Compound Classification Models for Recommender Systems

Lars Schmidt-Thieme
Institute of Computer Science,
University of Freiburg, Germany
lst@informatik.uni-freiburg.de

Abstract

Recommender systems recommend products to customers based on ratings or past customer behavior. Without any information about attributes of the products or customers involved, the problem has been tackled most successfully by a nearest neighbor method called collaborative filtering in the context, while additional efforts invested in building classification models did not pay off and did not increase the quality. Therefore, classification methods have mainly been used in conjunction with product or customer attributes.

Starting from a view on the plain recommendation task without attributes as a multi-class classification problem, we investigate two particularities, its autocorrelation structure as well as the absence of re-occurring items (repeat buying). We adapt the standard generic reductions 1-vs-rest and 1-vs-1 of multi-class problems to a set of binary classification problems to these particularities and thereby provide a generic compound classifier for recommender systems. We evaluate a particular specialization thereof using linear support vector machines as member classifiers on MovieLens data and show that it outperforms state-of-the-art methods, i.e., item-based collaborative filtering.

1. Introduction

Recommender systems are online information systems that recommend products to customers, i.e., perform some sort of automatic selling in e-commerce, or more generally, recommend information items to users, e.g., books to library users, research papers to citeseer users, courses to students etc. Contrary to static lists like best sellers, special offers, editor's choice, etc. recommender systems typically are personalized and targeted at the individual customer. To achieve personalization, they make use of customer profiles consisting of explicit and implicit product ratings. In computer science, the term recommender system often is used synonymously with collaborative filtering, in the context of information retrieval they are also known as relevance feedback.

A typical recommendation scenario is shown in fig. 1. The system contains a set of users and items, partial rating information by users about items as well as information about attributes of users and items. The task is to predict further ratings of users or recommend new items that will achieve high ratings by users.

This problem has been handled in two different ways in the research literature so far: (i) using heuristic correlation measures and a simple nearest neighbor method called collaborative filtering, and (ii) using learning methods to train a classification model that predicts further ratings or rated items. Already in an early publication [4], the nearest neighbor methods outperformed the classification models on the plain problem without item or user attributes. Until now the reasons for the failure of the classification models are not well understood. One would expect, that the investment in a more complex learning method pays off if applied correctly. Since then, several attempts have been made to put classification methods to work for recommendation tasks, mostly by using either information about item or user attributes (sometimes called content-based and demographic filtering), or by building probabilistic models customized for the problem. But many solutions turned out inferior, many solutions never have been compared to collaborative filtering at all or not on the well-known data sets such as MovieLens.

In this paper we will thoroughly re-investigate standard classification models for recommender systems. We will identify two specific properties of the problem that make straight-forward applications of classification models fail: the intrinsic autocorrelation structure and the absence of item re-occurrences (repeat buying). Furthermore we will adapt the well-known multi-class model setup strategies 1-vs-rest and 1-vs-1 to reflect these particularities, providing a generic framework for using any binary classifier for recommendation generation. Finally, we will evaluate a compound 1-vs-rest and 1-vs-1 classifier using linear SVMs as member models and show that the 1-vs-1 model outper-

Item attributes (upper-right table):

item	1	2	3	4	5	6	7	8	9	...
year	'95	'95	'95	'95	'95	'95	'95	'95	'95	
action	−	+	−	+	−	−	−	−	−	
children's	+	−	−	−	−	−	−	+	−	
...										

User attributes (lower-left table):

	gender	age	...
user 1	m	24	
2	f	53	
3	m	23	
4	m	24	
5	f	33	
6	m	42	
:			

Rating data (lower-right table):

item	1	2	3	4	5	6	7	8	9	...
user 1	4			3						
2	1									
3		4								
4				5						
5								2		
6								4		
:										

Figure 1. Different data involved in recommendation tasks. The lower-right-lower table contains rating data of users on items (here movies), the lower-left table contains attributes of users, the upper-right table cointains attributes of items.

forms the state-of-the-art methods, i.e., item-based collaborative filtering.

We will shortly review related work in the following section, give a formal outline of the different recommendation tasks in sections 3 and 4, and describe some baseline and state-of-the-art solutions for the problem that do not use learning methods in section 5. In section 6 we describe different model setups that turn the recommendation task in a classification problem, i.e., the adaptation of the 1-vs-rest and 1-vs-1 multi-class setups, and in section 7 provide some experimental support for the superiority of our approach. Finally, we conclude with some remarks on open problems.

2. Related Work

While having their very early roots in relevance feedback in information retrieval [17] and adaptive hypermedia [19], recommender systems gained momentum once they had been formulated as filtering techniques, generally grouped in three different types: (i) *collaborative filtering* is basically a nearest-neighbor model based on user–item correlations; if correlations are computed between users, it is called *user-based*, if between items, it is called *item-based*. (ii) *content-based* or *feature-based* recommender systems use similarities between rated items of a single user and items in the repository. User- and item-based collaborative filtering and content-based recommender systems have been introduced in [9, 16], [18] and [2], respectively, and are exemplified by the three systems presented there, MovieLens, Ringo, and fab. (iii) *Hybrid* recommender systems try to combine both approaches [2, 5, 15, 20].

While the so-called content-based and hybrid methods try to integrate classifiers based on item and/or user attributes with collaborative filtering techniques, two further approaches have been taken: (iv) recommender systems have been viewed as classification problems [3, 4, 12] and different model classes have been tried, but either classification models did not improve quality much or they have not been compared to strong collaborative filtering models such as item-based.

Finally, (v) special probabilistic models have been built to catch the particularities of the recommendation task [6, 14, 10, 1]. Again, either attributes are used or results do not improve much on collaborative filtering or results are not compared to state-of-the-art collaborative filtering methods; and anyway, specialized models do not allow for a pluggability of the learning component as our compound model does.

So still, item-based collaborative filtering models claim to provide the best published results on some of the public data sets such as MovieLens [7].

3. Recommendation Tasks

Generally, the data used in recommender systems can be described by

(i) a set of modes (users/customers/agents, items/documents/products, contexts, tasks, etc.),

(ii) the attributes of these modes (demographic information, properties of products, descriptions of circumstances etc.), and

(iii) a partial cube of ratings for combinations of instances of the modes.

All real-life systems we are aware of as well as most research literature focuses on just two modes, users and items, as the sparsity of the rating matrix already is a problem for two-modal data and would be even more severe if more modes are considered.

Let U and I be sets of uninterpreted elements (e.g., integers), called **users** and **items**, respectively. Let $S \subseteq \mathbb{R}$ be the set of possible ratings, e.g., $S = \{1, 2, 3, 4, 5\}$, where higher values indicate a stronger liking, and

$$r : U \times I \to S \quad \text{partial}$$

a partial function that associates ratings to user/item pairs, i.e., r is defined only for some, in general not for all pairs (u, i) for a user $u \in U$ and an item $i \in I$, as users typically rate only small subsets of items. We denote the set of pairs r is defined for as its domain $\mathrm{dom}_r \subseteq U \times I$. In data sets, r typically is represented as a list of tuples $(u, i, r(u, i))$; often a timestamp is available as a fourth field providing information about the actual order in which ratings have been entered by users.

We can distinguish two different tasks that should be accomplished by recommender systems:

(i) **predict the ratings**, i.e., given the rating matrix r at some point of time, predict the new ratings in the rating matrix r' at a later point in time, i.e., compare

$$\hat{r}(u, i) \text{ with } r'(u, i) \text{ for all } (u, i) \in \mathrm{dom}_{r'}$$

where $\hat{r}$ denotes the ratings predicted by the recommender system. As we evaluate $\hat{r}$ only for a given set of pairs, this problem sometimes is called "forced".

(ii) **predict the rating events**, i.e., given the rating matrix r at some point of time, predict the new pairs for which rating information is available in the rating matrix r' at a later point in time, i.e., compare

$$\mathrm{dom}_{\hat{r}} \text{ with } \mathrm{dom}_{r'}$$

For rating events, one typically is neither interested in a mere binary prediction (will occur, will not occur) nor in the exact probabilities of an event occurring, but in a ranking of all possible events by descending probability of occurrence.

In the literature, rating events typically are grouped by user: events, i.e., items, are predicted for each user, as recommender systems typically work in a pull-scenario like an online shop or an information portal where we have no control about which users are there. In push scenarios like mailings grouping by items also would make sense, e.g., if we look for customers to approach about a given product.

At first sight, it might look as if the second problem can be addressed by solutions of the first problem: when we already have a function to predict ratings, we predict events (items for a given user) by decreasing predicted rating. This makes perfect sense in many applications where we want to recommend only items that users will like, not items, we expect them to rate and eventually not like, especially, as in applications the predicted rating is not shown. But obviously, having access to both predictions, the rating and the rank of an item for a user, allows us to implement additional services like issuing warnings (high rank, low rating) and stress recommendations for items that are unlikely be found by the user (low rank, high rating). Furthermore, in evaluations in the lab, using rating predictions for rank predictions will only work if there is a very strong correlation between the rating and the item occurrence probability: in published data sets such as MovieLens such a correlation can be observed, but it might be overlaid by other effects, as, e.g., incomplete information (we do not know in advance if we will like a movie) and variety seeking customer behavior. Furthermore, the second problem is of importance on its own as in some applications ratings are collected implicitly by customer behavior, e.g., the web pages viewed or the products put in a market basket or bought, and no explicit ratings are available.

So both problems should be treated on their own. If events should be predicted, it is a good idea to discard the actual rating values, i.e., r becomes constant 1. Please remember that r is a partial function and only recorded for observed pairs. In dense matrix representations r often is encoded by two values, 0 and 1: but 0 does not mean that we observed a counterexample, but just that there is a missing value.

Recommender tasks can be further described by the availability of information on attributes of users and items. As already mentioned, most recent approaches that aimed at using classification models for recommender systems tried to gain advantage from taking attributes into account. But (reliable) attributes for customers and — in some domains such as community-driven information portals or product domains with high fluctuations in the assortment and a high heterogeneity of products (like auctions) — also for items may be hard to come by. In these cases we will have to resort to methods that work without attributes.

4. Predicting Rating Events or Items

In the following, we will address the most simple recommendation task, as we think that it is at the heart of the problem: (i) we do not consider ratings, but just events, i.e., unary data and prediction of ranks of items, (ii) we do not consider attributes of items or users, but just atomic entities.

Data for recommender systems with users and items

without attributes simply can be described by a multiset $\mathcal{T} \subseteq \mathcal{P}(I)$ of subsets of items called **transactions** in the following. A recommender system for users and items without attributes can be described by

$$\hat{r} : \mathcal{P}(I) \rightarrow \text{ranking}(I)$$

where $\text{ranking}(I)$ denotes the set of all bijective functions $I \rightarrow \{1, \dots, |I|\}$ that map each item to its unique rank on the recommendation list. For $n \in \mathbb{N}$ let

$$\hat{r}^{\leq n} : \quad \begin{array}{ccl} \mathcal{P}(I) & \rightarrow & \text{ranking}(I) \\ X & \mapsto & \hat{r}(X)^{-1}(\{1, \dots, n\}) \end{array}$$

be the set of items recommended at ranks 1 to n.

Such recommender systems are evaluated by a test multiset of split transactions, $\mathcal{T}^{\text{test}} \subseteq \mathcal{P}(I) \times \mathcal{P}(I)$ and a number n of relevant rank positions counting as hits by the usual recall measure:

$$\text{recall}_n^{\text{micro}}(r) := \frac{\sum_{(X,Y) \in \mathcal{T}^{\text{test}}} \frac{1}{|Y|} |Y \cap r^{\leq n}(X)|}{|\mathcal{T}^{\text{test}}|}$$

$$\text{recall}_n^{\text{macro}}(r) := \frac{\sum_{(X,Y) \in \mathcal{T}^{\text{test}}} |Y \cap r^{\leq n}(X)|}{\sum_{(X,Y) \in \mathcal{T}^{\text{test}}} |Y|}$$

In this evaluation scenario, precision is forced by taking into account only a restricted number n of recommendations, so that there is no need to evaluate precision or F1 measures: with fixed n, precision (and thus F1) is just the same as recall up to a multiplicative constant. Alternatively, evaluations could either (i) take into account the whole ranking, but weight down hits at lower ranks [4] or (ii) expect recommender systems to adapt the number of recommendations given by themselves and measure then full recall, precision, and F1. All three possibilities are not unproblematic as (i) depends on the choice of a decay constant and (ii) does not take into account the ranks at all. We choose recall of clipped rankings because it reflects some specifics of recommender web applications where a fixed number of products are shown per page, because it is the simplest one and because it is used frequently in recommender systems literature. — Please note that for a fair comparison we have to make sure that all recommender systems actually provide at least n recommendations; if not, ranks have to be filled by a fallback system.

In predicting ranks of items we have to take into account an additional property of most recommender systems: typically they are used to call users attentions to new items users do not know yet and have not rated already in the past, which may be just a desired feature of the system or be due to the fact that there is no repeat-buying in domains like books, movies, music etc. in which these systems typically operate, such that recommending already known items does not make much sense economically. There are, of course, other domains, such as food stores and broadcast media like radio and TV, where re-occurring events play a crucial role and thus have to be modeled. We will stick to domains without re-occurring events. All recommender systems should take advantage of this information and tweak their recommendation list by removing all items that already occurred in the past of a test case.

5. Baseline and State-of-the-art Methods

To assess the quality of recommendations found by our classification method, we compare it to a baseline model and one of the state-of-the-art methods.

We compare user-specific recommender systems to a (almost) constant recommender system that provides (almost) the same recommendations to all users (sometimes also called "most popular"). For this, items are recommended by decreasing total frequency in the training data. Recommendations may vary from user to user a little bit, as items already rated are removed from the constant list. The scores of this model will tell us, what we can achieve already without personalization and therefore, when we compare to personalized systems, how large the benefit from personalization actually is.

Furthermore, we compare to a simple, non-learned nearest neighbor classifier, called item-based collaborative filtering. Depending on a parameter $k \in \mathbb{N}$ called **neighborhood size**, it proceeds in three simple steps:

(i) Compute item correlations $C := (\text{corr}(i,j))_{i,j \in I}$:

$$\text{corr}(i,j) := \frac{|\{T \in \mathcal{T} \,|\, i, j \in T\}|}{|\{T \in \mathcal{T} \,|\, i \in T\}| |\{T \in \mathcal{T} \,|\, j \in T\}|}$$

(ii) Sparsify C by keeping only the highest k entries in each column.

(iii) For a test case $X \subseteq \mathcal{P}(I)$ compute item scores via

$$\text{score}(X) := C \cdot \text{ind}(X)$$

where $\text{ind}(X) \in \{0,1\}^I$ with

$$\text{ind}(X)(i) = 1 :\Leftrightarrow i \in X$$

and recommend in order of decreasing scores.

A more detailed description can be found in [7].

6. Multi-class Prediction Setups for Autocorrelation Models

Classification models for recommending items get as training data just a multiset $\mathcal{T} \subseteq \mathcal{P}(I)$ of itemsets containing the items a user has already rated in the past. They have

to solve two problems: (i) how to split the training data into cases and (ii) to learn a multi-class classifier.

If timestamps for ratings are available, training transactions can be split, s.t. the ordering is taken into account: each transaction $(x_1, \ldots, x_k)$ can be split at k different positions $m \in \{1, \ldots, k\}$ and a split at position m in past $(x_1, \ldots, x_{m-1})$ and future $(x_m, \ldots, x_k)$ comprises $k - m + 1$ cases $((x_1, \ldots, x_{m-1}), x_m)$, $((x_1, \ldots, x_{m-1}), x_{m+1})$, $\ldots, ((x_1, \ldots, x_{m-1}), x_k)$.

If no timestamp information is available or considered not important — what we will do in the following —, there are 2^k splits in past and future and several cases made from each split.

As there obviously are too many possible splits, we will restrict to the k splits that take out just one item as future, i.e., $((x_1, \ldots, \hat{x}_i, \ldots, x_k), x_i)$ where $\hat{x}_i$ denotes that the item is dropped from the list.

Once cases have been built, the problem is converted to a usual multi-class classification problem with one binary predictor variable X_i for each item $i \in I$ and a nominal target variable Y with $|I|$ different values, one for each item. In analogy to time-variant scenarios, where the target variable is predicted based on former states of itself, and in accordance with some literature from relational data mining [13] we call such a model an autocorrelation model.

When looking for methods to learn classifications from this data, we can either (i) restrict ourselves to methods that can handle multi-class classification tasks intrinsically (such as decision trees or special versions of SVMs) or (ii) use a generic model setup that allows to use a broader variety of learning methods. As the focus of this paper is not on a particular method, but on solving the problem generically, we choose the second possibility that employs standard methods to reduce the multi-class problem to a set of binary problems.

Two such methods are used in the literature (see, e.g., [11]): 1-vs-rest and 1-vs-1 model setups. In a 1-vs-rest model setup we build $|I|$ different classifiers, one for each class. The classifier for class $i \in I$ is trained with the full training data, where cases that belong to class i are re-labeled as positive (or i), while cases that belong to any other class are re-labeled as negative (or rest). The compound classifier is applied by computing the probability of each class by its member classifiers and then sort the items be decreasing probability.

To adapt a 1-vs-rest model setup to autocorrelation models, one can simplify:

(i) work on transactions, not on cases split in advance: consider a transaction x that contains an item i, say, $x_m = i$. If we split in advance, we will get the positive example $((x_1, \ldots, \hat{x}_m, \ldots, x_k), x_m)$, but also $k - 1$ negative examples $((x_1, \ldots, \hat{x}_{m'}, \ldots, x_k), x_{m'})$ for
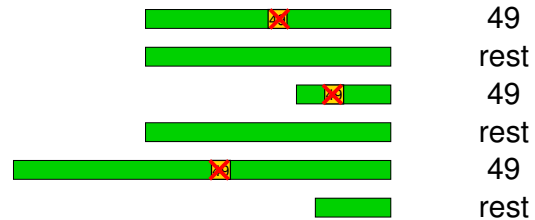
Figure 2. 1-vs-rest model setup adapted to autocorrelation models for $i = 49$.

$m' \neq m$. Furthermore, these negative examples are very similar to the positive example!

Instead, use transactions that contain item i as positive examples (removing i, of course) and transactions that do not contain i as negative examples (not removing any item). See fig. 2.

(ii) As the predictor X_i is missing in all cases of classifier i, it can be dropped.

Simplification (i) reduces the number of cases from the total number $\sum_{T \in \mathcal{T}} |T|$ of item occurrences to the number $|\mathcal{T}|$ of transactions dramatically.

As binary member classifier any classification method could be chosen, e.g., logistic regression, decision trees, support vector machines, neural networks etc.

Alternatively, in a 1-vs-1 model setup, one builds one classifier for each pair (i, j) of competing target classes $i, j \in I$. Each member classifier is trained on all cases that belong either to class i or to class j, all other cases are discarded.

Also the general 1-vs-1 model setup has to be adapted to fit autocorrelation models (see fig. 3):

(i) we have to remove predictor variables X_i and X_j for classifier (i, j).

(ii) when we apply a standard 1-vs-1 model setup to split transactions, then transactions that contain both target items i and j end up as both a case for i and a case for j. These cloned transactions pose a problem as they force the classifier to make a distinction where none can be made. Instead of cloning the transaction, it should be discarded completely, as it does not tell us anything about the difference between i and j.

1-vs-rest compound models contain only $|I|$ member models, while 1-vs-1 compound models contain $|I|(|I| - 1)/2$ different member models. But in usual non-autocorrelation scenarios, both model setups 1-vs-rest and
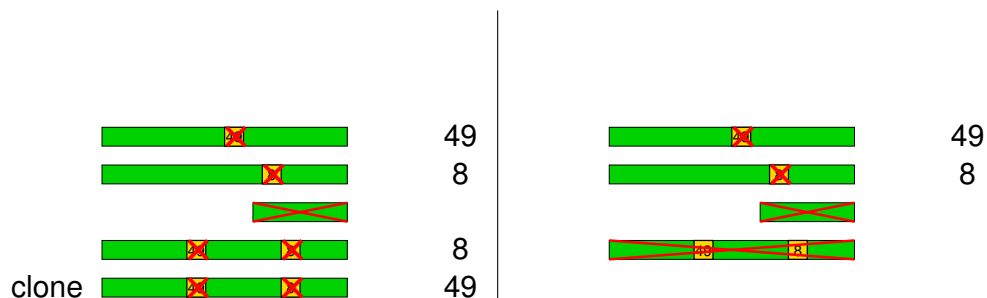
Figure 3. 1-vs-1 model setups adapted to autocorrelation models for $i = 49$ **and** $j = 8$**: a) cloning transactions that belong to both competing classes; b) discarding transactions that belong to both competing classes.**

1-vs-1 train on almost the same number of cases in total: let C denote the original number of cases, then 1-vs-rest trains on $|I| \cdot C$ cases, as each case is used in each classifier either as positive or negative example, while 1-vs-1 trains on $(|I| - 1) \cdot C$ cases, as each case is used in the classifier for each of the $|I| - 1$ competing classes. Also, in the literature both model setups are seen as competitive [11].

For the adaptations to autocorrelation models, it is different. Here, 1-vs-rest compound models are trained on only $|I| \cdot |U|$ many cases, while 1-vs-1 compound models still have to be trained on $(|I| - 1) \cdot C$ cases minus some cases that stem from correlated items. So 1-vs-1 compound models are expected to be much slower for autocorrelation scenarios.

7. Experiments and Evaluation

We build two different classifiers for recommending items without attributes: (i) a 1-vs-rest compound model and (ii) a 1-vs-1 compound model, both making use of a SVM with linear kernel as member model. We used libsvm [8] as work horse to train the member models. We compared these models with the simple constant baseline as well as with the state-of-the-art method for recommender systems, item-based collaborative filtering [7]. The latter is claimed to give the best published results on our evaluation dataset without taking into account attributes. Ties of the item-based collaborative filtering method were broken by the ranks of the constant system, ties in the constant system by smallest index. The neighborhood parameter of the item-based collaborative filtering method was chosen to give optimal results on the test data (neighborhood size 20) by means of a grid search over all neighborhood sizes in steps of 5. So the item-based collaborative filtering system used optimal parameters and thus was granted a small advantage: its score is more an upper bound than a fair quality

measure, but we expect that a calibration of the size of the neighborhood on the training data gives almost the same results.

We evaluated on the classical MovieLens 100k dataset.[1] It contains 100,000 ratings on a 5-point scale from 943 users on 1,682 items. The publicly available version is a subsample of a larger data set, where only users with at least 20 ratings have been retained.

As experimental setup we have chosen a leave-one-out split in training and test data (sometimes also called all-but-one) that was done at random. We used the recall measure that takes into account only the $n = 10$ top-ranked items for each user. For leave-one-out splits there is no difference between micro and macro averaged recall values.

In fig. 4 results are shown. The x-axis restricts the set of competing items: competing items are added from left to right by decreasing total frequency; the x-axis shows the cumulative relative frequency of competitive items considered. The right-most values of each curve give the final result: the figures for the constant and the item-based collaborative filtering system are in line with published results. Additionally, one can see that the constant system achieves its recall almost completely on the 10 most frequent items — it still improves a little bit on less frequent items, as items already in the past are removed from the recommendation list, that allows also items below position 10 to be shown. The item-based collaborative filtering system improves considerably on less frequent items until the items that make approx. 60% of all item occurrences in the dataset are used, from less frequent items it cannot improve much.

The outcome of the two classification methods is a surprise: the 1-vs-rest model completely fails and achieves a score that is way below the constant system. On the other hand, the 1-vs-1 model clearly outperforms the item-based collaborative filtering method. The sharp edge at

_____

[1]http://www.grouplens.org

approx. 60% competitive item occurrences stems from the fact, that we stopped at 240 most-frequent items due to time restrictions (see below about runtime). So actually, we expect the curve to rise even further a little bit.
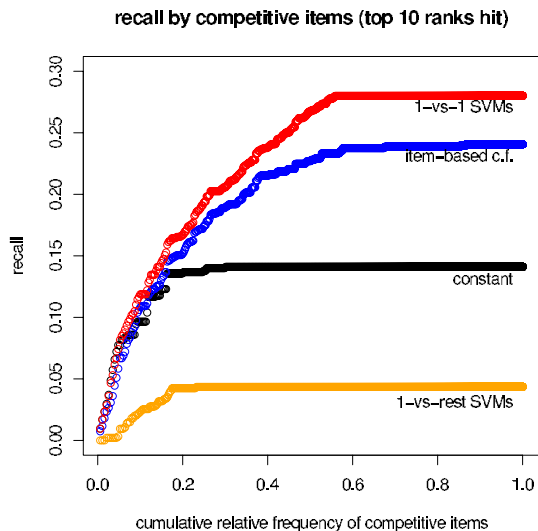
Figure 4. Recall on 10 top-ranked items for a) constant, b) item-based collaborative filtering, c) 1-vs-rest compound classifier using SVMs, and d) 1-vs-1 compound classifier using SVMs.

The failing of the 1-vs-rest model can be explained by the autocorrelation structure and the absence of re-occurring items. 1-vs-rest tries to learn the absolute concept of class i, but all its positive examples cannot be of class i in the evaluation as there is no re-occurrence of items and i has already been rated, and all transactions that eventually can contain item i later on are explicitly used as negative examples. So in a sense it learns just the wrong way. Contrary, 1-vs-1 tries only to learn differences between the concepts of two classes: and as there are many original transactions dropped that contain neither of the two classes considered, these transactions are not labeled as negative examples.

A characteristics of the recommendation task is the very high number of classes: in the example, we have 1682 different movies that play the role of a class. Most multi-class problems have only a comparatively small number of classes, such as 10 for classifying hand-written digits or 26 for hand-written letters. For 1682 classes we have to build 1,413,721 pairwise classifiers. As there are 100,000 item occurrences in total, the sum of all training set sizes is approx. 168,200,000 cases. Even if we could build a pair classifier in a second, we would need approx. 16 days to train

the system. Therefore, we restricted the pairwise classifier to the 240 most-frequent items, i.e., trained 28,680 pairwise classifiers, each taking approx. 3s on average on a standard Linux box (Intel 2.4 GHz, 1.5 GB RAM), resulting in a total training time of approx. 1 day for the whole system. — As we have already seen in section 6, the requirements for the 1-vs-rest classifier are much more modest; it was build in a few hours.

8. Conclusion and Outlook

In this paper, we have argued, that the basic recommendation problem, i.e., recommending items without having access to any attributes of items or users, can be viewed as a classification task. It features a special structure, where the same variable occurs in both roles, as predictor and target variable, what we called autocorrelation model in analogy with similar phenomena in other contexts (time-variant and relational data).

We showed that the standard methods for reducing a multi-class problem to a set of binary problems cannot be used literally for autocorrelation models, but have to be adapted slightly. This adaptation leads to a completely different complexity of these two approaches, contrary to the non-autocorrelation case. Furthermore, we have seen that for scenarios without item re-occurrence (repeat buying), the faster 1-vs-rest classifier completely fails, as it learns to predict past items that cannot occur anymore and treats transactions to which an item eventually is added explicitly as negative examples.

But the 1-vs-1 classifier clearly outperforms the state-of-the-art methods in this domain and for our test data set, item-based collaborative filtering.

Finally, runtime became a real issue as learning the 1-vs-1 classifier takes days, while building the collaborative filtering correlation matrix takes seconds. Further research has to address this problem: beneath obvious possibilities as parallelization that is trivially accomplished for building a set of pairwise classifiers, more sophisticated methods could be thought of, such as, e.g., building hierarchical classifiers only for a restricted set of contrasts that are induced by a taxonomy on the items. Once runtime issues are fixed, handling recommendation tasks as classification tasks will allow to use the whole data mining machinery to tackle the problem, especially, integrating attributes might then be done in a less ad-hoc and heuristic manner than before.

References

[1] A. Ansari, S. Essegaier, and R. Kohli. Internet recommendation systems. *Journal of Marketing Research*, 37:363–375, 2000.

[2] M. Balabanović and Y. Shoham. Fab - content-based, collaborative recommendation. *CACM*, 40(3):66–72, 1997.

[3] D. Billsus and M. J. Pazzani. Learning collaborative information filters. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 46–54, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[4] J. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proc. of the Fourteenth An. Conf. on Uncertainty in AI*, pages 43–52, Madison, WI, USA, July 1998. Morgan Kaufmann.

[5] R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User Adapted Interaction*, 12/4:331–370, 2002.

[6] M. K. Condliff, D. D. Lewis, D. Madigan, and C. Posse. Bayesian mixed-effects models for recommender systems. In *Proceedings of the ACM SIGIR Workshop on Recommender Systems: Algorithms and Evaluation, 22nd Intl. Conf. on Research and Development in Information Retrieval*, 1999.

[7] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Sys.*, 22(1):143–177, 2004.

[8] R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using the second order information for training svm. Technical report, Department of Computer Science, National Taiwan University, 2005.

[9] D. Goldberg, D. Nichols, B. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *CACM*, 35(12):61–70, 1992.

[10] T. Hofmann. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115, 2004.

[11] C.-W. Hsu and C.-J. Lin. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13:415–425, 2002.

[12] V. S. Iyengar and T. Zhang. Empirical study of recommender systems using linear classifiers. In *PAKDD '01: Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 16–27, London, UK, 2001. Springer-Verlag.

[13] D. Jensen and J. Neville. Linkage and autocorrelation cause feature selection bias in relational learning. In *Proceedings of the 19th International Conference on Machine Learning (ICML)*, pages 259–266, 2002.

[14] R. Jin, L. Si, C. Zhai, and J. Callan. Collaborative filtering with decoupled models for preferences and ratings. In *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*, pages 309–316, New York, NY, USA, 2003. ACM Press.

[15] P. Melville, R. J. Mooney, and R. Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Eighteenth national conference on Artificial intelligence*, pages 187–192, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[16] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proc. of the Conf. on Comp. Sup. Coop. Work (CSCW'94)*, pages 175–186, Chapel Hill NC, 1994. Addison-Wesley.

[17] G. Salton. Relevance feedback and the optimization of retrieval effectiveness. In G. Salton, editor, *The SMART system — experiments in automatic document processing*, pages 324–336. Prentice-Hall Inc., Englewood Cliffs, NJ, 1971.

[18] U. Shardanand and P. Maes. Social information filtering: algorithms for automating "word of mouth". In *Proc. of the SIGCHI conf. on Human factors in computing systems*, pages 210–217. ACM Press/Addison-Wesley Publishing Co., 1995.

[19] P. D. Stotts and R. Furuta. Dynamic adaptation of hypertext structure. In *Hypertext'91 Proc., San Antonio, TX, USA*, pages 219–231. ACM, 1991.

[20] C. Ziegler, L. Schmidt-Thieme, and G. Lausen. Exploiting semantic product descriptions for recommender systems. In *Proc. 2nd ACM SIGIR Semantic Web and IR WS (SWIR '04), 2004, Sheffield, UK*, 2004.