Real-time Recommendations from Connoisseurs

Noriaki Kawamae
Tokyo Denki University
2-2 Kanda-Nishiki-cho, Chiyoda-ku, Tokyo, Japan
Japan 101-8457

ABSTRACT

In this paper, we set the goal of real-time recommendation, to present items instantly, using an offline consumer selection method. Unlike the standard collaborative filtering approach, our offline approach focuses on just innovative consumers for making the recommendations. Since innovators exist in many communities, and their opinions will spread and then stimulate their followers to adopt the same behavior, our model, Connoisseurs over Consumers (COC), is based on the hypothesis that the sets of these innovative consumers are sufficient to represent the most representative opinions in each community. Following this hypothesis, we derive a scalable method to detect both communities and innovative consumers in each community from a large behavior log. This approach finds communities where connoisseurs having similar tastes congregate, and extracts real-time recommendations that will maximize the surprise of the consumers. Our evaluation shows that our proposed weighting method can accurately sample given logs, even though COC uses as few consumers as possible. This approach is compatible with previous algorithms for real-time recommendations.

Categories and Subject Descriptors

H.3.3 [**INFORMATION STORAGE AND RETRIEVAL**]: Information Search and RetrievalInformation filtering

Keywords

Personalization, real-time recommendations, Serendipitous Recommendations, Topic models, Nonparametric models

1. INTRODUCTION

The goal of this paper is to recommend items that will reduce the delay between their release and consumer adoption. Recommendations have been shown to be an effective and successful method of finding items of personal interest (books, dvds, movies music, news, etc.), as the volume of items available on the Internet now exceeds the ability of any individual consumer to accurately find, and assess their desirability. Against these problems, recommender systems based on Collaborative filtering (CF) have been designed to improve the consumer's experience with various goals by presenting items that consumers will probably like, from the potentially overwhelming set of choices. Since the underlying assumption in traditional CF is that similar consumers would prefer similar items, this method collects past consumer historical logs and generates personalized ranking of items by leveraging the preferences of other consumers with similar behavior. For example, many CF algorithms use similarity scores between all pairs of consumers in an attempt to improve the top-N precision; they present items in proportion to their popularity among like-minded users [20, 27]. Although consumers are eager to access fresh information and get recent items appearing in the news, we have seen that many conventional CF systems take too long to recognize emerging trends, and identify items that will satisfy the consumers' latest demands [21, 34].

Real-time recommendation is one of the most vital goals in increasing consumer stickiness and improving consumer loyalty to a service, since many items rapidly emerge and then disappear over time. Any delay results in a significant opportunity loss to content providers and time loss to potential consumers. Matching appropriate consumers to a target item at just the appropriate time is a crucial task to improve the effectiveness of real-time recommendations. For example, advertisers and merchants may want to target consumers who will be enthusiastic about these novel items. As this task requires the step of inferring consumer interests, it has attracted wide attention from the research community and yielded many models [7, 31]. Therefore, we define the task of real-time recommendations as matching potential consumers with novel items that they will like in the near future, as soon as these items appear.

We focus on serendipity as the measure of recommendation quality. Item serendipity seems be proportional to the surprise expressed when presented with the item and leads to enhanced consumer satisfaction [16]. For example, "Skyfall" is a new James Bond movie, but it is not a novel movie for many consumers. This movie has already pushed through various types of media and advertisements before its release, which tend to make many people feel overly exposed. Serendipity requires more careful selection of the items presented to each consumer.

Motivated by these requirements, we explore how to detect consumers whose logs will eventually be mirrored by

KDD'15, August 10-13, 2015, Sydney, NSW, Australia.

DOI:http://dx.doi.org/10.1145/2783258.2783260.

those of their compatriots, and can be processed to yield real-time recommendations. We call these consumers connoisseurs, and propose a latent variable model embedded nonparametric approach, Connoisseurs over Consumers (COC). COC consists of two components: (1) discovering communities present in a given log, and (2) weighting early adopters in each discovered community. In COC, every item is represented as associated with a mixture of communities, and each community attracts a set of consumers. COC weights consumers in proportion to the number of taste followers over many items in each community and defines consumers with high weight as connoisseurs. As most consumers know of or pay attention to only a small fraction of items (this is the age of big data), the huge number of available new items often requires real-time recommendations. When connoisseurs adopt new items, our method presents these items to their followers as soon as possible, by using some recommendation algorithms. Our vision of real-time recommendation thus differs from previous real-time recommendation works that suggest places that users have not visited before [33] or tweets [9].

Through experiments, we confirm the following advantages of our approach.

Theoretical contribution: Our model shows how to identify communities and corresponding early adopters as connoisseurs for performing real-time recommendations. COC considers not only consumers but also the estimated discovery time length instead of overt popularity to distinguish community layers and discover communities from a given log. As many consumers' insights could be community-dependent and vary over time, the concept of community layers allows this approach to detect and present more serendipitous items than other approaches. Additionally, connoisseurs allow us to solve the cold start problem, i.e., items that have been reviewed by few consumers.

Practical contribution: We propose a parallelized COC (PCOC) algorithm in the MapReduce programming framework[1] to realize the scalability necessary to handle the increasing prevalence of large data sets. Since our approach is a consumer-weighting method and requires only timestamped data instead of observed or explicit link structure between users, it is compatible with conventional recommendation algorithms/collaborative filtering methods, and a wide range of data.

2. PREVIOUS WORK

Research on real-time recommendations can be cast as having two aspects: 1) the analysis of network data: word-of-mouth recommendations including social networking services, and 2) the processing of streaming data or online ranking: online learning approaches based on latent factor models. Word-of-mouth recommendation has proven to be an effective approach to influence other consumers' decision making. There has been tremendous interest in the phenomenon of influence propagation in social networks recently [22]. Previous works in this area assume that there exists a social graph whose edges are labeled by the probabilities of influence between consumers. For example, viral marketing works assume that we can extract chain-reaction of influence driven via word-of-mouth by detecting the most influential consumers in the social network; the goal is to reach as many consumers as possible in the network with very small marketing cost [6]. Here, we encounter the influence maximization problem of the network: how to select an initial set of k consumers who influence the largest number of consumers in this network. Goyal et al. [12] present a mechanism that incentivizes individuals to become early adopters. Since the phenomenon of consumers recommending favorites to friends and followers plays an important role in shaping consumers' behaviors and their collection, a deep understanding of word-of-mouth recommendations leads to interesting research topics[15]. Other research focuses on influence maximization[3, 6]; the goal is to find a small set of seed nodes that can trigger a viral marketing event that can cover the maximal range of nodes on an underlying social network.

Since the trends in various social media networks and consumer generated content are highly unpredictable [32], ways of harnessing real-time and time sensitive data, such as tweets, have been studied by many researchers and has lead to applications that can recommend new items. While many of these applications utilize a given consumer-consumer network and build on the assumption that influence exists as a real phenomenon, in fact, the rich information sources needed are generally unavailable [2]. Consequently, we need an alternative, a way to find influencers or early adopters from the available data sets [9]. This is the reason why our approach uses a consumer behavior log as input for recommendations.

Although the improving consumer satisfaction is one recommendation goal, less attention has been paid to studying the quality and impact of recommended items. That is, the most popular items in any log collection could be presented to consumers with high probability, but they are often broadly familiar to them. Hence, highly accurate recommendations appear far too obvious and of little help to consumers, and seem to fail to enhance consumer satisfaction [8, 35]. In fact, recommendation systems that appropriately discount popularity lead to an increase in total sales volume [11, 13]. Our approach discovers layers of preferences to distinguish communities and thus to weight surprising items. Many existing CF methods overlook and previous real-time recommendation methods ignore these differences [26].

Additionally, we develop a distributed learning algorithm for our approach based on topic models. A promising approach to scaling topic models over large data sets is to distribute and parallelize both the data and the topic modeling algorithms over multiple processors [23, 25]. Mimno et al. [23] propose parallel algorithms for the collapsed sampler for LDA, while Newman et al. [25] parallelize the Hierarchical Dirichlet Processes (HDP) [30] through the straightforward mapping approach and show that their algorithm produces the same perplexity as the single-processor version of HDP. We continue this development for hierarchical topic modeling with HDP using collapsed Gibbs sampling.

3. PROBLEM FORMULATION

We give here our definition of real-time recommendations, introduce the concepts of connoisseurs, communities, serendip-

[1] Apache™Hadoop®: http://hadoop.apache.org/

[2] http://www2012.wwwconference.org/media/videos/keynote-tbl/

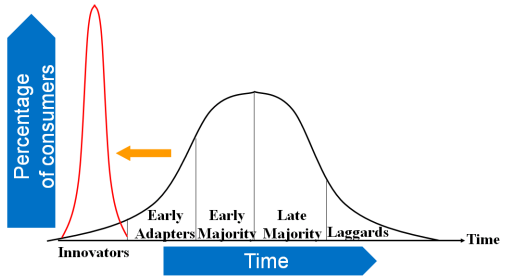

Figure 1: Adoption curve: The black line curve denotes the adoption curve of normal diffusion, and the red line curve denotes the adoption curve of the expected diffusion boosted by our real-time recommendations.

ity, and consider the scalability of algorithms. Connoisseurs and communities are basic requirements for performing these recommendations, and serendipity is the measure of recommendation quality. Scalable algorithms are inevitable if we are to deal with large data sets as they are beyond the capacity of a single node machine.

Real-time recommendations: We set the goal of real-time recommendations, to match novel items, that will become popular in the near future, with potential consumers as soon as these items appear. As shown in Figure 1, the process of consumers' adoption over time is typically illustrated as a normal distribution [28]. This curve indicates that the first consumers to try a new product, "innovators", are followed by "early adopters". Next come the early and late majority, and the last group to adopt the product are called "laggards". Following this definition, our definition of real-time recommendations is to present items newly selected by innovators to early adopters, early majority, late majority, and laggards, and thereby bias the adoption curve to the left shown in Figure 1. That is, the effectiveness of this approach lies in the degree to which adoption delay can be shortened.

Connoisseurs: We define connoisseurs as early innovators who anticipate and identify items, that many other consumers do not know about now, but will like later, soon after their release. While innovators are associated with individual items as shown in Figure 1, connoisseurs are defined over popular items and a community. Following this definition, we weight the importance of each consumer in proportion to the number of consumers who purchase the same item after him, and sum this weight over the items.

Community: Since the innovative degree of each consumer is community dependent, innovators of data science may not be the innovators in the fields of life-science or marketing-science. Even connoisseurs and their followers in the same community, exhibit different behavior to the same item, and influence others differently. These characteristics require us to form communities using both consumers and their temporal tastes. Since some items are "common items" that are not surprising and can not be used for discriminating consumer preference, our approach distinguishes the background preference layer from the others before putting consumers having similar preferences into the same community. For example, "Skyfall" and "The Dark Knight Rises" have been seen by so many people that their selection tells us nothing about differences in taste, and so are judged to belong to the background. On the other hand, "The Human Centipede" and "The Revenant" have been seen by far fewer people and so are good indicators of taste as well their fans, who thus can be judged as belonging to a community. This leads us to distinguish preference layers and explore both communities and the corresponding connoisseurs.

Serendipity: We focus on serendipity [14, 16] as the measure of recommendation quality. While improving prediction accuracy (i.e., p@N, precision, and recall) is a very important goal of recommendations, if the recommended items are obvious to the consumers, adoption rates will be poor. This is why we focus on serendipity for measuring recommendation quality and improving consumer satisfaction. A serendipitous recommendation helps consumers find surprisingly interesting items they might not have otherwise discovered [14]. The degree of the surprise that a given item brings a consumer is proportional to the difference between this item's release time (or first appear time) and the estimated time at which the target user is likely to find the item without the recommendation(we call this the estimated discovery time), which is defined as the average (avg) elapsed (AE) [16]. This measure coincides with the effect of our defined real-time recommendations shown previously.

Since the goal of our real-time recommendation definition is to match many consumers with target items soon after their release, our weighting method aims to enable conventional algorithms to present items that could surprise and give the consumers more serendipity than other items.

Algorithm scalability : The sheer scale of modern log sets raises significant challenges to recommender systems and their machine learning algorithms, particularly in terms of computation time and memory requirements. The primary motivation for developing a distributed algorithm for COC, parallelized COC (PCOC), is to ensure high scalability in terms of memory and computation time, and thus perform real-time ranking over a large scale data sets.

4. CONNOISSEURS OVER CONSUMERS

4.1 Design of COC

4.1.1 Motivation of COC

As shown in Figure 1, our real-time recommendation definition is to bias (move) the adoption curve to the left of normal diffusion. Therefore, real-time recommendations need to address the metric of "time" for quantifying serendipity, and "popularity". This requires us to employ the temporal information on which communities appear and who are connoisseurs in each community. In this paper, we need to map from a text-oriented time-aware model to the consumer-item space. This is why we build a new generative model that explains both communities and the corresponding connoisseurs by extending Switch TOT (sTOT) model [17], one of the extensions of latent Dirichlet allocation (LDA)[4]. Given this requirement, Connoisseurs over Consumers model (COC) is designed to group consumers with similar tastes into a community and their adoption patterns over time in a given community to find the connoisseurs. This model weights the consumers in each community similar to how sTOT gains probabilities over words via topics, where consumers, and communities correspond to words, and topics in this model, respectively.

Before detailing our model, let us review the concept of sTOT. Table 1 shows the notation used in this paper; Fig-

ure 2 shows a graphical representation of sTOT and our proposal, COC. sTOT explicitly models time jointly with word co-occurrence patterns, by focusing on discriminating trend-specific words and background topic words in each generative process of a given set of time-stamped documents. In this model, topics are responsible for generating both observed timestamps and the corresponding words. The background topic is common to almost all documents regardless of their content and time; they generate non temporal words, while the other topic variables generate temporal words. For distinguishing temporal words from background topic words, sTOT defines r, it acts as a switch to handle these words in each token, and takes value r_{ji}=0 if the i-th word w_{ji} in j is generated via the background topic variable, or r_{ji}=1 if word w_{ji} is generated via the topic variable.

In this paper, each log is mined for each purchased item, and becomes a record of consumers who purchased this item and their time-stamps. Like other conventional topic models, COC models consumer behavior logs, each of which stores a consumer's actions (click, check, purchase, review, etc.) on items, as a mixture of underlying communities, and each item attracts each consumer via each community. As shown in Figure 2, items purchased by similar consumers in the same period can reasonably be considered to indicate the existence of a community in COC, while documents having both a similar timestamp and words can be considered to share a common trend topic in sTOT. Unlike sTOT, COC associates a continuous distribution of discounted number of followers with each community, and exploits community specific trends and connoisseurs to allow us to forecast these trends. Since the number of communities is unknown in advance and changes in each data set, COC extends sTOT by employing a nonparametric approach and novel weighting methods. COC focuses on 1) handling the difference between communities, and 2) modeling the relationship between consumers and their influence in each community. That is, COC employs switch variable r_{ji} to distinguish communities, and the conditional probability distribution of two observed variables to capture the relationship.

4.1.2 Consumer weighting

COC defines consumer weighting function, f_{ji}, which corresponds to t_j in sTOT, while u_{ji} is drawn from community specific multinomial distribution ϕ like words in sTOT. f_{ji} is designed to weight consumers in proportion to both the length of interval between the item release time and the estimated discover time of a consumer on given item j, and its item popularity. As consumer preferences change over time, consumer patterns present in the early stage are generally of no practical use for later recommendations. Therefore, we use exponential decay to discount the number of consumers, and so suppress the discounted number of followers. This allows communities to be responsible for generating both connoisseurs as well as followers, and is given by:

$$f_{ji}(t) = N_{ji} \times v_{c_{ji}}(t), \tag{1}$$

where N_j denotes the weight of the i-th consumer for the j-th item, and $v_{c_{ji}}(t)$ denotes the weight of community where the j-th item attracts that i-th consumer at current time, t. Both these functions are time-sensitive, and defined from current time t, as below:

Table 1: Notations used in this paper

SYMBOL	DESCRIPTION
M (C)	number of items (communities)
U	number of consumers
N_j	number of consumers for item j
w_{ji}	the i-th word in document j
r_{ji}	the switch variable of i-th consumer
f_{ji}	consumer weighting function: the discounted followers of consumer i for item j
z_{ji}	the topic associated with the ith token
c_{ji}	the i-th community for item j
u_{ji}	the i-th consumer for item j
t_j	the time-stamp of document j
t_{ji}	the time when the i-th consumer adopts item j
$\bar{\theta}_j$	item j specific distribution of communities
θ_c	the weight of community c
μ_j	the j-th item specific distribution of r_{ji}
$\phi_c(\phi_j, \phi_b)$	community c (item j, background b) specific multinomial distribution of consumers $\phi_c(\phi_j, \phi_b) \sim Multinomial(\beta_c(\beta_j, \beta_b))$
λ_c	the parameters of community c
	specific exponential distribution of f
α	the corpus specific parameter $\alpha \sim Gamma(a, b)$
α_c	the community c specific parameter $\alpha_c \sim Gamma(a, b)$
α_j	the log (item j) specific parameter $\alpha_j \sim Gamma(a, b)$
β	the community specific parameter $\beta \sim Gamma(c, d)$
γ_j	the item j specific parameter $\gamma_j \sim Gamma(e, f)$

$$N_{ji} = \frac{N_{ji}}{N_j} \exp(-\frac{t_{ji} - t_{j*}}{\bar{t}_j}),$$
$$v_c(t) = \lambda_c \exp(-\lambda_c(t - t_{ji})) = \frac{1}{\bar{t}_c} \exp(-\frac{(t - t_{ji})}{\bar{t}_c}), \tag{2}$$

where N_{ji} denotes the number of consumers who have adopted item j after consumer i, N_j denotes the number of consumers who have adopted the j-th item, t_{ji} denotes the adoption time of j by i, t_{j*} denotes the first time when j-th was purchased j, $\bar{t}_j$ denotes the average of $t_{ji} - t_{j*}$ over all consumers, and $\bar{t}_c$ denotes the average of $t - t_{ji}$ over all consumers who have purchased j and are assigned to c. For example, when a new significant consumer co-adoption pattern emerges, COC will discover a community that has a steep curve. Because connoisseurs mean not only being early adopters but also having many followers in each community, items that have recently been purchased by connoisseurs will be purchased by their followers in the near future.

4.1.3 Communities

As each community consists of many consumers showing similar temporal preferences, communities can also change over time. Our approach focuses on separating community-specific consumers and background consumers in describing the generative process of a given time-stamped behavior log. Background consumers globally share items regardless of category and time; the background community generates these non temporal consumers, while the other communities generate dynamic consumers. For distinguishing dynamic consumers from background consumers to detect the differ-

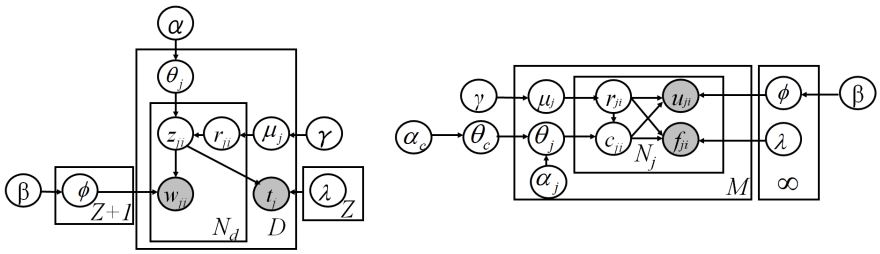

Figure 2: Graphical Models of sTOT and COC: In this figure, shaded and unshaded variables indicate observed and latent variables, respectively. An arrow indicates a conditional dependency between variables and stacked panes indicate a repeated sampling with the iteration number shown. COC uses an exponential decay to quantify each consumer as regards connoisseurship instead of the beta distribution in sTOT, as this model allows each member to follow a multinomial distribution like words in sTOT. Intuitively, COC is a unified probabilistic model that 1) captures the interdependency between the number of discounted followers, f, and the corresponding consumers, u using λ via c, and 2) handles the process of attracting a consumer in c_{ji}, and then 3) represents both a consumer and the corresponding number of discounted followers.

ences between consumers and identify connoisseurs, we extend r_{ji} as a switch for handling more kinds of communities as follows. If r_{ji}=0, COC selects the background community that is responsible for generating a background consumer. If r_{ji}=1, COC selects an item-specific community that consists of consumers who purchased only this item. If r_{ji}=2, COC selects the community that consists of dynamic consumers over short periods. Note that a community yields both consumers and discounted numbers of followers considering timestamps, while the background community generates only consumers.

4.2 Generative Process of COC

COC first samples community c for a given item from the community distribution of j-th item, θ_j, and generates both consumers and their corresponding values from the community specific consumer distribution and the weight function. Since the merit of nonparametric Bayesian methods is that they provide an explicit probabilistic explanation and automatic determination of the number of communities, COC incorporates HDP that is obtained by coupling draws from a Dirichlet process (DP) [10]. DP is a stochastic process used in Bayesian nonparametric models of data. Sethuraman [29] showed that measure G drawn from a DP is discrete, and defined by the stick breaking process.

As shown in Figure 2 with the notations in Table 1, the generative process of COC is defined as follows:

- $\tilde{\theta}_c \sim Beta(1, \alpha_c)$, $\bar{\theta}_c \sim \tilde{\theta}_c \prod_{i=1}^{c-1}(1-\tilde{\theta}_i)$.
- For each community indicator, $\phi_c \sim Dirichlet(\beta)$.

For each item j ($j = 1$ to M)

- $\theta_j \sim DP(\alpha_j, \theta_c)$, where θ_c represents a vector of $\bar{\theta}_c$;
- $\mu_j \sim Dirichlet(\gamma)$;
- For i-th ($i = 1$ to N_j) token in the j-th item,
  - $r_{ji} \sim Multinomial(\mu_j)$.
  - if $r_{ji} = 0$
    - $u_{ji} \sim Multinomial(\phi_b)$.
  - else if $r_{ji} = 1$
    - $u_{ji} \sim Multinomial(\phi_j)$.
  - else if $r_{ji} = 2$
    - $c_{ji} \sim Multinomial(\theta_j)$,
    - $u_{ji} \sim Multinomial(\phi_c)$.

We repeat this generative process for all items and consumers in a given log.

4.3 Inference in COC

Approximate inferencing over hidden topic variables is performed by Gibbs sampling, as it has been developed to handle large numbers of latent variables and parameters. This is why we employ this sampling in this paper. We can obtain a Gibbs sampler for COC, as HDP can be represented by the Chinese restaurant process (CRP) [1], since parameters can be integrated out. CRP, a stochastic process that generates partitions of integers and is used in DP [24], yields the same clustering structures as created by a Dirichlet process [10]. Alternatively, we can view this process in terms of $CRP(\gamma, G_0)$ and compute the probability of the i-th customer sitting at table k in restaurant j as follows:

$$P(z_i = k|\mathbf{z}_{\setminus i}, \gamma) = \begin{cases} \frac{n_{jk}}{\sum_k n_{jk}+\gamma}, k \text{ is an existing table,} \\ \frac{\gamma}{\sum_k n_{jk}+\gamma}, k \text{ is a new table.} \end{cases} \quad (3)$$

where $\mathbf{z}_{\setminus i}$ is the seating arrangement of the current $i-1$ customers, and n_{jk} is the number of customers sitting at table k in restaurant j.

Starting from the joint distribution $P(\mathbf{u}, \mathbf{r}, \mathbf{c}, \phi, \theta, \mu|\alpha, \beta, \gamma, \eta, \epsilon, \mathbf{t}, \lambda)$, we can work out the conditional distribution as:

$$P(c_{ji} = k, r_{ji} = r|\mathbf{u}, \mathbf{c}_{\setminus ji}, \mathbf{r}_{\setminus ji}, \mathbf{t}, \alpha, \beta, \gamma, \eta, \epsilon, \lambda)$$
$$\propto \begin{cases} \frac{n_{jk\setminus ji}}{\sum_c^C n_{jc\setminus ji}+\alpha_j}, \\ \text{if } k \text{ is an existing community for item } j, \\ \frac{\alpha_j}{\sum_c^C n_{jc\setminus ji}+\alpha_j}\left(\frac{n_k}{\sum_c^C n_c+\alpha_c} + \frac{\alpha_k}{\sum_c^C n_c+\alpha_c}\right), \\ \text{if } k \text{ is a new community for item } j, \end{cases}$$
$$\times \begin{cases} \frac{n_{j0\setminus ji}+\gamma_0}{\sum_r^R(n_{jr\setminus ji}+\gamma_r)} \frac{n_{bu\setminus ji}+\beta_u}{\sum_u^U n_{bu\setminus ji}+\beta_u)}, \text{if} \quad r_{ji} = 0, \\ \frac{n_{j1\setminus ji}+\gamma_1}{\sum_r^R(n_{jr\setminus di}+\gamma_r)} \frac{n_{du\setminus ji}+\beta_u}{\sum_u^U n_{du\setminus ji}+\beta_u)}, \text{if} \quad r_{ji} = 1, \\ \frac{n_{j3\setminus ji}+\gamma_3}{\sum_r^R(n_{jr\setminus di}+\gamma_r)} \frac{n_{cu\setminus ji}+\beta_u}{\sum_u^U n_{cu\setminus ji}+\beta_u)} \times f_{ji}(t), \\ \text{if} \quad r_{ji} = 2. \end{cases} \quad (4)$$

where $n_{jk\setminus ji}$ represents the number of communities that have been assigned to k in item j, except i in j, $n_{j0(1,2,3)\setminus ji}$

represents the number of communities assigned to switch r = 0; background (1: item specific, 2: community) in item j, except i in j, n_k represents the number of communities that have been assigned to k, and $n_{bu(du,cu)\backslash ji}$ represents the number of consumers u in background community (item specific, community c specific), except i in j. Since each consumer appearance pattern and its discounted weight of followers are assumed to be generated conditionally on community c, the resulting exponential and multinomial parameters will correspond. A discounted number of followers having a high probability under a certain community will likely appear with the set of consumers having a high probability in the same community. Each new consumer/follower is generated by again selecting from community c and repeating the entire process. Thus, we can view c as a high level representation of the ensemble of consumer/followers pairs in terms of a probability distribution over the factors that each consumer/followers can be assembled from.

4.4 Recommendations

Because consumers do not always need products, matching consumers to each item on a timely basis is an appropriate task for real-time recommendations. In practice, novel items are sufficient for connoisseurs, while only items adopted by these connoisseurs are necessary for their followers. Therefore, our approach considers items that connoisseurs already known about and have accepted as candidate items for recommendations. COC weights consumer i by f_{ji} and selects connoisseurs in descending order of f_{ji}. We normalize f_{ji} by dividing it by the number of all consumers choosing item j, N_j, and sum up these normalized values of all chosen items to avoid a one-hit wonder. Then, we select connoisseurs in descending order of this weight. Therefore, identifying such connoisseurs in commonly available log data sets is sufficient to permit the realization of real-time recommendations. It predicts which items consumers might adopt in the near future, while trends captured by previous models are general interests and so are not suitable for making attractive recommendations to each consumer. COC uses communities to compute the prediction score for a given consumer-item pair.

Only if connoisseurs adopt brand new items, we define the consumer ranking for each item as follows:

$$P(i|j, \theta_j, \mu_j, \lambda_j, \phi_c, \alpha_j, \beta, \gamma) \triangleq \sum_{c=1}^{C} \hat{\theta}_{jc}\hat{\phi}_{ci}, \tag{5}$$

where $\hat{\theta}_{jc}$ and $\hat{\phi}_{ci}$ are calculated as follows:

$$\hat{\theta}_{jc} = \frac{n_{jc} + \alpha_c}{\sum_c n_{jc} + \alpha_c}, \hat{\phi}_{ci} = \frac{n_{ci} + \beta_i}{\sum_i n_{ci} + \beta_i}. \tag{6}$$

As COC learns communities at the item level, this approach presents brand new items to consumers via connoisseurs.

4.5 PCOC: Distributed Algorithm of COC

Our distributed algorithm is inspired by the Approximate Distribution Hierarchical Dirichlet Processes (AD-HDP) [25] which allow task distribution over P distinct processors; we implement it in the framework of Hadoop, see Algorithm 1 and 2.

This algorithm is composed of two steps: a Gibbs sampling step (Map) and a synchronization step (Reduce), and each document is assigned to only one processor. As the data, parameters, and computation are distributed over distinct processors, our model can handle large data sets. That is, documents D (community assignment $\mathbf{c}$) are partitioned into $D = \{D_1, ..., D_p, ..., D_P\}$ ($\mathbf{c} = \{\mathbf{c}_1, ..., \mathbf{c}_p, ..., \mathbf{c}_P\}$ where each processor p stores p stores D_p ($\mathbf{c}_p$), In the sampling step, each processor samples its local community assignments ($\mathbf{c}_p$), that are community assignments of items stored in processor p, on each processor by using both previous community assignment $\mathbf{c}_p$, and the global counts and parameters. The difficulty of distributing and parallelizing Gibbs sampling updates is caused by the fact that Gibbs sampling is inherently sequential and no topic assignment can be concurrently updated with any other topic assignment.

After processor p updates $\mathbf{c}_p$, and swept through its local data, it modifies count, and sends the result to the master node. Although the item-community counts $\mathbf{n}_{jcp}$, that is the number of community c of item j stored in processor p, $\mathbf{n}_{jrp}$, and $\mathbf{n}_{jup}$ will be consistent with the community assignments $\mathbf{c}_p$. On the other hand, the consumer-community counts $\mathbf{n}_{cup}$ ($\mathbf{n}_{bup}$, $\mathbf{n}_{cp}$), that is the number of consumer u associated with community c (the background topic), and the number of community c stored in processor p, will be different on each processor and not globally consistent with $\mathbf{c}_p$. During this step, individual processors may generate new topics, according to Eq (4), where each count n_* is replaced by n_{*p}.

In the synchronization step, the master node aggregates the local counts from each processor to update a single set of globally consistent counts. To limit unnecessary growth in the number of topics and to synchronize topics, we merge newly found topics that were created on some processors using topic id k, as described by Newman et al. [25]. After this synchronization and update, each processor receives the latest counts from the master node, and so shares the same values with all other processors. For example, the update of count $\mathbf{n}_{cu}$ is given by

$$\mathbf{n}_{cu} \leftarrow \mathbf{n}_{cu} + \sum_p (\mathbf{n}_{cu} - \mathbf{n}_{cup}) \tag{7}$$

where $\mathbf{n}_{cup}$ denotes the number of consumers, u, in community c on processor p. n_c, n_{ju}, n_{bu} can also be updated in a similar manner.

Algorithm 1 Map Phase of PCOC: Distributed Algorithm for COC

```
// Initialization
// Input
<Key,Value>:=<item ID j, customer record u_ji, t_ji>
// Gibbs sampling over burn-in and sampling period
for iteration=1 to N_iteration do
  for each processor p in Parallel P do
    for i = 1 to N_j for each item m in M_p do
      Gibbs sampler: drawn c_ji, r_ji using Eq (4) with n_*p
      instead of n_*
      Report n_kp, n_bup, n_kup, sum_i t_ji, to the master node
    end for
  end for
  Send <Key,Value>:=<k, n_ku(n_bu, sum_i t_ji)>
  update n_ju
end for
```

This process is repeated for either a fixed number of iterations or until the algorithm has converged. To ensure practi-

Algorithm 2 Reduce Phase of PCOC: Distributed Algorithm for COC

```
// Input
<Key,Value>:=<k, n_ku(n_bu, Σ_i t_ji)>
// Gibbs sampling over burn-in and sampling period
if Value = w(u) then
  update n_ku(n_bu) using Eq (7)
  calculate t̄_c, λ_c from Σ_i t_ji and n_k
end if
Merge topics
Broadcast n_k, n_ku(n_bu), λ_k via Memcached
```

cality, we implement this algorithm on Hadoop[3], where each processor corresponds to a data node with a task tracker, each sampling step is performed by a mapper, and each synchronization step is performed by a reducer. Additionally, we use Memcached [4] instead of HDFS, a file system of Hadoop, to store $n_c, n_{bu}, n_{cu}, \lambda_c$ and broadcast them.

measure.

5. EVALUATION FRAMEWORK

5.1 Data sets

We focus here on the extraction of preference trends, and use the following data to qualitatively and quantitatively evaluate the proposed model.

Netflix : Netflix data[5]; This data set consists of 100,480,507 ratings that 480,189 users gave to 17,770 movies from Nov 1st, 1999 to Dec 31st, 2005. We set the period from Oct 1st, 2005 to Dec 31st, 2005 as the test period, and the rest as the learning period.

Last.fm data set[6]: was given directly by [5]. The dataset contains 992 users, 107,528 tracks and 19,150,868 lines.

Twitter: Twitter data[7]: Each line of this data consists of consumer id, tweet id, date, hashtag (option), number of followers, number of feedbacks, timestamps, and so on. In our experiments, we used only the tuple of <consumer id, retweet id, timestamp>. We selected tweets from 30/08/2012 to 29/01/2013, yielding a corpus consisting of 65,456,988 users and 321,513,597 tweets. In our evaluation, we ran the experiments on PCs with Dual Core 2.66 GHz Xeon processors.

5.2 Design of experiments

The tasks of recommendation schemes are either to predict the ratings of remaining entries, or to produce a list of unseen items for each consumer. The aim of this experiment is to answer three questions

- "Can COC detect connoisseurs, consumers who forecast trends in each community?",
- "Can COC model distinguish communities from the background?",
- "Can the history log of connoisseurs be useful for realizing real-time recommendations?",
- "Can PCOC, an approximated and distributed variant of COC, be applied to a large scale data sets?".

To evaluate models on the task of capturing an individual's preference, we compute performance on the collaborative filtering task. We conducted simulations to evaluate the predictive performance of recommendations via K cross-validation where the original data is partitioned into K sub samples at random. Each subsample is appropriately divided into test data (the latest 2months), and learning data(the rest of data). We call the data in the test period the test data and that in the learning period the learning data. The simulations treat each consumer/user in the test data as a target consumer to whom we applied each of the recommendation methods by using the consumer logs collected from the learning data.

6. EXPERIMENTS

6.1 Predictive performance

The aim of this experiment is to evaluate the effectiveness of our method for consumer selection.

First, we explore the characteristic of communities in view of preference similarity; the results are shown in Table 2. We selected "Batman Begins" as a relatively new movie (2005), and "Toy Story" as an older movie (2003) from the Netflix movie data set. For each selected movie, we counted how many other movie consumers rated it, and ranked the top 5 movies by the number of consumers. While Toy Story has been evaluated over the long term and the difference between methods seems to be small, Batman Begins has been evaluated in the short term and the difference between methods seems to be significant. Although time-independent methods (all and sim_r) heavily weight very popular movies such as "PC" and the series of "Lord of the Rings", time-dependent methods (sim_t and COC) lower them. Since these famous movies would not be rated again, time-independent methods could group movies that represent the near future of similar preferences.

Second, we examine the effect of COC on recommendations by incorporating consumers identified as connoisseurs into Slope One [19], non-trivial item-based collaborative filtering on ratings, and compared its prediction performance against that of a conventional model based recommendation method, singular value decomposition (SVD++) [18].

In this experiment, the score of the target consumer in the test period is predicted by the slope one algorithm, which is trained in the learning period, where we predict a rating score in Netflix, As a baseline of consumer/user ranking in each community to evaluate which weighting method is more effective, we selected 20(10,5)% consumer's logs by the number of followers, N_{ji} of Eq 1(i.e. customers who rated the same item (rated the same author)),#FL, or by the latest date order, LD. We employ the mean absolute error (MAE), a metric used to determine how close forecasts or predictions are to being correct. As shown in Figure 3, this result shows that 1) consumers selected by COC represent the general population, even at only 5% of all consumers, and 2) COC identifies more connoisseurs than the other methods do, as the selected consumers are sufficient to capture the most representative opinions for the given data and improve the

[3]Apache™Hadoop®: http://hadoop.apache.org/
[4]Memcached: http://memcached.org/
[5]Netflix: http://www.netflixprize.com/
[6]http: //http://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/index.html
[7]Twitter: http://twitter.com

Table 2: Comparison of similar movies: PC is an abbreviated title of "Pirates of the Caribbean: The Curse of the Black Pearl". In method column, the (sim_r, sim_t) entries indicate the co-rating consumers with no conditions (within rating difference under 1, within one week), and COC calculates the similarity between items using Kullback-Leibler divergence for the community of item, θ_j.

title	method	top 5 movies
Batman Begins	all	PC, Independence Day, Lord of the Rings I, The Incredibles, Lord of the Rings II
	sim_r	PC, Lord of the Rings II, The Incredibles, Lord of the Rings I, Indiana Jones and the Last Crusade
	sim_t	Hitch, National Treasure, Million Dollar Baby, Constantine, The Incredibles
	COC	Hitch, Million Dollar Baby, The Incredibles, National Treasure, The Aviator, Constantine
Toy Story	all	Finding Nemo, Forrest Gump, PC, Shrek, Monsters
	sim_r	Finding Nemo, Monsters, Shrek, Forrest Gump, The Incredibles
	sim_t	Finding Nemo, Shrek, The Incredibles, Forrest Gump, Monsters
	COC	Finding Nemo, Shrek, The Incredibles, Monsters, Raiders of the Lost Ark

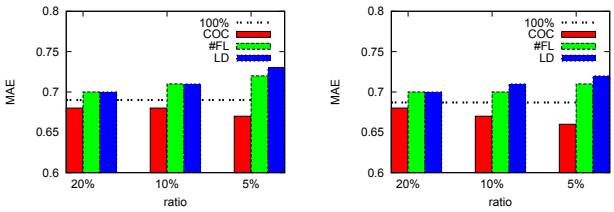

Figure 3: Comparison of MAE versus ratio of consumers using (left)Slope-one, (right)SVD++

score without being obstructed by noisy data or data sparseness. Additionally, we notice that straightforward methods that depend on the number of followers or latest logs could reduce the rating accuracy. Consequently, COC answers affirmatively one of the experiment's questions, 1)"Can our model detect early consumers (connoisseurs) who forecast trends in each community?".

6.2 Ranking performance

To evaluate the quality of the proposed method, we used top N precision, a measure commonly used for evaluating the performance of CF systems in suggesting top-N items to a consumer. We applied the proposed method and a baseline method to the data sets and compared the precision of their top-N item recommendations (N value is 10). We then presented the top N ranked items to the target consumer according to the probability that each algorithm assigned to them, and confirmed that these recommended items existed in the test data. Here, we recommend items in Netflix and tweets, that could be retweeted, in Last.fm dataset. For this recommendation task, we incorporated COC into Item-based collaborative filtering recommendation algorithms [2], and, as a baseline, prepared *Slope one*, which recommends items by the high predicted rating.

Since our goal is to provide the user with real-time recommendations that offer serendipitous discoveries, we also evaluated the performance in terms of "IC" (Item coverage: the percentage of the number of unique items appearing in the top-N recommendations over the total number of unique items), the Gini coefficient (a measure of the statistical dispersion of the distribution of consumers over items), "AE' (avg elapsed), and "AD" (avg difference) [16]; these results are shown in Table 3. These measures were proposed to evaluate the quality of recommended items in view of serendipity and are explained more detail in [16].

We can see that COC achieves the highest item coverage than the others, as well as achieving the greater differences in terms of AD and AE. More precisely, COC clarifies this phenomenon by reducing the number of consumers whose log is referenced when making recommendations. The fact that COC improves the item coverage indicates that it can offer different recommendations to several like-minded consumers, a necessary condition for serendipitous recommendation. These results imply that COC 1) helps consumers find not only items that match their latest preferences, but also novel and surprisingly interesting items he might not have otherwise discovered by excluding consumers associated with the background, and 2) can recommend a wide range of items with much less bias than the other methods, as well as assisting in the task of real-time recommendations. Because AE is small, novel items, those that have not adopted by many consumers, could be presented to each target consumer. Consequently, COC answers affirmatively the last two questions of the experiment, 2)"Can our model distinguish communities from a background community?", and 3)"Can the history log of connoisseurs be useful in performing real-time recommendations?".

6.3 Scalability

This experiment aims to investigate how our parallel algorithm of PCOC performs compared to its sequential model (original model), COC. Scalability is one of the key requirements for realizing real-time recommendations from large scale data sets. We are interested in two aspects of performance: the quality of the model learned, measured by the perplexity of test data; and the time taken to learn the model. We computed test set perplexity for a range of topics K, and for a number of processors, P, ranging from 1 to 80. The distribution algorithms were initialized by first randomly assigning communities to each user (consumer), in $\mathbf{z}$, then counting communities in log/tweet, N_{cjp}, and consumer u in community c, N_{ucp}, for each processor.

To compare our distributed topic model algorithm, PCOC, with COC, we used test perplexity, which is computed over two data sets. As COC is an extension of topic models, we use the test-set perplexity as evidence that partly supports scalability. Perplexity, which is widely used in the language modeling community to assess the predictive power of a model, is algebraically equivalent to the inverse of the geometric mean per-word likelihood (lower numbers are better). We computed the perplexity as follows. First, we randomly split 10% of each log associated with each item (Netflix), tweet text (Twitter), to create a test part; the remainder was used as the learning part. For every item/tweet, the test part was held out to compute perplexity. Second, the learn-

Table 3: Comparison of top N item ranking versus ratio of consumers: p@10 denotes the precision for the top 10 recommended items. Both AD and AE have units of days. Results that differ significantly, t-test $p < 0.01$, from #FL and LD on the same ratio are highlighted by underline.

		Netflix								LastFM							
	N	20				10				20				10			
Slope-one		p@N	IC	AD	AE	p@N	IC	AD	AE	p@N	IC	AD	AE	p@N	IC	AD	AE
	Full	0.471	0.23	32.3	653.3	0.256	0.17	31.1	712.5	0.185	0.25	5.6	34.2	0.156	0.18	5.5	31.7
#FL	20%	0.421	0.21	30.6	673.2	0.202	0.09	27.7	732.9	0.173	0.21	5.1	41.1	0.082	0.11	5.3	36.3
	10%	0.351	0.18	28.3	679.4	0.167	0.07	25.2	733.1	0.148	0.18	4.7	42.7	0.053	0.07	5.1	38.1
	5%	0.086	0.05	22.5	725.5	0.042	0.02	23.3	736.2	0.093	0.11	4.2	43.1	0.026	0.02	4.8	40.4
LD	20%	0.432	0.27	31.3	572.6	0.271	0.12	23.3	562.3	0.211	0.25	5.7	40.2	0.111	0.13	5.7	33.2
	10%	0.425	0.31	33.4	523.3	0.276	0.14	24.7	523.8	0.226	0.28	5.8	37.1	0.122	0.15	5.8	32.1
	5%	0.125	0.32	33.8	432.8	0.098	0.16	25.1	413.3	0.142	0.29	5.6	35.2	0.077	0.16	5.8	31.4
COC	20%	0.477	0.25	33.6	612.8	0.271	0.18	31.7	588.1	0.188	0.27	5.7	33.3	0.152	0.18	5.6	31.4
	10%	0.481	0.31	33.9	498.3	0.273	0.19	32.2	523.9	0.185	0.29	5.8	34.0	0.154	0.18	5.5	31.3
	5%	0.483	0.33	34.6	424.2	0.277	0.20	32.8	401.3	0.181	0.32	5.7	32.0	0.153	0.18	5.6	30.6
Item	Full	0.475	0.24	32.2	651.8	0.259	0.18	31.3	707.8	0.187	0.27	5.7	33.8	0.155	0.17	5.4	34.3
#FL	20%	0.425	0.22	32.7	662.6	0.221	0.09	28.1	729.8	0.175	0.20	5.2	43.2	0.080	0.10	5.1	38.4
	10%	0.356	0.19	28.8	674.2	0.173	0.07	25.6	683.8	0.151	0.19	4.8	43.3	0.052	0.06	5.0	44.2
	5%	0.088	0.05	23.1	703.3	0.046	0.02	25.6	710.0	0.094	0.10	4.4	42.5	0.027	0.02	5.0	39.8
LD	20%	0.455	0.28	33.5	545.3	0.280	0.13	25.1	532.7	0.223	0.26	5.8	38.3	0.112	0.13	5.9	33.1
	10%	0.425	0.26	35.7	527.7	0.277	0.14	26.3	511.3	0.222	0.27	5.9	36.5	0.115	0.13	6.2	31.9
	5%	0.133	0.27	34.6	414.2	0.103	0.15	24.8	408.1	0.144	0.31	5.9	38.7	0.078	0.15	5.4	30.9
COC	20%	0.481	0.27	34.2	582.6	0.287	0.19	33.2	543.4	0.199	0.29	5.9	31.5	0.161	0.19	5.8	30.9
	10%	0.494	0.29	35.6	446.7	0.288	0.22	33.8	498.8	0.202	0.30	5.9	31.2	0.158	0.19	5.9	28.9
	5%	0.502	0.33	38.1	388.7	0.295	0.21	34.1	397.6	0.193	0.29	6.1	30.6	0.156	0.18	6.0	28.8

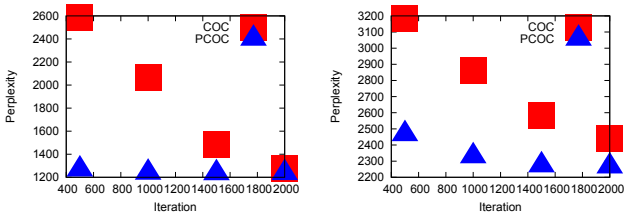

Figure 4: Convergence of test-set perplexity for (left)Netflix, (right)Twitter

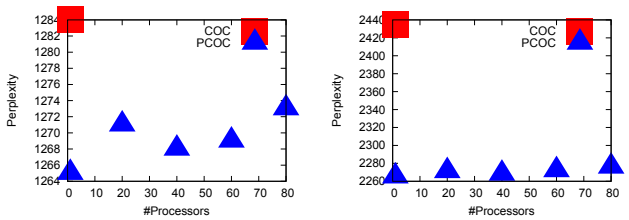

Figure 5: Test-set perplexity on (right)Netflix, (left)Twitter

ing part was used for estimating the parameters by Gibbs sampling. In all experiments, we set the number of mappers/reducers to 2/1, and the memory limit for every mapper and reducer instance to 2.0 GB on each processor. For running COC, a sample was taken at 2000 iterations for the Gibbs sampler, which is well after the typically burn-in period of the initial 200 iteration chains, and perplexities were computed for all algorithms using L=10 samples from the ten posterior independent chains using:

$$PPX = \exp\left(-\frac{1}{N_{W^{test}}}\sum_{j\in D_{\text{test}}}^{|D_{\text{test}}|}\sum_{w\in d_j}^{|N_j|}\log\frac{\sum_{r=1}^{L} p(w_{ji}^{l,test}|w_j^{l,train})}{L}\right), \quad (8)$$

where $N_{W^{test}}$ is the number of test movies (Netflix), hashtags (Twitter), and L is the number of samples (from L different chains). Note that we used consumer u instead of word w in Netflix data. This perplexity computation follows the standard practice of averaging over multiple chains, and its results are averaged over samples when making predictions as shown in previous works.

We performed experiments using various numbers of topics, P, to see whether our distributed algorithm, PCOC, with equal numbers of processors, P, matched the performance of COC on a single processor (P=1) with only a single topic.

As shown in Figure 4, the convergence rate for PCOC matched that of COC. One iteration of PCOC on multiple processors takes only a small fraction of the time needed for one iteration of COC. These results show that the perplexity of PCOC nearly matches that of COC on the same number of topics regardless of processor number, and this tendency is observed for various topics. As shown in Figure 5, these results show that PCOC offers the same performance as COC on a single processor, and so is truly scalable in terms of both speed and data size. Consequently, PCOC can process very large data sets rapidly while accurately detecting both communities and connoisseurs.

7. DISCUSSION

As AE measures how much time has passed from each item's release day to its day of purchase by each user [16], and connoisseurs help in decreasing AE shown in Table 3, our approach can suggest more novel items than is possible with other approaches. Because AD measures the difference between the observed time when each consumer purchased

a given item in the test data without recommendations and the time when this item is suggested to him/her, our real-time recommendation can achieve the longest AD and as such bias (move) the adoption curve of the normal diffusion leftward in Figure 1.

Given the goal of real-time recommendation, this paper focused on matching appropriate consumers to a target item instantly rather than processing streaming data in real-time. Indeed, our infrastructure took less than 2 hours to process all these data sets. Since this time is shorter than the time for a change in communities and their corresponding connoisseurs to affect current recommendations, our approach meets the requirement of real-time recommendation.

8. CONCLUSIONS

This paper proposed a novel real-time recommendation approach based on consumers who are reliable indicators of emerging trends. Rather than making predictions from the data of all consumers, we identify advance adopters as connoisseurs whose behavior logs are used for real-time recommendations. Our model classifies consumers into communities by considering how item popularity over time, and weights consumers in terms of their effectiveness as connoisseurs. In future work, we will incorporate available social media networks into the model, and investigate how to combine COC with them.

9. ACKNOWLEDGMENTS

This work is supported by the Grant-in-Aid for Scientific Research (KAKENHI Grant Number 26280090) from the Japan Society for the Promotion of Science.

10. REFERENCES

[1] D. Aldous. *Exchangeability and related topics*, volume 1117 of *Lecture Notes in Math*. Springer, Berlin, 1985.

[2] J. K. B. Sarwa, G. Karypis and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, pages 285–295, 2001.

[3] S. Bhagat, A. Goyal, and L. V. Lakshmanan. Maximizing product adoption in social networks. In *WSDM*, pages 603–612, 2012.

[4] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *JMLR*, 3:993–1022, 2003.

[5] O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.

[6] V. Chaoji, S. Ranu, R. Rastogi, and R. Bhatt. Recommendations to boost content spread in social networks. In *WWW*, pages 529–538, 2012.

[7] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In *KDD*, pages 199–208, 2009.

[8] D. Cosley, S. Lawrence, and D. M. Pennock. Referee: An open framework for practical testing of recommender systems using researchindex. In *VLDB*, pages 35–46, 2002.

[9] E. Diaz-Aviles, L. Drumond, L. Schmidt-Thieme, and W. Nejdl. Real-time top-n recommendation in social streams. In *RecSys*, pages 59–66, 2012.

[10] T. S. Ferguson. A bayesian analysis of some nonparametric problems. *The Annals of Statistics*, 1(2):209–230, Mar 1973.

[11] D. Fleder and K. Hosanagar. Blockbuster culture's next rise or fall: The impact of recommender systems on sales diversity. In *SSRN eLibrary*, pages 697–712, 2007.

[12] A. Goyal, F. Bonchi, and L. V. Lakshmanan. Learning influence probabilities in social networks. In *WSDM*, pages 241–250, 2010.

[13] Ò. Celma and P. Lamere. A new approach to evaluating novel recommendations. In *ACM Recsys*, pages 179–186, 2008.

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

[15] J. Huang, T. Lin, and R. W. White. No search result left behind: branching behavior with browser tabs. In *WSDM*, pages 203–212, 2012.

[16] N. Kawamae. Serendipitous recommendations via innovators. In *SIGIR*, pages 218–225, 2010.

[17] N. Kawamae. Trend analysis model: Trend consists of temporal words, topics, and timestamps. In *WSDM*, pages 317–326, 2011.

[18] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*, pages 426–434, 2008.

[19] D. Lemire and A. Maclachlan. Slope one predictors for online rating-based collaborative filtering. In *Proceedings of SIAM Data Mining (SDM'05)*, 2005.

[20] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[21] N. N. Liu, M. Zhao, E. Xiang, and Q. Yang. Online evolutionary collaborative filtering. In *RecSys*, pages 95–102, 2010.

[22] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King. Recommender systems with social regularization. In *WSDM*, pages 287–296, 2011.

[23] D. Mimno and A. McCallum. Organizing the oca: learning faceted subjects from a library of digital books. In *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, pages 376–385, 2007.

[24] R. M. Neal. Markov chain sampling methods for dirichlet process mixture models. *JCGS*, 9(2):249–265, 2000.

[25] D. Newman, A. Asuncion, P. Smyth, and M. Welling. Distributed algorithms for topic models. *JMLR*, 10:1801–1828, dec 2009.

[26] S. Rendle and L. Schmidt-Thieme. Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *Recsys*, pages 251–258, 2008.

[27] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. An open architecture for collaborative filtering of netnews. In *CSCW*, pages 175–186. ACM Press, 1994.

[28] E. M. Rogers. *Diffusion of Innovations*. The Free Press, New York, 1995.

[29] J. Sethuraman. A constructive definition of dirichlet priors. *Statistica Sinica*, 4:639–650, 1994.

[30] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei. Hierarchical dirichlet processes. *JASA*, 101(476):1566–1581, 2006.

[31] J. Yan, N. Liu, G. Wang, W. Zhang, Y. Jiang, and Z. Chen. How much can behavioral targeting help online advertising? In *WWW*, pages 261–270, 2009.

[32] J. Yang and J. Leskovec. Patterns of temporal variation in online media. In *WSDM*, pages 177–186, 2011.

[33] Q. Yuan, G. Cong, Z. Ma, A. Sun, and N. M.-T. Thalmann. Time-aware point-of-interest recommendation. In *SIGIR*, pages 363–372, 2013.

[34] X. Zhao, W. Zhang, and J. Wang. Interactive collaborative filtering. In *CIKM*, pages 1411–1420, 2013.

[35] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW*, pages 22–32, 2005.