Proceedings of the Eighth AAAI Conference on Human Computation and Crowdsourcing (HCOMP-20)

Batch Prioritization of Data Labeling Tasks for Training Classifiers

Masanari Kimura,[1] **Kei Wakabayashi, Atsuyuki Morishima**[2]
[1 2]University of Tsukuba
1-2, Kasuga, Tsukuba, Ibaraki, Japan
[1]mkimura@klis.tsukuba.ac.jp [2]{kwakaba, mori}@slis.tsukuba.ac.jp

Abstract

In a data labeling process for building machine learning, the choice of labeling data instances is known to have a significant impact on the performance of classifiers. So far, the study of active learning has addressed the issue of how to choose the subset by prioritizing the data instances based on the state of the current classifier. However, the active learning approach has two drawbacks that (i) require a training loop to update the priorities of labeling tasks and (ii) require us to choose a specific active learner while we do not know the optimal classification model. In this paper, we propose a new framework of priority-aware labeling system that allows a parallel task assignment to crowd workers without assuming a particular classifier, which is based on novel methods called "batch prioritization" and "label expansion". We conducted experiments with multiple datasets to examine the effectiveness of the approach and found that the proposed method improves the performance of the final classifiers more quickly than the active learning approach despite that the labeling tasks can be processed in a fully parallel manner.

Introduction

Data labeling is a fundamental process for building machine learning (ML) models. From the view of human computation systems (e.g., crowdsourcing), data labeling is a process that takes a set of data instances and produces the corresponding labels. Since this process requires human computations of which cost is not negligible, a label requester usually has to pick out a subset of data instances from a large data pool to put the request into a labeling system. To choose the subset, the requester needs to prioritize data instances for maximizing utility for the subsequent ML training. However, the prioritization to obtain a better training dataset is unclear in general (Lin et al. 2019).

So far, this issue has been addressed in the study of active learning (Yan et al. 2011; Yang et al. 2019). Typical active learning algorithms examine the data pool and choose data instances that are considered to be the most informative for updating model parameters of a ML model (Settles 2010). In fact, active learning considerably improves the performance of ML models when the budget available for labeling is limited. Figure 1 **(top)** shows the framework of active learning based labeling system. In this work, we suggest that the active learning approach has two fundamental problems.

(i) Active learning requires a repetitive update of the priorities, which restricts the parallelization of labeling work by the crowd. Moreover, in practice, we need a labeling platform (e.g., crowdsourcing service) that supports a dynamic control of the order of task display, which also brings a complexity of implementation.

(ii) Active learning requires us to choose a specific active learner at a moment we do not necessarily know what is the optimal ML model for the dataset. In fact, as we confirm later, the performance improvement is affected when we use an active learner that is different from the final ML model. We call this issue a *model mismatch effect*.

In this paper, we propose a new labeling framework that allows a parallel task assignment to crowd workers, which is effective for training classifiers without assuming a particular ML model. The proposed method comes with two key novel concepts; *batch prioritization* and *label expansion*, which addresses the issues (i) and (ii) respectively. Figure 1 **(bottom)** illustrates the entire proposed framework.

The batch prioritization, which addresses the issue (i), indicates a function that determines the priorities of the labeling tasks before the process starts based on an unsupervised analysis of the data pool. We present a batch prioritization algorithm that is based on a graph-based classification method. The proposed method analyzes the k-nearest neighbor graph of the instances in the data pool and determine the priorities of labeling tasks in an unsupervised manner.

However, the batch prioritization assigns priorities to optimize the performance of the graph-based classifier, which is not necessarily good for the final ML model (Baldridge and Osborne 2004). The model mismatch issue becomes a critical problem in this situation because there is a large gap between the criteria of unsupervised and supervised classifiers. For this reason, we claim that the model mismatch issue should be addressed jointly with the batch prioritization.

To this end, we integrate the label expansion function, which addresses the issue (ii), into the proposed labeling

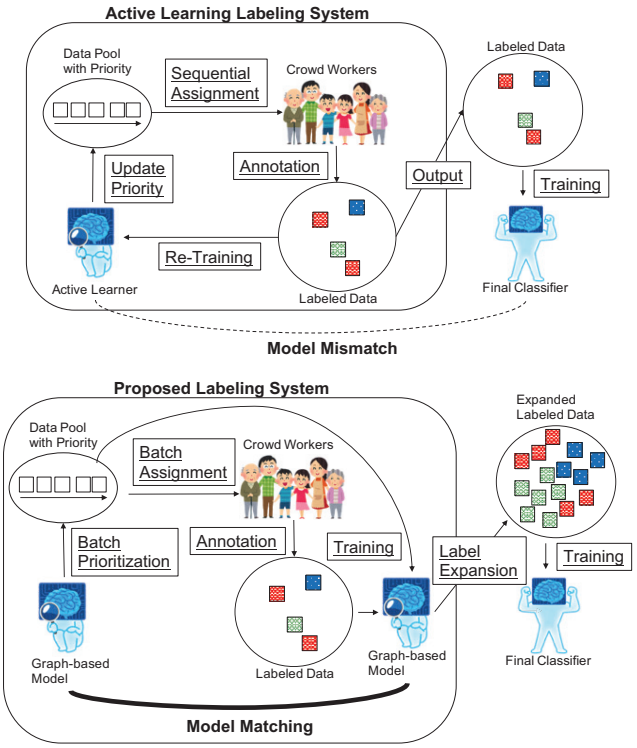

Figure 1: (top) A typical framework of labeling system based on an active learning loop. **(bottom)** The proposed labeling system based on batch prioritization that allows system to assign tasks to humans in a fully parallel manner.

framework to suppress the model mismatch effect. The label expansion is based on a co-training that trains two different ML models to help each other (Zhou, Zhan, and Yang 2007). In the proposed framework, the graph-based classifier behaves as a co-trainer that expands the labels given by human workers to help the final ML model. Since the graph-based classifier is what the batch prioritization assumes, there is no model mismatch at least in the label expansion phase.

The contribution of this paper is summarized as follows: **(a)** We present the concept of batch prioritization for data labeling tasks and propose a graph-based algorithm that determines the priorities in an unsupervised manner. **(b)** We point out the model mismatch issue in a priority-aware labeling framework. We propose a label expansion method that mitigates the model mismatch issue as a function on labeling systems. **(c)** We provide experimental results on multiple domains and show that the proposed framework produces a dataset that yields higher performance of the final classifier compared to the active learning approach despite that the labeling tasks can be processed in a fully parallel manner.

Graph-Based Batch Prioritization

Problem Statement

In this paper, we consider classification tasks where we have a set of data instances $X = \{x_i\}_{i=1}^{N}$ to be classified (*data pool*). We denote the label of data instance x_i by

Algorithm 1 Batch prioritization by coverage maximization

Require: $G = (X, E)$, Covering function f, Budget J
1: $H \leftarrow \phi$, $h(x) \leftarrow -\infty$ for all $x \in X$
2: **for** $j = J$ to 1 **do**
3: $\quad x \leftarrow \arg\max_{x} f(H \cup \{x\})\ (x \in X \backslash H)$
4: $\quad H \leftarrow H \cup x$
5: $\quad h(x) \leftarrow j$
6: **end for**
7: **return** h

$\psi(x_i) \in \mathcal{Y} \cup \phi$ where $\mathcal{Y}$ is a set of the possible classes. We use the symbol ϕ to represent where no label is assigned, i.e., $\psi(x_i) = \phi$ indicates x_i has not been labeled yet. When the labeling process starts, $\psi(x_i) = \phi$ for all $x_i \in X$.

There are one or more *human workers* (*annotators*) who classify the data instances correctly[1]. A *labeling task* or simply *task* refers to a request to the human workers for labeling a specific data instance. Let J be a *budget* that indicates the maximum number of tasks we are allowed to request. Since we assume $J \ll N$, we need to make a *priority* of each data instance $h(x_i)$. A higher $h(x_i)$ value indicates a higher priority. The human workers put the correct labels to data instances according to the order specified by the priorities. After the workers classify J data instances, classifiers are trained by using them as training data. We call these classifiers *final classifiers*, which are not assumed to be a specific ML model. We assume a *performance measure* of the final classifiers such as accuracy of a set of unseen data instances.

(Batch Prioritization Problem) Given a set of data instances X and a budget J, *batch prioritization* is the problem to make a priority function h such that maximizes the performance of final classifiers that are trained by using the labels produced by the labeling process. We emphasize the difference against the active learning problem that (i) updates the priorities repeatedly during the labeling process and (ii) aims at improving a single specific classifier.

Batch Prioritization as Coverage Maximization

In this section, we present the proposed batch prioritization method based on an unsupervised analysis of data distribution. The principle of the method is the maximization of *coverage* on the data pool, which is a notion that measures the representativeness of the selected set of data instances.

We define a function $\alpha : X \rightarrow 2^X$ that maps a data instance $x \in X$ to a set of data instances that are *represented* by x (the concrete definitions are presented later). Roughly speaking, we call x *represent* x' if the label of x' can be confidently estimated by knowing the label of x. On the basis of the definition of α, we define a coverage function $f : 2^X \rightarrow \mathbb{N}$ as $f(H) = \left|\bigcup_{x \in H} \alpha(x)\right|$. The proposed batch prioritization method attempts to select J data instances that maximize the coverage $f(H)$ where $H \subset X$ is the selected data instances that will have the top priorities. Algorithm 1

[1]In practice, human workers are not perfect. The effect of human errors can be mitigated by applying a quality management method based on redundant labeling (Yan et al. 2011).

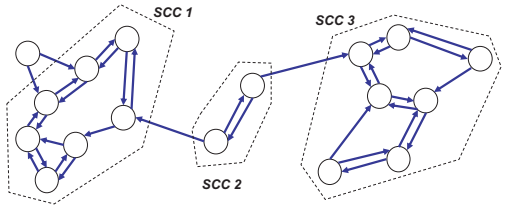

Figure 2: K-nearest neighbor graph (k=2) and strongly connected components (SCC) in the graph.

shows the procedure of the batch prioritization algorithm based on the greedy submodular optimization. The simple greedy method guarantees a $1 - 1/e \approx 0.63$ approximate solution and the actual result is likely to be much better than this (Nemhauser, Wolsey, and Fisher 1978).

In the following subsections, we present two definition of $\alpha(x)$ to implement this strategy. Both definitions are based on a k-nearest neighbor similarity graph of the data pool.

k-Nearest Neighbor Similarity Graph

Let $\mathcal{X}$ be the domain of data instances ($x_i \in \mathcal{X}$). We assume that a similarity function $sim : \mathcal{X}^2 \to [0, 1]$ can be defined on $\mathcal{X}$. For instance, we can use the set of d-dimensional vector $\mathbb{R}^d$ as $\mathcal{X}$ and the cosine similarity as sim.

Given X, sim, and $k \geq 1$, we construct k-nearest neighbor similarity graph $G = (X, E)$. G is a weighted directed graph that is defined by the set of nodes corresponding to the data instances X, the set of edges E. Let $\mathbf{N}_k(x_i) \subset X \backslash \{x_i\}$ be the set of the k most similar data of x_i with respect to the similarity function sim. The set of edges is defined as;

$$E = \{(u, v) | u \in X, v \in \mathbf{N}_k(u)\}. \quad (1)$$

Coverage Based on Strongly Connected Component The first way to define $\alpha(x)$ is based on strongly connected component (SCC) decomposition. The main idea of this definition is that, if we know the label of x, the label of the data instances in the SCC that contains x is likely to have the same label (Figure 2). An algorithm of Kosaraju (Sharir 1981) is known as an efficient SCC decomposition algorithm on a directed graph, which has a time complexity of $O(E+V)$. When we denote the set of nodes in the SCC containing x by $SCC(x)$, the definition of the coverage function is $f_{SCC}(H) = \left|\bigcup_{x \in H} SCC(x)\right|$. The maximization of $f_{SCC}(H)$ regarding H is easily computed: First, we sort the obtained SCCs in order of the size and take the top J SCCs. Then, we randomly choose a node from each SCC.

Coverage Based on Simple Cycles Enumeration The definition based on SCC decomposition can be ineffective when there are very large SCCs because we assume that only one node covers the whole component. We propose another definition involving enumeration of simple cycles, which is a cycle that does not intersect with itself. Jonson's algorithm (Johnson 1975) is a well-known enumeration algorithm for simple cycles on a directed graph with a time complexity $O((V+E)(C+1))$ where C is the number of simple cycles on the graph. When we denote the set of nodes in the simple cycles starting from node v by $cyc(v)$, we can define the coverage function as $f_{cycle}(H) = \left|\bigcup_{v \in H} cyc(v)\right|$.

Minimizing Model Mismatch Effect Using Co-Training Framework

As Figure 1 **(bottom)** shows, the proposed framework adopts a label expansion process to mitigate the model mismatching effect. Label expansion refers to a process that assigns labels to a subset of data instances. The label expansion process put a label to a data instance if the label can be confidently estimated from labels assigned by human workers. Otherwise, the process should leave the instance unlabeled. The label expander is supposed to follow the same principle as the batch prioritization, i.e., the proximity on the k-nearest neighbor similarity graph. We propose a graph-based label expansion method called *RD-Walks expander*.

Given the set of data instances $X = \{x_i\}_{i=1}^N$ and the map ψ where $\psi(x_i) \in \mathcal{Y} \cup \{\phi\}$ that maps x_i to the label that is annotated by human worker, the RD-Walks expander generates the expanded label map $\tilde{\psi}$ where also $\tilde{\psi}(x_i) \in \mathcal{Y} \cup \{\phi\}$. In the proposed framework, we provide the expanded labeled dataset $\{(x, \tilde{\psi}(x)) | x \in X\}$ for training final classifiers. In terms of the k-nearest neighbor graph, the set of nodes X contains both of the labeled and unlabeled data instances, which are denoted by $L = \{x_i \in X | \psi(x_i) \in \mathcal{Y}\}$ and $U = \{x_i \in X | \psi(x_i) = \phi\}$, respectively. We also denote the set of nodes that have the label y by $L_y = \{x_i \in L | \psi(x_i) = y\}$.

D-Walks Classifier

The RD-Walks expander is derived from a similar idea of *discriminative random walks (D-Walks) classifier*, which is a graph-based semi-supervised classification algorithm (Callut et al. 2008). The D-Walks classifier estimates the label of unlabeled node $v \in U$ based on *betweenness* in the graph.

For each class y, *D-Walks* D^y is defined as a set of walks such that start from a node in L_y and end with a node in L_y. We assign the probability $p(d)$ of a walk $d \in D^y$ as the probability of a random walk $(Z_1, \ldots, Z_n)$, which is calculated as the product of transition probabilities in the walk. For the k-nearest neighbor similarity graph, we define the transition probability as the normalized similarities $P[Z_{t+1} = u | Z_t = v] = \frac{sim(v,u)}{\sum_{v' \in X} sim(v,v')}$. Let $n_d(v)$ be the number of times that a node v is visited in a walk d. By using these notions, we can define the *betweenness* $B(v, y)$ of v on class y as the expected value of $n_d(v)$ w.r.t. $p(d | d \in D^y)$. The D-Walks classifier classifies the unlabeled node $v \in U$ into the class $\hat{\phi}(v) = \arg\max_y B(v, y)$.

RD-Walks Expander

The D-Walks classifier attempt to classify all the data instances discriminatively. Unlike the D-Walks classifier, the RD-Walks expander assigns a label to a data instance only when the instance is close enough to labeled instances.

We impose two transition restrictions on the D-Walks not to go out to an uncertain region. First, the RD-Walks R^y is not allowed to visit any other labeled node.

$R^y = \{(v_1, \ldots, v_n) | v_1, v_n \in L_y, v_t \in U, 1 < t < n\}$.

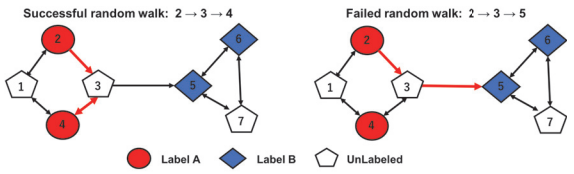

Figure 3: Property of RD-Walks. **(left)** Walk $2 \rightarrow 3 \rightarrow 4$ succeeds because the start node and the end node have the same label A. **(right)** Walk $2 \rightarrow 3 \rightarrow 5$ fails because the walk reaches the node having a different label to the start node.

Second, the walks should satisfy that the sum of the distance is lower than the maximum distance parameter p and the length is at most q. We use $W_{vu} = 1 - sim(v, u)$ as the distance from v to u. Therefore, the RD-Walks expander considers the subset of R^y denoted by $R^y_{\leq q,p}$ as follows:

$$R^y_{\leq q,p} = \{(v_1, \ldots, v_n) \in R^y | n \leq q, \sum_{t=1}^{n-1} W_{v_t v_{t+1}} \leq p\}.$$

To enumerate all the walks in $R^y_{\leq q,p}$, we execute the depth first search that starts from each labeled node $v \in L$. Figure 3 shows examples of the search of RD-Walks. In the successful random walk, the start node and the end node are in the same class, whereas the failed random walk reaches the node of the different label. The betweenness of node v on class y for RD-Walks expander is defined as follows:

$$B'_{q,p}(v, y) = E[n_d(v)\mathbf{1}_{R^y_{\leq q,p}}(d)] = \sum_{d \in R^y_{\leq q,p}} p(d) n_d(v) \quad (2)$$

Because of the restriction imposed on the RD-Walks, there will be a substantial number of the unlabeled nodes that are visited by no RD-Walk. By definition, the betweenness $B'_{q,p}(v, y)$ for such node v becomes zero for all class y. The expanded label is assigned to nodes that are visited by at least one RD-Walk, formally as follows:

$$\tilde{\psi}(v) = \begin{cases} \phi & (\max_{y \in \mathcal{Y}} B'_{q,p}(v, y) = 0) \\ \arg\max_{y \in Y} B'_{q,p}(v, y) & (otherwise) \end{cases} \quad (3)$$

If $\tilde{\psi}(v) = \phi$, i.e., the node v is not passed through by any RD-Walk, v remains at the unlabeled node. We denote the set of expanded labeled instances by $\tilde{L} = \{x | \tilde{\psi}(x) \neq \phi\}$. We can increase the amount of the expanded labeled instances by performing the RD-Walks expansion multiple times. Therefore, the number of iterations M is also the parameter of the method. Let $\tilde{L}^{(m)}$ be the set of expanded labeled instances obtained by the m-th execution of the RD-Walks expansion. The $(m+1)$-th RD-Walks expansion is performed as if $\tilde{L}^{(m)}$ is the set of annotated instances. As M increases, the number of instances to be labeled increases, but the accuracy of assigned labels may decrease. Algorithm 2 shows the procedure of RD-Walks expansion. $RD\text{-}Walks(G, \tilde{L}, q, p)$ is a function that carries out the depth first search and returns $R^y_{\leq q,p}$ for all y.

Algorithm 2 RD-Walks Expansion

Require: graph G = (X, E, W), annotated label map ψ, number of iteration M, max step q, max length p

$\tilde{L}^{(0)} = \{x | \psi(x) \neq \phi\}$
for $m = 1$ to $|M|$ **do**
  $R_{\leq q,p} = RD\text{-}Walks(G, \tilde{L}^{(m-1)}, q, p)$
  for $u \in X \backslash \tilde{L}$ **do**
    Compute $B'_{q,p}(u, y)$ by Eq. (2) for each $y \in \mathcal{Y}$
    Compute $\tilde{\psi}(u)$ by Eq. (3)
  end for
  $\tilde{L}^{(m)} = \{x | \tilde{\psi}(x) \neq \phi\}$
end for
return $\tilde{\psi}$

Experiments

Settings

We benchmark our model on three datasets of a UCI Machine-Learning Repository[2], **Letter**, **Mice Protein**, and **Digits**. The domains of the data instances are d-dimensional vector $\mathbb{R}^d$, and we adopt the cosine similarity as the similarity function sim. The number of data instances $|X|$, the number of classes $|\mathcal{Y}|$, and the dimension of the feature vector d of each dataset is: (20,000, 26, 16) in **Letter**, (1,080, 8, 77) in **Mice Protein**, and (5,620, 10, 64) in **Digits**. We use 70% of the dataset as the data pool and 30% for the test data. The performance of the final classifiers is measured by the classification accuracy on the test data.

Methods

We compare multiple combinations of prioritization methods, label expansion, and final classifiers. For prioritization, we choose J data instances from the data pool by using one of the following prioritization algorithms and receive the true labels of them: **(Random)** J data instances are chosen randomly. **(LC (MLP)*)** Active learning based on least confident using multi-layer perceptron as the active learner. **(LC (LR)*)** Active learning based on least confident using logistic regression as the active learner. **(Cycle)** Batch prioritization using the covering function f_{cycle}. **(SCC)** Batch prioritization using the covering function f_{SCC}. The active learning methods (marked as *) update the priorities for each time we obtain a new label. Although active learning does not meet the requirement of the batch prioritization scenario, we measured the performance for the comparison purpose.

For label expansion, we compared two options: **(None)** The obtained label is provided to final classifiers as it is. **(RDWalks)** RD-Walks expander is applied to provide the expanded labels $\tilde{\psi}$ to final classifiers.

Finally, we train one of the following final classifier: **(MLP)** Multi-layer perceptron. **(LR)** Logistic regression.

We compare the combination of these options. For example, we denote by "SCC + RDWalks + MLP" the method that uses SCC as the prioritization method, RD-Walks as the label expansion, and MLP as the final classifier. Note that

[2]http://archive.ics.uci.edu/ml/index.php

Table 1: Test accuracy of the final classifiers when the budget $J = 100$. The labels of the methods denote (prioritization method) + (label expansion method) + (final classifier).

	Mice Protein	Digits	Letter
Random + None + MLP	0.6532	0.8722	0.3002
Random + None + LR	0.6430	0.8704	0.2980
LC (MLP) + None + MLP	0.7566	0.9409	0.2778
LC (LR) + None + LR	0.7222	0.9216	0.3184
LC (LR) + None + MLP	0.7179	0.9365	0.2420
LC (MLP) + None + LR	0.7152	0.9155	0.4009
Random + RDWalks + MLP	0.8081	0.9004	0.4303
Random + RDWalks + LR	0.7832	0.9011	0.4378
LC (MLP) + RDWalks + MLP	0.8111	0.9653	0.3243
LC (LR) + RDWalks + MLP	0.8068	0.9524	0.2507
SCC + RDWalks + MLP	**0. 9318**	**0.9662**	**0.6411**
Cycle + RDWalks + MLP	0.9204	0.9568	0.5829
SCC + None + MLP	0.9284	0.8385	0.4663
Cycle + None + MLP	0.9208	0.8088	0.4881

"LC(MLP) + None + LR" and "LC(LR) + None + MLP" are model mismatch situation that uses inconsistent pair of active learner and final classifier. The parameters of the proposed method is set as $k = 3$, $p = 0.5$, $q = 5$, and $M = 5$.

Results

Table 1 shows the test accuracy of the final classifiers when the budget is $J = 100$. The result shows that the methods using proposed batch prioritization (SCC and Cycle) tend to achieve better performance compared with the active learning methods despite that the prioritization does not need to be updated. The performance of the active learning methods is surprisingly lower than the batch prioritization, particularly on the Letter and Mice Protein dataset, even though they update the priorities dynamically. This can be attributed to the fact that active learning relies on the output of the current classifier that can be poor performance when the amount of training instances is insufficient. The batch prioritization is rather stable when the budget is not sufficiently large.

Regarding the model mismatching effect, in active learning methods, we confirmed that the performance decreases when the models of the active learner and the final classifier are different (i.e., the models are mismatched) in most cases (only exception is "LC(MLP) + None + LR" against "LC(LR) + None + LR" in Letter dataset). The model mismatching effect is observed also in the batch prioritization methods. In the Digits dataset, the performance of "SCC + None + MLP" and "Cycle + None + MLP" is worse than methods with other prioritization methods. This result implies the methods suffer the model mismatch effect since the principles of the graph-based prioritization model and the algebraic classifiers are considerably different. However, the batch prioritization methods gain a significant performance improvement by applying the RD-Walks expander ("SCC + RDWalks + MLP" and "Cycle + RDWalks + MLP"), so that the proposed methods overtake the active learning methods. The RD-Walks expander improves the performance in any case, but it is especially effective for the methods with the batch prioritization. This result implies the RD-Walks successfully mitigate the model mismatch effect caused by the proposed batch prioritization.

Conclusion

In this paper, we proposed a new framework of priority-aware labeling system that allows a parallel task assignment to crowd workers, which does not assume a particular classifier. The proposed method is also practical because it can be applied just as a preprocessing and postprocessing of ordinary crowdsourced labeling tasks. Additionally, the proposed method is robust against unknown final classifiers by addressing the model mismatch effect. This property is advantageous when we collaborate with many different ML models (Kobayashi, Wakabayashi, and Morishima 2020).

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 19K20333, 16H02904 and JST CREST Grant Number JPMJCR16E3 including AIP challenge program.

References

Baldridge, J., and Osborne, M. 2004. Active learning and the total cost of annotation. In *Proc. EMNLP*, 9–16.

Callut, J.; Françoisse, K.; Saerens, M.; and Dupont, P. 2008. Semi-supervised classification from discriminative random walks. *Machine learning and knowledge discovery in databases* 162–177.

Johnson, D. B. 1975. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing* 4(1):77–84.

Kobayashi, M.; Wakabayashi, K.; and Morishima, A. 2020. Quality-aware dynamic task assignment in Human+AI crowd. In *Companion Proc. Web Conference*, 118–119.

Lin, B. Y.; Lee, D.-H.; Xu, F. F.; Lan, O.; and Ren, X. 2019. AlpacaTag: An active learning-based crowd annotation framework for sequence tagging. In *Proc. ACL: System Demonstrations*, 58–63.

Nemhauser, G. L.; Wolsey, L. A.; and Fisher, M. L. 1978. An analysis of approximations for maximizing submodular set functions – i. *Mathematical Programming* 14:265–294.

Settles, B. 2010. Active learning literature survey. Technical report, University of Wisconsin, Madison.

Sharir, M. 1981. A strong-connectivity algorithm and its applications in data flow analysis. *Computers & Mathematics with Applications* 7(1):67–72.

Yan, Y.; Rosales, R.; Fung, G.; and Dy, J. G. 2011. Active learning from crowds. In *Proc. ICML*, 1161–1168.

Yang, J.; Smirnova, A.; Yang, D.; Demartini, G.; Lu, Y.; and Cudre-Mauroux, P. 2019. Scalpel-CD: Leveraging crowdsourcing and deep probabilistic modeling for debugging noisy training data. In *Proc. WWW*, 2158–2168.

Zhou, Z.-H.; Zhan, D.-C.; and Yang, Q. 2007. Semi-supervised learning with very few labeled training examples. In *Proc. AAAI*, 675–680.