Using Control Theory for Stable and Efficient Recommender Systems

Tamas Jambor[§] Jun Wang[§] Neal Lathia[‡]
[§]Dept. of Computer Science,University College London,UK
[‡]Computer Laboratory,University of Cambridge,UK
{t.jambor, j.wang}@cs.ucl.ac.uk, neal.lathia@cl.cam.ac.uk

ABSTRACT

The aim of a web-based recommender system is to provide highly accurate and up-to-date recommendations to its users; in practice, it will hope to retain its users over time. However, this raises unique challenges. To achieve complex goals such as keeping the recommender model up-to-date over time, we need to consider a number of external requirements. Generally, these requirements arise from the physical nature of the system, for instance the available computational resources. Ideally, we would like to design a system that *does not deviate* from the required outcome. Modeling such a system over time requires to describe the internal dynamics as a combination of the underlying recommender model and the its users' behavior. We propose to solve this problem by applying the principles of modern control theory—a powerful set of tools to deal with dynamical systems—to construct and maintain a stable and robust recommender system for dynamically evolving environments. In particular, we introduce a design principle by focusing on the dynamic relationship between the recommender system's performance and the number of new training samples the system requires. This enables us to automate the control other external factors such as the system's update frequency. We show that, by using a Proportional-Integral-Derivative controller, a recommender system is able to automatically and accurately estimate the required input to keep the output close to a pre-defined requirements. Our experiments on a standard rating dataset show that, by using a feedback loop between system performance and training, the trade-off between the effectiveness and efficiency of the system can be well maintained. We close by discussing the widespread applicability of our approach to a variety of scenarios that recommender systems face.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information Filtering

Keywords

Recommender Systems, Control Theory, Temporal Analysis

1. INTRODUCTION

Collaborative Filtering (CF) algorithms have become the mainstream approach to building web-based recommender systems [3]. The popularity of CF stems from its ability to explore preference correlations between users; it uses a wide range of statistical approaches that allows recommender systems to identify items (movies, music, books, etc.) of interest to web users based on their historical preferences. In the research domain, this problem is often formalized as a prediction problem: predicting unknown user ratings based on a set of observed preferences [8, 16].

In the research literature, CF algorithms are evaluated in a relatively *static* context: datasets are examined using traditional machine learning methodologies to produce cross-validated results, by randomly splitting the data into training and test sets. In practice, these systems will experience a continuous influx of new ratings: the deployment scenario is dynamic and continuously subject to change [18]. Recent work has delved into this domain and examined the differences that emerge between the two contexts [9, 20]. In particular, CF algorithms produce up-to-date recommendations by being regularly re-trained (e.g., daily) in order to incorporate new ratings into the model of user preferences. Given that state-of-the-art CF algorithms are, in general, expensive to update, because of the limited resources available as well as the computation required to train the model. Thus, system developers have to make a critical decision: they must decide *when* and *how frequently* to update their systems. In doing so, they must balance between the benefit of a recently re-trained algorithm (which will produce recommendations based on the latest user ratings) and the cost that the business incurs from re-training. This problem is often solved by iteratively re-training at a pre-defined interval [23]. The point is further illustrated in Fig. 1. The figure depicts the performance dynamics of the Movielens 1M dataset[1] [10] (described in Section 4) for a period of 20 months. It shows that training only every fourth month results in a substantial performance loss compared to training every month. In addition, as new users enter the system, these users cannot receive reliable recommendations until the system is re-trained on the new data points. This further deteriorates the performance [4] and subsequently the system may not be able to provide accurate recommendations for those new users. This trend can be observed between *any* time interval as long as new data enters the system, because the additional information would more likely to help improving the performance.

In this paper, we propose a control-theoretic approach to designing recommender systems. We propose a new methodology on how to design and manage a complex recommender system in order to maintain the trade-off between the effectiveness and efficiency of the system by applying modern

WWW 2012, April 16–20, 2012, Lyon, France.
ACM 978-1-4503-1229-5/12/04.

[1]We tested this hypothesis with other larger dataset and found the same pattern.

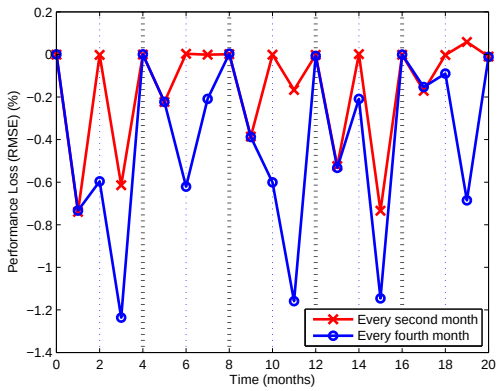


Figure 1: The effect on performance by reducing the update time of the system (Movielens 1m). The performance loss is calculated against a baseline of training every month.

control theory [25]. We demonstrate that modern control theory has the potential to provide the required mathematical tools to both analyze and model the stability and robustness (resistance to error and disturbance) of a recommender system over time. We validate our approach by examining the dynamic relationship between the recommender system's performance and the number of new training samples the system requires. We show that control theory would allow us to design the recommender system and its feedback loop to effectively mitigate the effects of any forces (such as noise in the data) that may arise during operation, that would otherwise negatively affect the recommender system's performance over time. We further discuss how, after defining the system dynamics this way, the control loop can be replaced with other signals, such as controlling the computational effort over time and the system's update frequency. We believe that, by grounding the operation of recommender systems in control theory, this work not only contributes to the growing body of research addressing temporal dynamics in recommender systems, but will also be of significant interests to the wider research areas such as dynamic information retrieval and filtering.

The remaining of the paper is organized as follows: we begin with the related work (Section 2). We then formalize the temporal recommendation as an instance of a control theory problem (Section 3). In Section 4, we assess the use of the principles of control theory to build systems that adaptively respond to the continuous dynamics of online deployment. This includes the parameter estimation of the dynamical system and the analysis of various types of controllers. In Section 5, we evaluate the controller design on a number of the practical scenarios. We then discuss the merits and weaknesses of this approach along with a number of practical applications (Section 6). Finally, we conclude our work by discussing further applications of control theory and future directions of research (Section 7).

2. RELATED WORK

The temporal dynamics of any recommender system is the combination of two intersecting components, which have each recently appeared in the literature. On the one hand, the preferences of the system's users may be subject to change, as they are affected by seasonal trends or discover new content. Evolution of preferences can be modeled as a decay, so that, in the longer term, parts of users' profiles can expire [28]. Koren [19] also examined this problem, distinguishing between the transient and long term patterns of user rating behaviors, so that only the relevant components of rating data can be taken into account when predicting preferences. In this way, recommender systems can retain separate models of the *core* and *temporary* taste of any user. Then, the core taste could be used in a longer period of time while the temporary taste can expire if the user becomes interested in items that do not match her previous temporary taste. In particular, we note that the above studies focus on user preference shift as a means toward improving prediction accuracy.

The flip side of the recommendation temporal dynamics relates to any temporal changes subsumed by the recommender *system* itself as time progresses and the system is updated. Current recommender systems address the fact that users continue to rate items over time by iteratively re-training their preference models. Modeling and evaluating the performance of recommender systems from the perspective of accuracy, diversity, and robustness over time was addressed in [20, 21]. The central tenet here is that the regular, iterative update of recommender systems can be both simulated and levied to improve various facets of the recommendations that users receive, which include temporal diversity.

A middle ground between purely static and dynamic approaches to CF is the use of online learning algorithms. For example, the authors in [27] proposed online regularized factorization models that did not require time-consuming batch-training. An active learning approach was proposed to identify the most informative set of training examples through minimum interactions with the target user [15]. However, the purpose here was to quickly address the problem of *cold-start* items and users; our model generalizes beyond this scenario.

In this work we adopt a different stance, by focusing on adapting the updates of the system itself. By drawing from the principles of control theory [26], we show that systems do not need to be iteratively updated, but rather can be updated as a response to fluctuating user activity. Control theory stems from research relating to guiding the behavior of dynamical systems; notable examples include dealing with network traffic [12]. At the broadest level, control systems require three components: (a) a *system* which produces an output in a dynamic context (e.g., a recommender system), (b) a *measure* of the quality of the output (such as the *RMSE* (Root Mean Squared Error) on the recommendations) and (c) a *mechanism* to adapt or tune the system's parameters according to the measured quality of the output. The control system binds these three components together into a closed loop; in doing so, it provides a means for feedback relating to the system's performance to be input into the system itself.

Researchers have found that control theory can accurately model the behavior of software systems and it provides a set of analytical tools that can predict, with high accuracy, the behavior of a real system. In [29], control theory was used to monitor system growth and control whether the system needs to perform an update. Based on the amount of new data entering the system, it calculated the loss in accuracy if the system was not updated, and decided whether the benefit of preforming an update would outweight the computational cost associated with the update, based on a predefined tolerable performance loss. This approach assumes that the rate of data growth is a linear function of the performance loss over time. The disadvantages of this is that the sys-

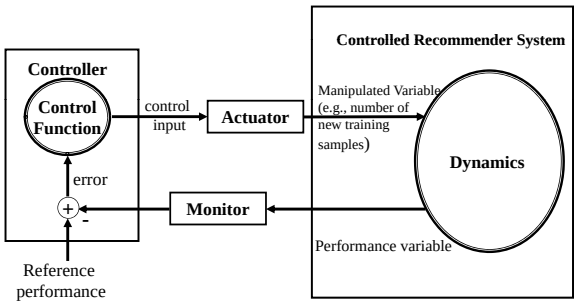


Figure 2: The closed-loop control of a dynamical recommender system.

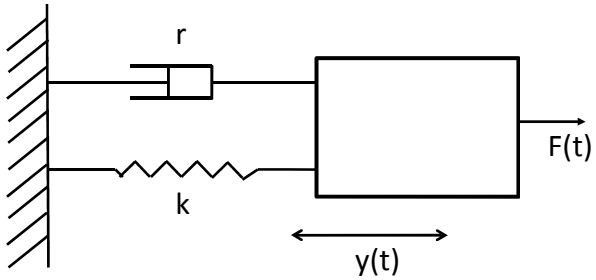


Figure 3: A mass attached to a spring and damper. $y(t)$ is the displacement. The damping coefficient is represented by r in this case, where the spring constant is k. $F(t)$ in the diagram denotes an external force. In our case, $y(t)$ is the performance value while $F(t) = u(t)$ number of new training samples.

tem can only rely on a simplified relationship between the data flowing into the system and the performance (which might not be linear and also include additional factors) and cannot directly control the disturbance introduced by the dynamics of the recommender system as the actual output of the system is not known to the controller. In [26], researchers presented a workflow, based on classical control theory, to design a feedback control system for an email server to maintain a reference queue length. They used a statistical model to estimate system dynamics, this was followed by a controller design phase to analyze the system response. They found that the analysis predicted to behavior of the real system with high accuracy. Control theory has also been applied to reputation systems, where the aim was to identify and reduce the impact of malicious peers on recommendation. In [22], a feedback controller was used to adjust users' recommendation trust based on the accuracy of their rating.

3. A FEEDBACK CONTROL APPROACH

In this section, we present an adaptive temporal recommendation architecture derived from control theory. The key feature of this architecture is a feedback control loop (illustrated in Figure 2) where the *control function* generates an input of the dynamical system that is adequate to maintain performance, this input signal is realized and entered into the system by the *actuator*. Then, the output of the dynamical system is *monitored* and fed back to to controller in order to compare it to the reference value, compute the error and generate the next control input.

The idea of the feedback loop can be adopted to recommender systems by the analogy of a simple physical system: a weight attached to a wall with a spring and damper, as illustrated in Figure 3. This simple system has one input, $F(t)$, which represents how much force has been applied to pull the weight from the wall. The output, $y(t)$, is how much the weight will be displaced from its the current position. If we try to keep the weight from the wall at a certain (stable) distance, we need to balance a number of factors. While there is an inherent relation between the input and output, the amount of movement over time will also be affected by the strengths of the spring and damper, which both exert different forces on the system. In effect, we have a system where the relation between an input and output value is dependent not only on the input itself, but a number of other implicit factors that are built into the system. A recommender system follows a similar pattern: the output (performance) relates to the input (training data) along with a variety of hidden factors in the system. In order to control the output, we therefore need to approximate this function: we do so by observing the system over time. The system keeps monitoring the performance and feeds the estimated performance measure back to a controller. The controller then reacts accordingly by modifying the input in order to maintain a particular stable outcome. Note that the motivation of this approach is to maintain instead of optimize the system. It is suggested in [17] that there are multiple objectives that a recommender system needs to fulfill, some of them might directly contradict with the performance of the system. This approach would enable us to directly define and control any aspect of a live system, given that the input-output relationship of the dynamical system can be defined.

In the following sections we formalize how this metaphor of a controller can be actualized and implemented in a recommender system. We take two distinct steps in designing this system. First, we consider the recommendation accuracy as a stochastic yet controllable variable, and develop a mathematical description of the underlying dynamic process that can be controlled. Such a description allows practitioners to analyze their system with a wide range of analytical techniques that are available from control theory. Then, we use these techniques to define and estimate the best controller strategy to generate the control of the system. We describe the two components in the following subsections.

3.1 Modeling Performance Dynamics

We denote the performance of a recommender system as $y(t)$, where t represents time. Without loss of generality, we assume lower values of $y(t)$ indicate better performance. We assume that performance can be observed directly by monitoring user feedback and comparing it to the prediction provided by the model (see Section 4). The number of *new* training samples is denoted as $u(t)$, and we consider this to be an input signal to the dynamical system.

A top down approach to model a system's temporal dynamics is to consider the physical laws that govern the system. For instance, Newton's second law of motion says that an object with mass m subject to a force $F(t)$ undergoes an acceleration $v'(t)$ that has the same direction as the force, and its magnitude is directly proportional to the force and inversely proportional to the mass, i.e., $F(t) = mv'(t)$. By taking the analogy, we consider that the change rate of the performance is proportional to the number of new training samples entering the system ($y'(t) \approx u(t)$). To approximate this, we assume that the number of *new* training samples is also proportional to the current performance of the system ($y(t) \approx u(t)$). This can also be interpreted as the need to

have a certain number of new training samples to maintain the performance of the system. As a result, the dynamics of the performance can be approximated by the following simple differential equation:

$$ry'(t) + ky(t) \approx u(t) \tag{1}$$

where $y'(t)$ (change rate of the system performance) is the derivative of $y(t)$. It is interesting to see that the above equation is similar to a mechanical system illustrated in Figure 3. $ky(t) \approx u(t)$ is the analogy to Hooke's law of elasticity that the extension of a spring is in direct proportion with the load applied to it, while $ry'(t) \approx u(t)$ follows a damping force. We thus have two parameters r and k, which need to be estimated from the performance dynamics of the recommender system. To estimate them, we rewrite the dynamical model in a form of discrete time:

$$y(t+1) = \frac{r}{r + \Delta t k} y(t) + \frac{\Delta t}{r + \Delta t k} u(t) + \varepsilon(t) \tag{2}$$

where Δt denotes the unit of time interval. In addition, we tackle the uncertainty in the system by adding random disturbance $\varepsilon(t)$ at time t. Eq. (2) is in fact a linear regression system. For simplicity, this model can also be written in matrix notation when drawing the observation over a time period $t \in [0, T]$:

$$\mathbf{y} = \mathbf{Y}^T \theta + \varepsilon \tag{3}$$

where $\mathbf{y} = (y(1), ..., y(t), ..., y(T))^T$, $\theta = \left(\frac{r}{r+k\Delta t}, \frac{\Delta t}{r+k\Delta t}\right)^T$, and

$$\mathbf{Y} = \begin{pmatrix} y(0) & ... & y(t) & ... & y(T-1) \\ u(1) & ... & u(t) & ... & u(T-1) \end{pmatrix}$$

Linear systems have been well studied in many research fields. One of the standard solutions to obtain the parameters is to employ the Maximum Likelihood (ML) estimation from the observation over a time period $t \in [0, T]$. We assume that the error ε has a multivariate normal distribution with mean 0 and variance matrix $\delta^2\mathbf{I}$, where $\mathbf{I}$ is an identity matrix of size T. Then the log-likelihood function of the parameters is

$$\mathbf{L}(\theta, \delta^2 | \mathbf{Y}) = \ln\left(\frac{1}{(2\pi)^{T/2}|\delta^2\mathbf{I}|^{1/2}} e^{-\frac{(\mathbf{y}-\mathbf{Y}^T\theta)^T(\mathbf{y}-\mathbf{Y}^T\theta))}{2\delta^2\mathbf{I}}}\right) \tag{4}$$

Differentiating this expression with respect to θ we find the ML estimates of the parameter θ:

$$\left(\frac{\hat{r}}{\hat{r}+\hat{k}}, \frac{1}{\hat{r}+\hat{k}}\right) = \underset{\theta}{\operatorname{argmax}}\, \mathbf{L}(\theta, \delta^2 | \mathbf{Y}) = (\mathbf{Y}\mathbf{Y}^T)^{-1}\mathbf{Y}\mathbf{y} \tag{5}$$

where we set $\Delta t \equiv 1$ to sample the data per day. A problem still remains: we need to choose input $u(t)$ to increase the accuracy and robustness of the parameter estimation. We introduce a flexible way to add disturbance by using the log-normal random walk model [7, 11]. We estimate the mean (μ) and standard deviation (σ) from the input data and simulate an additional time series that has the same mean and variance. Let us define $u(t)$ as the time series of the input values over a predefined time period $[0, T]$. The modified input $u(t)$ becomes as follows:

$$du(t) = \mu u(t)dt + \sigma u(t)dW(t) \tag{6}$$

where $W(t)$ is the Wiener process

$$\mathrm{dW}(t) = \varepsilon\sqrt{dt} \text{ and } \xi \text{ is the noise } N(0,1) \tag{7}$$

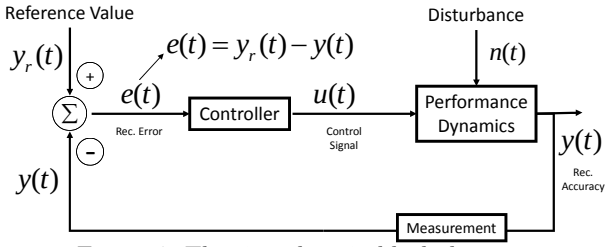


Figure 4: The time domain block diagram.

Essentially, this approach randomizes the time series depending on the standard deviation of the intended input. The advantage of this approach is that setting σ to zero we get the exponential curve that represents the natural way of data growth [20].

In summary, we use the Maximum Likelihood estimation (Eq. (5)) and and the lognormal random walk (generating the input) to obtain the parameters r and k for a given time series that represents the dynamics between the input and the output of the system. The detailed experiments to estimate parameter r and k can be found in Section 4.2.

3.2 Feedback Controller

The next question is how to design the input signal $u(t)$ so that the performance $y(t)$ will stay as stable as possible. That is if $y(t)$ deteriorates from the desired value, how do we change input $u(t)$ to quickly alter the systems dynamics and bring $y(t)$ back to the desired value? This is done by constructing a closed loop system. In a closed loop system, the output is fed back and compared with a reference value $y_r(t)$. The error signal, denoted as $e(t) = y_r(t) - y(t)$, will be sent to a controller (the process is illustrated in Figure 4). Upon receiving the error feedback signal, the controller then calculates how much modification is needed for $u(t)$ at this moment. In other words, the feedback on how the system is actually performing allows the controller to dynamically compensate for disturbances to the system and produce a response in the system that perfectly matches the user's wishes. Our discussion here is limited to the situation where the state of the system, in this case the recommendation performance, can be observed. In practice, this is possible as we can measure the recommendation performance periodically by looking at users' feedback or construct a validation set over time. The approach is further explained in Section 4. the PID (Proportional-Integral-Derivative) controller [5]. It not only captures the linear relationship between the error signal and input $u(t)$, but also covers both the derivative and the integral of the error signal. Formally, we have:

$$u(t) \equiv Ce(t) + B\sum_{0}^{t} e(t)\Delta t + D\frac{e(t) - e(t-1)}{\Delta t} \tag{8}$$

where the nonlinear relationship is represented by three parameters. The signal ($u(t)$) that the controller produces is the combination of the *proportional* gain (denoted as C) times the magnitude of the error, the *integral* gain (denoted as B) times the integral of the error and the *derivative* gain (denoted as D) times the derivative of the error. The conversion of the integration from continuous to discrete time is done by the *backward Euler* method [24]. The PID controller is the combination of the three basic types of controllers, each adding an extra layer to the system. The proportional controller makes a change to the output that is proportional to the current error value. The integral term accelerates the movement of the process towards the set-

point and eliminates the residual error that occurs with a pure proportional controller. The derivative term slows the rate of change of the controller output. In practice, not all elements are needed. For example, in some cases, one can find that D is not required and set to zero. In a nutshell, the above equation feedbacks the error and uses it as the outer force to change the dynamics of the performance. As explained in the previous section we rewrite this dynamic equation in a form of discrete time with uniform sampling ($\Delta t \equiv 1$), combining it with Eq. (1) gives:

$$\begin{aligned} &\hat{r}\Big(y(t) - y(t-1)\Big) + \hat{k}(y(t)) \\ \approx &Ce(t) + B\sum_{0}^{t} e(t) + D\Big(e(t) - e(t-1)\Big) \end{aligned} \tag{9}$$

where parameter $\hat{r}$ and $\hat{k}$ are obtained using the Maximum Likelihood estimation introduced in Eq. (5). Tuning the three parameters (C, B and D) offline is needed in order to guarantee the desired performance. In control theory, the system is usually modeled by transforming the discrete time signal into the frequency domain using the z-transform. A popular techniques to obtain the controller's parameters in z domain is called the Ziegler-Nichols method. In this paper, we adopt this method along with a number of software tools and manual tuning. We refer to [25] for the detailed z domain analysis and the underlying mechanism to obtain the three parameters while we primary focus on the design pattern of the system.

4. DESIGNING A CONTROLLED RECOMMENDER SYSTEM

In this section, a series of experiments were conducted in order to evaluate the performance of the proposed control-theoretical approach of recommender systems. Without loss of generality, the empirical study focused on the relationship between the input (the number of training samples) and the output (recommendation accuracy). We chose this relationship, because the input and the output generalizes the recommender system's dynamics best, therefore, with minimal modification it is applicable to a wide range of scenarios (see Section 5). Other inputs and outputs can be obtained by following the same design principle and steps presented here (for further suggestions see Section 6). The experiments were conducted by emulating the real use of recommendation systems with the widely used MovieLens data set. The benefits of such an evaluation setup compared to using an operational recommendation engine are two-fold: first, it allows us to flexibly test various scenarios, and make the individual experiments targeted and focused; second, using public datasets allows others to easily replicate and compare the results, and validate our conclusions. The MovieLens data consists of 1 million ratings for 3900 movies by 6040 users. Three widely used recommendation algorithms; the user-based, the item-based and the SVD method, were used as the basis of rating prediction. As no significant difference was found among them in terms of controllability, we only report the results obtained using the popular SVD algorithm [13].

The design principle was evaluated in several stages by building a dynamical recommender system step by step and sampling the rating data over time. Specifically, to prevent over-fitting, the rating data was randomly divided into three sets; training, test and hold-out set. We fed the data into the system incrementally, so that each day during the predefined period of 30 days we randomly added a pre-specified number of training samples to the training set based on the requirements set by our feedback model and re-trained the recommender system using all the data in the training set. We then estimated the recommendation accuracy using the available user feedback data from the next day. This process was repeated each day, adding all the tested data points to the hold-out set, which was then used to pick the required number of data samples for the next cycle of training.

Table 1: The R^2 performance of the three input data set which we estimated parameter r and k (Eq. (5)). The set marked in bold was used to estimate the parameters which was then tested on the remaining two sets. We used the row colored gray for the further experiments.

Increase	Stable	Decrease	Mean	Standard deviation
0.9863	0.6699	0.3974	0.6845	0.2947
0.9723	**0.9439**	0.9176	0.9446	0.0274
0.9702	0.9421	**0.9314**	0.9479	0.0200

(a) Input without noise

Increase	Stable	Decrease	Mean	Standard deviation
0.9869	0.8852	0.7766	0.8829	0.1251
0.9646	**0.9078**	0.8726	0.915	0.0464
0.8945	0.9008	**0.8840**	0.8931	0.0084

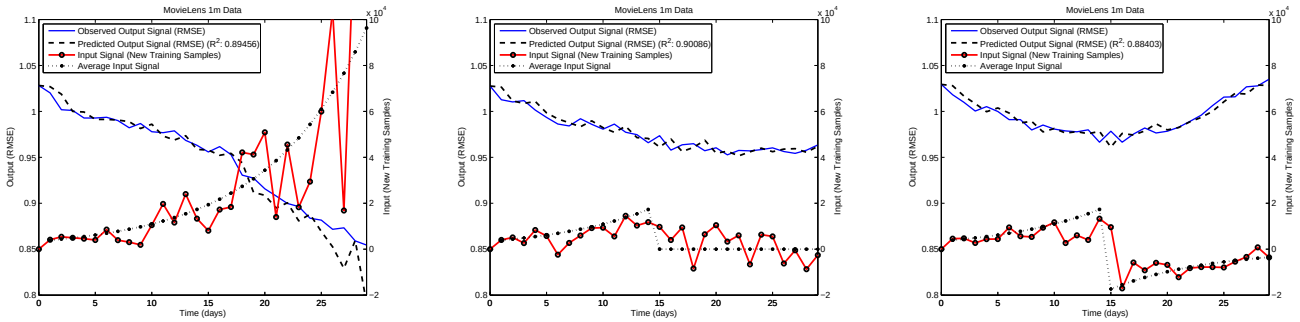
(b) Input with noise (see Eq. (6))

We started with the parameter estimation of performance dynamics based on historical data. It was then followed by the system analysis that aimed to identify the controllers characteristics and choose the right parameters before deploying the system. With the right parameters, we then evaluated the chosen controllers with real data and simulated scenarios. A potential drawback of such a controlled evaluation configuration might be its lack of testing the robustness of the system against any unseen environment noise. We thus provided additional experiments to test how well the system handles sudden changes over time. Finally, we tested how our approach can be applied to balance the trade off between computational cost and performance requirements.

4.1 Parameter Estimations

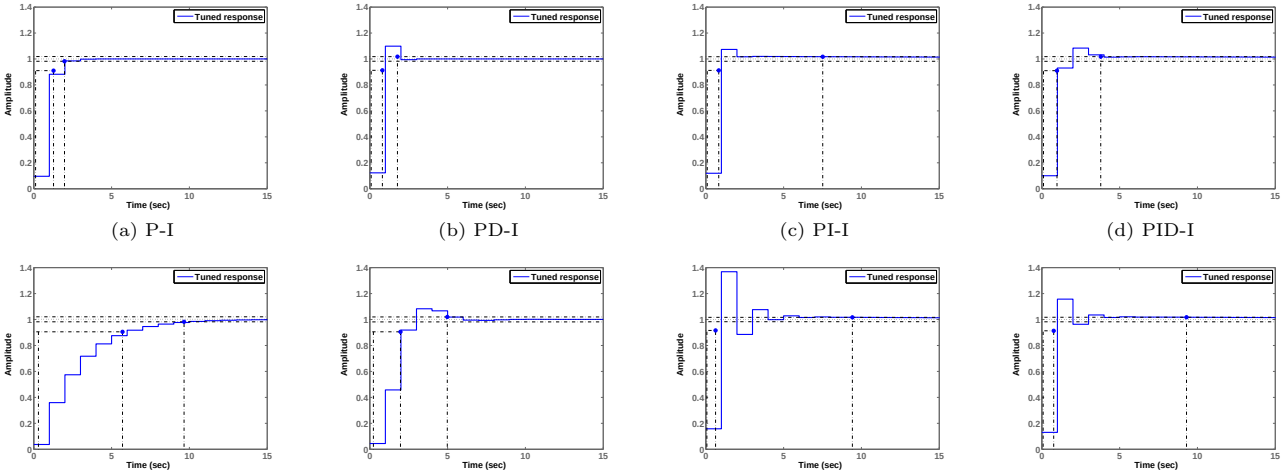
4.1.1 System Dynamics

In order to design a suitable controller, first we need to understand the dynamics of the system by obtaining the parameters that describes the recommender systems dynamics (introduced in Eq. 1). The dynamics of the system is needed for the controller design the simulate the feedback loop system (that includes the recommender system and the controller). Without an accurate description of the system dynamics, it is not possible to predict the system behavior and obtain a suitable controller. To estimate the parameters using the approach proposed in Section 3.1, we devised three scenarios that the recommender system might encounter. The first one covers the situation when the data is growing exponentially which is the case where many new users start using the system. The second is concerned with the situation where growth flattens after some initial growth, whereas the third one simulates the decrease of the training data. This last scenario was to cover the possibility that the controller might reduce the performance of the system.



(a) Increase (b) Stable (c) Decrease

Figure 5: Parameter estimation (MovieLens 1m). The parameters introduced in Eq. 5 were learned on set (c) and tested on (a) and (b).



(a) P-I (b) PD-I (c) PI-I (d) PID-I

(e) P-II (f) PD-II (g) PI-II (h) PID-II

Figure 6: Step response (MovieLens 1m). The first point represents the rise time of the system (90% of the target value) and the second point marks when the system reaches steady state. The first row shows the fast controllers with 2-5 sec response time, and the second row has the slower controllers with 5-10 sec response time.

Figure 5 illustrates the relationship between the number of new samples and output the recommendation accuracy measured by RMSE. The parameters were estimated using Eq. (5) on one of the scenarios and tested on the remaining ones to predict the accuracy of the system. The random walk technique (described in Section 3.1) was used to generate the input sequences (how the influx of new ratings evolves over time). They were estimated from the data to represent the natural way of data growth. Adding noise to the signal helped us model various input combinations. The input was put through our recommender model to generate the performance over a period of 30 days.

A key question for parameter estimation is how to construct an input sequence such that the prediction of output from a unseen input sequence would be as accurate as possible. We measured how well the prediction fits to the observed data using the coefficient of determination [14]

$$R^2 = 1 - \Big(\sum_{t=0}^{T} (y_h(t) - y(t))^2 / \sum_{t=0}^{T} (y(t) - \bar{y})^2\Big) \qquad (10)$$

where $R^2 \in [0,1]$, y_h is the predicted output, y is the observed output and $\bar{y}$ represents the mean of the observed output over a time period $t \in [0,T]$. This measure is widely used in statistical models to measure the prediction of future outcomes; the higher the value is, the better it fits to the data. Table 1 shows that the most robust parameters across all the three scenarios were obtained by estimating them on the data shown in Figure 5 (c) (the 6th row in Table 1) as it generates the smallest standard deviation across the three scenarios. This also suggests that adding noise using the random walk model helps improving the robustness of the estimation. As illustrated in Figure 5(a) and (b), the model produced consistent predictions despite the fact that the data was noisy, therefore the parameters were robust to be used for the next steps in the controller design.

4.1.2 *Step Response and the Feedback Controls*

Having obtained the system dynamics of recommender model, we then studied what kind of controller best suits our predefined control objectives and obtain the parameters defined in Eq. (8). We evaluated four kind of controllers including a Proportional (P), a Proportional-Derivative (PD), a Proportional-Integral (PI) and a Proportional-Integral-Derivative (PID) controller. To quickly test our ideas, we used a standard PID design tool provided in the Control System Toolbox in Matlab [6]. In a nutshell, this tool simulates the behavior of the dynamical system given the internal

Table 2: The general characteristics of the estimated recommendation controllers. The time measured by seconds corresponds to days in our experiments (e.g. one training cycle).

Type	Rise Time (sec)	Settling Time (sec)	Overshoot (%)
P-I	1	2	0
P-II	5	10	0
PD-I	0	2	9.84
PD-II	1	5	8.21
PI-I	0	8	3.24
PI-II	0	10	36.8
PID-I	0	4	8.41
PID-II	0	10	15.7

dynamics of the recommender system and the controller. It obtains the initial parameters using heuristic approaches, such as the Ziegler-Nichols tuning method [30] to satisfy certain initial requirements: they include closed-loop stability, adequate performance and robustness. Then, this initial design can be fine-tuned manually to achieve the specific requirements for the given system. For details of the method, we refer to [25], while staying focused on finding an ideal controller that has the characteristics that satisfies our requirements.

Four measures were used to obtain an insight into the stability and the sensitivity of a feedback controller before employing it in the system. *Step response* is defined as the time behavior of the output of a general system when its input suddenly changes from zero to one in a very short time; *Rise time* refers to the time required for the output to reach 90% of the reference value. For example if the reference value is set to one and the current output of the system is zero (regardless of the metric used to quantify the output), rise time shows how fast the system can reach 0.9. Essentially, this value indicates how fast the system can respond to a change in the rate of influx of ratings. However, fast rise time might cause the output to exceed the desired value: this is called the *overshoot* of the signal. In some cases, for example when we aim to control the computation of the system, it is desired to keep overshoot as low as possible, as if the system requires more computation than it is available this might cause server overflow. We also measured how long it takes for the feedback system to remain within a specified error band (*settling time*). The error band is defined to be $\pm 1\%$ of the target value. In some cases, the system stabilizes at a different value that is required; thus *steady-state error* is defined as the difference between the output and the target value after the output has reached the steady state.

A recommender system requires a controller with a fairly fast response time, that is the controller should react to changes as fast as possible. Ideally, this would be less five training cycles given the fact that a training cycle in practice can be as long as a day. This should also be accompanied by a fast settling time and a small steady-state error. These requirements would ensure that the system converges to the reference value quickly which is important in order to gain control of the system in a reasonable time frame. Figure 6 depicts the step response of the four controllers, whereas the main characteristics are also summarized in Table 2. For each of the controllers, we have two configurations, one of them was set to be "fast" (in terms of rise time) (mark I), and the other one is designed to be "slow" (mark II) with a maximum settling time of ten training cycles. The slower controllers are generally more stable and less likely to overshoot, which might be desirable in some scenarios, for example if the computational resources are scarce.

Table 3: The speed, precision, stability and overshoot of the controllers for reference value 0.86. The best performing controllers in terms of error are colored gray.

Type	Settling Time	RMSE-SS (error)	SD-SS (stability)	Overshoot (%)
P-I	18.6	0.00637	0.00479	0.35632
P-II	23	0.01049	0.00346	0
PD-I	19.8	0.00495	0.00448	0.49484
PD-II	20.8	0.00599	0.00377	0.18454
PI-I	18.8	0.00597	0.00620	1.14088
PI-II	18.6	0.00556	0.00479	0.49416
PID-I	19.2	0.00449	0.00460	0.83654
PID-II	18.6	0.00452	0.00455	0.72810

Figure 6 shows that all the controllers reach the desired output and remain stable. However, controllers that include an integral part (such as PID and PI) were slower to settle, and overshot the target considerably. This may be due to the fact that the integral controllers are intended to reduce the residual error over time by accumulating the errors. In addition, this becomes an important issues as, compared to physical systems, recommender systems might not always have the required number of training samples available which affects accumulated error over time. We thus introduce a linear discounted output to add it to the integral controller as follows

$$y_i(t) = \begin{cases} y(t) + \frac{1}{r+k}(u_i(t) - u(t)) & \text{if } u_i(t) > u(t) \\ y(t) & \text{if } u_i(t) \leq u(t) \end{cases} \tag{11}$$

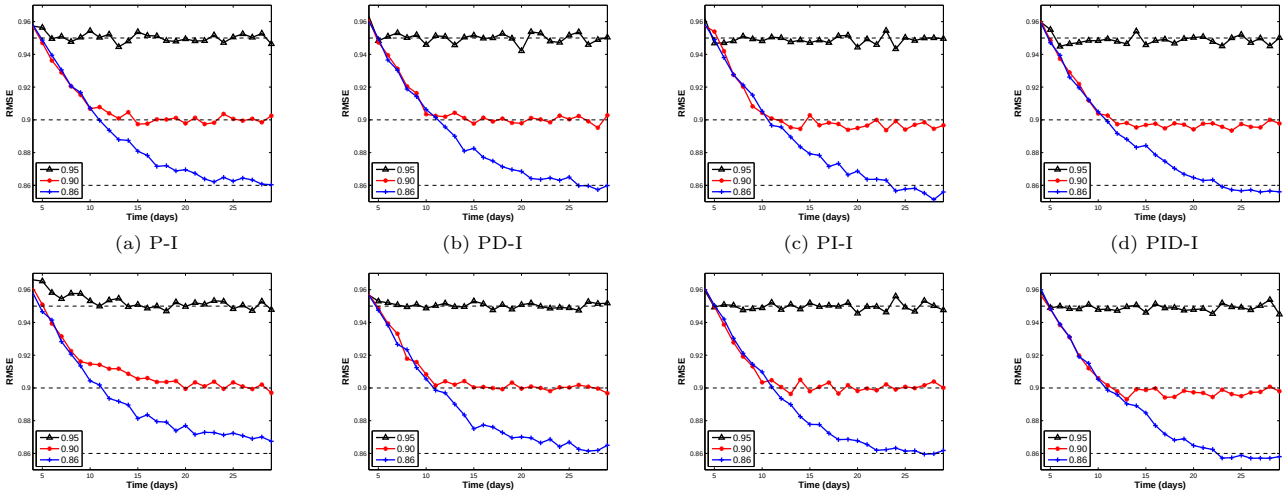
where $y_i(t)$ represents the modified output (performance) discounted by the difference between the required input ($u_i(t)$) (defined by the controller) and the actual input ($u(t)$). This compensates the discrepancy between the required and the available training samples. Essentially, the influence from the integral part is discounted when the required number of training samples is not available.

By contrast, we observe that the derivative term helped increasing the response time of the feedback system. The PD controller was faster than the P controller, while settling at the same time. In this regard, the PD controller possesses the more suitable characteristics of controlling a feedback recommender system in our case.

4.2 Controlled Recommendation

Having learned the parameters and understood the general characteristics, we are now ready to deploy the controlled recommender systems and evaluate their performance by replacing the simulated system dynamics with a real recommender system. We first studied the initialization. We picked three different reference values (0.95, 0.90, 0.86). Without any control and the use of all the available data: they can be reached in 4, 9 and 18 days respectively, given the rate of data growth in the data set. We ran the experiments with the four controllers discussed above to see how the system behaves with respect to the change of the reference value. This is executed by adding the controllers analytically obtained in the previous section to the real system and compare the behavior of the system predicted by the analysis to the behavior of the real system.

The five-fold cross validated results are shown in Figure 7.



(a) P-I (b) PD-I (c) PI-I (d) PID-I

(e) P-II (f) PD-II (g) PI-II (h) PID-II

Figure 7: The characteristics to reach certain reference values (0.95, 0.90, 0.86) per controller (MovieLens 1m). The controllers in the first row converge faster whereas the controllers in the second row are are slower but more stable.

First, we observe that all the controllers can stabilize the system to the reference value. The behavior of the system was consistent with our understanding of the offline step response analysis. The analysis also correctly predicted which controllers were tend to overshoot, but the extend of the overshoot was less than it was anticipated. This was due to the fact that the rate of performance improvement slows as the data grows. The rate of overshooting therefore decreased when the system approached a lower reference value (better performance). The PD controller (Fig 7(c)) overshot the target, but it was faster to settle with much smaller steady state error. We also observed that the PID controllers (Figure 7 (d) and (h)) had a relatively big steady state error, mainly due to the integral part of the controller which accumulated the residual error over time.

Our observations were further quantified in Table 3 by introducing the following metrics. Settling time was measured as the time spend to settle within $\pm$ 1% of the target value. We also monitored the stability and the steady state error of the system. To measure stability of the performance over time, we define SD-SS, which measures the error from the mean; to measure the error, we define RMSE-SS as the root mean squared error from the reference value (note that we also used RMSE to measure the recommendation accuracy). Both SD-SS and RMSE-SS were calculated from the point where the system settles. The results shows that all the controllers (except the P-II) have relatively small errors. The best controllers in terms of error (RMSE-SS) were the PD-I, PID-I and PID-II controller which are marked gray in the table. The differences between the three best controllers (in terms of RMSE-SS) were not statistically different, but their values were statistically different from the other controllers.

We also observed that that the slowest controller was the P-II (which did not even reach the reference value). It was followed by the PD-II, and the rest of the controllers produced similar results (not statistically different). This differs from Table 2, where the setting time varied significantly. In the case of the PI-I controller, it had a long predicted settling time in Table 2, but we observed in Figure 7(c) that it reached the target value fast and it stayed slightly below the reference value. We believe this may be due to the fact that

Table 4: The speed, precision, stability and overshoot of the controllers. The best performing controllers in terms error are marked gray.

Type	Settling Time	RMSE-SS (error)	SD-SS (stability)	Overshoot (%)
P-I	18.2	0.00473	0.00476	0.68955
P-II	20	0.00689	0.00529	0.50125
PD-I	18	0.00448	0.00463	0.93336
PD-II	18.8	0.00549	0.00559	0.98527
PI-I	17.6	0.00807	0.00629	1.82811
PI-II	20.6	0.01376	0.00751	2.50716
PID-I	17.8	0.00641	0.00540	1.32989
PID-II	17.8	0.00914	0.00592	1.86331

(a) Reference change from RSME 0.90 to 0.925.

Type	Settling Time	RMSE-SS (error)	SD-SS (stability)	Overshoot (%)
P-I	18.5	0.00569	0.00520	0.74247
P-II	25.4	0.00763	0.00325	0.00513
PD-I	18.5	0.00499	0.00503	0.85006
PD-II	20.7	0.00547	0.00489	0.46552
PI-I	17.9	0.00580	0.00543	1.25541
PI-II	17.8	0.00489	0.00505	1.04460
PID-I	18.3	0.00546	0.00499	1.22198
PID-II	17.6	0.00552	0.00518	1.27250

(b) Untunning the parameters of the used recommendation predictor (SVD algorithm).

in practice (as in the simulation) the change of the reference value is much smaller than the theoretical change in step response. Thus, the settling happens faster in the simulation. In addition, fewer overshoots were observed in the simulation than in step response. This is mainly due to fact that increasing recommendation accuracy becomes more difficult as accuracy increases (the non-linear relationship between the input and output in Eq. (1)). In terms of stability, as predicted, the slower controllers (mark II) were more more stable their faster counterparts.

Table 4 (a) extends the experiments by considering the reference value change from RSME 0.90 to 0.925. To further evaluate the reaction of system the with respect to unexpected changes, we introduced disturbance within the un-

derlying recommender model (SVD algorithm). We ran the experiments with the same settings as before, but in day 15 we untuned the parameters of the SVD model, in order the simulate an extreme version of disturbance in the model. This change has a sudden effect on the overall recommendation accuracy. The controllers did stabilize the system by compensating the number of new training samples. Table 4 (b) summarizes the results of the experiment.

In Table 4(a) and (b), the top performing controllers were marked gray and they were statistically significant from the other ones. It is interesting to see that the PD controllers still performed consistently well in these two experiments, suggesting that the PD controllers are the best for our recommender system. We also observed that the controllers (the PI and the PID controllers) with a integral part were unstable in our system, thus not recommended. The integral part performed well when the system was required to improve the recommendation accuracy rapidly (Table 3 and Table 4(b)), but performed poorly when the performance was to be reduced (e.g., Table 4(a)).

5. EVALUATION

5.1 Computation Cost and Update Frequency

One of the benefits of the proposed feedback control is the ability of handling the trade-off between computational complexity and performance[2]. This would enable recommendation providers to have a principled tool to control the system and predict the resources needed for a predefined quality of service even if the near future usage of the system is unknown. Computing recommendations is expensive, particularly for a large scale system. The feedback controller can be fine tuned to achieve the best level of recommendation performance given limited resources. In addition, by fixing the required computational resources needed to train the model, it is possible to predict the update frequency of the system, given the resources available. As the fixed computational complexity would determine how long it would take to train the model using the available resources. We can also produce highly regular updates independent from the rate the new samples enter the system. To achieve this, two main characteristics of the system should be considered. First, it is critical to reduce the overshoot, as overshooting the target performance could threaten the stability of the system. Second, it is desirable to have the fastest response time so that it would minimize performance loss.

We evaluated the ability with the PD-II controller as it has been proved to be the best in the previous experiments. Figure 8 (a) depicts the computational efforts (measured in CPU time) for the baseline (no controller) and for the PD-II controller with two reference values (0.86 and 0.90), respectively. As illustrated, once the required performance has reached, the computational cost becomes stable. Figure 8 (b) and (c) further demonstrate this by plotting the computational gain (i.e., reduced computational effort) versus performance loss over the baseline. Setting the reference to 0.86 can save up to 15% in computation while the accuracy is reduced only by less than 1%. This difference becomes more obvious if we set the reference to 0.90, where the computational effort can be reduced by up to 55% while the performance loss is not more than 6.2%. This ratio is due to the fact that performance improvement slows down as the number of training samples increases (modeled in Eq. (1)). Moreover, the PD-II controller has 0.5% - 1.0 % overshoot predicted by the analysis and the preliminary experiments, this would guarantee that the system stays stable over time and would not go over the predefined computational cost. Therefore, this approach would enable practitioners to use the system to its *full extent* without risking the overall system stability (by going over the resources available). By controlling the computational effort we can make accurate predictions of how many times the system can be updated, so that the system can provide fresh recommendations when it is required. If we set the reference value to 0.90 (using less training samples), we can provide 3200 updates a day (with our resources it takes 27 seconds to update the model), but we could only provide 2400 updates with the reference value 0.86 where we use more training samples to train.

It is important to note that this approach shows the upper bound of the performance loss, as we chose to randomly subsample the data, but by using simple selection strategies (e.g. always train with the latest data), it would substantially reduce the performance loss. Also, the approach can easily be tailored to produce incremental updates or updates per user instead of considering the whole data as long as the system dynamics is learn on the appropriate inputs/outputs.

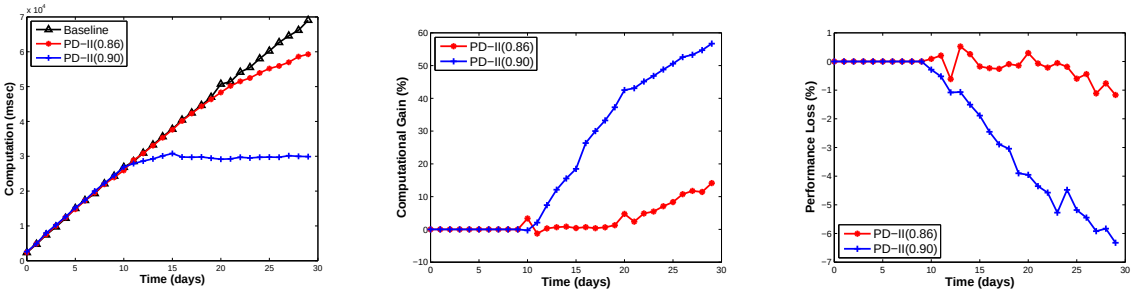
6. DISCUSSION

There are a number of shortcomings of this method that can be addressed by extending a model towards a more fine-tuned way of controller design. We chose to represent the system with one single state (performance) and consequently one controller in this paper. The reason behind this is that we aimed to concentrate on understanding the controllability of the system dynamics, and evaluate how a control-theory oriented approach can model system dynamics. This approach can be easily extended and broken down to control each individual user and item or any other way that is convenient in practice.

Instead of controlling the update frequency we can extend this approach to provide a prediction of when the system needs to be retrained given the worst performance acceptable. As the controller described in the previous section produces a signal of how many training samples are needed to converge to the reference value at a given time, this value can be used to compute the next time the system needs retraining. As shown earlier, the performance of recommender system decreases if it is not retrained periodically, this rate of decrease can be used to calculate the next time our system reaches the pre-defined reference performance. We define this as the *deterioration rate per sample* and can be estimated from historical data. It shows how long it would take for the recommender system to return to its initial performance after adding one sample. In other words, if we know that we need to remove a number of samples to reach the reference value immediately, this would be a good indication of how long it would take for the recommender system the converge to the reference value without removing any samples.

It is important to note that breaking down the performance of the system to days or even hours shows that the system dynamics is sensitive to a number of factors. These include how much the system knows about the user and the item in question, the temporal change in taste of the user, how obscure or mainstream the item is. This paper has

[2]It should be emphasized that an alternative formulation is needed to obtain the optimal control signals by directly optimizing the specific goal (either the computation costs or recommendation accuracy). We leave this formulation for future work.



(a) Computational effort (b) Computational gain over baseline (c) Performance loss over baseline

Figure 8: Computation vs performance (MovieLens 1m). The baseline approach has no controller employed. This is compared to keeping the performance at 0.86 and 0.90 respectively with the PD-I controller

not directly provided answers to these questions, but the study about the impact from the number of training sample provides a guideline and design pattern as to how to deal with those factors. If we are able to identify all the factors that constitute to the system dynamics, we might be able to stabilize them over time by designing a control loop for each individual factor, or considering them as disturbance or noise.

7. CONCLUSION

In this paper, we have shown how Modern Control Theory can help us design and analyze dynamical recommender systems. By proposing to use a simple spring and damper model to deal with the performance dynamics and deploying it with PID controllers, we have achieved a stable control of the recommender system over time. Not only providing a flexible method to handle the trade off between computational cost and performance, this approach also provides a way to identify and analyze the characteristics of the dynamics, and provides principled tools to achieve the desired objectives.

There are other fruitful opportunities in this research direction. An interesting study would be to consider what types of input signals might be useful to keep the performance steady. For instance, it is worth exploring the control strategies with active learning, that defines which samples should be added to the training set to maximize performance. Instead of choosing randomly from the availably samples the controller can follow pre-defined strategies, e.g., the data points could be selected based on their age, the estimated difficulty of the user/item etc. These factors would provide further practical solutions to manipulate the performance of the system.

The idea is also useful to apply to other online services as they all inherently exist in a time-dependent context [18]. Similar dynamics occurs in a web page content [2] and revisitation habits [1]: control theory could be used to respond to the flow of information to control the outcome of the dynamical system.

8. REFERENCES

[1] E. Adar, J. Teevan, and S. Dumais. Resonance on the Web: Web Dynamics and Revisitation Patterns. In *ACM CHI*, 2009.

[2] E. Adar, J. Teevan, S. Dumais, and J. Elsas. The Web Changes Everything: Understanding the Dynamics of Web Content. In *ACM WSDM*, 2009.

[3] G. Adomavicius and A. Tuzhilin. Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE TKDE*, 2005.

[4] H. Ahn. A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. *Information Sciences*, 2008.

[5] K. H. Ang, G. Chong, S. Member, and Y. Li. Pid control system analysis, design, and technology. In *IEEE Control Systems*, 2005.

[6] K. Astrom and T. Hagglund. *Advanced PID control*. 2006.

[7] E. Bacry, J. Delour, and J. Muzy. Multifractal random walk. *Physical Review E*, 2001.

[8] R. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *ICDM*, 2007.

[9] R. Burke. Evaluating the dynamic properties of recommendation algorithms. In *ACM RecSys*, 2010.

[10] DataSet. MovieLens: `http://www.grouplens.org/`.

[11] A. Einstein. On the theory of the brownian movement. *Annalen der Physik*, 1906.

[12] T. Friesz, J. Luque, R. Tobin, and B. Wie. Dynamic network traffic assignment considered as a continuous time optimal control problem. *Operations Research*, 1989.

[13] S. Funk. Netflix update: Try this at home. *http://sifter.org/ simon/journal/20061211.html*, 2006.

[14] J. Hamilton. *Time series analysis*. 1994.

[15] A. S. Harpale and Y. Yang. Personalized active learning for collaborative filtering. In *ACM SIGIR*, 2008.

[16] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM*, 2008.

[17] T. Jambor and J. Wang. Optimizing multiple objectives in collaborative filtering. In *ACM RecSys*, 2010.

[18] J. Kleinberg. Temporal Dynamics of On-Line Information Streams. *Data Stream Management: Processing High Speed Data Streams*, 2004.

[19] Y. Koren. Collaborative filtering with temporal dynamics. In *ACM SIGKDD*, 2009.

[20] N. Lathia. *Evaluating Collaborative Filtering Over Time*. PhD thesis, Dept. of Computer Science, University College London, 2010.

[21] N. Lathia, S. Hailes, and L. Capra. Temporal Diversity in Recommender Systems. In *ACM SIGIR*, 2010.

[22] K. Meng, Y. Wang, X. Zhang, X. C. Xiao, and G. du Zhang. Control theory based rating recommendation for reputation systems. In *Proc. of ICNSC*, 2006.

[23] M. Mull. Characteristics of High-Volume Recommender Systems. In *Recommenders Workshop*, 2006.

[24] K. Ogata. *Discrete-time control systems*. 1987.

[25] K. Ogata. *Modern Control Engineering*. 2001.

[26] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using Control Theory to Achieve Service Level Objectives In Performance Management. *Real Time Systems*, 2002.

[27] S. Rendle and L. Schmidt-Thieme. Online-Updating Regularized Kernel Matrix Factorization Models for Large-Scale Recommender Systems. In *ACM RecSys*, 2008.

[28] T. Tsunoda and M. Hoshino. Automatic metadata expansion and indirect collaborative filtering for TV program recommendation system. *Multimedia Tools and Applications*, 2008.

[29] V. Zanardi and L. Capra. Dynamic updating of online recommender systems via feed-forward controllers. In *SEAMS*, 2011.

[30] J. Ziegler and N. Nichols. Optimum settings for automatic controllers. *Journal of dynamic systems, measurement, and control*, 1993.