Yuen *et al. Big Data Analytics* (2016) 1:14
DOI 10.1186/s41044-016-0012-2

Big Data Analytics

RESEARCH Open Access

An online-updating algorithm on probabilistic matrix factorization with active learning for task recommendation in crowdsourcing systems

Man-Ching Yuen*, Irwin King and Kwong-Sak Leung

*Correspondence:
mcyuen@cse.cuhk.edu.hk
Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, NT, Hong Kong, China

Abstract

Background: To ensure the output quality, current crowdsourcing systems highly rely on redundancy of answers provided by multiple workers with varying expertise, however massive redundancy is very expensive and time-consuming. Task recommendation can help requesters to receive good quality output quicker as well as help workers to find their right tasks faster. To reduce the cost, a number of previous works adopted active learning in crowdsourcing systems for quality assurance. Active learning is a learning approach to achieve certain accuracy with a very low cost. However, previous works do not consider the varying expertise of workers for various task categories in real crowdsourcing scenarios; and they do not consider new workers who are not willing to work on a large amount of tasks before having a list of preferred tasks recommended. In this paper, we propose ActivePMFv2, Probabilistic Matrix Factorization with Active Learning (version 2), on a task recommendation framework called TaskRec to recommend tasks to workers in crowdsourcing systems for quality assurance. By assigning the most uncertain task for new workers to work on, this paper identifies a flaw in our previous ActivePMFv1, Probabilistic Matrix Factorization with Active Learning (version 1). Therefore, ActivePMFv2 can give new workers a list of preferred tasks recommended faster than that of ActivePMFv1. Our factor analysis model considers not only worker task selection preference, but also worker performance history. It actively selects the most uncertain task for the most reliable workers to work on to retrain the classification model. Moreover, we propose a generic online-updating method for learning the model, ActivePMFv2. The larger the profile of a worker (or task) is, the less important is retraining its profile on each new work done. In case of the worker (or task) having large profile, our online-updating algorithm retrains the whole feature vector of the worker (or task) and keeps all other entries in the matrix fixed. Our online-updating algorithm runs batch update to reduce the running time of model update.

Results: Complexity analysis shows that our model is efficient and is scalable to large datasets. Based on experiments on real-world datasets, the result shows that the MAE results and RMSE results of our proposed ActivePMFv2 are improved up to 29 % and 35 % respectively comparing with ActivePMFv1, where ActivePMFv1 outperforms the PMF

(Continued on next page)

(Continued from previous page)
with other active learning approaches significantly as shown in previous work. Experiment results show that our online-updating algorithm is accurate in approximating to a full retrain of the learning model while the average runtime of model update for each work done is reduced by more than 80 % (decreases from a few minutes to several seconds).

Conclusions: To the best of our knowledge, we are the first one to use PMF, active learning and dynamic model update to recommend tasks for quality assurance in crowdsourcing systems for real scenarios.

Keywords: Crowdsourcing, Task recommendation, Matrix factorization, Probabilistic matrix factorization

Background

Crowdsourcing is an idea of outsourcing a task to a large group of networked people in the form of an open call to reduce the production cost [1, 2]. In recent years, crowdsourcing systems attract much attentions at present [3, 4]. Some examples of crowdsourcing systems are Amazon Mechanical Turk (or MTurk) [5], CrowdFlower [6], Taskcn [7] and TopCoder [8]. In a crowdsourcing system, the output quality of a completed task in a crowdsourcing system is "the extent to which the provided outcome fulfills the requirements of the requester" [9]. For quality assurance, a requester has to verify the quality of every answer submitted by workers, and it is very time-consuming. Alternatively, requesters highly rely on redundancy of answers provided by multiple workers with varying expertise, but massive redundancy is very expensive and time-consuming. "If we ask 10 workers to complete the same task, then the cost of crowdsourcing solutions tends to be comparable to the cost of in-house solutions" [10]. Therefore, it is important to investigate on how to support task requesters to verify correct answers on crowdsourcing platforms easily and effectively. On the other hand, it is not efficient that the amount of time for a worker spent on selecting a task is comparable with that spent on working on a task, but the monetary reward of a task is just a small amount. To address this problem, task recommendation can help to provide a list of preferred tasks to workers in crowdsourcing systems. However, new workers do not want to work on a large number of tasks or wait for a long time before having a list of preferred tasks recommended. Therefore, it is important to help workers to find their right tasks as quick as possible and minimize the number of task assignments to achieve a target output quality [11]. The worker performance history makes it possible to mine workers' preference on tasks and to provide an indication of worker quality on tasks. Based on worker performance history, an active learning approach on task recommendation can be used to help requesters to receive good quality output quicker with lower cost, thus achieve quality assurance in crowdsourcing systems. Moreover, by assigning the most uncertain task for new workers to work on, it not only helps new workers having a list of preferred tasks recommended faster, but also improves the output quality of crowdsourcing systems.

Task recommendation can help requesters to receive good quality output quicker as well as help workers to find their right tasks faster. Probabilistic Matrix Factorization (PMF) [12] is the state-of-the-art approach for recommendation systems. A factorization model

has to be trained and learned before the model can be applied for prediction. In real-world applications, the performance of a factorization model is highly affected by how the model is updated, and thus dynamic updating a model is very important [13]. When updating a worker's profile, the profile will not change much if the worker having large profile; while the profile will have great change if the worker having small profile.

Active learning is a learning approach to improve the prediction accuracy with a low cost. By performing active learning in a task recommendation model, it can guarantee the accuracy of recommendations with a very low cost, but it still needs to consider the minimizaton of the user waiting time. The tasks recommended to the new workers have to carefully selected, because new workers are not willing to work on a lot of tasks before having their preferred tasks recommended. Therefore, systems that provide recommendations in large user waiting times are not suitable for real-world applications [14]. Moreover, it does not make sense to retrain the model from scratch whenever a worker of large profile completes a task, because the performance improvement by retraining the model in the case is tiny but the cost of retraing model is high. Furthermore, when a large number of workers are working in the crowdsourcing system at the same period of time, the computational complexity is very high if the model is retrained after each worker completes a task. Batch update provides a method of reducing both user waiting time and computational complexity.

Our contributions are as follows:

- First, we propose a way for quality assurance by performing ActivePMFv2, Probabilistic Matrix Factorization with Active Learning (version 2), for task recommendation in crowdsourcing systems, where ActivePMF is an active learning approach on factor analysis based on probabilistic matrix factorization, such that the worker latent feature space, task latent feature space and task category latent feature space are learned. ActivePMF considers the varying expertise of workers for different tasks in real crowdsourcing scenarios. The most informative task and the most skillful worker are selected to learn the factor analysis model.
- Second, we first assign all new tasks to the most reliable workers based on the task categories in our proposed ActivePMFv2, so new workers can receive a list of preferred tasks recommended faster than that of ActivePMFv1 [15]. Our proposed ActivePMFv2 also has better output prediction quality than that of ActivePMFv1.
- Third, we propose a generic online-updating method for learning a factor analysis model, ActivePMFv2. In our proposed online-updating approach, our online-updating algorithm applies on a learned PMF model without having retrain the whole model. The proposed update methods are generic and appliable for all PMF models.
- Fourth, we demonstrate the performance of our proposed ActivePMFv2 by using the real word dataset. The experimental results show that ActivePMFv2 outperforms ActivePMFv1 by 29 % in the MAE results and 35 % in RMSE results, where ActivePMFv1 outperforms PMF with various active learning approaches significantly (PMF is the state-of-the-art approach for recommendation systems).
- Fifth, we demonstrate the performance of our online-updating algorithm by using the real world dataset. The experiment results show that the prediction of online-updating ActivePMF on TaskRec model approximates to that of a full retrain of ActivePMF on TaskRec model while the running time of online-updating

algorithm is significantly lower than that of a full retrain of the model. By using online-updating algorithm, the average runtime of model update for each work done is reduced by more than 80 % (decreases from a few minutes to several seconds).
- Finally, complexity analysis shows that our model is efficient and is scalable to large datasets.

Related work

Crowdsourcing systems

Crowdsourcing is outsourcing a task to a large group of networked people in the form of an open call to reduce the production cost. A crowdsourcing process involves operations of both requesters and workers. A requester submits a task request; a worker selects and completes a task; and the requester only pays the worker for the successful completion of the task. Task recommendation in crowdsourcing is important because of the following reasons:

- **Motivate workers of diverse background to work on crowdsourcing tasks in long run.** Currently, on crowdsourcing sites, most workers only provide moderate contributions [16] and there is a significant population of young and well-educated Indian workers [17]. It can attract more workers to contribute their efforts in long run if a worker find a suitable task on a crowdsourcing site easily.
- **Improve the quality of work.** Workers perform better if they are familiar with the tasks. Chilton et al. [18] showed that task workers only browsed the first few pages on crowdsourcing sites when searching for tasks. The task list for a worker of Amazon MTurk site is usually displayed on hundreds of pages. A worker selects a task from the list of available tasks sorted by a specified feature of tasks such as task creation date and reward amount. When the tasks posted on the first few pages are not suitable for a worker, the worker might choose a task that he does not familiar with and try to complete it to earn the rewards; otherwise, the worker does not select any task. Working with a unfamiliar task might decrease the quality of work.

Recommendation systems

Broadly speaking, recommendation systems are based on either content filtering approach or collaborative filtering approach. The content filtering approach creates a profile for each user or product, for example, a movie, to characterize its nature. The profiles of users and products allow programs to associate users with matching products. The advantage is that it can address the system's new products and users. However, the profile information might not be available or easy to collect. On the other hand, the collaborative filtering approach relies only on past user behavior. This approach analyzes relationships between users and interdependencies among products to identify new user-item associations. It is generally more accurate than content filtering approach. However, collaborative filtering cannot address the system's new products and users [19], which is the cold-start problem.

To address the cold-start problem, latent factor models are an alternative approach that can approximate the ratings by characterizing both users and items on a number of factors inferred from the ratings patterns. "Some of the most successful realizations of latent factor models are based on matrix factorization." [19] Matrix factorization has a lot of applications [20, 21]. Although matrix factorization can solve the cold-start problem, it is

not scalable. Probabilistic matrix factorization (PMF) model [22] can scale linearly with the number of observations, and performs very well on large, sparse, and imbalanced datasets.

Recently, several probabilistic matrix factorization methods [23] have been proposed for collaborative filtering approach in recommendation systems. These methods focus on using low-rank approximations to model the user-item rating matrix for making further predictions. The premise behind a low-dimensional factor model is that there is only a small number of factors influencing preferences, and that a user's preference vector is determined by how each factor applies to that user. The above approaches are used for user recommendation in social tagging systems.

Task recommendation in crowdsourcing systems

A recommendation system can improve the performance of crowdsourcing systems by providing task requesters some output quality controls based on a number of parameters, such as task requirements, task properties, worker interests, worker incentives, and costs [24]. Based on collaborative filtering, Organisciak et al. [25] proposed two approaches for capturing personal preferences in personalized item recommendation in crowdsourcing systems; and they are taste-matching and taste-grokking. Taste-matching uses workers' taste to infer the requester's taste where workers and the requester have similar tastes, while taste-grokking uses workers' explicit prediction on the requester's taste. Both taste-matching and taste-grokking have better performance than the use of generic workers. Later, Organisciak et al. [26] demonstrated the performance of personalized crowdsourcing in a complexer environment by carrying out case studies on personalized text highlighting in film reviews. The results show that both approaches have better performance than a non-personalized baseline. Besides, the taste-grokking approach performs well in simpler tasks and the taste-matching approach performs well with larger crowds and tasks with latent decision-making variables. Ambati et al. [27] proposed classification based task recommendation approach to recommend tasks to users based on implicit modeling of skills and interests. However, these approaches can not solve cold-start problem. Besides, task recommendation is much difficult than product recommendation, and workers do not have to give ratings to tasks to indicate the extent of their favor of each task. A crowdsourcing system needs some signals indicating types of available tasks, and the number of tasks workers select and complete [28].

Active learning

Active learning is a learning approach to achieve certain accuracy with a very low cost. Broadly speaking, active learning systems are based on either stream-based approach [29] or pool-based approach [30]. The stream-based approach considers one unlabeled instance each time, and decides whether to query its label or ignore it. This approach is useful when unlabeled instance is continuously available but cannot be stored easily, such as sensor data. However, as the stream-based approach relies on a real underlying input distribution, it is difficult to decide whether to query the label of an instance or ignore it at its arrival time. On the other hand, the pool-based approach ranks all unlabeled instances in order of informativeness, and queries the label for the most informative

unlabeled instance in the pool. The advantage is that large amount of unlabeled data are available in many domains at present, this approach is very important. However, it is difficult to find a good way to choose good queries from the pool. In both stream-based approach and pool-based approach, a query selection strategy is required to achieve high accuracy with as low labeling cost as possible.

Several query selection strategies are commonly used in active learning systems. For example, uncertainty sampling selects instances, which the current model is the most uncertain about, to query. There are many ways to measure the uncertainty, such as smallest margin [31], least confidence [32] and maximum entropy [33]. Another example of query selection strategies is query by committee [34]. This approach maintains a committee of independently trained classification models, and queries the instance for which the committee models disagree the most. Among all query selection strategies, uncertainty sampling is one of the simplest and widely used strategies in active learning systems [35].

Active learning for task recommendation in crowdsourcing systems to achieve quality assurance

Our motivation is the observation of the increase of difficulty for requesters to obtain good output with low cost. Currently, active learning approach has been applied on a number of quality assurance methods in crowdsourcing systems [35–37] because active learning can achieve certain accuracy by using fewer annotations even in noisy annotation scenarios. Laws et al. [36] demonstrated that actively selecting instances for label query can achieve performance gains in natural language processing tasks. However, they did not consider actively selecting annotators. To further improve the output quality, Yan et al. [35] proposed to use uncertainty sampling to select the most uncertain instance to query, and also uses an optimization formulation to choose multiple confident workers to query from. However, it is costly to pay for multiple labelers. To characterize the strength of each worker and improve the compentency of weak workers, Fang et al. [37] proposed a Self-Taught Active Learning paradigm, where a weak worker can learn complementary knowledge from a strong worker. Each labeler has a knowledge set (a set of confidence scores), which is used for worker selection. By using the knowledge set of the most reliable labeler to replace that of the most unreliable labeler, the most reliable labeler can teach the unreliable labeler. However, the learning curves of different labelers vary in real scenarios. Later, Fang and Zhu [38] proposed to use diversity density to characterize the oracle's uncertain knowledge. However, an oracle does not exist in real-world applications.

In recent years, a number of research works [39–42] proposed recommendation systems based on a Probabilistic Matrix Factorization (PMF) model to improve the output quality in crowdsourcing systems, where Probabilistic Matrix Factorization (PMF) is the state-of-the-art approach for recommendation systems. Jung and Lease [39] proposed to use a PMF model to infer unobserved labels to reduce the bias of the existing crowdsourced labels, thus improve the quality of labels. Later, Jung [40] proposed to use a PMF model to improve the quality of crowdsourcing tasks. Experimental results proved that the strength of PMF over Singular Value Decomposition (SVD) and baseline methods. However, it is not suitable for a huge number of tasks on crowdsourcing systems in reality. Yuen et al. [41, 42] considered various task categories in real scenarios in crowdsourcing systems and proposed a PMF model for task recommendation in crowdsourcing systems.

They proved that considering task categories in PMF can improve the performance. However, it does not consider to reduce the labeling cost by applying active learning approach. Later, Yuen et al. [15] proposed Probabilistic Matrix Factorization with Active Learning (version 1) for task recommendation systems. The model outperforms the PMF model with other active learning approaches, but new workers have to wait for a long time before having a list of perferred tasks recommended due to lack of worker performance history for all new workers.

Task recommendation for new workers in crowdsourcing systems

In crowdsourcing systems, workers prefer to have a list of recommended tasks, but they are not willing to work on a large number of tasks before having a list of preferred tasks recommended. Since new workers have not worked on a lot of tasks yet, it is difficult for a recommendation system to make a better recommendation for new workers due to the small working profiles of new workers. When performing active learning in recommendation systems, besides accuracy of recommendations and minimization of cost, it is also important for new workers to have a list of preferred tasks recommended as soon as they start working in a crowdsourcing system.

Dynamic-updating crowdsourcing systems for real-world scenarios

Some previous works proposed various ways to improve the performance of recommendation systems for real-world scenarios [13, 14]. Rendle et al. [13] proposed an online-updating algorithm for three kernal matrix factorization models, they are linear, logistic and linear non-negative matrix factorization model. They demonstrated that the output quality of their proposed online-updating algorithm approximates to that of fully retraining the models. However, the three kernal matrix factorization models are designed for user-item matrices. Besides, the user preference on item category, which is necessary for real-world applications, is not considered in the models. Karimi et al. [14] proposed an active learning method for aspect model in recommendation systems. They observed that "users are not willing to provide information for a large amount of items, thus the quality of recommendations is affected specially for new users" [14], and a full retrain of an aspect model needs a long time especially for a system of large number of existing users. In their online-updating method, the aspect model is updated to learn user latent factors for new users only and not the other users. Experimental results show that the user waiting time of their method is significantly less than that of bayesian method, but the prediction accuracy of their method is not always better than that of bayesian method.

Our motivation

Our motivation is the observation of the increase of difficulty for workers to find their preferred tasks [18, 43, 44], the increase of demand on task recommendation in crowdsourcing systems [45–47], and no previous works on task recommendation model for crowdsourcing systems that considers dynamic-updating for reducing the user waiting time on model update. To achieve certain output quality with a very low cost, we propose ActivePMFv2, Probabilistic Matrix Factorization with Active Learning (version 2), for task recommendation in crowdsourcing systems to actively select the most uncertain tasks and the most reliable workers for retraining the classification model. To reduce the

user waiting time on the learning model update, we propose a generic online-updating method for learning the model, ActivePMFv2.

Task recommendation framework

Our task recommendation framework (TaskRec) is based on matrix factorization method, to perform factor analysis to learn the worker latent feature, the task latent feature and the task category latent feature. To reduce the labeling cost and guarantee the output quality, Probabilistic Matrix Factorization with Active Learning (version 2) selects the most informative task to be learned and selects the best worker to query from.

The problem we study in this paper is how to effectively predict the missing values in the worker-task performance matrix so as to select the most informative task for the best worker to query from to achieve high accuracy with as few instances as possible. We define the problem of quality assurance in crowdsourcing systems as follows:

Definition 1 ***Quality assurance problem:*** *Given a set of workers* $WS = \{w_i\}_{i=1}^{m}$*, a set of tasks* $VS = \{v_j\}_{j=1}^{n}$*, a set of ratings* $R = \{r_{ij}\}$ *associated between worker* w_i *and task* v_j *where* $r_{ij} \in \mathbb{R}^{M \times N}$ *and only certain elements of R are initially known. The binary matrix* $I = \{I_{ij}\}$ *of the same shape as R represents the known points, so that* I_{ij} *is 1 if* r_{ij} *is observed and 0 otherwise. The set of (i, j) indexes where* $I_{ij} = 0$ *is denoted by PS. Predict the set of the unknown elements of* $R = \{r_{ij}\}$ *where* $(i, j) \in PS$*. The aim of ActivePMF is to query the most informative tasks selected from the set of the unknown elements of* $R = \{r_{ij}\}$ *where* $(i, j) \in PS$*, and to query from the most reliable workers.*

To facilitate our discussions, Table 1 defines basic terms and notations used throughout this paper.

Probabilistic Matrix Factorization on Task Recommendation Framework

Our model consists of three parts. First, we connect workers' task preferring information with workers' category preferring information through the shared worker latent feature

Table 1 Basic notations throughout this paper

Notation	Description
$WS = \{w_i\}_{i=1}^{m}$	WS is the set of workers, w_i is the i-th worker, m is the total number of workers
$VS = \{v_j\}_{j=1}^{n}$	VS is the set of tasks, v_j is the j-th task, n is the total number of tasks
$CS = \{c_k\}_{k=1}^{o}$	CS is the set of task categories, c_k is the k-th task category, o is the total number of task categories
$l \in \mathbb{R}$	l is the number of dimensions of latent feature space
$W \in \mathbb{R}^{l \times m}$	W is the worker latent feature matrix
$V \in \mathbb{R}^{l \times n}$	V is the task latent feature matrix
$C \in \mathbb{R}^{l \times o}$	C is the task category latent feature matrix
$R = \{r_{ij}\}$, $R \in \mathbb{R}^{m \times n}$	R is the worker-task preferring matrix, r_{ij} is the extent of the favor of task v_j for worker w_i
$U = \{u_{ik}\}$, $U \in \mathbb{R}^{m \times o}$	U is the worker-category preferring matrix, u_{ik} is the extent of worker w_i's preference for category c_k
$D = \{d_{jk}\}$, $D \in \mathbb{R}^{n \times o}$	D is the task-category grouping matrix, d_{jk} indicates the task category c_k that task v_j belongs to
$(i, j) \in PS$	PS is the set of indexes where the rating r_{ij} is unknown
$N(x\|\mu, \sigma^2)$	Probability density function of the Gaussian distribution with mean μ and variance σ^2

space. Second, we connect workers' task preferring information with tasks' category grouping information through the shared task latent feature space. Third, we connect workers' category preferring information with tasks' category grouping information through the shared category latent feature space. The graphical model of the TaskRec framework is represented in Fig. 1.

By using a worker-task preferring matrix, we can measure the extend the worker prefer to work the task and provide output that accepted by requesters. Unlike traditional recommendation systems, workers do not have to give ratings to tasks to indicate the extent of their favor of each task. To have ratings on tasks, we transform workers' behaviors into values as follows:

Worker Behavior		**Value**
Worker's work done is accepted by requester.	$\longrightarrow$	5
Worker's work done is rejected by requester.	$\longrightarrow$	4
Worker completes a task and submits the work done.	$\longrightarrow$	3
Worker selects a task to work on but not complete it.	$\longrightarrow$	2
Worker browses the detailed information of a task.	$\longrightarrow$	1
Worker does not browse the detailed information of a task.	$\longrightarrow$	0

In some cases, the ratings based on value transformation of worker behavior would be inaccurate on reflecting workers' task preference. For example, a worker's work done is being accepted, but he might not like the task very much.

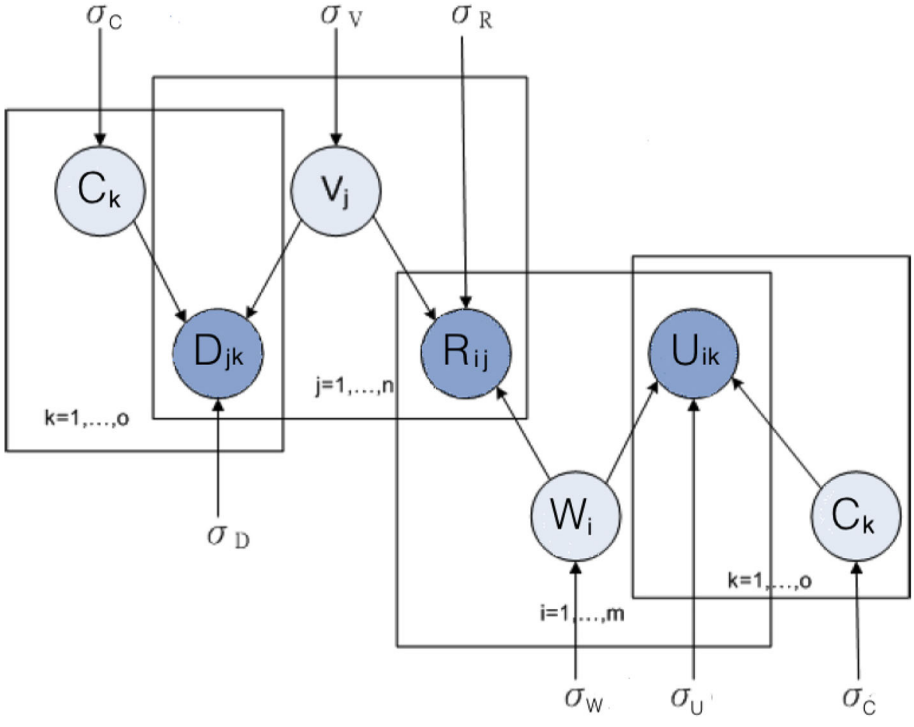

Fig. 1 Graphical Model for TaskRec. This is the graphical model for TaskRec, a task recommendation framework in crowdsourcing systems

Worker-task preferring matrix factorization

We have m workers, n tasks. The worker-task preferring matrix is denoted as R, the element r_{ij} in R means the extent of the favor of task v_j for worker w_i, where values of r_{ij} are within the range [0, 1]. Without loss of generality, we first map the ratings that inferred from worker behavior 1, ..., 5 to the interval [0, 1] using the function $f(x) = x/5$. Hence, we are given a partially observed worker-task preferring matrix, R, with m workers and n tasks.

To learn the workers' preference on the tasks, we employ matrix factorization, more specifically, Probabilistic Matrix Factorization (PMF) [12], to recover the worker-task preferring matrix. Given the partial observed matrix R, we aim at decomposing the matrix R into two l-dimensional low-rank feature matrices, W and V, where $W \in \mathbb{R}^{l\times m}$ is the latent feature matrix for workers with column vector W_i, and $V \in \mathbb{R}^{l\times n}$ is the latent feature matrix for tasks with column vector V_j.

To learn the matrices, a Gaussian distribution on the residual of the observed ratings is assumed as [12], and it is defined in Eq. (1):

$$p(R|W,V,\sigma_R^2) = \prod_{i=1}^{m}\prod_{j=1}^{n}\left[N(r_{ij}|g(W_i^T V_j),\sigma_R^2)\right]^{I_{ij}^R}, \tag{1}$$

where $N(x|\mu,\sigma^2)$ is the probability density function of the Gaussian distribution with mean μ and variance σ^2, and I_{ij}^R is the indicator function that is equal to 1 if the entry r_{ij} is observed and equal to 0 otherwise. The Gaussian distribution model can make predictions outside of the range of valid values. The function $g(x)$ is the logistic function $g(x) = 1/(1 + exp(-x))$, which makes it possible to bound the range of $W_i^T V_j$ within the range $[0, 1]$. Similar to [48], to avoid overfitting, zero-mean spherical Gaussian priors are also placed on the worker and task feature matrices, which are defined in Eq. (2):

$$p\left(W|\sigma_W^2\right) = \prod_{i=1}^{m} N\left(W_i|0,\sigma_W^2\right), p\left(V|\sigma_V^2\right) = \prod_{j=1}^{n} N\left(V_j|0,\sigma_V^2\right). \tag{2}$$

Hence, through a Bayesian inference, the posterior distributions of W and V based only on the observed ratings are derived in Eq. (3):

$$\begin{aligned} &p\left(W,V|R,\sigma_R^2,\sigma_W^2,\sigma_V^2\right) \propto p\left(R|W,V,\sigma_R^2\right) p\left(W|\sigma_W^2\right) p\left(V|\sigma_V^2\right) \\ &= \prod_{i=1}^{m}\prod_{j=1}^{n}\left[N\left(r_{ij}|g(W_i^T V_j),\sigma_R^2\right)\right]^{I_{ij}^R} \times \prod_{i=1}^{m} N(W_i|0,\sigma_W^2) \times \prod_{j=1}^{n} N\left(V_j|0,\sigma_V^2\right). \end{aligned} \tag{3}$$

Worker-category preferring matrix factorization

We have m workers and o task categories. The worker-category preferring matrix is denoted as U, where the element u_{ik} in U represents the extent of worker w_i's preference for task category c_k. Workers' performance histories indicate workers' preference for task categories, so the meaning of u_{ik} can be interpreted as whether the worker w_i has completed a task of the category c_k where the task is accepted (a binary representation), or how strong the worker w_i's preference is for the task category c_k (a real value representation).

We represent u_{ik} as shown in Eq. (4):

$$u_{ik} = g(f(w_i, c_k)), \tag{4}$$

where $g(.)$ is the logistic function, and $f(w_i, c_k)$ represents the number of times worker w_i completes a task of the category c_k where the task is accepted.

The idea of worker-category preferring matrix factorization is to derive two low-rank l-dimensional matrices W and C, where $W \in \mathbb{R}^{l \times m}$ and $C \in \mathbb{R}^{l \times o}$ are the latent feature matrices for workers and task categories, respectively. The column vectors W_i and C_k representing the l-dimensional worker-specific and category-specific latent feature vectors of worker w_i and category c_k, respectively. We can define the conditional distributions over the observed worker-category preferring matrix in Eq. (5):

$$p(U|W, C, \sigma_U^2) = \prod_{i=1}^{m} \prod_{k=1}^{o} \left[N(u_{ik}|g(W_i^T C_k), \sigma_U^2) \right]^{I_{ik}^U}, \tag{5}$$

where $N(x|\mu, \sigma^2)$ is the probability density function of the Gaussian distribution with mean μ and variance σ^2, and I_{ik}^U is the indicator function that is equal to 1 if worker w_i has at least one completed task of the category c_k being accepted and equal to 0 otherwise.

To avoid overfitting, zero-mean spherical Gaussian priors are placed on the worker and the category latent feature matrices, which are defined in Eq. (6):

$$p(W|\sigma_W^2) = \prod_{i=1}^{m} N(W_i|0, \sigma_W^2), p(C|\sigma_C^2) = \prod_{k=1}^{o} N(C_k|0, \sigma_C^2). \tag{6}$$

Hence, through a Bayesian inference, the posterior distributions of W and C based only on the observed ratings are derived in Eq. (7):

$$\begin{aligned} &p(W, C|U, \sigma_C^2, \sigma_W^2, \sigma_U^2) \propto p(U|W, C, \sigma_U^2) p(W|\sigma_W^2) p(C|\sigma_C^2) \\ &= \prod_{i=1}^{m} \prod_{k=1}^{o} \left[N(u_{ik}|g(W_i^T C_k), \sigma_U^2) \right]^{I_{ik}^U} \times \prod_{i=1}^{m} N(W_i|0, \sigma_W^2) \times \prod_{k=1}^{o} N(C_k|0, \sigma_C^2). \end{aligned} \tag{7}$$

Task-category grouping matrix factorization

We have n tasks and o task categories. The task-category grouping matrix is denoted as D, where the element d_{jk} in D shows the category c_k that task v_j belongs to. The meaning of d_{jk} can be interpreted as whether the task v_j belongs to the category c_k (a binary representation) . We represent d_{jk} as shown in Eq. (8):

$$d_{jk} = f(v_j, c_k), \tag{8}$$

where $f(v_j, c_k)$ is an indicator variable with the value of 1 if the task v_j belongs to the category c_k, and 0 otherwise.

The idea of task-category grouping matrix factorization is to derive two low-rank l-dimensional matrices V and C, where $V \in \mathbb{R}^{l \times n}$ and $C \in \mathbb{R}^{l \times o}$ are the latent feature matrices for tasks and task categories, respectively. The column vectors V_j and C_k representing the l-dimensional task-specific and category-specific latent feature vectors of task v_j and category c_k, respectively. We can define the conditional distributions over the

observed task-category grouping matrix in Eq. (9):

$$p(D|V,C,\sigma_D^2) = \prod_{j=1}^{n}\prod_{k=1}^{o}\left[N(d_{jk}|g(V_j^T C_k),\sigma_D^2)\right]^{I_{jk}^D}, \tag{9}$$

where $N(x|\mu,\sigma^2)$ is the probability density function of the Gaussian distribution with mean μ and variance σ^2, and I_{jk}^D is the indicator function that is equal to 1 if the entry d_{jk} is observed and equal to 0 otherwise.

To avoid overfitting, zero-mean spherical Gaussian priors are placed on the task and the category latent feature matrices, which are defined in Eq. (10):

$$p(V|\sigma_V^2) = \prod_{j=1}^{n} N(V_j|0,\sigma_V^2), p(C|\sigma_C^2) = \prod_{k=1}^{o} N(C_k|0,\sigma_C^2). \tag{10}$$

Hence, through a Bayesian inference, the posterior distributions of V and C based only on the observed ratings are derived in Eq. (11):

$$\begin{aligned} &p(V,C|D,\sigma_C^2,\sigma_V^2,\sigma_D^2) \propto p(D|V,C,\sigma_D^2)p(V|\sigma_V^2)p(C|\sigma_C^2) \\ =&\prod_{j=1}^{n}\prod_{k=1}^{o}\left[N(d_{jk}|g(V_j^T C_k),\sigma_D^2)\right]^{I_{jk}^D} \times \prod_{j=1}^{n} N(V_j|0,\sigma_V^2) \times \prod_{k=1}^{o} N(C_k|0,\sigma_C^2). \end{aligned} \tag{11}$$

A unified matrix factorization for TaskRec

According to the graphical model of the TaskRec framework described in Fig. 1, we derive the log function of the posterior distributions of TaskRec in Eq. (12):

$$\begin{aligned} &\ln p(W,V,C|R,U,D,\sigma_W^2,\sigma_V^2,\sigma_C^2,\sigma_R^2,\sigma_U^2,\sigma_D^2) \\ =&-\frac{1}{2\sigma_R^2}\sum_{i=1}^{m}\sum_{j=1}^{n} I_{ij}^R\left(r_{ij}-g\left(W_i^T V_j\right)\right)^2 - \frac{1}{2\sigma_U^2}\sum_{i=1}^{m}\sum_{k=1}^{o} I_{ik}^U\left(u_{ik}-g\left(W_i^T C_k\right)\right)^2 \\ &-\frac{1}{2\sigma_D^2}\sum_{j=1}^{n}\sum_{k=1}^{o} I_{jk}^D\left(d_{jk}-g\left(V_j^T C_k\right)\right)^2 - \frac{1}{2\sigma_W^2}\sum_{i=1}^{m} W_i^T W_i - \frac{1}{2\sigma_V^2}\sum_{j=1}^{n} V_j^T V_j \\ &-\frac{1}{2\sigma_C^2}\sum_{k=1}^{o} C_k^T C_k - \sum_{i=1}^{m}\sum_{j=1}^{n} I_{ij}^R \ln\sigma_R - \sum_{i=1}^{m}\sum_{k=1}^{o} I_{ik}^U \ln\sigma_U - \sum_{j=1}^{n}\sum_{k=1}^{o} I_{jk}^D \ln\sigma_D \\ &-l\sum_{i=1}^{m}\ln\sigma_W - l\sum_{j=1}^{n}\ln\sigma_V - l\sum_{k=1}^{o}\ln\sigma_C + \mathcal{C}, \end{aligned} \tag{12}$$

where $\mathcal{C}$ is a constant independent of the parameters. We can see the Eq. (12) is an unconstrained optimization problem, and maximizing the log-posterior distributions with fixed hyper parameters is equivalent to minimizing the sum-of-squared-errors objective function with quadratic regularized terms in Eq. (13):

$$
\begin{aligned}
&E(W, V, C, R, U, D) \\
=&\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n} I_{ij}^{R}\left(r_{ij} - g\left(W_i^T V_j\right)\right)^2 + \frac{\theta_U}{2}\sum_{i=1}^{m}\sum_{k=1}^{o} I_{ik}^{U}\left(u_{ik} - g\left(W_i^T C_k\right)\right)^2 \\
&+ \frac{\theta_D}{2}\sum_{j=1}^{n}\sum_{k=1}^{o} I_{jk}^{D}\left(d_{jk} - g\left(V_j^T C_k\right)\right)^2 \\
&+ \frac{\theta_W}{2}\sum_{i=1}^{m} W_i^T W_i + \frac{\theta_V}{2}\sum_{j=1}^{n} V_j^T V_j + \frac{\theta_C}{2}\sum_{k=1}^{o} C_k^T C_k,
\end{aligned} \tag{13}
$$

where $\theta_U = \sigma_R^2/\sigma_U^2$, $\theta_D = \sigma_R^2/\sigma_D^2$, $\theta_W = \sigma_R^2/\sigma_W^2$, $\theta_V = \sigma_R^2/\sigma_V^2$, and $\theta_C = \sigma_R^2/\sigma_C^2$. The local minimum can be found by performing the gradient descent on W_i, V_j and C_k, and the derived gradient descent equations are described in Eq. (14), Eq. (15) and Eq. (16) respectively:

$$
\begin{aligned}
\frac{\partial E}{\partial W_i} =& \sum_{j=1}^{n} I_{ij}^{R}\left(g\left(W_i^T V_j\right) - r_{ij}\right) g'\left(W_i^T V_j\right) V_j + \theta_W W_i \\
&+ \theta_U \sum_{k=1}^{o} I_{ik}^{U}\left(g\left(W_i^T C_k\right) - u_{ik}\right) g'\left(W_i^T C_k\right) C_k,
\end{aligned} \tag{14}
$$

$$
\begin{aligned}
\frac{\partial E}{\partial V_j} =& \sum_{i=1}^{m} I_{ij}^{R}\left(g\left(W_i^T V_j\right) - r_{ij}\right) g'\left(W_i^T V_j\right) W_i + \theta_V V_j \\
&+ \theta_D \sum_{k=1}^{o} I_{jk}^{D}\left(g\left(V_j^T C_k\right) - d_{jk}\right) g'\left(V_j^T C_k\right) C_k,
\end{aligned} \tag{15}
$$

$$
\begin{aligned}
\frac{\partial E}{\partial C_k} =& \theta_U \sum_{i=1}^{m} I_{ik}^{U}\left(g\left(W_i^T C_k\right) - u_{ik}\right) g'\left(W_i^T C_k\right) W_i + \theta_C C_k \\
&+ \theta_D \sum_{j=1}^{n} I_{jk}^{D}\left(g\left(V_j^T C_k\right) - d_{jk}\right) g'\left(V_j^T C_k\right) V_j,
\end{aligned} \tag{16}
$$

where $g'(.)$ is the first-order derivative of the logistic function. To reduce the model complexity, we set $\theta_W = \theta_V = \theta_C$ in our experiments. The training time for our model scales linearly with the number of observations.

ActivePMFv2 - active learning for concurrent selection of task and worker

We first query all new tasks, which have not been tried by anyone, from the most reliable worker. Then, we query the most uncertain task from all new workers. Next, we query the most uncertain task, and select the most reliable worker for the task to query from. Our proposed ActivePMFv2 is presented in Algorithm 1.

Random new task selection for reliable worker

To learn the most accurate classifier with the least number of work done, we first query all new tasks, as given in Eq. (17), and select the most reliable worker in the task category to query from, as given in Eq. (18).

Algorithm 1 Probabilistic Matrix Factorization with Active Learning (version 2), ActivePMFv2

Input:

Partially observed worker-task matrix, R;

Maximum num of queries, *MaxQueries*;

Active-sampling heuristic, h (use uncertainty-sampling using the Maximum Difference between predicted rate and observed rate: $\arg\max_{v_j \in VS} \sum_{i=1}^{m} \frac{1}{\sum_{i=1}^{m} I_{ij}} \left| I_{ij}^{R} \left(g\left(W_i^T V_j \right) - r_{ij} \right) \right|$);

Output:

Full worker-task matrix R_{full} valued within the interval [0, 1] predicting unobserved entries of R;

Initialize:

$R_{tmp} = R$; /* currently observed data */

NumQueries = 0; /*num of queries done by workers*/

1: $\bar{R}_{full}$ = PMF(R_{tmp}); /* compute full matrix $\bar{R}_{full}$ */

2: *UnO* = set of unobserved entries of R_{tmp};

3: *NewT* = Select all new tasks;

4: *NewW* = Select all new workers;

5: *Set* = ActiveSelect(h, $\bar{R}_{full}$, *UnO*); /* select the most uncertain unobserved instances (Maximum Difference between predicted rate and observed rate) from *UnO-New* using h and current predictions $\bar{R}_{full}$ */

6: **if** $|NewT| > 0$ **then**

Select a new task v^* from *New* using Eq. (17);

Select the most reliable worker w^* for task v^* using Eq. (18);

Request worker w^* to work on task v^*;

Add the rate to R_{tmp};

NumQueries = *NumQueries* + 1;

7: **else**

8: **if** $|NewW| > 0$ **then**

Select the most uncertain task v^{**} from *Set* using Eq. (19);

Select a new worker w^{**} for task v^{**} using Eq. (20);

Request worker w^{**} to work on task v^{**};

Add the rate to R_{tmp};

NumQueries = *NumQueries* + 1;

9: **else**

Select the most uncertain task v^{***} from *Set* using Eq. (21);

Select the most reliable worker w^{***} for task v^{***} using Eq. (22);

Request worker w^{***} to work on task v^{***};

Add the rate to R_{tmp};

NumQueries = *NumQueries* + 1;

10: **end if**

11: **end if**

12: **if** (*NumQueries* < *MaxQueries*) **then**

Go to Step 1

13: **else**

14: **return** $\bar{R}_{full}$;

15: **end if**

$$v^* = \{v_j | \exists v_j \in VS; I_{ij} = 0, \forall w_i \in WS\}, \tag{17}$$

$$w^* = \arg\max_{w_i \in WS} u_{ik} \quad where \quad d_{jk} = 1, v_j = v^*, \tag{18}$$

In Algorithm 1, Step 6 represents the process of new task selection for most reliable worker in the category.

Uncertainty sampling for task selection for randomly selected new worker

The algorithm assumes a particular active learning heuristic specified as an input, and we adopt uncertainty-sampling [30] using the Maximum Difference between predicted rate and observed rate as in Eq. (19) to choose the most uncertain task, that requires minimization of uncertainty. To let new workers having a list of preferred tasks recommended but not having to work on a large amount of tasks beforehand, we randomly select a new worker (if any) to query from, as given in Eq. (20).

$$v^{**} = \arg\max_{v_j \in VS} \sum_{i=1}^{m} \frac{1}{\sum_{i=1}^{m} I_{ij}} \left| I_{ij}^R \left(g\left(W_i^T V_j \right) - r_{ij} \right) \right|, \tag{19}$$

$$w^{**} = \{w_i | \exists w_i \in WS; I_{ij} = 0, \forall v_j \in VS\}. \tag{20}$$

In Algorithm 1, Step 8 represents the process of most uncertain task selection for new worker.

Uncertainty sampling for task selection for reliable worker

The algorithm assumes a particular active learning heuristic specified as an input, and we adopt uncertainty-sampling [30] using the Maximum Difference between predicted rate and observed rate as in Eq. (21) to choose the most uncertain task, that requires minimization of uncertainty. To select the most reliable worker for the most uncertain task, we select the worker with the maximum worker-category perferring score where the category that the task belongs to as in Eq. (22).

$$v^{***} = \arg\max_{v_j \in VS} \sum_{i=1}^{m} \frac{1}{\sum_{i=1}^{m} I_{ij}} \left| I_{ij}^R \left(g\left(W_i^T V_j \right) - r_{ij} \right) \right|, \tag{21}$$

$$w^{***} = \arg\max_{w_i \in WS} u_{ik} \quad where \quad d_{jk} = 1, v_j = v^{***}. \tag{22}$$

In Algorithm 1, Step 9 represents the most uncertain task selection for the most reliable worker.

After annotation, the selected task is removed from the unlabeled data set. Next, the selected task and its rate are added to the set of labeled dataset. The model is then retrained on the labeled tasks and the informativeness of the remaining tasks in the unlabeled data set is re-evaluated.

Online-updating on ActivePMFv2 - online-update on active learning for concurrent selection of task and worker

In this section, we present our online-updating approach for learning the matrix factorization model, ActivePMFv2 model. The online-updating approach of ActivePMFv2 is

presented in Algorithm 2, while the full retrain approach of ActivePMFv2 is presented in Algorithm 1. The online-updating approach uses the same sampling heuristics and the same active learning approach in query method as that of the full retrain approach. The main difference between the online-updating approach and the full retrain approach is the model update methods they used. In the full retrain approach, it retrains the whole learning model after each work done. In the online-updating approach, it has two main parts: (1) It retrains the learning model in batch mode where model update occurs after a number of work done; (2) For each work done related to a worker (or task) having profile larger than the threshold, it updates the whole feature vector of the worker (or task) and keeps all other entries in the feature matrix fixed. On the other hand, for work done related to a worker (or task) having profile smaller than the threshold, it updates the whole feature matrix.

Partial update

The impact on retraining the whole learning model decreases as the profile size of the worker (or task) increases. Especially when work done by new workers or work done on task having small profile, updating the feature matrix is crucial. For a new worker, each work done by him will result in much change in his task perference in his worker profile; while for a worker that has already completed a lot of tasks, each work done by him will not change much in his worker profile. Updating feature vectors for a worker (or task) having smaller profile results in a much better model. As a result, for a worker (or task) having large profile, we observe that the model learned from retraining the feature vector of the worker (or task) only is approximate to that learned from a full retrain.

As mentioned before, through a Bayesian inference, the posterior distributions of W and V based only on the observed ratings are derived in Eq. (3), while the posterior distributions of W and C based only on the observed ratings are derived in Eq. (7). For partial update on a large worker profile, we only retrain the feature vector of the selected worker $w_{m'}$ in worker-task preferring matrix and worker-category preferring matrix as shown in Eq. (23) and Eq. (24) respectively.

$$FV_{WV}(w_{m'}) = \prod_{i=m'}^{m'} \prod_{j=1}^{n} \left[N(r_{ij}|g(W_i^T V_j), \sigma_R^2) \right]^{I_{ij}^R} \times \prod_{i=m'}^{m'} N(W_i|0, \sigma_W^2 \mathbb{I}) \times \prod_{j=1}^{n} N(V_j|0, \sigma_V^2 \mathbb{I}), \quad (23)$$

$$FV_{WC}(w_{m'}) = \prod_{i=m'}^{m'} \prod_{k=1}^{o} \left[N(u_{ik}|g(W_i^T C_k), \sigma_U^2) \right]^{I_{ik}^U} \times \prod_{i=m'}^{m'} N(W_i|0, \sigma_W^2 \mathbb{I}) \times \prod_{k=1}^{o} N(C_k|0, \sigma_C^2 \mathbb{I}). \quad (24)$$

As stated previously, through a Bayesian inference, the posterior distributions of W and V based only on the observed ratings are derived in Eq. (3), while the posterior distributions of V and C based only on the observed ratings are derived in Eq. (11). For partial update on a large task profile, we only retrain the feature vector of the selected task $v_{n'}$ in worker-task preferring matrix and task-category grouping matrix as shown in Eq. (25) and Eq. (26) respectively.

Algorithm 2 Online-Updating on ActivePMFv2

Input:

Partially observed worker-task matrix, R;
Threshold, *Threshold*;
Batch size, *BatchSize*;
Active-sampling heuristic, h (use uncertainty-sampling using the Maximum Difference between predicted rate and observed rate: $\arg\max_{v_j \in VS} \sum_{i=1}^{m} \frac{1}{\sum_{i=1}^{m} I_{ij}} \left| I_{ij}^{R} \left(g\left(W_i^T V_j \right) - r_{ij} \right) \right|$);

Output:

Full worker-task matrix R_{full} valued within the interval [0, 1] predicting unobserved entries of R;

Initialize:

$R_{tmp} = R$; /* currently observed data */
NumQueries = 0; /*num of queries done by workers*/

1: $\bar{R}_{full}$ = PMF(R_{tmp}); /* compute full matrix $\bar{R}_{full}$ */
2: *Un* = set of unobserved entries of R_{tmp};
3: *NewT* = Select all new tasks;
4: *NewW* = Select all new workers;
5: *Set* = ActiveSelect(h, $\bar{R}_{full}$, *Un*); /* select the most uncertain unobserved instances (Maximum Difference between predicted rate and observed rate) from *Un-New* using h and current predictions $\bar{R}_{full}$ */
6: **if** $|NewT| > 0$ **then**
  Select a new task v^* from *New* using Eq. (17);
  Select the most reliable worker w^* for task v^* using Eq. (18);
  Request worker w^* to work on task v^*;
  Add the rate to R_{tmp};
  NumQueries = *NumQueries* + 1;
7: **else**
8: **if** $|NewW| > 0$ **then**
    Select the most uncertain task v^{**} from *Set* using Eq. (19);
    Select a new worker w^{**} for task v^{**} using Eq. (20);
    Request worker w^{**} to work on task v^{**};
    Add the rate to R_{tmp};
    NumQueries = *NumQueries* + 1;
9: **else**
    Select the most uncertain task v^{***} from *Set* using Eq. (21);
    Select the most reliable worker w^{***} for task v^{***} using Eq. (22);
    Request worker w^{***} to work on task v^{***};
    Add the rate to R_{tmp};
    NumQueries = *NumQueries* + 1;
10: **end if**
11: **end if**
12: **if** (Worker Profile >= *Threshold*) and (Task Profile >= *Threshold*) **then**
  Update the feature vectors of the selected worker $w_{m'}$ using Eq. (23), Eq. (24);
  Update the feature vectors of the selected task $v_{n'}$ using Eq. (25), Eq. (26);
13: **else**
14: **if** (Worker Profile >= *Threshold*) **then**
    Update the feature vectors of the selected worker $w_{m'}$ using Eq. (23), Eq. (24);
15: **else**
16: **if** (Task Profile >= *Threshold*) **then**
      Update the feature vectors of the selected task $v_{n'}$ using Eq. (25), Eq. (26);
17: **else**
      Update the feature vectors of all workers;
      Update the feature vectors of all tasks;
18: **end if**
19: **end if**
20: **end if**
21: **if** (No new incoming work done) **then**
22: **return** $\bar{R}_{full}$;
23: **end if**
24: **if** (*NumQueries* mod *BatchSize* = 0) **then**
  NumQueries = 0;
  Go to Step 1;
25: **else**
  Go to Step 2;
26: **end if**

$$FV_{WV}(v_{n'}) = \prod_{i=1}^{m}\prod_{j=n'}^{n'}\left[N(r_{ij}|g(W_i^T V_j),\sigma_R^2)\right]^{I_{ij}^R} \times \prod_{i=1}^{m} N(W_i|0,\sigma_W^2\mathbb{I}) \times \prod_{j=n'}^{n'} N(V_j|0,\sigma_V^2\mathbb{I}), \quad (25)$$

$$FV_{VC}(v_{n'}) = \prod_{j=n'}^{n'}\prod_{k=1}^{o}\left[N(d_{jk}|g(V_j^T C_k),\sigma_D^2)\right]^{I_{jk}^D} \times \prod_{j=n'}^{n'} N(V_j|0,\sigma_V^2\mathbb{I}) \times \prod_{k=1}^{o} N(C_k|0,\sigma_C^2\mathbb{I}). \quad (26)$$

In Algorithm 2, Step 12, 14 and 16 represents the process of partial update. For a worker (or task) having profile size larger than the threshold, the algorithm retrains the feature vector of the worker (or task) and keep all other entries in the matrix unchanged; otherwise, the algorithm retrains the feature vectors of all workers and all tasks.

Batch update

The time for retraining a learning model is proportional to the computational complexity of the model and the amount of information stored in the model; while the amount of information depends on the number of workers, the number of tasks and the number of work done. Retraining a large learning model takes a long time. For a large real-world crowdsourcing system, it is inefficient if the whole model is retrained from scratch once a worker completes a task.

In Algorithm 2, Step 24 and 25 represents the process of batch update. When the number of work done is smaller than the batch size, the learning model is not retrained. On the other hand, when the number of work done is larger than the batch size, the algorithm retrains the learning model.

General update problem

Our proposed online-updating approach can also be applied in a general update problem. In a general recommendation system, a new rating $r_{u,i}$ might affect the features of both user u and item i. The partial update method works based on the following conditions: (1) If the profiles of both user and item are large, it could update the feature vectors of both the selected user and the selected item; (2) If the user profile is large (but the item profile is small), it could update the user feature vector only; (3) If the item profile is large (but the user profile is small), it could update the item feature vector only; (4) If the profiles of both user and item are small, it could update the feature vectors of all users and all items. Besides, based on the number of new incoming ratings, the batch update method retrains the whole learning model. A general update algorithm is shown in Algorithm 3.

Complexity analysis

To compute the complexity of our ActivePMFv2, we consider both the computation of the gradient descent methods and the computation of selecting the most uncertain task for the most reliable worker. The main computation of the gradient descent methods is evaluating objective function E and corresponding gradients on variables. Because of the sparsity of matrices R, U, and D, the complexity of evaluating the objective function in Eq. (13) is $\mathcal{O}(n_R l + n_U l + n_D l)$, where n_R, n_U and n_D are the number of non-zero entries

Algorithm 3 General Online-Updating Algorithm

Input:
Partially observed user-item matrix, R;
Threshold, *Threshold*;
Batch size, *BatchSize*;
Active-sampling heuristic, h;

Output:
Full user-item matrix R_{full} valued within the interval [0, 1] predicting unobserved entries of R;

Initialize:
$R_{tmp} = R$; /* currently observed data */
NumQueries = 0; /*num of queries done by workers*/

1: $\bar{R}_{full}$ = compute full matrix $\bar{R}_{full}$ by using R_{tmp}
2: Un = set of unobserved entries of R_{tmp};
Request the rating from user u on item i;
Add the rate to R_{tmp};
NumQueries = *NumQueries* + 1;
3: **if** (User Profile >= *Threshold*) and (Item Profile >= *Threshold*) **then**
Update the feature vector of the user in the matrix R_{tmp};
Update the feature vector of the item in the matrix R_{tmp};
4: **else**
5: **if** (User Profile >= *Threshold*) **then**
Update the feature vector of the user in the matrix R_{tmp};
6: **else**
7: **if** (Item Profile >= *Threshold*) **then**
Update the feature vector of the item in the matrix R_{tmp};
8: **else**
Update the feature vectors of all users;
Update the feature vectors of all items;
9: **end if**
10: **end if**
11: **end if**
12: **if** (No new incoming rating) **then**
13: **return** $\bar{R}_{full}$;
14: **end if**
15: **if** (*NumQueries* mod *BatchSize* $= 0$) **then**
NumQueries = 0;
Go to Step 1;
16: **else**
Go to Step 2;
17: **end if**

in matrices R, U, and D respectively, and l is the number of dimensions of latent feature space. By using the similar approach, we can derive the complexities of Eq. (14), Eq. (15) and Eq. (16). For the computation of selecting the most uncertain task for the most reliable worker, the complexity of selecting the most uncertain task in Eq. (21) is $\mathcal{O}(n_R l)$, the complexity of selecting the most reliable worker in Eq. (22) is $\mathcal{O}(m)$, and thus the total complexity of assigning a task to a worker is $\mathcal{O}(m + n_R l)$. As a result, the total complexity for one iteration is $\mathcal{O}(m + n_R l + n_U l + n_D l)$. It means that the complexity is linear with respect to the number of workers and the number of observations in the three sparse matrices. The complexity analysis shows that ActivePMFv2 can scale to very large datasets.

To apply online-updating approach when learning ActivePMFv2 model, we consider both the partial update method and the batch update method. Since only some non-zero entries in matrices is updated in the partial update method, the complexity of evaluating the objective function in Eq. (13) is $\mathcal{O}(z_R l + z_U l + z_D l)$, where z_R, z_U and z_D are the number of non-zero entries to be updated in matrices R, U, and D respectively, and l is the number of dimensions of latent feature space. As a result, the total complexity for one

iteration is $\mathcal{O}(m + z_R l + z_U l + z_D l)$, where $z_R << n_R$, $z_U << n_U$, $z_D << n_D$. Based on our observation, the complexity of learning ActivePMFv2 model can be highly reduced by using the partial update method. Besides, the batch method can greatly reduce the number of iterations, thus further reduce the computational complexity.

Experimental analysis

In this section, our experiments are intended to address the following five research questions:

1. How is ActivePMFv1 approach compared with PMF with various active learning approaches?
2. How is ActivePMFv1 approach compared with PMF with various active sampling heuristics?
3. How is our ActivePMFv2 approach compared with ActivePMFv1?
4. How is the partial update part of online-updating approach compared with the full retrain when learning ActivePMFv2 model?
5. How is the batch update part of online-updating approach compared with the full retrain when learning ActivePMFv2 model?

Description of dataset

We use the same dataset as shown in [41]. Our dataset is retrieved from the recent NAACL 2010 workshop on crowdsourcing, which has made publicly available all the data collected as part of the workshop [49]. The data was collected within a month from multiple requesters seeking data for a diverse variety of tasks on MTurk. Table 2 provides some statistics about our dataset. Our dataset is mainly related to tasks for creating speech and language data. The task categorization is shown in Table 3.

Evaluation metrics

For performance comparison with our proposed ActivePMFv2, we implement PMF (Probabilistic Matrix Factorization) with the following Active Learning baselines:

- **T[Rand]W[Rand]:** It assigns a randomly selected task to a randomly selected worker.
- **T[MaxDiff]W[Rand]:** It assigns the task of maximum difference between its observed values and its predicted values to a randomly selected worker.
- **T[Rand]W[Reli]:** It assigns a randomly selected task to the most reliable worker in the task category.

Table 2 Statistics of our dataset

Number of workers	1,592
Number of different tasks	6,639
Number of categories	43
Total HITs from all tasks	19,815
Number of ratings	19,815
Max number of HITs of a worker	2,691
Min number of HITs of a worker	1
Average number of HITs of a worker	12.4
1st quartile (25th percentile) of number of HITs of a worker	1
2nd quartile (50th percentile) of number of HITs of a worker	2
3rd quartile (75th percentile) of number of HITs of a worker	5

Table 3 Task categorization by both language and keywords given by MTurk in our dataset

1	English-Afrikaans translations	23	English-Romanian translations
2	English-Azeri translations	24	English-Russian translations
3	English-Bulgarian translations	25	English-Slovak translations
4	English-Bangla translations	26	English-Somali translations
5	English-Bosnian translations	27	English-Albanian translations
6	English-Welsh translations	28	English-Serbian translations
7	English-Spanish translations	29	English-Tamil translations
8	English-Basque translations	30	English-Thai translations
9	English-Farsi translations	31	English-Turkmen translations
10	English-Irish translations	32	English-Tagalog translations
11	English-Hindi translations	33	English-Turkish translations
12	English-Indonesian translations	34	English-Tatar translations
13	English-Korean translations	35	English-Ukrainian translations
14	English-Kurdish translations	36	English-Urdu translations
15	English-Latin translations	37	English-Uzbek translations
16	English-Latvian translations	38	English annotations
17	English-Mongolian translations	39	Spanish annotations
18	English-Maltese translations	40	Arabic annotations
19	English-Nepali translations	41	English relevance judgment
20	English-Punjabi translations	42	English creative writing
21	English-Kapampangan translations	43	English transcription
22	English-Polish translations		

- **T[MaxDiff]W[Reli]:** It assigns the task of maximum difference between its observed values and its predicted values to the most reliable worker in the task category.
- **T[N]W[Reli]+T[MaxPredictErr]W[Reli]:** It has two parts. First, it assigns all new tasks to the most reliable worker in the task categories. Second, it assigns the task of maximum average prediction error of all its predicted values to the most reliable worker in the task category.
- **T[N]W[Reli]+T[MaxEntropy]W[Reli]:** It has two parts. First, it assigns all new tasks to the most reliable worker in the task categories. Second, it assigns the task of maximum Entropy on the posterior variance to the most reliable worker in the task category.
- **T[N]W[Reli]+T[MaxDiff]W[Reli] (ActivePMFv1):** It has two parts. First, it assigns all new tasks to the most reliable worker in the task categories. Second, it assigns the task of maximum difference between its observed values and its predicted values to the most reliable worker in the task category.
- **T[N]W[Reli]+T[MaxDiff]W[N]+T[MaxDiff]W[Reli] (ActivePMFv2):** It has three parts. First, it assigns all new tasks to the most reliable worker in the task categories. Second, for all new workers, it assigns the task of maximum difference between its observed values and its predicted values to a randomly selected new worker. Third, it assigns the task of maximum difference between its observed values and its predicted values to the most reliable worker in the task category.

To compare the prediction quality of our method ActivePMFv2, we use the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE) as the comparison metrics. MAE and RMSE are defined in Eq. (27):

$$MAE = \frac{\sum_{i,j} |r_{i,j} - \hat{r}_{i,j}|}{N}, RMSE = \sqrt{\frac{\sum_{i,j} \left(r_{i,j} - \hat{r}_{i,j}\right)^2}{N}}, \quad (27)$$

where $r_{i,j}$ denotes the rating that indicates the extent of the favor of task j for worker i, $\hat{r}_{i,j}$ denotes the predicted rating, and N is the total number of testing ratings.

Performance comparison

To show the prediction performance improvements of ActivePMFv2, we first compare ActivePMFv1 [15] with PMF with various active learning approaches and different active sampling heuristics, where Probabilistic Matrix Factorization (PMF) [12] is the state-of-the-art approach for recommendation systems. Next, we compare our proposed ActivePMFv2 with ActivePMFv1.

From our dataset, we randomly select 80 % of ratings as training data (20 % as initial training set + 60 % as active set), and leave the remaining 20 % as prediction performance testing. The procedure is carried out 10 times independently, and we report the average values in this paper. For the value transformation, we have 10,411 approved tasks (value transformed to 5), 9,399 submitted tasks (value transformed to 3) and only 5 rejected tasks (value transformed to 4). Most rejected tasks are already removed in our dataset. In the comparison, we set $\theta_W = \theta_V = \theta_C = 0.00004$, set $\theta_U = 0.0001$ and $\theta_D = 0.01$. The MAE results and the RMSE results are reported from Tables 4, 5, 6, 7, 8 and 9. MAE measures the average magnitude of the errors in predicted values; while RMSE gives a relative high weight to large errors.

In Tables 4 and 5, we compare among PMF with three sampling heuristics on selecting tasks, which are Maximum Difference (ActivePMFv1), Maximum Prediction Error and Maximum Entropy. The performance results of the three approaches are similar when the number of selected samples is small. However, as the number of selected samples increases, Maximum Difference approach (ActivePMFv1) outperforms the other two approaches.

In Tables 6 and 7, we compare among PMF with different active learning approaches on task selection and worker selection. When the number of selected samples is very small,

Table 4 MAE comparison among various active sampling heuristics in PMF (A smaller MAE means a better performance)

	PMF with active sampling heuristics		
No. of selected samples	T[N]W[Reli]+ T[MaxDiff]W[Reli] (ActivePMFv1)	T[N]W[Reli]+ T[MaxPredictErr]W[Reli]	T[N]W[Reli]+ T[MaxEntropy]W[Reli]
1000	0.3658	0.3658	0.3658
2000	0.2778	0.2778	0.2778
3000	0.1955	0.1955	0.1955
4000	0.1473	**0.1443**	0.1482
5000	**0.1151**	0.1184	0.1262
6000	**0.0977**	**0.0977**	0.1100
7000	0.0945	**0.0839**	0.0963
8000	0.0899	**0.0664**	0.0809
9000	0.0652	**0.0479**	0.0536
10000	**0.0334**	0.0340	0.0366
11000	**0.0176**	0.0221	0.0237

Table 5 RMSE comparison among various active sampling heuristics in PMF (A smaller MAE means a better performance)

No. of selected samples	PMF with active sampling heuristics		
	T[N]W[Reli]+ T[MaxDiff]W[Reli] (ActivePMFv1)	T[N]W[Reli]+ T[MaxPredictErr]W[Reli]	T[N]W[Reli]+ T[MaxEntropy]W[Reli]
1000	0.7101	0.7101	0.7101
2000	0.6097	0.6097	0.6097
3000	0.5110	0.5110	0.5110
4000	**0.4164**	0.4364	0.4583
5000	**0.3180**	0.4025	0.4409
6000	**0.2621**	0.3626	0.4300
7000	**0.2566**	0.3411	0.4195
8000	**0.2563**	0.2862	0.3856
9000	**0.2054**	0.2342	0.3023
10000	**0.1259**	0.1975	0.2390
11000	**0.0880**	0.1532	0.1786

assigning tasks to workers randomly gives the best performance in both the MAE results and the RMSE results. However, as the number of selected samples increases, assigning new tasks in the first stage can greatly improve the performance in both the MAE results and the RMSE results. Compared with random selection on task and worker to the PMF learning model (i.e. *T[Rand]W[Rand]*), ActivePMFv1 can greatly improve both MAE and RMSE performance.

In Tables 8 and 9, we compare our proposed ActivePMFv2 with ActivePMFv1. The performance results of the three approaches are similar when the number of selected samples is small. However, as the number of selected samples increases, ActivePMFv2 outperforms ActivePMFv1 in both MAE and RMSE results. Compared with ActivePMFv1, the MAE results and the RMSE results of ActivePMFv2 are improved up to 29 % and 35 % respectively.

In Table 10, we compare the partial update part of our online-updating approach with the full retrain on ActivePMFv2 model learning. The prediction quality of ActivePMFv2 with partial update (threshold $t = 0.001$) approximates to that of ActivePMFv2 with full retrain, but the average runtime on model update per work done of ActivePMFv2 with partial update (threshold $t = 0.001$) is greatly reduced by 12 % compared with that of ActivePMFv2 with full retrain.

By using the batch update method, the average runtime on model update per work done can be further reduced. In Table 11, we compare the batch update part of our online-updating approach with the full retrain on ActivePMFv2 model learning. As batch size increases, both the MAE results and the RMSE results also increase, while the average runtime on model update per work done decreases significantly. For instance, compared with full retrain, when batch size is 500 with partial update (threshold $t = 0.001$), the average runtime on model update per work done is reduced by 99.6 % (decreases from 3.839 min to 0.017 min), but both the MAE results and the RMSE results increases by several times. On the other hand, compared with full retrain, when batch size is 10 with partial update (threshold $t = 0.001$), the average runtime on model update per work done is reduced by 82.4 % (decreases from 3.839 min to 0.675 min), while the MAE result is just increased by 22.4 % (increases from 0.0156 to 0.0191) and the RMSE result is only slightly

Table 6 MAE comparison among various active learning approaches in PMF (A smaller MAE means a better performance)

No. of selected samples	PMF with active learning approaches				
	T[N]W[Reli]+ T[MaxDiff]W[Reli] (ActivePMFv1)	T[MaxDiff]W[Reli]	T[MaxDiff]W[Rand]	T[Rand]W[Reli]	T[Rand]W[Rand]
1000	0.3658	0.4731	0.4728	0.4179	**0.3607**
2000	0.2778	0.4579	0.4592	0.3580	**0.2739**
3000	**0.1955**	0.4567	0.4567	0.3083	0.2340
4000	**0.1473**	0.4325	0.4371	0.2642	0.2090
5000	**0.1151**	0.4089	0.4122	0.2296	0.1866
6000	**0.0977**	0.2427	0.3887	0.1941	0.1677
7000	**0.0945**	0.1872	0.2646	0.1716	0.1435
8000	**0.0899**	0.1334	0.1908	0.1447	0.1122
9000	**0.0652**	0.0898	0.1214	0.0994	0.0728
10000	**0.0334**	0.0357	0.0583	0.0394	0.0406
11000	**0.0176**	0.0247	0.0312	0.0287	0.0388

Table 7 RMSE comparison among various active learning approaches in PMF (A smaller MAE means a better performance)

	PMF with active learning approaches				
No. of selected samples	T[N]W[Reli]+ T[MaxDiff]W[Reli] (ActivePMFv1)	T[MaxDiff]W[Reli]	T[MaxDiff]W[Rand]	T[Rand]W[Reli]	T[Rand]W[Rand]
1000	0.7101	0.8407	0.8396	0.7745	**0.6715**
2000	0.6097	0.8417	0.8417	0.7112	**0.5355**
3000	0.5110	0.8423	0.8420	0.6578	**0.4833**
4000	**0.4164**	0.8017	0.8089	0.6052	0.4533
5000	**0.3180**	0.7597	0.7655	0.5643	0.4247
6000	**0.2621**	0.6918	0.7245	0.5191	0.3978
7000	**0.2566**	0.6067	0.6150	0.4501	0.3605
8000	**0.2563**	0.5186	0.5381	0.3205	0.2991
9000	**0.2054**	0.3899	0.4844	0.2952	0.2210
10000	**0.1259**	0.1683	0.2139	0.1843	0.1524
11000	**0.0880**	0.1245	0.1357	0.1342	0.1396

Table 8 MAE comparison between ActivePMFv2 and ActivePMFv1 (A smaller MAE means a better performance)

No. of selected samples	T[N]W[Reli]+T[MaxDiff]W[N]+ T[MaxDiff]W[Reli] (ActivePMFv2)	T[N]W[Reli] + T[MaxDiff]W[Reli] (ActivePMFv1)
1000	0.3658	0.3658
2000	0.2778	0.2778
3000	0.1955	0.1955
4000	**0.1052**	0.1473
5000	**0.1001**	0.1151
6000	**0.0948**	0.0977
7000	**0.0897**	0.0945
8000	**0.0879**	0.0899
9000	0.0652	0.0652
10000	**0.0283**	0.0334
11000	**0.0156**	0.0176

Table 9 RMSE comparison between ActivePMFv2 and ActivePMFv1 (A smaller RMSE means a better performance)

No. of selected samples	T[N]W[Reli]+T[MaxDiff]W[N] + T[MaxDiff]W[Reli] (ActivePMFv2)	T[N]W[Reli] + T[MaxDiff]W[Reli] (ActivePMFv1)
1000	0.7101	0.7101
2000	0.6097	0.6097
3000	0.5110	0.5110
4000	**0.2710**	0.4164
5000	**0.2594**	0.3180
6000	**0.2513**	0.2621
7000	**0.2479**	0.2566
8000	**0.2494**	0.2563
9000	**0.2030**	0.2054
10000	**0.1116**	0.1259
11000	**0.0845**	0.0880

Table 10 Comparison on a Full-Retrain with Partial Update on Online-Updating Approach on ActivePMFv2 model learning (Feature k = 20; No of Work Done = 11,000; Batch = 1)

ActivePMFv2 model	MAE	RMSE	Avg Runtime per Work Done (min)
Full Retrain	0.0156	0.0845	3.839
Online-Updating (Partial = 0.001)	0.0156	0.0845	**3.374**

Table 11 Comparison on a Full-Retrain with Batch Update on Online-Updating Approach on ActivePMFv2 model learning (Feature k = 20; No of Work Done = 11,000)

ActivePMFv2 model	MAE	RMSE	Avg Runtime per Work Done (min)
Full Retrain	**0.0156**	**0.0845**	3.839
Online-Updating (Partial = 0.001; Batch = 1)	**0.0156**	**0.0845**	3.374
Online-Updating (Partial = 0.001; Batch = 10)	0.0191	0.0914	0.675
Online-Updating (Partial = 0.001; Batch = 50)	0.0313	0.1353	0.142
Online-Updating (Partial = 0.001; Batch = 100)	0.0515	0.2103	0.137
Online-Updating (Partial = 0.001; Batch = 150)	0.0768	0.2977	0.049
Online-Updating (Partial = 0.001; Batch = 200)	0.0513	0.2033	0.038
Online-Updating (Partial = 0.001; Batch = 500)	0.1022	0.3445	**0.017**

increased by 8.2 % (increases from 0.0845 to 0.0914). Therefore, by adjusting the batch size, the average runtime on model update per work done can be reduced significantly, but only very small performance degradation is resulted. In the cases shown in Table 11, batch size 10 is the best choice among all the listed choices.

Conclusion

In this paper, we have proposed Probabilistic Matrix Factorization with Active Learning (version 2), ActivePMFv2, on Task Recommendation framework, TaskRec, for quality assurance in crowdsourcing systems. It first randomly assigns new tasks to the most reliable worker in the task categories. Second, it actively selects the most uncertain task, and then request new workers to complete the task. Third, it actively selects the most uncertain task, and then request the most reliable workers to complete the task for retraining the classification model. Experimental results show that our ActivePMFv2 outperforms ActivePMFv1, where the MAE results and the RMSE results of ActivePMFv2 are improved up to 29 % and 35 % respectively. Previous work shows that ActivePMFv1 outperforms the PMF with other active learning approaches significantly.

Moreover, we have proposed online-updating on the task recommendation model to reduce the runtime of retraining the model. In a large-scale crowdsourcing system, it does not make sense to retrain the model from scratch whenever a worker of large profile completes a task, because the performance improvement by retraining the model in the case is tiny but the cost of retraing model is high. Moreover, when a large number of workers are working in the crowdsourcing system at the same period of time, the computational complexity is very high if the model is retrained after each worker completes a task. The larger the profile of a worker (or task) is, the less important is retraining its profile on each new work done. In case of the worker (or task) having large profile, our online-updating algorithm retrains the whole feature vector of the worker (or task) and keeps all other entries in the matrix fixed. Our online-updating algorithm runs batch update to reduce the running time of model update. Experiment results show that our online-updating algorithm is accurate in approximating to a full retrain of the learning model while the average runtime of model update for each work done is reduced by more than 80 % (decreases from a few minutes to several seconds).

Acknowledgements

This research was in part supported by grants from the National Grand Fundamental Research 973 Program of China (No. 2014CB340405), the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK 413212), and Microsoft Research Asia Regional Seed Fund in Big Data Research (Grant No. FY13-RES-SPONSOR-036).

Authors' contributions

All authors made substantial contributions to conception and design of the work. MC carried out experiments, data analysis and data interpretation. Besides, MC participated in drafting and revising the manuscript. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Received: 24 November 2015 Accepted: 14 September 2016
Published online: 01 October 2016

References

1. Howe J. The rise of crowdsourcing. Wired. 2006;14(6).
2. Howe J. Crowdsourcing: Why the Power of the Crowd Is Driving the Future of Business. New York: Crown Business; 2008.
3. Yuen MC, King I, Leung KS. A survey of crowdsourcing systems. In: SocialCom '11: Proceedings of The Third IEEE International Conference on Social Computing. Boston: IEEE Computer Society; 2011. p. 766–73.

4. Yuen MC, Chen LJ, King I. A survey of human computation systems. In: CSE '09: Proceedings of IEEE International Conference on Computational Science and Engineering. Vancouver: IEEE Computer Society; 2009. p. 723–8. doi:10.1109/CSE.2009.395.
5. Amazon Mechanical Turk. https://www.mturk.com/.
6. CrowdFlower. http://crowdflower.com/.
7. Taskcn. http://www.taskcn.com/.
8. TopCoder. http://www.topcoder.com/.
9. Allahbakhsh M, Benatallah B, Ignjatovic A, Motahari-Nezhad HR, Bertino E, Dustdar S. Quality control in crowdsourcing systems: Issues and directions. IEEE Internet Computing. 2013;17(2):76–81. doi:10.1109/MIC.2013.20.
10. Ipeirotis PG, Provost F, Wang J. Quality management on amazon mechanical turk. In: HCOMP '10: Proceedings of the ACM SIGKDD Workshop on Human Computation. New York: ACM; 2010. p. 64–7.
11. Karger DR, Oh S, Shah D. Iterative learning for reliable crowdsourcing systems. In: Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a Meeting Held 12–14 December 2011, Granada; 2011. p. 1953–1961. http://papers.nips.cc/paper/4396-iterative-learning-for-reliable-crowdsourcing-systems.
12. Salakhutdinov R, Mnih A. Probabilistic matrix factorization. In: NIPS '07: Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems. Taipei: Curran Associates, Inc.; 2007.
13. Rendle S, Schmidt-Thieme L. Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In: Proceedings of the 2008 ACM Conference on Recommender Systems. RecSys '08. New York: ACM; 2008. p. 251–8. doi:10.1145/1454008.1454047. http://doi.acm.org/10.1145/1454008.1454047.
14. Karimi R, Freudenthaler C, Nanopoulos A, Schmidt-Thieme L. Active learning for aspect model in recommender systems. In: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, Part of the IEEE Symposium Series on Computational Intelligence 2011, April 11–15, 2011, Paris; 2011. p. 162–7. doi:10.1109/CIDM.2011.5949431. http://dx.doi.org/10.1109/CIDM.2011.5949431.
15. Yuen MC, King I, Leung KS. Probabilistic matrix factorization with active learning for quality assurance in crowdsourcing systems. In: ICWI 2015: Proceedings of The IADIS International Conference WWW/Internet 2015. Greater Dublin; 2015.
16. Stewart O, Lubensky D, Huerta JM. Crowdsourcing participation inequality: a scout model for the enterprise domain. In: Proceedings of the ACM SIGKDD Workshop on Human Computation. HCOMP '10. New York: ACM; 2010. p. 30–3. doi:10.1145/1837885.1837895. http://doi.acm.org/10.1145/1837885.1837895.
17. Ross J, Irani L, Silberman MS, Zaldivar A, Tomlinson B. Who are the crowdworkers?: shifting demographics in mechanical turk. In: CHI EA '10: Proceedings of the 28th of the International Conference Extended Abstracts on Human Factors in Computing Systems. New York: ACM; 2010. p. 2863–872. doi:10.1145/1753846.1753873. http://doi.acm.org/10.1145/1753846.1753873.
18. Chilton LB, Horton JJ, Miller RC, Azenkot S. Task search in a human computation market. In: HCOMP '10: Proceedings of the ACM SIGKDD Workshop on Human Computation. New York: ACM; 2010. p. 1–9. doi:10.1145/1837885.1837889. http://doi.acm.org/10.1145/1837885.1837889.
19. Koren Y, Bell R, Volinsky C. Matrix factorization techniques for recommender systems. Computer. 2009;42(8):30–7. doi:10.1109/MC.2009.263.
20. Yang S, Ye M. Global Minima Analysis of Lee and Seung's NMF Algorithms. Neural Process Lett. 2013;38(1):29–51. doi:10.1007/s11063-012-9261-x.
21. Yang S, Yi Z. Convergence Analysis of Non-Negative Matrix Factorization for BSS Algorithm. Neural Process Lett. 2010;31(1):45–64. doi:10.1007/s11063-009-9126-0.
22. Salakhutdinov R, Mnih A. Probabilistic matrix factorization. In: Advances in Neural Information Processing Systems; 2008.
23. Zhou TC, Ma H, King I, Lyu MR. Tagrec: Leveraging tagging wisdom for recommendation. In: Proceedings of the 2009 International Conference on Computational Science and Engineering - Volume 04. Washington: IEEE Computer Society; 2009. p. 194–9. doi:10.1109/CSE.2009.75. http://dl.acm.org/citation.cfm?id=1632710.1633781.
24. Allahbakhsh M, Benatallah B, Ignjatovic A, Motahari-Nezhad HR, Bertino E, Dustdar S. Quality control in crowdsourcing systems: Issues and directions. IEEE Internet Computing. 2013;17(2):76–81. doi:10.1109/MIC.2013.20.
25. Organisciak P, Teevan J, Dumais ST, Miller RC, Kalai AT. A crowd of your own: Crowdsourcing for on-demand personalization. In: Proceedings of the Seconf AAAI Conference on Human Computation and Crowdsourcing, HCOMP 2014, November 2–4, 2014, Pittsburgh; 2014. http://www.aaai.org/ocs/index.php/HCOMP/HCOMP14/paper/view/8972.
26. Organisciak P, Teevan J, Dumais S, Miller RC, Kalai AT. Matching and grokking: Approaches to personalized crowdsourcing. In: Proceedings of the 24th International Conference on Artificial Intelligence. IJCAI'15. Pittsburgh: AAAI Press; 2015. p. 4296–302. http://dl.acm.org/citation.cfm?id=2832747.2832856.
27. Ambati V, Vogel S, Carbonell J. Towards task recommendation in micro-task markets. In: AAAI '11: Proceedings of The 25th AAAI Workshop in Human Computation. Pittsburgh: AAAI Publications; 2011.
28. Lin CH, Kamar E, Horvitz E. Signals in the silence: Models of implicit feedback in a recommendation system for crowdsourcing. In: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City; 2014. p. 908–15. http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8425.
29. Atlas L, Cohn D, Ladner R, El-Sharkawi MA, Marks II RJ. Training connectionist networks with queries and selective sampling. Adv Neural Inf Process Syst. 1990;2:566–573.
30. Lewis DD, Gale WA. A sequential algorithm for training text classifiers. In: SIGIR '94: Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Dublin: Springer; 1994. p. 3–12.
31. Scheffer T, Decomain C, Wrobel S. Active hidden markov models for information extraction. In: IDA '01: Proceedings of the 4th International Conference on Advances in Intelligent Data Analysis. Cascais: Springer; 2001. p. 309–18.
32. Culotta A, McCallum A. Reducing labeling effort for structured prediction tasks. In: AAAI '05: Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2. Pittsburgh: AAAI Press; 2005. p. 746–51.

33. Dagan I, Engelson SP. Committee-based sampling for training probabilistic classifiers. In: ICML '95: Proceedings of the Twelfth International Conference on Machine Learning. Tahoe City, California: Morgan Kaufmann; 1995. p. 150–7.
34. Seung HS, Opper M, Sompolinsky H. Query by committee. In: COLT '92: Proceedings of the Fifth Annual Workshop on Computational Learning Theory. Pittsburgh: ACM; 1992. p. 287–94.
35. Yan Y, Rosales R, Fung G, Dy JG. Active learning from crowds. In: ICML'11: Proceedings of the 28th International Conference on Machine Learning. Bellevue: Omnipress; 2011. p. 1161–1168.
36. Laws F, Scheible C, Schütze H. Active learning with amazon mechanical turk. In: EMNLP '11: Proceedings of the Conference on Empirical Methods in Natural Language Processing. Edinburgh: Association for Computational Linguistics; 2011. p. 1546–1556.
37. Fang M, Zhu X, Li B, Ding W, Wu X. Self-taught active learning from crowds. In: ICDM; 2012. p. 858–63.
38. Fang M, Zhu X. Active learning with uncertain labeling knowledge. Pattern Recogn Lett. 2014;43:98–108. doi:10.1016/j.patrec.2013.10.011.
39. Jung HJ, Lease M. Improving quality of crowdsourced labels via probabilistic matrix factorization. In: Human Computation Workshop at the Twenty-Sixth AAAI Conference on Artificial Intelligence; 2012.
40. Jung HJ. Quality assurance in crowdsourcing via matrix factorization based task routing. In: International Conference on World Wide Web 2014; 2014.
41. Yuen MC, King I, Leung KS. Taskrec: Probabilistic matrix factorization in task recommendation in crowdsourcing systems. In: ICONIP (2); 2012. p. 516–25.
42. Yuen MC, King I, Leung KS. Taskrec: A task recommendation framework in crowdsourcing systems. Neural Process Lett. 2015;41(2):223–38. doi:10.1007/s11063-014-9343-z.
43. Yuen MC, King I, Leung KS. Task matching in crowdsourcing. In: CPSCom '11: Proceedings of The 4th IEEE International Conference on Cyber, Physical and Social Computing. Boston: IEEE Computer Society; 2011. p. 409–12.
44. Yuen MC, King I, Leung KS. Task recommendation in crowdsourcing systems. In: KDD '12: Proceedings of ACM KDD 2012 Workshop on Data Mining and Knowledge Discovery with Crowdsourcing (CrowdKDD). New York: ACM; 2012.
45. Schnitzer S, Rensing C, Schmidt S, Borchert K, Hirth M, Tran-Gia P. Demands on Task Recommendation in Crowdsourcing Platforms - The Worker's Perspective. In: CrowdRec Workshop. Vienna; 2015.
46. Geiger D, Schader M. Personalized task recommendation in crowdsourcing information systems – current state of the art. Decision Support Systems. 2014;65:3–16. doi:10.1016/j.dss.2014.05.007. Crowdsourcing and Social Networks Analysis.
47. Aldhahri E, Shandilya V, Shiva SG. Towards an effective crowdsourcing recommendation system: A survey of the state-of-the-art. In: SOSE. Boston: IEEE; 2015. p. 372–7. http://dblp.uni-trier.de/db/conf/sose/sose2015.html#AldhahriSS15.
48. Dueck D, Frey BJ, Dueck D, Frey BJ. Probabilistic sparse matrix factorization. Technical report, University of Toronto. 2004.
49. NAACL 2010 Workshop. http://sites.google.com/site/amtworkshop2010/data-1.