Dynamic Updating of Online Recommender Systems via Feed-Forward Controllers

Valentina Zanardi
Dept. of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
V.Zanardi@cs.ucl.ac.uk

Licia Capra
Dept. of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
L.Capra@cs.ucl.ac.uk

ABSTRACT

Recommender systems have become an essential software component of many online businesses, supporting customers in finding the items (e.g., books on Amazon, movies on Netflix, songs on Last.fm) they are interested in. Key to their success is the level of accuracy they achieve: the more precisely they can predict how much a customer will enjoy an item, the higher the profit that the business will make (e.g., in terms of more purchases). In quantifying the accuracy of recommender systems, the evaluation methodology followed by researchers has so far neglected an important aspect: that these businesses grow continuously over time, both in terms of users and items. The data structures used by the recommender system to compute predictions become stale and thus have to be updated regularly. Intuitively, the more often the data structures are being updated, the higher the accuracy achieved, but the higher the computational cost afforded, because of the extremely large volume of data being handled. System administrators often perform the update at fixed intervals of time (e.g., weekly, fortnightly), in an effort to balance accuracy versus cost. We argue that such an approach benefits neither accuracy nor cost, as businesses do not grow linearly in time, thus risking the fixed update interval to be at times too coarse (with negative impact on accuracy), and at other times too fine grained (with negative impact on cost). We thus advocate for a self-monitoring and self-adaptive approach, whereby the system monitors its own growth over time, estimates the loss in accuracy it would endure if an update were not being performed based on the observed growth, and dynamically decides whether the benefit of performing an update (accuracy) outweighs its computational cost. Using real data from the Bibsonomy website, we demonstrate how this simple technique enables system administrators to transparently balance these two conflicting requirements.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*Performance Measures*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information Filtering*

General Terms

Performance, Experimentation

Keywords

Recommender Systems, Self-Monitoring, Self-Adaptation

1. INTRODUCTION

Digital distribution has dramatically changed retailers' business models: while traditional retailers carry only a limited number of items (in particular, those that have the best chance to sell) due to the limited space where they stock them, Internet businesses are no longer bound by the same physical constraints, so that a much wider variety of items can be offered from the so called 'long tail'[2]. As a result, while a traditional bookshop can hardly be expected to sell more than 100,000 different titles, an online service such as Amazon.com can offer its costumers millions of different products. The difference becomes even sharper now that producing the content itself is within easy reach of almost anybody: consider the difference of choice between traditional broadcast or cable TV, and sites like YouTube.com. Providing people with a massive choice is detrimental, if that means they have to browse through thousands, or even millions, of potentially relevant items. Rather, filters must be used to connect supply and demand, making it easier for users to find the particular items that they would enjoy. Recommender systems have emerged in the last decade as powerful tools that businesses can rely upon, to help their customers navigate large databases according to their own interests. The engine that underlies most of these systems is Collaborative Filtering (CF) [10], which is an automated means of ranking content based on the observation that users' tastes and preferences are quite stable; by first identifying users with similar preferences, CF recommends items that people with similar tastes have enjoyed.

The research community has been very active in devising ever more accurate recommender algorithms, where accuracy is usually measured as the difference between how much the engine predicts a user will enjoy an item (for example, in a star rating scale $[1, 5]$), and how much they actually enjoy it. The evaluation methodology that has been followed so far by the research community follows this pattern: the available dataset (e.g., set of triplets $user, item, rating$) is split in two parts, respectively called training set and test set; the CF algorithm is first trained on the train set, then asked to make predictions for all items in the test set. This evaluation methodology falls pray to a common mistake: it treats the prediction problem as a static, one-shot process, while in practice businesses grow over time (i.e., new users join the system, new items are being added, and new ratings stored), so that the train/test process actually operates in a cyclical manner. Intuitively, the more frequently the training process is repeated, the more accurate the predictions will be, as more information is leveraged upon; however, because of the size of real datasets (with hundreds of thousands of users, items, and millions of ratings), training a CF algorithm is a very expensive operation, often requiring many hours or even days to complete, and cannot thus be repeated at will. Standard practice requires system administrators to setup iterative updates, which are repeated at regular intervals (e.g., weekly or fortnightly [18]) and whose frequency attempts to find the right balance between accuracy and computational cost.

We argue that setting fixed update intervals benefits neither accuracy nor cost. This is because businesses do not grow uniformly over time: if the interval is set to high frequency, accuracy will be high, but so will be cost; at times where growth is slow, the cost in re-training the CF algorithm may bring no tangible gain to the business (in terms of accuracy improvement). Vice versa, if the interval is set to low frequency, cost will be moderate, but accuracy may be severely compromised, especially in periods of steep growth (where performing an update more means, for example, being able to provide customers with fresher recommendations of newly released items). A question thus arises as to how to set the update frequency of the online recommender system, so to strike the right balance between accuracy and cost.

In the last decade, the software engineering community has been actively researching solutions to support the design and development of self-adaptive software systems [7]. Rather than relying on ad-hoc approaches to self-adaptation, built out of speculative assertions about the deployed system, the community has been researching tools and techniques in support of a systematic development of complex self-adaptive software system, with the aim to maintain non-functional requirements, such as robustness, availability, fault-tolerance, and security, despite an ever-changing operating environment. State-of-the-art solutions have been developed based the notion of *feed-back loops*: data from the executing system is collected, the accumulated data is subsequently analysed, a decision is made about whether to alter the behaviour of the system, and such decision is then enacted; this four-step process repeats itself continuously over time, so moving traditional design-time decisions at run-time, to be able to control the dynamic behaviour of the system [5].

In this paper, we argue that a deployed recommender system should be treated as an instance of a self-adaptive system: system administrators should not be asked to decide, at design time, the frequency with which the system should update itself, nor should they accrue ad-hoc rules for dynamic adaptation which would give no guarantee in terms of recommendation accuracy. Rather, systematic software engineering solutions for self-adaptive systems should be called upon. In particular, we draw inspiration from *feed-forward* control theory [6] to engineer a system that self-monitors the growth speed of the dataset since its last training process, and uses this information to estimate the accuracy loss that not performing an update would entail; it then decides whether the benefit of performing such update outweighs its cost, in practice self-adapting the system update frequency. Businesses (and, more precisely, their system administrators) only have to set a threshold, representing the maximum accuracy loss they are willing to endure before an update is carried out.

The reminder of the paper is structured as follow: we first provide a brief overview of recommender systems research, highlighting the drift between theory and practice in terms of evaluation methodology (Section 2). Using a large dataset from a real-world system, we demonstrate how businesses grow at non-linear speed, and quantify the tension between accuracy and cost, with detrimental consequences when setting fixed update intervals (Section 3). We then illustrate how the recommender system can be seen as instance of a self-adaptive system (Section 4.1), review the literature in the area of software engineering for self-adaptive systems (Section 4.2), and propose to adopt a very light-weight self-monitoring and self-adaptive technique, grounded on feed-forward control theory, which relieves system administrators from deciding upfront on a fixed update frequency, while dynamically and automatically adjusting it based on observed actual growth (Section 4.3). We return to the real-world system to evaluate the trade-off between accuracy and cost that this technique entails (Section 5), before concluding the paper with our future directions of research (Section 6).

2. RECOMMENDER SYSTEMS - A BRIEF OVERVIEW

Over the last decade, recommender systems have gained a dominant presence on the web, encompassing e-commerce portals (e.g., Amazon.com), movie portals (e.g., Netflix.com), music web sites (e.g., Last.fm), and the like. Collaborative Filtering (CF) [10] has emerged as the dominant algorithm behind deployed recommender systems [4]; as the name suggests, it uses the collaborative effort of an entire community of users to help each individual sift through the endless amount of items (be them books, movies or songs). The grounding assumption of CF is that historically like-minded individuals will also share similar tastes in the future. Measuring similarity thus plays a central role; only the top-k most similar users are allowed to contribute their ratings, and each contribution is weighted according to the specific degree of similarity the neighbour shares with the current user. A wide variety of metrics exists to quantify users' similarity [14]; for example, if we represent each user a as a vector r of its past ratings $r_{a,i}$ over a set of items i, then the similarity $w_{a,b}$ between two users a and b can be computed as the cosine of the angle between their profile vectors:

$$w_{a,b} = \sum_{i=1}^{N} \frac{r_{a,i} * r_{b,i}}{\sqrt{\sum_{i=1}^{N} r_{a,i}^2}\sqrt{\sum_{i=1}^{N} r_{b,i}^2}} \quad (1)$$

For a given user a, the CF algorithm can then compute the predicted rating $p_{a,i}$ of how much a will enjoy a new item i as the weighted average of ratings given to item i by the top-k most similar users to a:

$$p_{a,i} = \frac{\sum_k r_{u_k,i} * w_{a,u_k}}{\sum_k w_{a,u_k}} \quad (2)$$

In order to compute the above prediction, the recommender system relies on a two-step process: a matrix of user-user similarities is computed offline, containing similarity values for each pair of users in the system (formula 1), based on all the ratings they entered thus far; whenever recommendations have to be computed for a given user (e.g., whenever the user logs into the business website), predictions are computed for all items not yet consumed (e.g., bought) by the user (formula 2), and those with the highest predicted ratings are finally recommended.

In recent years, there has been an explosion of websites where explicit numerical ratings are actually not available, and users' preferences are expressed in terms of descriptive tags they associate to items (e.g., photos on Flickr.com, videos on YouTube.com, URLs on delicious.com). The two-dimensional space of $(user, item)$ over which traditional CF operates, has then become a three-dimensional one of $(user, item, tag)$, with pairwise similarities computed offline over both users (to identify top-k recommenders) and tags (to identify top-k related topics) [22].

The recommender system research community has been active in proposing ever more accurate algorithms, for example, by varying similarity metrics [20], reasoning upon contextual information [1], manipulating raw data using matrix factorisation [12, 17]. A common characteristic to all these efforts is the offline pre-processing of the ratings entered thus far in the system, to produce support data structures (e.g., similarity matrices) to be used online to finally compute recommendations. In quantifying and comparing the accuracy achieved by these techniques, the following evaluation methodology has been used throughout the research community: a dataset of user/ item (or user/item/tag) has been split into a training set and a test set. The training set has been used to compute, *once and for all*, the supporting data structures; a prediction is then computed for each item in the test set, and the difference between such prediction and the actual rating measured. By aggregating these differences, accuracy of the various techniques can be quantified and compared. This static, one-shot evaluation methodology is not representative of what happens in *deployed* recommender systems: in particular, the training set grows over time, with new users, items, ratings and tags entering the system over time. As recently shown [13], this temporal dimension plays an important role in quantifying *actual* accuracy of deployed recommender system algorithms. Intuitively, the more up-to-date the pre-processed data structures are, the better the achieved accuracy. However, because of the size of the datasets at hand, very frequent (e.g., daily) updates are often not affordable given the available computational resources; system administrators thus set the updating processes to run offline at regular intervals (e.g., weekly, fortnightly), as can be afforded by available resources. As we demonstrate next, setting a fixed update frequency cannot solve the problem of balancing accuracy and cost, as the growth of the dataset is not constant over time.

3. DOMAIN PROBLEM ANALYSIS

3.1 Dataset

In order to quantify the effect of data growth on the performance of a deployed recommender system, we conducted an experiment using real usage data from the Bibsonomy website. Bibsonomy (`www.bibsonomy.org`) is a social tagging website that aims at promoting the sharing of both scientific references and general URLs. Rather than expressing preferences via numeric ratings (for example, in a scale $[1, 5]$), users of this website attach tags to items, to describe their interests (we refer to the association $(user, item, tag)$ as 'bookmark'); Bibsonomy thus requires the application of the CF engine to the three dimensional problem of users/ items/tags. We downloaded a snapshot of the website in June 2009, with bookmarks made between February 1995 and June 2009. We first preprocessed the dataset to remove all non-alphabetical and non-numerical tags; to further remove noise in the data, we used the p-core preprocessing strategy [9], where users, items and tags are iteratively removed from the dataset to produce a smaller but denser subset where each user, item and tag occurs in at least $p = 2$, bookmarks. We thus obtained a dataset with $1,360$ users, $23,649$ items, $11,668$ tags, and $72,741$ bookmarks.

We then looked at the the growth of the dataset over time. Figure 1 plots the total number of users, items, tags and bookmarks belonging to the dataset, as observed at monthly intervals. As shown, there exists an initial period (February 1995-February 2005) after the system creation in which growth is very slow across all of these dimensions; this is typical of a system in its early stages (e.g., when its first prototype version has just been released and only a few users know and use it). However, as the system gains popularity, growth rate speeds up, and with different patterns across different dimensions. As expected, bookmarks have the fastest growth, followed by items, tags and finally users (i.e., new bookmarks most probably refer to an existing user tagging a new item with existing tags); indeed, items generally grow faster than both users and tags (e.g., as new content is being uploaded on the website). There are however also time periods when tags grow faster, especially at

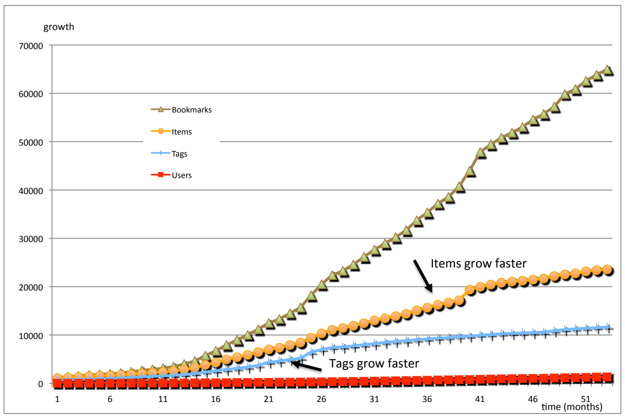


Figure 1: Growth of Users, Items and Tags in the Bibsonomy Dataset

the initial stages of the system, when the vocabulary of tags is still being constructed with new terms. The question we face now is whether, and if so to what extent, these different growth rates impact the accuracy of the recommender system.

3.2 Method of Assessment

To answer the above question, we have performed the following experiment. We have isolated three different time periods, of one-month duration each, during which users, tags and items grow at different speed. In particular, as summarised in Table 1, time period T_3 sees both users, items and tags grow according to the overall average growth rate of the dataset, while T_1 sees tags grow at a much higher rate (17% growth rather than the usual 1-2%), and T_2 sees items grow at a much higher rate than normal (11% growth rather than the usual 2-3%).

Time Period	User	Tag	Item
T_1: Oct06-Nov06	6%	**17%**	2%
T_2: Jan08-Feb08	6%	1%	**11%**
T_3: Jun07-Jul07	6%	2%	3%

Table 1: Growth Rates of Data Snapshots

For each of the above periods T_i, we have then conducted the following experiment (see Figure 2). We have constructed two different training/test splits of the dataset: in the former ($T_i(1)$), all data entered up to the *beginning* of T_i is used in the training process; in the latter ($T_i(2)$), all data entered up to the *end* of T_i is used in the training process (in other words, the latter training set contains a full month worth of data more than the former). Each training set is fed in input to a CF algorithm [22], so that matrices of users' and tags' similarities can be pre-computed. We then test each instance as follow: we construct three different test sets of 1,500 items each, so that the first contains 25% tests referring to either items or tags which first appeared during T_i, while 75% of them refer to items/tags which appeared before T_i; the second contains a 50-50 split, and the latter contains a 75-25 split. We then compute the *performance loss* between the two CF instances, one operating on $T_i(1)$ and one operating on $T_i(2)$. Performance has been measured in terms of precision and recall, following standard recommender systems practice:

$$precision = \frac{|relevantItems| \cup |retrieveditems|}{|retrievedItems|}$$

$$recall = \frac{|relevantItems| \cup |retrieveditems|}{|relevantItems|}$$

While the latter CF instance relies on up-to-date data structures computed based on all available data up to this point ($T_i(2)$), the former relies on stale data ($T_i(1)$), as a full month worth of activity has passed without an update. In particular, the more information has been inserted during the interval, the more severe we expect its impact to be on performance. We quantify this performance loss next.

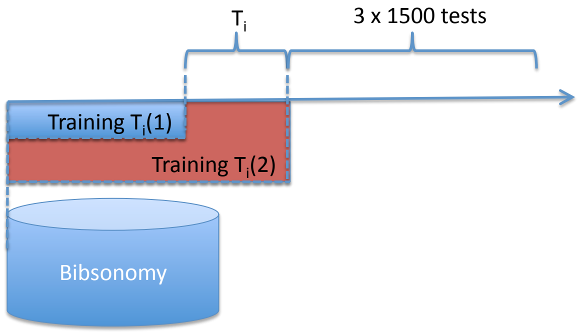


Figure 2: Assessment Methodology

3.3 Performance Analysis

We have compared performance loss across the three different time periods previously identified, to quantify the impact that various growth rates have. Results are reported in Table 2 (as precision and recall loss was the same in percentage terms, we report one value only). Two observations can be made: first, as expected, the more tests refer to data which first entered the system during the one month period, the higher the loss. More importantly, for each test set, such loss is notably different across the three time periods; let us focus on the column labelled 25-75 (i.e., where a quarter of the tests refers to data entered during T_i): as shown, the loss is as little as 3% when growth is average across users, items and tags (T_3), while it is *four times higher* (i.e., 12%) in T_2, when users' growth peaks, and *eight times higher* (i.e., 24%) in T_1, when tags' growth is fastest. Indeed, tags' growth (row T_1) seems to have the most detrimental effect on precision and recall, no matter the composition of the test set (i.e., across all columns).

To cater for data growth of different intensity, a system administrator might set the update frequency to very fine-grained intervals. However, the cost of each update is non-negligible, as it requires the computation of pairwise similarity across all users U and tags T in the system ($U*(U-1)/2+T*(T-1)/2 \approx O(U^2+T^2)$). In periods of average growth (i.e., T_3), such expensive computations represent a big waste of resources, bringing very little improvement to precision and recall. This confirms our hypothesis that setting a fixed update interval is not a suitable strategy, as this may cause either detrimental loss of accuracy (e.g., when the system is experiencing high growth rate) or unnecessary computational cost (during average growth). A self-monitoring and self-adaptive technique is thus called upon, that constantly monitors data growth, and dynamically decides to re-train the system should the expected gain in recommendation efficacy (i.e., precision and recall) outweigh the updating cost. We present this dynamic update methodology next.

Time Period	25-75	50-50	75-25
T_1	24%	32%	45%
T_2	12%	20%	23%
T_3	3%	10%	14%

Table 2: Precision/Recall Loss After 1 Month

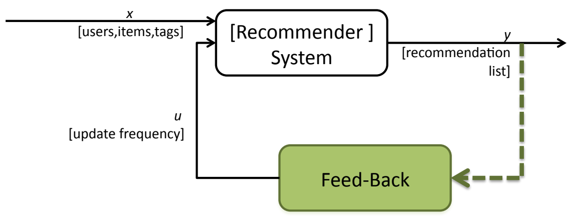


Figure 3: Recommender System with Feed-Back Control

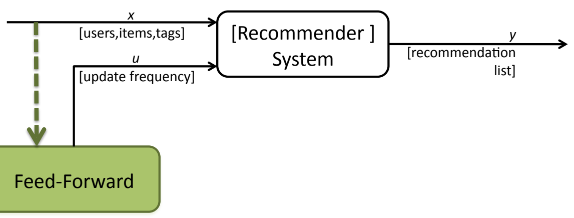


Figure 4: Recommender System with Feed-Forward Control

4. DYNAMIC UPDATE METHODOLOGY

4.1 Recommender Systems as Self-Adaptive Systems

The recommendation engine can be seen as a system with dynamic behaviour (Figure 3): as new data $x(t)$ (be that new users, items, or tags) come in over time, new outputs $y(t)$ are being observed (that is, new recommendation lists). Such outputs should follow some desirable behaviour (i.e., maximise precision and recall); a *controller* could be put in place, so to continuously monitor the produced output $y(t)$, quantify whether the observed behaviour follows the expected one (that is, whether the observed precision/recall are as high as required), and should this not be the case, manipulate some system input variable u (that is, the recommender update frequency), thus inducing the system to behave as desired. This is an example of *feed-back* (or closed-loop) *control system* [6]: all is required from system administrators is to define what the *desirable behaviour* is (that is, what is the maximum error μ they tolerate from the recommender engine); once this is set, the feed-back controller takes care of the self-monitoring and self-adaptation process. Control theory has been widely studied by the software engineering community, and in particular within the area of self-adaptive software systems. We review the state-of-the-art in this field next, before presenting the technique we propose to address the problem of dynamic updating of online recommender systems.

4.2 Software Engineering for Self-Adaptive Systems

Feed-back control systems have been widely explored within the area of self-adaptive software systems as they provide the key mechanism (i.e., feedback) to tackle challenges of predictability of the deployed system, and cost-effectiveness entailed by its adaptation. In order to guarantee certain non-functional properties of a dynamic software system (e.g., performance, stability, accuracy), the feedback cycle of *collection* of relevant data from the environment, *analysis* of the collected raw data to infer the system's state, *decision* about how to adapt to reach a desirable state, and *action* to implement such transition is called upon [7]. In so doing, the uncertainty that is inherent in this kind of systems can be modelled (e.g., as noise in some system variables, or imperfections in the models of the environment) and reasoned upon, thus effectively reducing its effects.

This generic feed-back loop has been mostly engineered in one of two ways: the Model Identification Adaptive Control (MIAC) [21] scheme, and the Model Reference Adaptive Control (MRAC) [3]. In order to decide whether the current controller needs adjustment, the former observes and reasons upon the inputs and outputs of the system only; self-optimisation components deployed in performance-tuning and resource-provisioning scenarios often implement this kind of loops [15, 16]. The latter, instead, also requires a model describing the desired behaviour of the system, so that adaptation choices do not only depend on observed inputs/ outputs, but also on how much the actual outputs deviate from those of the reference model. These schemes have been mainly used in flight-control domains [3, 8], where a reference model exists, and where variations in the controller are not expected to be substantial.

Although MIAC schemes would appear to be suitable for the dynamic adaptation of *recommender* systems, their fundamentally *closed-loop* nature limits their applicability. With reference to Figure 3, the output of the recommender system $y(t)$ cannot be *immediately* used by the controller to measure prediction error and thus adjust the update frequency u: this is because $y(t)$ simply contains recommendations of items the users may enjoy; it is only later in time, *if* and *when* users consume the items and leave some feed-back, that prediction error can be measured (and u adjusted if necessary). However, with humans involved in the whole recommendation process, we cannot control whether this is going to happen, and with what delay.

In such scenario, a *feed-forward control system* represents a better alternative: as shown in Figure 4, rather than monitoring the output $y(t)$ of the system, the controller now monitors its input $x(t)$ (that is, the dataset growth) instead, and makes decisions as to whether/how to adjust input parameter u based on this information alone. To operate reliably, feed-forward control systems rely on the assumption that the *effect of the output* $y(t)$ can somehow be *predicted* from the input $x(t)$, and that such input-output dependency is unchanging with time t. Under this assumption, system administrators can still reason in terms of maximum tolerable error μ (the effect of $y(t)$), and the controller simply transforms this value into the corresponding input $x(t)$ that would cause such an effect.

4.3 Feed-Forward Controller for Dynamic Recommender System Updating

We have engineered our dynamic update methodology based on feed-forward control theory. A controller component is deployed at run-time to monitor the growth of the recommender system data $x(t)$, in terms of percentage of new users, items and tags being entered. When the growth exceeds a certain threshold χ, the controller launches a system update process, effectively adjusting the update frequency continuously over time. As system administrators reason in terms of tolerable error μ, and not growth rate χ, we need to relate μ back to χ. Recall the assumption that there exists a linear function f, unchangeable over time, for which $\chi \approx f(\mu)$; this function can be empirically derived using the very same methodology presented in Section 3.2: using historical data collected from the deployed recommender website, we compute the actual loss in precision/recall (that is, μ) that is observed on the reference system, should the recommender engine not be updated. We do so for an overall data growth (aggregate of users, items and tags) of χ%, $\forall \chi \in [1, maxGrowth], \chi \in \mathbb{N}$; should $\chi > maxGrowth$, the system would undergo a forced update, assuming the error caused by stale data would be too high. Using this empirical approach to quantify f, we look at data growth only, without making any assumption about the composition of the test set; indeed, following the methodology described in Section 3.2, error is measured, in each time period, under three different test sets with 25-75, 50-50, and 75-25 splits between old and new test data. This approach provides upper/lower bound estimates of prediction error which may be quite far apart from each other; a more accurate methodology could look at the actual system usage in the past, to predict the most likely composition of the test set (e.g., for example, using a Kalman filter [11]), thus narrowing down the predicted error *range*. We leave this open for future experimentation, and proceed with an evaluation of the trade-off between recommendation accuracy and computational cost, as achieved by this feed-forward controller, on the Bibsonomy website.

5. EVALUATION

In this section, we evaluate the performance of the proposed adaptive update technique (subsequently referred to as *Adaptive*), on the Bibsonomy dataset presented in Section 3.1. Performance has been measured in terms of: *cumulative prediction error* over the period of experimentation and *total number of updates* over such period; the two metrics thus represent accuracy and cost respectively. We have compared our *Adaptive* technique against two benchmark strategies, which perform updates at fixed intervals of time: *Weekly*, which we expect to have high accuracy but also high cost, and *Monthly*, which we expect to have low cost, but also detrimental accuracy loss. The frequency of these fixed strategies has been chosen in relation to the dataset at hand; it is worth noting that Bibsonomy's growth is slow when compared to the most popular folksonomic websites (e.g., del.icio.us), for which more aggressive strategies (e.g., *Daily*) should be chosen to achieve high accuracy. The results we present next stand all the same, as our *Adaptive* technique makes no assumption at all about the actual data growth.

As a preparatory step, we have followed the empirical technique described in the previous section to estimate f, and thus the relation between data growth χ and error μ. Figure 5 depicts the prediction error (i.e., percentage loss in precision/recall), for data growth between 1% and 10% (we stopped at $\chi = 10\%$ growth as the error μ measured in such case is sufficiently high to expect an update to be always necessary, that is, $\mu \leq f(10\%)$). Note that the ten time periods $T_i, i \in [1, 10]$, used to measure prediction error, cover ten different and potentially unrelated time periods, each of different duration (i.e., $i < j \nRightarrow T_i \subseteq T_j$); these periods T_i were selected so to exhibit an overall growth rate $\chi = i\%$.

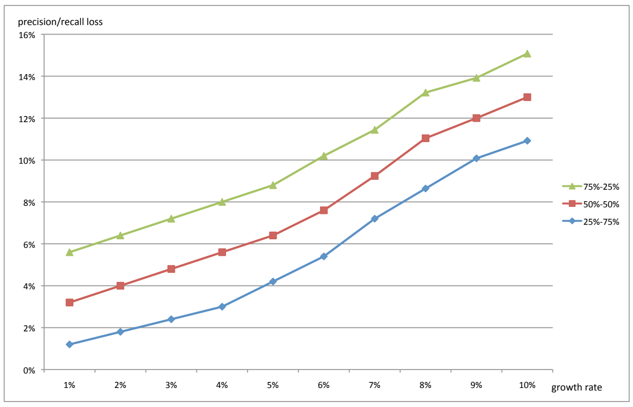


Figure 5: Performance Loss μ on Bibsonomy for Different Data Growth Rates $\chi \in [1, 10]$

Error has been measured, in each time period, under three different test sets of 1500 queries each, using 25-75, 50-50, and 75-25 splits between old and new test data. For example, as shown in Figure 5, a growth rate $\chi = 3\%$ entails a prediction loss μ in between 2.5% and 7.5%, depending on the test set composition.

Having completed this preparatory step, we now proceed with the actual evaluation. Our goal is to demonstrate that the *Adaptive* strategy has a computational cost which is comparable to the *Monthly* strategy (the cheapest approach in this experiment setup, but also the least accurate) and a recommendation efficacy comparable to the *Weekly* strategy (which is the most effective but also the most computationally expensive). The experiment has unfolded as follow: we considered an 81-week long period between October 2006 and March 2008 (note that this period includes T_1, T_2 and T_3 previously analysed in Section 3.2); the system starts in an up-to-date state (that is, all data entered in Bibsonomy prior to October 2006 has been processed and used to train the current recommender engine). We then run four recommender systems in parallel: one that is being regularly updated at the end of each week (thus 80 times in total), one that is being regularly updated at the end of each month (thus 18 times in total), and two using our adaptive technique, *Adaptive 4%* which updates the system when the predicted error μ exceeds 4%, and *Adaptive 6%* which updates the system when the predicted error μ exceeds 6%. The number of system updates that these strategies perform varies, as it depends on data growth over the 81-week long experiment; the feed-forward controller estimates how much new data χ must enter the system, before running an update, using the pre-computed $\chi - \mu$ relation (Figure 5), with reference to the 50-50 test split; in this case, the system must grow of $\chi = 2\%$ overall to expect $\mu = 4\%$, and of $\chi = 5\%$ overall to expect $\mu = 6\%$. Note that a more resource-conservative controller would have used the 25-75 split to estimate χ (for which more data must enter the system before causing the expected error, thus having less frequent updates overall), while an accuracy-conscious controller would have used the 75-25 split (for which little new data is enough to cause the maximum tolerable error, thus calling for more frequent updates overall). As previously

Technique	Number of updates
Weekly	80
Adaptive $\mu = 4\%$ $(\chi = 2\%)$	**29**
Adaptive $\mu = 6\%$ $(\chi = 5\%)$	**13**
Monthly	18

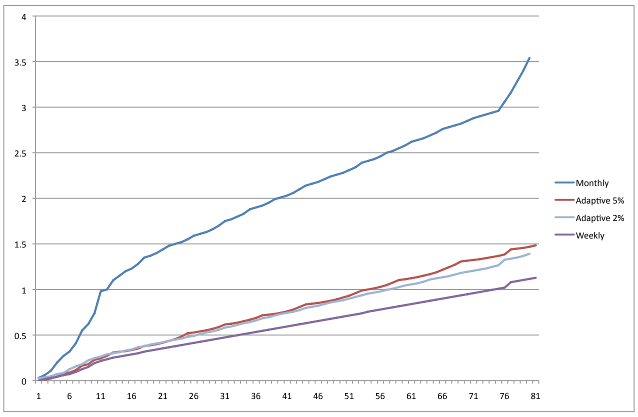
Table 3: Number of System Updates Performed by Each Strategy

mentioned, we leave the problem of predicting actual system usage (and thus composition of the test set) for future research. Intuitively, relatively small websites can rely on accuracy-conscious controllers, as their size can afford more frequent updates, while also gaining the most from them (for new systems, having an update more can make the difference between knowing nothing about the preferences of a user, and knowing enough to make the recommender engine compute accurately). Vice versa, large websites can rely on resource-conscious controllers, under the assumption that most recommendations will be computed for users whose preferences are somehow known. We used the middle approach (50-50), as Bibsonomy is a medium-sized website, with a slow-growing users' base.

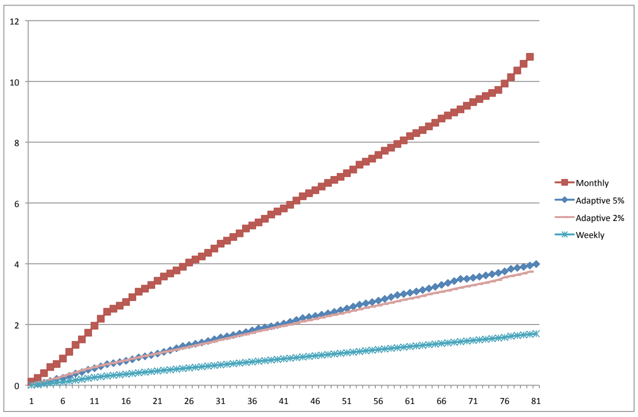
Table 3 summarises the number of system updates that each strategy entails. As expected, for both $\mu = 4$ and $\mu = 6$, the total number of updates with respect to a weekly strategy is dramatically reduced (from 80 down to 29 and 13 respectively); also, the higher the tolerance to error of the system administrator, the smaller the number of updates. It is indeed interesting to observe that, for a tolerance of $\mu = 6\%$, the number of updates performed by our adaptive strategy is less than a regular monthly strategy. Having ascertained a dramatic reduction in computational cost using our dynamic update strategy, we now turn our attention to assess the accuracy loss it entails.

For each update strategy, we have computed the *cumulative* precision/recall loss entailed over the 81-week deployment period. More precisely, during *each* week, 1500 tests were conducted (i.e., recommendation lists were generated); the prediction error was computed, and added to the error experienced thus far (intuitively, the older the system update we operate on, the more stale the data used to compute recommendations, the higher the error). Figures 6(a), 6(b), and 6(c) report results, under different compositions of the test sets (as before, 25-75, 50-50, 75-25); we assumed here the composition of actual tests remained the same (in terms of split between new and old data) throughout the 81-week period. Figure 6(a) thus represents a sort of best case scenario, while Figure 6(c) is a worst case one; Figure 6(b) represents the case where the actual tests follow a distribution that is aligned with what the controller used to learn the $\chi - \mu$ relationship.

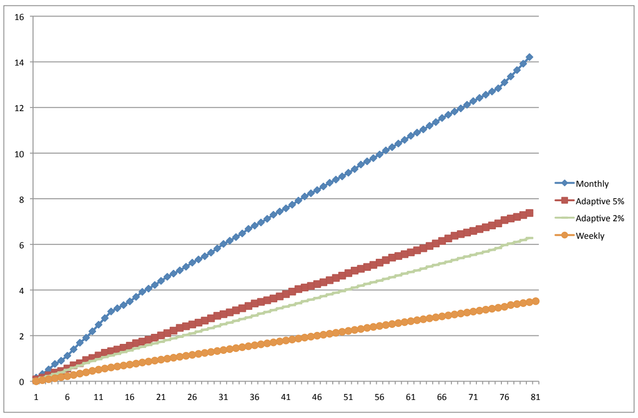
The following observations can be made: first, the error produced by the two adaptive strategies is almost the same; as the number of updates is less than half when $\mu = 6$ (from 29 down to 13), we concentrate on *Adaptive 6%* from now on. Second, across all test sets, the error entailed by *Adaptive* is much smaller than that obtained by *Monthly*; indeed, when 25% of tests refer to newly entered data (Figure 6(a)), the *Adaptive* strategy performs almost as well as the *Weekly* one in terms of precision/recall loss, while dramatically reducing the number of updates to 1/6. Finally, and perhaps



(a) Cumulative Error for 25-75 Test Set



(b) Cumulative Error for 50-50 Test Set



(c) Cumulative Error for 75-25 Test Set

Figure 6: Cumulative Error on Bibsonomy

most interestingly, *Adaptive 6%* has a much smaller accuracy loss than *Monthly* across all test sets, whilst also performing less updates overall (that is, both better accuracy *and* lower cost); this means that our *Adaptive* strategy is capable of choosing *when* an update is most required, that is, when it is expected to lessen the prediction error the most, with the biggest gain for the recommender website.

6. CONCLUSIONS AND FUTURE WORK

There is a drift between the way recommender systems are being evaluated by the research community, and the way they operate in practice. While researchers look at the recommender system as a one-shot process, in practice such systems operate on data that grows dynamically over time. The data structures that recommender systems operate on must thus be re-trained and kept up-to-date, so to maintain high prediction accuracy. Current state-of-the-practice sees system administrators manually set fixed update intervals; however, as we demonstrated in this paper, businesses do not grow uniformly over time, and thus such a rigid strategy benefits neither accuracy nor cost. We have then proposed to model a recommender system as a self-adaptive system, and turned to the literature on control theory in software engineering to offer a sound solution to the problem of self-monitoring and self-adaptation in this domain. Although feed-back (closed-loop) control systems have wide applicability in many application areas, we have observed this is not the case in recommender systems, as the output of the system may never be observed, or be observed at an unpredictable time in the future. We have then proposed to use feed-forward (open-loop) control theory, whereby the system simply monitors its own growth speed since its last updating process, uses this information to estimate the accuracy loss that not performing an update would entail, and ultimately decides whether the benefit of performing such update outweighs its cost. This results in a self-monitoring and self-adapting system that is capable of achieving a good trade-off between accuracy and cost, as our experimental evaluation conducted on the Bibsonomy website demonstrates.

Future work spans different directions: first, as mentioned earlier, we intend to study the predictability of the incoming recommendation queries, so to narrow the gap between the expected upper/lower bounds of prediction error, when empirically deriving the relationship between maximum tolerable error μ and observed data growth χ. Rather than manually conducting this laborious preparatory step, or even deciding a priori whether to monitor data growth alone or incoming queries too, a middleware-based approach to self-adaptive systems could be investigated, that conducts empirical evaluations over time, so to autonomously adjust the predictive function f to error μ. Finally, a single pre-defined goal (i.e., high accuracy) has been used to drive the controller behaviour; however, there is an ongoing discussion in the recommender systems community in relation to what metrics should define the quality of a recommender system, with accuracy being complemented by dimensions like diversity, serendipity, and surprise. Understanding the relation between these requirements and the aspects of the environment that a recommender systems controller should collect and analyse is an open research question.

Acknowledgement. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under the Grant Agreement n. 234239. The authors are solely responsible for it and it does not represent the opinion of the Community. The Community is not responsible for any use that might be made of information contained therein.

7. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Context-Aware Recommender Systems. In *Proc. of the ACM Conference on Recommender Systems*, pages 335–336, 2008.

[2] C. Anderson. *The Long Tail: Why the Future of Business Is Selling Less of More.* Hyperion, 2006.

[3] K. Astrom and B. Wittenmark. *Adaptive Control.* 2nd Edition, Addison-Wesley, Reading (1995)

[4] R. M. Bell and Y. Koren. Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights. In *Proc. of the 7th IEEE International Conference on Data Mining*, pages 43–52, 2007.

[5] Y. Brun, G. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw. Engineering Self-Adaptive Systems through Feedback Loops. In *Software Engineering for Self-Adaptive Systems*, pages 48–70, 2009. Springer-Verlag.

[6] Z. Bubnicki. *Modern Control Theory.* Springer, 2005.

[7] B.H. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In *Software Engineering for Self-Adaptive Systems*, pages 1–26, 2009. Springer-Verlag.

[8] G. Dumont and M. Huzmezan. Concepts, Methods and Techniques in Adaptive Control. *IEEE American Control Conference*, Anchorage, AK, USA, vol. 2, pp. 1137-1150. 2002.

[9] J. Gemmell, T. Schimoler, M. Ramezani, and B. Mobasher. Adapting k-Nearest Neighbour for Tag Recommendation in Folksonomies. *Proc. of the 7th Workshop on Intelligent Techniques for Web Personalization & Recommender Systems*, 2009.

[10] J. Herlocker, J. Konstan, A. Borchers, and J. Riedl. An Algorithmic Framework for Performing Collaborative Filtering. In *Proc. of the 22nd Annual International Conference on Research and Development in Information Retrieval*, pages 230–237, New York, NY, USA, 1999. ACM.

[11] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME - Journal of Basic Engineering*, 82:35–45, 1960.

[12] M. Kurucz, A. A. Benczur, and K. Csalogany. Methods for Large Scale SVD With Missing Values. In *Proc. KDD Cup and Workshop*, August 2007.

[13] N. Lathia. *Evaluating Collaborative Filtering Over Time.* PhD thesis, University College London, 2010.

[14] N. Lathia, S. Hailes, and L. Capra. The Effect of Correlation Coefficients on Communities of Recommenders. In *Proc. of 23rd Annual ACM Symposium on Applied Computing*, 2008.

[15] M. Litoiu, M. Woodside and T. Zheng. Hierarchical Model-based Autonomic Control of Software Systems. In *ACM ICSE Workshop on Design and Evolution of Autonomic Software*, St. Louis, MO, USA, pp. 1âĂŞ7. 2005.

[16] M. Litoiu, M. Mihaescu, D. Ionescu and B. Solomon. Scalable Adaptive Web Services. In *ACM ICSE Workshop on Development for Service Oriented Architectures*, Leipzig, Germany. 2008.

[17] H. Ma, I. King, and M. R. Lyu. Effective missing data prediction for collaborative filtering. In *Proc. of the 30th ACM SIGIR*, pages 39–46, 2007.

[18] M. Mull. Characteristics of High-Volume Recommender Systems. In *Proc. of Recommenders Workshop*, Bilbao, Spain, September 2006.

[19] H. Müller, M. Pezzè, and M. Shaw. Visibility of Control in Adaptive Systems. In *Proc. of the 2nd International Workshop on Ultra-Large-Scale Software-Intensive Systems*, pages 23–26, Leipzig, Germany, 2008. ACM.

[20] G. Potter. Putting the Collaborator Back into Collaborative Filtering. In *Proc. of the 2nd Netflix-KDD Workshop*, Aug. 2008.

[21] T. Söderström and P. Stoica. *System Identification.* Prentice-Hall, Englewood Cliffs. 1988.

[22] V. Zanardi and L. Capra. Social Ranking: Uncovering Relevant Content using Tag-based Recommender Systems. In *Proc. of the Conference on Recommender Systems*, pages 51–58, 2008. ACM.