India-Norway Workshop on Web Concepts and Technologies, Trondheim, Norway, October 3rd, 2011

Adaptive Aggregation of Recommender Systems

Olav Bjørkøy (olavfrih@stud.ntnu.no)
Department of Computer Sciences
Norwegian University of Science and Technology
Trondheim, Norway

Abstract—**Modern recommender systems combine multiple standard recommenders in order to leverage disjoint patterns in available data. These aggregations can for example be done by estimating weights that result in an optimal combination. However, we posit these systems have an important weakness. There exists an underlying, misplaced subjectivity to relevance prediction. In an aggregation of different recommenders, each chosen algorithm reflects one view of how users and items *should* be modeled. We believe this selection should be adaptively and automatically chosen based on how accurate their prediction is likely to be for each user and item. This paper presents a novel method for prediction aggregation that we call *adaptive recommenders*. Multiple recommender systems are combined on a per-user and per-item basis by estimating how accurate each recommender will be for the current user and item. This is done by creating a set of secondary error estimating recommenders. Our initial results are promising, showing that adaptive recommenders can outperform basic methods and simple aggregation approaches.**

1. Introduction

Having too much information can be as harmful as having no information at all. While having too little is an obvious problem, too much leads to information overload where relevant content drowns in irrelevant noise. This is a common problem. Whenever we have enough information, anything extra only leads to confusion. Our ability to make properly informed decisions is often the first thing to go [17, p.1].

While people struggle with excessive information, many algorithms in the field of artificial intelligence can increase their performance by accessing more information. [19] calls this the "unreasonable effectiveness of data". Perhaps surprisingly, more data often trumps more efficient algorithms. For example, [5, p.3] show how common algorithms in AI can be substantially improved by giving them a lot more data to work with. As much as researchers chase elegant algorithms, finding more data to work with may be time better spent.

Few places is this difference of users and computers more apparent than in *recommender systems*. A recommender system is a technique in user modeling to estimate the relevance of an item to a user. An item can be just about anything, for example documents, websites, movies, events or other users. These systems use data such as search query logs, ratings from similar users, social connections and much more to predict unknown relevance. Recommender systems are especially prolific on the Web. Wherever there is personalized recommendations of news, books, movies, articles, social connections, search results, et cetera, recommender systems are working behind the scenes.

Modern recommendation approaches often embrace the unreasonable effectiveness of data, by combining multiple recommender systems, that each predict relevance in various ways. By considering different aspects of users and items when making predictions, the methods provide quite complex predictions that rely on much evidence. For example, Bell et al. took this approach its logical conclusion in [7] by combining 107 different recommender systems when winning the Netflix movie recommender challenge (see [8]).

While the name "recommender systems" might seem limiting, they are incredibly powerful tools. If we can accurately predict how users will react to items, we will have come a long way towards solving information overload.

Despite their apparent power, recommender systems are often confined to simple tasks like creating small lists of recommended items or computing similar items to the ones being considered. Common examples are lists of recommended items based on the one being viewed, recommending new social connections, or suggesting news articles based on previous reading. Seldom are the full potential of recommender systems reached by creating completely adaptive content systems that work hard to mitigate any signs of information overload.

We posit that traditional recommender systems have an important weakness. There exists an underlying, misplaced subjectivity to relevance prediction. We believe this fundamental weakness hinders the full adoption

of these systems. There is a mismatch between how recommender systems perform predictions, and how each user and item wants these predictions to be made.

Consider this: when an algorithm is selected for use in a recommender system, there is a concious decision of which predictive data pattern to use. Before any user modeling is performed, the researcher or developer selects one or more methods that is thought to best model every user and item in the system. This selection reflects how the researcher or developer assumes how each user and item *should* be modeled. This underlying subjectivity is not desirable. We call this the *latent subjectivity problem* (see [11, p.33]).

Examples are not hard to come by. For instance, while one user might appreciate social influence in their search results, another user might not. While one user might find frequency of communication maps well to relevance, another might not. One user might feel the similarity of movie titles are a good predictor, while another might be more influenced by production year. Some users may favor items rated highly on a global scale, while others are more interested in what users similar to themselves have to say. The same problem exists for items: while one item might best be judged by its content, another might be better described by previous ratings from other users. One item's relevance may be closely tied to when it was created, while other items may be timeless. The exact differences are not important, only that they exist.

Aggregate modeling methods face the same problem of misplaced subjectivity. Aggregation is done on a generalized, global level, where where individual users and items are expected to place the same importance on every modeling method (e.g. [4], [16], [7], [15], [30]). While aggregation is selected to minimize some error over a testing set, the subjective nature remains. The compiled aggregation is a generalization, treating all users the same — hardly a goal of user modeling.

We believe the priority of the algorithms should be implicitly and automatically based on how well they have previously worked for individual users and items. The scope of users and items is simply too great for any one or generalized combination of methods to capture the nuanced nature of relevance prediction.

We propose a novel method that we call *adaptive recommenders*, where the selection of algorithms is made by each user and item. This provides an extra level of abstraction and personalization. The decisions are implicit, and happens in the background, without any extra interaction required. This adaptive selection has an important consequence. If each algorithm is contextually used based on how well it performs in the current context, any possibly useful recommender algorithm suddenly becomes a worthy addition.

This paper attempts to make two contributions. First, we explain the *latent subjectivity problem*, and describe how an extra layer of abstraction can help solve this issue. Second, we develop and test one type of *adaptive recommenders* in an effort to measure the performance of such a system. As far as we know, such adaptive prediction aggregation has not been done before.

2. Theory & Related Work

There are many ways of predicting the relevance of an item to a user. In fact, judging by the number of different approaches, the only limiting factor seems to be the different patterns researchers discover in available data. We will explain many of these systems throughout this paper. See [1], [26], [28] or [10] for a comprehensive exploration of different types of recommenders.

We are interested in current approaches to recommender prediction aggregation, that is, combining multiple predictions made by different algorithms. [4] describes a number of simple approaches. Min, max and sum models combine the individual predictions in some way, or select one or more of the results as the final prediction. Other models use the average, or log-average of the different methods. The linear combination model trains weights for each predictor, and weighs predictions accordingly. At slightly more complex approach is to train a logistic regression model ([4, p3]) over a training set, in an effort to find the combination that gives the lowest possible error. This last method improved on the top-scoring predictor by almost 11%, showing that even fairly simple combinations have merit.

Early approaches in recommender systems experimented with aggregating content-based and collaborative approaches. Content-Based (CB) methods look at item similarities and the previous actions of a single user. On the other hand, Collaborative Filtering (CF) approaches consider previous actions from other users in the system. [16] combined the two approaches in an effort to thwart problems with each method. CF methods have problems rating items for new users, radically different users or when dealing with very sparse data. CB methods do not have the same problems, but are less effective than CF in the long run, as CB does not tap into the knowledge of other users in the system — knowledge that out-performs simple content analysis. In [16], the two types of recommenders were used to create a simple weighted result.

Burke calls this mixing of CF and CB methods *hybrid recommenders*, and describes a number of approaches to combining their results [14, p.4]. In other words, the technique of combining multiple recommenders is nothing new. The goal of this paper is to remove the subjectivity inherent in the selection of the combined methods, as we shall see. See [10] for a more comprehensive look at how our technique compares to existing hybrid recommender systems.

Generally, methods for aggregating predictions in the field of machine learning is called *ensemble methods* (EM) ([18]). While most often used to combine classifiers that classify items with discrete labels, these methods are also used for aggregating numerical values (see the numerical parts of [13]). Approaches include *bootstrap aggregation* (bagging) and *boosting* for selecting proper training and testing sets, and creating a *mixture of experts* for combining the predictors [27, p.27]. See [11] for more background theory and related work.

3. Adaptive Recommenders

Adaptive Recommenders (AR) is a technique for combining many recommender systems in a way that is optimal for the each user and each item. Given that we wish to predict the relevance of an item to a user, using many methods that consider disjoint data patterns, we have two important questions:

1) What rating does each method predict?
2) How accurate will each of these predictions be?

The first generation of recommender systems were only interested in the first question, and used a single method to predict an unknown rating. Modern aggregation techniques look at both questions by combining multiple methods, often by an optimal generalized weighting scheme. For example, each combined recommender system may be weighted in a way that results in the lowest average error across all users and items. In this paper, we wish to make the aggregation *adaptive*, so that the aggregation itself depends on which user and which item we are considering.

Formally, we define adaptive recommenders as *adapting a set of recommender systems with another complementary set of recommender systems* (see Fig. 1). This is then a form of meta-modeling, where one set of modeling methods is adapted by another set of modeling methods. The first set creates standard prediction scores, and answers the first question. The second set predicts how accurate each method will be for the current user and item, answering the second question. The interesting bit is that AR can use recommender systems for both these tasks, as we shall soon see.

A system for adaptive recommenders is specified by a 6-tuple, $\mathrm{AR} = (I, U, R, F, M, A)$. We have sets of $Users$ (U) and $Items$ (I), and a set of $Ratings$ (R). Any user $u \in U$ can produce a rating $r \in R$ of an item $i \in I$. $r_{u,i}$ then refers to the rating of item i by user u. As mentioned, items can be just about anything, for example documents, websites, movies, events, or indeed, other users. The ratings can be explicitly provided by users, for example by rating movies, or they can be mined from existing data, for example by using search query logs (e.g. [23], [33], [31], [32]).

We use the term *rating* loosely — equivalent terms include *relevance*, *utility*, *score* or *connection strength*. In other words, this is a measure of what a user thinks of an item in the current domain language. However, since *rating* matches the data in our experiment, we will be using this term.

The $Framework$ (F) variable specifies how the data is represented. The two canonical ways of representing users, items and ratings are graphs and matrices (see [25]). We shall use a matrix, where the first dimension corresponds to users, the second to items, and each populated cell is an explicit or implicit rating:

$$R_{u,i} = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,i} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,i} \\ \vdots & \vdots & \ddots & \vdots \\ r_{u,1} & r_{u,2} & \cdots & r_{u,i} \end{pmatrix}. \quad (1)$$

As we wish to leverage disjoint data patterns, we have a set of modeling $Methods$, with their own ways of estimating unknown ratings. Every model $m \in M$ is used to compute independent and hopefully complimentary predictions. In our case, these methods are recommender systems.

As demonstrated in [1], [26], [28], [29], there are many different recommendation algorithms, that consider different aspects in the data, for example users, items and ratings, as well as sources such as intra-user connections in social networks or intra-item connections in information retrieval systems. Examples of such recommender systems include Slope One predictions [22], SVD factorization (e.g. [9, p.5], [34] and [7]) and Nearest Neighbor weighted predictions [29, p.11]. These methods predict unknown connections between users and items based on some pattern in the data, for example user profile similarity, rating correlations or social connections. As previously explained, to achieve the best possible combined result, we wish to use methods that look at disjoint patterns, i.e. complementary predictive parts of the data (as described by [6]).

The $Adapters$ (A) part of our 6-tuple refers to the second level of user modeling methods. In traditional prediction aggregation this is a simple linear function for combining the different predictions, for example by pre-computing a set of weights, one for each method. As found by [7, p.6] the accuracy of the combined predictor is more dependent on the ability of the various predictors to expose different aspects of the data, than on the individual accuracy of each predictor. Multiple prediction results are normally combined into a final singular result, based on a generalized combination found by minimizing some error across all users.

With adaptive recommenders, the $Adapters$ are themselves user modeling methods. Instead of modeling users, we wish to model the behavior of recommender systems. More specifically, we wish to model the *accuracy* of the recommender systems. Methods in this second layer are used to predict how accurate each of their

corresponding basic recommenders will be. It is these methods that will allow us to do adaptive aggregation based on the current user and item. In other words, we have two distinct layers of user modeling (see Fig. 1):

3.1. Adaptive Aggregation

To perform adaptive aggregation, we need the $Adapters$ to be actual recommender systems. The simplest generalized way of prediction aggregation is to take the average of all predictions made by the different methods (e.g. [4, p.3]). Many aggregators attempt to weigh each method differently (e.g. [16]):

$$\hat{r}_{u,i} = \sum_{m \in M} w_m \times p(m, u, i). \tag{2}$$

Here, w_m is the weight applied to modeling method m. These weights fall in the range $[0, 1]$ and sum up to 1. The weights can be estimated through different machine learning methods. However, as discussed in Section 1, this is still a generalized result, averaged across every prediction. The system assumes that the best average result is the best result for each individual user and item. Even with method-specific weights we are still hindered by the latent subjectivity problem.

In order to leverage as many patterns as possible while sidestepping the latent subjectivity, we need *adaptive weights* that are computed specifically for combinations of users and items. If we wish each weight to be combination-specific, then pre-computing weights for every method becomes unfeasible. We would have to compute a weight for every method for all possible ratings. In other words, these adaptive weights also have to be estimated, just as the ratings themselves:

$$\hat{r}_{u,i} = \sum_{m \in M} p_w(m, u, i) \times p_r(m, u, i). \tag{3}$$

Here, $p_w(m, u, i)$ is the predicted optimal weight for method m when applied to user u and item i. Adaptive recommenders is one way to estimating these weights, i.e. one way to implement p_w.

We wish to use standard recommender systems for predicting optimal adaptive weights. To do this, we need to create a matrix (or graph) that stores known values of how accurate some of the rating predictions will be.

This can be done my modeling the errors of the methods. By modeling the errors with standard recommender systems, we can in turn predict errors for untested combinations. If we predict the error of a recommender system for a user and an item, we have also predicted its accuracy. To achieve this, we create an *error matrix*:

$$E_{u,i} = \begin{pmatrix} e_{1,1} & e_{1,2} & \cdots & e_{1,i} \\ e_{2,1} & e_{2,2} & \cdots & e_{2,i} \\ \vdots & \vdots & \ddots & \vdots \\ e_{u,1} & e_{u,2} & \cdots & e_{u,i} \end{pmatrix} \tag{4}$$

Creating error matrices for the modeling methods is done by splitting the ratings data in two, the first set can be used for the actual training, and the second can be used to populate the error matrices.

Every standard modeling method produces an error matrix where some cells have values. A value in this matrix corresponds to the prediction error for a combination of a user and an item. Each modeling method is trained with a part of the ratings data. The error matrix is populated from the rest of the data, by computing the error of all known ratings the method was not trained on:

$$\forall (u, i, r) \in (d_e - d_m) : E(m)_{u,i} = |r - p(m, u, i)|. \tag{5}$$

Here, D is the current dataset, and d_m and d_e are subsets of D. m is a modeling method trained with the subset d_m. To populate the error matrix for this method, we take every rating which have not been used to train the method and calculate the error of the method on this combination. The result is a sparse error matrix that can be used to predict unknown errors. Notice the similarity of this matrix and the previously introduced ratings matrix. This similarity is what will allow standard recommender systems to perform adaptive aggregation.

3.2. Modeling Phase

Whenever we wish to train a new modeling method, we apply the following algorithm:

1) Split the ratings data into two sets for training and error estimation.
2) Train the modeling method in its specific way with the first training set.
3) Use the error estimation data set to create the error matrix.
4) Train an error model based on the error matrix.

Constructing the subsets of the available data is a common task in ensemble learning [27, p.7]. We use *bootstrap aggregation*, also known as *bagging* (introduced by [13]). Originally, bagging is used by ensemble learning classification methods, where multiple classifiers are trained by uniformly sampling a subset of the available training data. Each model is then trained on one of these subsets, and the models are aggregated by averaging their individual predictions.

Bagging suits our needs for a few reasons. First, the method helps create disjoint predictors, since each predictor is only trained (or specialized for) a subset of

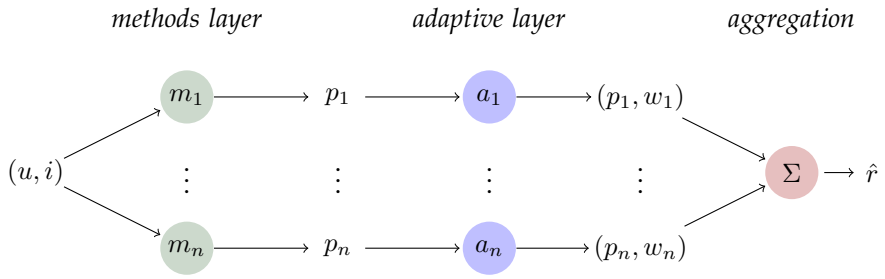

Fig. 1: Layers of recommenders: The method layer consists of ordinary modeling methods that predict the rating between a user and an item. This produces a set of predicted ratings (p). The adaptive layer estimates how well each modeling method will perform for the current user and item, and weighs the predictions accordingly. This produces a set of predictions and weights $[(p, w)]$. The aggregation weighs the predictions into a final score $\hat{r}$.

the available data. Second, it allows us to easily train the underlying modeling methods without any complex partitioning of the data.

The reason for disjoint predictors is explained by [6, p.6]: *"We strongly believe that the success of an ensemble approach depends on the ability of its various predictors to expose different, complementing aspects of the data. Experience shows that this is very different from optimizing the accuracy of each individual predictor. Quite frequently we have found that the more accurate predictors are less useful within the full blend."*. In other words, we need predictors that look at different aspects of the data, which can be achieved through bagging, by training each predictor on a random subset of the available data. As shown by [6, p.6], the performance of the combined recommenders is often not dependent on their individual accuracy, but rather by their skill at predicting ratings for different parts of the entire dataset.

The error models are trained using standard recommender systems. After all, the expected input and output are the same. We have two dimensions, with a sparse set of known connections, and wish to predict unknown connections from this data. The result is a set of modeling methods that can predict the error of a recommender system when it is used on a particular user and item combination.

What will happen when we train a recommender system with the error matrix? First of all, the errors will be on the same scale as the initial ratings. Second, just as the ratings matrix will include noise (ratings that do not contribute to any underlying pattern), this will be true for the error matrix as well.

For example, one method might have a large error for a particular user and item combination, yet still work well for both these elements. Luckily, this is just the kind of noise recommender systems are good at pruning away. What we are interested in are situations where a method has stable and significant errors for many ratings from a user, or many ratings of an item. In this case, there is a pattern where this method does not work well for the element in question. This is exactly the kind of pattern recommender systems are good at identifying.

In other words, the same capabilities that makes recommender systems work well on the ratings matrix, will also make them work well on the error matrix. The properties we need for predicting ratings are the same as those needed to predict accuracy. However, the weaknesses of the ratings matrix are also carried over to the error matrix. As with the ratings matrix, the error matrix is a sparse matrix with weaknesses such as biased users and items, concept drift and a varying amount of noise. We face the same classic problems such as the cold start and average user problems. The adaptive recommenders should be able to counter these weaknesses, as they are they same problems they face when computing predictions based on the ratings matrix. There may exist unique challenges when using an error matrix, but as our scope is limited, we will not examine these possible weaknesses in detail.

Of course, some recommender systems will work better than others for the adaptive layer. Most often we are seeking global patterns in the data. We are looking for groups of users or items (or both) that suite some recommenders especially well, or that some recommenders will not work for. SVD-based recommenders are one type of recommender systems that can be used for this purpose. By reducing the method-error space into an error category space, we can identify how well a set of groups suite each available method. We will return to this when testing the performance of our approach in Section 5.

3.3. Prediction Phase

When we have an error model for every modeling method, we can use these errors to estimate their contextual weights. Whenever we wish to create an adaptive aggregate prediction, we apply the following algorithm:

$$\hat{r}_{u,i} = \sum_{(m_e,m_r)\in M} (1 - \frac{p(m_e,u,i)}{error(u,i)}) \times p(m_r,u,i) \quad \text{where} \quad error(u,i) = \sum_{m_e \in M} p(m_e,u,i) \tag{6}$$

1) Collect predictions from every modeling method for (u,i).
2) Collect estimated errors for every method for (u,i).
3) Compute weights for each method based on their relative predicted errors.
4) Sum the weighted predictions to get the adaptively predicted rating.

The next section will explain these steps in detail. We can now express the prediction phase of adaptive recommenders as an equation. Each rating/relevance prediction is weighted by its predicted accuracy, conditioned on the current user and item, as seen in Eq. 6.

In Eq. 6, each recommender method has two corresponding models: m_r is the ratings model, used to predict ratings, and m_e is the error model, used to predict errors. $p(m,u,i)$ is the prediction of the model m (a recommender system) for the relevance of item i to user u. Each method is weighted by its predicted accuracy. The weights are computed by taking the opposite of the predicted errors of the methods. The errors are normalized across users and items by $errors(u,i)$, which is the sum of the errors of each method for the current combination. This gives us weights in the range $[0,1]$ ensuring final rating predictions on the same scale as that returned by the basic recommenders.

Notice that the *only* difference between m_e and m_r is how they are created. m_r is trained with the standard ratings matrix, and m_e is trained using the error matrix. This means we can use *any* standard recommender system to perform adaptive aggregation. Hence, the name *adaptive recommenders*. A set of secondary recommenders is used to adapt a set of standard recommenders to individual users and items.

It is also important to note that the types of recommenders used for the adaptive layer are independent of the basic recommenders. The adaptive recommender need only predicted ratings from the basic recommenders, and does not care which algorithm it employs. When making predictions, the calculations in the methods layer and adaptive layer are independent, as both use pre-computed models. The method layer use the ratings matrix, or their own models created during training, while the adaptive layers use the error matrices for each basic method.

The result of this is a system that does not only aggregate a number of predictions for each unknown combination of users and items, but that also combines these methods based on how accurate each prediction is likely to be.

4. Experimental Setup

We chose the MovieLens dataset[1] to test the performance of our system. This dataset is often used to test the performance of recommender systems (as described in [3, p.9], [22, p.4], [1, p.1] and [20, p.2]). It consists of a set of users, a set of movies, and a set of movie ratings from users, on the scale 1 through 5. The MovieLens collection comes in multiple formats and sizes, and we chose a subset of the entire MovieLens collection, with 100,000 ratings from 943 users on 1,682 movies.

The reason for using a subset was twofold. First, using the entire set did not result in significant differences in performance (see [11, p.63] for an experiment using the entire MovieLens dataset). Second, while we maintain that our approach is scalable to real-world systems, measuring performance for our experiments requires multiple computations of the different methods *and* computing predicted ratings for every user/item permutation. In other words, the computational complexity of our experiments is far higher than that of a real recommender system using this approach. We shall discuss this further in Section 6.

The dataset was split into disjoint sets ($D = \{d_1, ..., d_5\}$) to perform five-fold cross-validation. Each of these five disjoint sets were then randomly split into two sets, also disjoint, for training and testing each method.

To evaluate how our model performs during prediction aggregation, we need a measure for computing the total error across a large number of predictions. The canonical measure for estimating the error of a recommender system is the *Root mean squared error* (RMSE) measure (for example in [20, p.17], [1, p.13] and [7, p.6]). We use this measure to estimate the performance of our adaptive prediction aggregation algorithms. The RMSE of a set of estimations $\hat{R}$, compared the correct rating values R, is defined as

$$\text{RMSE}(\hat{R}, R) = \sqrt{\frac{\sum_{i=1}^{n}(\hat{R}_i - R_i)^2}{n}}, \tag{7}$$

where n is the total number of predictions. The RMSE combines a set of errors into one single combined error. A beneficial feature of the RMSE is that the resulting error will be on the same scale as the estimations. For example, if we are predicting values on the scale $1-5$, the computed error will be on this scale as well. In this case, an error of 1 would then say that we are on average 1 point away from the true ratings on our $1-5$ scale.

[1] See http://www.grouplens.org/node/73 — accessed 10.05.2011

TABLE I: adaptive modeling methods: A short overview of the recommender methods used in our experiment. The first column refers to the type of recommender, *S* for standard systems and *A* for aggregation methods. Every recommender is used in every experiment.

	method	algorithm	description
S	svd1	SVD	ALSWR factorizer, 10 features.
S	svd2	SVD	ALSWR factorizer, 20 features.
S	svd3	SVD	EM factorizer, 10 features.
S	svd4	SVD	EM factorizer, 20 features.
S	slope_one	Slope One	Rating delta computations.
S	item_avg	Baseline	Item averages.
S	baseline	Baseline	User and item averages.
S	cosine	Cosine sim.	From similar items.
S	knn	Pearson Corr.	From similar users.
A	median	Aggregation	Median aggregate rating.
A	average	Aggregation	Average aggregate rating.
A	adaptive	Adaptive agg.	*The approach of Sec 3.*

In addition to the dataset, we need a collection of recommender systems. Standard recommenders will be used for both the basic predictions, and the accuracy estimations, as described in Section 3.

Naturally, we need a large number of different recommenders, preferably ones that consider disjoint patterns in the data. Table I gives a short overview of the recommender systems we shall employ. Each experiment will use every recommender in this table. The following section gives a short introduction to these systems.

4.1. Basic Recommenders

As seen in Table I, we have two types of recommenders. First, we have the basic recommenders, denoted by *S* in the table. Each recommender system looks at the data in a different ways to predict ratings. This wide range of recommenders was chosen for just this reason. As previously explained, the performance of aggregate recommenders are more dependent on the dissimilarity of the basic recommenders than their individual performance.

Let us briefly explain how the basic recommenders work. In recommender systems, *Singular Value Decomposition* (SVD) is used to compress the ratings space into what is sometimes called a *taste space*, where users are mapped to higher-level *taste* categories (e.g. [2, p.5], [12, p.4] or [24, p.2]). The factorizers refers to algorithms used to factorize the ratings matrix (e.g. the ALSWR factorizer [35]).

The Slope One, item_average and baseline algorithms look at average ratings for items and from users, and use these to predict ratings. Note that these are not simple, generalized averages, but averages based on solving a least squares equation for creating user and item *baseline ratings*, not average ratings. This is based on the technique of [21, p.2]. This computes predicted average ratings that are far superior to simple average ratings. As we shall see, these algorithms perform well compared to other approaches (see also [11, p.15]).

The cosine similarity algorithm looks for items that are rated similarly by the same users, and infers item similarity from this measure. New ratings are then predicted by taking known ratings of other items, weighted by their item's similarity to the new item. This is based on the same method used in the vector space model, from the field of information retrieval.

The KNN algorithm employs yet another approach. This algorithm, similar in strategy to the cosine similarity algorithm, looks for users with similar rating patterns. The similarity is measured with the Pearson Correlation Coefficient. Predictions are created by collecting ratings from similar users of the item in question, weighted by their respective similarity. See [1], [26] or [28] for a more comprehensive exploration of different types of recommenders.

4.2. Aggregate Recommenders

The second type of recommenders are the aggregation methods, that combine the result of each of the basic recommender systems The first two of these methods are simple aggregation approaches. The median aggregation method choses the median value of the predictions produced by the standard recommenders. Similarly, the average aggregation method takes the mean of the standard predictions. While not complex in nature, these methods will help us see how our method compares to simple, traditional aggregation techniques.

The last entry in Table I refers to our technique. This is the recommender outlined by Section 3, that create secondary accuracy estimating recommender systems, in order to adaptively weigh the basic recommenders. All the aggregation approaches, including our technique, employ every basic recommender system described so far.

As explained in Section 3, any basic recommender system can be used for the adaptive method. The only difference is how this method is trained: while the basic methods are trained using the ratings matrix, the adaptive methods are trained using the error model. In other words, we have as many possibilities for choosing the adaptive recommenders as the basic recommenders.

For our experiment, we went with SVD recommenders for the adaptive models. That is, every basic recommender method gets a secondary accuracy predicting recommender, which in this case is a standard SVD recommender. The SVD recommender is a natural choice in this case, since we wish to uncover latent patterns of accuracy for each model. Examples of these patterns include groups of items or users a specific recommender works well for.

It is important to note that the same configuration of recommenders was used for all three experiments. Neither the basic nor the aggregate or adaptive recommenders were heavily tailored to the datasets. To be sure, higher performance could probably have been achieved by tailoring the recommenders to the available data. However, as our goal is to compare our finite set of methods, we are currently only interested in how they perform compared to each other.

As with the basic recommenders, the same SVD recommender configuration was used for the adaptive layer in the experiment. We chose to use an EM-factorizer to perform the actual decomposition, consisting of 20 features. The decomposition was performed by 20 iterations. We chose the EM algorithm instead of the previously mentioned ALSWR approach, as this was shown to perform better on the MovieLens dataset (as seen in Table II). We will discuss our choice further in Section 6.

5. Results

Table II gives the resulting RMSE values when our algorithm is used with the MovieLens dataset. A cell corresponds to the RMSE values for a dataset and a recommender system. The bottom entry in this table refers to our adaptive recommenders method. As seen in this table, our approach outperforms both the standard recommenders and the aggregation approaches.

By outperform we mean that our model should have a lower mean RMSE score than the other singular methods. As we can see in Table II, our approach outperforms both the standard recommenders and simple generalized aggregation approaches. While we can not generalize too much on this basis, the fact that this dataset is a common testing ground for recommender systems, and that RMSE is the de facto measure for determining performance, we have grounds for some confidence in these results.

Statistics for the experiments are given in the last part of Table II. The statistical values are the minimum, maximum and mean values for the methods. We also include the standard deviation (σ) for the methods, across each collection of subsets. This table confirms the results from the full results table: Our adaptive recommenders approach improves the mean performance of our system. The mean performance and the standard deviation are shown in Fig. 2.

It might seem counter-intuitive that the ”baseline” and ”item_avg” methods outperform the SVD-, KNN- and PCC-based methods. However, as previously mentioned, these methods are not as simple as their names might suggest. Computing these ratings are as complex as the other methods, as averages are computed as a least squares problem in a way that removes noise and tries to find baselines that best fit the available data, while taking the problems of overfitting and user/item bias into account. In addition, these averages are combined per user and item, resulting in quite capable rating estimations (see [21, p.2]).

However, as their rivaling methods are most often found to still outperform these baseline methods, their performance is a curious result. This is most likely related to the fact that none of the methods were particularly tweaked and optimized for the data at hand. If more time had been spent optimizing the SVD-based recommenders, their performance might have been different. As this optimization is outside the scope of our paper, we will not attempt to explain this result. In addition, as previously mentioned, the performance of individual recommenders are not indicative of their contribution to an aggregation [6, p.6]. Additional optimization of the more complex methods could possibly be

TABLE II: (a) Results from our experiment. Each cell contains an RMSE value. The first table gives errors for subsets of the data (d_x). Lower values indicate better results. Bold values indicate the best result in a column. S refers to singular methods, and A to aggregation methods. (b) Statistics for the methods. The *mean* values are macro-averaged measures for each method. σ refers to the standard deviation of each method across the subsets. Δ shows the increase in mean performance of each method compared to the next best method.

(a) RMSE values for the five disjoint subsets:

	method	d_1	d_2	d_3	d_4	d_5
S	svd1	1.2389	1.1260	1.1327	1.1045	1.1184
S	svd2	1.2630	1.1416	1.1260	1.1458	1.1260
S	svd3	1.0061	0.9825	0.9830	0.9815	0.9797
S	svd4	1.0040	0.9830	0.9849	0.9850	0.9798
S	slope_one	1.1919	1.0540	1.0476	1.0454	1.0393
S	item_avg	1.0713	0.9692	0.9662	0.9683	0.9725
S	baseline	1.0698	0.9557	0.9527	0.9415	0.9492
S	cosine	1.1101	0.9463	0.9412	0.9413	0.9382
S	knn	1.4850	1.1435	1.1872	1.2156	1.2022
A	median	0.9869	0.8886	0.8857	0.8857	0.8855
A	average	0.9900	0.8536	0.8525	0.8525	0.8519
A	adaptive	**0.9324**	**0.8015**	**0.7993**	**0.8238**	**0.8192**

(b) Statistics for the methods:

	method	min	max	mean	σ	Δ
S	knn	1.1435	1.4850	1.2467	0.3487	-
S	svd2	1.1260	1.2630	1.1605	0.2277	6.9%
S	svd1	1.1045	1.2389	1.1441	0.2197	1.4%
S	slope_one	1.0393	1.1919	1.0756	0.2415	5.9%
S	item_avg	0.9662	1.0713	0.9895	0.2023	8.0%
S	svd4	0.9798	1.0040	0.9873	**0.0924**	2.2%
S	svd3	0.9797	1.0061	0.9865	0.0991	0.1%
S	cosine	0.9382	1.1101	0.9754	0.2595	1.1%
S	baseline	0.9415	1.0698	0.9738	0.2196	1.6%
A	median	0.8855	0.9865	0.9065	0.2005	6.9%
A	average	0.8519	0.9900	0.8801	0.2344	2.9%
A	adaptive	**0.7993**	**0.9324**	**0.8352**	0.2225	5.1%

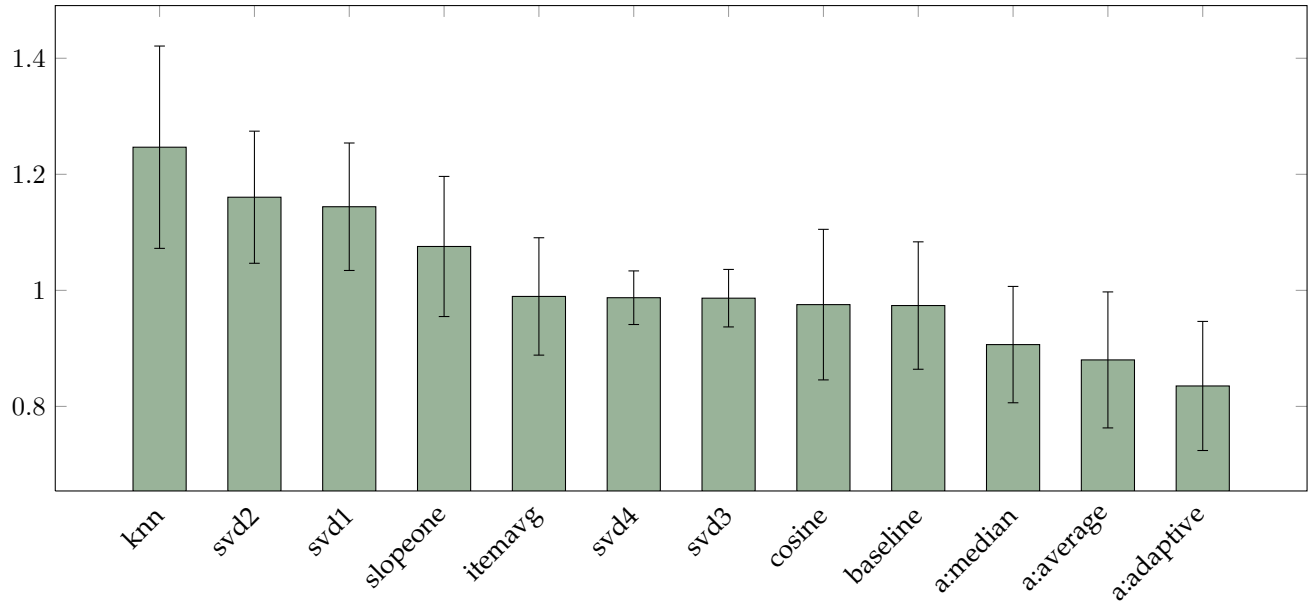

Fig. 2: Average RMSE plot: This plot shows the average RMSE for the basic and aggregation methods (denoted "a:"). The actual numbers are given in Table II. The error bars indicate the standard in our experiment. Note the scale on the y-axis — the errors are not as pronounced as they might seem.

counter-productive to our goal of comparing traditional aggregators to adaptive aggregation.

Let us now take a look at the standard deviation measures from the different methods. As seen in Fig. 2, most of the methods, including the adaptive models, exhibit quite a lot of variation in their results. If these variations occurred as a result of unstable predictions on the same dataset, this would be a substantial problem, resulting in unreliable predictions. However, the standard deviation is mostly caused by the differing performance across the varying datasets. As we see, the performance of the aggregation methods, as well as the best performing standard recommender, follow each other closely. At the same time, performance varies across the different datasets, which results in high values for σ.

There are two important limitations. First, our approach is much more complex than those we test it against. The question whether the methods performance is worth its extra complexity becomes important. Second, the generalized aggregation methods we chose to test our method with are simpler than the most powerful modern approaches to aggregate recommenders. This fundamentally limits how confident we can be that our system is indeed an improvement. We shall discuss this further in Section 6.

In [11], we perform more experiments with adaptive recommenders, including one where this approach is used to provide personalized search results.

6. Discussion

We have made two main contributions with this paper: (1) we have described the latent subjectivity problem and (2) we have developed the technique of adaptive recommenders.

The latent subjectivity problem is one we think hinders standard recommender systems reaching their full potential. As far as we know, this problem has not been described in the context of recommender systems. The main choice for any such system is how to predict unknown ratings. To do this, a pattern in the available ratings data must be leveraged. These patterns are plentiful, and their individual performance depends on the users and items of the system. Modern aggregation recommenders utilize many patterns, but on a generalized level, where every user and item is treated the same. This underlying subjectivity leads to a mismatch between the notions of whoever developed the systems, and the users and items of the service.

Averaged or generalized weighted approaches will always choose the combination that performs best *on average*, with little concern to the uniqueness of items (and users). In other words, this is a comprehensive problem that may be discovered amongst many machine learning techniques.

Adaptive recommenders is our attempt to solve the latent subjectivity problem. As far as we know, this type of adaptive prediction aggregation has not been done before. Section 5 showed that an aggregation that combines predictions based on estimated accuracy can outperform both standard recommenders and simple

aggregation approaches. Our technique is strengthened by the fact that standard recommender algorithms are used for the accuracy estimations. This is the core insight of this paper. *By creating error models for the recommenders, we can predict their accuracy for each user/item combination. These predictions can then be used to weigh the combined algorithms accordingly.*

We believe there are greater opportunities in systems where there are even more diverging patterns to be leveraged. The prime examples of this are systems that may or may not use social connections between users, and systems that predict the relevance of widely varying items.

6.1. Limitations

There are some important general limitations to our research related to (1) the complexity of our method, (2) our choice of data and evaluation metrics, (3) the general usefulness of this approach, and (4) common issues with recommender systems.

(1) Complexity: As our approach is more complicated than standard recommenders, it is worth questioning whether or not the performance gains are worth the added complexity. This depends on the basic recommenders that are to be combined. If the system is made up by many different recommenders that each user might place varying importance on, and that may have varying success with each item, adaptive recommenders may provide gains in accuracy.

On the other hand, if the recommenders are simple in nature, and look at similar patterns in the data, generalized aggregation methods might be more applicable. Clearly, the performance gains in our experiments are not substantial enough to declare anything without reservation.

The computational cost of our proposed system would be about twice the cost of the systems we compare ourself to. Modern recommenders already employ multiple methods, and our system would add another layer of recommenders, doubling the number of recommender algorithms and amount of required computation. However, the scalability of using multiple recommenders is not a new problem, and solutions do exist. For instance, many of the methods, including our SVD-based aggregation recommenders allow for offline computation of the matrix factors, allowing for scalable predictions.

While we believe this technique has potential, without real-world success stories, it is hard to suggest that our method is particularly better than a simple standard recommender.

(2) Evaluation: we chose traditional datasets and evaluation metrics to validate the adaptive recommenders technique. While our initial results are promising, it is important to stress that this is only one test on one dataset. Considering the vast scope of applicable data, and the number of ways these results many be evaluated, the results must be seen for what they are, that is, initial and preliminary explorations of a new technique that has yet to be proved useful in the real world.

More specifically, we would like to perform the same experiments on the NetFlix dataset, to be able to compare our performance to [7] (and other modern approaches that utilize this dataset). However, this dataset is no longer available from NetFlix and redistribution seems to be prohibited.

(3) Usefulness: When considering the additional complexity of our approach, a natural response is whether or not current approaches to recommender systems are good enough. We do not think so. Information overload is such a nuanced problem that the only solution lies in intelligent, adaptive systems. However, as most current recommender systems perform quite simple tasks, they may be more than good enough for their purpose.

There will always be a trade-off, between complexity and required accuracy. As in many other cases, the systems described in this paper have their use cases. In the end, the requirements of the system in question must decide which method best suit their needs.

(4) Common issues: The topic of recommenders and adaptive systems in general raise a number of questions which is outside the scope of this paper. For example, user privacy is a big issue. Whenever we have a system that tries to learn the tastes, habits and traits of its users, how each user will react to this must be considered. This is often a trade-off between adaptability and transparency. The most adaptive systems will not always be able to explain to the users what is going on and what it knows about each person, especially when dealing with emergent behavior based on numerical user models.

In addition to the general limitation, our experiment carries a few drawbacks. Our method was only tested against a limited number of standard recommenders. The key word is standard: these recommenders were not heavily customized to fit the available data. As in all machine learning, achieving relatively good performance is quite simple. Any improvements above this standard requires deep domain knowledge, and methods customized to the problem at hand. In an actual system, the adaptive recommenders should be tested against carefully selected standard recommenders, optimized for the current domain. Other future work includes looking at how the error matrix is influenced by common recommender system problems, such as the cold start and average user problems.

Similarly, our method was only tested against simple aggregators. Many more complex aggregations are possible, for example by solving the problem of finding optimal generalized weights for each method.

7. Conclusion

This paper has described the *latent subjectivity problem* and why we believe this is a fundamental issue with many recommender systems. *Adaptive Recommenders* is our attempt at a solution. By adaptively choosing which algorithms are employed based on the current context, truly adaptive systems can be created. Our initial results show that this approach can outperform standard recommenders and simple aggregation approaches. While our tests show the basic viability of our approach, more testing against complex aggregation functions is still required.

Acknowledgments

This paper is a short version of my Master Thesis in Artificial Intelligence [11]. This thesis can be accessed online[2], and provides more background theory, experiments and results.

I would like to thank my supervisor, assistant professor Asbjørn Thomassen, for valuable guidance and feedback throughout the process. In addition, thanks are in order for my fellow students Kim Joar Bekkelund and Kjetil Valle, who helped me formulate my thoughts and provided feedback on the work represented by this paper.

References

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, June 2005.

[2] J. Ahn and T. Hong. Collaborative filtering for recommender systems: a scalability perspective. *International Journal of Electronic Business*, 2(1):77–92, 2004.

[3] M. Alshamri and K. Bharadwaj. Fuzzy-genetic approach to recommender systems based on a novel hybrid user model. *Expert Systems with Applications*, 35(3):1386–1399, Oct. 2008.

[4] J. a. Aslam and M. Montague. Models for metasearch. *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '01*, pages 276–284, 2001.

[5] M. Banko and E. Brill. Mitigating the paucity-of-data problem: Exploring the effect of training corpus size on classifier performance for natural language processing. In *Proceedings of the first international conference on Human language technology research*, pages 1–5. Association for Computational Linguistics, 2001.

[6] R. Bell, Y. Koren, and C. Volinsky. Modeling relationships at multiple scales to improve accuracy of large recommender systems. *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '07*, page 95, 2007.

[7] R. Bell, Y. Koren, and C. Volinsky. The BellKor solution to the Netflix prize. *KorBell Team's Report to Netflix*, 2007.

[8] J. Bennett and S. Lanning. The netflix prize. In *Proceedings of KDD Cup and Workshop*, volume 2007, page 8. Citeseer, Oct. 2007.

[9] D. Billsus and M. Pazzani. Learning collaborative information filters. In *Proceedings of the Fifteenth International Conference on Machine Learning*, volume 54, page 48, 1998.

[10] O. Bjørkøy. User Modeling on The Web: An Exploratory Review, 2010.

[11] O. Bjørkøy. Adaptive Aggregation of Recommender Systems. Technical report, NTNU, Trondheim, 2011.

[12] M. Brand. Fast online SVD revisions for lightweight recommender systems. *SIAM International Conference on Data Mining*, 2003.

[13] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[14] R. Burke. Hybrid web recommender systems. In *The adaptive web*, pages 377–408. Springer-Verlag, 2007.

[15] D. Carmel, N. Zwerdling, I. Guy, S. Ofek-Koifman, N. Har'el, I. Ronen, E. Uziel, S. Yogev, and S. Chernov. Personalized social search based on the user's social network. *Proceeding of the 18th ACM conference on Information and knowledge management - CIKM '09*, page 1227, 2009.

[16] M. Claypool, A. Gokhale, T. Miranda, P. Murnikov, D. Netes, and M. Sartin. Combining Content-based and collaborative filters in an online newspaper. In *Proceedings of ACM SIGIR Workshop on Recommender Systems*, number June, pages 60–64. Citeseer, 1999.

[17] T. Davenport and J. Beck. *The attention economy: Understanding the new currency of business*. Harvard Business Press, 2001.

[18] T. Dietterich. Ensemble methods in machine learning. *Multiple classifier systems*, pages 1–15, 2000.

[19] A. Halevy and P. Norvig. The unreasonable effectiveness of data. *Intelligent Systems, IEEE*, 24(2):8–12, Mar. 2009.

[20] J. Herlocker, J. Konstan, L. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, Jan. 2004.

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

[22] D. Lemire and A. Maclachlan. Slope one predictors for online rating-based collaborative filtering. *Society for Industrial Mathematics*, 2005.

[23] F. Liu, C. Yu, and W. Meng. Personalized web search by mapping user queries to categories. *Proceedings of the eleventh international conference on Information and knowledge management - CIKM '02*, page 558, 2002.

[24] H. Liu, P. Maes, and G. Davenport. Unraveling the taste fabric of social networks. *International Journal on Semantic Web and Information Systems*, 2(1):42–71, 2006.

[25] B. Mirza and B. Keller. Studying recommendation algorithms by graph analysis. *Journal of Intelligent Information*, 2003.

[26] M. Pazzani and D. Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer-Verlag, 2007.

[27] R. Polikar. Ensemble based systems in decision making. *Circuits and Systems Magazine, IEEE*, 6(3):21–45, 2006.

[28] J. Schafer, D. Frankowski, J. Herlocker, and S. Sen. Collaborative filtering recommender systems. *The adaptive web*, pages 291–324, 2007.

[29] T. Segaran. *Programming collective intelligence*. O'Reilly Books, 1st edition, 2007.

[30] B. Sergey and P. Lawrence. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[31] X. Shen, B. Tan, and C. Zhai. Implicit user modeling for personalized search. *Proceedings of the 14th ACM international conference on Information and knowledge management - CIKM '05*, page 824, 2005.

[32] M. Speretta and S. Gauch. Personalized Search Based on User Search Histories. *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)*, pages 622–628, 2000.

[33] K. Sugiyama, K. Hatano, and M. Yoshikawa. Adaptive web search based on user profile constructed without any effort from users. *Proceedings of the 13th conference on World Wide Web - WWW '04*, page 675, 2004.

[34] J. Sun, H. Zeng, H. Liu, and Y. Lu. CubeSVD: a novel approach to personalized Web search. *on World Wide Web*, pages 382–390, 2005.

[35] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-Scale Parallel Collaborative Filtering for the Netflix Prize. In *Algorithmic aspects in information and management: 4th international conference, AAIM 2008, Shanghai, China, June 23-25, 2008. proceedings*, volume 5034, page 337. Springer-Verlag New York Inc, 2008.

[2] See github.com/olav/thesis/raw/master/thesis/dist/thesis.pdf.