movies is:

$$sim = \left(\left(0.2 \cdot |year1 - year2| + dist \right) \cdot 0.7^{\#\text{overlaps}(ttl1, ttl2)} \right)^{-0.3}$$

Acknowledgments

We would like to thank AT&T Labs for supporting and facilitating this work. We are very lucky to work in a place that values true long term research.

We had interesting discussions with some of our competitors. Especially, we would like to thank the ML@Toronto team for making their inspiring work publicly available. To Lester Mackey from Dinosaur Planet team for his clear explanations on RBMs. To Arek Paterek for NSVD1/2. To all competitors, especially the Gravity team, for giving us a hard time and driving us to distil our techniques and seek new directions. Also, to all people that took their time for posting in the web forum. Finally, we are grateful to Netflix for putting on this amazing competition and conducting it flawlessly.

References

1. R. Bell and Y. Koren, ``Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights", *IEEE International Conference on Data Mining (ICDM'07)*, IEEE, 2007.

2. R. Bell and Y. Koren, ``Improved Neighborhood-based Collaborative Filtering'', *KDD-Cup and Workshop*, ACM press, 2007.
3. R. Bell, Y. Koren and C. Volinsky, ``Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems'', *Proceedings of the 13th ACM Int. Conference on Knowledge Discovery and Data Mining (KDD'07)*, ACM press, 2007.
4. A. Paterek, ``Improving regularized singular value decomposition for collaborative filtering'', *KDD-Cup and Workshop*, ACM press, 2007.
5. R, Salakhutdinov, A. Mnih and G. Hinton, ``Restricted Boltzmann machines for collaborative filtering'', *Proceedings of the 24th International Conference on Machine Learning (ICML'07)*, 2007.
6. Y. Zhang and J. Koren, ``Efficient Bayesian Hierarchical User Modeling for Recommendation Systems'', *Proceedings of the 30st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'07)*, ACM press, 2007.