Combining Predictions for an accurate Recommender System

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

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

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

ABSTRACT

The application of ensemble learning to recommender systems is analyzed with the Netflix Prize dataset. We found that simple linear combination of predictions is not optimal in the sense of minimize the prediction RMSE. To predict ratings with collaborative filtering we use a set of predictions from different models (SVD, KNN, Restricted Boltzmann machine, Asymmetric Factor model, Global Effects). These models are state of the art in the field of collaborative filtering. We show that a large ensemble of blenders outperforms the neural network as best single blending algorithm. Dataset and learning software is avaliable online [9].

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: [Data mining, Recommender Systems, Ensemble Learning, Collaborative Filtering, Netflix Competition]

General Terms

Ensemble Learning, Recommender Systems

Keywords

Recommender Systems, Netflix, Supervised Learning, Ensemble Learning

1. INTRODUCTION

A recommender system helps users to navigate through portals or web shops with a lot of content by aggregating data generated by other users. For example, `amazon.com` provides each user with a personalized shop page on login ("your personal shop"), based on the user's past purchase data. A user is a unique person, which generates events, like purchases, ratings, bookmarks or clicks. An item is an individual product or a unique piece in an online portal (e.g. movies, jeans, watches, sunglasses). User-item predictions are personalized item scores for a user. Item-item correlations are unpersonalized relationships between items (not user dependent). A typical recommendation system consists of two parts, the first part is the prediction model, which is responsible for delivering accurate user-item predictions and item-item correlations. The prediction model should exhibit good scaling to large number of users and items. The second part is the recommendation module, this module is responsible for presenting a set of items to the user. A typical way of doing this is to predict for a given user all available items and sort their scores in descending order and take the top-K products as personalized recommendation (top-K recommendation).

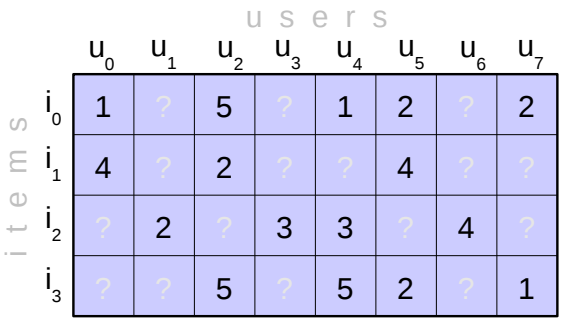

Figure 1: The sparse user x item matrix in an recommendation system. The data sketch here is from a 5-star rating system. The target for the prediction model is to accurately predict the missing values in the matrix.

Consider the case of having a 5-star (from one the five starts, one star means bad and 5 stars means excellent) rating system on a web portal. The problem of prediction can be seen as a sparse user-item matrix (Figure 1). The goal is to make accurate prediction for the missing values. In collaborative filtering, the system infers a model from all available data. For example a low-rank matrix factorization leads to an accurate prediction model. Another approach is to use content filtering for prediction, which uses meta information

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
KDD2010 Washington DC, USA
Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...$10.00.

on user and item side, for example product sub-categories or geographic user information.

Collaborative filtering is based on data analysis. Data can be any source of user-generated information, such as purchases, ratings, clicks, bookmarks, favorite-adds, whishlist-adds, etc. Users and items become anonymous numbers in the collaborative filtering system. A user is fully described by the set of events (what the user does). The Netflix Prize [1] is a benchmark for collaborative filtering algorithms. It is the largest public available dataset, which contains about 10^8 ratings, collected in a time period of 7 years. Before the Netflix Prize started in October 2006, the MovieLens dataset with about 10^6 ratings was the largest dataset for applying collaborative filtering algorithms on real world data [1].

Content filtering relies on additional data from users and items. In general, users complete a form when they create a login to the site, so we have the name, address, zip code, age, gender, preferences, etc. An item can be for example a music song, a t-shirt or a DVD. The same on item side, a shop or a portal keeps additional information of the items, such as main category, subcategory, price category, color, size, language(movies), etc. A common technique is to analyze this information by find similar users, who bought items from the same subcategory. Another approach is to use simple string matching to find products that the user like.

Research in the field of collaborative filtering has become popular since the 1M $ Netflix Prize was announced in 2006. Before that, the work of researchers in this field was focused on neighborhood models, such as item-item or user-user KNN algorithms. Herlocker et. al. give a good overview of neighborhood based approaches in [6].

The error function is crucial in measuring the accuracy of collaborative filtering algorithms. The Netflix Prize uses RMSE, which is a perfect measure in a competition due to the smooth behavior. Other error measures such as average rank or hitrate are more appropriate in evaluating the quality in a top-K recommender system.

$$RMSE = \sqrt{\frac{1}{|\mathcal{L}|} \sum_{(u,i)\in\mathcal{L}} (\widehat{r_{ui}} - r_{ui})^2} \quad (1)$$

The list of ratings is denoted by $\mathcal{L} = \langle (u_1, i_1), ..., (u_L, i_L) \rangle$, predicted ratings are $\widehat{r_{ui}}$ and the real ratings are r_{ui}. Other error measures such as mean absolute error or ranking based errors are discussed by Herlocker [7].

The prediction of a collaborative filtering model should lie between a lower and an upper bound. In the case of 5-star ratings, the bounds are 1 and 5. One can show that predictions of a SVD model are not bounded, they can have values greater or lower than the bounds. Clipping at the bound values is not correct due to the loss of ranking possibility. When a recommendation system does top-K recommendations, it is common to denote the predictions as "scores". Scores can have any scale or offset, since the ranking of the scores stays the same.

In a web shop, there is often a lack of a rating system. The shop operator has a large database of the customer's consumption history. In other words the webmaster has access to data of who bought what. This is known as binary view of the user-item rating matrix and it is similar to a "I like it" rating system. Purchase information has high-quality because the customer actually spends money on the item. The matrix cells, which have value of 0 are potential candidates for recommended items. Consider the case of a full-filled matrix in the Netflix Prize dataset, the number of values in the matrix is about $500k \cdot 18k \approx 9 \cdot 10^9$ (9 billion numbers). The training time would be prohibitive for a SVD algorithm based on gradient descent. Training on a full-filled matrix would result in $O(U \cdot M)$ training time (standard is $O(|\mathcal{L}|)$, where $|\mathcal{L}|$ is the number of data points). U is the number of users and M is the number of items in the system. We found a simple way to do gradient descent with a SVD model on purchase data. The idea is to use a random subset of the zero values rather than all. Stochastic gradient descent runs for example 30 epochs through the training list $\mathcal{L}$ while apply updates to the user and item features. This training can be done effectively user-wise. Now we draw per user a new random subset of items with target rating 0 addidionally to the 1's. We found that 1% of $|M|$ works well for 0-sampling. Hu and Koren in [8] has done analysis on implicit feedback datasets, for example with data from IPTV [2] where users watch films with their set-top boxes. They propose an alternating least squared matrix factorization algorithm, that overcome the problem of handling the large filled user-item matrix by considering only the 1's in the data.

For the evaluation we use the Netflix Prize dataset. Training set size is 100M, Netflix held back a qualifying set of 2.8M ratings, which should be predicted by the participants of the contest. RMSE feedback was calculated on a 50% random subset - called the quiz set. One of the largest challenges is to handle the huge size of the data, hence we need algorithms with good asymptotic runtime, both in the training and in the prediction phase. For example user-user neighborhood approaches have horrible runtime bounds. Furthermore they have $O(U^2)$ memory consumption, where U is the total number of users. For the Netflix data set, we have 500k users, which leads to approx. 500GB of memory consumption when we store the upper triangle matrix of all user-user

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

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

[1] http://www.grouplens.org/node/73, in 2008 groupslens published a 10M dataset (10^7 ratings)

[2] IPTV means Internet Protocol television

nr	name	RMSE	description
1	AFM-1	0.9362	AFM, 200 features, $\eta = 1e-3$, $\lambda = 1e-3$, η multiplied with 0.95 from epoch 30, 120 epochs
2	AFM-2	0.9231	AFM, 2000 features, $\eta = 1e-3$, $\lambda = 2e-3$, 23 epochs, based on residuals from KNN-5.
3	AFM-3	0.9340	AFM, 40 features, $\eta = 1e-4$, $\lambda = 1e-3$, 96 epochs
4	AFM-4	0.9391	AFM, 900 features, $\eta = 1e-3$, $\lambda = 1e-2$, 43 epochs
5	GE-1	0.9079	GE, based on residuals KNN-1
6	GE-2	0.9710	GE, on raw ratings.
7	GE-3	0.9443	GE, based on residuals KNN-5.
8	GE-4	0.9209	GE(time), on residuals AFM-2.
9	KNN-1	0.9110	KNN item, Pearson correlations, $k = 24$ neighbors, based on residuals AFM-1.
10	KNN-2	0.8904	KNN item, set correlation [16], $k = 122$, based on residuals from the chain RBM-KNN-GE(with time).
11	KNN-3	0.8970	KNN item, Pearson corr., $k = 55$, based on residuals of a RBM with $nHid = 150$
12	KNN-4	0.9463	KNN item, Pearson corr., $k = 21$, based on residuals 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, 158 epochs, $\eta = 8e-4$, $\lambda = 0.01$, item centered.
16	SVD-2	0.9172	SVD, 20 features, 158 epochs, $\eta = 0.002$, $\lambda = 0.02$, item centered.
17	SVD-3	0.9033	SVD, 1000 features, AUF [16]. 158 epochs, $\eta = 0.001$, $\lambda = 0.015$.
18	SVD-4	0.8871	SVDe, 150 features. η's and λ's auto tuned on the probe set [16].
19	support	-	Number of ratings per user

Table 2: These predictors are the input for a blender. The table summarizes major results from the Netflix Prize competition. The listed RMSE is on the probe set.

correlations in floating point accuracy. In Table 1 we list the most useful collaborative filtering algorithms from the Netflix Prize challenge. The RMSE column is the error 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, for example a linear combination of all single models would lead to a 0.87 RMSE score. Netflix own prediction algorithm, called "Cinematch" has an RMSE of about 0.95 at the time when the competition starts in 2006 [1]. More sophisticated blending techniques lowers the RMSE below 0.87. In the following section we use the word "blending" as an acronym for combining a set of predictions.

The following list gives a technical overview of collaborative filtering algorithm listed in Table 1. We focus on the overview of each individual, not going deep into the training process. For further detailed explanation, we recommend to read the Netflix Prize Winner Reports [16], [10], [12].

KNN item-item

A prediction $\widehat{r_{ui}}$ in an item based k-nearest neighborhood model is made by calculating a weighted sum over k-nearest items, the weights are proportional to the correlations c_{ij}. Therefore a precalculated item-item correlation matrix $\mathbf{C}$ is useful, due to the need of constant access time of any item-item correlation c_{ij}. This results in a memory consumption of $O(M^2)$ (M is the number of items). Training time is building the item-item correlation matrix $\mathbf{C}$, which is limited to $O(U \cdot M^2)$ operations. For one prediction, the KNN selects k-best correlations to the item i, this can take up to $O(U)$ operations. Sort the list takes $O(U \cdot log(U))$, which is the upper bound in 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 training bounds are also flipped. See Table 1 for the complete list. For the Netflix Prize dataset this method is unpractical due to the huge memory consumption. Here, 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 dimensionality of 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 [17].

SVD (matrix factorization)

This is probably the most popular collaborative filtering technique. A prediction is given by the dot product of a user feature $\mathbf{p}_u$ and an item feature $\mathbf{q}_i$: $\widehat{r_{ui}} = \mathbf{p}_u^T \mathbf{q}_i$, hence $O(1)$ 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. In practice this means the whole training needs a few ten epochs over the whole dataset until convergence - therefore $O(|\mathcal{L}|)$ training time. The model parameterizes two matrices with f rows, this leads to $O(M+U)$ memory consumption. 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, this makes the SVD an excellent candidate for large scale recommendation applications. As an extention to the excellent prediction capability of SVD models, item-item and user-user correlations can be calculated efficiently with a normalized euclidean distance measure between user or item feature vectors [15].

AFM (asymmetric factor model)

In the plain SVD model, a user is represented by the feature $\mathbf{q}_u$. The AFM model represents a user by the items he has rated, this means that 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 $\mathbf{y}_u = |N(u)|^{-1/2} \sum_{i \in N(u)} \mathbf{p}_i$. The set of items, which was rated by the user u is $N(u)$. $\mathbf{p}_i$ 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 [11]. The prediction time is constant (like SVD), because we can store the precalculated virtual user features after training, $\widehat{r_{ui}} = \mathbf{y}_u^T \mathbf{q}_i$. 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. Further explanation can be found in [16].

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 as frequency the number of votes a user gives on a particular day t. Integration of this information is not straight forward, because for each additional feature the learn rate and regularization parameters has to be set correctly by optimizing them on a validation set. Training time rises by a constant, therefore 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 [16]), 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.

RBM (Restricted Boltzmann Machine)

In general, a Boltzmann machine is a stochastic generative model. The restricted Boltzmann machine [13] is a neural network with one input layer and one hidden layer. Neither the visible nor the hidden units have connections to itself, which means that the net has no recurrences. For collaborative filtering, the number of visible units are the number of items in the system. The number of hidden units represents the number of features, each user is mapped to a low-dimensional representation in the hidden layer. Learning works well with contrastive divergence learning [13] 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 [16] 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" dependent features, which is equivalent to a SVD with either fixed item or fixed user features. 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. A detailed explanation can be found in the Netflix Prize Winner Report [16].

Combinations

A popular way of combining algorithms is residual training. This means the raw ratings are subtracted by the prediction from a model when they are used. We found that item-item KNNs are most effective, when they are applied on residuals of RBMs. When constructing such a residual chain, the ensemble of collaborative filtering (CF) results becomes more diverse, which is good for the final blend. The final blender has access to all the predictors generated by various CF models on various residuals of other models.

2. BLENDING

Combining different kinds of CF algorithms is the motivation of the next chapters (acronym "blending"). We use 18 predictors and the logarithm of the support (Table 2) from 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 not a random subset of the training data, it reflects the optimization goal by taking per user the latest ratings. Furthermore every user was sampled with equal probability. The probeset was divided by us in two random halfs, the train and the test set. We begin 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 [2]. Additionally, the computational costly k-nearest neighbors and the kernel ridge regression algorithm are applied to blend predictions by averaging multiple models trained on small random subsets. 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 to correct one we use eiter cross validation or bagging [2]. With cross validation we train k copies of the model and use the RMSE on the whole training set (merged leave out sets) as feedback for parameter selection. Prediction of new samples can be done 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"). 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.

Notation

Blending predictions is a supervised machine learning problem. The features are $\mathbf{X}$, a NxF matrix of predictions, where N is the number of samples and F the number of predictors. Target values are $\mathbf{Y}$, a Nx1 vector, in our case: Integer ratings from 1 to 5. The blending algorithm is formally a function $\Omega(\mathbf{x}) : \mathbb{R}^F \mapsto \mathbb{R}$. 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.

$$RMSE = \frac{1}{N} \sum_{i=1}^{N} \left(\Omega(\mathbf{x}_i) - y_i\right)^2 \tag{2}$$

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.

$$\mathbf{X} \cdot \mathbf{w} = \mathbf{Y} \tag{3}$$

For any input vector $\mathbf{x}$, the prediction is $\Omega(\mathbf{x}) = \mathbf{x}^T\mathbf{w}$. Weights $\mathbf{x}$ are calculated with ridge regression, $\mathbf{x} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}$. Cross-validation is used in order to select proper ridge regression constant λ.

Histogram based Linear Regression

Due to the huge size of the training set one can divide the set into disjoint B bins and calculate separate blending weights per bin. With linear regression, the blending weights $\mathbf{w}$ become $\mathbf{w}_b$, where b is the corresponding bin to the predicted rating $\widehat{r_{ui}}$. Each bin should approximately have the same number of ratings. The training set can be split by using a histogram on one of the following criteria.

- Support : The number of votes from a user. The blender can now overweight particular predictions dependent on how many rating the user gave. RBMs are prone to receive high weight when the user has only a few votes in the data. SVDs are overweighted when many information from a user is available.
- Time : The day, when 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 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. The explanation in different number of votes on one day can be that all people in a household using the same account.

A prediction is given by

$$\widehat{r_{ui}} = \mathbf{x}^T\mathbf{w}_b \tag{4}$$

This means we calculate B separate blending weights $\mathbf{w}_b$. For example, when we divide the Netflix probe-set into 5 support-bins (approximately equal sized bins), following formula gives the bin b. $|N(u)|$ denotes the number of ratings of a user u.

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

Neural Network - NN

A neural network is a function approximator. Small nets can be trained efficiently on huge data sets, therefore it is very suitable for a nonlinear blending algorithm. The training of neural nets is performed by stochastic gradient descent. The output neuron has again a sigmoid activation function with an output swing of $-1... + 1$, to generate rating predictions in the range of 1...5 we use a simple output transformation. For example the output is multiplied by 3.6 and the constant 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. No weight decay or batch update is used.

Bagged Gradient Boosted Decision Tree - BGBDT

Decision trees are known as a good supervised learning tool. The main drawback of a single decision tree is the moderate accuracy. Breiman [2] shows how bagging can be applied to improve the accuracy of decision trees. The discretized output function of a tree limits the ability of modelling a smooth function. A tree is a rule-based learner, a simple linear combination of inputs is not possible in a decision tree. The number of possible output values of a tree corresponds to the number of leafs. For regression problems, such as blending predictions, this is a big disadvantage. Blending predictions should result in a smooth function. In 1999, Jerome Friedman introduced an interesting idea to improve the accuracy of a learning machine. He called his technique "Stochastic Gradient Boosting" [4], [5]. The core idea is to train multiple learner in a chain. Each model learns only a fraction of the desired function Ω, controlled by the learn rate η. Stochastic gradient boosting has a distinct similarity to train on residuals.

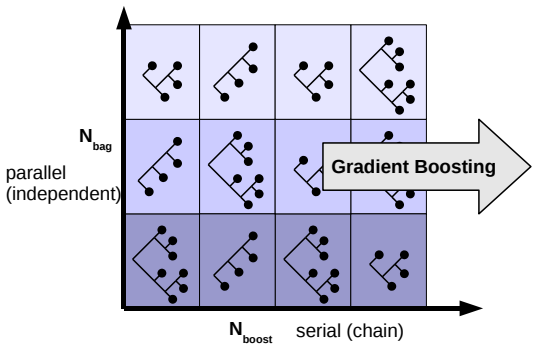

Figure 2: Bagged Gradient Boosted Decision Tree. A prediction from the BGBDT consists of results from $N_{bag} \cdot N_{boost}$ single decision trees. Both, Bagging and Boosting improves the accuracy.

A single tree is trained recursively by splitting always the largest node, up to K leafs are in a single tree. 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 [3]). The subspace size K is the number of features considered in each node to find the optimal split. The learning rate η is the learned fraction per tree in each cascade layer. Random splits are a random target value taken as treshold when building the tree. Finally, we end up with a tree blending technique that combines the benefits from Bagging, Gradient Boosting and Random Subspace selection. The algorithm is called now BGBDT - Bagged Gradient Boosted Decision Tree (see Figure 2). The prediction of a BGBDT is much smoother than from a single tree. We use the bagging size N_{bag} and the number N_{boost} of boosting steps in the chain.

Kernel Ridge Regression Blending - KRR

The learning algorithm is described in [16], Sec. 5.14.1. There is only one step required to train the model.

$$\mathbf{W} = (k^{dot}(\mathbf{X}^T, \mathbf{X}) + \lambda \mathbf{I})^{-1}\mathbf{Y} \quad (6)$$

$\mathbf{X}$ is the training set, $\mathbf{Y}$ are the training targets. The weights $\mathbf{W}$ are needed for prediction. The notation $k^{dot}(\cdot,\cdot)$ denotes the kernelized dot product of two matrices, this means that the point-wise dot product is calculated with a kernel function. The Gauss kernel $k(\mathbf{x},\mathbf{y}) = exp(-(||\mathbf{x}-\mathbf{y}||^2)/\sigma^2)$ works best on our experiments. A single prediction is given by

$$\Omega(\mathbf{x}) = k^{dot}(\mathbf{x}, \mathbf{X}^T)\mathbf{W} \quad (7)$$

KRR has a training time complexity of $O(N^3)$ (invert the Gram matrix) and space requirements of $O(N^2)$, hence it is 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 with respect to the accuracy measured in RMSE on the test set. We tune the kernel width σ and the regularization constant λ with cross validation.

K-Nearest Neighbors Blending - KNN

The template-based KNN algorithm is very slow in predicting new samples when the training set is large. All distances are calculated on the fly, training and prediction time rise with $O(N^2)$. The model is given by the following formula.

$$\Omega(\mathbf{x}) = \frac{\sum_{k\in D} \mathbf{y}_k \cdot d(\mathbf{x},\mathbf{x}_k)}{\sum_{k\in D} |d(\mathbf{x},\mathbf{x}_k)|} \quad (8)$$

The set D 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 is adjusted by cross-validation. We use again a small subset of the train set to build the model. Averaging over many random subsets delivers our final prediction. Again, we investigate the impact of the size of the subset. We found out that KNN has very bad accuracy compared to all other blending techniques. One possible explanation is that predictions on new test samples are based on a weighted average of the features from the training set, which are themselves predictions. KNN is not able to deliver a successful blending model.

3. RESULTS

In the first step of training the collaborative filtering algorithms, we remove the probe set from the dataset. The probe is a hold out set and is a very reliable performance measure during training, because of its huge size of 1408395 ratings.

We decide to evaluate blending methods based on a 50% random subset probe. The probe set is splited into two halves, one half for training and the other half for testing. All reported RMSE values in the next Figures and Tables are evaluated on this test set. It is never touched during training the blending models.

The previously described algorithms are now trained on 704197 and tested on 704198 samples. The Netflix Prize contest awards the winner for 10% improvement in RMSE. The baseline was set by Netflix's internal collaborative filtering system "Cinematch" with a RMSE of 0.9525 on the Netflix quiz set. The 10% improvement is equivalent to an RMSE of 0.8563. The leaderboard shows the submission feedback with an accuracy of 4 digits after the comma. Which means that a RMSE of 0.8600 or 0.8599 makes a difference. At the end of the Netflix Prize contest, the winning team has the better score on a hidden test set. The second placed team "The Ensemble" shows in his blog [3] that the rounded RMSE score of both teams were the same, the first-submitter rule determines the winner (both teams had a test set RMSE of 0.8567, which corresponds to 10.06% improvement).

Our reported RMSE values have at least 4 digit accuracy, a significance improved blending technique lowers the RMSE score at 0.0001. Linear regression with RMSE = 0.87525 is used as baseline. A marginal improvement results in a 0.0001...0.0002 lower RMSE. For example the KNN algorithm has significantly inferior performance (RMSE 0.884), this shows that KNN is the wrong supervised learning technique for blending CF predictions.

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 have a baseline estimate of the RMSE on the test 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.

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

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

The Table 3 shows the weights, from each of the predictors (from Table 2) including the support and the constant input. The largest weight has the constant 1 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 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.

Binned linear regression

This is linear regression on predefined subsets of the training data. For each of the training and test samples we have the support (number of ratings), the date (day of the rating) and

[3] `http://www.the-ensemble.com/`

the frequency (number of ratings per day). 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 test set. The small values in the brackets are RMSEs from the cross validation.

We get best results with the support binning. Too many bins increase the RMSE. The best results are obtained by 5 bins.

Neural Network Blending

We investigate different number of neurons in the hidden layer and two hidden layers. Table 5 shows the outcome. We tried one and two hidden layer, the RMSEs with one hidden layer are slightly better. For one hidden layer we get the best results with 100 neurons (RMSE = $0.87xxx$). The performance can be enhanced by apply bagging, this lowers the RMSE to $0.87xxx$ (generated by a 1-layer net with 100 neurons).

net setup	validation type	RMSE validation	train time	RMSE test
19-30-1	retraining 8-CV	0.87363	xh	0.873361
19-30-1	cross valid. mean 8-CV	0.87363	xh	0.873313
19-30-1	bagging size=32	0.87347	xh	0.873191
19-30-1	bagging size=128		xh	
19-50-1	bagging size=128		xh	
19-100-1	bagging size=128		xh	
19-200-1	bagging size=128		xh	
19-50-30-1	bagging size=128		xh	

Table 5: Results from different neural network blends. We use in all nets the same $\eta = 5e-4$, $\eta^{(-)} = 5e-7$.

Bagged Gradient Boosted Decision Tree

The analysis of the Bagged Gradient Boosted Decision Tree is done by varying the bagging size N_{bag}, the subspace size S, the number of leafs K and the learning rate η in the gradient boosting approach. Additionally we found that random splits further improve the accuracy. Bagging size is the number of model copies trained simultaneously.

Our results suggest that smaller learning rates and larger bagging sizes improve the RMSE. The optimal subspace size dependens on the data, a good value to start with is the square root of the number of features.

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**
$S = 2$	2249[s]	4362[s]	6978[s]	11788[s]
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
$S = 2$	1741[s]	2249[s]	2620[s]	5009[s]
fixed:	$N_{bag} = 16$	$N_{bag} = 32$	$N_{bag} = 64$	$N_{bag} = 128$
$\eta = 0.02$	0.87xxx	0.874783	0.87xxx	0.87xxx
$K = 300$	0.87xxx	0.87437	0.87xxx	0.87xxx
$S = 2$	x[s]	2249[s]	x[s]	x[s]
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
$N_{bag} = 32$	1741[s]	1525[s]	2432[s]	4014[s]

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 test RMSE and fourth line is the training time. We vary the learn rate η, the subspace size K and the baggig size N_{bag}. For all results we use optimal splits in training a single tree.

We found that the BGBDT blender deliver better results when the splits in building a single tree are choosen at random. For example a BGBDT with $K = 20$ (full subspace), $S = 50$, $\eta = 0.1$ and trained 255 epochs results in a test RMSE of 0.873842, validation RMSE is 0.874103 and a training time of 26235[s].

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 with 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: Training many KRR models on random subsets and average their predictions, the 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 test 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.

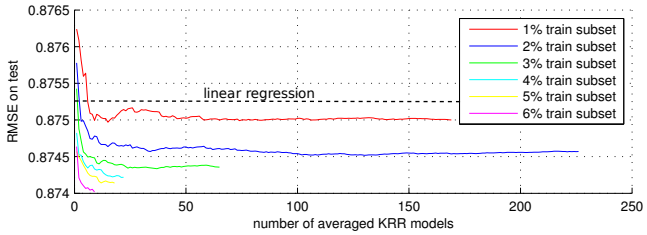

Figure 3: Kernel ridge regression applied to the blending of CF predictions. The KRR is trained on a random subset of the data (1%...6%). More data and more averaged models result in a lower RMSE.

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 try the same approach as in the KRR blending model. We investigate the effect of averaging many models, where each of them is trained on a random subset from 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 test set.

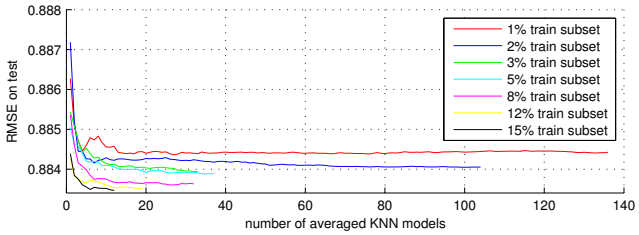

Figure 4: K-nearest neighbors applied to the blending of CF predictions. The model is trained on a random subset, we averaged the predictions from many random subsets. In each model, the neighborhood size k is optimized, typical values are $k = 200$.

Bagging with Neural Networks, Polynomial Regression and GBDT

In this experiment we use bagging as model validation. A chain of models is trained by optimizing the linear regression of the out-of-bag estimates. This means that the ensemble of blenders is build in a greedy way, starting with the first model. The first model optimizes the linear regression of itself and a constant, the second model the linear regression of itself, the first model and a constant. The order of each single model is the same as listed in Table 7. The models in the ensemble are neural networks, gradient boosted decision trees and polynomial regression. Polynomial regression is a linear regression with an extended feature space. The extention is done with the help of a polynomial series $(\mathbf{x}+1)^n$. We mean by "no cross interactions" that the single features x_i^n are rised to the power of n. Where n is the polynomial order.

model	RMSE (blend)	weight	parameters
const. 1	-	0.xxx	-
NN	0.87xxx (0.87xxx)	0.xxx	19-100-1, $of = 3.0$, $sc = 3.6$, $\eta = 5e-4$, $\eta^{(-)} = 5e-7$, 870 epochs
BGBDT	0.87xxx (0.87xxx)	0.xxx	$S = 20$, $K = 50$, $\eta = 0.1$, xxx epochs, randomSplitPoint
BGBDT	0.87xxx (0.87xxx)	0.xxx	$S = 2$, $K = 300$, $\eta = 0.02$, xxx epochs, optSplitPoint
PR	0.87xxx (0.87xxx)	0.xxx	order=2, $\lambda = 0.xxx$, with cross interactions
PR	0.89xxx (0.87xxx)	0.xxx	order=3, $\lambda = 0.xxx$, no cross interactions
NN	0.87xxx (0.87xxx)	0.xxx	19-100-1, $of = 3.0$, $sc = 2$, $\eta = 5e-4$, $\eta^{(-)} = 5e-7$, xxx epochs
NN	0.87xxx (0.87xxx)	0.xxx	19-50-30-1, $of = 3.0$, $sc = 2$, $\eta = 5e-4$, $\eta^{(-)} = 5e-7$, xxx epochs
blend	0.87xxx test:0.87xxx		total train time: xx[h] total prediction time xx[h]

Table 7: Bagging and linear combination of many models applied to blend collaborative filtering predictions from the Netflix Prize dataset. The first column is 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. Accurate blending methods receive higher weights.

Results on the Netflix qualifying set

The RMSE on the Netflix qualifying predictions with the linear regression model from Table 2 is 0.868088. This value is much better compared to our test set (0.87525) because each collaborative filtering algorithm includes the ratings from the probe set when they get trained [16]. Polynomial Regression: 0.866922. Neural Network 19-50-1: 0.866378. Bagging many models from 7: 0.86xxxx.

4. CONCLUSION

This paper shows the advantage of ensemble learning applied to the combination (blending) of different collaborative filtering algorithms. As input we use 18 different predictors. We divide the Netflix Prize probe set randomly in two halfs, a train and a test set. The baseline is a regularized linear regression, which leads to an RMSE of 0.8752 on the test set. Best single blending algorithm is a 19-100-1 neural network with an test RMSE of 0.8733. We combine in an larger experiment neural networks, gradient boosted decision trees and polynomial regression with the help of bagging and optimizing directly the linear regression of the out-of-bag estimates. This result in an test RMSE of 0.8730, which is about 0.0022 improvement to linear regression. The second placed team "The Ensemble" published a paper [14], they reported 0.0020 improvement over linear regression baseline. When we apply the blenders to the predictions of the Netflix Prize qualifying set: Linear regression leads to 0.8xxx RMSE and the bagged ensemble 0.8xxx RMSE (0.00xx improvement). As a summary we show a RMSE roadmap at Figure 5. Above the RMSE scale we place best outcomes of blending algorithms, below there are the best individual collaborative filtering results. The dataset and the source

code of the learning framework are freely avaliable under `www.commendo.at/elf-project`.

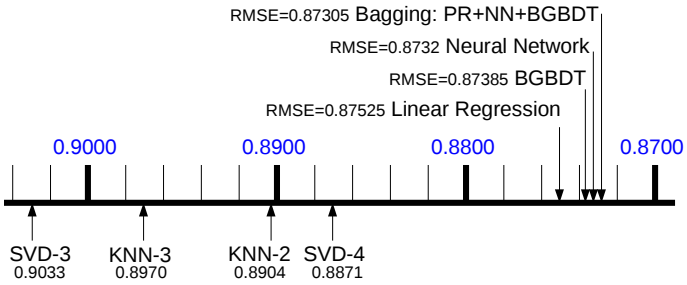

Figure 5: RMSE values on the Netflix Prize probe set. Above there are various blending methods, below the best performing single algorithms (from 19 total).

We can show that a large ensemble of different collaborative filtering models leads to an accurate prediction system. Furthermore a large ensemble of supervised learning techniques can improve the accuracy of the whole ensemble by clever blending.

5. REFERENCES

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

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

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

[4] J. Friedman. Greedy function approximation: A gradient boosting machine. Technical report, Salford Systems, 1999.

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

[6] L. Herlocker, Jon, A. Konstan, A. Joseph, and J. Riedl. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Inf. Retr.*, 5(4):287–310, 2002.

[7] L. Herlocker, A. Konstan, G. L. Terveen, John, and T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22:5–53, 2004.

[8] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *IEEE International Conference on Data Mining ICDM*, 2008.

[9] M. Jahrer. ELF - Ensemble Learning Framework. an open source C++ framework for supervised learning. http://www.commendo.at/elf-project, 2010.

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

[11] 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.

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

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

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

[15] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Investigation of various matrix factorization methods for large recommender systems. In *Proc. of the 2nd KDD Workshop on Large Scale Recommender Systems and the Netflix Prize Competition*, August 2008.

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

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