Incremental Semi-Supervised Learning from Streams for Object Classification

Ioannis Chiotellis*[1] Franziska Zimmermann*[1,2] Daniel Cremers[1] Rudolph Triebel[1,3]

***Abstract*— The Label Propagation (LP) algorithm, first introduced by Zhu and Ghahramani [1], is a semi-supervised method used in transductive learning scenarios, where all data are available already in the beginning. In this work, we present a novel extension of the LP algorithm for applications where data samples are observed sequentially – as is the case in autonomous driving. Specifically, our "Incremental Label Propagation" algorithm efficiently approximates the so called harmonic solution on a nearest-neighbor graph that is regularly updated by new labeled and unlabeled nodes. We achieve this by reformulating the original algorithm based on an active set of nodes and by introducing a threshold to decide whether the label of a given node should be updated or not. Our method can also deal with graphs that are not fully connected, and we give a formal convergence proof for this general case. In experiments on the challenging KITTI benchmark data stream, we show superior performance in terms of both test accuracy and number of required training labels compared to state-of-the-art online learning methods.**

I. INTRODUCTION

Most standard learning approaches used for classification tasks in robotics have the drawback that they either require a large amount of hand-labeled training data, or they are hardly adaptive to newly observed data samples. In particular, when considering an object classification problem from a given stream of training data, there is often just not enough ground truth information available to train a complex model such as a deep neural network. And even if there were sufficient training data, it is very difficult to efficiently update the classifier on newly arriving data from the stream, be it labeled or unlabeled.

Therefore, in this paper, we propose a learning algorithm for classification that is both semi-supervised and incremental, i.e. it can perform very fast model updates for new data samples. Furthermore, and in contrast to other semi-supervised approaches like transductive SVMs (Vapnik [2]) or more recent deep learning approaches (Kingma et al. [3], Sajjadi et al. [4], Haeusser et al. [5]), we can more flexibly and more directly trigger the learning process from a single sample instead of using batches of data with a fixed size. Our approach is based on the idea that newly observed data samples – both labeled and unlabeled ones – only have a *local* influence on the class predictions of the given unlabeled graph nodes. Thus, it is actually not necessary to recompute LP from scratch every time a sample arrives. In our formulation, this notion of locality is guided by a threshold ϑ, by which the trade-off between efficiency and accuracy is managed. In practice, however, it turns out that a relatively small area of influence (i.e. a large value of ϑ) already results in a very good performance while reducing the number of propagation iterations significantly.

To summarize, our key contributions are:

- A novel incremental semi-supervised learning method, we call "Incremental Label Propagation", where the area of influence of the algorithm can be easily tuned with the one hyper-parameter ϑ.
- A proof of convergence of the Label Propagation algorithm for partially connected graphs.
- An empirical evaluation of our algorithm on a challenging benchmark data stream (See Fig. 1), showing the effectiveness of exploiting unlabeled data for stream-based classification.

II. RELATED WORK

The Label Propagation (LP) algorithm [1] has been used by several authors for semi-supervised learning tasks. Chapelle et al. [6] show that the LP algorithm is equivalent to the well known Jacobi algorithm for solving sparse linear systems. In fact, LP solves the problem of inverting the matrix Δ_{UU}, i.e. the submatrix of the graph Laplacian corresponding to the unlabeled nodes. As introduced by Zhu et al. [7], the resulting solution is the so called harmonic solution on the graph. Zhu and Ghahramani [1] prove convergence of their algorithm for fully connected graphs, but we also give a convergence proof on partially connected graphs, where each connected component contains at least one labeled node.

In the literature, one can find several ideas for efficiently approximating the harmonic solution for a large non-growing graph. The algorithm of Ganu and Kveton [8] computes the harmonic solution on a subgraph of nodes for which the label can be predicted with high certainty. However, for the algorithm to be fast, the subgraph must be small, which means that only the labels of a small subset of nodes will be predicted. Delalleau et al. [9] suggest to compute the harmonic solution on a subset of unlabeled nodes $\mathcal{S}$, where the label vectors of the nodes outside of $\mathcal{S}$ are set equal to the weighted average of their labeled neighbors and the neighbors in $\mathcal{S}$. While this is more efficient, no theoretical analysis is given on how much the solution differs from the exact harmonic solution on the full graph. Moreover, it is not clear how to get from the harmonic function approximation

*The authors contributed equally.

[1]Computer Vision Group, Dep. of Computer Science, TU Munich, 85748 Garching, Germany {chiotell,zimmermann,cremers,triebel}@in.tum.de

[2]Carl Zeiss Microscopy GmbH, 81379 Munich, Germany zimmermann.franzi@t-online.de

[3]Institute of Robotics and Mechatronics, Dep. of Perception and Cognition, German Aerospace Center (DLR), 82234 Wessling, Germany rudolph.triebel@dlr.de

on a fixed graph to the approximation on an enlarged graph without computing everything from scratch.

Delalleau et al. [9] derive an inductive formula to assign an optimal label to a new point given the harmonic solution on the old points. However, optimality only holds under the assumption that the label vectors of old points cannot change. If many unlabeled or a few labeled points arrive, the new information is not utilized to update the labels of previously classified points. In the algorithm of Valko et al. [10], new incoming points are first assigned to one of k clusters using the doubling algorithm for incremental k-center clustering (Charikar et al. [11]). Then, each new point is replaced by its cluster center and the harmonic solution is computed on the cluster centers. Therefore, only a $k \times k$ instead of a $n \times n$ matrix has to be inverted to get the exact harmonic solution. Moreover, only the cluster centers and the cluster sizes have to be stored instead of all the data. However, the number of clusters k, that shall represent the data points, must be fixed beforehand. Also, once a point is assigned to a cluster, its original feature vector is discarded, so its position inside the cluster is not taken into account any more.

Zhu [12] also uses the idea of clustering to approximate the harmonic solution on large graphs. He computes the harmonic solution on a backbone graph consisting of mixture components, the so called harmonic mixture solution. An advantage of this approach is that a generative mixture model can naturally handle unseen points. Although a prediction for the label of a new point is easy to obtain, new points do not influence the model, except if the whole model is retrained. This is problematic if there are only a few points available in the beginning. Moreover, the problem of determining the number of clusters remains. The authors show that for a good approximation of the harmonic solution, the number of mixture components must be above a certain threshold that is data-dependent. Thus, the determination of a good number of clusters that keeps the effort of finding the harmonic mixture solution low, but still yields an acceptable approximation of the harmonic solution remains an open question.

Therefore, in this work, we do not reduce the data to a backbone graph but present the "Incremental Label Propagation" algorithm that allows us to approximate the harmonic solution on the full graph consisting of all nodes efficiently, in the context of a permanently growing graph.

III. INCREMENTAL LABEL PROPAGATION

A. Reviewing Offline Label Propagation

We denote a partially labeled data set $\mathcal{D}$ of size n as the union of two subsets: the set $\mathcal{L}$ with l labeled data points $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)$ where $\mathbf{x}_i \in \mathbb{R}^d$ are feature vectors and $y_i \in \{1, \ldots, C\}$ are class labels, and the set $\mathcal{U}$ with u unlabeled points $\mathbf{x}_{l+1}, \ldots, \mathbf{x}_{l+u}$, i.e. $n = l + u$. Our aim is to infer class labels for unlabeled data points from the labeled ones. To do this, we compute edge weights $w_{i,j}$ based on a given metric M and build a k-nn graph $\mathcal{G}$ with edge weights

$$w_{i,j} = \begin{cases} \exp(-(\mathbf{x}_i - \mathbf{x}_j)^T M (\mathbf{x}_i - \mathbf{x}_j)) & \text{if } j \in \mathcal{N}(i) \\ 0 & \text{otherwise,} \end{cases} \quad (1)$$

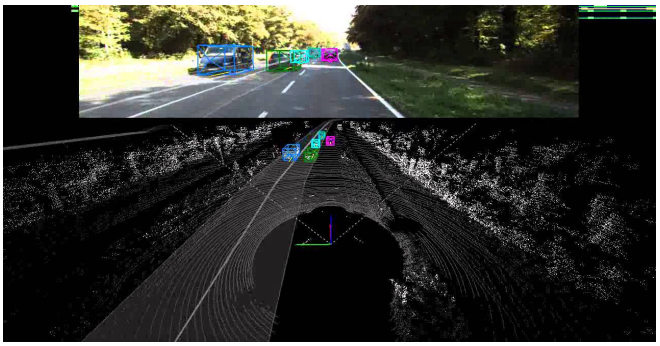

Fig. 1: An example image (top) and the corresponding reconstructed 3D environment (bottom) with found objects from the KITTI Vision Benchmark Suite Geiger et al. [13].

where by $j \in \mathcal{N}(i)$ we denote that j is a neighbor of i. In particular, we find the k_l labeled and k_u unlabeled nearest neighbors of i. We define the *weight matrix* of all $w_{i,j}$ with $W \in \mathbb{R}^{n \times n}$, the diagonal matrix $D = \text{diag}(d_1, \ldots, d_n)$ where $d_i = \sum_{j=1}^{n} w_{i,j}$, and the *transition matrix* $P = D^{-1}W$. Labels are represented in a *label matrix* $F \in \mathbb{R}^{n \times C}$, where ideally entry $f_{k,c} = 1$ if $\mathbf{x}_k$ has label $y_k = c$ and $f_{k,c} = 0$ otherwise. From the convention that all labeled points appear first and all unlabeled ones afterwards, it follows that P consists of four blocks:

$$P = \begin{pmatrix} P_{LL} & P_{LU} \\ P_{UL} & P_{UU} \end{pmatrix}, \quad (2)$$

where the subscripts of the blocks indicate transitions between labeled, unlabeled and mixed point pairs. Similarly, F consists of a labeled block F_L - the one-hot encoding of the true labels - and an unlabeled block F_U, i.e. $F = (F_L^T \; F_U^T)^T$, where F_U initially contains zero vectors.

With this notation, the LP algorithm introduced by Zhu and Ghahramani [1] can be formalized in two steps that are repeated until convergence. In the first step, a new label matrix F^{new} is computed by propagating the given labels according to the transitions, i.e. $F^{new} \leftarrow PF^{old}$. In the second step, the labels of the labeled samples are reset to those from the ground truth, i.e. $F_L^{new} \leftarrow F_L$. A possible convergence criterion is whether the norm $\|F_U^{new} - F_U^{old}\|_\infty$ drops under a given threshold τ.

B. Convergence on Partially Connected Graphs

Zhu and Ghahramani [1] prove the convergence of the LP algorithm but their proof (and the equivalent proof of Zhu [12]) only applies if there is a $\gamma < 1$ s.th. $\forall i \in \{1, ..., u\}$: $\sum_{j=1}^{u} (P_{UU})_{i,j} \leq \gamma$, which means that each unlabeled node has to be connected to a labeled node by an edge of weight greater than 0. This assumption is of course true for fully connected graphs with positive weights. We will show here convergence on not fully connected graphs. First we assume that each connected component of nodes contains at least one labeled node. The LP algorithm can be written as

$$\begin{aligned} F_U^{t+1} &= P_{UU} F_U^t + P_{UL} F_L^t, \\ F_L^{t+1} &= F_L^0 \; \forall t\,. \end{aligned} \quad (3)$$

Obviously F_L stays constant. To show the convergence of F_U, we show the convergence of the columns of F_U. Let f_U be the j-th column of F_U and f_L the j-th column of F_L, $j \in \{1, \dots, C\}$. We use the Banach fixed-point theorem to show that the sequence defined by $f_U^{t+1} = P_{UU} f_U^t + P_{UL} f_L$ converges independently of the starting point f_U^0 to a fixed point. Let us define $T : [0,1]^u \to \mathbb{R}^u$ as $T(x) = P_{UU}x + P_{UL}f_L$. It is easy to show that $T([0,1]^u) \subseteq [0,1]^u$ since P and F are both row stochastic. Thus, we need to prove the following

Theorem 1: The mapping $T : x \mapsto P_{UU}x + P_{UL}f_L$ is a contraction.

Proof:

As a first step, we show that $\rho(P_{UU}) = \max_i |\lambda_i| < 1$ where λ_i are the eigenvalues of P_{UU}. Let λ be an eigenvalue of P_{UU} and $v \in \mathbb{R}^u$ a corresponding eigenvector. W.l.o.g. we can assume that

$$\|v\|_\infty = 1 \;\Rightarrow\; |v_j| \leq 1 \; \forall j. \tag{4}$$

Using the definition of P, one can show that $\|P_{UU}v\|_\infty \leq 1$ and therefore it must hold that $|\lambda| \leq 1$ because for $|\lambda| > 1$ we would get the contradiction $\|P_{UU}v\|_\infty = |\lambda| \|v\|_\infty > \|v\|_\infty = 1$. It remains to show that $|\lambda| \neq 1$: Assume that $|\lambda| = 1$. We define the set $\mathcal{K} = \{k \in \{1, ..., u\} : |v_k| = 1\}$ that is not empty since $\|v\|_\infty = 1$. With a short proof one can show that

$$\forall\, i \in \mathcal{K},\; j \in \{1, ..., u\} \setminus \mathcal{K} :\; w_{i+l,j+l} = 0. \tag{5}$$

Let us define the set $\tilde{\mathcal{K}} = \{j \in \{l+1, \dots, l+u\} \mid j - l \in \mathcal{K}\}$. $\tilde{\mathcal{K}}$ and $\mathcal{U} \setminus \tilde{\mathcal{K}}$ are separate connected components of unlabeled nodes and each of them contains at least one labeled node by assumption. It follows that

$$\exists r \in \tilde{\mathcal{K}}, s \in \mathcal{L},\; \text{s.th. } w_{r,s} > 0 \Rightarrow \frac{\sum_{j=1}^{u} (W_{UU})_{r-l,j}}{\sum_{k=1}^{n} w_{r,k}} < 1 \tag{6}$$

With that we get

$$\begin{aligned} 1 =& |v_{r-l}| = |\lambda||v_{r-l}| = |(\lambda v)_{r-l}| = |\,(P_{UU}v)_{r-l}\,| = |\left(\left(D^{-1}W\right)_{UU} v\right)_{r-l}| \\ =& |\sum_{j=1}^{u} \frac{w_{r,j+l}}{\sum_{k=1}^{n} w_{r,k}} v_j| \leq \frac{1}{\sum_{k=1}^{n} w_{r,k}} \sum_{j=1}^{u} w_{r,j+l}|v_j| \leq \frac{\sum_{j=1}^{u} w_{r,j+l}}{\sum_{k=1}^{n} w_{r,k}} < 1\,. \end{aligned} \tag{7}$$

Because of this contradiction, the assumption $|\lambda| = 1$ must have been wrong and therefore we get $\rho(P_{UU}) < 1$. It immediately follows that there is an $\epsilon > 0$ s.th. $\rho(P_{UU}) + \epsilon < 1$. Now, we define a special vector norm $\|.\|_\omega$ and induced matrix norm $|||.|||_\omega$ s.th. $|||P_{UU}|||_\omega \leq \rho(P_{UU}) + \epsilon$. With that it follows that T is a contraction (with Lipschitz constant $\rho(P_{UU}) + \epsilon$) in the normed space $(\mathbb{R}^u, \|.\|_\omega)$. For details, see Appendix. ■

So far we assumed that each connected component contains at least one labeled node. We can extend the proof also for graphs that do not satisfy this assumption. The nodes in connected components that do not contain a labeled node (isolated nodes) keep their label vector equal to the zero vector throughout the algorithm. The steps performed then are equivalent to removing the isolated nodes from the graph, performing LP on the remaining nodes and assigning all isolated nodes a zero label vector. And for graphs without isolated nodes, we have proven convergence already.

Algorithm 1 Incremental Label Propagation (ILP)

Require:
data set $\mathcal{D}$ of size n; new data sample $\mathbf{x}_{n+1}$;
metric M; graph $\mathcal{G}$; labels F; transitions P;
number of neighbors k_l, k_u
thresholds ϑ, T_{max}
Ensure: updated label matrix F_U;

$\mathbf{w}_{n+1} \leftarrow$ ComputeWeights($\mathcal{D}, \mathbf{x}_{n+1}, M, k_l, k_u$) ▷ Eq. (1)
$P \leftarrow$ UpdateTransition($W, \mathbf{w}_{n+1}$)
$\mathbf{f}_{n+1} \leftarrow P_{n+1,1:n} \cdot F$ ▷ Estimation
$\mathcal{Q} \leftarrow \{k \mid k \in \mathcal{N}^T(n+1) \cap \mathcal{U}\}$
$\tilde{F}_U \leftarrow F_U$
for $k \in \mathcal{Q}$ **do**
  $\tilde{F}_{U(k)} \leftarrow P_{UL(k)}F_L + P_{UU(k)}F_U$
end for
$t_i \leftarrow 0$
while $\mathcal{Q} \neq \emptyset$ and $t_i < T_{max}$ **do**
  $\mathcal{A}, F_U \leftarrow$ FilterAndUpdate($\mathcal{Q}, \tilde{F}_U, F_U, \vartheta$)
  $\mathcal{Q}, \tilde{F}_U \leftarrow$ GetNextCandidates($\mathcal{A}, \tilde{F}_U, P_{UU}$)
  $t_i \leftarrow t_i + 1$
end while

function FilterAndUpdate($\mathcal{Q}, \tilde{F}_U, F_U, \vartheta$)
  $\mathcal{A} \leftarrow \emptyset$
  for $i \in \mathcal{Q}$ **do**
    $\delta\mathbf{f}_i \leftarrow \tilde{F}_{U(i)} - F_{U(i)}$
    if $|\delta\mathbf{f}_i| > \vartheta$ **then** ▷ Filter
      $F_{U(i)} \leftarrow \tilde{F}_{U(i)}$ ▷ Update
      $\mathcal{A} \leftarrow \mathcal{A} \cup (i,\ \delta\mathbf{f}_i)$
    end if
  end for
  return $\mathcal{A}, F_U$
end function

function GetNextCandidates($\mathcal{A}, \tilde{F}_U, P_{UU}$)
  $\mathcal{Q} \leftarrow \emptyset$
  for $(j, \delta\mathbf{f}_j) \in \mathcal{A}$ **do**
    for $k \in \{i \mid i \in \mathcal{N}^T(j) \cap \mathcal{U}\}$ **do**
      $\tilde{F}_{U(k)} \leftarrow \tilde{F}_{U(k)} + P_{UU(k,j)}\delta\mathbf{f}_j$
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup k$
    end for
  end for
  return $\mathcal{Q}, \tilde{F}_U$
end function

C. Incremental Label Propagation

Our main idea is that when a new sample arrives, many computation steps can be saved by propagating labels only locally and stopping the propagation process if no significant change of labels is achieved. The latter is formulated by introducing a *threshold* ϑ and a decision function that returns only indices of data samples, for which the label change is larger than ϑ according to some norm (we use the ℓ_1-norm). Another difference to standard LP is that we formulate the algorithm based on a set $\mathcal{Q}$ of candidate nodes to propagate labels to and a set $\mathcal{A}$ of nodes which actually get updated.

This leads to a simple formulation of the stopping criterion, since label propagation is stopped when $\mathcal{Q}$ becomes empty [1].

The pseudocode for Incremental Label Propagation is given in Alg. 1. We denote by $F_{U(i)}$ the i-th row of matrix F_U, namely the estimated label for point i. Here, we give details for the individual steps. First, we compute the edge weights $\mathbf{w}_{n+1} = (w_{n+1,1}, \ldots, w_{n+1,n})$ between the new data sample $\mathbf{x}_{n+1}$ and the observed samples $\mathbf{x}_1, \ldots, \mathbf{x}_n$ according to (1). This we use to update the matrices W, D, and P, which then have $n+1$ rows and columns. With P, we estimate a label vector $\mathbf{f}_{n+1}$ for the new node as a weighted average of the labels of its labeled and unlabeled neighbors (line 3). After that, we initialize the candidates set $\mathcal{Q}$, which at first contains only the nodes from the unlabeled set $\mathcal{U}$, that have the new node as a nearest neighbor. We denote these reverse nearest neighbors of sample $\mathbf{x}_{n+1}$ as $\mathcal{N}^T(n+1)$. We also initialize the *tentative* label updates matrix $\tilde{F}_U$ (line 5). (In practice we only need to store the rows of $\tilde{F}_U$ that get updated during the algorithm.) In lines 6 to 8, the label propagation starts by computing the tentative labels of the reverse nearest neighbors of the new node. This is the only step where information from the transition submatrix P_{UL} is used.

In lines 10 to 14, the main idea of the ILP algorithm is presented. From the set $\mathcal{Q}$ of candidates that received a tentative label update, we identify the ones whose update is considered *significant*. In particular, the ℓ_1-distance between their previous and tentative label has to be larger than ϑ. Note that ϑ implicitly defines the area of influence of the algorithm: for $\vartheta = 0$ all changes are considered significant and we have offline LP with sets starting from the new node. Any other value for ϑ constrains the range of the region growing process, leading to a more local effect. We apply the significant tentative updates by storing the new labels in the original matrix F_U (line 6 of function *FilterAndUpdate*). In set $\mathcal{A}$ we store the indices of the updated nodes along with their label difference $\delta\mathbf{f}$, so we can continue the propagation with their reverse neighbors in the next iteration.
The ILP algorithm presented in Alg. 1 for new unlabeled points is almost identical for new labeled points with the only difference that $\mathbf{f}_{n+1}$ does not need to be estimated.

IV. RUNTIME ANALYSIS

We consider the insertion of a single node. We denote by k_l and k_u the number of labeled and unlabeled neighbors of each node in the graph respectively. The main burden of the computation of the node's edge weights lies in the distance computation between the new node and the existing nodes in the graph which in turn depends on the choice of nearest neighbor algorithm. The trivial computation takes $O(n_t d)$ where d is the data dimensionality and $n_t = l_t + u_t$ is the number of nodes at iteration t. This can be reduced to logarithmic time using an efficient data structure for online node insertion and nearest neighbor search, such as Ball Trees as described in the online insertion algorithm of Omohundro [14]. We denote this runtime as O_{knn}.

A major advantage of using a k-nn as opposed to an ε-nn graph in an incremental setting is that updating the old entries of W and P is independent of the number of nodes n_t. Using a fixed capacity heap for storing the adjacency list of each node, a node's labeled neighbors can be updated in $\log(k_l)$ and equivalently unlabeled neighbors in $\log(k_u)$, as only one neighbor might get pushed out of the heap to be replaced by the new node. Therefore the weight matrix can be updated in $O((k_l + k_u)\log(\max(k_l, k_u)))$. The update of P is done in $O((k_l + k_u)^2)$ since only the rows of nodes connected to the new point must be updated.

The estimation of the new label takes $O((k_l + k_u)C)$ time. Accessing the neighbors and reverse neighbors of points can be done in constant time using a hash map at the cost of $O(n)$ auxiliary space. Each inner iteration in the functions *GetNextCandidates* and *FilterAndUpdate* takes $O(C)$ time, therefore the runtime depends mainly on how the sizes of $\mathcal{Q}$ and $\mathcal{A}$ evolve. The size of $\mathcal{A}$ is always bounded by the size of $\mathcal{Q}$ but the size of $\mathcal{Q}$ depends on the number of reverse neighbors, which is not deterministic. In practice though this is close to ${k_u}^2$. With this assumption and with $\vartheta = 0$, $\mathcal{Q}$ and $\mathcal{A}$ grow as $k_u, k_u^2, k_u^3, \ldots$, etc., as we consider all label updates significant. Therefore, the number of iterations until all unlabeled points have been reached by label propagation is bounded by $\log_{k_u}(u_t)$. The total number of operations in the main loop then is

$$C(k_u + k_u^2 + \ldots + k_u^{\log_{k_u} u_t}) = C\frac{u_t - 1}{k_u - 1} = O(u_t C\frac{1}{k_u - 1}). \quad (8)$$

And thus the total runtime of incremental label propagation for a node arriving at time t is

$$O_{knn} + O((k_l + k_u)^2 + (k_l + k_u)C + u_t C\frac{1}{k_u - 1}). \quad (9)$$

V. EXPERIMENTS

A. Evaluation and Setup

In this work, we use the standard Euclidean metric ($M = I$) to compute the edge weights (See Eq.1). For evaluation we use several metrics: the ℓ_1 error, the 0-1 classification error and the cross-entropy between ground truth labels and the predictions F_U. We also compute the entropy of our predictions F_U, the number of label propagation iterations per new point and the wall-clock computation time per new point. We set $T_{max} = \log_{k_u} u_t$ if $k_u > 1$ and $T_{max} = 30$ if $k_u = 1$.[3]

We first examine the influence of different hyperparameters and the performance of ILP against its fully supervised counterpart, using the MNIST dataset of handwritten digits [15] that consists of 60000 training and 10000 test images of dimensions 28 by 28. We do not compute any features but use the raw pixel values normalized to lie in $[0, 1]$.

[1]Note that the use of the sets $\mathcal{Q}$ and $\mathcal{A}$ alone does not turn LP into an incremental algorithm. In fact, for $\vartheta = 0$, ILP with sets is equivalent to offline LP for the observed samples at the time. The insight is, that for any point, only its reverse neighbors can change their label in every LP iteration, i.e. the graph can be equally processed using a region-growing strategy.

[2]One can enforce the number of reverse neighbors to be equal to k_u by using a *mutual* instead of a regular k-nn graph.

[3]Code available at
`https://github.com/johny-c/incremental-label-propagation`.

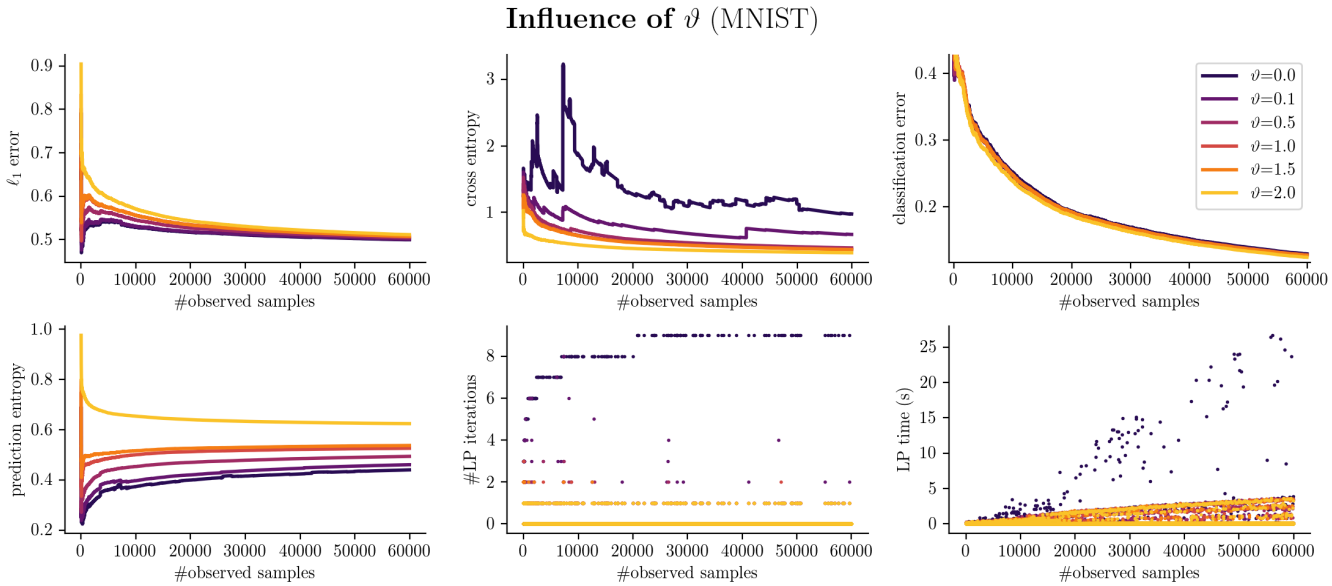

Fig. 2: Illustration of how ϑ influences different metrics. **Top row**: ℓ_1 error, cross entropy and classification error (based on arg max) w.r.t. the predictions F_U. **Bottom row**: Entropy of the predictions, number of label propagation iterations and actual computation time after each new node.

Due to lack of space, we present results with $k_u = k_l = 3$. However, we found the algorithm to be very robust with respect to values of k_u and k_l up to 19, except for the case of $k_l = 1$, where the final accuracy dropped by 5%.

B. Influence of ϑ

The ratio of observed labels is fixed to 5% and we set $k_l = 3$ and $k_u = 3$. In Fig. 2 we show how different metrics are affected by ϑ. As expected, the ℓ_1-error decreases as we decrease ϑ, namely as we move closer to offline LP. On the other hand, the number of iterations per observation and therefore the runtime decreases when we increase ϑ, as the algorithm is constrained to act more locally. For reference, with $\vartheta = 0$, the total runtime was 4162.40s, while with $\vartheta = 1.0$ only 3143.46s. We observe several interesting facts:

- The estimation error is robust to changes in ϑ, as the max. likelihood of the correct predictions is mostly preserved.
- The entropy of the predicted label distributions behaves very similar to the ℓ_1-error making it a good candidate metric for self-evaluation/introspection during learning.
- From the cross-entropy evaluation, it seems that choosing a large enough ϑ has a regularizing effect: When ϑ is small, the cross-entropy is strongly oscillating, as every label update is trusted equally. When ϑ is larger, the "propagation region" is narrower and therefore cross-entropy is smoother. In fact for $\vartheta = 2$, where we only have label estimation and no propagation, the cross-entropy and often also the estimation error is minimized. However, when testing on a holdout set, we find that intermediate values of ϑ achieve the best accuracy. This suggests that a too small ϑ causes overfitting, and a ϑ that is too large causes underfitting.

C. Influence of number of observed labels

In this experiment we evaluate how the accuracy of the algorithm is affected by the number of observed labels (Fig. 3). We compare the test error when using just the labeled samples against the test error when also using the labels estimated by ILP for the unlabeled training samples. We use $k_l = 3, k_u = 3$ and $\vartheta = 0.3$. Predicting the labels of the test set amounts to computing $F_T = P_{TL}F_L + P_{TU}F_U$. Taking into account only the first summand $P_{TL}F_L$ is weighted k-nn querying with respect to the given labeled samples. In Table I we show that ILP is consistently better than weighted k-nn, demonstrating the utilization of the unlabeled samples.

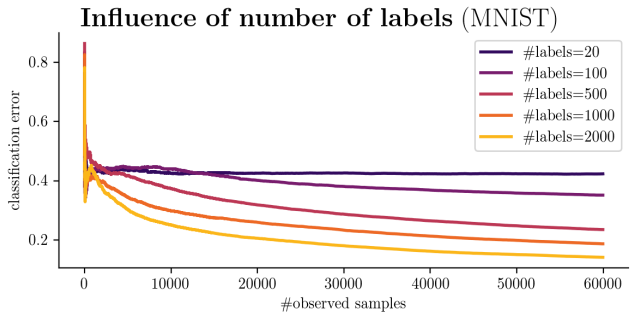

Fig. 3: Estimation error on the observed unlabeled samples of MNIST for different number of observed labels in total.

D. Confusion Analysis

In continuation of the previous experiment, we trained incrementally on MNIST observing only 5% of the labels. In Fig. 4 we show the final confusion matrices of k-nn, where only labeled neighbors vote for the label of test nodes, and

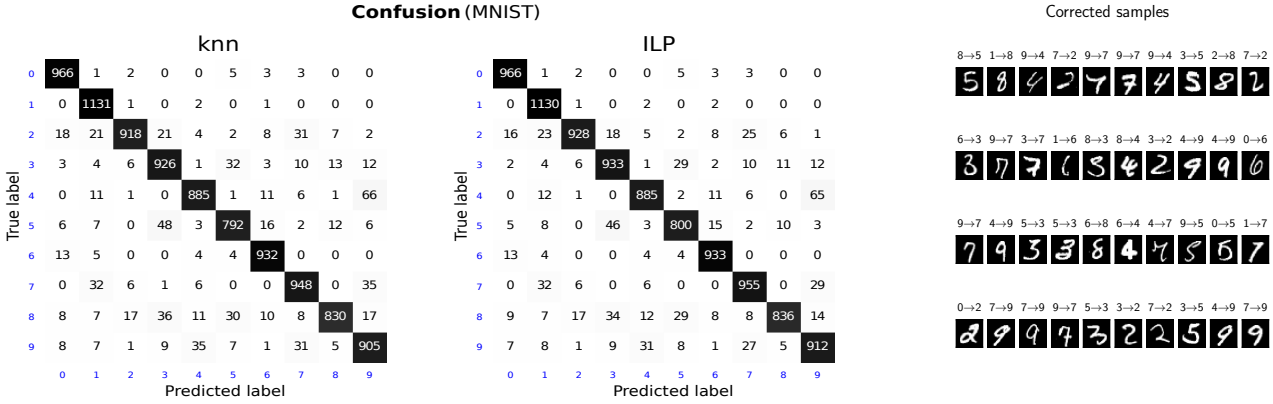

Fig. 4: Confusion matrices for the test set of MNIST. **Left:** Predicting only with 3 labeled neighbors (test error=7.67%), **Middle:** Predicting with ILP, taking into account also the estimated labels of 3 unlabeled neighbors (test error=7.22%). **Right:** Some samples for which the label predictions were corrected by ILP.

#Labels	Est. error (%)	*k*nn error (%)	ILP error (%)
20	42.35	41.17	40.11
100	35.17	27.01	26.61
500	23.54	14.54	14.09
1000	18.73	11.27	10.68
2000	14.17	8.56	8.2

TABLE I: Final estimation error on the unlabeled training set, *k*nn error and ILP error on the test set of MNIST.

ILP, where the estimated labels of the unlabeled training nodes are also considered. We present examples of images misclassified by *k*-nn that were correctly classified by ILP.

E. KITTI Benchmark

We further evaluate Incremental Label Propagation on the more challenging setting of a *stream* of data. For this experiment we use data from the KITTI benchmark [4]. We conduct the same experiment as Narr et al. [16] [5]. We concatenate 18 streams of segmented 3D point clouds from urban traffic environments [13] to form one long stream. Each of the 25090 segments corresponds to a 3D bounding box containing points that represent a given object candidate. For each such candidate, a 60-dimensional feature vector was computed as proposed by Himmelsbach et al. [17]. These features consist of global characteristics such as box volume and mean intensity, as well as of distributions of local ones such as scatterness or flatness. These features can be computed in real time. For the test data, each of the 18 subsets was split at a 2:1 ratio to obtain a stream of 16000 training samples and a set of 9,090 test samples. We used the 100 first training samples (50 labeled and 50 unlabeled) as a "burn-in" set to initialize the algorithm. In Fig. 6, we show the order of appearance of labeled and unlabeled samples from each class in the described training stream, as well as the behavior of ILP when only 10% of the labels are observed.

[4] `http://www.cvlibs.net/datasets/kitti/`

[5] Data were provided by the authors upon our request.

The methods mentioned by Narr et al. [16] were evaluated on the test set after every 1000 observations and their final test error is presented in Table II. We note that we do not compare against offline methods, as they are not relevant in a stream-based learning scenario. We fixed ϑ to 1.0 and ran the experiment for different ratios of given labels (See Fig. 5). With only 5% of the labels, ILP already outperforms Online Random Forests (Saffari et al. [18]) and online multi-class Gradient Boost (Saffari et al. [19]) and achieves comparable accuracy to Mondrian Forests (Lakshminarayanan et al. [20]). Note that all other methods use all training labels. With 20% of the labels, we achieved a test error of 9.53%, which outperforms even the Mondrian Forest method.

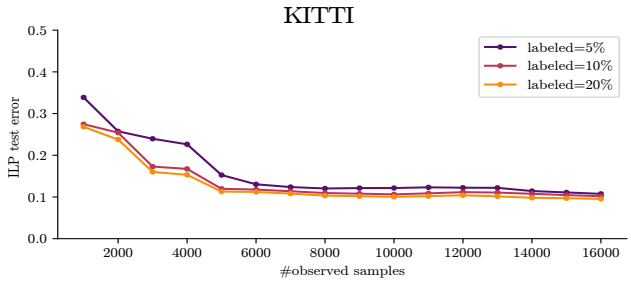

Fig. 5: Test error of ILP over time for different ratios of labeled data in the stream.

Method	Final Test error(%)
OMCGB (Saffari et al. [19]	13.00
ORF (Saffari et al. [18])	13.70
MF (Lakshminarayanan et al. [20])	10.00
ILP (5% labels)	10.75
ILP (10% labels)	10.18
ILP (20% labels)	**9.53**

TABLE II: Final test error of different online learning methods for the described experiment on the KITTI benchmark.

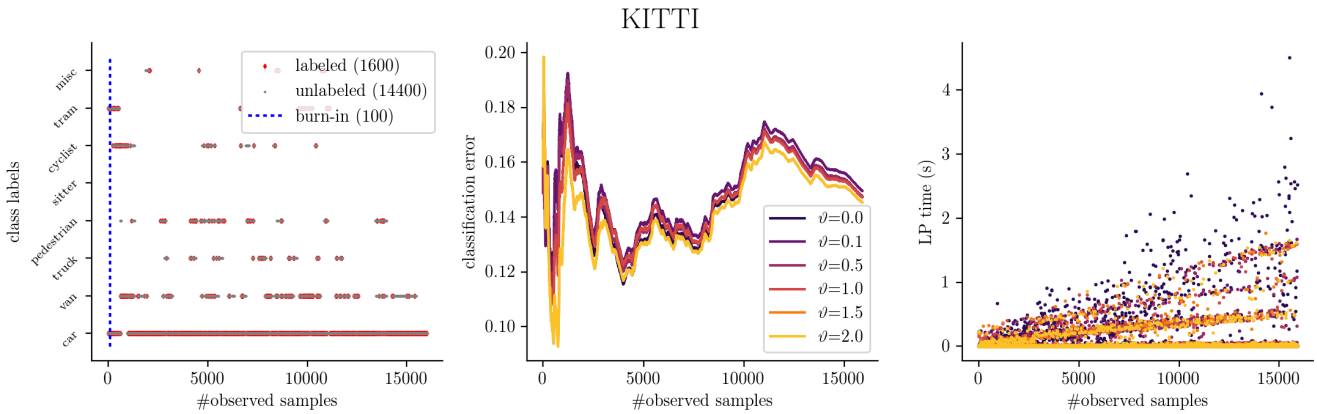

Fig. 6: Results on the constructed stream from the KITTI dataset given 10% of labels (see text). **Left:** The order of appearance of labels in the stream, **Middle:** ILP error on the unlabeled training set, **Right:** Computation time for different values of ϑ.

VI. CONCLUSIONS

In this paper we presented "Incremental Label Propagation", an efficient incremental variant of the Label Propagation algorithm introduced by Zhu and Ghahramani [1] that is useful for object classification from sparsely labeled streams of data. We provided a proof of convergence of the original algorithm for partially connected graphs and gave an analysis for the runtime of our algorithm. With various experiments we investigated the influence of hyperparameters on the behavior of ILP, showed the utilization of the unlabeled samples over its purely supervised counterpart and demonstrated the performance improvement, even over fully supervised online learning methods on a challenging benchmark dataset.

APPENDIX

To get a suitable norm on $\mathbb{R}^u$ to show that T is a contraction we use lemma 5.6.10 of Roger A. Horn [21]. It states that for any matrix $A \in \mathbb{R}^{n\times n}$ and $\epsilon > 0$ there is a matrix norm $|||.|||_\omega$, s.th. $|||A|||_\omega \leq \rho(A) + \epsilon$. This matrix norm is constructed as follows: Because of theorem 2.3.1 in [21] there is a unitary matrix $Z \in \mathbb{R}^{n\times n}$ and an upper triangular matrix $\Delta \in \mathbb{R}^{n\times n}$ s.th. $A = U\Delta U^*$. Let be $D_t = diag(t, t^2, t^3, ..., t^n)$ and $Q := D_t U^*$. Then the matrix norm we are interested in is defined by

$$|||A|||_\omega := |||D_t U^* A U D_t^{-1}|||_1 = |||\left(D_t U^*\right) A \left(D_t U^*\right)^{-1}|||_1$$
$$\text{where } |||A|||_1 := \max_{1\leq j\leq n} \sum_{i=1}^{n} |a_{i,j}|\,. \tag{10}$$

Example 5.6.4 in [21] shows that the norm $|||.|||_1$ is induced by the ℓ_1-vector-norm $\|x\|_1 = \sum_{i=1}^n |x_i|$ on $\mathbb{R}^n$. Following theorem 5.6.7 in [21] the matrix norm $|||.|||_\omega$ is then induced by the vector-norm $\|x\|_\omega = \|D_t U^* x\|_1$. As shown in the proof of lemma 5.6.10 in [21], for t large enough the matrix norm $|||.|||_\omega$ fulfills $|||A|||_\omega \leq \rho(A) + \epsilon$.

REFERENCES

[1] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," Carnegie-Mellon Univ., Pittsburgh, Tech. Rep. CMU-CALD-02-107, 2002.

[2] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1995.

[3] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling, "Semi-supervised learning with deep generative models," in *Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 3581–3589.

[4] M. Sajjadi, M. Javanmardi, and T. Tasdizen, "Regularization with stochastic transformations and perturbations for deep semi-supervised learning," in *NIPS*, 2016, pp. 1163–1171.

[5] P. Haeusser, A. Mordvintsev, and D. Cremers, "Learning by association - a versatile semi-supervised training method for neural networks," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[6] O. Chapelle, B. Schölkopf, and A. Zien, *Semi-Supervised Learning*. The MIT Press, 2006.

[7] X. Zhu, Z. Ghahramani, J. Lafferty *et al.*, "Semi-supervised learning using gaussian fields and harmonic functions," in *ICML*, vol. 3, 2003, pp. 912–919.

[8] G. Ganu and B. Kveton, "Nearly optimal semi-supervised learning on subgraphs," 2013.

[9] O. Delalleau, Y. Bengio, and N. Le Roux, "Efficient non-parametric function induction in semi-supervised learning." in *AISTATS*, vol. 27, 2005, p. 100.

[10] M. Valko, B. Kveton, H. Ling, and T. Daniel, "Online semi-supervised learning on quantized graphs," in *in UAI*. Citeseer, 2010.

[11] M. Charikar, C. Chekuri, T. Feder, and R. Motwani, "Incremental clustering and dynamic information retrieval," *SIAM Journal on Computing*, vol. 33, no. 6, pp. 1417–1440, 2004.

[12] X. Zhu, "Semi-supervised learning with graphs," Ph.D. dissertation, Language Technologies Institute, School of Computer Science, Carnegie Mellon University, 2005.

[13] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[14] S. M. Omohundro, *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[16] A. Narr, R. Triebel, and D. Cremers, "Stream-based active learning for efficient and adaptive classification of 3d objects," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 227–233.

[17] M. Himmelsbach, T. Luettel, and H.-J. Wuensche, "Real-time object classification in 3d point clouds using point feature histograms," in *IEEE/RSJ IROS*, 2009, pp. 994–1000.

[18] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof, "On-line random forests," in *Computer Vision Workshops at International Conference on Computer Vision (ICCV)*, 2009, pp. 1393–1400.

[19] A. Saffari, M. Godec, T. Pock, C. Leistner, and H. Bischof, "Online multi-class lpboost," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010, pp. 3570–3577.

[20] B. Lakshminarayanan, D. M. Roy, and Y. W. Teh, "Mondrian forests: Efficient online random forests," in *Adv. in Neural Inf. Processing Systems (NIPS)*, 2014, pp. 3140–3148.

[21] C. R. J. Roger A. Horn, *Matrix analysis*. New York: Cambridge University Press, 2013.