SCALE AND TRANSLATION INVARIANT COLLABORATIVE FILTERING SYSTEMS

DANIEL LEMIRE

ABSTRACT. Collaborative filtering systems are prediction algorithms over sparse data sets of user preferences. We modify a wide range of state-of-the-art collaborative filtering systems to make them scale and translation invariant and generally improve their accuracy without increasing their computational cost. Using the EachMovie and the Jester data sets, we show that learning-free constant time scale and translation invariant schemes outperforms other learning-free constant time schemes by at least 3% and perform as well as expensive memory-based schemes (within 4%). Over the Jester data set, we show that a scale and translation invariant Eigentaste algorithm outperforms Eigentaste 2.0 by 20%. These results suggest that scale and translation invariance is a desirable property.

1. INTRODUCTION

To be competitive, businesses need to help clients find quickly and accurately interesting products. Designing software for this task becomes important as on-line shopping often does away with salespersons and offers a limited view of the products to the prospective clients. Fortunately, businesses are often gathering large amounts of data about their clients which makes automated recommendation systems possible. In a wider context, one of the most valuable characteristic of the modern web is the ability to search through large amounts of dynamic data and any process that can support these searches is valuable to the users.

Collaborative filtering systems are recommender systems where the recommendations are based on a database of user ratings as opposed to content-based recommender systems which are based on the characteristics of the objects to recommend. The basic principle behind collaborative filtering is that clients must first share some information about themselves by rating some of the products or features they know, so that, in turn, they can get accurate recommendations. Content-based recommender systems tend to work well with objects where the content can be processed with some convenience such as text [1, 13]. With other types of objects such as movies or books, it is not always easy to access the content on-line, and even if possible, automated content processing is likely to be inaccurate. Also, content-based filtering is sometimes difficult as the user may simply not have enough information about the product or service required. Someone surfing on a e-commerce web site might not always have a specific request and the burden is on the web site to provide an interesting recommendation. In such cases and if we can get some ratings from the users either explicitly or implicitly, we may prefer collaborative filtering systems. In other cases where content-based filtering is efficient, collaborative filtering may serve to help sort results.

However, one of the challenges we face is that most users rate only few objects and thus, we have to deal with sparse data [6]. In many information retrieval tasks, the software is faced with large sets of accurate data and specific queries that must be matched. On the other hand, collaborative filtering has to deal with a severe lack of information and the information available is both imprecise and inaccurate. Thus, collaborative filtering is a prediction rather than a search problem.

From an algorithmic point of view, it is convenient to classify collaborative filtering algorithms in three classes depending on their query and update costs: learning-free, memory-based and model-based. Obviously, there might be many types of operations that could be described as an update or a query, but we focus our attention on adding a user and its ratings to a database (update) or asking for a prediction of all ratings for a given user (query). We say that an operation whose complexity is independent of the number of users offers constant-time performance (with respect to the number of users). Essentially, the cheapest schemes are described as learning-free and have both constant-time updates and queries while schemes involving a comparison with users in the database are classified as memory-based and offer constant-time updates but linear-time queries, and finally the schemes requiring more than linear time learning or more sophisticated updates are said to be model-based (see Tab. 1). There are schemes that would not fit in any one of these three classes of algorithms and others that would fit in more than one class.

Key words and phrases. Recommender System, Incomplete Vectors, Regression, e-Commerce.

Published in Information Retrieval, **8** (1), 129–150, 2005. This document may differ from the article published by Kluwer, please refer to it when available. NRC 46508.

	update	query	learning
learning-free	$O(1)$	$O(1)$	$O(m)$
memory-based	$O(1)$	$O(m)$	No
model-based	Variable	$O(1)$	Variable

TABLE 1. Typical complexities with respect to the number of users m of some classes of collaborative filtering algorithms.

Typically, learning-free schemes are derived from vectors $\{\mathbf{v}_k\}$ that are computed in linear time irrespective of the current user and the prediction is written as

$$Prediction(u) = \sum_{k=0}^{N} \beta_k(u)\mathbf{v}^{(k)}$$

where the result of the predictor is itself a vector where each component is the rating corresponding to an item. For example, the simplest learning-free scheme is obtained when $N = 0$, $\mathbf{v}^{(0)} = \mathbf{1}$ where $\mathbf{1} = (1, \ldots, 1)$ and $Prediction(u) = \bar{u}$ where $\bar{u}$ is the average over the known ratings. Another such scheme is obtained when $N = 0$, $\mathbf{v}_k^{(0)}$ is the average rating received by item number k, and $Prediction(u) = \mathbf{v}^{(0)}$.

Memory-based collaborative filtering systems usually compute weighted averages over ratings already in the database where the weights are given by a correlation measure [3, 12] or any similar measure [17] including probabilistic ones [10]. Generally, we can write a memory-based prediction as

$$Prediction(u) = F(u) + \sum_{w} \omega(w,u)w$$

where $F(u)$ is a learning-free prediction and where the *sum* is over all users in the database with $\omega(w,u)$ some measure of *similarity* between w and u. Because not all users have rated all items, the sum can be different for each item and we will make this point precise later. As there is little precomputation, updates to the database are fast, but queries tend to be slow as we need to match the current user against the entire database each time. Memory-based systems can outperform a wide range of model-based systems [3, 10] and accordingly, they are often used as reference collaborative filtering systems for benchmarking purposes. The main drawback of memory-based scheme is their lack of scalability. Some authors have proposed selecting the most representative or useful users from the database [18, 19] making memory-based systems more balanced in terms of update and query performance while preserving and even increasing slightly the accuracy. However, unlike learning-free and model-based schemes, memory-based systems require access to a database at all time and thus there are privacy issues [4] and a memory-based system cannot run conveniently on devices with very limited storage.

If all possible preference sets were equally likely, no prediction would be possible and since predictions have been shown to be reliable [3], it must be that there are many hidden constraints and few remaining degrees of freedom which suggests making predictions based on a model. Model-based collaborative filtering systems extract from the database some key parameters and do not use the database directly to answer queries. Examples include Principal Components Analysis (PCA) [7], Factor Analysis [4], Singular Value Decomposition [5, 15], Bayesian Networks [3], Item-Based models [16, 14] and Neural Networks [2]. Model-based systems tend to answer queries fast, most often in constant time with respect to the number of users, but also run potentially expensive learning routines and are often static in nature: updating the database can be expensive as it may require up to a completely new learning phase. Another possible drawback is that most model-based systems assume a large database is available whereas we would like collaborative filtering to work in a wide range of contexts.

One can test the accuracy of an algorithm by applying it on data where some of the ratings have been hidden. While results vary depending on the data set and the experimental protocol, most published collaborative filtering algorithms have similar prediction accuracies. For example, with the EachMovie data set, the accuracy improvement in going from a naive prediction based on per-item average (learning-free) to a sophisticated Factor Analysis approach is of no more than 17% [4]. Similarly, extensive work has been done to improve the Pearson correlation approach [3, 8] and yet, accuracy improvements do not exceed 20%. The differences between inexpensive schemes and more sophisticated ones are even smaller when one upgrades simple averaging scheme to the Bias From Mean algorithm introduced by Herlocker et al. [8]. In the results presented in this paper, the difference between the best and the worse scheme is of the order of 33% irrespective of the data set. In this context, systematic improvements by small percentages are significant.

One of the limitation researchers face is that there is no established set of desirable properties that are known to be needed in the design of a new collaborative filtering algorithm. Pennock et al. [11] outline properties or axioms for collaborative filtering algorithms but without measuring the practical usefulness of each axiom. They present four collaborative filtering properties : universal domain and minimal functionality, unanimity, independence of irrelevant alternatives, and scale invariance. Whereas scale invariance is a simple and compelling axiom, few scale invariant algorithms have been proposed. This paper investigates further scale invariance and aims to show that it is a useful axiom for collaborative filtering. To achieve this goal, we will consider several state-of-the-art collaborative filtering algorithms and propose novel variants that are scale invariant. Then, we show that the new algorithms perform better or as well as the old ones.

1.1. **Structure and Main Results.** The paper is organized as follows. We first introduce the collaborative filtering problem, then present some of the most competitive schemes, introduce two types of scale and translation invariant collaborative filtering systems, and finally, we conclude with some experimental results on two significantly different data sets.

The main results of the paper are evidence that scale and translation invariance is an important property that can be used to improve existing schemes and a set of novel highly scalable algorithms with good performance. We show that by normalizing users with respect to the mean, the amplitude, and, possibly, the number of their ratings, we improve accuracy. We stress that the normalization is *per user* as opposed to *per item* . One novel collaborative filtering systems ($STIN2$) performs well on both data sets, is simple to implement, and offers constant time performance for updates and queries.

1.2. **Notation and Terminology.** Vectors and arrays are written in bold ($\mathbf{v}$) as opposed to scalar values (i). Components of a vector $\mathbf{v}$ are noted $\mathbf{v}_i$. Incomplete vectors, that is vectors where some ratings are unknown, are written using the letters u,v, and w without bold face. Given a set S, we note $card(S)$ the cardinality of such as set. The Greek alphabet is used throughout for convenience without any special meaning. We note averages using the notation $\overline{a}$. Sums can be taken over indexes as in $\sum_{i=0}^{n} \mathbf{x}_i = \mathbf{x}_0 + \ldots + \mathbf{x}_n$ or over sets $S = \{a,b,c\}$ as in $\sum_{x \in S} f(x) = f(a) + f(b) + f(c)$. The variables m and n have special meaning consistent with other authors [3] and refer respectively to the number of users under consideration and to the number of items to be rated.

A *norm* $\|\cdot\|$ is typically defined as a non-negative real-valued function satisfying $\|\alpha v\| = |\alpha|\|v\|$ whenever α is a real number, $\|x+y\| \leq \|x\| + \|y\|$, and $\|x\| = 0 \Leftrightarrow x = 0$. In some sense, the norm of an object measures its *size*. We will abuse the terminology by dropping the condition that $\|x+y\| \leq \|x\| + \|y\|$ whenever $x+y$ is not defined.

Two norms $\|\cdot\|_{Norm1}$ and $\|\cdot\|_{Norm2}$ are *equivalent* if there exists positive numbers $A, B \in \mathbb{R}$ such that $A\|\mathbf{x}\|_{Norm2} \leq \|\mathbf{x}\|_{Norm1} \leq B\|\mathbf{x}\|_{Norm2}$ for all $\mathbf{x}$. For finite dimensional vector spaces such as $\mathbb{R}^k$, all norms are equivalent.

For the purpose of this paper, we define Lebesgue norms for $p = 1, 2, \ldots$ as

$$\|\mathbf{x}\|_{l_p} = \sqrt[p]{\sum_{i=1}^{N} \frac{|\mathbf{x}_i|^p}{N}}$$

where the sum is over all indexes of $\mathbf{x}$. We define the norm so that $\|(1,\ldots,1)\|_{l_p} = 1$ with the drawback that $\|\mathbf{x}\|_{l_2}$ is not given by $\sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$ but with the benefit that some of our notation is simpler. As examples, $\|(1,2,0)\|_{l_1} = 1$, $\|(1,2,0)\|_{l_2} = \sqrt{\frac{5}{3}} \approx 1.29$, and $\|(1,2,0)\|_{l_{10}} \approx 1.98$. For large p, Lebesgue norms become close to $\|\mathbf{x}\|_{l_\infty} = \max_{i \in \{1,\ldots,N\}} \{|\mathbf{x}_i|\}$ where the maximum is taken over all indexes. So that, intuitively, for small p's ($p = 1$) all components contribute to the norm value whereas for larger p's only the larger components contribute up to the point where only the very largest component matters. We have

$$\frac{\|\mathbf{x}\|_{l_\infty}}{\sqrt[p]{N}} \leq \|\mathbf{x}\|_{l_p} \leq \|\mathbf{x}\|_{l_\infty}.$$

2. Definitions

Let ι be an ordered set of n *items* labeled from 1 to n which we write $\iota = \{1, \ldots, n\}$ for simplicity. Each *user* in the system is allowed to give one and only one rating to each item, but will generally rate only a small subset of all possible items. We refer to these ratings as an *evaluation* and there is a one-to-one map between users and evaluations. Given an evaluation u, let $S(u)$ be the set of items rated and let u_i be the rating given to item $i \in S(u)$ by this user. For the purpose of this paper, we assume that ratings are real numbers even though ratings are often taken from a finite set such as {good, average, bad}. In other words, an evaluation is a function $u : \iota \supset S(u) \to \mathbb{R}$ where the cardinality of the domain $S(u)$ is typically much smaller than $card(\iota)$. Alternatively, u can be thought of as a incomplete vector in $\mathbb{R}^n$, that is, a vector where some components are unknown. In practice, a set of evaluations χ is given for training and we note the cardinality of the set χ by $m = card(\chi)$. On the other hand, the set of all possible evaluations is noted $\Xi \supset \chi$. Note there may be some constraints on Ξ: for example that some items are rated by all users or that all users have rated at least a given number of items.

A complete evaluation is an evaluation u with ratings over all items ($S(u) = \iota$). Given a new evaluation w with, again, arbitrary numerous ratings over up to n items, we seek to find a complete evaluation $\mathbf{w}$ such that w and $\mathbf{w}$ are *close* and such that $\mathbf{w}$ agrees *as much as possible* with χ. We say that $\mathbf{w}$ is a *prediction* and we write $\mathbf{w} = P(w)$ where P is a function (called a predictor) from the space of all evaluations to the space of complete evaluations. Thus, a prediction is a map from incomplete vectors to complete vectors. Note that this definition is not general since it excludes top-N algorithms [9].

Definition 2.1. A function $P : \Xi \rightarrow \mathbb{R}^n$ is called a *predictor*.

Predictors are often built dynamically using a set of evaluations or *training set* and we refer to this process as a collaborative filtering system.

Definition 2.2. A *collaborative filtering system* (CFS) is a function from sets of evaluations χ to predictors.

Given two numbers $\alpha, \beta \in \mathbb{R}$ and an evaluation u, we define a new evaluation $w = \alpha u + \beta$ by the ratings $w_i = \alpha u_i + \beta$ for all items $i \in S(v)$ and $S(r) = S(v)$. We note $u_{|\sigma}$ for $\sigma \subset \iota$ the evaluation u limited to ratings over the set of items σ: $u_{|\sigma}$ satisfies $S(u_{|\sigma}) = \sigma \cap S(u)$ and $u_{|\sigma,i} = u_i$ for all $i \in \sigma \cap S(u)$. Given a constant $\beta \in \mathbb{R}$ and a set of ratings $\sigma \subset \iota$, we define the constant evaluation $w = \beta_{|\sigma}$ by $w_i = \beta$ for all i in σ and $S(w) = \sigma$. We note $\bar{u}$ the average rating of evaluation u. Similarly, $\bar{u}_{|\sigma}$ for $sigma \subset \iota$ is the average rating of u over items in $sigma \cap S(u)$. We define the inner product of $u \in \Xi$ and $\mathbf{x} \in \mathbb{R}^n$ by $\langle u, \mathbf{x} \rangle = \sum_{i \in S(u)} \mathbf{x}_i u_i$ and the inner product of $u, w \in \Xi$ by $\langle u, w \rangle = \sum_{i \in S(u) \cap S(w)} w_i u_i$, and note $\|\mathbf{x}\|^2_{l_2(\sigma)} = \frac{1}{card(\sigma)} \sum_{i \in \sigma} \mathbf{x}_i^2$, $\|u\|^2_{l_2} = \frac{1}{card(S(u))} \sum_{i \in S(u)} u_i^2$, $\langle \mathbf{x}^{(1)}, \mathbf{x}^{(2)} \rangle_\sigma = \sum_{i \in \sigma} \mathbf{x}^{(1)}{}_i \mathbf{x}^{(2)}{}_i$.

Ratings are inaccurate if only because there are malicious users. We say that a CFS is stable if a single user in a large user set doesn't make a difference. This is often the case if we take averages over the entire database for example. However, it is necessary to make some assumptions about the evaluation set χ for stability to be possible. For example, we must assume that, for every item, there is a large set of evaluations with corresponding ratings. Otherwise, if a given item is rated by only a few users, and these users have given inaccurate ratings, then the predictions regarding this item may be inaccurate. Among the CFS schemes that are **not** stable are the N closest neighbor schemes unless N is large because these systems assume that the N closest neighbors have given accurate ratings. All schemes considered in this paper are stable under *reasonable assumptions* .

Because the mapping from user ratings to $\mathbb{R}$ is arbitrary, a CFS must be independent of such a mapping or *normalization* . For example, we can map ratings such as {good, average, bad} to numerical values $\{-10, 0, 10\}$ or $\{1, 2, 3\}$ and clearly, both choices are equally sensible. For $\alpha > 0, \beta \in \mathbb{R}$, let $m_{\alpha,\beta}(u) = \alpha u + \beta$, we say that a CFS is *normalization invariant* if the predictor $P_{\alpha,\beta}$ obtained with the evaluation set $m_{\alpha,\beta}(\chi)$ relates to the predictor P obtained with the set χ by $P_{\alpha,\beta}(m_{\alpha,\beta}(u)) = m_{\alpha,\beta}(P(u)) = \alpha P(u) + \beta$ for all $\alpha > 0, \beta \in \mathbb{R}$ and all $u \in \Xi$. All CFS considered in this paper are normalization invariant.

Scale and Translation Invariance (STI) states that each user may have its own scale when rating items and is a stronger condition than normalization invariance. If

$$P(m_{\alpha,\beta}(u)) = m_{\alpha,\beta}(P(u))$$

for all $\alpha \geq 0, \beta \in \mathbb{R}$ that is P commutes with $m_{\alpha,\beta}$, then the predictor is said to be STI. Similarly, if replacing any evaluation $u \in \chi$ by $m_{\alpha,\beta}(u)$ for $\alpha > 0, \beta \in \mathbb{R}$ doesn't change the predictor P and that such a P is STI for all χ, we say that the CFS is STI. This property is based on the assumption that each user has its own static frame of reference that needs to be compensated for: some users might tend to be naturally generous, others might be more critical, whereas others might rate most items as roughly similar while others tend to use more often extreme ratings. **Note that we do not allow α to be negative**: a user who likes item A and dislikes item B is not *equivalent* to a user who dislikes item A and likes item B. A predictor is scale invariant if $P(\alpha u) = \alpha P(u)$ for all $0 < \alpha \in \mathbb{R}$ and all $u \in \Xi$ and it is translation invariant if $P(u + \beta) = P(u) + \beta$ for all $\alpha, \beta \in \mathbb{R}$ and all $u \in \Xi$. Notice that a STI CFS is automatically normalization invariant because the predictor is invariant under a transformation $m_{\alpha,\beta}$ of the evaluation set χ so that $P_{\alpha,\beta} = P$ and P commutes with $m_{\alpha,\beta}$ because it is STI.

All schemes considered in this paper are translation invariant except for Per Item Average and Eigentaste 2.0. In the case of memory-based CFS, it is documented [3] that non translation invariant (cosine-based) are inferior to translation invariant ones (Pearson-based). On the other hand, there is no comparable studies regarding scale invariance. Assuming that the scale of the ratings is not useful information, we argue that it is actually better to normalize the ratings. Indeed, otherwise a user with more extreme ratings will *count more* than a user with more modest ratings. For example on range from 1 to 10, if user A gives movie 1 a 10 and movie 2 a 1 whereas user B gives movie 1 a 4 and movie 2 a 6, we have two users in disagreement and without scale invariance, user B's opinion is going to be overwritten by user A.

2.1. **Measuring the Accuracy.** Many authors use the Mean Absolute Error [3, 8] of the prediction error where only a subset of items $\sigma \subset S(u)$ is assumed to be known and the rest is hidden

$$MAE(u,\sigma) = \frac{\sum_{i \in S(u)-\sigma} \left| P(u_{|\sigma})_i - u_i \right|}{card(S(u)-\sigma)}$$

and one important MAE error measure is obtained by subtracting a single element i from $S(u)$ and setting $\sigma = S(u) - \{i\}$, that is $S(u) - \sigma = \{i\}$,

$$ALLBUT1(P,u,i) = \left| P(u_{|S(u)-\{i\}})_i - u_i \right|.$$

Canny [4] pointed out that such an AllBut1 error measure is the most realistic error measure given a large enough database. Another argument for using the AllBut1 measure is that there are many different error measures for vector elements, when $S(u) - \sigma$ is not a singleton, whereas it is unique up to a factor when the error over one element is concerned (AllBut1). In the remainder of this paper, we will use the AllBut1 Mean Absolute Error (*AllBut1 MAE*) [3] to measure the prediction error of a predictor P over a test evaluation set $\chi' \subset \Xi$

$$\frac{1}{card(\chi')} \sum_{u \in \chi'} \frac{1}{card(S(u))} \sum_{i \in S(u)} AllBut1(P,u,i). \tag{1}$$

With some schemes such as Eigentaste 2.0 or STI Eigentaste the sum over $S(u)$ above must exclude items that are part of the *standard* set of items because the predictor assumes that some fixed items have been rated. Note that some authors prefer the NMAE which is defined as the MAE divided by the range of observed rating values [7].

3. Competitive Collaborative Filtering Systems

We begin by describing the most commonly used learning-free schemes. The simplest CFS is given by $P_{average}(u) = \bar{u}$ which says that a user is likely to rate new items as the average of its known ratings and it is STI. While nearly as accurate as other schemes using AllBut1 MAE as measure, it proves of little use in practice and is only used as for benchmarking. Indeed, it doesn't provide any order on the items since it predicts they are all rated equal.

The next scheme is the Per Item Average or POP [7] given by $P_{per\,item}(u)_i = \frac{1}{card(S_i(\chi))} \sum_{w \in S_i(\chi)} w_i$ where $S_i(\chi) = \{w \in \chi : i \in S(w)\}$ and it can be described as predicting that any given user will like any given item as much as the average rating for that item. Most applications where users are invited to rate items use Per Item Average implicitly by averaging the ratings. We argue that it is probably the best one can do, if nothing is known about the current user. Happily, the literature and our experimental results show that it is possible to leverage the knowledge we have of the current user to improve predictions.

Finally, there is one more commonly used learning-free scheme [8] called Bias From Mean which tends to outperform the previous two in our experiment,

$$P_{bias}(u)_i = \bar{u} + \frac{1}{card(S_i(\chi))} \sum_{w \in S_i(\chi)} w_i - \bar{w}. \tag{2}$$

It combines both the average and the Per Item Average approaches in a single scheme. It does better than the Per Item Average because it uses some information about the current user (mean rating).

Assuming that the $card(S_i(\chi))$ are stored for $i \in \iota$, all of three of these schemes can be updated in constant time with respect to the number of users whenever a value is changed or a user is added. Queries are in constant time.

The Bias From Mean scheme is normalization invariant as a corollary of the following proposition by setting $\omega \equiv 1$.

Proposition 3.1. *Weighted sum CFS of the form*

$$P(u)_i = \bar{u} + \frac{1}{card(S_i(\chi))} \frac{\sum_{w \in S_i(\chi)} \omega(u,w)\,(w_i - \bar{w})}{\sum_{w \in S_i(\chi)} |\omega(u,w)|}$$

	item 1	item 2	item 3	item 4
u_1	unrated	5	unrated	3
u_2	2	unrated	4	unrated
u_3	unrated	3	unrated	1
u_4	1	unrated	5	unrated
u_5	4	5	4	3
u_6	1	3	5	unrated

TABLE 2. Example of an evaluation set where ratings are from 1 (very poor) to 5 (very good).

for all $i \in \iota$, are normalization invariant if and only if ω is normalization invariant:

$$\omega(m_{\alpha,\beta}(u), m_{\alpha,\beta}(w)) = \omega(u,w).$$

In the proposition above, whenever $\omega(u,w)$ which might measure the similarity between u and w, depends on u, CFS is memory-based. A commonly used normalization invariant choice is $\omega = \omega_{Pearson}$ where

$$\omega_{Pearson}(u,w) = \frac{\langle u - \overline{u}, w - \overline{w}\rangle}{\sqrt{\langle u_{|S(w)} - \overline{u}, u_{|S(w)} - \overline{u}\rangle \langle w_{|S(u)} - \overline{w}, w_{|S(u)} - \overline{w}\rangle}},$$

is the **Pearson correlation** [12] between u and w over $S(u) \cap S(w)$. There exists variants of this scheme [8, 3] using *case amplification* where

$$\omega(u,w) = \omega_{Pearson}(u,w)\,|\omega_{Pearson}(u,w)|^{\rho-1}$$

with $\rho \geq 1$. Intuitively, case amplification tends to favor close neighbor as small values raised to a power become negligible, and it improves accuracy for some values of ρ such as $\rho \approx 2.5$ [3].

Per Item Average, Bias From Mean and all the memory-based schemes we discussed are not STI. We will propose STI variants and show that they tend to perform better.

4. SCALE AND TRANSLATION INVARIANT CFS

We say that u and v are equivalent, $u \sim v$, if there exists $\alpha > 0, \beta \in \mathbb{R}$ such that $\alpha u_i + \beta = v_i$ for all $i \in S(u) \cap S(v)$, $u_i = \overline{u}$ when $i \in S(u) - S(v)$, and $v_i = \overline{v}$ when $i \in S(v) - S(u)$. In other words, u and v contain the same information as they are identical up to a change of scale over $S(u) \cap S(v)$ and contain no information elsewhere ($u_i = \overline{u}$ and $v_i = \overline{v}$). In Tab. 2, for example, evaluations 1 and 3 are equivalent by translation (off by 2), whereas evaluations 2 and 4 are equivalent by scale in the sense that $2(u^{(2)} - \overline{u^{(2)}}) = u^{(4)} - \overline{u^{(4)}}$.

We can show that the condition $\alpha u_i + \beta = v_i$ can be replaced by the simpler condition that $v_i - \overline{v} = \alpha(u_i - \overline{u})$.

Proposition 4.1. *$u \sim v$ if and only if there exists $0 < \alpha \in \mathbb{R}$ such that $v_i - \overline{v} = \alpha(u_i - \overline{u})$ for all $i \in S(u) \cap S(v)$, $u_i = \overline{u}$ when $i \in S(u) - S(v)$, and $v_i = \overline{v}$ when $i \in S(v) - S(u)$.*

To see why this is true, assume $u \sim v$ then $\alpha u_i + \beta = v_i$ when $i \in S(u) \cap S(v)$, and so $\overline{v} = \alpha\overline{u} + \beta$ and $v_i - \overline{v} = \alpha(u_i - \overline{u})$. The reciprocal is true, if $u_i = \overline{u}$ when $i \in S(u) - S(v)$, and $v_i = \overline{v}$ when $i \in S(v) - S(u)$ and $v_i - \overline{v} = \alpha(u_i - \overline{u})$ for some $\alpha > 0 \in \mathbb{R}$ for all $i \in S(u) \cap S(v)$, then $u \sim v$ by choosing $\beta = -\alpha\overline{u} + \overline{v}$.

Because we can show that the condition that u and v differ only in scale ($\alpha u_i + \beta = v_i$) can be expressed by $v_i - \overline{v} = \alpha(u_i - \overline{u})$, the next step is to subtract from ratings their means and divide by their norm. This normalization is justified because STI schemes should not depend on either the average or the norm of ratings: by normalizing evaluations we ensure that resulting schemes are STI. Whereas the mean of an evaluation is well defined, the norm of a set of ratings can be defined in many ways especially because the number of ratings may differ from a user to another. Given a norm $\|\cdot\|$, we can define a map $m_{\|\cdot\|}(u)$ from all incomplete vectors (Ξ) to $\mathbb{R}^n$ by

$$m_{\|\cdot\|}(u)_i = \begin{cases} \frac{u_i - \overline{u}}{\|(u_k - \overline{u})_{k \in S(u)}\|} & i \in S(u) \\ 0 & i \notin S(u) \end{cases} \tag{3}$$

where by convention, $0/0 = 0$ and $\|\cdot\|$ is any norm. Empirically, we found that l_p norms were a good choice and we write $m_p = m_{\|\cdot\|_{l_p}}$. Because the l_p norm, defined the way we did, is normalized against the number of ratings, it doesn't tend to grow as users rate more items which intuitively means that we don't penalize users who rate a large number of items. We also consider $\mu_p = m_{\|\cdot\|_{l_p} \times card(S(\cdot))}$ where the l_p norm is multiplied by the number of known ratings. The maps μ_p will penalize users who rated a large number of items and scale down their ratings accordingly. Many other norms are possible, but we only choose these two as representatives.

	item 1	item 2	item 3	item 4
$m_2(u^{(1)})$	0	1	0	-1
$m_2(u^{(2)})$	-1	0	1	0
$m_2(u^{(3)})$	0	1	0	-1
$m_2(u^{(4)})$	-1	0	1	0
$m_2(u^{(5)})$	0	$\sqrt{2}$	0	$-\sqrt{2}$
$m_2(u^{(6)})$	$-\sqrt{3/2}$	0	$\sqrt{3/2}$	0

TABLE 3. Evaluations from Tab. 2 transformed using map m_2. We see evaluations 1 and 3 as well as evaluations 2 and 4 are equivalents.

Given two users in disagreement but with different amplitude, $u^{(1)} = (-1, 1, unrated)$ and $u^{(2)} = (10, -10, unrated)$, a STI scheme would first normalize them so that

$$m_p(u^{(1)}) = \mu_p(u^{(1)}) = -m_p(u^{(2)}) = -\mu_p(u^{(2)}).$$

And therefore, they would cancel each other. On the other hand, for two users with different rating sets, such as $u^{(1)'} = (-1, 1, -1, 1)$ and $u^{(2)'} = (-1, 1, unrated, unrated)$, we can either say that the evaluations have the same norm (m_p) or else that evaluation $u^{(1)'}$ has greater norm from the fact that it rated twice the number of items (μ_p). As we shall see, it is possible to avoid these difficulties by using a standard set of items rated by all users, but in general, we must cope with many perfectly valid normalizations and choose based on empirical results.

Observe that for any $\alpha > 0, \beta \in \mathbb{R}$, $m_{\|\cdot\|}(\alpha u + \beta) = m_{\|\cdot\|}(u)$ so that $m_{\|\cdot\|}$ is defined over equivalence classes. The next lemma makes this precise whereas Tab. 3 gives an example.

Lemma 4.2. *$u \sim v$ if and only if $m_{\|\cdot\|}(u) = m_{\|\cdot\|}(v)$.*

Classes of equivalence can be made into an Hilbert space $H(m_{\|\cdot\|})$, and in this sense, our approach to CFS design is geometrical. For example, $w + v$ is defined by the equivalence class of all u's such that $m_p(u) = m_p(w) + m_p(v)$. Similarly, the inner product between w and r is defined by $\langle w, v \rangle_p = \langle m_p(w), m_p(v) \rangle$ and the norm of u is $\|m_p(u)\|_{l_2}$. We will define our novel STI schemes using m_2 and μ_2.

5. LEARNING-FREE SCALE AND TRANSLATION INVARIANT CFS

We consider learning-free predictors of the form $P(u) = \sum_{i=0}^{k} \omega_i(u)\mathbf{v}^{(i)}$ where $\mathbf{v}^{(0)}, \ldots, \mathbf{v}^{(k)} \in \mathbb{R}^n$ and $\omega_i : \Xi \to \mathbb{R}$ for $i \in \{0, \ldots, k\}$ are corresponding functions mapping evaluations to coefficients. If P is STI then we must have $P(u + \beta) = P(u) + \beta$ and so, we choose $\mathbf{v}^0 = \mathbf{1}$. We found empirically that it was efficient to use *regression* and thus, to set the coefficients $\omega_i(u)$ such as to **minimize**

$$\Delta_2(P, u) = \left\| P(u)_{|S(u)} - u \right\|_{l_2(S(u))}.$$

In other words, we choose the coefficients in such a way as to make $P(u)$ as close as possible to u. This choice also makes P STI. The simplest such scheme is defined by $k = 0$ and it amounts to $P(u) = \overline{u}$.

The next step is to introduce a STI variant of both the Per Item Average and Bias From Mean schemes. Thus, we define the first-order STI non personalized scheme ($STIN1(m_{\|\cdot\|})$) with

$$\mathbf{v}_i^{(1)} = \frac{1}{card(S_i(\chi))} \sum_{u \in S_i(\chi)} m_{\|\cdot\|}(u)_i$$

where $\mathbf{v}_i^{(1)}$ is the i^{th} component of $\mathbf{v}^{(1)}$ and $card(S_i(\chi))$ is a short-hand for the number of evaluations u such that $i \in S(u)$. Intuitively $\mathbf{v}^{(1)}$ is the average of the evaluations $u \in \chi$ over the space $m_{\|\cdot\|}(\chi)$. Minimizing the residual energy $\left\| u - P_{STIN1(m_{\|\cdot\|})}(u) \right\|_{l_2(S(u))}^2$ and defining $\mathbf{v}_u^{(1)} = \mathbf{v}^{(1)} - \overline{\mathbf{v}_{|S(u)}^{(1)}}$, we have

$$P_{STIN1(m_{\|\cdot\|})}(u) = \overline{u} + \frac{\left\langle u, \mathbf{v}_u^{(1)} \right\rangle}{\langle \mathbf{v}_u^{(1)}, \mathbf{v}_u^{(1)} \rangle_{S(u)}} \mathbf{v}_u^{(1)}.$$

See Appendix for a practical example of how to compute efficiently $STIN1(m_2)$.

We can extend this framework further using several vectors $\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \ldots, \mathbf{v}^{(k)} \in \mathbb{R}^N$ by defining

$$m_{\|\cdot\|}^{\mathbf{v}^{(1)},\mathbf{v}^{(2)},\ldots,\mathbf{v}^{(k)}}(u)_i = \begin{cases} \frac{u_i - u'_j}{\left\| \left(u_k - u'_k\right)_{k \in S(u)} \right\|} & i \in S(u) \\ 0 & i \notin S(u) \end{cases}$$

where $u' = Proj_{v_1, v_2, \ldots, v_k}(u)$ is the vector of the form $\alpha_{\mathbf{v}^{(1)}}\mathbf{v}^{(1)} + \ldots + \alpha_{\mathbf{v}^{(k)}}\mathbf{v}^{(k)}$ where $\alpha_{\mathbf{v}^{(1)}}, \ldots, \alpha_{\mathbf{v}^{(k)}} \in \mathbb{R}$ are chosen to minimize

$$\left\| u - (\alpha_{\mathbf{v}^{(1)}}\mathbf{v}^{(1)} + \ldots + \alpha_{\mathbf{v}^{(k)}}\mathbf{v}^{(k)}) \right\|_{l_2}.$$

Just like with $m_{\|\cdot\|}$, there are equivalence classes corresponding to $m_{\|\cdot\|}^{\mathbf{v}^{(1)},\ldots,\mathbf{v}^{(k)}}$. Explicitly, we define a second-order STI non personalized scheme ($STIN2(m_{\|\cdot\|})$) by using $\mathbf{v}^{(0)}, \mathbf{v}^{(1)}$ as previously defined, using the specified norm, and

$$\mathbf{v}_i^{(2)} = \frac{1}{card(S_i(\chi))} \sum_{u \in S_i(\chi)} m_\infty^{\mathbf{v}^{(0)},\mathbf{v}^{(1)}}(u)_i.$$

Again, $\mathbf{v}_2$ can be thought of an average over $m_\infty^{\mathbf{v}^{(0)},\mathbf{v}^{(1)}}(\chi)$. Notice that we always use the l_∞ norm when computing $\mathbf{v}^{(2)}$, irrespective of the choice that was make for $\mathbf{v}^{(1)}$ as it was found to slightly improve results basis. Just like with $STIN1(m_{\|\cdot\|})$, we minimize the residual energy $u - P_{STIN2(m_{\|\cdot\|})}(u)$ by choosing

$$P_{STIN2}(u) = P_{STIN1}(u) + \frac{\left\langle u, \mathbf{v}_u^{(2)} \right\rangle}{\langle \mathbf{v}_u^{(2)}, \mathbf{v}_u^{(2)} \rangle_{S(u)}} \mathbf{v}_u^{(2)}$$

where $\mathbf{v}_u^{(2)} = \mathbf{v}^{(2)} - Proj_{\mathbf{v}^{(0)},\mathbf{v}^{(1)}}(u)$. Because $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$ are computed once and updated only when there are new ratings, P_{STIN2} is easy to implement.

Higher order $STINx$ schemes exists, but are likely to be of little practical use because the difference in practice between $STIN1$ and $STIN2$ is already small (see Tab. 4). $STINx$ schemes can be updated in constant time with respect to the number of users and they are always STI.

6. Memory-Based Scale and Translation Invariant CFS

We define the STI equivalent of the Pearson correlation predictor called $STIPearson(m_2)$ by using the form

$$P_{STIPearson(m_2)}(u)_i = \bar{u} + \frac{\langle \tau(u,\chi), u \rangle}{\langle \tau(u,\chi), \tau(u,\chi) \rangle_{S(u)}} \tau(u,\chi)$$

and $\tau(u,\chi)$ is the weighted average over the space $m_2(\chi)$

$$\tau(u,\chi) = \frac{\sum_{w \in \chi, i \in S(w)} \omega(u,w) m_2(w)}{\sum_{w \in \chi, i \in S(w)} |\omega(u,w)|}$$

where by convention $0/0 = 0$. In this last equation, we choose

$$\omega(u,w) = \langle m_2(u), m_2(w) \rangle \left| \langle m_2(u), m_2(w) \rangle \right|^{\rho - 1}$$

and $\rho \geq 1$ is a case amplification power where $\rho = 2.5$ is typically chosen. Note that despite the name "$STIPearson$", $\omega(u,w)$ is not correlation-based, but uses a simple scalar product in the $m_2(\Xi)$ Hilbert space and as a side-effect, it can be computed faster that the Pearson correlation assuming that the $m_2(w)$ for $w \in \chi$ are precomputed. As for $STIPearson(\mu_2)$, it is identical to $STIPearson(m_2)$ except that we replace every occurrence of m_2 by μ_2.

7. EigenTaste 2.0 and STI Eigentaste

The Jester data set [7] was acquired on the web by asking users to first rate a common set of jokes and then providing these users with recommendations. The Eigentaste 2.0 scheme is a collaborative filtering system which was designed specifically for this data set. It uses a normal set γ of 10 jokes that all users have rated. The basic idea is that we can greatly simplify the analysis if we have a normal set since the restriction of the evaluations to this normal set becomes a vector space. Intuitively, one might expect that the existence of a normal set can be used to outperform schemes that don't make use of such a normal set. The Eigentaste 2.0 scheme applies a Principal Component Analysis, also sometimes called a Karhunen-Loève transform, on this vector space.

The authors Eigentaste 2.0 normalize the ratings by subtracting the per item mean and dividing this bias from mean by the standard deviation of these ratings. We implement Eigentaste 2.0 both with and without this normalization of the items in the normal set and find that this per item normalization actually degrade the accuracy in our experiment possibly because our tests involve smaller training sets. Consequently, we only present the simpler version of Eigentaste 2.0 without normalization.

We first compute the 10×10 matrix

$$U = \frac{1}{card(\chi)} \sum_{u \in \chi} u_{|\gamma} u_{|\gamma}^T$$

where $u_{|\gamma}$ is treated as a vector of length 10 and $u_{|\gamma}^T$ is the transpose of this vector. We then find two dominant *eigenvectors* $\upsilon^{(1)}$ and $\upsilon^{(2)}$ of U, that is two eigenvectors corresponding to the two highest eigenvalues. This allows us to map any evaluation $u \in \chi$ to the (x,y) coordinates

$$(\left\langle u_{|\gamma}, \upsilon^{(1)} \right\rangle, \left\langle u_{|\gamma}, \upsilon^{(2)} \right\rangle).$$

We then use these two eigenvectors to partition the evaluation set χ in $4\eta^2$ clusters where $\eta \in \mathbb{N}$ is a positive integer (see Fig. 1). To do so, first find $M_i = \max_{u \in \chi} \left| \left\langle u_{|\gamma}, \upsilon^{(i)} \right\rangle \right|$ for $i = 1,2$ to define the range of values in the eigenplane. This division of the plane into ever smaller rectangles is easy to visualize, but for implementation purposes, we need a precise formula such as what we present next. Given $u \in \chi$, let λ_i be the *logarithm* of $\left\langle u_{|\gamma}, \upsilon^{(i)} \right\rangle$ defined as

$$\lambda_i(u) = \max \left\{ k \in \mathbb{N} : \left| \left\langle u_{|\gamma}, \upsilon^{(i)} \right\rangle \right| / M_i \leq 1/2^{k-1} \right\}$$

and $\lambda_{i,\eta}(u) = \max\{1, \min\{\eta, \lambda_i\}\}$ for $i = 1,2$. Define integer-valued functions $I(u), J(u)$ satisfying $-\eta \geq I(u), J(u) \geq -\eta$ and $I(u), J(u) \neq 0$ with

$$I(u) = \lambda_{1,\eta}(u) sign\left(\left\langle u_{|\gamma}, \upsilon^{(1)} \right\rangle \right)$$

and

$$J(u) = \lambda_{2,\eta}(u) sign\left(\left\langle u_{|\gamma}, \upsilon^{(2)} \right\rangle \right)$$

where $sign(x) = 1$ when $x \geq 0$ and $sign(x) = -1$ otherwise.

These functions I, J allow us to determine in which cluster of the eigenplane any given evaluation is. That is, $u, v \in \chi$ are in the same cluster if and only if $I(u) = I(v)$ and $J(u) = J(v)$. Correspondingly, we can look at the set of all evaluations in a given cluster $\chi_{i,j} = \{u \in \chi : I(u) = i, J(u) = j\}$.

Once we have determined in which cluster of evaluations u belongs, it is then reasonable to simply predict that u will rate according to the per item average using its *neighbors* (evaluations in the same cluster). Thus, for each *cluster* $\chi_{i,j}$, we compute the averages $\mathbf{A}_{i,j} \in \mathbb{R}^n$ just like we did with the Per Item Average scheme,

$$(\mathbf{A}_{i,j})_k = \frac{1}{card(u \in \chi_{i,j}, k \in S(u))} \sum_{u \in \chi_{i,j}, k \in S(u)} u_k.$$

The Eigentaste predictor is then defined by $P(u) = \mathbf{A}_{I(u),J(u)}$.

Because the averages $\mathbf{A}$ and the eigenvalues $\upsilon^{(i)}$ need only the be updated when new data is added, the queries can be done in constant time with respect to the number of users. For high η, we have more clusters and thus, better granularity, but each cluster contains less evaluations and thus, averages might be less reliable. We choose $\eta = 4$ as it is the value reported in [7] and it is found empirically to be a good choice. Eigentaste 2.0 is not STI. It is interesting to note that in the limit case where there is a single cluster of evaluations, this scheme amounts to the Per Item Average CFS.

To produce a STI variant of the Eigentaste algorithm, we use exactly the same algorithm as Eigentaste 2.0 except that we replace u throughout by $\frac{u_i - \bar{u}_{|\gamma}}{\|u_{|\gamma} - \bar{u}_{|\gamma}\|_{l_2}}$ in the computation of the eigenvectors and clusters. Notice that unlike the maps $m_{\|\cdot\|}$, this map doesn't depend on the number of items u has rated since it relies exclusively on the items in the standard set γ.

We note the set of evaluations in each cluster by $\chi_{i,j}^{STI}$ instead of $\chi_{i,j}$ and we have integer-valued functions I^{STI} and J^{STI} corresponding to I and J. The *per cluster* averages are given by

$$(\mathbf{A}_{i,j}^{STI})_k = \frac{1}{card(u : u \in \chi_{i,j}^{STI}, k \in S(u))} \sum_{u \in \chi_{i,j}^{STI}, k \in S(u)} \frac{u_i - \bar{u}_{|\gamma}}{\|u_{|\gamma} - \bar{u}_{|\gamma}\|_{l_2}},$$

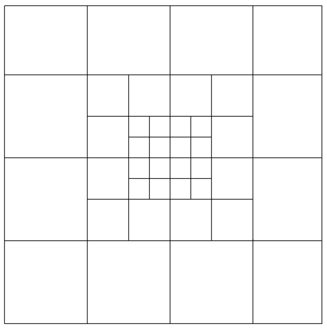


FIGURE 1. Eigentaste clusters ($\eta = 4$).

just like the corresponding $\mathbf{A}_{i,j}$. The STI Eigentaste predictor is defined by regression as $P(u) = \bar{u} + \alpha \mathbf{A}_{I^{STI}(u),J^{STI}(u)}$ where α is chosen to minimize

$$\left\| u - \bar{u} - \alpha \mathbf{A}_{I^{STI}(u),J^{STI}(u)} \right\|_{l_2}.$$

One could view STI Eigentaste as Eigentaste 2.0 with regression and *per user* normalization. STI Eigentaste is STI.

While much more lightweight than memory-based schemes, Eigentaste schemes are not learning-free. The averages $\mathbf{A}_{i,j}$ and $\mathbf{A}^{STI}_{i,j}$ can be updated in constant time with respect to the number of users if we assume that the eigenvectors are constant, however once we take into account that the eigenvectors will change albeit slowly as we add more users, the update cost is linear in the number of users $O(m)$. Computing the new eigenvectors themselves is a constant time operation and it only need to be done when more users are added and not when users add ratings. We argue that having slow updates is not as much a problem as having slow queries since updates can be implemented off-line as a background task.

8. EXPERIMENTAL RESULTS

8.1. **Data Sets.** The EachMovie data set is a the result of a movie rating web site. The DEC Systems Research Center ran this web site for 18 months and 72,916 users entered a total of 2,811,983 numerical ratings for 1,628 different movies (films and videos). It has ratings from 0.0 to 1.0 in increments of 0.2.

The Jester data set is the outcome of a joke rating web site [7]. Users are rate a fixed number of jokes and they are then presented with recommendations. According to the documentation, the Jester data set has *continuous* ratings from -10.0 to 10.0 however we found that very few ratings (less than 1%) were beyond this range.

As a basis for comparison, we used Amazon SOAP open API to retrieve the information about Music CDs. On June 20th 2003, metadata about all Music CDs from the web site Amazon.com were downloaded and only the 5,958 CDs with ratings were kept. The API provides the average rating for each item. We present the plots giving the frequency of various ratings on three data sets: EachMovie, Jester, and Amazon (see Tab. 2). We notice that users tend to give positive ratings more often than negative ratings and maybe it can be explained by saying that users tend to rate what they like.

8.2. **Methods.** We used case amplification on both Pearson and *STI Pearson* with a power of ρ=2.5 as this improves results with both algorithms and is the power value chosen by other authors [3]. We only kept evaluations with at least 20 ratings as in [8]. For each algorithm, we computed the AllBut1 MAE (see equation 1) using enough evaluations to have a total of 50,000 ratings as a training set (χ) and another set of evaluations with a total of at least 100,000 ratings as the testing set (χ'), and we repeated the process 6 times over different pairs χ, χ' for each data set keeping only the average and the standard deviation [8]. Using larger training sets is difficult when benchmarking memory-based schemes because of the computational burdens. Because all ratings in the testing set are hidden once, each of the 6 pairs χ, χ' involves 100,000 predictions. The only exception to this rule is with Eigentaste and STI Eigentaste where we never hide one of the 10 items in the standard set. The typical relative standard deviation ($N = 6$) for AllBut1 MAE values in both data sets is 5%.

As an additional step, we attempt to improve predictions by replacing predicted ratings above or below the allowed range of values ($[0.0, 1, 0]$ for EachMovie and $[-10.5, 10.5]$ for Jester) by the nearest value inside the range: this step proves futile as it doesn't improve results in a noticeable way over such large sets. In practice, such a rounding step might still be implemented for practical reasons.

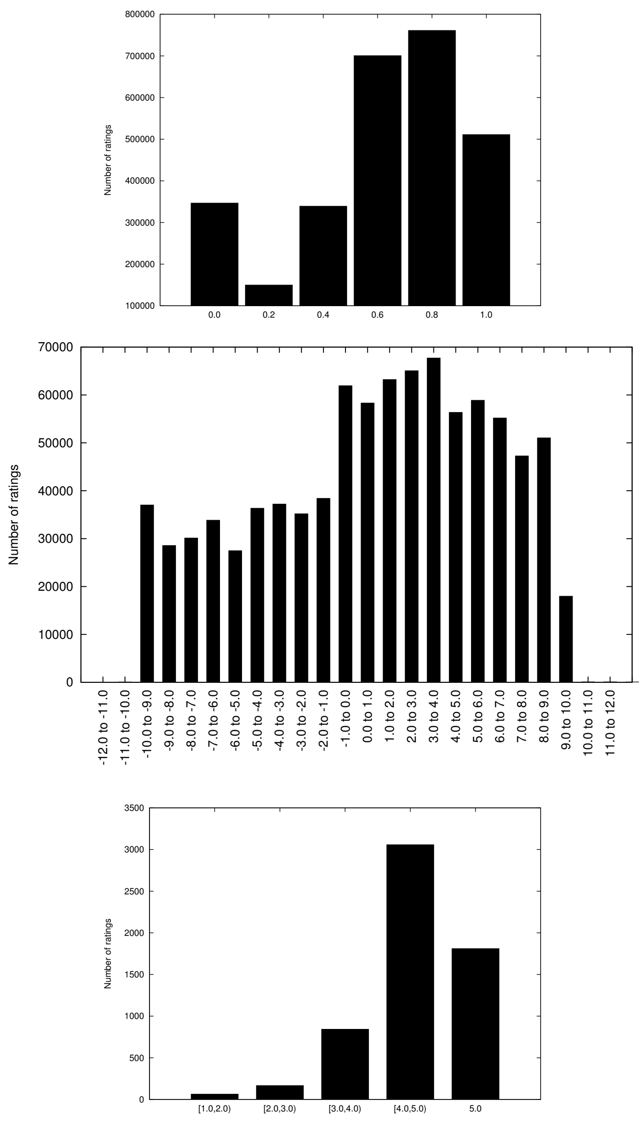


FIGURE 2. Frequency Bar Charts for the EachMovie, Jester, and Amazon Album Ratings.

8.3. **Results.** In EachMovie, we find 36,656 users with at least 20 ratings each for a total of 2,579,985 ratings at an average of 70.3 ratings per evaluations. Movies are labeled using integers from 1 to 1,649. The density of ratings over the chosen users is at about 4%. The typical AllBut1 MAE for EachMovie is 0.2. Because EachMovie has a rating range of 1, normalized MAE (NMAE) are the same as the MAE values (AllBut1 NMAE=AllBut1 MAE).

In Jester, a total of 756 users with ratings outside the -10.5 to 10.5 range were removed. One value was a clear outlier (87.09). There are 17,154 evaluations with at least 20 ratings for a total of 894,584 ratings with an average of 52.2 ratings per evaluation. There are 100 jokes labeled from 1 to 100. The density of ratings is therefore 52.2%. For Eigentaste and STI Eigentaste, we used joke numbers 5,7,8,13,15,16,17,18,19,20 as a standard set rated by all users. In our implementation of the Eigentaste algorithms, we compute the eigenvectors from the training set each time and do not use the eigenvectors provided with the documentation of the data set. In this sense, we penalize Eigentaste as the eigenvectors could be computed over an arbitrarily large number of users without running time penalty. However, we also penalize STI Eigentaste, Bias From Mean, Per Item Average, and $STINx$ schemes as they all stand to benefit from a large number of users. Nevertheless, our experiment show that Eigenstate 2.0 outperforms Per Item Average as reported in [7]. For both Eigenstate and STI Eigenstate, we chose $\eta = 4$. If we divide the typical AllBut1 MAE of 3.75 by the range of values (20.0), we get a NMAE of 0.19 which implies an accuracy similar to that of EachMovie

(EachMovie)	AllBut1 MAE	std. dev.	query cost
Average (STIN0)	0.232	0.001	$O(1)$
Per Item Average	0.223	0.003	$O(1)$
Bias From Mean	0.203	0.001	$O(1)$
$STIN1(\mu_2)$	0.203	0.005	$O(1)$
$STIN2(\mu_2)$	0.198	0.004	$O(1)$
$STIN1(m_2)$	0.195	0.002	$O(1)$
$STIN2(m_2)$	0.194	0.002	$O(1)$
$STIPearson(\mu_2)$	0.194	0.01	$O(m)$
Pearson	0.187	0.01	$O(m)$
$STIPearson(m_2)$	0.166	0.03	$O(m)$

(Jester)	AllBut1 MAE	std. dev.	query cost
Per Item Average	4.06	0.03	$O(1)$
Eigentaste 2.0	3.96	0.04	$O(1)$
Average (STIN0)	3.71	0.05	$O(1)$
Bias From Mean	3.42	0.06	$O(1)$
$STIN2(\mu_2)$	3.37	0.06	$O(1)$
$STIN1(\mu_2)$	3.35	0.06	$O(1)$
$STIN1(m_2)$	3.35	0.06	$O(1)$
$STIN2(m_2)$	3.32	0.06	$O(1)$
STI Eigentaste	3.30	0.07	$O(1)$
Pearson	3.24	0.10	$O(m)$
$STIPearson(\mu_2)$	3.07	0.16	$O(m)$
$STIPearson(m_2)$	3.05	0.18	$O(m)$

TABLE 4. AllBut1 Mean Absolute Error (MAE) of different normalization invariant CFS for the EachMovie and Jester data sets. The complexity of the queries relative to the number of users m is given. For EachMovie, ratings ranged from 0 to 1 in increments of 0.2 whereas for Jester, the range of values is given to be -10.0 to 10.0. Average and standard deviations where computed over 6 trials of at least 100,000 predictions each with training sets including at least 50,000 ratings.

data set. It was already reported [7] that these two data sets appear to lead to the same NMAE even though they are very different.

Overall, our results (see Tab. 4 and Fig. 3) indicate that $STIN2(m_2)$ outperforms Bias From Mean by at least 3%, it outperforms Per Item Average by at least 15%, and is within 4% of the memory-based Pearson scheme while being significantly faster. $STIN2(\mu_2)$ also performs well: only about 2% less accurate than $STIN2(m_2)$. $STIPearson(m_2)$ outperforms Pearson in this study by at least 6%. Because both schemes based on μ_2 and m_2 perform well, we have evidence that STI is a desirable property. As additional evidence, note that STI Eigentaste doesn't use either m_2 or μ_2 and it also outperforms significantly Eigentaste 2.0.

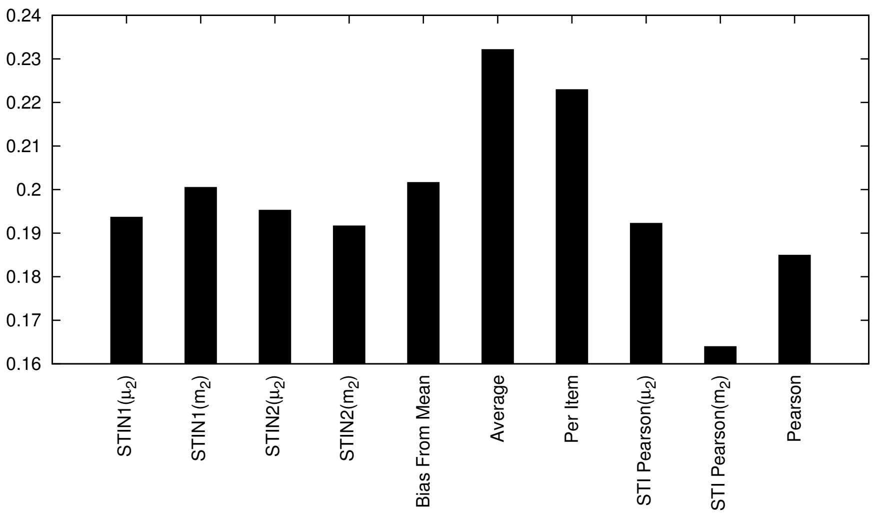
While $STIN1(\mu_2)$ performs as well as Bias From Mean for the EachMovie data set, it lags behind $STIN1(m_2)$ by 4%. On the other hand, in both data sets, $STIN2(m_2)$ performs within 2% of $STIN2(\mu_2)$. Because schemes based on m_2 tend to outperform schemes based on μ_2, it appears that it is better not to penalize frequent raters, that is, not being too *democratic*. Our tests reveal that if a standard item set rated by all users is available, Eigentaste schemes such as STI Eigentaste are competitive.

There is only one instance in our experiment where a STI scheme did not systematically outperform or at least match the performance of the corresponding non-STI scheme: $STIPearson(\mu_2)$ lags behind Pearson on the Each-Movie data set by about 4%. However notice that it outperforms Pearson by about 6% on the Jester data set so that overall $STIPearson(\mu_2)$ and Pearson have comparable accuracy.

The storage requirements for the $STINx$ schemes is $O(xn+1)$ where n is the number of items. For example, the EachMovie data set has at most 1949 items and because we use 32 bits floating point numbers for ratings even though they only have 6 possible values, $STIN2$ has a storage requirement of 15 KB. Similarly, the storage requirement for the Jester data set which has 100 items is under 1 KB. Therefore $STINx$ schemes can easily run on very small devices. Comparatively, memory-based schemes such as Pearson and $STIPearson$ require around 256 KB to store a training set with at least $50,000$ ratings, additional memory might be needed to buffer computations, and since a full database of ratings is needed there are privacy issues. Note that 256 KB is not enough to store the whole EachMovie database but just a sample training set as the whole database in a flat binary file occupies around 22 Megs. As far as the computational cost of the $STINx$ predictors, we can compute regression coefficients in time $O(card(S(u)))$ where $card(S(u))$ is the number of items the active user has rated and typically $card(S(u)) \ll m$ so that the total computation cost is close to $(1+x)n$ operations which is $O(nx)$. As with the memory-based scheme it is possible to reduce the computational burden by requesting predictions over only a subset of ι. Comparatively, memory-based schemes have a computational cost $O(mn)$ and so they are at least two orders of magnitude slower in practice ($m = card(\chi) \gg 1+x$). Eigentaste schemes have roughly the same storage and computational characteristics as $STINx$.

9. APPENDIX: NUMERICAL EXAMPLE

We present an example based on Tab. 2 for the $STIN1(m_2)$ scheme which is one of the most successful in our experiment and also the easiest to implement efficiently. We use the same notation as in the table. The first step to make predictions based on this data set is to compute the $m_2(\chi) = \{m_2(u^{(1)}), m_2(u^{(2)}), \ldots, m_2(u^{(6)})\}$ from $\chi = \{u^{(1)}, u^{(2)}, \ldots, u^{(6)}\}$ as it was done in Tab. 3. This can be done offline irrespective of the current user.



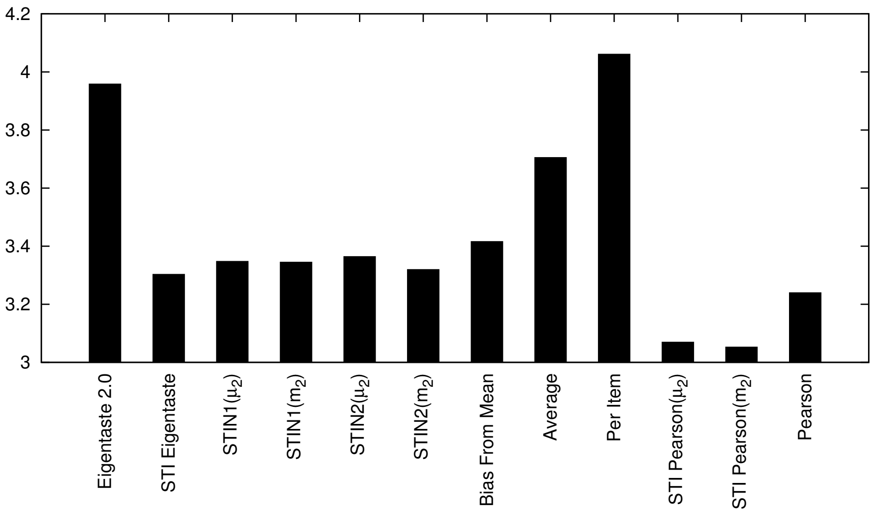


FIGURE 3. AllBut1 Mean Absolute Error (MAE) on the EachMovie (top) and Jester (bottom) data sets (see Tab. 4).

As an example, we show how to compute $m_2(u^{(1)})$ and $m_2(u^{(6)})$. Given $u^{(1)} = (\mathit{unrated}, 5, \mathit{unrated}, 3)$, we have that $\overline{u^{(1)}} = \frac{5+3}{2} = 4$ hence the *bias from mean* is given by $u^{(1)} - \overline{u^{(1)}} = (\mathit{unrated}, 1, \mathit{unrated}, -1)$. Finally, $\left\| u^{(1)} - \overline{u^{(1)}} \right\| = \sqrt{\frac{1^2+(-1)^2}{2}} = 1$ and thus, $m_2(u^{(1)}) = (0, 1, 0, -1)$. For $m_2(u^{(6)})$, the average is 3 so that the bias from mean is $(-2, 0, 2, \mathit{unrated})$ and the l_2 norm is $\sqrt{8/3} = 2\sqrt{2/3}$ hence, $m_2(u^{(6)}) = \sqrt{3/2}(-1, 0, 1, 0)$.

Computing $\mu_2(u^{(1)})$ and $\mu_2(u^{(6)})$ would be similar except that we must also divide by the number of ratings each evaluation contains: 2 and 3 respectively. Hence, $\mu_2(u^{(1)}) = (0, 1/2, 0, -1/2)$ and $\mu_2(u^{(6)}) = \sqrt{1/6}(-1, 0, 1, 0)$. Note that $\mu_2(u^{(6)})$ is scaled down because it contains more ratings.

Recall that $STIN1(m_2)$ predictions are built from two vectors: $\mathbf{v}^{(0)}$ and $\mathbf{v}^{(1)}$. We have $\mathbf{v}^{(0)} = \mathbf{1}$ and we must compute $\mathbf{v}^{(1)}$ using the formula:

$$\mathbf{v}_i^{(1)} = \frac{1}{card(S_i(\chi))} \sum_{u \in S_i(\chi)} m_2(u)_i.$$

The formula requires us to know the sets $S_i(\chi)$ for all $i \in \iota$. By inspection, we have $S_1(\chi) = \{u^{(2)}, u^{(4)}, u^{(5)}, u^{(6)}\}$, $S_2(\chi) = \{u^{(1)}, u^{(3)}, u^{(5)}, u^{(6)}\}$, $S_3(\chi) = \{u^{(2)}, u^{(4)}, u^{(5)}, u^{(6)}\}$, $S_4(\chi) = \{u^{(1)}, u^{(3)}, u^{(5)}\}$, and so $card(S_1) = 4$, $card(S_2) = 4$, $card(S_3) = 4$, and $card(S_4) = 3$. Hence, by using Tab. 3, we have

$$\mathbf{v}^{(1)} = (\frac{-2-\sqrt{3/2}}{4}, \frac{2+\sqrt{2}}{4}, \frac{2+\sqrt{3/2}}{4}, \frac{-2-\sqrt{2}}{3},) \approx (-0.81, 0.85, 0.81, -1.14)$$

How do we use this in practice? Let the evaluation of the current user be $u = (2, 1, unrated, unrated)$. In this case $S(u) = \{1, 2\}$. We first compute the average $\bar{u} = \frac{2+1}{2} = \frac{3}{2}$ so that $u - \bar{u} = (\frac{1}{2}, \frac{-1}{2}, unrated, unrated)$, and thus $P(u) \approx \frac{3}{2} + \alpha(-0.81, 0.85, 0.81, -1.14)$ and we must solve for α by regression so as to minimize the residual energy of $u - P(u)$. A convenient formula is

$$\alpha = \frac{\langle \mathbf{v}^{(1)} - \overline{\mathbf{v}^{(1)}_{|S(u)}}, u - \bar{u} \rangle}{\langle \mathbf{v}^{(1)} - \overline{\mathbf{v}^{(1)}_{|S(u)}}, \mathbf{v}^{(1)} - \overline{\mathbf{v}^{(1)}_{|S(u)}} \rangle_{S(u)}}.$$

We have that $\mathbf{v}^{(1)}_{|S(u)} \approx (-0.81, 0.85, unrated, unrated)$ and so $\overline{\mathbf{v}^{(1)}_{|S(u)}} \approx 0.04$. Doing some arithmetic, we get $\langle \mathbf{v}^{(1)}_{|S(u)} - \overline{\mathbf{v}^{(1)}_{|S(u)}}, \mathbf{v}^{(1)}_{|S(u)} - \overline{\mathbf{v}^{(1)}_{|S(u)}} \rangle \approx 1.38$ whereas $\langle \mathbf{v}^{(1)}_{|S(u)} - \overline{\mathbf{v}^{(1)}_{|S(u)}}, u - \bar{u} \rangle \approx -0.83$. Hence, $\alpha \approx \frac{-0.83}{1.38} \approx -0.60$. So that, $P(u) \approx \frac{3}{2} + 0.60(-0.81, 0.85, 0.81, -1.14) \approx (2, 1, 1, 2.2)$.

Therefore the scheme $STIN1(m_2)$ predicts that this new user is going to give ratings of 1 and 2.2 to items 3 and 4 respectively. Comparatively, the Per Item Average scheme would predict $\frac{9}{2}$ and $\frac{7}{3}$ respectively.

Acknowledgement. The author would like to thank Compaq Research and professor Ken Goldberg for convenient access to the EachMovie and Jester databases respectively. He would also like to thank Sean McGrath for providing the script which retrieved ratings from the web site Amazon. The author is supported by a Canadian NSERC/CRNSG grant.

Source code and scripts necessary to reproduce the experimental results are freely available from the author.

References

[1] Amrani, M. Y. E., S. Delisle, and I. Biskri (2001) , Coping with Information Retrieval Problems on the Web: Towards Personal Web Weaver Agents . In: *IC-AI'01*. pp. 1225–1231.

[2] Billsus, D. and M. Pazzani (1998) , Learning collaborative information filterings . In: *AAAI Workshop on Recommender Systems*.

[3] Breese, J. S., D. Heckerman, and C. Kadie (1998) , Empirical Analysis of Predictive Algorithms for Collaborative Filtering . Technical report, Microsoft Research.

[4] Canny, J. (2002) , Collaborative Filtering with Privacy via Factor Analysis . In: *SIGIR 2002*.

[5] Drineas, P., I. Kerenidis, and P. Raghavan (2002) , Competitive recommendation systems . In: *Proc. of the thiry-fourth annual ACM symposium on Theory of computing*. pp. 82–90.

[6] Ghahramani, Z. and M. Jordan (1994) , Learning from incomplete data . Technical Report 108, MIT Center for Biological and Computational Learning.

[7] Goldberg, K., T. Roeder, D. Gupta, and C. Perkins (2001) , Eigentaste: A Constant Time Collaborative Filtering Algorithm . *Information Retrieval* **4**(2), 133–151.

[8] Herlocker, J., J. Konstan, A. Borchers, and J. Riedl (1999) , An Algorithmic Framework for Performing Collaborative Filtering . In: *Proc. of Research and Development in Information Retrieval*.

[9] Karypis, G. (2000) , Evaluation of Item-Based Top-N Recommendation Algorithms . Technical Report 00-046, University of Minnesota, Department of Computer Science.

[10] Pennock, D. M. and E. Horvitz (1999) , Collaborative Filtering by Personality Diagnosis: A Hybrid Memory- and Model-Based Approach . In: *IJCAI-99*.

[11] Pennock, D. M., E. Horvitz, and C. L. Giles (2000) , Social Choice Theory and Recommender Systems: Analysis of the Axiomatic Foundations of Collaborative Filtering . In: *AAAI-2000*. pp. 729–734.

[12] Resnick, P., N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl (1994) , Grouplens: An open architecture for collaborative filtering of netnews . In: *Proc. ACM Computer Supported Cooperative Work*. pp. 175–186.

[13] Salton, G. and C. Buckley (1998) , Term-weighting approaches in automatic text retrieval . *Information processing and management* **24**(5), 513–523.

[14] Sarwar, B. M., G. Karypis, J. A. Konstan, and J. Riedl (2001) , Item-based Collaborative Filtering Recommender Algorithms . In: *WWW10*.

[15] Sarwar, B. M., G. Karypis, J. A. Konstan, and J. T. Riedl (2000) , Application of Dimensionality Reduction in Recommender System - A Case Study . In: *WEBKDD '00*. pp. 82–90.

[16] Vucetic, S. and Z. Obradovic (2000) , A Regression-Based Approach for Scaling-Up Personalized Recommender Systems in E-Commerce . In: *WEBKDD '00*.

[17] Weiss, S. and N. Indurkhya (2001) , Lightweight Collaborative Filtering Method for Binary Encoded Data . In: *PKDD '01*.

[18] Yu, K., X. Xu, J. Tao, M. Ester, and H.-P. Kriegel (2002) , Instance Selection Techniques for Memory-Based Collaborative Filtering . In: *SDM '02*.

[19] Yu, K., X. Xu, J. Tao, M. E. Kri, and H.-P. Kriegel (2003) , Feature Weighting and Instance Selection for Collaborative Filtering: An Information-Theoretic Approach . *Knowledge and Information Systems* **5**(2).