# Learning Better while Sending Less: Communication-Efficient Online Semi-Supervised Learning in Client-Server Settings

Han Xiao*† , Shou-De Lin‡, Mi-Yen Yeh§, Phillip B. Gibbons¶ and Claudia Eckert*
*Technische Universität München, 85748 Garching, Germany. xiaoh@in.tum.de
†Zalando SE, 10178 Berlin, Germany. han.xiao@zalando.de
‡National Taiwan University, Taipei 110, Taiwan. sdlin@csie.ntu.edu.tw
§Academia Sinica, Taipei 115, Taiwan. miyen@iis.sinica.edu.tw
¶Intel Labs, Pittsburgh, PA 15213 USA. phillip.b.gibbons@intel.com

***Abstract*—We consider a novel distributed learning problem: A server receives potentially unlimited data from clients in a sequential manner, but only a small initial fraction of these data are labeled. Because communication bandwidth is expensive, each client is limited to sending the server only a small (high-priority) fraction of the unlabeled data it generates, and the server is limited in the amount of prioritization hints it sends back to the client. The goal is for the server to learn a good model of all the client data from the labeled and unlabeled data it receives. This setting is frequently encountered in real-world applications and has the characteristics of online, semi-supervised, and active learning. However, previous approaches are not designed for the client-server setting and do not hold the promise of reducing communication costs.**

**We present a novel framework for solving this learning problem in an effective and communication-efficient manner. On the server side, our solution combines two diverse learners working collaboratively, yet in distinct roles, on the partially labeled data stream. A compact, online graph-based semi-supervised learner is used to predict labels for the unlabeled data arriving from the clients. Samples from this model are used as ongoing training for a linear classifier. On the client side, our solution prioritizes data based on an active-learning metric that favors instances that are close to the classifier's decision hyperplane and yet far from each other. To reduce communication, the server sends the classifier's weight-vector to the client only periodically. Experimental results on real-world data sets show that this particular combination of techniques outperforms other approaches, and in particular, often outperforms (communication expensive) approaches that send all the data to the server.**

***Keywords*—big data; semi-supervised learning; online learning; distributed system;**

## I. Introduction

Distributed data acquisition is at the heart of the big data explosion. Smartphones, surveillance videos, wearable sensors, and a variety of smart devices (Internet of Things) generate data at geographically distributed points, and the goal is to learn valuable insights from these massive data streams. This paper considers such a setting where a set of distributed clients each generate an ongoing stream of data and a server seeks to learn a model of the data. We impose two practical limitations on the setting. First, because of the costs of having humans label large quantities of data, we assume that only a small fraction of the data are labeled. In particular, we focus on a setting where only the first, e.g., 2% of the training data are labeled. Second, because communication bandwidth is often expensive and battery-draining (e.g., a mobile device on a cellular network), we seek communication-efficient solutions such that each client is limited to sending to the server only a small fraction of the unlabeled data it generates, and limited in how much information it receives from the server.

As a motivating example, consider an intelligent traffic management system comprised of a set of surveillance cameras and a server. The server analyzes images from cameras and provides applications such as helmet violation, high-occupancy vehicle detection, and wrong-way vehicle alarms. To develop such a system, the model on the server needs to be configured by teaching it baseline images. Traditionally, it requires each camera to constantly upload images, and human annotators to manually label those uploads on the server. In practice, however, the network bandwidth is restricted and the labeling effort is limited. Therefore, a workable solution would be training an initial model with limited labels on the server, and selectively transmitting only the most informative images from each camera.

As another example, consider wearable devices (e.g., smartwatches) that measure sensory data, which is increasing dramatically both temporally and in fidelity. However, the device does not offer heavy computing power and may just serve as a front end for a remote system. To utilize the sensory data for intelligent tasks (e.g., recognizing human activities), the collected data on the device need to be transmitted wirelessly to a more powerful device (e.g., a smartphone or laptop). However, due to the bandwidth and battery limitations, it is unrealistic for the device to transmit every measurement. Often, the connection is only established at set intervals or manually by users, at which time only a selective subset of the measurements may be transmitted.

An elegant solution to these problems will face many challenges. First, the amount of data generated by clients can be huge, and even potentially unlimited. As a result, the vast majority of data on the server are unlabeled. Typically, it is not sufficient to train a model with a good generalization ability based merely on limited labeled data. Second, when

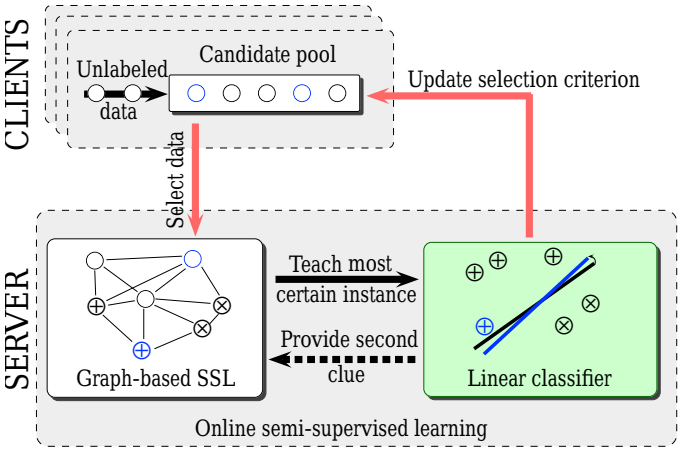


Fig. 1. An illustration of the proposed framework. The server contains two learners: a graph-based semi-supervised model and a linear classifier. They collaborate together to learn from a partially labeled data stream. At any point in time, the linear classifier can be used as a standalone component for predicting labels for new test data. The communication flow between each client and the server is represented by red arrows.

the volume and velocity of data is high, it is very costly and impossible to store all data either on clients or the server. Thus, traditional approaches that first store data and then train on a static collection are not appropriate in this case. Third, transmitting massive data on the network is discouraged in practice, especially when the network bandwidth is restricted or the communication cost is expensive (e.g., on a cellular network). It may also be mis-classified as a *denial-of-service attack*, and dropped/blocked.

At first sight, this learning problem seems to share some characteristics of online, semi-supervised, and active learning, which have been extensively studied in the machine learning community. However, it should be noted that our setting differs from these traditional learning settings and may require evolutionary changes to existing algorithms. Unlike online learning problems where all training data are assumed to be labeled, there is only a limited amount of labeled data in our setting. It also differs from typical semi-supervised learning where all labeled and unlabeled data is available ahead of time. Moreover, it differs from standard active learning in that there is no oracle available for providing feedback. Although both settings involve selective sampling, their intentions are different: active learning aims to save labeling efforts, whereas we attempt to reduce the bandwidth consumption between the server and clients (while also keeping the labeling effort to only a small fraction of the data). By considering online, semi-supervised, and active learning jointly, our goal is to develop a modular framework for learning from a remote partially labeled data stream while reducing the bandwidth consumption.

We present a novel framework for solving this learning problem in an effective and communication-efficient manner (see Figure 1). On the server side, our solution combines two diverse learners working collaboratively, yet in distinct roles, on the partially labeled data stream. A compact, online graph-based semi-supervised learner is used to predict labels for the unlabeled data arriving from the clients. Specifically, we adapt the Harmonic Solution learner to online use via an incremental $k$-center clustering approach that maintains the graph structure solely on a set of $k$ centroid nodes. Random samples are then repeatedly drawn from the model according to the confidence of its prediction, and used to train a second learner on the server, a linear classifier (specifically, a soft confidence-weighted classifier). The second learner updates its hypothesis based on these samples and their predicted labels. We show how these two learners can be combined in an optimization problem. On the client side, our solution prioritizes data based on an active-learning metric that favors instances that are more uncertain (i.e., close to the classifier's decision hyperplane) and yet far from each other (as measured by covariance). To reduce communication, the server sends the classifier's weight-vector to the client only periodically. At any point in time, the classifier can be used as a standalone model for predicting labels for new test data.

The main contributions of the paper are:

- We introduce a novel learning setting motivated by many big data applications, and present a general framework that surmounts the challenges inherent in this setting. The proposed framework is modular in design, flexible, and can be practically incorporated into a variety of useful systems.
- We devise new algorithms that are well-suited to providing high classification accuracy with reduced communication and labeling costs.
- Our experimental results on real-world data sets show that this particular combination of techniques outperforms other approaches, and in particular, often outperforms (communication expensive) approaches that send *all* the data to the server.

## II. Related Work

Online learning, semi-supervised learning and active learning are three different problem settings, which have been studied both separately and jointly. Perhaps the earliest exploration in combining semi-supervised learning with active learning is by McCallum et al. [1], where they combined an expectation-maximization algorithm with an active learning algorithm. Recently, many extensions of semi-supervised methods (e.g., S3VM [2], harmonic solution [3] and co-training [4]) to the active learning setting have been proposed (e.g., [5], [6], [7], [8]). In practice, active semi-supervised learning has a wide range of applications, from spoken language understanding [9] to document clustering [10] to content-based image retrieval [11], [12], [13]. Unfortunately, these methods do not meet the requirements of our problem setting, in which data items arrive in an online fashion, not in batch.

Another line of research is combining online learning with semi-supervised learning, which is extremely useful for adaptive systems with partially labeled input. Most of the algorithms in this line rely on indirect forms of feedback, such as a model's own prediction and the structure of data, to incrementally improve themselves. Grabner et al. [14] used a heuristic method to greedily label unlabeled examples in an object tracking application. Goldberg et al. [15] extended the online SVM [16] to the semi-supervised setting by adding a regularization term to the objective function of SVM. Valko et al. [17] extended the graph-based semi-supervised learning method [3] to the online setting, by computing the harmonic

solution on an approximate similarity graph in an incremental fashion. In our setting, this family of methods can be adapted for the server's use. However, it does not reduce communication costs because no selection is performed prior to transmission to the server.

The intention of online active learning was to extend the traditional active learning from the *pool-based* setting to the *stream-based* setting [18]. Zhu et al. [19] introduced a minimal variance principle to guide instance selection from a data stream. Bifet et al. [20] presented a weighted ensemble classifier and cluster model to handle large data stream volumes. Chu et al. [21] designed optimal instrumental distributions for allowing unbiased sampling in data streams. However, such methods are not applicable to our setting, as they cannot learn from unlabeled instances.

Finally, the idea of integrating online learning, semi-supervised learning, and active learning into one framework can be traced back to Shen et al. [22]. They extended the self-organizing incremental neural network [23] with semi-supervised learning and active learning. On each round, the algorithm selects some "teacher" nodes from each cluster and uses them to label all unlabeled nodes in the corresponding cluster. Later, Goldberg et al. [24] provided a Bayesian model for this learning setting. The model maintains a posterior distribution of weights through particle filtering and sequential Monte Carlo techniques. Instances that are highly disagreed according to the current particles are queried for labeling.

Unlike these prior works [22], [24], which intended to reduce the labeling effort for adaptive systems, our goal is to reduce the communication and labeling costs in a distributed client-server system. Moreover, the following three obstacles limit the possibility of adapting previous methods to our problem setting. First, the prior methods are not applicable to the client-server model, where the concerns of client and server must be well-separated. Most previous methods are developed in a bottom-up manner, by gradually extending the availability of original supervised learning methods to give rise to more complex settings. Thus, they are not *modular* in design. For example, Shen et al. [22] used the self-organizing incremental neural network [23] as the "seed" model. Their active learning and semi-supervised learning extensions work exclusively with the seed model, making it difficult to isolate each component. In a distributed setting, it is important to elucidate each subsystem for addressing a separate concern, as they may be deployed in different physical locations with different configurations.

Second, prior methods are not communication-efficient. More precisely, there is no efficient way to transmit the selection criterion to the clients. For example, Zhu et al. [6] selected instances based on their estimated risk on a graph, which would require each client to maintain a graph locally. Similar difficulty can be found in Goldberg et al. [24], where the uncertainty score is computed based on a set of particles (parameterized by a set of vectors), thereby requiring each client to maintain a set of vectors. High communication costs are incurred in keeping a client's set up-to-date.

Third, prior methods are computationally demanding. For example, Goldberg et al. [24] used a sequential Monte Carlo technique to update the model, requiring a number of iterations for learning a new datum. In the distributed setting, algorithms on both client and server should be lightweight and avoid time-consuming computations. This is because clients usually have few resources other than essential input and output functions. The server, though, offers more resources, and must respond agilely so that the new selection criterion can be quickly generated and distributed without forcing clients to wait.

## III. Preliminaries

Let us denote by $\mathcal{X}$ an instance domain and by $\mathcal{Y}$ a set of labels. Let $\mathcal{H}$ be a hypothesis class, where each $h \in \mathcal{H}$ is a mapping from $\mathcal{X}$ to $\mathcal{Y}$. In this paper, we concentrate on the confidence-rated binary classification problem, where $\mathcal{H}$ is the class of linear separators. In this case, $\mathcal{X}$ is a subset of the Euclidean space $\mathbb{R}^d$, $\mathcal{Y} = \{+1, -1\}$, and each hypothesis in $\mathcal{H}$ is a linear function parametrized by a weight vector $\mathbf{w} \in \mathbb{R}^d$. For each $\mathbf{x} \in \mathcal{X}$, define $h(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}$. In practice, we can handle a bias term by adding a dummy feature to all $\mathbf{x}$ and set $d = d + 1$. We interpret $\operatorname{sign}(h(\mathbf{x}))$ as the actual binary label predicted by $h$, and $|h(\mathbf{x})|$ is a degree of confidence in this prediction. The quality of a prediction is measured by a loss function $\ell(h; (\mathbf{x}, y))$, which represents the penalty of predicting $\operatorname{sign}(h(\mathbf{x}))$ when the correct label is $y \in \mathcal{Y}$. Two common choices of loss function are *zero-one loss* and *hinge loss*.

For the sake of simplicity, we will present the techniques in this paper assuming there is only a single client. The framework can be readily generalized to multiple clients, as discussed in Section VIII. Denote the set of unlabeled instances on the client by $V = \{\mathbf{x}_t\}_{t=1}^{v}$, where each $\mathbf{x}_t \in \mathcal{X}$. The client selects instances for uploading. On the server side, we have a small set of labeled data $L = \{(\mathbf{x}_t, y_t)\}_{t=1}^{l}$ only at the beginning, followed with a set of unlabeled data $U = \{\mathbf{x}_t\}_{t=l+1}^{n}$ uploaded from the client. We assume the server receives incoming data one-by-one. The total number of instances received by the server is $n$, and in our setting $l \ll n$, and $n \ll v$. Starting from an initial hypothesis $h_0$, the server incrementally constructs a sequence of hypotheses $h_1, h_2 \ldots, h_n$ according to $L$ and $U$. Ultimately, the goal of the server is to find a hypothesis that will exhibit high classification accuracy (e.g., under zero-one loss) on some unseen test set.

## IV. General Framework

We present a general framework for communication-efficient online semi-supervised learning in the client-server setting. The framework will be described in a way that the modules can be easily understood in isolation, and changes or extensions to functionality would be easily localized. Specifically, we start in this section with a big picture by describing the philosophy behind the system design, and a high-level overview of the framework. Later, Section V and Section VI will drill down on the techniques on the server and client, respectively.

### A. Design Philosophy

When it comes to a practical framework, several pressing concerns have to be kept in mind. First, it requires careful coordination and control of data being passed between the server and clients. In particular, the server sends a criterion to

guide the client to select instances. The client sends selected instances to the server, which may affect the selection criterion of next rounds. In both directions, the transmission must be efficient. We use a windowed pool-based method wherein each client maintains a small bounded-size buffer of its most recent data. When the buffer fills, a subset of the data is chosen for uploading to the server. After that, the buffer is emptied so that new data can be accommodated. The selection criterion is only updated every time the buffer is emptied. This enables a fine-grained control over the communication bandwidth by simply changing the buffer size and the size of the uploaded subset.

On the server side, the employed learning algorithm should be efficient enough to perform (near) real-time online learning, and be flexible enough to be a standalone module for predicting labels on a new set of test data. Moreover, the selection criterion should be represented in a way that the server can transmit it to the client with a low communication cost. Fortunately, existing state-of-art machine learning algorithms already have many lightweight and flexible aspects that can serve as a good start.

In the context of online semi-supervised learning, it is natural to train a model using labels obtained by the model's own predictions [13], [25]. However, this approach may suffer significantly from the accumulation of wrongly predicted labels over many rounds, resulting in an inaccurate hypothesis. For this reason, it is preferable to update a hypothesis conservatively, thereby alleviating the fluctuations in the performance of the hypothesis.

### *B. Proposed Framework*

Our framework (Figure 1) is designed based on the above considerations. It can be decomposed into several components that drive different functionalities. On the client side, we perform data triage by selecting instances from a candidate pool, where the selection criterion is controlled by the server. On the server side, an online semi-supervised learning algorithm is employed to handle unlabeled submissions. The key is to maintain two learners—a graph-based semi-supervised model and a linear classifier—and let them collaborate to exploit unlabeled data. Specifically, incoming instances are added to the training set of the first learner, which is represented by a graph. The nodes of the graph are instances, and the edges between nodes reflect the similarity between the corresponding instances. Then, the first learner predicts labels for all unlabeled instances in the graph, and randomly samples an instance according to the confidence of its predictions in order to teach the second learner. The second learner updates its hypothesis, and delivers a new selection criterion to the client. At any time, the second learner can be used as a standalone model for predicting new test data.

While different machine learning algorithms can be used as a part of this framework, some techniques lend themselves to our problem setting better than others. In this work, we use the harmonic solution (HS) [3] as the first learner and the soft confidence-weighted classifier (SCW) [26] as the second leaner. Our choice offers several advantages. First, SCW is simple, fast and enjoys state-of-the-art performance on classification. Second, SCW performs a conservative update especially with noisy labels. Third, SCW can be parameterized by a weight vector and a covariance matrix, allowing the server to deliver the selection criterion to the client with a low communication cost. In this work, we simply transmit the weight vector of SCW to the client. On the other hand, HS nicely complements SCW by providing feedback using the data manifold. It can leverage the similarities between instances, which is something that SCW overlooks, to determine labels of unlabeled data. By peering these two models together, we enjoy the best of both worlds, efficient learning and simple parameterization due to SCW, and the ability to exploit manifold information disclosed by unlabeled examples due to HS. Moreover, SCW and HS can be incorporated into a single optimization problem.

One may find it is debatable whether a two-learner structure is really a preferable choice compared to a single learner. For example, one of the alternatives is to train a linear classifier using its own predicted labels without leveraging data manifold information [25]. Unfortunately, such an idea is not effective according to our experiments. Sometimes, the results are even worse than not using any unlabeled data. The reason is twofold. First, a single unlabeled instance can hardly provide any useful information. Second, most of the online linear classifiers only return a single hypothesis on each round, precluding any other possible hypotheses. Hence, some previous work employed Bayesian methods to update a (posterior) distribution over the hypothesis [27], [24]. Unfortunately, the posterior is often complicated. It is not known how to perform the update analytically. Therefore, the learning process can be easily misled and stuck in a wrong direction. Another alternative is to use a graph-based method solely. However, due to the nonparametric nature of graph-based methods, it is not straightforward to deliver the server's model to clients with a low communication cost (for the same reason, nonparametric methods are not favorable in our problem setting). Moreover, graph-based methods are also less efficient for predicting new data, as they usually involve matrix inversion. A two-learner structure, in contrast, surmounts the above problems by complementing each other's drawbacks. The choice of two learners with different underlying mechanisms is a key for good performance.

If we define the communication cost as the total number of vectors in $\mathbb{R}^d$ transmitted over the network, then a straightforward implementation of our proposed framework incurs a cost of at most

$$\underbrace{l + \lfloor \frac{v-l}{q} \rfloor \omega + \min\left((v-l) \bmod q, \omega\right) +}_{\text{client to server}} \quad \underbrace{\lfloor \frac{v-l}{q} \rfloor}_{\text{server to client}} , \tag{1}$$

where $l$ is the number of labeled instances on the server; $v$ is the total length of the unlabeled sequence on the client; $q$ is the size of the pool on the client; and $\omega$ is the number of uploaded instances every time the pool gets full. By ignoring rounding issues, this can be approximated as $l + \frac{v-l}{q}(\omega + 1)$.

In the rest of the paper, we shall refine each component of the proposed framework, presenting their technical details and describing how they cooperate with each other to meet our global goal.

## V. Online Semi-Supervised Learning on the Server

In this section, we describe the server's two learners. First, we review the two standard learners we adapt to our setting, and then we show how to adapt and combine them to handle a partially labeled data stream.

### A. Soft Confidence-Weighted Classifier

We first describe the soft confidence-weighted (SCW) classifier for constructing hypotheses $h_1, h_2, \ldots, h_l$ in an incremental fashion. In a nutshell, the SCW algorithm maintains a Gaussian distribution parameterized by a mean $\mathbf{w} \in \mathbb{R}^d$ and a full covariance matrix $\mathbf{\Sigma} \in \mathbb{R}^{d\times d}$. The mean $\mathbf{w}$ corresponds to the current linear function as described in Section III. The covariance matrix $\mathbf{\Sigma}$ captures the uncertainty and correlation of each feature in $\mathbf{w}$. Given a new labeled instance $(\mathbf{x}_t, y_t) \in L$, SCW sets the new distribution to be the solution of the following optimization problem,

$$(\mathbf{w}_t, \mathbf{\Sigma}_t) = \arg\min_{\mathbf{w},\mathbf{\Sigma}} \{\mathrm{D_{KL}}\left(\mathcal{N}(\mathbf{w}, \mathbf{\Sigma}) \,\|\, \mathcal{N}(\mathbf{w}_{t-1}, \mathbf{\Sigma}_{t-1})\right) + C\max(0, \phi\sqrt{\mathbf{x}_t^\top \mathbf{\Sigma}\mathbf{x}_t} - y_t\mathbf{x}_t^\top \mathbf{w})\}, \quad (2)$$

where the hyperparameter $\phi$ controls the confidence of each update, and $C$ balances between passiveness and aggressiveness. Intuitively, the optimization problem trades off between two requirements. The first term forces the Kullback-Leibler divergence $\mathrm{D_{KL}}$ between the new weight distribution and the old one to be small, so that the parameters do not change dramatically per instance. The second term requires that the new vector $\mathbf{w}_t$ should perform well on $(\mathbf{x}_t, y_t)$.

This optimization problem has a closed-form solution:

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \alpha_t y_t \mathbf{\Sigma}_{t-1}\mathbf{x}_t, \quad \mathbf{\Sigma}_t = \mathbf{\Sigma}_{t-1} - \beta \mathbf{\Sigma}_{t-1}\mathbf{x}_t\mathbf{x}_t^\top \mathbf{\Sigma}_{t-1}. \quad (3)$$

The updating coefficients are calculated as follows:

$$\alpha = \min\{C, \max\{0, \frac{1}{\upsilon\zeta}(-m\psi + \sqrt{\frac{1}{4}m^2\phi^4 + \upsilon\phi^2\zeta})\}\}, \quad (4)$$

$$\beta = \frac{\alpha\phi}{\sqrt{u} + \upsilon\alpha\phi}, \quad (5)$$

where $u = \frac{1}{4}(-\alpha\upsilon\phi + \sqrt{\alpha^2\upsilon^2\phi^2 + 4\upsilon})^2$, $\upsilon = \mathbf{x}_t^\top \mathbf{\Sigma}_{t-1}\mathbf{x}_t$, $m = y_t\mathbf{x}_t^\top \mathbf{w}_{t-1}$, $\psi = 1 + \frac{\phi^2}{2}$ and $\zeta = 1 + \phi^2$.

Compared to other online linear algorithms such as passive-aggressive [28], confidence-weighted [29] and adaptive regularization of weights [30], SCW enjoys the adaptive margin property and reduces the total number of updates over rounds. Most importantly, SCW performs a more conservative update when dealing with a mislabeled instance [26]. In fact, in our experiments SCW outperformed other alternatives on many real-world data sets with noise. For this reason, we later use SCW to learn the instances with "noisy" labels predicted by the first learner.

### B. Harmonic Solution

Harmonic solution (HS) is a graph-based semi-supervised learning method, which assumes that labeled data and unlabeled data are available in advance. Specifically, let $\tilde{\mathbf{y}} = \begin{bmatrix}\mathbf{y}_L \\ \tilde{\mathbf{y}}_U\end{bmatrix}$ where $\mathbf{y}_L = [y_1, y_2, \ldots, y_l]^\top$ and $\tilde{\mathbf{y}}_U$ denote the estimated values on unlabeled data instances. The goal of HS is to minimize the quadratic objective function,

$$\tilde{\mathbf{y}}^* = \min_{\tilde{\mathbf{y}}} \tilde{\mathbf{y}}^\top \mathbf{\Delta}\tilde{\mathbf{y}}, \quad (6)$$

where $\mathbf{\Delta} = \mathbf{D} - \mathbf{S}$ is the graph Laplacian of the similarity graph, which is represented by a matrix $\mathbf{S}$ of weights $s_{i,j}$ that encode pairwise similarities, and $\mathbf{D}$ is a diagonal matrix whose entries are given by $\sum_j s_{i,j}$. HS can be computed in a closed form, which has three representations as follows:

$$\tilde{\mathbf{y}}_U^* = (\mathbf{D}_{UU} - \mathbf{S}_{UU})^{-1}\mathbf{S}_{UL}\mathbf{y}_L \quad (7)$$

$$= -\mathbf{\Delta}_{UU}^{-1}\mathbf{\Delta}_{UL}\mathbf{y}_L \quad (8)$$

$$= (\mathbf{I} - \mathbf{P}_{UU})^{-1}\mathbf{P}_{UL}\mathbf{y}_L, \quad (9)$$

where $\mathbf{P} = \mathbf{D}^{-1}\mathbf{S}$ is the transition matrix on the graph.

In HS, the confidence of using labeled instances to predict unlabeled instances can be achieved in two ways. One way is to regularize $\mathbf{\Delta}$ in Eq. (8) as $\mathbf{\Delta} + \lambda\mathbf{I}$ where $\lambda$ is a scalar and $\mathbf{I}$ is the identity matrix. When $\lambda = 0$, the solution turns into the ordinary harmonic solution. When $\lambda = \infty$, the confidence of labeling unlabeled instances decreases to zero. Alternatively, one can incorporate the knowledge given by $h_l = \{\mathbf{w}_l, \mathbf{\Sigma}_l\}$, i.e. the hypothesis constructed by SCW on labeled instances alone, back into HS. This is illustrated by the dashed line in Fig. 1. Specifically, denote by $\mathbf{g}_U$ the soft labels in $[0, 1]$ on unlabeled data produced by $h_l$, each element of which is computed by $\Phi(\frac{|\mathbf{x}^\top \mathbf{w}_l|}{\sqrt{\mathbf{x}^\top \mathbf{\Sigma}_l\mathbf{x}+1}})$, where $\Phi$ is the cumulative function of the normal distribution. Similar to Eq. (9), the harmonic solution after incorporating $h_l$ is given by

$$\tilde{\mathbf{y}}_U^* = (\mathbf{I} - (1-\eta)\mathbf{P}_{UU})^{-1}((1-\eta)\mathbf{P}_{UL}\mathbf{y}_L + \eta\mathbf{g}_U), \quad (10)$$

where $\eta$ is a scalar in $[0, 1]$. Setting $\eta = 0$ would reduce the solution to the ordinary harmonic solution. At another extreme, setting $\eta = 1$ would ignore the data manifold and completely rely on the predictions of $h_l$ to train SCW.

### C. Efficient Online Adaptation of HS

Note that we need an efficient online version of HS to fit it into the framework. We assume the server receives instances one-by-one. An obvious method is taking each new unlabeled instance, connecting it to its neighbors, and recomputing the harmonic solution. However, the matrix inversion involved has the computational complexity $\mathcal{O}(n^3)$ when the graph contains $n$ nodes. Consequently, this naive solution quickly becomes impractical as more and more instances are added to the graph.

To address this problem, we restrict the size of the graph by substituting the vertices with a smaller set of $k$ distinct centroids. Specifically, we make use of a *doubling algorithm* for incremental $k$-center clustering, which assigns points to centroids in a near optimal way [31]. The original algorithm maintains a set of centroids such that the distance between any two centroids in is at least $R$.

In our framework, the algorithm is adapted as follows. For initialization, we set $R$ to a small positive number, $k$ to be larger than $l$, and $V_0 = \{\mathbf{x}_1, \ldots, \mathbf{x}_l\}$. On round $t$, a new instance $\mathbf{x}_t$ is directly added to the set of centroids if $|V_{t-1}| < k$. If $|V_{t-1}| = k$, then we first try to greedily remove a centroid from $V_{t-1}\backslash\{\mathbf{x}_1, \ldots, \mathbf{x}_l\}$ such that every two centroids in the remained set are no closer than $R$. If such attempt is not successful, then we double $R$ and do the removal again. Finally, $V_t$ is obtained by adding $\mathbf{x}_t$ to the modified $V_{t-1}$.

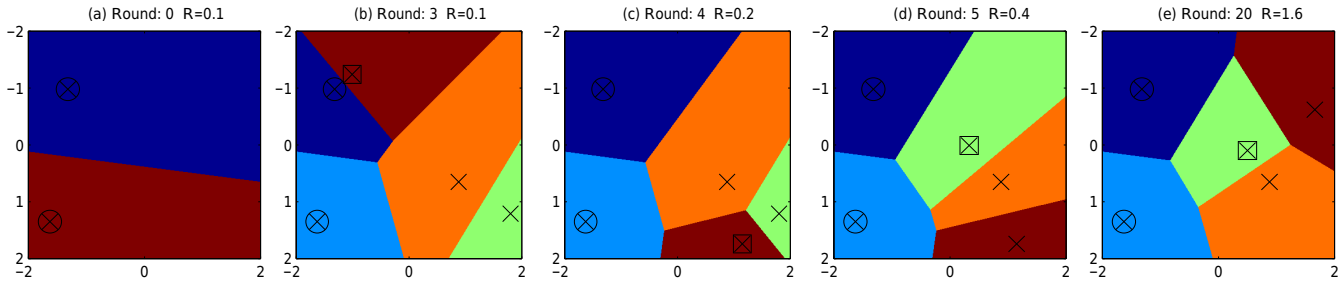


Fig. 2. Adapted doubling algorithm in our framework. ◯ is labeled point, × is centroid and □ is the current point on the $t^{\text{th}}$ round. Color indicates the partition of the space according to the centroids. For this example, we set $l = 2$, $k = 5$ and $R = 0.1$. (a) Initially, the centroid set $V_0$ contains only two labeled points. (b) In the first three rounds, each new point is directly added to the centroid set. (c) On the $4^{\text{th}}$ round, as $V_{t-1}$ is already full, we have to remove a centroid from it. We double $R$ to 0.2, remove the centroid corresponding to the red region from the $3^{\text{rd}}$ round, and add the current point to the centroid set. (d) We double $R$ again, remove the centroid of the green region from the $4^{\text{th}}$ round, and add the new point. (e) The centroid set after 20 rounds.

Figure 2 illustrates this procedure. Note that on each round $t$, $V_t \subseteq \{\mathbf{x}_1, \ldots, \mathbf{x}_t\}$ and $\mathbf{x}_t \in V_t$. Moreover, unlike the original version, the modified algorithm only guarantees that every two centroids in $V_t \setminus \{\mathbf{x}_1, \ldots, \mathbf{x}_l, \mathbf{x}_t\}$ are no closer than $R$.

After restricting the size of the graph, the remaining bottleneck of HS includes updating $\mathbf{S}$ and inverting a $k \times k$ matrix. While the incremental update of $\mathbf{S}$ can be easily done with a block matrix, speeding up the matrix inversion is less straightforward [32]. In this work, we use conjugate gradient descent to solve an equivalent linear system and therefore avoid the expensive inversion. Note that the solution of $\tilde{\mathbf{y}}_U$ in Eq. (8) is equivalent to the solution of the following linear system,

$$\mathbf{\Delta}_{UU}\tilde{\mathbf{y}}_U = -\mathbf{\Delta}_{UL}\mathbf{y}_L. \tag{11}$$

The hope is that each iteration is $\mathcal{O}(k)$ and convergence can be reached in relatively few iterations, in contrast to the naive inversion that costs $\mathcal{O}(k^3)$. To ensure fast convergence, we use the Jacobi preconditioner, which is simply the diagonal of $\mathbf{\Delta}_{UU}$; and set the initial guess of $\tilde{\mathbf{y}}_U$ to be the solution of the last round.

### D. Combining HS with SCW

We now show the construction of hypotheses $h_{l+1}, \ldots, h_n$ from unlabeled data $U$ by combining HS with SCW. Intuitively, we let HS teach its most confident predictions to SCW. To see that, we first rewrite Eq. (9) so that each element of $\tilde{\mathbf{y}}_U$ is given by

$$\begin{aligned}\tilde{y}_i &= \sum_{j:y_j=1} (\mathbf{I} - \mathbf{P}_{Ui})^{-1}\mathbf{P}_{Uj} - \sum_{j:y_j=-1} (\mathbf{I} - \mathbf{P}_{Ui})^{-1}\mathbf{P}_{Uj} \\ &= p_i^1 - p_i^{-1},\end{aligned} \tag{12}$$

where $p_i^1$ and $p_i^{-1}$ can be interpreted as the probability of instance $\mathbf{x}_i$ belonging to the positive and negative class, respectively. Therefore, we can use $|\tilde{y}_i| \in [0, 1]$ to represent the confidence of predicting the label $\text{sign}(\tilde{y}_i)$ to the instance $\mathbf{x}_i$.

Though SCW and HS are conceptually separated, they can now be combined in an optimization problem as follows,

$$\begin{aligned}(\mathbf{w}_t, \mathbf{\Sigma}_t) \coloneqq \arg\min_{\mathbf{w},\mathbf{\Sigma}} \{&\mathrm{D}_{\mathrm{KL}}\left(\mathcal{N}(\mathbf{w}, \mathbf{\Sigma}) \,\|\, \mathcal{N}(\mathbf{w}_{t-1}, \mathbf{\Sigma}_{t-1})\right) \\ &+ C \max(0, \phi\sqrt{\mathbf{x}_j^\top \mathbf{\Sigma}\mathbf{x}_j} - \tilde{y}_j \mathbf{x}_j^\top \mathbf{w})\} \\ \text{s.t.} \quad & j \sim \text{categorical}(\bar{p}_{l+1}, \bar{p}_{l+2}, \ldots, \bar{p}_k) \\ & \bar{p}_i = \frac{|\tilde{y}_i|}{\sum_i |\tilde{y}_i|}, \quad \text{where } \tilde{y}_i \text{ is given by Eq. (12).}\end{aligned} \tag{13}$$

The combined algorithm works as follows. On round $t$, the new unlabeled instance $\mathbf{x}_t$ is added to $V_{t-1}$ to construct $V_t$. The training instances fed to SCW are sampled according to HS confidence into labels of $V_t$. The highly uncertain predictions are likely to be excluded from learning.

Note that on round $t$ the current instance $\mathbf{x}_t$ is always learned by HS (because $\mathbf{x}_t$ is added to $V_{t-1}$ for constructing $V_t$), but it is not necessarily learned by SCW. Depending on the confidence of HS, SCW may be fed with any instance in $V_t$. More precisely, there are three outcomes of an unlabeled instance: (i) it is taught to SCW by random sampling; (ii) it is retained in the centroids set $V$; or (iii) it is removed from $V$ by the clustering algorithm in Section V-C.

One can observe some similarity between Eq. (13) and the objective function of online manifold regularization [15]. The latter used the manifold constraint as a regularization term in the objective function. While both methods attempt to learn a large margin separator using manifold information, the major difference is in the search space. In particular, online manifold regularization searches on a class of hypotheses to find one that is smooth on the graph. But when the hypothesis space is severely restricted, such as linear functions, the manifold regularization term simply turns into a penalty on the weight-vector, preventing the algorithm from harnessing any useful information about the manifold. Our method, in contrast, learns a linear function conditioned on labels induced by the manifold, providing better performance and flexibility.

### E. Predicting New Data

At any point in time, the learned SCW on the server can be used as a standalone component for predicting the labels for new (test) data. An obvious way is to use the last hypothesis directly returned by SCW as the output classifier. However, the training set could happen to be such that we end up with a bad last hypothesis. To promote robustness and stability, we employ the *cutoff averaging* technique to build an ensemble as the output classifier [33], rather than committing to a single online hypothesis.

In cutoff averaging, each distinct online hypothesis is associated with a *survival time*, which is defined as the number of consecutive rounds the corresponding hypothesis survives before SCW replaces it with a new hypothesis. On the last round $n$, we have observed a sequence of online hypothesis $\{h_t\}_{t=0}^{n-1}$. Let $\Theta_\nu \subseteq \{h_t\}_{t=0}^{n-1}$ be the set of distinct hypotheses

whose survival time is greater than $\nu$. The cutoff averaging technique defines the output hypothesis $h^*$ as a weighted average over the hypothesis in $\Theta_\nu$, where the weight of a hypothesis with survival time $r$ is proportional to $r - \nu$. The cutoff parameter $\nu$ sets the bar for acceptance into the ensemble. Define the sequence of binary variables $\{B_t\}_{t=0}^{n-1}$ as follows

$$B_t = \begin{cases} 1 & \text{if } t = 0 \text{ or if } t \geq \nu \text{ and } h_{t-\nu} = \ldots = h_t \\ 0 & \text{otherwise} \end{cases} \quad (14)$$

The optimal $\nu^*$ can be determined by solving the following optimization problem:

$$\nu^* = \arg \min_{\nu:\Theta_\nu \neq \emptyset} \left\{ \bar{\ell} + \sqrt{\frac{\gamma \bar{\ell}}{\sum B_t}} + \frac{7\gamma}{2 \sum B_t} \right\} \quad (15)$$

$$\text{s.t.} \qquad \bar{\ell} = (\sum_{t=0}^{n} B_t)^{-1} \sum_{t=0}^{n} B_{t-1} \ell(h_{t-1}; \mathbf{x}_t, \tilde{y}_t),$$

where $\gamma$ is a constant with respect to $\nu$. This solution ensures a large $\nu$ and a sparse ensemble if a few online hypotheses stand out with significantly long survival times. If most of the hypotheses have short survival times, then a small $\nu$ is preferred and the output ensemble is dense. Note that the maximal number of distinct survival times in a sequence of $n$ hypotheses is $\mathcal{O}(\sqrt{n})$. Thus, the search space of $\nu^*$ is small enough for efficient computation.

## VI. Selective Sampling on Clients

Given a communication budget, the client needs to select instances from an unlabeled candidate pool such that the model on the server might be improved by learning these instances. Random selection is a simple approach, but a better selection criterion should meet the current demands of the server's model. To design such a criterion, the client needs (full/partial) information about the model currently on the server. Although we have two learners on the server, we transmit only the weight-vector of SCW from the server to the client because SCW directly determines the performance that we are interested in (while HS serves to reduce the uncertainty of SCW) and transmitting only the weight-vector is communication-efficient.

Two important aspects of a good criterion are the utility and redundancy. The *Utility* measures the potential improvement of SCW associated with each instance. The *Redundancy* measures the degree of information sharing by the selected instances. For a candidate pool $Q = \{\mathbf{x}_1, \ldots, \mathbf{x}_q\}$, let the *utility score* be the sum of their individual utilities, i.e., $f_u(Q) = \sum_{i=1}^{q} f_u(\mathbf{x}_q)$. The redundancy is denoted by $f_r(Q)$. The desired selections should be optimal in terms of both utility and redundancy. Formally, given a communication budget $\omega$ for processing the pool $Q$, the goal is to select a subset $T$ from $Q$ such that

$$T = \arg \max_{T \subseteq Q : |T| = \omega} f_u(T) - f_r(T). \quad (16)$$

Previous research on active learning has proposed several choices for $f_u$ and $f_r$ [34], [35]. We use function value based scores, namely $\frac{1}{1+|\mathbf{x}^\top \mathbf{w}|}$, as $f_u$ [36], and the sample covariance of instances as $f_r$ [37]. As $f_u$ is linear and $-f_r$ is *submodular*, Eq. (16) turns into a submodular function, which satisfies a *diminishing returns* property. A near-optimal solution of Eq. (16) can be found efficiently using a greedy algorithm [38]. Intuitively, we favor the instances that are close to the decision hyperplane of the current SCW and far away from each other. Note that the submitted instances are of low confidence according to SCW, and by querying HS for their labels, they may offer some supervision to SCW.

## VII. Experiments

We conducted a series of experiments to verify the effectiveness of the proposed framework in the context of communication-efficient online semi-supervised distributed learning. The first experiment focuses on the server's model and compares the proposed two-learner method against several baselines including its single-learner counterparts. The second experiment focuses on the client's selection criterion.

### A. Experimental Setup

Experiments were conducted on seven data sets downloaded from either the UCI ML repository (wearable, skin) or the LIBSVM website (mushroom, mnist, webspam, gisette, ijcnn1). The motion recognition data set wearable and digit recognition data set mnist were converted into a set of binary problems, respectively, where each class is discriminated against every other class. Totally, we produced 10 problems from wearable and 45 from mnist. For each data set, we balanced the number of instances of each class and linearly rescaled the feature values into the range $[-1, 1]$.

We evaluated the algorithms using a set of trials with different partitions of the training and test data. In each trial, we randomly held out half of the data for testing; all instances in the test set were labeled by the algorithms. The remaining data was used for training, of which only a small amount was labeled. Both training and test sets were class-balanced. Next, we randomly permuted the training data and kept labeled data always at the beginning. All algorithms were then incrementally trained with the same permutation in each trial. For evaluation, we paused the training at regular intervals, computed the output hypothesis so far, and calculated its test accuracy. We used the first trial to tune the hyperparameters (e.g., $C$, $\phi$ in Eq. (13)), and the best choice for its hyperparameter is then fixed in the remaining trials. We used $\eta = 0$ in Eq. (10). The reported results were averaged over 100 trials.

In all experiments, we used a 5-nearest neighbor graph as the similarity graph of HS on the server. The edges were weighted as $s_{i,j} = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2d\sigma^2})$, where $d$ is the number of features and $\sigma$ denotes the mean of their standard deviations. The maximum number of centroids in the graph was 300. In Sections VII-B and VII-C, the initial 2% of the training instances are labeled. The size of the candidate pool on the client was 50, from which 10 instances were submitted to the server (a 20% sampling rate). Our code is public available at https://db.tt/eMeD6nlp.

### B. Comparison of Server's Model

We first compare different models on the server and show the effectiveness of the proposed method. To focus on the server side, we let the client randomly select instances to upload. In particular, the following methods were evaluated in this experiment.

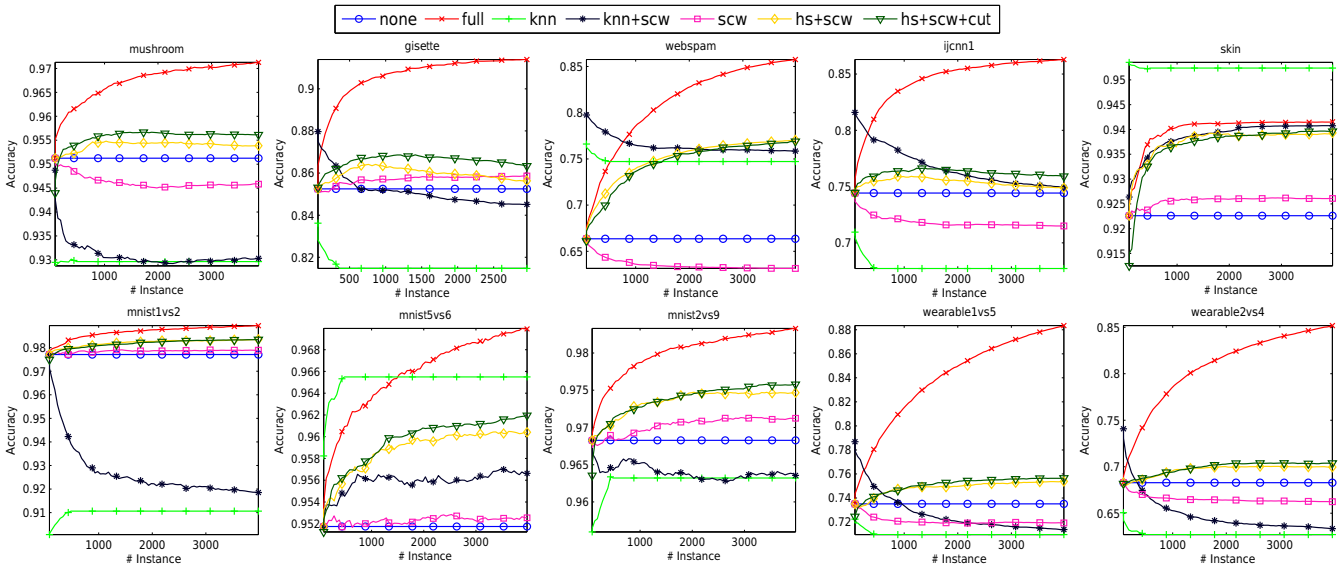


Fig. 3. Test accuracy of different models on the server. The x-axis represents the number of unlabeled instances on the client. The origin corresponds to the point where the initial 2% of the data has been labeled and learned and the first unlabeled instance comes in. The client randomly selected 10 instances from every 50 instances. `full` is an idealized approach in which an oracle labels all selected instances. `none` does not upload any unlabeled instance to the server, so the corresponding test accuracy is constant. A tournament result between algorithms on all problems of mnist and wearable are presented in Tables I(a) to I(b).

**`none`** No unlabeled instances are uploaded to the server. The server stops learning right after labeled instances. Assuming that unlabeled instances can provide useful information, then this approach should give the worst performance.

**`full`** All uploaded instances are labeled by an oracle. Intuitively, this approach should give the best result due to the availability of full information. This is an idealized case with 10x (20% vs. 2%) more labeled data.

**`knn`** The server employs $k$-nearest neighbors algorithm, where $k = 5$. The training set is built by first including all labeled instances, and then adding unlabeled instances with its corresponding predicted labels. The maximum number of allowed training examples is 300.

**`knn+scw`** The server consists of a two-learner model: `knn` followed by `scw`. The prediction of `knn` is used for training `scw`.

**`scw`** The server consists of an SCW model only, which "learns" each unlabeled instance using its own prediction.

**`hs+scw`** Proposed two-learner model on the server.

**`hs+scw+cut`** Proposed `hs+scw` model with cutoff averaging for predicting test data.

Note that, `none` and `full` are essentially standard online learning, in which models are trained with labeled data. We also implemented a baseline containing HS only on the server. However, we had to terminate it due to its poor efficiency. A comparison of the above methods is shown in Fig. 3. In Tables I(a) to I(b) we computed the number of problems for which one algorithm achieved a higher test accuracy than another algorithm in $t$-test.

It can be observed that proposed `hs+scw` and `hs+scw+cut` enjoy superior performance on 8 out of 10 problems compared to other partial label competitors. On 45 mnist problems, `hs+scw` and `hs+scw+cut` yielded on average 0.966 and 0.971 accuracy, respectively. On 20 wearable problems, `hs+scw` and `hs+scw+cut` gave 0.699

TABLE I. PERFORMANCE COMPARISON OF DIFFERENT SELECTION STRATEGIES. THE VALUE IN THE TABLE DENOTES THE NUMBER OF TIMES THAT THE ROW ALGORITHM ACHIEVED SIGNIFICANTLY BETTER ACCURACY THAN THE COLUMN ALGORITHM UNDER THE $t$-TEST WITH $p = 0.05$.

(a) Different server's models on mnist (45 problems).

| | none | full | knn | knn+scw | scw | hs+scw | hs+scw+cut |
|---|---|---|---|---|---|---|---|
| none | - | 0 | 12 | 9 | 6 | 0 | 0 |
| full | 45 | - | 39 | 35 | 35 | 30 | 34 |
| knn | 20 | 0 | - | 10 | 13 | 4 | 3 |
| knn+scw | 13 | 0 | 3 | - | 10 | 0 | 1 |
| scw | 2 | 0 | 3 | 2 | - | 1 | 1 |
| hs+scw | 31 | 1 | 15 | 22 | 24 | - | 3 |
| hs+scw+cut | 30 | 1 | 15 | 21 | 23 | 2 | - |

(b) Different server's models on wearable (10 problems).

| | none | full | knn | knn+scw | scw | hs+scw | hs+scw+cut |
|---|---|---|---|---|---|---|---|
| none | - | 0 | 6 | 6 | 5 | 0 | 0 |
| full | 10 | - | 10 | 10 | 10 | 10 | 10 |
| knn | 0 | 0 | - | 1 | 0 | 0 | 0 |
| knn+scw | 0 | 0 | 2 | - | 0 | 0 | 0 |
| scw | 0 | 0 | 3 | 2 | - | 0 | 0 |
| hs+scw | 6 | 0 | 9 | 8 | 8 | - | 1 |
| hs+scw+cut | 6 | 0 | 8 | 8 | 7 | 0 | - |

(c) Different selection strategies on mnist (45 problems).

| | none | full | all | rand | certain | uncertain | submod |
|---|---|---|---|---|---|---|---|
| none | - | 0 | 0 | 9 | 6 | 0 | 0 |
| full | 45 | - | 45 | 45 | 45 | 45 | 45 |
| all | 30 | 0 | - | 2 | 27 | 2 | 1 |
| rand | 30 | 0 | 1 | - | 1 | 0 | 0 |
| certain | 34 | 0 | 9 | 2 | - | 0 | 0 |
| uncertain | 35 | 0 | 15 | 22 | 24 | - | 0 |
| submod | 38 | 0 | 17 | 25 | 25 | 6 | - |

(d) Different selection strategies on wearable (10 problems).

| | none | full | all | rand | certain | uncertain | submod |
|---|---|---|---|---|---|---|---|
| none | - | 0 | 0 | 0 | 0 | 0 | 0 |
| full | 10 | - | 10 | 10 | 10 | 10 | 10 |
| all | 3 | 0 | - | 0 | 1 | 1 | 0 |
| rand | 3 | 0 | 0 | - | 1 | 0 | 0 |
| certain | 3 | 0 | 0 | 0 | - | 0 | 0 |
| uncertain | 5 | 0 | 2 | 2 | 2 | - | 0 |
| submod | 6 | 0 | 3 | 2 | 2 | 0 | - |

and $0.714$ accuracy, respectively. They are consistently better than the single-learner counterpart `scw` on all data sets. This indicates the effectiveness of leveraging manifold information of the graph. In fact, on webspam, ijcnn1 and wearable, `scw` is even worse than `none`. On webspam, its test accuracy starts with $0.658$, decreasing over time and finally yielded $0.637$. This is due to the fact that `scw` completely relies on its own prediction for learning. When the labeling rate is small, the initial hypothesis constructed by labeled data may not be accurate enough. As a consequence, the prediction of `scw` on the new instance is likely to be wrong, which in turn might mislead the learning procedure. The `knn`-based approaches, which employ majority voting based on local information, did not show consistent performance. On gisette, webspam, and ijcnn1, the test accuracy of `knn` decreases until the maximum number of training instances is reached, whereas on mnist it increases. This indicates that a simple bootstrapping for `knn` is not robust. Also note that, it is not straightforward to formulate a communication-efficient selection policy for `knn` due to its nonparametric nature. The idea of using the prediction of `knn` to teach `scw` is not effective, often resulting in degraded performance of `scw` over time. One may note that `knn` enjoys superior performance on skin. This is probably due to the characteristics of this data set. Each instance in skin has only three features, representing red, green, and blue color, respectively. The task of distinguishing skin from non-skin on such data is particularly suitable for `knn`.

### C. Comparison of Selection Strategy

Fixing the model on the server as `hs+scw+cut`, we study the following strategies on the client side.

**`all`** All unlabeled instances are uploaded without selection. This incurs 5x the communication costs versus other approaches.

**`rand`** Randomly selects instances for uploading.

**`certain`** The most certain instances according to the current server model $\mathbf{w}$ are uploaded. The score is defined as $|\mathbf{x}^\top \mathbf{w}|$ . This method is similar in spirit to [25].

**`uncertain`** The most uncertain instances are uploaded. The score is defined as $\frac{1}{1+|\mathbf{x}^\top \mathbf{w}|}$.

**`submod`** Selection is done by optimizing the submodular function described in Section VI. It simultaneously considers the uncertainty and redundancy.

Note that there are many ways to wrap $|\mathbf{x}^\top \mathbf{w}|$ into a selection criterion, such as transforming it into a probability value [24], [21]. However, despite introducing extra hyperparameters into the model, they are not significantly different in essence. For the sake of clarity, we concentrate on the above five strategies. The result is shown in Fig. 4, in which `none` and `full` are as defined in Section VII-B. In Tables I(c) to I(d) we computed the number of problems for which one selection strategy achieved a higher test accuracy than another algorithm in $t$-test with $p = 0.05$.

It is interesting to see that `all`, which transmits all unlabeled data, does not lead to better performance. In fact, on mnist, mushroom, and gisette, `all` yields worse test accuracy compared to selective transmission. This confirms the intuition that not all unlabeled instances are useful. It also suggests the necessity of using a selective sampling strategy on the client. Not only the communication costs can be saved, but also a better model might be learned. Moreover, it can be observed that `uncertain` and `submod` show significant improvements over `rand`. They often converge faster than `rand` and lead to better optimal hypotheses. On the contrary, selecting most certain instances is not beneficial. On ijcnn1 and skin, the accuracy decreases over time (the accuracy of `certain` on skin drops to under 80% at 4000 instances, and is not shown to better see the other results), showing that a bad client selection strategy can have negative impact on the performance of the server's model. On mnist and mushroom, `submod` further improves over `uncertain`, while `uncertain` is better for gisette and ijcnn1.

## VIII. Discussion and Conclusion

This paper poses a new learning problem on the client-server design, which is motivated by real-world applications such as intelligent traffic systems and wearable devices. To solve this problem, we have presented a framework that provides communication-efficient online semi-supervised learning in the client-server setting. The framework consists of two parts. On the server side, two learners work collaboratively to learn from a partially labeled data stream. The two-learner structure can effectively exploit the data manifold to determine labels for unlabeled data. It is also efficient in the sense that it does not require storing all the data. The proposed method enjoys superior and stable performance on several real-world data sets. On the client side, we investigated several selection criteria and showed how the server communicates with the client. We showed that a selection criterion based on uncertainty and redundancy is effective. It is worth highlighting that intelligent sampling on the client not only saves communication costs, but, perhaps surprisingly, also may result in a better model on the server compared to uploading all instances.

While for simplicity we considered the case of a single client, our framework can be readily extended to learn a model across multiple clients. Because a client's candidate pool is discarded once its selection has been made (i.e., there is no per-client history), processing a full pool of data is the same regardless of which client processes it (assuming iid data streams). Thus, as long as the server sends the current weight-vector to the client who is about to send data next, the processing and hence the accuracy is effectively the same as in the single client case, with the same overall sampling rate. If the data generation rate (data instances per second) increases linearly with the number of clients, though, two issues arise. First, the aggregate weight-vectors per second that the server sends increases linearly. This can be mitigated by having the server operate in rounds such that instances from multiple clients are processed in each round, and an updated weight-vector is sent only at the end of the round. This implies that clients select instances based on a less-frequently-updated uncertainty measure. Note also that in this case, one would like to apply the redundancy measure *across* the clients, which would require additional communication and coordination.

## Acknowledgment

This work was supported in part by the Ministry of Science and Technology, National Taiwan University, and Intel

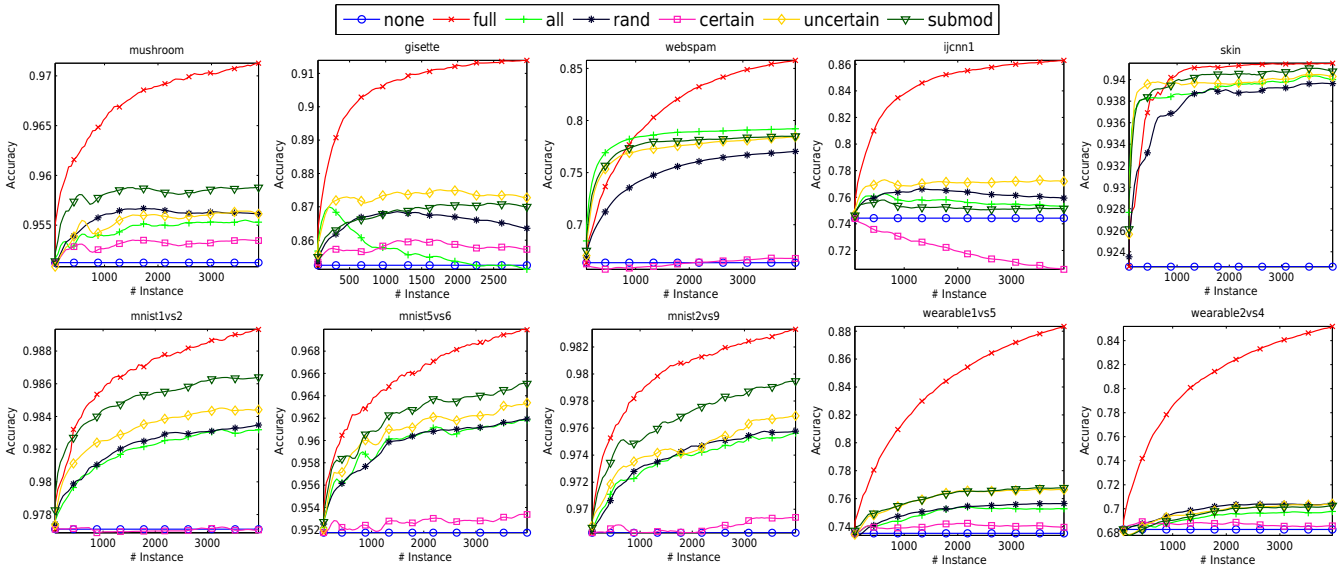


Fig. 4. Test accuracy of different selection strategies for a fixed communication budget. The client selected 10 instances from every 50 instances, except for `all`, which selected all instances and hence incurs 5x the communication costs. The server used `hs+scw+cut`. The labeling rate was 2%, except for `full`, which labeled all selected instances using an oracle. A tournament result between algorithms on all problems of mnist and wearable are presented in Tables I(a) to I(b).

Corporation under Grants MOST 103-2911-I-002-001, NTU-ICRP-104R7501, and NTU-ICRP-104R7501-1.

## REFERENCES


[1] A. McCallum, K. Nigam *et al.*, "Employing EM and pool-based active learning for text classification." in *Proceedings of ICML*, 1998.

[2] K. Bennett, A. Demiriz *et al.*, "Semi-supervised support vector machines," in *Proceedings of NIPS*, 1999.

[3] X. Zhu, Z. Ghahramani, J. Lafferty *et al.*, "Semi-supervised learning using gaussian fields and harmonic functions," in *Proceedings of ICML*, 2003.

[4] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in *Proceedings of COLT*. ACM, 1998.

[5] I. Muslea, S. Minton, and C. A. Knoblock, "Active+ semi-supervised learning= robust multi-view learning," in *Proceedings of ICML*, 2002.

[6] X. Zhu, J. Lafferty, and Z. Ghahramani, "Combining active learning and semi-supervised learning using gaussian fields and harmonic functions," in *ICML workshop on the continuum from labeled to unlabeled data in machine learning and data mining*, 2003.

[7] Z. Wang, Y. Song, and C. Zha, "Efficient active learning with boosting." in *SDM*, 2009.

[8] Z.-H. Zhou and M. Li, "Semi-supervised learning by disagreement," *Knowledge and Information Systems*, vol. 24, no. 3, 2010.

[9] G. Tur, D. Hakkani-Tür, and R. E. Schapire, "Combining active and semi-supervised learning for spoken language understanding," *Speech Communication*, vol. 45, no. 2, 2005.

[10] R. Huang and W. Lam, "An active learning framework for semi-supervised document clustering with language modeling," *Data & Knowledge Engineering*, vol. 68, no. 1, 2009.

[11] V. S. Iyengar, C. Apte, and T. Zhang, "Active learning using adaptive resampling," in *Proceedings of SIGKDD*. ACM, 2000.

[12] Z.-H. Zhou, K.-J. Chen, and Y. Jiang, "Exploiting unlabeled data in content-based image retrieval," in *Proceedings of ECML*. Springer, 2004.

[13] Z.-H. Zhou, D.-C. Zhan, and Q. Yang, "Semi-supervised learning with very few labeled training examples," in *Proceedings of AAAI*, 2007.

[14] H. Grabner, C. Leistner, and H. Bischof, "Semi-supervised on-line boosting for robust tracking," in *ECCV*. Springer, 2008.

[15] A. B. Goldberg, M. Li, and X. Zhu, "Online manifold regularization: A new learning setting and empirical study," in *Machine Learning and Knowledge Discovery in Databases*. Springer, 2008.

[16] G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," *Proceedings of NIPS*, 2001.

[17] M. Valko, B. Kveton, H. Ling, T. Daniel *et al.*, "Online semi-supervised learning on quantized graphs," in *Proceedings of UAI*, 2010.

[18] D. Cohn, L. Atlas, and R. Ladner, "Improving generalization with active learning," *Machine Learning*, vol. 15, no. 2, 1994.

[19] X. Zhu, P. Zhang, X. Lin, and Y. Shi, "Active learning from data streams," in *Proceedings of ICDM*. IEEE, 2007.

[20] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà, "New ensemble methods for evolving data streams," in *Proceedings of SIGKDD*. ACM, 2009.

[21] W. Chu, M. Zinkevich, L. Li, A. Thomas, and B. Tseng, "Unbiased online active learning in data streams," in *Proceedings of SIGKDD*. ACM, 2011.

[22] F. Shen, H. Yu, K. Sakurai, and O. Hasegawa, "An incremental online semi-supervised active learning algorithm based on self-organizing incremental neural network," *Neural Computing and Applications*, vol. 20, no. 7, 2011.

[23] S. Furao and O. Hasegawa, "An incremental network for on-line unsupervised classification and topology learning," *Neural Networks*, vol. 19, no. 1, 2006.

[24] A. B. Goldberg, X. Zhu, A. Furger, and J.-M. Xu, "Oasis: Online active semi-supervised learning." in *Proceedings of AAAI*, 2011.

[25] Y. Haimovitch, K. Crammer, and S. Mannor, "More is better: Large scale partially-supervised sentiment classification," in *Proceedings of ACML*, 2012.

[26] J. Wang, P. Zhao, and S. C. Hoi, "Exact soft confidence-weighted learning," in *Proceedings of ICML*, 2012.

[27] T. Jaakkola and M. Jordan, "A variational approach to bayesian logistic regression models and their extensions," in *AISTATS*. Citeseer, 1997.

[28] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, "Online passive-aggressive algorithms," *JMLR*, vol. 7, 2006.

[29] K. Crammer, M. Dredze, and F. Pereira, "Exact convex confidence-weighted learning," in *Proceedings of NIPS*, 2008.

[30] K. Crammer, A. Kulesza, and M. Dredze, "Adaptive regularization of weight vectors," *Machine Learning*, 2009.

[31] M. Charikar, C. Chekuri, T. Feder, and R. Motwani, "Incremental clustering and dynamic information retrieval," in *STOC*. ACM, 1997.

[32] V. V. Williams, "Breaking the Coppersmith-Winograd barrier," 2011.

[33] O. Dekel, "From online to batch learning with cutoff-averaging," in *Proceedings of NIPS*, 2008.

[34] B. Settles, "Active learning literature survey," *University of Wisconsin, Madison*, 2010.

[35] Y. Fu, X. Zhu, and B. Li, "A survey on instance selection for active learning," *Knowledge and information systems*, 2013.

[36] N. Cesa-Bianchi, C. Gentile, and L. Zaniboni, "Worst-case analysis of selective sampling for linear classification," *JMLR*, vol. 7, 2006.

[37] C. Guestrin, A. Krause, and A. P. Singh, "Near-optimal sensor placements in gaussian processes," in *Proceedings of ICML*. ACM, 2005.

[38] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions I," *Mathematical Programming*, vol. 14, no. 1, 1978.