Adaptive Objective Functions and Distance Metrics for Recommendation Systems

Michael C. Burkhart[0000−0002−2772−5840] and Kourosh Modarresi[0000−0002−9419−8235]

Adobe Inc., 345 Park Ave, San José, California 95110 USA
{mburkhar, modarres}@adobe.com

Abstract We describe, develop, and implement different models for the standard matrix completion problem from the field of recommendation systems. We benchmark these models against the publicly available Netflix Prize challenge dataset, consisting of user's ratings of movies on a 1-5 scale. While the original competition concentrated only on RMSE, we experiment with different objective functions for model training, ensemble construction, and model/ensemble testing.
Our best-performing estimators were (1) a linear ensemble of base models trained using linear regression (see ensemble e_1, RMSE: 0.912) and (2) a neural network that aggregated predictions from individual models (see ensemble e_4, RMSE: 0.912). Many of the constituent models in our ensembles had yet to be developed at the time the Netflix competition concluded in 2009. To our knowledge, not much research has been done to establish best practices for combining these models into ensembles. We consider this problem, with a particular emphasis on the role that the choice of objective function plays in ensemble construction.

1 Background

In 2006, Netflix released a dataset containing 100,480,507 movie ratings (on a 1-5 scale) from m =480,189 users on n =17,770 movies [10]. The set was divided into 99,072,112 training points and 1,408,395 probe points for contestants to train and validate models. Netflix's in-house algorithm Cinematch scored an RMSE (root mean square error) of 0.9514 on the probe set. The prize of one million dollars went to the first contestants to improve RMSE on a hidden test set by 10%. A combined team of "Bellkor in BigChaos" and "Pragmatic Theory" accomplished this in 2009 with an RMSE of 0.8558 on the probe data [9,61,79]. As one of the largest real-life datasets available, the Netflix Prize data remains a benchmark for innovations in recommender systems today.

1.1 Results

Tables 1 and 2 catalogue the final results. Full details and methods follow in the subsequent sections.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_43

Table 1. Normalized model performance under different metrics.

model	name	n. MAE	n. RMSE	n. L_3	n. L_4	n. Cos. Dist.
m_0	baseline	0.878	0.901	0.913	0.923	0.807
m_1	irlba 5	0.860	0.888	0.903	0.916	0.784
m_2	irlba 7	0.857	0.886	0.902	0.915	0.781
m_3	irlba 13	0.853	0.883	0.899	0.913	0.775
m_4	softImpute 5	0.787	0.841	0.876	0.904	0.702
m_5	softImpute 7	0.780	0.837	0.874	0.903	0.695
m_6	softImpute 13	0.773	0.836	0.876	0.909	0.692
m_7	softImpute 100	0.824	0.903	0.952	0.989	0.806
m_8	movie k-NN	0.855	0.901	0.924	0.942	0.808
m_9	user k-means	0.845	0.893	0.922	0.947	0.794
m_{10}	avg of m_8, m_9	0.832	0.869	0.890	0.906	0.751
m_{11}	cross k-NN	0.816	0.862	0.891	0.915	0.738
m_{12}	time-aware cross k-NN	0.793	0.862	0.907	0.943	0.735
m_{13}	neural v5	0.784	0.854	0.897	0.932	0.723
m_{14}	neural v11	0.780	0.830	**0.861**	**0.886**	0.684
m_{15}	neural v13	0.771	0.836	0.879	0.915	0.695
m_{16}	time-aware neural	**0.764**	**0.829**	0.87	0.904	**0.681**
m_{17}	neural one-hot	0.791	0.842	0.874	0.901	0.702
m_{18}	time-binned SVD	0.827	0.873	0.900	0.922	0.757
m_{19}	WALS	0.778	0.837	0.876	0.909	0.695
m_{20}	xgboost	0.79	0.845	0.879	0.908	0.708
m_{21}	Gaussian factorization	0.769	0.839	0.885	0.922	0.696
m_{22}	sparse tensor factorization	0.87	0.901	0.917	0.931	0.808
m_{23}	Poisson factorization	1.013	1.002	0.971	0.942	0.741
m_{24}	factorization machine	0.862	0.892	0.908	0.921	0.792
m_{25}	VAE	0.881	0.905	0.917	0.927	0.814

Table 2. Normalized ensemble performance under different metrics.

ensemble	name	n. MAE	n. RMSE	n. L_3	n.L_4	n. Cos. Dist.
e_0	subset avg	0.762	0.814	0.849	0.878	0.658
e_1	linear regression	0.750	**0.808**	**0.847**	0.879	0.649
e_2	random forest	0.757	0.811	**0.847**	**0.877**	0.653
e_3	xgboost	0.754	0.809	0.848	0.880	0.649
e_4	one-hot NN	**0.749**	**0.808**	0.848	0.881	**0.648**
e_5	neural network	0.752	0.811	0.851	0.885	0.653

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_43

2 Problem Description

Let M be an $m \times n$ matrix where entry $M_{ij} \in \{1, \ldots, 5\}$ contains user i's rating of movie j. The matrix completion problem aims to recover the matrix M from a subset $\Omega \subset [m] \times [n]$ of its entries. We let $\mathcal{P}_\Omega(M)$ denote the projection of M onto this subset, which amounts to zeroing out unobserved elements of M. We further divide the probe data randomly into a test set of 1,000,000 points and a validation set of the remaining 408,395 points. Our aim is to construct ensembles of predictors. We build individual predictors using the training set and build ensemble estimators using the validation set. We then report RMSE performance on the test set.

As the de facto standard error for regression tasks, RMSE penalizes larger errors more than MAE (mean absolute error). However, the extent to which RMSE accurately represents misclassification loss remains debatable. For example, correctly distinguishing between 3- and 5-star ratings may prove much more important than distinguishing between 1- and 3-star ratings [30]. If the ultimate goal is to produce a user's top-N movies, [17] argue precision- and recall-based metrics much better characterize success.

3 SVD

SVD methods play an important role in matrix completion. We discuss this approach first because many of the subsequent methods benefit greatly from leveraging a learned SVD decomposition, either by using SVD parameters for initialization or to estimate user-user and movie-movie similarities.

Under the common additional assumption that M is of low rank r, where $r \ll \min\{m, n\}$, we can consider the SVD of M given by $M = U\Sigma V^\top$, where U is an $m \times r$ matrix with orthonormal columns, Σ is an $r \times r$ diagonal matrix of positive entries, and V is an $r \times n$ matrix with orthonormal columns. Such a matrix has $\mathcal{O}(mr)$ degrees of freedom (assuming, as in our case, that $m > n$). Considering uniform sampling with replacement as a coupon collector's problem, we then require at least $\mathcal{O}(mr \log m)$ entries from M in order for a successful reconstruction [15]. We hypothesize that users rate movies based upon r underlying features, having importance relative to the singular values of M. The rows of U correspond to feature weightings for each user and the rows of V correspond to feature weightings for each item.

3.1 Low Rank Recovery

[14,15] described sufficient conditions on the incoherence of the rows of U and V such that M could be recovered with high probability through nuclear norm-minimization. Recall that the nuclear norm is given $\|M\|_* = \sum_{k=1}^{r} \sigma_k(M)$ where $\sigma_k(M)$ denotes the kth singular value of M. The precise optimization problem was to minimize $\|\hat{M}\|_*$ subject to $\mathcal{P}_\Omega(M) = \mathcal{P}_\Omega(\hat{M})$, where $\hat{M}$ denotes the estimate for M. Such a choice of objective function makes the problem convex

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_43

(indeed, all norms are convex), as opposed to minimizing rank($\hat{M}$). In a similar vein, [52] developed "soft-thresholded" SVD to find $\hat{M}$ minimizing $\frac{1}{2}\|\mathcal{P}_\Omega(\hat{M} - M)\|_{\mathrm{F}}^2 + \lambda\|\hat{M}\|_*$, where $\|\cdot\|_F$ denotes the Frobenius matrix norm.

4 Factorization Models

We implemented numerous approaches to matrix factorization. In this approach, we estimate $\hat{M} = UV^\top$ where U is an $n \times k$ matrix of user factors and V is an $m \times k$ matrix of item factors. Unconstrained matrix factorization [75,76] simply finds $\hat{U}, \hat{V} = \arg\min_{U,V} \|\mathcal{P}_\Omega(UV^\top - M)\|^2$, where the matrix norm is the Frobenius norm, and predicts $\hat{M} = \hat{U}\hat{V}^\top$. The straightforward mathematical description leaves a handful of implementation choices. We can initialize U, V either randomly or from the SVD decomposition by taking, for example, $U\sqrt{\Sigma}, \sqrt{\Sigma}V$ from a learned SVD model. We can perform optimization with the whole dataset in memory or perform batch-based optimization by iterating over subsets of the training data.

We also implemented Tikhonov-regularized [77,78] matrix factorization [66,73] to solve $\hat{U}, \hat{V} = \arg\min_{U,V} \|\mathcal{P}_\Omega(UV^\top - M)\|^2 + \lambda(\|U\|^2 + \|V\|^2)$ and non-negative matrix factorization [46,47], with the constraint that all entries of U, V be non-negative. In our experience, model performance seemed tightly coupled with initialization.

4.1 Accounting for Implicit Preferences

Hu et al. developed a weighted matrix factorization method that accounts for the implicit preference a user gives to a movie through the act of watching and rating it [38]. Called weighted alternating least squares (WALS), this method seeks $\hat{U}, \hat{V} = \arg\min_{U,V} \|\mathcal{P}_\Omega\big(\sqrt{W} \odot (UV^\top - M)\big)\|^2 + \lambda(\|U\|^2 + \|V\|^2)$ where W denotes the number of movies a user has rated and $\odot$ denotes element-wise multiplication. We used the contributed TensorFlow model and initialized with SVD output.

4.2 Neural Network Matrix Factorizaton

This estimator constituted a feedforward fully-connected neural network mapping learned representation vectors for users and movies through the network to predict the corresponding rating. Such an approach is commonly referred to as neural network matrix factorization [21,32]. Learned model parameters consist of m user vectors in $\mathbb{R}^r$, n movie vectors in $\mathbb{R}^r$, and all parameters for the neural network. We initialized user vectors with the U matrix and the movie vectors with the V matrix from the soft-thresholded SVD model. Neural network parameters received Glorot uniform initialization [28]. For each batch, we performed three training steps: neural network parameters were updated, user representations were updated, and movie representations were updated. We applied Tikhonov L_2-regularization to U and V. For all parameter updates, we used the Adam

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_43

optimizer [41] that maintains different learning rates for each parameter like AdaGrad [20] and allows these rates to sometimes increase like Adadelta [84] but adapts them based on the first two moments from recent gradient updates. We used a leaky rectified linear unit (ReLU) activation [57,50], and applied dropout after the first hidden layer to prevent overfitting [74]. Neither Nesterov momentum-aided Adam [19] nor Batch Normalization [39] appeared to improve our results. Many different versions of neural networks were developed with minor variations in architecture, initialization, and training.

4.3 Probabilistic Matrix Factorization

Factorization can also be performed in a probabilistic setting, by specifying a generative graphical model and then finding the maximum a posteriori (MAP) parameters [70] or by performing Gibbs sampling in a Bayesian setting [69]. In Gaussian matrix factorization, we model $M_{ij} \sim^{\text{i.i.d.}} \mathcal{N}(U_i V_j^\top + b_{ij}, \sigma^2)$ where, as before, U is an $n \times k$ matrix of user factors and V is an $m \times k$ matrix of item factors. The b_{ij} denote the average of the mean rating from user i and the mean rating of movie j, and account for user- and movie- effects. We learn U and V to maximize the log likelihood of the observed data under this model. In Poisson matrix factorization [29], $M_{ij} \sim^{\text{i.i.d.}} \text{Poisson}(U_i V_j^\top + b_{ij})$. We predict $\hat{M}_{ij} = \mathbb{E}[\hat{X}_{ij} | \hat{X}_{ij} \in \{1, \ldots, 5\}]$ where $\hat{X}_{ij} \sim \text{Poisson}(\hat{U}_i \hat{V}_j^\top + b_{ij})$ and $\hat{U}, \hat{V}$ denote our learned model parameters.

4.4 Factorization Machine

A factorization machine [64,65] of second degree learns a regression model $\hat{y}(x) = w_0 + \sum_{i=1}^{\ell} w_i x_i + \sum_{1 \le i < j \le \ell} \langle v_i, v_j \rangle x_i x_j$ for parameters $w_k \in \mathbb{R}$ and $v_k \in \mathbb{R}^\ell$, $k = 1 \ldots, \ell$. Such models were designed for sparsity, and in our case, we let $x \in \mathbb{R}^{m+n}$ denote a one-hot vector representation for the user concatenated with a one-hot vector representation for the movie.

5 Neighborhood Models

Neighborhood-based techniques prove to be useful ingredients in a Netflix ensemble [42,80]. A k-NN model estimates a function's value at a test point by averaging the values of the k nearest training points. In our case, we can predict the rating for a (user, movie)-pair by averaging the user's ratings for the k nearest movies or averaging the k nearest users' ratings of the movie. To do so, we must first set k and a metric for comparing two movies (or users). Given the sparseness of the Netflix dataset, we must also account for the cases where we have no training data on any of the k nearest neighbors of a given test point. In this case, we typically fall back to the baseline global rating. A lower value of k tends to increase the accuracy of the ratings we can calculate, but also increases the number of test points for which we have insufficient training data.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_43

In addition to comparing the performance yielded by choice of metric, it may also be illustrative to consider how the choice of metric impacts training data availability. Suppose, given a (user, movie)-pair, the user tends to have rated many more of the 20 nearest movies under the cosine metric than of the 20 nearest under the Euclidean metric. Under the hypothesis that the act of expressing a rating indicates preference (the ratings matrix is not revealed uniformly at random), this fact might also provide us with information, independent of how closely the ratings aligned to the target.

In [6], Bell, Koren, and Volinksy remark that neighborhood-based kernel regression approaches may fail to account for relationships in the similarity space. They describe Lord of the Rings as an example, where a neighborhood in movie-movie similarity space might include all three movies from the trilogy, causing the underlying effect from the trilogy to be counted three times. To account for this, they optimize weights with shrinkage instead of relying on a predefined similarity metric [5,7]. They also allow a neighborhood based method to defer judgement, when provided insufficient or low-quality neighborhood information [8]. There are also factorized versions of learned user similarity [80].

5.1 k-NN on SVD Latent Space

Consider the r-dimensional rows of U from the soft-thresholded SVD decomposition of $\mathcal{P}_\Omega(M)$. These vectors give a dense, low-dimensional (we let $r = 5$) representation for each user. We use a k-d tree [11] to find the $k = 15$ nearest neighbors for each user according to the Euclidean metric. For a given (user, movie)-pair in the probe set, we determine if any of the user's neighbors rated the queried movie, and if so, calculate a weighted average over these ratings, where the weights are proportional to the exponentiated negative distance between the user and her neighbors (*cf.* Nadaraya–Watson kernel-regression [56,82]).

A smaller value for k restricts to only the most similar neighbors, and so decreases the bias of this estimate. However, it also increases the chance that very few (or none) of the neighbors will have expressed a rating for the given movie. In the case that fewer than three of the $k = 15$ nearest neighbors to a user expressed a preference for a queried movie, then this method does not return a rating, and the average ensemble is taken over the remaining estimators. This allows the estimator to abstain from rating when it is not sufficiently confident, and elegantly fall back to estimators that will be more reliable for a given (user, movie)-pair. In a similar vein, we can cluster users according to k-means and use the above approach with cluster members in place of neighbors. Clustering can yield improved efficiency through memoization [54], as ratings for a given cluster need only be computed once, and can then be applied to subsequent queries. We also created a similar estimator that instead operates on the latent representation for movies.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_43

5.2 Crossing User Neighborhoods with Movie Neighborhoods

The original k-NN approach would find neighbors for either users or movies and then aggregate ratings along a vector in the dual dimension (movies or users, respectively). It is possible instead to use k-NN to find neighbors for both rows and columns, and then aggregate along the sub-matrix consisting of the cross product between neighboring users and neighboring movies. In other words, to predict on a rating for user i on movie j, we would find indices $\mathcal{N}_{u_i} \subset [m]$ corresponding to the neighbors of user i, and indices $\mathcal{N}_{v_j} \subset [n]$ corresponding to the neighbors of movie j, and compute a weighted average over the available rankings in $\mathcal{N}_{u_i} \times \mathcal{N}_{v_j}$, where the weights account for distances in user-space, movie-space, and the difference in time between the ratings. This allows us to leverage ratings of similar movies provided by similar users.

6 Gradient Boosted Trees

Ensemble methods combine multiple weak learners (estimators) into a single strong estimator [18,22,31,71]. Breiman's bootstrap aggregating ("bagging") approach trains multiple learners (in parallel) on bootstrapped samples. In contrast, boosting algorithms like Adaboost [25] and gradient boosting [51,26,27] iteratively add weak learners to improve an ensemble, concentrating effort on currently misclassified examples. (Note that concentrating on currently misclassified examples is not required of a boosting algorithm: see, for example, Boost by Majority [23] and Brown Boost [24].) In gradient boosting, we supply a loss function $L(\cdot, \cdot)$ and a method to train new weak learners h_i; in our case, we use regression trees [13]. For our application, we learned ratings for a (user, movie) pair as a function of their representations in thresholded SVD feature space. We used XGBoost [16] to build the estimator.

7 Variational Autoencoding

Variational autoencoding learns parameters for an autoencoder using a common Bayesian technique known as variational inference. An autoencoder models the identity function with a neural network [68,33]. Its architecture includes a hidden layer of relatively small dimensionality that serves as an information bottleneck. Upon training, the output from this layer yields a lower-dimensional representation of the original data. If we restrict to linear maps and impose L_2 loss on our reconstruction, autoencoding solves for the principal components from PCA [59,37]. In this way, we can consider autoencoding to be a nonlinear extension of PCA.

In Bayesian statistics, variational inference approximates intractable integrals (expectations) through optimization, by substituting the integrand (probability distribution) for the closest member of a parametrized family of distributions [40]. When optimization is batch-based, this process is known as stochastic variational inference [67,36].

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_43

A variational autoencoder learns a probabilistic autoencoding model, as two conditional distributions described by neural networks. The encoding distribution $q_\theta(z|x)$ describes how to sample the latent low-dimensional representation from an observation x and the decoding distribution $p_\varphi(x|z)$ describes how to sample a reconstructed x from the latent representation. Optimization aims to maximize $\mathcal{L}(\theta, \varphi) = \mathbb{E}_{q_\theta(z|x)}[\log p_\varphi(x|z)] - D_{\mathrm{KL}}\big(q_\theta(z|x)||p(z)\big)$ For computational expediency, the expectations above are often approximated via single-sample Monte Carlo integration [53]. In particular, for the ith datapoint X_i, we sample $Z_i \sim q_\theta(\cdot|X = x_i)$and form the unbiased approximations $\mathbb{E}_{q_\theta(z|x)}[\log p_\varphi(x, z)] \approx \log p_\varphi(X_i, Z_i)$ and $\mathbb{E}_{q_\theta(z|x)}[\log q_\theta(z|x)] \approx \log q_\theta(Z_i|X_i)$.

Multiple authors have implemented VAE's for collaborative filtering. [48] learned item representations from known content data. [49] concentrated on implicit ratings data; they consider observations in the form of a single user's (sparse) vector counts for item consumption, and argued that their two adjustments, using a multinomial likelihood and adjusting the VAE objective, were key to their performance.

We take $X_i \in \mathbb{R}^{17770}$ to be user i's ratings for each movie (more precisely, the residual ratings after subtracting off half of user i and movie j's mean ratings). We model $q_\theta(z|x) = \eta_{10}(z; f_1(x), \exp(f_2(x)I_{10}))$ where η_{10} denotes a 10-dimensional Gaussian, I_{10} is the identity matrix, and f_1, f_2 are leaky relu-activated neural networks. Here, θ corresponds to the parameters for the neural networks f_1, f_2. We model $p_\phi(z_i|x) = \eta_{m_i}(z; g(x), I_{m_i}))$ where m_i denotes the number of movies user i rated, g is a leaky relu-activated neural networks with a single hidden layer, and ϕ denotes the parameters for g.

8 Incorporating Rating Time

The Netflix training and probe sets include a time stamp for each rating event. We consider ways to leverage the effect of time in our model.

8.1 Time-aware Neural Factorization

Building on the success of Neural Network Matrix Factorization, this model added two time components as inputs to the neural network: (1) the time of rating, normalized to lie in $[0, 1]$ and (2) the approximate number of years between the movie's release and the time of rating. As updates to U and V are sparse (any given row only updates a handful of times for each run through the data set), we used a Nesterov Momentum optimizer [62,58] to train them, while continuing to apply the Adam optimizer for the neural network parameters (all of which are updated at each training step). Newer optimizers such as Adam tweak the learning rate for each parameter depending on a window of previous gradients for each parameter. This approach may not be best when updates to a given parameter occur only sporadically [63], so here we use Nesterov.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_43

8.2 Neural One-Hot Factorization with a Time Component

In this approach, we designed a neural network that takes user- and movie-representations, and time features "movie release year" and "time of rating" and "time of user's first rating," and outputs a probability distribution on $\{1, \ldots, 5\}$. Training minimizes the cross-entropy between the the (point-mass, or slightly modified point-mass) distribution on the underlying label and the model's predicted distribution. In addition to providing estimates for (user, movie, time)-ratings, this model allows us to predict the variance or uncertainty of our estimate.

8.3 Time-Binned SVD

We can partition the training and probe data into approximately equally sized bins based on the time stamps associated to them (so ratings that occur around the same time will be placed in the same or a neighboring bin) and learn a separate SVD model for each time window. Each of these models can then be used to predict a rating for a given (user, movie, time) probe pair, and a weighted average formed over all such predictions, with a higher weight given to the bin into which the query was placed.

8.4 Tensor Factorization

After partitioning our data into time bins (in the same way as for time-binned SVD), we can view our training data as a ratings tensor, where the users by movies matrix now extends along a third, temporal dimension. This allows us to perform time-aware factorization into three tensors, one for each dimension. Hitchcock pioneered a generalization of SVD to tensors, known as the minimal canonical polyadic (CP) decomposition [34] , yielding the model $M = \sum_{i=1}^{r} \lambda_i a_i^1 \otimes a_i^2 \otimes a_i^3$. We initialize with higher-order SVD [35,81,45] and use alternating least squares to fit the model.

9 Ensembling

Famously, the winning solution to the Netflix Prize challenge consisted of a blend of 107 different models [9,61,79]. Our best-performing models, too, aggregated predictions from other models. After building individual models using only data from the training set, we built ensembles of models using data only from the validation set. We distinguish ensembles with the letter e from our base models lettered m.

9.1 Average over a Selected Subset

Considering all $\binom{19}{10}$ size 10 subsets of $\{m_1, \ldots, m_{19}\}$, we found the subset whose simple average produced the smallest RMSE on the validation set. The average of these models was then computed for the test set. We performed the brute-force search with the Numba package that allows for just-in-time compilation (to LLVM) and parallelized for-loops.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_43

9.2 Linear Regression

We performed stepwise variable selection using the Bayesian Information Criterion [72] (see also Akike's Information Criterion [2]) for a multiple linear regression model. We also used linear regression to select the most informative subset of 10 predictors [55].

9.3 Random Forests for Regression

We used Breiman's random forest regression algorithm [12] on the 10 top predictors, as determined by linear regression in the above section. We also tried a boosting approach for ensembling, the XGBoost algorithm [16]. The importance matrix calculated from boosting gives the top 10 models, in order, as: m_{14}, m_{16}, m_{15}, m_{19}, m_7, m_{13}, m_{17}, m_{12}, m_{18}, m_{10}.

9.4 Neural Network Regression

We trained neural networks on the validation set using predictions from individual models as inputs and true values on the validation set as outputs. We adopted two main architectures: (1) a direct continuous-valued function that was trained to minimize RMSE, and (2) a distributional function that was trained to minimize cross-entropy between its pdf-outputs and one-hot representations of the true values. These models were then applied to the test set to measure performance.

9.5 Impact of Objective Function on Ensemble Building

To illustrate the role that the choice of objective function plays in ensemble building, we took the boosted ensemble model and optimized it on the validation data under a range of different objective functions. See Table 3 for results. As extreme gradient boosting uses second derivatives of the objective function, we do not include L_1 or Huber loss.

Table 3. RMSE performance after optimizing e_5 under different objective functions.

objective	n. MAE	n. RMSE	n. L_3	n. L_4	n. L_5	n. Cos. Dist.
L_2	**0.752**	**0.811**	0.851	0.885	0.912	**0.653**
L_3	0.780	0.817	**0.842**	0.863	0.881	0.655
L_4	0.810	0.836	0.846	**0.855**	0.864	0.664
L_5	0.844	0.859	0.858	0.857	**0.857**	0.677
Cosh	0.766	0.813	0.843	0.868	0.889	**0.653**

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_43

10 Conclusions

The vast majority of benchmarks against the Netflix dataset report only RMSE performance, in line with the prize's original objective. For estimators that aim to select a user's top n movies, people sometimes consider precision-recall metrics. We summarize our results under numerous metrics in Table 1 for base models and Table 2 for ensembles. We normalize the L_p metrics by diving by the loss obtained by predicting the mean training value for all test points. For example, predicting the training mean (3.67 stars) for all movies in the test set yields an RMSE of 1.127, so all reported normalized RMSE's correspond to standard RMSE divided by 1.127.

Ensembles proved essential to the winning solution for the Netflix Grand Prize. In 2009 when the competition concluded, matrix factorization and neighborhood methods provided the fundamental components from which the ensembles were built. Since 2009, researchers introduced many new machine learning approaches for recommender systems including neural network matrix factorization, factorization machines, extreme gradient boosting, and variational autoencoding. These methods have been tested individually, but little work has been done to consider how these new approaches can be combined to form more effective ensemble estimators. This paper provides a first step in that direction.

References

1. Abadi, M., et al.: TensorFlow: Large-scale mach. learn. on heterogeneous systems. In: USENIX. pp. 265–283 (2016)
2. Akaike, H.: A new look at the statistical model identification. IEEE Trans. Automat. Contr. **19**(6), 716–723 (1974)
3. Baglama, J., Reichel, L.: Augmented implicitly restarted Lanczos bidiagonalization methods. SIAM J. Sci. Comput **27**(1), 19–42 (2005)
4. Behnel, S., Bradshaw, R., Citro, C., Dalcín, L., Seljebotn, D.S., Smith, K.: Cython: The best of both worlds. Comput. Sci. Eng. **13**(2), 31–39 (2011)
5. Bell, R., Koren, Y.: Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In: IEEE Int. Conf. Data Min. pp. 43–52 (2007)
6. Bell, R., Koren, Y., Volinsky, C.: Modeling relationships at multiple scales to improve accuracy of large recommender systems. In: SIGKDD. pp. 95–104 (2007)
7. Bell, R.M., Koren, Y.: Improved neighborhood-based collaborative filtering. In: SIGKDD (2007)
8. Bell, R.M., Koren, Y.: Lessons from the Netflix prize challenge. SIGKDD Explorations **9**(2), 75–79 (2007)
9. Bell, R.M., Koren, Y., Volinsky, C.: The BellKor solution to the Netflix prize (2009)
10. Bennett, J., Lanning, S.: The Netflix prize. In: KDD Cup and Workshop (2007)
11. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM **18**(9), 509–517 (1975)
12. Breiman, L.: Bagging predictors. Mach. Learn. **24**(2), 123–140 (1996)
13. Breiman, L., Friedman, J., Stone, C.J., Olshen, R.: Classification and Regression Trees (1984)
14. Candès, E.J., Recht, B.: Exact matrix completion via convex optimization. Found. Comput. Math. **9**(6) (2009)

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_43

15. Candès, E.J., Tao, T.: The power of convex relaxation: Near-optimal matrix completion. IEEE Trans. Inf. Theory **56**(5), 2053–2080 (2010)
16. Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: SIGKDD. pp. 785–794 (2016)
17. Cremonesi, P., Koren, Y., Turrin, R.: Performance of recommender algorithms on top-n recommendation tasks. In: RecSys. pp. 39–46 (2010)
18. Dasarathy, B.V., Sheela, B.V.: A composite classifier system design: Concepts and methodology. Proc. IEEE **67**(5), 708–713 (1979)
19. Dozat, T.: Incorporating Nesterov momentum into Adam. Int. Conf. Learn. Represent. (2016)
20. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12**, 2121–2159 (2011)
21. Dziugaite, G.K., Roy, D.M.: Neural network matrix factorization (2015), arXiv:1511.06443
22. Efron, B.: Bootstrap methods: Another look at the jackknife. Ann. Stat. **7**(1), 1–26 (1979)
23. Freund, Y.: Boosting a weak learning algorithm by majority. Inf. Comput. **121**(2), 256–285 (1995)
24. Freund, Y.: An adaptive version of the boost by majority algorithm. Mach. Learn. **43**(3), 293–318 (2001)
25. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci. **55**(1), 119–139 (1997)
26. Friedman, J.H.: Greedy function approximation: A gradient boosting machine. Ann. Stat. **29**(5), 1189–1232 (2001)
27. Friedman, J.H.: Stochastic gradient boosting. Comput. Stat. Data Anal. **38**(4), 367–378 (2002)
28. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Int. Conf. Artif. Intell. Stats. vol. 9, pp. 249–256 (2010)
29. Gopalan, P., Hofman, J.M., Blei, D.M.: Scalable recommendation with hierarchical poisson factorization. In: Uncertain. Artif. Intell. pp. 326–335 (2015)
30. Gunawardana, A., Shani, G.: A survey of accuracy evaluation metrics of recommendation tasks. J. Mach. Learn. Res. **10**, 2935–2962 (2009)
31. Hansen, L.K., Salamon, P.: Neural network ensembles. IEEE Trans. Pattern Anal. Mach. Intell. **12**(10), 993–1001 (1990)
32. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., Chua, T.S.: Neural collaborative filtering. In: Int. World Wide Web Conf. pp. 173–182 (2017)
33. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science **313**(5786), 504–507 (2006)
34. Hitchcock, F.L.: The expression of a tensor or a polyadic as a sum of products. J. Math. Phys. **6**, 164–189 (1927)
35. Hitchcock, F.L.: Multiple invariants and generalized rank of a p-way matrix or tensor. J. Math. Phys. **7**, 39–79 (1928)
36. Hoffman, M., Blei, D.M., Wang, C., Paisley, J.: Stochastic variational inference. J. Mach. Learn. Res. **14**, 1303–1347 (2013)
37. Hotelling, H.: Relations between two sets of variates. Biometrika **28**, 321–377 (1936)
38. Hu, Y., Koren, Y., Volinsky, C.: Collaborative filtering for implicit feedback datasets. In: IEEE Int. Conf. Data Min. pp. 263–272 (2008)
39. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Int. Conf. Mach. Learn. pp. 448–456 (2015)
40. Jordan, M.I., Ghahramani, Z., Jaakkola, T.S., Saul, L.K.: An introduction to variational methods for graphical models. Mach. Learn. **37**(2), 183–233 (1999)

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_43

41. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. Int. Conf. Learn. Represent. (2015)
42. Koren, Y.: Factorization meets the neighborhood: A multifaceted collaborative filtering model. In: SIGKDD. pp. 426–434 (2008)
43. Lam, S.K., Pitrou, A., Seibert, S.: Numba. In: Proc. Workshop LLVM Compil. (2015)
44. Lanczos, C.: An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. J. Res. NIST **45**(4), 255–282 (1950)
45. Lathauwer, L.D., Moor, B.D., Vandewalle, J.: A multilinear singular value decomposition. SIAM J. Matrix Anal. Appl. **21**(4), 1253–1278 (2000)
46. Lee, D.D., Seung, H.S.: Learning the parts of objects by non-negative matrix factorization. Nature **401**, 788—791 (1999)
47. Lee, D.D., Seung, H.S.: Algorithms for non-negative matrix factorization. In: Adv. Neural. Inform. Process. Syst., pp. 556–562 (2000)
48. Li, X., She, J.: Collaborative variational autoencoder for recommender systems. In: ACM SIGKDD. pp. 305–314 (2017)
49. Liang, D., Krishnan, R.G., Hoffman, M.D., Jebara, T.: Variational autoencoders for collaborative filtering. In: Int. World Wide Web Conf. pp. 689–698 (2018)
50. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. Int. Conf. Mach. Learn. **30** (2013)
51. Mason, L., Baxter, J., Bartlett, P.L., Frean, M.R.: Boosting algorithms as gradient descent. In: Adv. Neural. Inform. Process. Syst., pp. 512–518 (2000)
52. Mazumder, R., Hastie, T., Tibshirani, R.: Spectral regularization algorithms for learning large incomplete matrices. J. Mach. Learn. Res. **11**, 2287–2322 (2010)
53. Metropolis, N., Ulam, S.M.: The Monte Carlo method. J. Am. Stat. Assoc. **44**(247), 335–341 (1949)
54. Michie, D.: Memo functions and machine learning. Nature **218**, 19–22 (1968)
55. Miller, A.J.: Selection of subsets of regression variables. J. Royal Stat. Soc. Ser. A **147**(3), 389–425 (1984)
56. Nadaraya, E.A.: On estimating regression. Teor. Veroyatnost. i Primenen. **9**(1), 157–159 (1964)
57. Nair, V., Hinton, G.: Rectified linear units improve restricted Boltzmann machines. Int. Conf. Mach. Learn. (2010)
58. Nesterov, Y.: A method of solving a convex programming problem with convergence rate o(1/sqr(k)). Soviet Math. Dokl. **27**, 372–376 (1983)
59. Pearson, K.: On lines and planes of closest fit to systems of points in space. Philos. Mag **2**(11), 559–572 (1901)
60. Pedregosa, F., et al.: Scikit-learn: Machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
61. Piotte, M., Chabbert, M.: The Pragmatic Theory solution to the Netflix grand prize (2009)
62. Polyak, B.T.: Some methods of speeding up the convergence of iteration methods. USSR Comput. Math. Math. Phys. **4**(5), 1–17 (1964)
63. Reddi, S.J., Kale, S., Kumar, S.: On the convergence of adam and beyond. Int. Conf. Learn. Represent. (2018)
64. Rendle, S.: Factorization machines. In: IEEE Int. Conf. Data Min. pp. 995–1000 (2010)
65. Rendle, S.: Factorization machines with Libfm. ACM Trans. Intell. Syst. Technol. **3**(3) (2012)
66. Rennie, J.D.M., Srebro, N.: Fast maximum margin matrix factorization for collaborative prediction. In: Int. Conf. Mach. Learn. (2005)

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_43

67. Robbins, H., Monro, S.: A stochastic approximation method. Ann. Math. Stat. **22**(3), 400–407 (1951)
68. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning Internal Representations by Error Propagation, vol. 1, pp. 318–362 (1986)
69. Salakhutdinov, R., Mnih, A.: Bayesian probabilistic matrix factorization using markov chain monte carlo. In: Int. Conf. Mach. Learn. pp. 880–887 (2008)
70. Salakhutdinov, R.R., Mnih, A.: Probabilistic matrix factorization. In: Adv. Neural. Inform. Process. Syst., pp. 1257–1264 (2008)
71. Schapire, R.E.: The strength of weak learnability. Mach. Learn. **5**(2), 197–227 (1990)
72. Schwarz, G.: Estimating the dimension of a model. Ann. Stat. **6**(2), 461–464 (1978)
73. Srebro, N., Rennie, J., Jaakkola, T.S.: Maximum-margin matrix factorization. In: Adv. Neural. Inform. Process. Syst., pp. 1329–1336 (2005)
74. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**, 1929–1958 (2014)
75. Tenenbaum, J.B., Freeman, W.T.: Separating style and content. In: Adv. Neural. Inform. Process. Syst., pp. 662–668 (1997)
76. Tenenbaum, J.B., Freeman, W.T.: Separating style and content with bilinear models. Neural Comput. **12**(6), 1247–1283 (2000)
77. Tikhonov, A.N.: On the stability of inverse problems. Proc. USSR Acad. Sci **39**(5), 195–198 (1943)
78. Tikhonov, A.N.: Solution of incorrectly formulated problems and the regularization method. Proc. USSR Acad. Sci **151**(3), 501–504 (1963)
79. Töscher, A., Jahrer, M.: The BigChaos solution to the Netflix grand prize (2009)
80. Töscher, A., Jahrer, M., Legenstein, R.: Improved neighborhood-based algorithms for large-scale recommender systems. In: KDD Cup (2008)
81. Tucker, L.R.: Some mathematical notes on three-mode factor analysis. Psychometrika **31**(3), 279–311 (1966)
82. Watson, G.S.: Smooth regression analysis. Sankhyā Ser. A **26**(4), 359–372 (1964)
83. Wright, M., Ziegler, A.: ranger. J. Stat. Softw. **77**(1) (2017)
84. Zeiler, M.D.: Adadelta: An adaptive learning rate method (2012), arXiv:1212.5701

A Implementation Details

In Python v3.6.0, we used Scikit-learn for k-NN and k-means [60] and Tensorflow for neural network training [1]. We performed some just-in-time for-loop optimization with Numba [43]. We used Cython [4] to precompile some Python functions for additional performance. The ttfm package provided our factorization machine implementation. We compiled Tensorflow from source with MKL (Intel) support and additional instructions.

In R v3.5.1, we used 'irlba' for truncated SVD [3] (*cf.* [44]) and 'softImpute' for soft-thresholded SVD [52]. While increasing the rank r tended to result in improved performance for both algorithms, returns were modest. We used 'MASS' for linear regression and 'leaps' for subset variable selection. We trained random forests with 'ranger' [83] and performed boosting with 'xgboost.' We used the R package 'Reticulate' and the Python package 'rp2' to transfer data between the two languages.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_43