Combining Predictions for Accurate Recommender Systems

Michael Jahrer
commendo research&consulting, Graz
University of Technology
8580 Köflach
Austria
michael.jahrer@commendo.at

Andreas Töscher
commendo research&consulting, Graz
University of Technology
8580 Köflach
Austria
andreas.toescher@commendo.at

Robert Legenstein
Institute for Theoretical Computer Science, Graz
University of Technology
8010 Graz
Austria
robert.legenstein@igi.tugraz.at

ABSTRACT

We analyze the application of ensemble learning to recommender systems on the Netflix Prize dataset. For our analysis we use a set of diverse state-of-the-art collaborative filtering (CF) algorithms, which include: SVD, Neighborhood Based Approaches, Restricted Boltzmann Machine, Asymmetric Factor Model and Global Effects. We show that linearly combining (blending) a set of CF algorithms increases the accuracy and outperforms any single CF algorithm. Furthermore, we show how to use ensemble methods for blending predictors in order to outperform a single blending algorithm. The dataset and the source code for the ensemble blending are available online [9].

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: [Data mining]

General Terms

Algorithms, Measurement, Performance

Keywords

Recommender Systems, Netflix, Supervised Learning, Ensemble Learning

1. INTRODUCTION

Recommender systems help users to discover items within large web shops, to navigate through portals or to find friends with similar interests. The most interesting applications for recommender systems have thousands of users which generate huge amounts of data. For example, online shops collect purchase data and provide each user with a personalized shopping page on the login. The sources of information used for the recommender system can be widespread. Users generate events like the purchase of a product, rating a product, creating a bookmark or clicking on a specific item. Independently of the area of application or the type of information used, it is a major goal to increase the accuracy while retaining the capability of being able to use big datasets.

Generating more accurate predictions is of general interest. For a subscription service like Netflix, good recommendations are a key to customer loyalty. In the case of online stores better recommendations directly increase the revenue. Davis et al. [6] showed how to use collaborative filtering for disease prediction.

Over the last two decades many powerful collaborative filtering algorithms were published. However, the performance can be significantly improved, if ensemble methods are used. An ensemble method combines the predictions of different algorithms (the ensemble) to obtain a final prediction. The combination of different predictions into a final prediction is also referred to as "blending". Ensemble methods were key to the solutions of the top teams in the Netflix Prize contest [20, 15, 11, 17]. The most basic blending method is to compute the final prediction simply as the mean over all the predictions in the ensemble. Better results can be obtained, if the final prediction is given by a linear combination of the ensemble predictions. In this case, the combination coefficients have to be determined by some optimization procedure, in general by regularized linear regression. Linear blending was widely used in the Netflix Prize contest, but more elaborate schemes are possible. However, not all available ensemble methods are practical for large-scale recommender systems because the massive amount of data leads to massive time- and memory consumption.

In this article, we provide a systematic empirical analysis of different blending methods on the Netflix dataset. The Netflix dataset [2] is one of the largest available benchmark datasets for collaborative filtering algorithms today. It contains about 10^8 ratings, collected in a time period of 7 years. We discuss and test several promising algorithms for blending, including neural network blendings, bagged gradient boosted decision trees, and kernel ridge regression. Our results show that linear blending is not optimal, and that it can be significantly outperformed by the presented methods.

These methods are not limited to blending collaborative filtering predictors, they can be used for supervised regression problems in general.

KDD'10, July 25–28, 2010, Washington, DC, USA.

The remaining article is structured as follows. In Section 2 we give an overview of the collaborative filtering algorithms which will be used as base predictors for the blending experiments. Section 3 introduces the algorithms used for blending, before we present the results of our analysis in Section 4. Conclusions are given in Section 5.

2. COLLABORATIVE FILTERING

Collaborative filtering uses the history of user-item events in order to predict future ones. These events can be any source of user-generated information, such as purchases, ratings, clicks, bookmarks, favorite-adds, whishlist-adds, etc. Within this work we focus on rating based collaborative filtering.

The problem of rating prediction can be described with the help of the user-item rating matrix $\mathbf{R} = [r_{ui}]$ where entry r_{ui} is the rating of user u for item i. This matrix has the size $U \times M$ with U being the number of users and M the number of items. In general $\mathbf{R}$ is sparsely filled, because a user typically does not rate every item. The goal of the prediction model is to accurately predict missing values of this matrix, i.e. to produce predictions $\hat{r}_{ui}$ for how an item i would be rated by user u. In collaborative filtering, the system infers a model from all available data in $\mathbf{R}$. One possibility to do so is to use a low-rank matrix factorization.

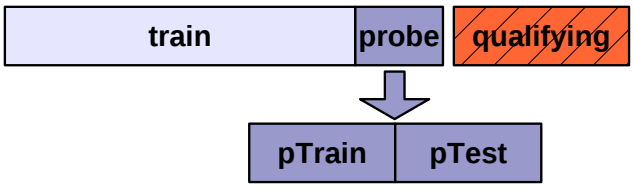

Figure 1: The Netflix dataset consists of a training set with approximately 100 million ratings, with a fixed hold out set called probe set. The goal of the Netflix competition was to predict the qualifying set, which was unknown during the competition. In order to generate a dataset for blending we used predictions from different collaborative filtering algorithms for the probe set. For our experiments we randomly split the probe set with 1.4 million ratings into two disjoint sets called pTrain and pTest.

In Table 1 we list the most useful collaborative filtering algorithms of the Netflix Prize challenge. These algorithms include k-nearest neighbor (KNN) methods (KNNitem, KNNuser), methods based on matrix factorization (SVD, AFM, SVDe), restricted Boltzmann machines (RBM), and global effects (GE). The RMSE column lists the root-mean-squared error (RMSE) that a single algorithm can achieve with good learning parameters, e.g. proper learn rate and regularization constants. Each single algorithm models the data in a different way. Therefore a suitable combination of these predictions can significantly improve the prediction performance. For example a linear combination of all single models leads to an RMSE of 0.87. The prediction algorithm of Netflix, called "Cinematch", achieved an RMSE of about 0.95 at the time when the competition was started in 2006 [2].

algorithm	RMSE (approx.)	training time	prediction time	memory
KNNitem	0.92	$O(U \cdot M^2)$	$O(Mlg(M))$	$O(M^2)$
KNNuser	0.93	$O(M \cdot U^2)$	$O(Ulg(U))$	$O(U^2)$
SVD	0.90	$O(\lvert\mathcal{L}\rvert)$	$O(1)$	$O(M+U)$
AFM	0.92	$O(\lvert\mathcal{L}\rvert)$	$O(1)$	$O(M)$
SVDe	0.88	$O(\lvert\mathcal{L}\rvert)$	$O(1)$	$O(M+U)$
RBM	0.90	$O(\lvert\mathcal{L}\rvert)$	$O(1)$	$O(M)$
GE	0.95	$O(\lvert\mathcal{L}\rvert)$	$O(1)$	$O(M+U)$
Blend	<0.87			

Table 1: A list of various collaborative filtering algorithms with asymptotic training/prediction time and memory consumption. We add approximate RMSE values on the Netflix Prize dataset (qualifying set). The prediction time is defined as the asymptotic time needed to generate a single prediction $\hat{r}_{ui}$. M is the total number of items, U is the total number of users and $|\mathcal{L}|$ the total number of ratings in the training set. Red values are critical for large scale applications.

More sophisticated blending techniques lower the RMSE below 0.87.

In the following we provide an overview of collaborative filtering algorithms listed in Table 1. Detailed explanations of these algorithms can be found in the Netflix Prize Winner Reports [20, 11, 15]. With these algorithms we construct a dataset (Table 2), which we use for the blending experiments in Section 4. This dataset is available online [9].

The provided dataset is constructed as follows. Each collaborative filtering algorithm is trained on the Netflix train set, with the probe set excluded. The predictions for the probe set are used for the blending dataset. For the experiments in Section 4 we split the probe set randomly in two equally sized sets called pTrain and pTest, as visualized in Figure 1. For the blending experiments we will train the models on the pTrain set and use the pTest set as a hold out set.

KNN item-item

A prediction $\hat{r}_{ui}$ for the rating of a user u on item i in an item based k-nearest neighborhood model is obtained by calculating a weighted sum over the ratings of the user u for the k items most similar to item i. The weights and the similarities are proportional to the correlations c_{ij} between item i and other items j. Therefore, a precalculated item-item correlation matrix $\mathbf{C}$ is useful, due to the constant access time to any item-item correlation c_{ij} in this case. This results in a memory consumption of $O(M^2)$, where M is the number of items. The training time is dominated by the calculation of the item-item correlation matrix $\mathbf{C}$, which needs $O(U \cdot M^2)$ operations. For one prediction, the KNN selects the k best correlated items to item i. This can take up to $O(M)$ operations. The sorting of this list takes $O(M \cdot \log(M))$ time, which is the asymptotic bound for the prediction time.

KNN user-user

The model is exactly the same as in the KNN item-item, but items and users are flipped. Hence the prediction and train-

nr	name	RMSE	description
1	AFM-1	0.9362	AFM, 200 features, $\eta = 1e{-}3$, $\lambda = 1e{-}3$, learnrate η is multiplied with 0.95 from epoch 30, trained for 120 epochs
2	AFM-2	0.9231	AFM, 2000 features, $\eta = 1e{-}3$, $\lambda = 2e{-}3$, trained for 23 epochs, based on residuals of KNN-4.
3	AFM-3	0.9340	AFM, 40 features, $\eta = 1e{-}4$, $\lambda = 1e{-}3$, trained for 96 epochs
4	AFM-4	0.9391	AFM, 900 features, $\eta = 1e{-}3$, $\lambda = 1e{-}2$, trained for 43 epochs
5	GE-1	0.9079	GE, 16 effects, based on residuals of KNN-1
6	GE-2	0.9710	GE, 16 effects, on raw ratings
7	GE-3	0.9443	GE, 16 effects, based on residuals of KNN-4
8	GE-4	0.9209	GE(with time), 24 effects, based on residuals of AFM-2
9	KNN-1	0.9110	KNN item, Pearson correlation, $k = 24$ neighbors, based on residuals of AFM-1
10	KNN-2	0.8904	KNN item, Set correlation [20], $k = 122$, based on residuals from a chain of algorithms RBM-KNN-GE(with time)
11	KNN-3	0.8970	KNN item, Pearson correlation, $k = 55$, based on residuals of a discrete RBM model with $nHid = 150$
12	KNN-4	0.9463	KNN item, Pearson correlation, $k = 21$, based on residuals of GE-2
13	RBM-1	0.9493	RBM, discrete, $nHid = 10$, $\eta = 0.002$, $\lambda = 0.0002$
14	RBM-2	0.9123	RBM, discrete, $nHid = 250$, $\eta = 0.002$, $\lambda = 0.0004$
15	SVD-1	0.9074	SVD, 300 features, $\eta = 8e{-}4$, $\lambda = 0.01$, trained for 158 epochs, based on residuals of 1GE (item mean)
16	SVD-2	0.9172	SVD, 20 features, $\eta = 0.002$, $\lambda = 0.02$, trained for 158 epochs, based on residuals of 1GE (item mean)
17	SVD-3	0.9033	SVD, 1000 features, with adaptive user factors (AUF [20]), $\eta = 0.001$, $\lambda = 0.015$, trained for 158 epochs
18	SVD-4	0.8871	SVD extended, 150 features, individual learnrates η and regularization constants λ are automatically tuned on the probe set [20].
19	support	-	The number of ratings per user; we take the natural logarithm of the support as additional input.

Table 2: The predictors listed above form the dataset used for the blending experiments in Section 4. The predictors are trained on the Netflix train set, the probe set being excluded. The reported RMSE values are on the probe set. The complete dataset is freely available online [9].

ing bounds are also reversed. See Table 1 for the complete list. For the Netflix Prize dataset this method is unpractical due to the huge memory consumption for precomputing the user-user correlation matrix. In this context we want to mention the possibility of learning an implicit factorization of the full user-user correlation matrix. This reduces the amount of required memory down to $O(U \cdot K)$ where K is the number of factors in the factor matrices. This enables us to keep all the user-user correlations in memory by storing the factorized version. One particular correlation is then just a dot product of the corresponding features. For details see [21].

SVD (matrix factorization)

This is probably the most popular collaborative filtering technique. A prediction is given by the dot product of a user feature vector $\mathbf{p}_u$ and the item feature vector $\mathbf{q}_i$: $\hat{r}_{ui} = \mathbf{p}_u^T \mathbf{q}_i$, leading to $O(1)$ runtime per prediction. The SVD learns two factor matrices, user features $\mathbf{P} = [\mathbf{p}_1, ..., \mathbf{p}_U]$ and item features $\mathbf{Q} = [\mathbf{q}_1, ..., \mathbf{q}_M]$ via stochastic gradient descent. There are many extensions to SVD in the literature, for example the SVD++ model described in [10]. In practice the whole training needs a few tens of epochs over the whole dataset until convergence, leading to $O(|\mathcal{L}|)$ training time. The model parameterizes two matrices with f rows. Thus, the memory consumption is $O(M + U)$. We assume the number of f is a constant, in practice usual values are e.g. $f = 50$. Both, training and prediction time have optimal asymptotic runtime behavior, which makes the SVD an excellent candidate for large scale recommendation applications.

AFM (asymmetric factor model)

The asymmetric factor model was first described by Paterek in [14]. In the plain SVD model, a user is represented by the feature vector $\mathbf{q}_u$. The AFM model represents a user by the items he has rated. Thus, no explicit user feature is stored as parameter. In other words, the AFM model parameterizes only item features. A so called "virtual user feature" $\mathbf{y}_u$ is given by $\mathbf{y}_u = |N(u)|^{-1/2} \sum_{i \in N(u)} \mathbf{p}_i$, where the set of items, which was rated by the user u, is denoted by $N(u)$. $\mathbf{p}_i$ are item-dependent features. One can show that the special normalization of the item feature sum is necessary, when assuming normal-distributed feature values. This representation offers several benefits, for example integration of new data and new users without retraining the whole model [13]. The prediction time is constant (like SVD), because one can store the precalculated virtual user features after training, $\hat{r}_{ui} = \mathbf{y}_u^T \mathbf{q}_i$. Training time is similar to SVD, because the AFM is trained with stochastic gradient descent and a batch update on the virtual features p_i.

SVD extended

The Netflix Prize dataset comes with rating date information. This enables us to add additional user and item features, based on the time and rating frequency. We define frequency as the number of votes a user gives on a particular day. For each additional feature the learn rate and regularization parameters have to be set correctly by optimizing them on a validation set, as suggested in [18]. Training time rises by a constant, therefore one obtains the same asymptotic complexity as plain SVD: $O(|\mathcal{L}|)$. The same applies for prediction time and memory consumption. Large extended

SVD models, (called SBRAMF and extensions in [20]), have shown outstanding accuracy over the rest of collaborative filtering algorithms. They are specialized SVD models and need a lot of effort in training and tuning various meta-parameters. Koren describes various ways of integrating the date information into collaborative filtering models [12].

RBM (Restricted Boltzmann Machine)

In general a Boltzmann machine is a stochastic generative model. The restricted Boltzmann machine [16] is a neural network with one input layer and one hidden layer. For collaborative filtering, the visible units correspond to items. The training is done epoch-wise over all users. For each user the visible units get activated with the items rated by the user. Learning works well with contrastive divergence learning [16] and has $O(|\mathcal{L}|)$ training time. Prediction complexity is constant, because the probabilities of the hidden layer can be precalculated user-wise, hence $O(1)$. This leads to a simple dot product enclosed by a sigmoid function for generating recommendations. The accuracy of RBMs applied on collaborative filtering problems are superior compared to AFMs because of the non-linearity. Training is performed user-wise and converges after a few ten epochs.

GE (global effects)

Global effects are based on user and item features, such as support (number of votes), mean rating, mean standard deviation, mean rating date, etc. The idea of global effects is to calculate "hand-designed" features, which are equivalent to a SVD with fixed item or user features. Bell et al. originally described ten global effects in [1]. Six additional global effects are described in [20]. The RMSE of 16 global effects applied to the Netflix Data is about 0.95. Global effects can be effective when applied to residuals of other algorithms.

Combinations

A popular method to combine collaborative filtering algorithms is residual training [20]. In residual training several models are trained sequentially. The first model is trained on the raw data. Then, the i^{th} model in the sequence is trained on the errors of the $(i-1)^{th}$ model for $i > 1$, i.e. the prediction of the $(i-1)^{model}$ is subtracted from the raw data and used as input to the i^{th} model. We found that item-item KNNs are most effective, when they are applied on residuals of RBMs. When constructing such a residual chain, the predictions of the individual algorithms in the ensemble becomes more diverse, which is beneficial for the final blend. The final blender has access to all the predictors generated by various CF models on various residuals of other models. Table 2 lists all algorithms used for this blending. Some of them are trained on raw ratings, whereas others are based on residuals of others. This aspect is explicitly denoted in the description column.

3. BLENDING

The combination of different kinds of collaborative filtering algorithms leads to significant performance improvements over individual algorithms. Blending predictions is a supervised machine learning problem. Each input vector $\mathbf{x}_i$ is the F-dimensional vector of the predictions in the ensemble. For N data samples, one collects the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$ in an $N \times F$ matrix $\mathbf{X}$ of predictions. The target values for these n data points are collected in an N-dimensional vector $\mathbf{y}$. For the case of the Netflix dataset used in this article, the entries of $\mathbf{y}$ are integer ratings between 1 and 5. The blending algorithm is formally a function $\Omega : \mathbb{R}^F \mapsto \mathbb{R}$. The input $\mathbf{x}$ is a vector of individual predictions, the output is a scalar. We want to minimize the prediction RMSE on a test set

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\Omega(\mathbf{x}_i) - y_i)^2}. \quad (1)$$

We use 18 predictors and the logarithm of the user support (Table 2) of the Netflix Prize dataset as input. Many machine learning models are not directly applicable because of the huge number of samples. Evaluation is done on the probe dataset (1.4M samples), which is a hold-out set of the 100M training set. The probe set is the same as specified by Netflix for the competition. We divided the probe set randomly into two equally sized subsets, the pTrain set and the pTest set. We note here that the individual algorithms in the ensemble were trained on the original training set, i.e. the whole dataset excluding the probe set. However, the training of the blending algorithm was done on the pTrain set, i.e., one half of the probe set. Then the evaluation was performed on the pTest set, i.e. the probe set without the pTrain set. We perform blending on the probe set because it represents the desired distribution of users and ratings we want to optimize.

We begin our empirical evaluation with simple methods like linear blending, then we move to binned blending, which is the application of learners on structured subsets. Gradient boosted decision trees and neural networks deliver most accurate results when they are combined with bagging [3]. The k-nearest neighbors algorithm and kernel ridge regression are computationally too costly to be applied to the whole pTrain set. Therefore, multiple models are trained on small random subsets of pTrain and predictions are averaged. Finally we compare the results.

Parameter selection

Every blending algorithm has dataset dependent parameters (e.g. regularization in linear regression, number of training epochs in neural networks). In order to select the correct one we use either k-fold cross validation or the out-of-bag estimate in bagging [3] as feedback for parameter selection. For the non-gradient descent based algorithms we use a simple coordinate search for the exploration (APT2 in [19]). Prediction of new samples can be done either by retraining the whole model with found parameters on all data (called "retraining") or the mean prediction of the k models in the k-fold cross validation can be used to generate predictions (called "cross validation mean" [5]). In validation with bagging we use the out-of-bag estimate as feedback. Bagging many copies of the model on slightly different training data delivers superior accuracy compared to retraining or cross validation mean. Cross validation mean delivers better results than retraining in complex models.

Linear Regression - LR

Assuming a quadratic error function, optimal linear combination weights $\mathbf{w}$ (vector of length N) can be obtained by solving the least squares problem. For any input vector $\mathbf{x}$,

the prediction is $\Omega(\mathbf{x}) = \mathbf{x}^T\mathbf{w}$. Weights $\mathbf{w}$ are calculated with ridge regression, $\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$, where $\mathbf{I}$ denotes the identity matrix. Cross-validation is used in order to select a proper ridge regression constant λ.

Binned Linear Regression

Due to the huge size of the training set one can divide the set into B disjoint subsets and determine a separate blending for each subset $b = 1, \ldots, B$. For linear regression we denote the blending weights for subset b by $\mathbf{w}_b$. Each bin should have approximately the same number of ratings. The training set can be split by using a histogram on one of the following criteria.

- Support : The support of a data point (u, i) is the number of votes by user u. The blender can now base the weighting of predictors dependent on how many rating the user has given. RBMs are prone to receive high weight when the user has only a few votes in the data. SVDs are highly weighted when much information from a user is available.
- Time : The time of a data point (u, i) is the day on which the rating r_{ui} was performed. Predictions are mixed together with time dependency. When using this binning criteria, the blender can easily model time-dependent blending.
- Frequency : The frequency of a data point (u, i) is the number of ratings from a user at day t. This criteria enables the blender to be selective, based on the user's rating day frequency. The blender has the ability to give predictions other weights when a user votes many times on a particular day.

A prediction is given by

$$\hat{r}_{ui} = \mathbf{x}^T\mathbf{w}_b. \tag{2}$$

Here, b is in general chosen based on information about user u and item i. For example, if we divide the Netflix probe-set into 5 support-bins, the following formula gives the bin b:

$$b = \begin{cases} 1, & |N(u)| < 34 \\ 2, & 35 \leq |N(u)| < 70 \\ 3, & 71 \leq |N(u)| < 146 \\ 4, & 147 \leq |N(u)| < 321 \\ 5, & \text{else,} \end{cases} \tag{3}$$

where $|N(u)|$ denotes the number of ratings of a user u. We tried neural networks as blending algorithm per bin, but this was not as successful as taking the whole data set (results are not shown).

Neural Network - NN

Small neural networks can be trained efficiently on huge data sets. The training of neural networks is performed by stochastic gradient descent. The output neuron has a sigmoid activation function with an output swing of -1 to $+1$. To generate rating predictions in the range of $[1, 5]$ we use a simple output transformation. For example the output is multiplied by $\alpha = 3.6$ and the constant $\beta = 3.0$ is added (works well on our experiments). The learning rate is η and every epoch the constant $\eta^{(-)}$ is subtracted, which helps to find a good local minima. This works better compared to an exponential learnrate decay (multiplication with a constant). No weight decay or batch update is used.

Bagged Gradient Boosted Decision Tree - BGBDT

The main drawback of a single decision tree is its moderate accuracy. The discretized output function of a tree limits its ability to model smooth functions. The number of possible output values of a tree corresponds to the number of leaves. For regression problems, such as blending predictions, this is a big disadvantage. Blending predictions should result in a smooth function.

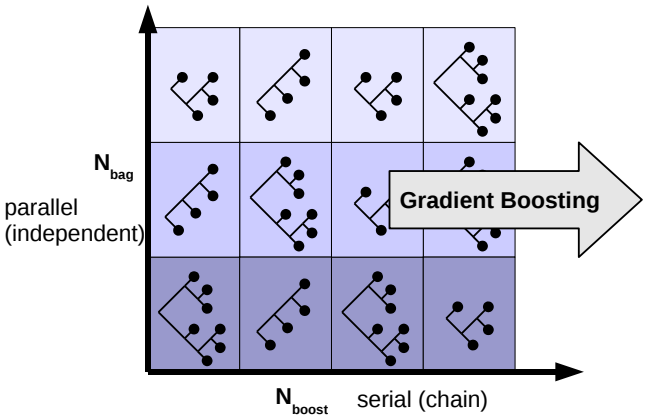

Figure 2: Bagged Gradient Boosted Decision Tree. A prediction of the BGBDT consists of results by $N_{bag} \cdot N_{boost}$ single decision trees. Both, bagging and gradient boosting improves the accuracy.

A single tree is trained recursively by splitting always that leaf which provides the output value for the largest number of training samples. The growth of the tree is stopped when the number of leaves exceeds the integer constant K. A split is found by taking the one which reduces the RMSE best. This implies that we search on all inputs a threshold value. The predicted value of a leaf is a constant.

Additionally we found it useful to add the random subspace idea in the determination of the optimal split on each node (like in random forests from Breiman [4]). The subspace size S is the number of features considered in each node to find the optimal split.

Breiman [3] shows how bagging can be applied to improve the accuracy of decision trees. Bagging is a simple schema that can be applied to every supervised learning problem in order to improve accuracy. It is based on training N_{bag} copies of the model on slightly different training sets. Each training set is produced by sampling with replacement. The samples, which are not inside the training set are used for validation. Averaging over all model copies results in the out-of-bag estimate (oob). The drawback of bagging is the increased training time. We use N_{bag} as the bagging size.

Friedman introduced an interesting idea to improve the accuracy of a learning machine. He called the technique "Stochastic Gradient Boosting" [7]. The main idea is to train multiple learners of the same type in a chain. Each model learns only a fraction of the desired function Ω, controlled by the learn rate η. This works as follows: the first model

Ω_1 in the chain is trained on the unmodified targets $\mathbf{y}_1 = \mathbf{y}$ of the dataset. The second model trains on $\mathbf{y}_2 = \mathbf{y} - \eta\,\mathbf{y}_1$. So targets in each cascade are $\mathbf{y}_i = \mathbf{y} - \sum_{j=1}^{i-1} \eta\,\mathbf{y}_j$. The final prediction model is $\Omega(\mathbf{x}) = \sum_{i=1}^{M} \eta\,\Omega(\mathbf{x})$, where M is the length of the boosting chain.

Finally we end up with a model that combines the benefits of bagging, gradient boosting and random subspace selection. The algorithm is called now BGBDT - Bagged Gradient Boosted Decision Tree (see Figure 2).

Kernel Ridge Regression Blending - KRR

The learning algorithm is described in [20]. The Gauss kernel $k(\mathbf{x}, \mathbf{y}) = exp(-(||\mathbf{x} - \mathbf{y}||^2)/\sigma^2)$ is used in our experiments. The width σ of the kernel and the regularization parameter λ are tuned with cross validation.

KRR has a training time complexity of $O(N^3)$ (invert the Gram matrix) and space requirements of $O(N^2)$, hence it is practical impossible to train on all data points. To make it work, we use a small subset of the data to train the KRR model. Doing this multiple times and average all outcomes, we obtain an accurate blending model. We evaluate the impact of the subset size and the number of averaged models, with respect to the accuracy measured in RMSE on the pTest set.

K-Nearest Neighbors Blending - KNN

The template-based KNN algorithm is very slow in predicting new samples, if the training set is large. The model is given by the following formula:

$$\Omega(\mathbf{x}) = \frac{\sum_{k \in D(\mathbf{x})} \mathbf{y}_k \cdot d(\mathbf{x}, \mathbf{x}_k)}{\sum_{k \in D(\mathbf{x})} |d(\mathbf{x}, \mathbf{x}_k)|}, \tag{4}$$

where the set $D(\mathbf{x})$ consists of indices of the k-nearest neighbors to $\mathbf{x}$ in the training samples. The distance $d(\cdot, \cdot)$ is the inverse of the Euclidean distance. The neighborhood size K is adjusted by cross validation. We use again a small subset of the training set to build the model. Averaging over many random subsets delivers our final prediction.

4. RESULTS

In the first step of training the collaborative filtering algorithms, we remove the probe set from the dataset. The evaluation of the blending techniques is performed on the probe set, which contains 1408395 samples.

We split the probe set randomly in two equally sized halves, 704197 samples are used for training (pTrain) and 704198 for testing (pTest). All reported runtimes are measured on an Intel i7 machine running at 3.8GHz with 12GB of main memory.

Linear Regression

The linear regression technique is used to obtain a baseline estimate of the RMSE on the pTest set. This is done with regularized linear regression. The ridge regression constant $\lambda = 5e{-}6$ is set to minimize the RMSE on the cross-validation set.

Table 3 shows the weights for each of the predictors in the ensemble (from Table 2) including the support and the constant input. The largest weight is assigned to the constant input because of the uncentered target values. The weight of 3.673 corresponds to the mean value of the targets. The second largest weight has the strongest KNN model, KNN-2 with a weight of 0.272. Due to the nature of linear regression, the weights are not restricted to be positive. Some of the predictors receive negative weights. This can be interpreted as negative-compensation of a particular effect in the data. With linear regression one can achieve an RMSE of 0.87525 on the pTest set.

AFM-1 (0.9362)	AFM-2 (0.9231)	AFM-3 (0.9340)	AFM-4 (0.9391)	GE-1 (0.9079)	GE-2 (0.9710)	GE-3 (0.9443)	GE-4 (0.9209)	KNN-1 (0.9110)	KNN-2 (0.8904)	KNN-3 (0.8970)	KNN-4 (0.9463)	RBM-1 (0.9493)	RBM-2 (0.9123)	SVD-1 (0.9074)	SVD-2 (0.9172)	SVD-3 (0.9033)	SVD-4 (0.8871)	log(support)	const. 1
−0.083	−0.084	−0.077	+0.088	+0.098	−0.003	−0.081	+0.176	+0.029	+0.272	−0.094	+0.010	+0.025	+0.066	−0.008	+0.094	+0.080	+0.227	−0.008	+3.673

Table 3: Blending weights of an optimal linear combination. This leads to 0.875258 RMSE on the pTest set. The RMSE on the cross validation set is 0.87552. The number in the brackets is the probe RMSE per model.

Binned linear regression

This is linear regression on predefined subsets of the training data. For each of the training and test samples, the support (number of ratings), the date (day of the rating) and the frequency (number of ratings of the user per day) is available. Based on this information we split the data into 2, 5, 10 or 20 nearly equal sized bins. We select the proper regularization constant per bin with cross validation.

type	2 bins	5 bins	10 bins	20 bins
support	0.874877 (V:0.87517)	**0.874741** (V:0.8750)	0.874744 (V:0.87499)	0.87485 (V:0.87513)
date	0.875212 (V:0.87545)	**0.875195** (V:0.87541)	0.87527 (V:0.87544)	0.87537 (V:0.87558)
frequency	0.87518 (V:0.87537)	**0.87510** (V:0.87521)	0.87512 (V:0.8752)	0.87517 (V:0.87531)

Table 4: RMSE values obtained with binned linear regression on the pTest set. The small values in the brackets below are RMSEs from the cross validation.

The best results are obtained with binning that is based on the support. Too many bins increase the RMSE. The best RMSE of 0.874741 on the pTest set was obtained with a bin size of 5 and is a slight improvement over non-binned linear regression.

Neural Network Blending

We investigated different numbers of neurons of a neural network with a single hidden layer and networks with two hidden layers. Table 5 summarizes the results. The RMSEs with one hidden layer are slightly better. For cross validation with retraining (prediction type "retraining") we obtain an RMSE of 0.873365 with 30 hidden neurons. The performance can be enhanced by bagging. This lowers the RMSE

to 0.873191 (generated by averaging 128 networks with 30 neurons in the hidden layer).

network setup	validation type	RMSE validation	train time	RMSE pTest
19-30-1	retraining 8-CV	0.873633	1.5[h]	0.873365
19-30-1	cross valid. mean 8-CV	0.873633	0.88[h]	0.873316
19-30-1	bagging size=32	0.87347	4.3[h]	0.873191
19-30-1	bagging size=128	0.873436	17.3[h]	0.873185
19-70-1	bagging size=128	0.87342	33.6[h]	0.873163
19-150-1	bagging size=128	0.873473	65.8[h]	0.873169
19-50-30-1	bagging size=128	0.873455	48.6[h]	0.87318

Table 5: Results from different neural network blends. We use in all networks the same learning rate $\eta = 5e{-}4$, $\eta^{(-)} = 5e{-}7$. Output transformation constants are $\alpha = 3.6$, $\beta = 3.0$.

We increased the number of neurons in the hidden layer in combination with bagging. The best results are obtained with 70 neurons, which leads to an RMSE of 0.873163. The advantage of neural network based blending is the excellent accuracy and the fast prediction time (a few ten seconds for the complete pTest set). The drawback of neural networks is the long training time.

Bagged Gradient Boosted Decision Tree

We analyzed Bagged Gradient Boosted Decision Trees with varying bagging size N_{bag}, varying subspace size S, varying number of leaves K, and varying learning rate η.

Our results suggest that smaller learning rates and larger bagging sizes improve the RMSE. The optimal subspace size depends on the data. A good value to start with is the square root of the number of features. Also the maximum number of leaves in a single tree is data dependent.

We found that the BGBDT blender deliver better results when the splits in building a single tree are chosen at random. This idea stems from P. Geurts "Extremely randomized trees" [8]. Thus, the threshold in a single tree node is chosen by selecting a random target value. For example a BGBDT with $N_{bag} = 128$, $K = 50$, $S = 20$ (full subspace), and $\eta = 0.1$ trained for 255 epochs results in a RMSE of 0.873842 on Test. The validation RMSE is 0.874103 and the training time is 7.3[h].

Kernel Ridge Regression Blending

Kernel ridge regression is not directly applicable to the training set of $N = 7 \cdot 10^5$ samples. The Gram matrix of size of NxN must be inverted. A PC with 16GB of main memory can store and invert matrices up to $N = 60000$ (single precision). We therefore use the following method: We are training many KRR models on random subsets and average their predictions. Figure 3 shows curves where subsets with 1% to 6% of the training set were used. The regularization constant and the width of the Gaussian kernel are optimized on a 4-fold cross-validation set for every single model. Not very surprisingly, the outcome shows that more data is better. All models benefit from averaging over multiple runs with different data. This approach can be seen as a form of bagging. An average of nine KRR models on 6% data achieves an RMSE of 0.8740 on the pTest set, which is significantly better than the linear regression baseline RMSE = 0.87525. The curves with 1% and 2% data show a saturation effect at about 100 averaged models, i.e. no improvement can be expected with an increase of the number of averaged models above 100.

fixed:	$\eta = 0.1$	$\eta = 0.05$	$\eta = 0.03$	$\eta = 0.02$
$N_{bag} = 32$	0.874783	0.87467	0.874624	0.874593
$K = 300$	0.87437	0.874352	0.87433	**0.874309**
$S = 2$	0.62[h]	1.21[h]	1.94[h]	3.27[h]
fixed:	$K = 500$	$K = 300$	$K = 200$	$K = 100$
$N_{bag} = 32$	0.874838	0.874783	0.874767	0.874934
$\eta = 0.1$	0.874427	**0.87437**	0.874399	0.874546
$S = 2$	0.48[h]	0.62[h]	0.73[h]	1.39[h]
fixed:	$N_{bag} = 16$	$N_{bag} = 32$	$N_{bag} = 64$	$N_{bag} = 128$
$\eta = 0.02$	0.874936	0.874593	0.874554	0.874517
$K = 300$	0.874381	0.874309	0.874293	**0.874288**
$S = 2$	1.1[h]	3.27[h]	7.01[h]	14.66[h]
fixed:	$S = 1$	$S = 2$	$S = 4$	$S = 8$
$\eta = 0.1$	0.874838	0.874784	0.87477	0.874841
$K = 500$	0.874427	**0.874377**	0.874405	0.874504
$N_{bag} = 32$	0.48[h]	0.42[h]	0.68[h]	1.11[h]

Table 6: BGBDT blending results. The first column denotes the fixed parameters. In the next columns the first line is the tested parameters, second line is the validation RMSE, third line is the pTest RMSE and fourth line is the training time. We vary the learn rate η, the subspace size K and the bagging size N_{bag}. For all results we use optimal splits in training a single tree.

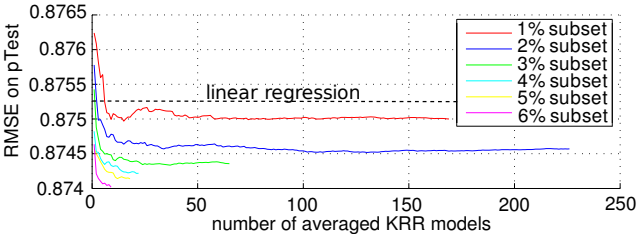

Figure 3: Kernel ridge regression applied to the blending of CF predictions. The KRR is trained on a random subset of the data (1%...6%). More data and more averaged models result in a lower RMSE.

KRR is worse than neural networks, but the results are promising. An increase of the training set size would lead to a more accurate model. But the huge computational requirements of KRR limits us to about 6% data. The train time for one KRR model on 6% subset (about 42000 samples) is 4 hours.

K-Nearest Neighbors Blending

The runtime of the k-nearest neighbors algorithm is quadratic in N (the number of training samples). Training and meta parameters tuning on all 700k data is too time consuming. We therefore tried the same approach as in the KRR blending model. We investigated the effect of averaging many models, where each of them is trained on a random subset of data. The results are shown in Figure 4. Again, more data and more averaged models are better. But the KNN shows very bad performance in terms of RMSE. The reported RMSEs are in the region of 0.885...0.884. The linear regression baseline achieves RMSE = 0.87525 on the pTest set.

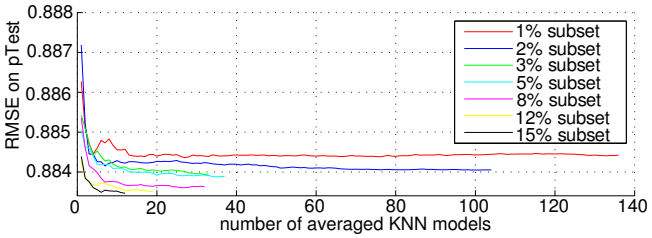

Figure 4: K-nearest neighbors applied to the blending of CF predictions. The model is trained on a random subset, we averaged the predictions of many random subsets. In each model, the neighborhood size K is optimized, typical values are $K = 200$.

One possible explanation for the bad performance is that predictions on new test samples are based on a weighted average of the features of the training set, which are themselves predictions. KNN is not able to deliver a successful blending model.

Bagging with Neural Networks, Polynomial Regression and GBDT

In this experiment we train a large ensemble in order to obtain extra accuracy. Bagging is applied on three different blending models: polynomial regression, gradient boosted decision trees, and neural networks. Table 7 shows the utilized algorithms with their parameters. Simple linear regression is used to combine them.

The training is performed sequentially on models listed in Table 7. As feedback for parameter selection we use linear regression of the out-of-bag estimates (oob) of all predecessors. When training the first model the linear regression of the oob estimate and a constant is used as prediction. Parameter selection in the second model is performed with the RMSE feedback of the linear regression of the oob estimates from model 1, model 2 and a constant. This is called the "blend RMSE" (numbers in brackets of the second column in Table 7. Hence we select parameters based on the linear combination of models.

Neural networks and gradient boosted decision trees were explained above. Polynomial regression is a linear regression with an extended feature space. The extension is done with the help of a polynomial of order n with out cross interactions between features, i.e., only single features x_i are raised to the powers of up to n.

The bagged ensemble reaches an RMSE of 0.87297 on the pTest set. This is a significant improvement over the linear regression baseline 0.87526.

model	RMSE (blend)	weight	parameters
const. 1	-	-0.014	-
NN	0.87345 (0.873445)	0.170	19-100-1, $\alpha = 3.6$, $\beta = 3.0$, $\eta = 5e{-}4$, $\eta^{(-)} = 5e{-}7$, 870 epochs, 44.4[h]
GBDT	0.874111 (0.873387)	0.054	$S = 20$, $K = 50$, $\eta = 0.1$, 226 epochs, randomSplitPoint, 6.6[h]
GBDT	0.874603 (0.873384)	0.098	$S = 2$, $K = 300$, $\eta = 0.02$, 267 epochs, optSplitPoint, 8.1[h]
PR	0.874358 (0.87336)	0.141	order=2, $\lambda = 2.4e{-}6$, with cross interactions, 1.9[h]
PR	0.895951 (0.873351)	-0.033	order=3, $\lambda = 0.054$, no cross interactions, 0.3[h]
NN	0.87345 (0.873296)	0.202	19-100-1, $\alpha = 2$, $\beta = 3.0$, $\eta = 5e{-}4$, $\eta^{(-)} = 5e{-}7$, 998 epochs, 47.1[h]
NN	0.873449 (0.873227)	0.371	19-50-30-1, $\alpha = 2$, $\beta = 3.0$, $\eta = 5e{-}4$, $\eta^{(-)} = 5e{-}7$, 952 epochs, 49.8[h]
blend	0.873227 pTest:0.87297		total train time: 158.2[h] total prediction time 4.5[h]

Table 7: Bagging and linear combination of many blending models applied to collaborative filtering for the Netflix Prize dataset. The first column indicates the model type. The second column reports the individual out-of-bag RMSE estimate, the number in brackets below is the blend RMSE. The third column shows the weight of the model. The fourth column shows metaparameters and the training time of each model. Accurate blending methods receive higher weights. $N_{bag} = 128$ in all models.

Results on the Netflix qualifying set

The submission to the Netflix Prize contest and the leaderboard feedback was based on the qualifying set. It consists of 2817131 ratings and has the same statistical properties as the probe set. The RMSE on the qualifying set with the linear regression model of Table 2 is 0.868088. This value is much better compared to our pTest set RMSE of 0.87525, because each collaborative filtering algorithm is retrained on the ratings from the test set and the whole probe set after the blending algorithm was trained [20]. For blending with a neural network (19-30-1), this results in an RMSE of 0.866345 (prediction time 3[s]) on the qualifying set, compare to Table 5. For bagging of many models (Table 7), an RMSE of 0.866004 can be achieved on the qualifying set (prediction time 17.8[h]). Further improvements can be expected when the blending models are trained on the whole probe set. Here, all trained blending models are based on data by a 50% random probe subset.

5. CONCLUSIONS

This paper demonstrates the advantage of ensemble learning applied to the combination (blending) of different collaborative filtering algorithms. As input we used 18 different predictors. We divided the Netflix Prize probe set randomly in two halves, a pTrain and a pTest set. The baseline is a regularized linear regression, which leads to an RMSE of 0.8752

on the pTest set. The best single blending algorithm is a 19-30-1 neural network with an RMSE of 0.873365 on pTest. We combined in an larger experiment neural networks, gradient boosted decision trees and polynomial regression with the help of bagging and optimized directly the linear regression of the out-of-bag estimates. This results in RMSE of 0.87297 on pTest, which improves linear regression by 0.0022. "The Ensemble" the second placed team of the Netflix prize competition reported a 0.0020 improvement over linear regression baseline [17]. When we applied the blenders to the predictions of the Netflix Prize qualifying set we found that Linear regression leads to 0.868088 RMSE and the bagged ensemble to 0.866004 RMSE (0.0021 improvement). To summarize, we show an RMSE roadmap in Figure 5. Above the RMSE scale one sees the best outcomes of the utilized blending algorithms. The best individual collaborative filtering results are indicated below the scale. The dataset and the source code of the learning framework are freely available under `http://elf-project.sourceforge.net` [9].

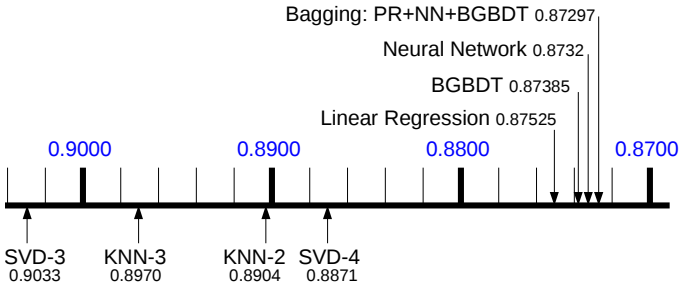

Figure 5: RMSE values on the Netflix Prize probe set. Shown are various blending methods (above the scale) and the best performing single algorithms (below the scale).

For practical applications we recommend to use a neural network in combination with bagging due to the fast prediction speed. A set of 10^6 samples can be predicted in a few seconds with this technique. We showed that a large ensemble of different collaborative filtering models leads to an accurate prediction system.

6. REFERENCES

[1] R. M. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *ICDM '07: Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, pages 43–52, Washington, DC, USA, 2007. IEEE Computer Society.

[2] J. Bennet and S. Lanning. The netflix prize. KDD Cup workshop, 2007. `"http://www.netflixprize.com"`.

[3] L. Breiman. Bagging predictors. In *Machine Learning*, pages 123–140, 1996.

[4] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[5] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *In Proceedings of the 21st International Conference on Machine Learning*, pages 137–144. ACM Press, 2004.

[6] D. A. Davis, N. V. Chawla, N. A. Christakis, and A.-L. Barabási. Time to CARE: a collaborative engine for practical disease prediction. Springer, November 2009.

[7] J. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 2002.

[8] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Mach. Learn.*, 63(1):3–42, 2006.

[9] M. Jahrer. ELF - Ensemble Learning Framework. An open source C++ framework for supervised learning. http://elf-project.sourceforge.net, 2010.

[10] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM, 2008.

[11] Y. Koren. The BellKor solution to the Netflix Grand Prize, 2009.

[12] Y. Koren. Collaborative filtering with temporal dynamics. In *KDD '09: Proceeding of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009.

[13] Y. Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. In *KDD: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009.

[14] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. *Proceedings of KDD Cup and Workshop*, 2007.

[15] M. Piotte and M. Chabbert. The Pragmatic theory solution to the Netflix Grand Prize, 2009.

[16] R. Salakhutdinov, A. Mnih, and G. E. Hinton. Restricted boltzmann machines for collaborative filtering. In *ICML*, pages 791–798, 2007.

[17] J. Sill, G. Takacs, L. Mackey, and D. Lin. Feature-weighted linear stacking. *arXiv:0911.0460v2*, 2009.

[18] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Matrix factorization and neighbor based algorithms for the netflix prize problem. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*, pages 267–274. ACM, 2008.

[19] A. Töscher and M. Jahrer. The BigChaos solution to the Netflix Prize 2008. Technical report, commendo research & consulting, October 2008.

[20] A. Töscher, M. Jahrer, and R. M. Bell. The BigChaos solution to the Netflix Grand Prize, 2009.

[21] A. Töscher, M. Jahrer, and R. Legenstein. Improved neighborhood-based algorithms for large-scale recommender systems. In *KDD Workshop at SIGKDD 08*, August 2008.