Unifying Collaborative and Content-Based Filtering

Justin Basilico BASILICO@CS.BROWN.EDU
Department of Computer Science, Brown University, Providence, RI 02912 USA

Thomas Hofmann TH@CS.BROWN.EDU
Department of Computer Science, Brown University, Providence, RI 02912 USA
Max Planck Institute for Biological Cybernetics, Tübingen, Germany

Abstract

Collaborative and content-based filtering are two paradigms that have been applied in the context of recommender systems and user preference prediction. This paper proposes a novel, unified approach that systematically integrates all available training information such as past user-item ratings as well as attributes of items or users to learn a prediction function. The key ingredient of our method is the design of a suitable kernel or similarity function between user-item pairs that allows simultaneous generalization across the user and item dimensions. We propose an on-line algorithm (JRank) that generalizes perceptron learning. Experimental results on the EachMovie data set show significant improvements over standard approaches.

1. Introduction

Predicting ratings and preferences for users interacting with a computer system is a key challenge in application areas such as electronic commerce, information filtering, and user interface design. The standard learning paradigm employed in this context is widely known as *collaborative* or *social filtering* (Goldberg et al., 1992; Resnick et al., 1994; Shardanand & Maes, 1995). Collaborative filtering exploits correlations between ratings across a population of users, in its most popular incarnation by first finding users most similar to some active user and by then forming a weighted vote over these neighbors to predict unobserved ratings. *Content-based filtering* is an alternative paradigm that has been used mainly in the context of recommending items such as books, web pages, news, etc. for which informative content descriptors exist. Standard machine learning methods like naive Bayes classification (Lang, 1995; Pazzani et al., 1996; Mooney & Roy, 2000) have been used in this context.

In this paper, we pursue the philosophy that collaborative and content-based filtering are complementary views that should be unified in a common learning architecture. In fact, it should also be possible to take into account other information such as demographic user data. To that extent, we propose a technique that shares many desirable features with kernel-based methods (Schölkopf & Smola, 2001) and perceptron learning, more precisely the PRank algorithm of (Crammer & Singer, 2002). Formally, we follow a modeling approach first suggested in (Basu et al., 1998) which is to learn a mapping from user-item pairs to a set of ratings. A similar approach was suggested in (Billsus & Pazzani, 1998) and utilized in many recently proposed machine learning algorithms.

The most common way of casting the present prediction problem as a standard classification problem – or more generally as an ordinal regression problem – is to treat every user u as an independent, separate classification problem. Hence, every item x for which u has provided a rating $r(u, x)$ is considered as a training instance with the rating as its target value. In order to represent items x as feature vectors, one may utilize item attributes or encode known ratings provided by other users $u' \neq u$ as features (Billsus & Pazzani, 1998). Predictions for a specific user are then made by applying the learned classification rule to items with unknown ratings.

Yet, one can also take the opposite view and treat every item as a separate classification problem for which a user constitutes an instance. In the latter case, a user needs to be represented by some feature vector, for example, by encoding his/her ratings on other items or

Appearing in *Proceedings of the 21st International Conference on Machine Learning*, Banff, Canada, 2004.

by utilizing demographic attributes. Interchanging the role of items and users is an alternative approach that is known as item-based collaborative filtering (Sarwar et al., 2001).

We suggest to avoid this polarity of using a feature representation either over items or over users, by allowing features to be extracted jointly from user-item pairs (u, x). The crucial ingredient is a joint feature map Ψ, $(u, x) \mapsto \Psi(u, x) \in \Re^D$. Some features may only depend on the item or only on the user, but they may also combine aspects of both. For example, features may indicate that a pair (u, x) deals with a particular (type of) user *and* a particular class of items (say, of a particular genre or category). One may also think of the inner product $\langle \Psi(u, x), \Psi(u', x') \rangle$ as a similarity measure that governs how generalization occurs over user-item pairs. Special cases on how to compute such similarities are, for example, to require that $u = u'$ and to use some similarity measure between items x and x', or to require that $x = x'$ and to use some similarity measure between users u and u', which yields the two extreme cases discussed above. However, as we will show, one can do better by defining joint feature maps that keep some middle ground and allow for simultaneous generalization along both dimensions.

2. Joint Feature Maps and Kernels

2.1. Hypothesis Classes for Ordinal Regression

To state our modeling approach more formally, we denote by $\mathcal{U}$ a set of users and by $\mathcal{X}$ a set of items. A joint feature map is a mapping $\Psi : \mathcal{U} \times \mathcal{X} \to \Re^D$ which extracts features ψ_r, $1 \leq r \leq D$ from user-item pairs. We will define a family of functions F that are linear in the chosen feature map via $F(u, x; w) = \langle \Psi(u, x), w \rangle$, where $w \in \Re^D$ is a weight vector. In order to predict a rating for a pair (u, x), we use a set of adaptive thresholds θ to quantize F into bins. If there are k response levels (say a rating scale from 0 to $(k-1)$ stars), then there will be thresholds $\theta_j \in \Re$ with $1 \leq j \leq k-1$. For convenience we also define $\theta_k = +\infty$. The prediction function simply picks the number of the bin the computed F-value falls into, formally:

$$f(u, x; w, \theta) = \min\{j \in \{1, \ldots, k\} : F(u, x; w) < \theta_j\}. \quad (1)$$

As we will show in the next section, the estimation of w and θ in the proposed algorithm only depends on inner products $\langle \Psi(u, x), \Psi(u', x') \rangle$ for user-item pairs with observed rating. This means that instead of specifying Ψ explicitly, we can also define kernels functions K over $\mathcal{U} \times \mathcal{X}$ which define a joint feature map implicitly, i.e. $K((u, x), (u', x')) \equiv \langle \Psi(u, x), \Psi(u', x') \rangle$. It will turn out that this is a more convenient way of designing appropriate representations over user-item pairs.

2.2. Joint Feature Maps via Tensor Products

In this paper, we restrict ourselves to joint feature maps that are generically constructed from feature maps for items and users via the tensor product. By this we mean a relatively simple operation of first defining $\Lambda : \mathcal{U} \to \Re^G$ and $\Phi : \mathcal{X} \to \Re^H$ and then combining every dimension of Λ multiplicatively with every dimension of Φ to define $\Psi(u, x) = \Lambda(u) \otimes \Phi(x) \in \Re^D$ where $D = G \cdot H$. Notice that in the simplest case of binary features, the multiplicative combination corresponds to a logical conjunction. Explicitly computing feature maps Ψ constructed in this manner may be prohibitively expensive. However, the following simple lemma shows that inner products of this sort can be computed very efficiently.

Lemma 1. *If $\Psi(u, x) = \Lambda(u) \otimes \Phi(x)$ then the inner product of Ψ vectors can be expressed as the inner product between Λ and Φ vectors, respectively, as follows:*

$$\langle \Psi(u, x), \Psi(u', x') \rangle = \langle \Lambda(u), \Lambda(u') \rangle \langle \Phi(x), \Phi(x') \rangle .$$

This implies that we may design kernel functions K_U for users and K_X for items independently and combine them multiplicatively to define a joint kernel.

2.3. Designing Kernels

We propose to build kernels for users and items by additively combining elementary kernel functions, which are then combined multiplicatively to yield the joint kernel function.

2.3.1. Identity Kernel

The simplest kernel function is the diagonal kernel, which is defined via the Kronecker delta $K^{\mathrm{id}}(z, z') = \delta_{z,z'}$. Interpreting K^{id} as an inner product, this corresponds to a feature map that encodes the identity of each object z by a separate Boolean feature. We will denote the diagonal kernel induced by the user and item identity by K_U^{id} and K_X^{id}, respectively.

2.3.2. Attribute Kernel

The second type of kernel function is built from an explicit attribute representation for items or users. For users these attributes may correspond to demographic information such as gender, age, nationality, location, or income. For items such as documents this may encode a standard tf-idf vector space representation, whereas for movies it may include attributes such as genre or attributes extracted from cast, crew, or a syn-

opsis of the plot. We will refer to the attribute-based kernels by K_U^{at} and K_X^{at}.

2.3.3. Correlation Kernel

Collaborative filtering has demonstrated that predictions and recommendations can be learned based on correlations computed from a given matrix of ratings. The most popular correlation measure is the Pearson correlation coefficient, which corresponds to an inner product between normalized rating vectors. For instance, if applied to correlate users, one can define the so-called z-scores, by computing the user-specific mean $\mu(u)$ and variances $\sigma(u)$ and setting $z(u,x) = \frac{r(u,x)-\mu(u)}{\sigma(u)}$. However, since not all ratings are observed one needs to specify how to deal with missing values. We consider two *ad hoc* strategies for doing this: mean imputation and pairwise deletion. In the first case, unobserved values are identified with the mean value, i.e. their z-score is zero. One can then simply define a kernel via

$$K_U^{\text{co}}(u,u') = \frac{1}{|\mathcal{X}|}\sum_x z(u,x)\cdot z(u',x)\,. \qquad (2)$$

In the second case, one computes the correlation between two users only from the subset of items that have been rated by both. If we use $X(u,u') \subseteq \mathcal{X}$ to denote those intersections, then one can define a correlation matrix via

$$C(u,u') = \frac{1}{|X(u,u')|}\sum_{x\in X(u,u')} z(u,x)\cdot z(u',x)\,, \quad (3)$$

and $C(u,u') = 0$ for $X(u,u') = \emptyset$. The main difference between (2) and (3) is the normalization. Notice that (3) deviates somewhat from the standard use of Pearson correlation in that mean and variance are estimated over the set of all ratings of user u and not just over subsets $X(u,u')$ (Resnick et al., 1994).

While conceptually less preferable, (2) has the advantage to lead to positive semi-definite correlation matrices and hence can be directly used as a kernel. However, as shown in the following paragraph, the symmetry of C in (3) is sufficient to use it as the generator for a kernel. Finally, note that a similar kernel K_X^{co} can be defined over items by interchanging the role of users and item in the above derivation.

2.3.4. Quadratic Correlation Kernels

There are two disadvantages of the above correlation kernel. First of all, it is not possible to use the standard pairwise deletion, because this may result in an improper (i.e. not positive semi-definite) correlation matrix C. Second, correlations between users that have very few items in common are often unreliable and corresponding entries of C may be noisy. One way to remedy these two problems is to define a kernel matrix by taking the square of the correlation matrix, $K^{\text{qu}} = C^2$. Notice that K^{qu} is positive semi-definite, since C is symmetric. Intuitively, K_U^{qu} measures user similarity in terms of how similar two users are correlated with other users. Again, a similar kernel K_X^{qu} can be derived for items.

2.3.5. Combining Kernels

The above kernels can be combined by first additively combining kernel functions into a single kernel,

$$K_* = K_*^{\text{id}} + K_*^{\text{at}} + K_*^{\text{co}} + K_*^{\text{qu}}, \qquad (4)$$

where $* \in \{U, X\}$. Different kernels may also be scaled by appropriate scaling factors. Then the joint kernel is obtained as the tensor product $K = K_U \otimes K_X$,

$$K((u,x),(u',x')) = K_U(u,u')K_X(x,x')\,. \qquad (5)$$

Finally, we would like to stress that the joint kernel is perfectly symmetric in users and items. However, by making specific (asymmetric) choices with respect to the kernels K_U and K_X one can derive data representations used in previous work. In particular, the choice of $K_U = K_U^{\text{id}}$ orthogonalizes representations for different users, which implies that predictions for u and $u' \neq u$ are governed by different weights in the weight vector w. In this case, the weight vector can be thought of as a stacked version of these user-specific weights $w = (w_u)_{u\in\mathcal{U}}$.

Lemma 2. *Define $K = K_U^{id} \otimes K_X$ with $K_X(x,x') = \langle\Phi(x),\Phi(x')\rangle$ then there is a partition $w = (w_u)_{u\in\mathcal{U}}$ such that $F(u,x;w) = \langle w_u, \Phi(x)\rangle$.*

Obviously, there will be no generalization across users in this case. A similar observation holds for $K_X = K_X^{\text{id}}$.

3. Perceptron Algorithm

3.1. Design Goals

While most previous machine learning approaches decouple the learning problems associated with each user, our approach leads to a joint problem which couples learning across different users. In order to avoid an undue increase in complexity compared to other methods, we have investigated the use of a perceptron-like training algorithm, which has advantages due to its on-line nature (e.g. early stopping, fast re-training, small memory footprint).

Moreover we have identified two additional design goals. We would like to work with multi-level response

Algorithm 1 JRank: joint kernel perceptron ranking.

```
 1: input: number of iterations, training set of ratings
 2: α(u,x) = 0 for all training pairs (u,x,r)
 3: for s = 1,...,k−1 do θ_s = 0; θ_k = ∞
 4: for a fixed number of iterations do
 5:    for all training ratings (u,x,r) do
 6:       r̂ = f(u,x;α,θ) from (1)
 7:       if r̂ > r then
 8:          α(u,x) = α(u,x) + (r − r̂)
 9:          for s = r,...,r̂ − 1 do θ_s ← θ_s + 1
10:       else if r̂ < r then
11:          α(u,x) = α(u,x) + (r − r̂)
12:          for s = r̂,...,r − 1 do θ_s ← θ_s − 1
13:       end if
14:    end for
15: end for
16: output: parameters α and θ
```

variables on an *ordinal scale*, since this is appropriate for most applications. This means that we consider the total order among ratings, but avoid interpreting the rating as an absolute numeric value. The resulting problem is well-known as *ordinal regression*. Secondly, since we want to use implicit data representations via kernel functions, it is mandatory to work in a *dual representation* which only makes use of inner products between (joint) feature vectors. Putting all three aspects (on-line, ordinal, kernel) together, we propose to generalize the perceptron ranking algorithm of (Crammer & Singer, 2002) as described in the sequel.

3.2. Joint Perceptron Ranking (JRank)

The generalization of perceptron learning to ordinal regression has been called perceptron ranking or *PRank* (Crammer & Singer, 2002). Here we use basically the same algorithm, with the key difference that the prediction problems for different users are coupled through the use of joint kernel functions. Moreover, in our model the thresholds that define the binning of the F-values are shared by all users. Similarly to the dual-form perceptron learning algorithm for binary classification, we introduce parameters $\alpha(u,x)$ for every training observation (u,x). The resulting algorithm is described in Algorithm 1. Updates occur if the predicted rating of an example (u,x) is incorrect (line 7 or 10). In w space, the updates are performed in direction of $\Psi(u,x)$ with a step size given by the (difference) between true and predicted rating. Notice that f is defined in terms of F, which is computed as

$$F(u,x;\alpha) = \sum_{(u',x')} \alpha(u',x') K((u,x),(u',x')) \,. \quad (6)$$

Convergence proofs under suitable separability conditions and a mistake bound analysis can be found in (Crammer & Singer, 2002). Also, it is straightforward to verify that JRank reduces to binary perceptron learning when $k = 2$. In our experiments, we have actually used a more aggressive margin-sensitive update rule, which also updates if no sufficient margin is obtained, i.e. for (u,x,r) if $\langle w, \Psi(u,x)\rangle - \theta_{r-1} < \gamma$ or $\theta_r - \langle w, \Psi(u,x)\rangle < \gamma$ for some fixed constant $\gamma \geq 0$.

4. Related Work

Clearly, we are not the first ones to point out potential benefits of combining collaborative and content-based filtering techniques. The most popular family of methods are *hybrid* in nature. In the Fab system (Balabanovic & Shoham, 1997), content analysis is employed to generate user profiles from Web page ratings. The concept of filterbots was introduced in (Sarwar et al., 1998) to refer to fictive users (bots) who generate ratings based on content. This approach was further extended in (Good et al., 1999) by including various user-specific filterbots (Claypool et al., 1999) propose a modular approach where independent predictions are computed by separate content filtering or collaborative filtering modules. (Melville et al., 2002) uses content-based predictors (naive Bayes) to impute missing values and create pseudo-user profiles.

A second family of approaches treat user rating prediction as a machine learning problem, where the prediction function is learned from labeled examples. In (Basu et al., 1998) this philosophy was implemented by constructing set-valued features that contain either a set of users who like a specific movie or a set of movies which are liked by a particular user. In addition, content features and hybrid features are defined and used as the input representation for a rule induction system. Similarly, the approaches of (Billsus & Pazzani, 1998) and (Crammer & Singer, 2002) described above can directly incorporate item features, if available.

5. Experiments

5.1. Data Sets and Experimental Setup

To evaluate the approach outlined above we use the EachMovie[1] data set that consists of 72,916 users and 2,811,983 recorded ratings on 1,628 different movies. We have scaled each rating to be on a zero to five star scale. The Internet Movie Database[2] was used to collect item attributes relating to genre, cast, crew,

[1] courtesy of Digital Equipment Corporation

[2] `http://www.imdb.com`

User features				Mean average error		Mean zero-one error		Expected rank utility	
$K_{\mathcal{U}}^{\mathrm{id}}$	$K_{\mathcal{U}}^{\mathrm{at}}$	$K_{\mathcal{U}}^{\mathrm{co}}$	$K_{\mathcal{U}}^{\mathrm{qu}}$	PRank	JRank	PRank	JRank	PRank	JRank
○				1.122	**0.959**	0.651	**0.648**	0.721	**0.750**
	○			1.322	1.110	0.682	0.684	0.698	0.722
		○		1.115	0.989	0.650	0.654	0.719	0.745
			○	1.121	0.991	0.651	0.654	0.721	0.745
○	○			1.353	1.010	0.688	0.659	0.696	0.745
○		○		1.222	0.958	0.667	**0.646**	0.707	0.752
○			○	1.227	**0.956**	0.668	**0.646**	0.708	**0.753**
	○	○		1.356	1.017	0.689	0.661	0.693	0.742
	○		○	1.353	1.019	0.688	0.662	0.696	0.740
		○	○	1.221	0.978	0.666	0.651	0.707	0.748
○	○	○		1.380	0.993	0.691	0.655	0.692	0.746
○	○		○	1.381	0.994	0.691	0.656	0.692	0.746
○		○	○	1.284	**0.961**	0.676	**0.647**	0.702	**0.751**
	○	○	○	1.380	1.001	0.691	0.657	0.692	0.744
○	○	○	○	1.402	**0.991**	0.695	**0.655**	0.688	**0.747**

Table 1. Different combinations of user features tested with item correlations. Results are averaged over 100 trials with 100 training users, 2000 input users, and 800 training items.

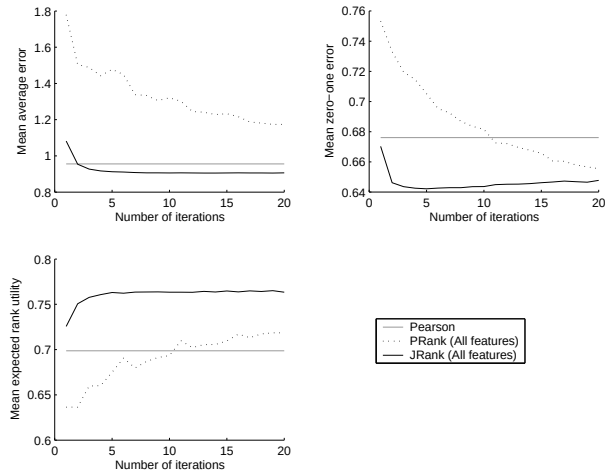

Figure 1. Convergence of JRank and PRank using all features for both users and items. Plots show averages over 100 trials with 1000 input users, 100 training users, and 800 training/input items.

country, language, and keywords of a movie. The plot synopsis was utilized to generate a tf-idf representation. We also used the (often incomplete) demographic information about users in EachMovie about gender, age, and residence (zip-code, first two digits used). All multivariate attributes were encoded using a standard orthogonal (binary) feature representation.

The randomized generation of training and test data has been conducted as follows. First, we have eliminated users with incomplete attributes and with fewer than 10 ratings, leaving a total of 22,488 out of the initial 72,916 users. Items without ratings or with no valid attributes have also been removed; 1,613 out of the 1,628 items were retained. Second, we have randomly subsampled rows and columns of the rating matrix to produce a submatrix of user-item pairs. Third, we randomly divide the selected items into training and test items. In order to compute correlation matrices, we also make use of the users and items that are not part of the selected submatrix. Hence, these ratings only enter on the *input* side and do not contribute as training instances. The reported results are averaged over multiple trials of this procedure.

We have evaluated the JRank algorithm using various combinations of kernels. To show the competitiveness of our approach, we also compare JRank with a standard collaborative filtering algorithm based on the Pearson correlation coefficient (Resnick et al., 1994) and with single user PRank. Unless otherwise specified, JRank and PRank are trained in random order for a maximum of 5 iterations with γ set to 1.

5.2. Evaluation Metrics

We have utilized three evaluation metrics which quantify the accuracy of predicted ratings $\hat{R} = (\hat{r}_1, ..., \hat{r}_n)$ with respect to true ratings $R = (r_1, ...r_n)$.

- *Mean average error* - The mean average error is just the average deviation of the predicted rating from the actual rating $E(R, \hat{R}) = \|R - \hat{R}\|_1/n$.
- *Mean zero-one test error* - The zero-one error gives an error of 1 to every incorrect prediction $E(R, \hat{R}) = |\{i : r_i \neq \hat{r}_i\}|/n$.
- *Expected rank utility* - In many applications, such as generating recommendations, correctly ranking items may be more important than predicting ratings for individual items. In particular, one would

Item features				Mean average error		Mean zero-one error		Expected rank utility	
$K_{\mathcal{X}}^{\mathrm{id}}$	$K_{\mathcal{X}}^{\mathrm{at}}$	$K_{\mathcal{X}}^{\mathrm{co}}$	$K_{\mathcal{X}}^{\mathrm{qu}}$	PRank	JRank	PRank	JRank	PRank	JRank
○				1.139	1.269	0.696	0.739	0.559	0.559
	○			1.120	1.040	0.666	0.695	0.671	0.685
		○		1.119	0.998	0.649	0.657	0.721	**0.743**
			○	**0.961**	0.985	**0.630**	0.653	0.731	0.732
○	○			1.121	1.042	0.666	0.696	0.671	0.685
○		○		1.114	0.997	0.649	0.657	0.722	0.744
○			○	0.938	0.947	0.642	0.654	0.722	0.728
	○	○		1.139	**0.930**	0.650	0.648	0.721	0.756
	○		○	1.100	1.013	0.659	0.687	0.685	0.701
		○	○	1.086	0.958	**0.638**	0.643	0.734	**0.758**
○	○	○		1.139	0.931	0.650	0.648	0.720	0.756
○	○		○	1.100	1.016	0.659	0.689	0.685	0.700
○		○	○	1.084	0.952	**0.639**	0.643	0.733	0.758
	○	○	○	1.128	**0.918**	0.645	0.644	0.725	**0.761**
○	○	○	○	1.130	**0.921**	**0.646**	**0.646**	0.723	**0.762**

Table 2. Different combinations of item features with user ratings. Results are averaged over 100 trials with 100 training users, 2000 input users, and 800 training items.

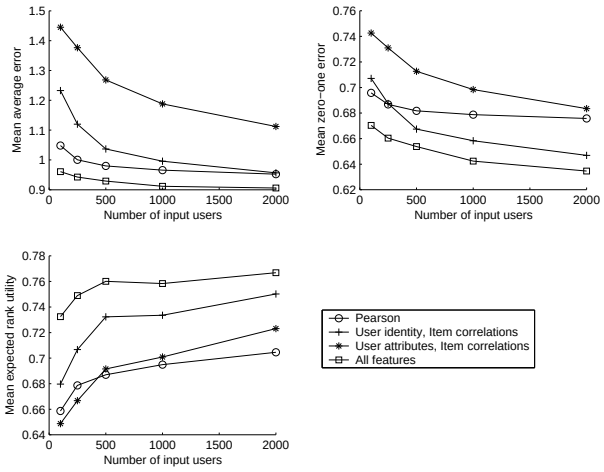

Figure 2. Prediction accuracy for different values of input users and for various combinations of features, averaged over 100 trials with 100 training users and 800 training/input items.

like to put more emphasis on the quality of the top ranked items. Thus, we have utilized the metric proposed in (Breese et al., 1998), which measures the expected utility of a proposed ranking by scoring items in a recommendation list from top to bottom using an exponential discounting factor.

5.3. Results and Discussion

In the evaluation of JRank we have focused on the following aspects: (i) Prediction accuracy for different combinations of features on the user and item side. (ii) Convergence behavior and speed. (iii) Prediction accuracy as a function of the size of the training data and the amount of data used to compute the kernel matrices.

5.3.1. Features

Since one of the main advantages of our approach is that it allows for user features to be utilized when learning the prediction function, we sought to investigate the relative performance of different sets of user features. Each of the 15 possible combinations of user kernels were evaluated using only correlation features on the item side. The results in Table 1 indicate that user identity features are helpful, which is not surprising, since they enable user-specific learning and the set of training and test users is the same. In addition, learning on the user-item pairs with just the user identity features does better in all evaluation metrics than PRank. The coupled learning of JRank almost always has better performance than PRank. The user correlation and quadratic correlation features are also very useful when enough collaborative information is provided, though when combined together they provide only a minor additional benefit. The least predictive features are the user attributes, which had to be expected because of the low quality of that data. The best results are obtained by combining user identity with the correlation and/or quadratic correlation.

We have repeated the investigation for different combinations of item features. As the results in Table 2 show, using the item identity feature does not help, which is an artifact of our splitting scheme since test items are not part of the training set. The correlation and quadratic correlation features are again the most useful ones. The item attributes seem to have more predictive value than the user attributes and including them is beneficial. The best combination of features

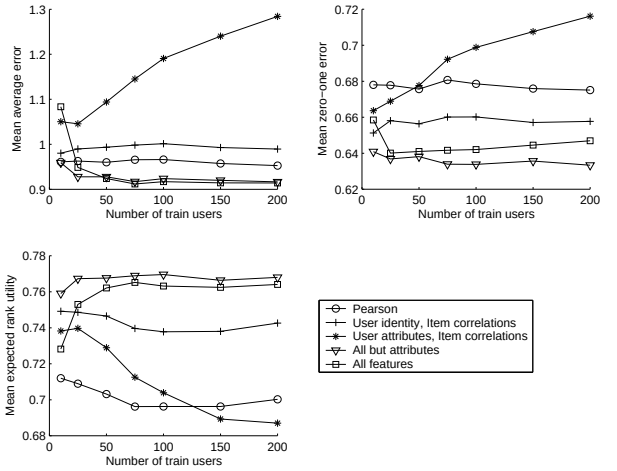

Figure 3. Performance for different number of training users. The number of trials was chosen so that: number of training users $\times$ number of trials $= 10,000$.

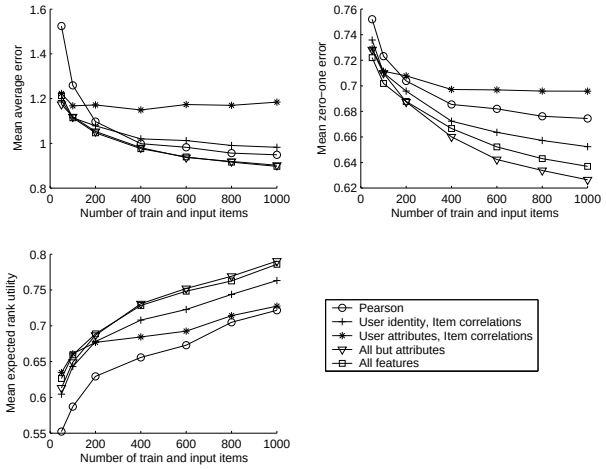

Figure 4. Varying the number of training and input items.

for items is correlation and quadratic correlation features, though adding the identity and attributes might also be useful when not as much collaborative information is given as input.

5.3.2. Convergence

We have investigated how the performance of the algorithm is affected by the number of performed training iterations. The results are summarized in Figure 1. After three iterations JRank has better performance than Pearson according to all three evaluation metrics and it always does better than PRank. The performance for JRank improves according to all three metrics up to five iterations. Training for more iterations causes the mean zero-one error to worsen again while the mean average error and expected rank utility continue to improve slightly.

5.3.3. Varying Input Conditions

We have varied the number of users or items used in training and the number of users used to compute the kernel matrices. Figure 2 shows that the availability of a larger user base yields improvements for all three performance measures. Using attributes can reduce the performance loss on a small user base.

Next we have varied the number of training users while fixing the number of input users at 1000. This allows us to study how the accuracy changes when different numbers of users are coupled together during training. The results shown in Figure 3 show an advantage for using more training users under most conditions. While the improvements are larger initially, after grouping together about 50 to 100 training users the improvements start to flatten out. In a similar manner, we can keep the number of training users fixed at 100 and vary the number of items used for both training and input (which are equal), as shown in Figure 4. Clearly, using more items, and thus ratings, in training improves the prediction accuracy.

Overall, providing more collaborative information and more training users or items improves the performance of JRank according to all three metrics. Table 3 shows the result of running Pearson and 10 iterations of JRank for 100 trials with 100 training users, 5000 input users, and 1000 training/input users. The results indicate that JRank with all features (or all features except the (weak) attributes) consistently outperforms the Pearson correlation method. JRank has a longer running time than training a separate function for each user, however it is an on-line algorithm that can be incrementally updated and methods such as caching can significantly reduce the running time.

6. Conclusion

We have presented a novel learning architecture for the problem of predicting user ratings based on the idea of defining kernel functions over user-item pairs. The proposed JRank algorithm yields substantial improvement over state-of-the art methods and can systematically integrate any information available. Moreover, we have seen that the coupling of learning problems in JRank with a a shared set of thresholds and using hybrid information is advantageous compared to learning a separate PRank function for each user. The joint learning setting seems to help with degenerate situations when few training ratings are given for a single user, e.g. when no rating for a particular response level are available. While the user and item attributes are

$K_{\mathcal{U}}^{\mathrm{id}}$	$K_{\mathcal{U}}^{\mathrm{at}}$	$K_{\mathcal{U}}^{\mathrm{co}}$	$K_{\mathcal{U}}^{\mathrm{qu}}$	$K_{\mathcal{X}}^{\mathrm{id}}$	$K_{\mathcal{X}}^{\mathrm{at}}$	$K_{\mathcal{X}}^{\mathrm{co}}$	$K_{\mathcal{X}}^{\mathrm{qu}}$	Mean average error	Mean zero-one error	Expected rank utility
○		○	○			○	○	0.880	**0.621**	0.791
○		○	○	○		○	○	**0.877**	**0.621**	**0.793**
○	○	○	○	○	○	○	○	0.882	0.624	0.792
			Pearson					0.936	0.673	0.736

Table 3. Results for 100 training users, 5000 input users, and 1000 training/input items.

not very helpful when a large database of ratings is given, they may prove to be more useful in cold-start situations, for example, when new users or items enter the system for which little or no rating information is available.

Acknowledgments

This work was sponsored by an NSF-ITR grant, award number IIS-0312401.

References

Balabanovic, M., & Shoham, Y. (1997). Fab: Content-based, collaborative recommendation. *Communications of the ACM*, *40*, 66–72.

Basu, C., Hirsh, H., & Cohen, W. W. (1998). Recommendation as classification: Using social and content-based information in recommendation. *Proceedings of the 15th National Conference on Artificial Intelligence* (pp. 714–720).

Billsus, D., & Pazzani, M. J. (1998). Learning collaborative information filters. *Proceedings of the 15th International Conference on Machine Learning* (pp. 46–54).

Breese, J. S., Heckerman, D., & Kardie, C. (1998). Empiricial analysis of predictive algorithms for collaborative filtering. *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence* (pp. 43–52).

Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D., & Sartin, M. (1999). Combining content-based and collaborative filters in an online newspaper. *Proceedings of ACM SIGIR Workshop on Recommender Systems.*

Crammer, K., & Singer, Y. (2002). Pranking with ranking. *Advances in Neural Information Processing Systems 14* (pp. 641–647).

Goldberg, D., Nichols, D., Oki, B., & Terry, D. (1992). Using collabrorative filtering to weave an information tapestry. *Communications of the ACM*, *35*, 61–70.

Good, N., Schafer, J. B., Konstan, J. A., Borchers, A., Sarwar, B. M., Herlocker, J. L., & Riedl, J. (1999). Combining collaborative filtering with personal agents for better recommendations. *Proceedings of the 16th National Conference on Artificial Intelligence* (pp. 439–446).

Lang, K. (1995). NewsWeeder: Learning to filter netnews. *Proceedings of the 12th International Conference on Machine Learning* (pp. 331–339).

Melville, P., Mooney, R. J., & Nagarajan, R. (2002). Content-boosted collaborative filtering for improved recommendations. *Proceedings of the 18th National Conference on Artificial Intelligence* (pp. 187–192).

Mooney, R. J., & Roy, L. (2000). Content-based book recommending using learning for text categorization. *Proceedings of the 5th ACM Conference on Digital Libraries* (pp. 195–204).

Pazzani, M., Muramatsu, J., & Billsus, D. (1996). Syskill & Webert: Identifying interesting web sites. *Proceedings of the 13th National Conference on Artificial Intelligence* (pp. 54–61).

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., & Riedl, J. (1994). GroupLens: An open architecture for collaborative filtering of netnews. *Proceedings of the ACM Conference on Computer Supported Cooperative Work* (pp. 175–186).

Sarwar, B. M., Karypis, G., Konstan, J. A., & Reidl, J. (2001). Item-based collaborative filtering recommendation algorithms. *Proceedings of the 10th International World Wide Web Conference* (pp. 285–295).

Sarwar, B. M., Konstan, J. A., Borchers, A., Herlocker, J. L., Miller, B. N., & Riedl, J. (1998). Using filtering agents to improve prediction quality in the GroupLens research collaborative filtering system. *Proceedings of the ACM Conference on Computer Supported Cooperative Work* (pp. 345–354).

Schölkopf, B., & Smola, A. J. (2001). *Learning with kernels.* Cambridge, MA: MIT Press.

Shardanand, U., & Maes, P. (1995). Social information filtering: Algorithms for automating 'word of mouth'. *Human Factors in Computing Systems ACM CHI* (pp. 210–217).