# Chapter 8
# Combinational Collaborative Filtering, Considering Personalization

**Abstract** For the purpose of multimodal fusion, collaborative filtering can be regarded as a process of finding relevant information or patterns using techniques involving collaboration among multiple views or data sources. In this chapter,[†] we present a collaborative filtering method, *combinational collaborative filtering* (CCF), to perform recommendations by considering multiple types of co-occurrences from different information sources. CCF differs from the approaches presented in Chaps. 6 and 7 by constructing a latent layer in between the recommended objects and multimodal descriptions of these objects. We use community recommendation throughout this chapter as an example to illustrate critical design points. We first depict a community by two modalities: a collection of documents and a collection of users, respectively. CCF fuses these two modalities through a latent layer. We show how the latent layer is constructed, how multiple modalities are fused, and how the learning algorithm can be both effective and efficient in handling massive amount of data. CCF can be used to perform virtually any multimedia-data recommendation tasks such as recommending labels to images (annotation), recommending images to images (clustering), and images to users (personalized search).

**Keywords** Collaborative filtering · Multimodal fusion · Personalization · Recommendation systems · Semantic gap · Social media

## 8.1 Introduction

Collaborative filtering is a method of filtering for information or patterns using techniques involving collaboration among multiple agents, viewpoints, data sources, etc.

[†] © ACM, 2008. This chapter is a minor revision of the author's work with Wen-Yen Chen and Dong Zhang [1] published in KDD'08. Permission to publish this chapter is granted under copyright license #2587660697730.

E. Y. Chang, *Foundations of Large-Scale Multimedia Information Management and Retrieval*, DOI: 10.1007/978-3-642-20429-6_8,

(defined in Wikipedia). Collaborative filtering can be regarded as a push model of search. A search engine provides recommendations to the users proactively based on information collected from the user and her social networks. In this chapter, we tackle the problem of *community recommendation* for social networking sites. (One can substitute community with any data object such as image, video, or music, and the techniques remain the same.) What differentiates our work from prior work is that we propose a fusion method, which combines information from multiple sources. We name our method CCF for *combinational collaborative filtering*. CCF views a community from two simultaneous perspectives: *a bag of users* and *a bag of words*. A community is viewed as a bag of participating users; and at the same time, it is viewed as a bag of words describing that community. Traditionally, these two views are independently processed. Fusing these two views provides two benefits. First, by combining *bags of words* with *bags of users*, CCF can perform *personalized* community recommendations, which the *bags of words* alone model cannot. Second, augmenting *bags of users* with *bags of words*, CCF improves data density and hence can achieve better personalized recommendations than the *bags of users* alone model.

A practical recommendation system must be able to handle large-scale data sets and hence demands scalability. We devise two strategies to speed up training of CCF. First, we employ a hybrid training strategy, which combines Gibbs sampling with the Expectation–Maximization (EM) algorithm. Our empirical study shows that Gibbs sampling provides better initialization for EM, and thus can help EM to converge to a better solution at a faster pace. Our second speedup strategy is to parallelize CCF to take advantage of the distributed computing infrastructure of modern data centers.

Though in this chapter we use community recommendation as our target application, one can simply replace communities with multimedia data e.g., images. An image can be depicted as a bag of pixels, a bag of contextual information, or a bag of users who have assessed the image. CCF can then fuse these modalities to perform recommendation tasks such as recommending labels to images (annotation), recommending images to images (clustering), and images to users (personalized search). The techniques presented in Chap. 6 are suitable for fusing metadata at the lowest, syntactic layer such as color, shape, and texture descriptions. Chapter 7 presents a model to fuse content with context. This chapter considers fusion with semantics, which is a layer above the syntactic ones.

The remainder of this chapter is organized as follows. In Sect. 8.2, we discuss the related work on probabilistic latent aspect models. In Sect. 8.3, we present CCF, including its model structure and semantics, hybrid training strategy, and parallelization scheme. In Sect. 8.4, we present our experimental results on both synthetic and Orkut data sets. We provide concluding remarks and discuss future work in Sect. 8.5.

## 8.2 Related Reading

Several algorithms have been proposed to deal with either *bags of words* or *bags of users*. Specifically, Probabilistic Latent Semantic Analysis (PLSA) [2] and Latent

Dirichlet Allocation (LDA) [3] model document-word co-occurrence, which is similar to the *bags of words* community view. Probabilistic Hypertext Induced Topic Selection (PHITS) [4], a variant of PLSA, models document-citation co-occurrence, which is similar to the *bags of users* community view. However, a system that considers just bags of users cannot take advantage of content similarity between communities. A system that considers just bags of words cannot provide personalized recommendations: all users who joined the same community would receive the same set of recommendations. We propose CCF to model multiple types of data co-occurrence simultaneously. CCF's main novelty is in fusing information from multiple sources to alleviate the information sparsity problem of a single source.

Several other algorithms have been proposed to model publication and email data.[1] For instance, the *author-topic* (AT) model [5] employs two factors in characterizing a document: the document's authors and topics. Modeling both factors as variables within a Bayesian network allows the AT model to group the words used in a document corpus into semantic topics, and to determine an author's topic associations. For emails, the *author-recipient-topic* (ART) model [6] considers email recipient as an additional factor. This model can discover relevant topics from the sender–recipient structure in emails, and enjoys an improved ability to measure role-similarity between users. Although these models fit publication and email data well, they cannot be used to formulate personalized community recommendations, whereas CCF can.

## 8.3 Combinational Collaborative Filtering

We start by introducing the baseline models. We then show how our CCF model combines baseline models. Suppose we are given a collection of co-occurrence data consisting of communities $C = \{c_1, c_2, \ldots, c_N\}$, community descriptions from vocabulary $D = \{d_1, d_2, \ldots, d_V\}$, and users $U = \{u_1, u_2, \ldots, u_M\}$. If community $c$ is joined by user $u$, we set $n(c, u) = 1$; otherwise, $n(c, u) = 0$. Similarly, we set $n(c, d) = R$ if community $c$ contains word $d$ for $R$ times; otherwise, $n(c, d) = 0$. The following models are latent aspect models, which associate a latent class variable $z \in Z = \{z_1, z_2, \ldots, z_K\}$.

Before modeling CCF, we first model community–user co-occurrences (C–U), shown in Fig. 8.1a; and community–description co-occurrences (C–D), shown in Fig. 8.1b. Our CCF model, shown in Fig. 8.1c, builds on C–U and C–D models. The shaded and unshaded variables in Fig. 8.1 indicate latent and observed variables, respectively. An arrow indicates a conditional dependency between variables.

### *8.3.1 C–U and C–D Baseline Models*

The C–U model can be derived from PLSA for community–user co-occurrence analysis. The co-occurrence data consists of a set of community–user pairs $(c, u)$, which

[1] We discuss only related model-based work since the model-based approach has been proven to be superior to the memory-based approach.

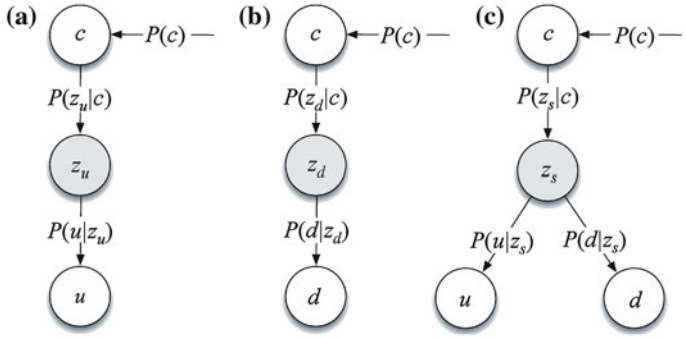


**Fig. 8.1** **a** Graphical representation of the Community–User (C–U) model. **b** Graphical representation of the Community–Description (C–D) model. **c** Graphical representation of Combinational Collaborative Filtering (CCF) that combines both bag of users and bag of words information

are assumed to be generated independently. The key idea is to introduce a latent class variable $z$ to every community–user pair, so that community $c$ and user $u$ are rendered conditionally independent. The resulting model is a mixture model that can be written as follows:

$$P(c,u) = \sum_z P(c,u,z) = P(c) \sum_z P(u|z)P(z|c), \tag{8.1}$$

where $z$ represents the topic for a community. For each community, a set of users is observed. To generate each user, a community $c$ is chosen uniformly from the community set, then a topic $z$ is selected from a distribution $P(z|c)$ that is specific to the community, and finally a user $u$ is generated by sampling from a topic-specific distribution $P(u|z)$.

The second model is for community–description co-occurrence analysis. It has a similar structure to the C–U model with the joint probability written as:

$$P(c,d) = \sum_z P(c,d,z) = P(c) \sum_z P(d|z)P(z|c), \tag{8.2}$$

where $z$ represents the topic for a community. Each community's interests are modeled with a mixture of topics. To generate each description word, a community $c$ is chosen uniformly from the community set, then a topic $z$ is selected from a distribution $P(z|c)$ that is specific to the community, and finally a word $d$ is generated by sampling from a topic-specific distribution $P(d|z)$. (One can model C–U and C–D using LDA. Please see Chap. 12 for indepth discussion on LDA.)

### *8.3.2 CCF Model*

In the C–U model, we consider only *links*, i.e., the observed data can be thought of as a very sparse binary $M \times N$ matrix $W$, where $W_{i,j} = 1$ indicates that user

$i$ joins (or linked to) community $j$, and the entry is unknown elsewhere. Thus, the C–U model captures the linkage information between communities and users, but not the community content. The C–D model learns the topic distribution for a given community, as well as topic-specific word distributions. This model can be used to estimate how similar two communities are in terms of topic distributions. Next, we introduce our CCF model, which combines both the C–U and C–D.

For the CCF model (Fig. 8.1c), the joint probability distribution over community, user, and description can be written as:

$$\begin{aligned} P(c,u,d) &= \sum_z P(c,u,d,z) \\ &= P(c)\sum_z P(u|z)P(d|z)P(z|c). \end{aligned} \tag{8.3}$$

The CCF model represents a series of probabilistic generative processes. Each community has a multinomial distribution over topics, and each topic has a multinomial distribution over users and descriptions, respectively.

### *8.3.3 Gibbs and EM Hybrid Training*

Given the model structure, the next step is to learn model parameters. There are some standard learning algorithms, such as Gibbs sampling [7], Expectation–Maximization (EM) [8], and Gradient descent. For CCF, we propose a hybrid training strategy: we first run Gibbs sampling for a few iterations, then switch to EM. The model trained by Gibbs sampling provides the initialization values for EM. This hybrid strategy serves two purposes. First, EM suffers from a drawback in that it is very sensitive to initialization. A better initialization tends to allow EM to find a "better" optimum. Second, Gibbs sampling is too slow to be effective for large-scale data sets in high-dimensional problems [9]. A hybrid method can enjoy the advantages of Gibbs and EM.

#### 8.3.3.1 Gibbs Sampling

Gibbs sampling is a simple and widely applicable Markov chain Monte Carlo algorithm, which provides a simple method for obtaining parameter estimates and allows for combination of estimates from several local maxima of the posterior distribution. Instead of estimating the model parameters directly, we evaluate the posterior distribution on $z$ and then use the results to infer $P(u|z)$, $P(d|z)$ and $P(z|c)$.

For each user–word pair, the topic assignment is sampled from:

$$\begin{aligned} &P(z_{i,j} = k|u_i = m, d_j = n, \mathbf{z}_{-i,-j}, U_{-i}, D_{-j}) \\ &\quad \propto \frac{C^{UZ}_{mk} + 1}{\sum_{m'} C^{UZ}_{m'k} + M} \frac{C^{DZ}_{nk} + 1}{\sum_{n'} C^{DZ}_{n'k} + V} \frac{C^{CZ}_{ck} + 1}{\sum_{k'} C^{CZ}_{ck'} + K}, \end{aligned} \tag{8.4}$$

where $z_{i,j} = k$ represents the assignment of the $i$th user and $j$th description word in a community to topic $k$. $u_i = m$ represents the observation that the $i$th user is the $m$th user in the user corpus, and $d_j = n$ represents the observation that the $j$th word is the $n$th word in the word corpus. $\mathbf{z}_{-i,-j}$ represents all topic assignments not including the $i$th user and the $j$th word. Furthermore, $C_{mk}^{UZ}$ is the number of times user $m$ is assigned to topic $k$, not including the current instance; $C_{nk}^{DZ}$ is the number of times word $n$ is assigned to topic $k$, not including the current instance; $C_{ck}^{CZ}$ is the number of times topic $k$ has occurred in community $c$, not including the current instance.

We analyze the computational complexity of Gibbs sampling in CCF. In Gibbs sampling, one needs to compute the posterior probability

$$P(z_{i,j} = k | u_i = m, d_j = n, \mathbf{z}_{-i,-j}, U_{-i}, D_{-j})$$

for user–word pairs $(M \times L)$ within $N$ communities, where $L$ is the number of words in community description (note $L \geq V$). Each $P(z_{i,j} = k | u_i = m, d_j = n, \mathbf{z}_{-i,-j}, U_{-i}, D_{-j})$ consists of $K$ topics, and requires a constant number of arithmetic operations, resulting in $O(K \cdot N \cdot M \cdot L)$ for a single Gibbs sampling. During parameter estimation, the algorithm needs to keep track of a topic-user $(K \times M)$ count matrix, a topic-word $(K \times V)$ count matrix, and a community-topic $(N \times K)$ count matrix. From these count matrices, we can estimate the topic-user distributions $P(u_m|z_k)$, topic-word distributions $P(d_n|z_k)$ and community-topic distributions $P(z_k|c_c)$ by:

$$\begin{aligned} P(u_m|z_k) &= \frac{C_{mk}^{UZ} + 1}{\sum_{m'} C_{m'k}^{UZ} + M}, \\ P(d_n|z_k) &= \frac{C_{nk}^{DZ} + 1}{\sum_{n'} C_{n'k}^{DZ} + V}, \\ P(z_k|c_c) &= \frac{C_{ck}^{CZ} + 1}{\sum_{k'} C_{ck'}^{CZ} + K}, \end{aligned} \tag{8.5}$$

where $P(u_m|z_k)$ is the probability of containing user $m$ in topic $k$, $P(d_n|z_k)$ is the probability of using word $n$ in topic $k$, and $P(z_k|c_c)$ is the probability of topic $k$ occurring in community $c$. The estimation of parameters by Gibbs sampling replaces the random seeding in EM's initialization step.

#### 8.3.3.2 Expectation–Maximization Algorithm

The CCF model is parameterized by $P(z|c)$, $P(u|z)$, and $P(d|z)$, which are estimated using the EM algorithm to fit the training corpus with community, user, and description by maximizing the log-likelihood function:

$$L = \sum_{c,u,d} n(c, u, d) \log P(c, u, d), \tag{8.6}$$

$$n(c,u,d) = n(c,u)n(c,d) = \begin{cases} R & \text{if community } c \text{ has user } u \\ & \text{and contains word } d \text{ for } R \textit{ times}; \\ 0 & \text{otherwise.} \end{cases} \tag{8.7}$$

Starting with the initial parameter values from Gibbs sampling, the EM procedure iterates between Expectation (E) step and Maximization (M) step:

- *E-step*: where the probability that a community $c$ has user $u$ and contains word $d$ explained by the latent variable $z$ is estimated as:

$$P(z|c,u,d) = \frac{P(u|z)P(d|z)P(z|c)}{\sum_{z'} P(u|z')P(d|z')P(z'|c)}. \tag{8.8}$$

- *M-step*: where the parameters $P(u|z)$, $P(d|z)$, and $P(z|c)$ are re-estimated to maximize $L$ in (8.6):

$$P(u|z) = \frac{\sum_{c,d} n(c,u,d)P(z|c,u,d)}{\sum_{c,u',d} n(c,u',d)P(z|c,u',d)}, \tag{8.9}$$

$$P(d|z) = \frac{\sum_{c,u} n(c,u,d)P(z|c,u,d)}{\sum_{c,u,d'} n(c,u,d')P(z|c,u,d')}, \tag{8.10}$$

$$P(z|c) = \frac{\sum_{u,d} n(c,u,d)P(z|c,u,d)}{\sum_{u,d,z'} n(c,u,d)P(z'|c,u,d)}. \tag{8.11}$$

We analyze the computational complexity of the E-step and the M-step. In the E-step, one needs to compute the posterior probability $P(z|c,u,d)$ for $M$ users, $N$ communities, and $V$ words. Each $P(z|c,u,d)$ consists of $K$ values, and requires a constant number of arithmetic operations to be computed, resulting in $O(K \cdot N \cdot M \cdot V)$ operations for a single E-step. In the M-step, the posterior probabilities are accumulated to form the new estimates for $P(u|z)$, $P(d|z)$ and $P(z|c)$. Thus, the M-step also requires $O(K \cdot N \cdot M \cdot V)$ operations. Typical values of $K$ in our experiments range from 28 to 256. The community–user $(c,u)$ and community–description $(c,d)$ co-occurrences are highly sparse, where $n(c,u,d) = n(c,u) \times n(c,d) = 0$ for a large percentage of the triples $(c,u,d)$. Because the $P(z|c,u,d)$ term is never separated from the $n(c,u,d)$ term in the M-step, we do not need to compute $P(z|c,u,d)$ for $n(c,u,d) = 0$ in the E-step. We compute only $P(z|c,u,d)$ for $n(c,u,d) \neq 0$. This greatly reduces computational complexity.

### *8.3.4 Parallelization*

The parameter estimation using Gibbs sampling and the EM algorithm described in the previous sections can be divided into parallel subtasks. We consider Message

**Input**: $N \times M$ community-user matrix; $N \times V$ community-description matrix; $I$: number of iterations; $P$: number of machines
**Output**: $P(u|z)$, $P(d|z)$, $P(z|c)$
**Variables**:
$x_{ic}$: the $i^{th}$ row of community-user matrix with community id $c$
$y_{ic}$: the $i^{th}$ row of community-word matrix with community id $c$

1: **for** $i = 0$ to $N - 1$ **do**
2: Load $x_{ic}$ into machine $c\%P$.
3: Load $y_{ic}$ into machine $c\%P$.
4: **end for**
5: Gibbs sampling initialization.
6: **for** $iter = 0$ to $I - 1$ **do**
7: **for** each <user, word> pair **do**
8: Each machine $i$ performs Gibbs sampling as in Eq. (8.4) and updates local counts $C^{UZ}_{m_ik}$, $C^{DZ}_{n_ik}$ and $C^{CZ}_{c_ik}$.
9: **end for**
10: Each machine *reduces* the local difference to a specified root, and root *broadcasts* the global difference to others to update global counts:
11: $C^{UZ}_{mk} = C^{UZ}_{mk} + \sum_i (C^{UZ}_{m_ik} - C^{UZ}_{mk})$
12: $C^{DZ}_{nk} = C^{DZ}_{nk} + \sum_i (C^{DZ}_{n_ik} - C^{DZ}_{nk})$
13: **end for**

**Fig. 8.2** Parallel Gibbs Sampling of CCF

Passing Interface (MPI) for implementation as it is more suitable for parallelizing iterative algorithms than MapReduce. Since standard MPI implementations (MPICH2) cannot be directly ported to our system, we implemented our own system by modifying MPICH2 [10].

#### 8.3.4.1 Parallel Gibbs Sampling

We distribute the computation among machines based on community IDs. Thus, each machine $i$ only deals with a specified subset of communities $c_i$, and is aware of all users $u$ and all descriptions $d$. We then perform Gibbs sampling simultaneously on each machine independently and update local counts. Afterward, each machine *reduces* the local difference ($C^{UZ}_{m_ik} - C^{UZ}_{mk}$, $C^{DZ}_{n_ik} - C^{DZ}_{nk}$) to a specified root, then the root *broadcasts* the global difference (sum of all local differences) to other machines to update global counts ($C^{UZ}_{mk}$ and $C^{DZ}_{nk}$) [11]. This is an *MPI_AllReduce* operation in MPI. We summarize the process in Fig. 8.2.

#### 8.3.4.2 Parallel EM Algorithm

The parallel EM algorithm can be applied in a similar fashion. We describe the procedure below and summarize the process in Fig. 8.3.

- *E-step*: each machine $i$ computes the $P(z|c_i, u, d)$ values, the posterior probability of the latent variables $z$ given communities $c_i$, users $u$ and descriptions $d$, using the

current values of the parameters $P(z|c_i)$, $P(u|z)$ and $P(d|z)$. As this posterior computation can be performed locally, we avoid the need for communications between machines in the E-step.
- *M-step*: each machine $i$ computes the local parameters $P(z|c_i)$, $P(u_i|z)$ and $P(d_i|z)$ using the previously calculated values $P(z|c_i, u, d)$. After that, each machine *reduces* the local parameters ($P(u_i|z)$, $P(d_i|z)$) to a specified root, and the root *broadcasts* the global parameters to other machines. This is done through a *MPI_AllReduce* operation in MPI.

We analyze the computational and communication complexities for both algorithms using distributed machines. Assuming that there are $P$ machines, the computational complexity of each training algorithm reduces to $O((K \cdot N \cdot M \cdot L)/P)$ (for Gibbs) and $O((K \cdot N \cdot M \cdot V)/P)$ (for EM) since $P$ machines share the computations simultaneously. For communication complexity, two variables are reduced and broadcasted among $P$ machines for next iteration training: $C_{mk}^{UZ}$, $C_{nk}^{DZ}$ in Gibbs sampling, and $P(u|z)$, $P(d|z)$ in EM. The communication cost is $O(\alpha \cdot \log P + \beta \cdot \frac{P-1}{P} K(M+V) + \gamma \cdot \frac{P-1}{P} K(M+V))$, where $\alpha$ is the startup time of a transfer, $\beta$ is the transfer time per byte, and $\gamma$ is the computation time per byte for performing the reduction operation locally on any machine.

### 8.3.5 Inference

Once we have learned the model parameters, we can infer three relationships using Bayesian rules, namely user–community relationship, community similarity, and user similarity. We derive these three relationships as follows:

- *User–community relationship*: communities can be ranked for a given user according to $P(c_j|u_i)$, i.e. which communities should be recommended for a given user? Communities with top ranks and communities that the user has not yet joined are good candidates for recommendations. $P(c_j|u_i)$ can be calculated using

$$\begin{aligned} P(c_j|u_i) &= \frac{\sum_z P(c_j, u_i, z)}{P(u_i)} \\ &= \frac{P(c_j)\sum_z P(u_i|z)P(z|c_j)}{P(u_i)} \\ &\propto \sum_z P(u_i|z)P(z|c_j), \end{aligned} \tag{8.12}$$

  where we assume that $P(c_j)$ is a uniform prior for simplicity.
- *Community similarity*: communities can also be ranked for a given community according to $P(c_j|c_i)$. We calculate $P(c_j|c_i)$ using

$$
\begin{aligned}
P(c_j|c_i) &= \frac{\sum_z P(c_j, c_i, z)}{P(c_i)} \\
&= \frac{\sum_z P(c_j|z)P(c_i|z)P(z)}{P(c_i)} \\
&= P(c_j)\sum_z \frac{P(z|c_j)P(z|c_i)}{P(z)} \\
&\propto \sum_z \frac{P(z|c_j)P(z|c_i)}{P(z)},
\end{aligned} \tag{8.13}
$$

where we assume that $P(c_j)$ is a uniform prior for simplicity.

- *User similarity*: users can be ranked for a given user according to $P(u_j|u_i)$, i.e. which users should be recommended for a given user? Similarly, we can calculate $P(u_j|u_i)$ using

$$
\begin{aligned}
P(u_j|u_i) &= \frac{\sum_z P(u_j, u_i, z)}{P(u_i)} \\
&= \frac{\sum_z P(u_j|z)P(u_i|z)P(z)}{P(u_i)} \\
&= P(u_j)\sum_z \frac{P(z|u_j)P(z|u_i)}{P(z)} \\
&\propto \sum_z \frac{P(z|u_j)P(z|u_i)}{P(z)},
\end{aligned} \tag{8.14}
$$

where we assume that $P(u_j)$ is a uniform prior for simplicity.

## 8.4 Experiments

We divided our experiments into two parts. The first part was conducted on a relatively small synthetic dataset with ground truth to evaluate the Gibbs and EM hybrid training strategy. The second part was conducted on a large, real-world dataset to test out CCF's performance and scalability. Our experiments were run on up to 200 machines at our distributed data centers. While not all machines are identically configured, each machine is configured with a CPU faster than 2 GHz and memory larger than 4 GB (a typical Google configuration in 2007).

### *8.4.1 Gibbs + EM Versus EM*

To precisely account for the benefit of Gibbs and EM over the EM-only training strategy, we used a synthetic dataset where we know the ground truth. The synthetic

**Input**: $N \times M$ community-user matrix; $N \times V$ community-description matrix; $I$: number of iterations; $P$: number of machines; $P(u|z)$, $P(d|z)$, $P(z|c)$ of Gibbs sampling
**Output**: $P(u|z)$, $P(d|z)$, $P(z|c)$
**Variables**:
$x_{ic}$: the $i^{th}$ row of community-user matrix with community id $c$
$y_{ic}$: the $i^{th}$ row of community-word matrix with community id $c$

Load $P(u|z)$, $P(d|z)$, $P(z|c)$ of Gibbs sampling.
**for** $i = 0$ to $N - 1$ **do**
    Load $x_{ic}$ into machine $c\%P$.
    Load $y_{ic}$ into machine $c\%P$.
**end for**
**for** $iter = 0$ to $I - 1$ **do**
    **for** $k = 0$ to $K - 1$ **do**
        **E-step**:
        Each machine $i$ computes $P(z_k|u, c_i, d)$
        **M-step**:
        Each machine $i$ computes parameters $P(z_k|c_i)$, $P(u_i|z_k)$, $P(d_i|z_k)$, and *reduces* local parameters to a specific root, then root *broadcasts* the global parameters to others:
            $P(u|z_k) = \sum_i P(u_i|z_k)$
            $P(d|z_k) = \sum_i P(d_i|z_k)$
    **end for**
**end for**

**Fig. 8.3** Parallel EM algorithm of CCF

dataset consists of 5,000 documents with 10 topics, a vocabulary size 10,000, and a total of 50,000,000 word tokens. The true topic distribution over each document was pre-defined manually as the ground truth. We conducted the comparisons using the following two training strategies: (1) EM-only strategy (without Gibbs sampling as initialization) where the number of EM iterations is 10 through 100 respectively, (2) Gibbs and EM strategy where the number of Gibbs sampling iterations is 5, 10, 15 and 20, and the number of EM iterations is 10 through 70, respectively. We used Kullback–Leibler divergence (K–L divergence) to evaluate model performance since the K–L divergence is a good measure for the difference between the true topic distribution ($P$) and the estimated topic distribution ($Q$) defined as follows:

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}. \tag{8.15}$$

The smaller the K–L divergence is, the better the estimated topic distribution approximates the true topic distribution.

Figure 8.4 compares the average K–L divergences over 10 runs. It shows that more rounds of Gibbs sampling can help EM reach a solution that enjoys a smaller K–L divergence. Since each iteration of Gibbs sampling takes longer than EM, we must also consider *time*. Figure 8.5 shows the values of K–L divergence as a function of the training time, where EM-only strategy began with 20 EM iterations. We can make two observations. First, given a large amount of time, both EM and the hybrid scheme can reach very low K–L divergence. On this dataset, when the training time

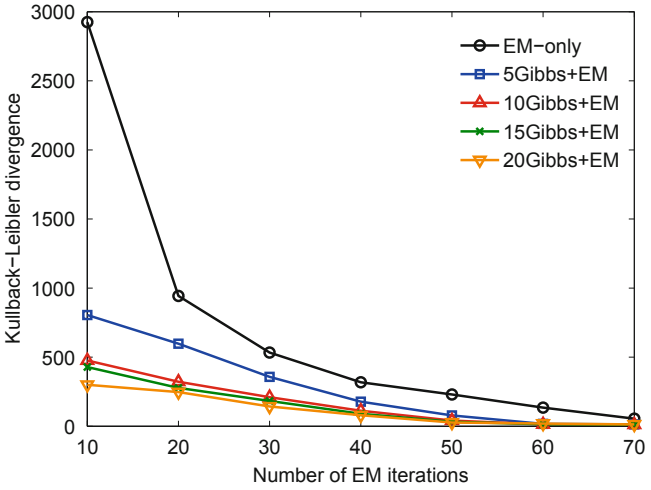


**Fig. 8.4** The Kullback-Leibler divergence as a function of the number of iterations

exceeded 350 s, the value of K–L divergence approached zero for all strategies. Nevertheless, on a large dataset, we cannot afford a long training time, and the Gibbs and EM hybrid strategy provides a earlier point to stop training, and hence reduces the overall training time.

The second observation is on the number of Gibbs iterations. As shown in both figures, running more iterations of Gibbs before handing over to EM takes longer to yield a better initial point for EM. In other words, spending more time in the Gibbs stage can save time in the EM stage. Figure 8.5 shows that the best performance was produced by 10 iterations of Gibbs sampling before switching to EM. Finding the "optimal" switching point is virtually impossible in theory. However, the figure shows that different Gibbs iterations can all outperform the EM-only strategy to obtain a better solution early, and a reasonable number of Gibbs iterations can be obtained through an empirical process like our experiment. Moreover, the figure shows that a range of number of iterations can achieve similar K–L divergence (e.g., at time 250). This indicates that though an empirical process may not be able to pin down the "optimal" number of iterations (because of e.g., new training data arrival), the hybrid scheme can work well on a range of Gibbs-sampling iterations.

### *8.4.2 The Orkut Dataset*

Orkut is an extremely active community site with more than two billion page views a day world-wide. The dataset we used was collected on July 26, 2007, which contains two types of data for each community: community membership information and community description information. We restrict our analysis to English

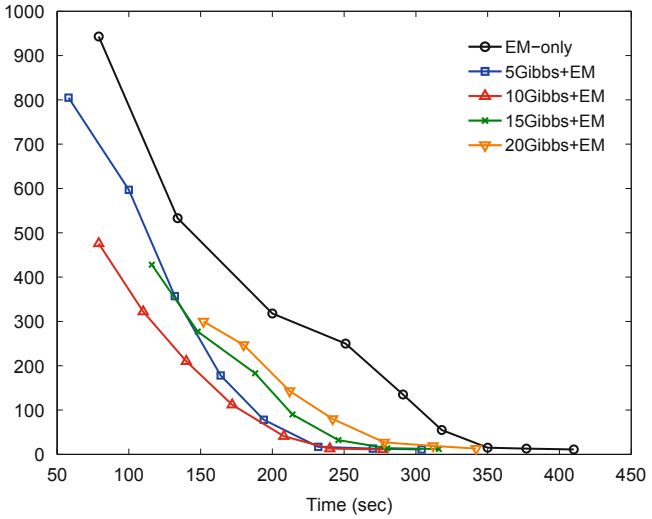


**Fig. 8.5** The Kullback-Leibler divergence as a function of the training time

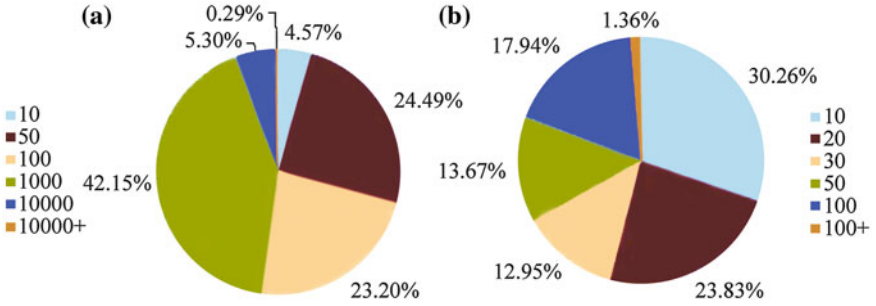


**Fig. 8.6** **a** Distribution of the number of users per community, and **b** distribution of the number of description words per community (see color insert)

communities only. We collected 312,385 users and 109,987 communities.[2] The number of entries in the community–user matrix, or the number of community–user pairs, is 35,932,001. As the density is around 0.001045, this matrix is extremely sparse. Figure 8.6a shows a distribution of the number of users per community. About 52% of all communities have less than 100 users, whereas 42% of all communities have more than 100 but less than 1,000 users.

For the community description data, after applying downcasing, stopword filtering, and word stemming, we obtained a vocabulary of 191,034 unique English words. The distribution of the number of description words per community is displayed in Fig. 8.6b. On average, there are 27.64 words in each community description after processing.

[2] All user data were anonymized, and user privacy is safeguarded, as performed in [12].

In order to establish statistical significance of the findings, we repeated all experiments 10 times with different random seeds and parameters, such as the number of latent aspects (ranging from 28 to 256), the number of Gibbs sampling iterations (ranging from 10 to 30) and the number of EM iterations (ranging from 100 to 500). The reported results are the average performance over all runs.

**Results**

*Community recommendation*: $P(c_j|u_i)$. We use two standard measures from information retrieval to measure the recommendation effectiveness: *precision* and *recall*, defined as follows:

$$\begin{aligned} \text{Precision} &= \frac{\left|\{\text{recommendation list}\} \bigcap \{\text{joined list}\}\right|}{|\{\text{recommendation list}\}|}, \\ \text{Recall} &= \frac{\left|\{\text{recommendation list}\} \bigcap \{\text{joined list}\}\right|}{|\{\text{joined list}\}|} \end{aligned} \tag{8.16}$$

Precision takes all recommended communities into account. It can also be evaluated at a given cut-off rank, considering only the topmost results recommended by the system. As it is possible to achieve higher recall by recommending more communities (note that a recall of 100% is trivially achieved by recommending all communities, albeit at the expense of having low precision), we limit the size of our community recommendation list to at most 200.

To evaluate the results, we randomly deleted one joined community for each user in the community–user matrix from the training data. We evaluated whether the deleted community could be recommended. This evaluation is similar to *leave-one-out*. Figure 8.7 shows the precision and recall as functions of the length (up to 200) of the recommendation list for both C–U and CCF. We can see that CCF always outperforms C–U for all lengths. Figure 8.8 presents precision and recall for the top 20 recommended communities. As both precision and recall of CCF are nearly twice higher than those of C–U, we can conclude that CCF enjoys better prediction accuracy than C–U. This is because C–U only considers community–user co-occurrence, whereas CCF considers users, communities, and descriptions. By taking other views into consideration, the information is denser for CCF to achieve higher prediction accuracy.

Figure 8.9 depicts the relationship between the precision of the recommendation for a user and the number of communities that the user has joined. The more communities a user has joined, the better both C–U and CCF can predict the user's preferences. For users who joined around 100 communities, the precision is about 15% for C–U and 27% for CCF. However, for users who joined just 20 communities, the precision is about 7% for C–U, and 10% for CCF. This is not surprising since it is very difficult for latent-class statistical models to generalize from sparse data. For large-scale recommendation systems, we are unlikely to ever have enough direct data with sufficient coverage to avoid sparsity. However, at the very least, we can

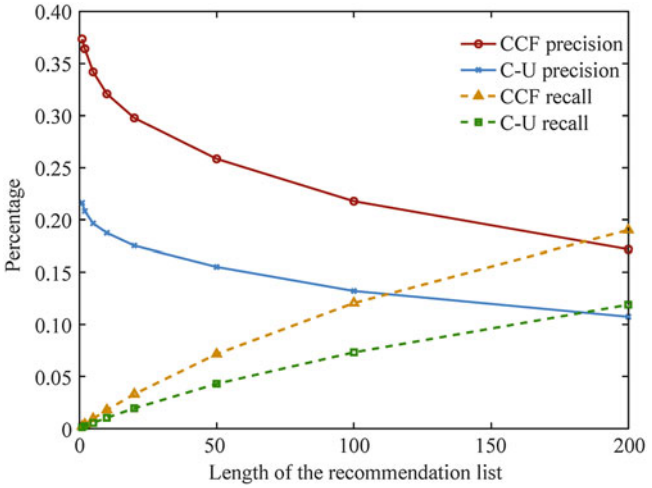


**Fig. 8.7** The precision and recall as functions of the length of the recommendation list (see color insert)

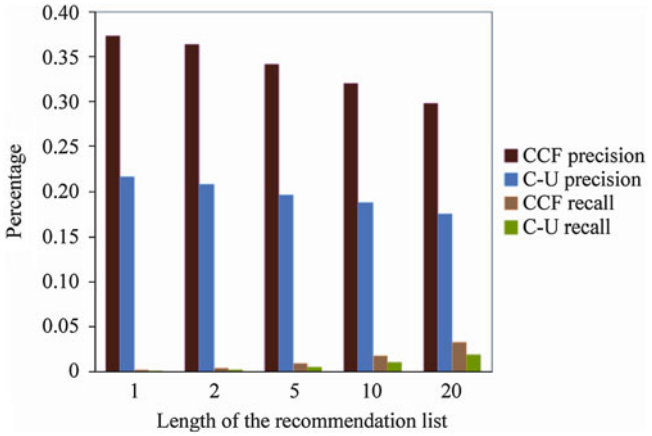


**Fig. 8.8** The precision and recall as functions of the length (up to 20) of the recommendation list (see color insert)

try to incorporate indirect data to boost our performance, just as CCF does by using bags of words information to augment bags of users information.

*Remark* Because of the nature of leave-one-out, our experimental result can only show whether a joined community could be recovered. The low precision/recall reflects this necessary, restrictive experimental setting. (This setting is necessary for objectivity purpose as we cannot obtain ground-truth of all users' future preferences.) The key observation from this study is not the absolute precision/recall values, but is the relative performance between CCF and C–U.

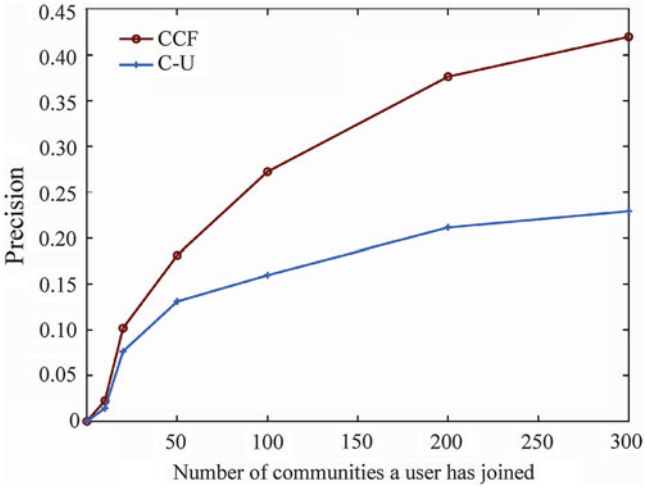


**Fig. 8.9** The precision as a function of the number of communities a user has joined. Here, the length of the recommendation list is fixed at 20

*Community similarity* $P(c_j|c_i)$. We next report the results of community similarities calculated by the three models. We used *community category* (available at Orkut websites) as the ground-truth for clustering communities. We also assigned each community an estimated label for the latent aspect with the highest probability value. We treated communities with the same estimated label as members of the same *community cluster*. We then compared the difference between community clusters and categories using the Normalized Mutual Information (NMI).

NMI between two random variables *CAT* (category label) and *CLS* (cluster label) is defined as $NMI(CAT; CLS) = \frac{I(CAT;CLS)}{\sqrt{H(CAT)H(CLS)}}$, where $I(CAT; CLS)$ is the mutual information between *CAT* and *CLS*. The entropies $H(CAT)$ and $H(CLS)$ are used for normalizing the mutual information to be in the range [0, 1]. In practice, we made use of the following formulation to estimate the *NMI* score [13]:

$$NMI = \frac{\sum_{s=1}^{K}\sum_{t=1}^{K} n_{s,t} \log\left(\frac{n \cdot n_{s,t}}{n_s \cdot n_t}\right)}{\sqrt{\left(\sum_s n_s \log\frac{n_s}{n}\right)\left(\sum_t n_t \log\frac{n_t}{n}\right)}}, \tag{8.17}$$

where $n$ is the number of communities, $n_s$ and $n_t$ denote the numbers of community in category $s$ and cluster $t$, $n_{s,t}$ denotes the number of community in category $s$ as well as in cluster $t$. The *NMI* score is 1 if the clustering results perfectly match the category labels and 0 for a random partition. Thus, the larger this score, the better the clustering results.

Table 8.1 shows that CCF slightly outperforms both C–U and C–D models, which indicates the benefit of incorporating two types of information.

*User similarity* $P(u_j|u_i)$. An interesting application is friend suggestion: finding users similar to a given user. Using 8.14, we can compute user similarity for all pairs

**Table 8.1** The comparison results of the three models using Normalized Mutual Information (NMI)

| Model | C–U | C–D | CCF |
|---|---|---|---|
| NMI | 0.4508 | 0.3127 | 0.4526 |

**Table 8.2** The top recommended users using the C–U and CCF models for the query user 79

| Model | *Rank* 1*st* | | *Rank* 2*nd* | | *Rank* 3*rd* | |
|---|---|---|---|---|---|---|
| | User ID | Communities | User ID | Communities | User ID | Communities |
| C–U | 2390 | 551 (102, 18.5%) | 8207 | 456 (100, 21.9%) | 6734 | 494 (95, 19.2%) |
| CCF | 7931 | 518 (106, 20.5%) | 10968 | 680 (102, 15.0%) | 6776 | 680 (91, 13.4%) |

The number of communities that user 79 joined is 339. (Note that the "Communities" field contains three numbers: the first number $n$ is the total number of communities a user joined; the second number $k$ is the number of overlapping communities between the recommended user and the query user, and the last number is percentage of $\frac{k}{n}$)

of users. From these values, we derive a ranking of the most similar users for a given query user. Due to privacy concerns, we were not able to obtain the friend graph of each user to evaluate accuracy. Table 8.2 shows an example of this ranking for a given user.

"Similar" users typically share a significant percentage of commonly-joined communities. For instance, the query user also joined 18.5% of the communities joined by the top user ranked by C–U, compared to 20.5% for CCF. It is encouraging to see that CCF's top ranked user has more overlap with the query user than C–U's top ranked user does. We believe that, again, incorporating the additional word co-occurrences has improved information density and hence yields higher prediction accuracy.

### *8.4.3 Runtime Speedup*

In analyzing runtime speedup for parallel training, we trained CCF with 20 latent aspects, 10 Gibbs sampling, and 20 EM iterations. As the size of a dataset is large, a single machine cannot store all the data—($P(u|z)$, $P(d|z)$, $P(z|c)$, and $P(z|c, u, d)$—in its local memory, we cannot obtain the running time of CCF on one machine. Therefore, we use the runtime of 10 machines as the baseline and assume that 10 machines can achieve 10 times speedup. This assumption is reasonable as we will see shortly that our parallelization scheme can achieve linear speedup on up to 100 machines. Table 8.3 and Fig. 8.10 report the runtime speedup of CCF using up to 200 machines. The Orkut dataset enjoys a linear speedup when the number of machines is up to 100. After that, adding more machines receives diminishing returns. This result led to our examination of overheads for CCF, presented next.

No parallel algorithm can infinitely achieve linear speedup because of the Amdahl's law. When the number of machines continues to increase, the commu-

**Table 8.3** Runtime comparisons for different number of machines

| Machines | Time (s) | Speedup |
|---|---|---|
| 10 | 9,233 | 10 |
| 20 | 4,326 | 21.3 |
| 50 | 2,280 | 40.5 |
| 100 | 1,014 | 91.1 |
| 200 | 796 | 116 |

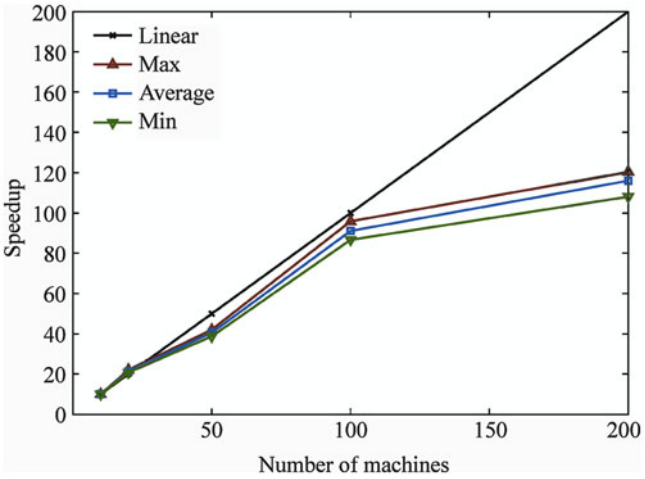


**Fig. 8.10** Speedup analysis for different number of machines

nication cost starts to dominate the total running time. The running time consists of two main parts: computation time (Comp) and communication time (Comm). Figure 8.11 shows how Comm overhead influences the speedup curves. We draw on the top the computation only line (Comp), which approaches the linear speedup line. The speedup deteriorates when communication time is accounted for (Comp + Comm). Figure 8.12 shows the percentage of Comp and Comm in the total running time. As the number of machines increases, the communication cost also increases. When the number of machines exceeds 200, the communication time becomes even larger than the computation time.

Though the Amdahl's law eventually kicks in to forbid a parallel algorithm to achieve infinite speedup, our empirical study draws two positive observations.

1. When the dataset size increases, the "saturation" point of the Amdahl's law is deferred, and hence we can add more machines to deal with larger sets of data.
2. The speedup that can be achieved by parallel CCF is very significant to enable near-real-time recommendations. As shown in the table, the parallel scheme reduces the training time from one day to less than 14 min. The parallel CCF can be run every 14 min to produce a new model to adapt to new access patterns and new users.

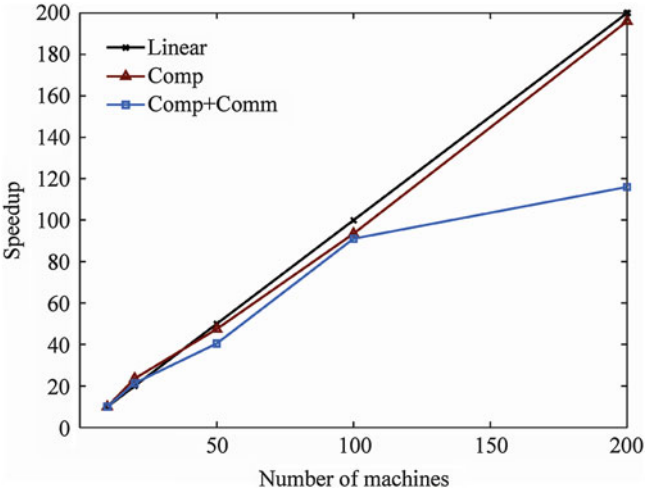


**Fig. 8.11** Speedup and overhead analysis

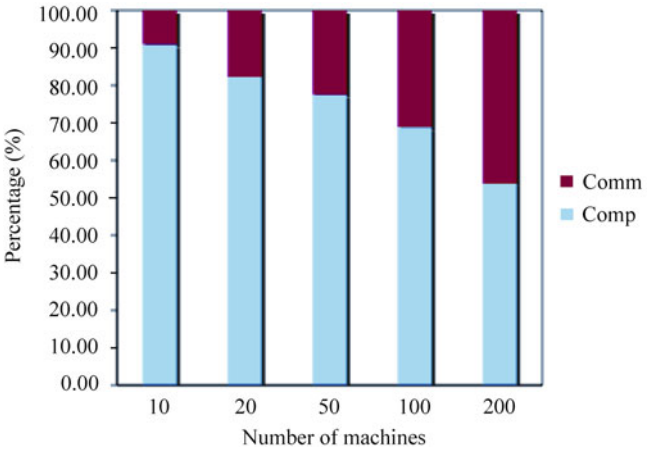


**Fig. 8.12** Runtime (computation and communication) composition analysis

## 8.5 Concluding Remarks

This chapter has presented a generative graphical model, Combinational Collaborative Filtering (CCF), for collaborative filtering based on both bags of words and bags of users information. CCF uses a hybrid training strategy that combines Gibbs sampling with the EM algorithm. The model trained by Gibbs sampling provides

better initialization values for EM than random seeding. We also presented the parallel computing required to handle large-scale data sets. Experiments on a large Orkut data set demonstrate the approaches to successfully produce better quality recommendations, and accurately cluster relevant communities/users with similar semantics.

There are several directions for future research. First, one can consider expanding CCF to incorporate more types of co-occurrence data. More types of co-occurrence data would help to overcome sparsity problem and make better recommendation. Second, in our analysis, the community–user pair value equals one, i.e. $n(u_i, c_j) = 1$ (if user $u_i$ joins community $c_j$). An interesting extension would be to give this count a different value, i.e. $n(u_i, c_j) = f$, where $f$ is the frequency of the user $u_i$ visiting the community $c_j$. Third, as we have mentioned, one can replace PLSA with LDA (see Chap. 12) or the causality strength model (Chap. 7) to conduct inference. Finally, CCF, as a general framework of combining multiple types of co-occurrence data, has many applications in information retrieval, social network mining, and other related areas.

## References

1. W. Chen, D. Zhang, E.Y. Chang, Combinational collaborative filtering for personalized community recommendation, in *Proceedings of ACM SIGKDD*, 2008, pp. 115–123
2. T. Hofmann, Probabilistic latent semantic analysis, in *Proceedings of the 15th UAI Conference*, 1999, pp. 289–296
3. D.M. Blei, A.Y. Ng, M.I. Jordan, Latent Dirichlet allocation. J. Mach. Learn. Res. **3**, 993–1022 (2003)
4. D. Cohn, H. Chang, Learning to probabilistically identify authoritative documents, in *Proceedings of the 17th ICML Conference*, 2000, pp. 167–174
5. M. Steyvers, P. Smyth, M. Rosen-Zvi, T. Griffiths, Probabilistic author-topic models for information discovery, in *Proceedings of the 10th ACM SIGKDD Conference*, 2004, pp. 306–315
6. A. McCallum, A. Corrada-Emmanuel, X. Wang, The author-recipient-topic model for topic and role discovery in social networks: experiments with enron and academic email. Technical report, Computer Science, University of Massachusetts Amherst, 2004
7. S. Geman, D. Geman, Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. IEEE Trans. Pattern Anal. Mach. Intell. **6**, 721–741 (1984)
8. A.P. Dempster, N.M. Laird, D.B. Rubin, Maximum likelihood from incomplete data via the EM algorithm. J. R. Stat. Soc. Ser. B (Methodol.) **39**(1), 1–38 (1977)
9. D.M. Blei, M.I. Jordan, Variational methods for the Dirichlet process, in *Proceedings of the 21st ICML Conference*, 2004, pp. 373–380
10. W. Gropp, E. Lusk, A. Skjellum, *Using MPI-2: Advanced Features of the Message-Passing Interface* (MIT Press, Cambridge, 1999)
11. D. Newman, A. Asuncion, P. Smyth, M. Welling, Distributed inference for latent Dirichlet allocation, in *Proceedings of NIPS*, 2007
12. E. Spertus, M. Sahami, O. Buyukkokten, Evaluating similarity measures: a large-scale study in the orkut social network, in *Proceedings of the 11th ACM SIGKDD Conference*, 2005, pp. 678–684
13. A. Strehl, J. Ghosh, Cluster ensembles—a knowledge reuse framework for combining multiple partitions. J. Mach. Learn. Res. **3**, 583–617 (2002)