Re-revisiting Learning on Hypergraphs: Confidence Interval and Subgradient Method

Chenzi Zhang [*1] **Shuguang Hu** [*1] **Zhihao Gavin Tang** [1] **T-H. Hubert Chan** [1 2]

Abstract

We revisit semi-supervised learning on hypergraphs. Same as previous approaches, our method uses a convex program whose objective function is not everywhere differentiable. We exploit the non-uniqueness of the optimal solutions, and consider confidence intervals which give the exact ranges that unlabeled vertices take in any optimal solution. Moreover, we give a much simpler approach for solving the convex program based on the subgradient method. Our experiments on real-world datasets confirm that our confidence interval approach on hypergraphs outperforms existing methods, and our sub-gradient method gives faster running times when the number of vertices is much larger than the number of edges.

1. Introduction

Given a dataset, similarity relationship between examples can be represented by a graph in which each example is represented by a vertex. While pairwise relationship between two vertices can be represented by an edge in a normal graph, a higher order relationship involving multiple vertices can be captured by a hyperedge, which means that all the corresponding examples are similar to one another. Hypergraphs have been used in several learning applications such as clustering of categorical data (Gibson et al., 1998), multi-label classification (Sun et al., 2008), Laplacian sparse coding (Gao et al., 2013), image classification (Yu et al., 2012), image retrieval (Huang et al., 2010), mapping users across different social networks (Tan et al., 2014) and predicting edge labels in hypernode graphs (Ricatte et al., 2014).

[*]Equal contribution [1]University of Hong Kong, Hong Kong. [2]This research was partially supported by the Hong Kong RGC under the grant 17200214. Correspondence to: T-H. Hubert Chan <hubert@cs.hku.hk>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

In this paper, we consider semi-supervised learning on an edge-weighted hypergraph $H = (V, E, w)$, with a set L of labeled vertices, whose labels are given by $f_L^* \in \{-1, +1\}^L$. The task is to predict the labels of the unlabeled vertices N, with the working principle that vertices contained in a hyperedge $e \in E$ are more similar to one another if the edge weight w_e is larger. This problem is also known as *transductive inference* and has been studied in (Zhou et al., 2006) and (Hein et al., 2013).

However, the methods in (Zhou et al., 2006) have been criticized by (Agarwal et al., 2006), because essentially a hypergraph is converted into a normal graph. For instance, given a hyperedge e containing vertices S, either (i) a clique is added between the vertices in S, or (ii) a star is formed by adding a new vertex v_e connecting every vertex in S to v_e. Then, a standard convex program using a regularization potential function for normal graphs can be applied (Zhu et al., 2003). By choosing appropriate edge weights, it was shown in (Agarwal et al., 2006) that the two approaches are equivalent to the following convex program relaxation:

$$\begin{aligned} \min \quad & \Phi_{old}(f) := \frac{1}{2} \sum_{e \in E} w_e \sum_{\{u,v\} \in \binom{e}{2}} (f_u - f_v)^2 \\ \text{subject to} \quad & f_u \in [-1, 1], \quad \forall u \in V \\ & f_u = f_u^*, \quad \forall u \in L. \end{aligned}$$

On the other hand, it was proposed in (Hein et al., 2013) that the following regularization function is more suitable to capture hyperedge expansion:

$$\Phi_{new}(f) := \frac{1}{2} \sum_{e \in E} w_e \cdot (\max_{u \in e} f_u - \min_{v \in e} f_v)^2.$$

Indeed, it was shown in (Hein et al., 2013) that their approach outperforms (Zhou et al., 2006) on several datasets from the UCI Machine Learning Repository (Lichman, 2013).

Loss Function. In (Hein et al., 2013), a squared loss function was added by considering the convex program with objective function $\Phi_{new}(f) + \mu \|f - f^*\|_2^2$ on $f \in [-1, 1]^V$, where $\mu > 0$ is a parameter to be tuned, f_L^* is given by the labeled vertices L, and for the unlabeled vertices $f_N^* = \mathbf{0}$.

The loss function allows errors in the labeled vertices, and also ensures that the minimizer is unique. However, as a result, unlabeled vertices have a tendency to acquire f values close to 0. This might remove useful information as illustrated in the following example.

Example. In Figure 1.1, vertices $a, b \in L$ are labeled as $+1$ and $c \in L$ is labeled as -1. Vertices $x, y \in N$ are unlabeled. There are three (undirected) edges: $\{a, x\}$, $\{b, x\}$ and $\{x, y, c\}$, each with unit weight.

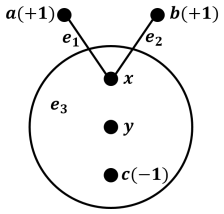

Figure 1.1. Example

By choosing $\mu = \frac{1}{2}$ for squared loss function, the unique minimizer gives $f_x = \frac{1}{5}$ and $f_y = 0$. Hence, this solution gives no useful information regarding the label for vertex y.

On the other hand, if we just use the objective function $\Phi_{new}(f)$ with the constraints $f_L = f_L^*$, then in an optimal solution, $f_x = \frac{1}{3}$, but f_y could be anywhere in the *confidence interval* $[-1, \frac{1}{3}]$. Hence, in this case, we could use the average value $-\frac{1}{3}$ to predict -1 for vertex y.

Our Contributions. In this paper, we revisit the approach used in (Hein et al., 2013) and consider several extensions and simplifications. We summarize our results and give an outline of the paper as follows.

1. Unified Framework for Directed Hypergraphs. Inspired also from the recent result on Laplacians for directed normal graphs (Yoshida, 2016), we introduce a semi-supervised learning framework using directed hypergraphs that can capture higher order *causal* relationships. This notion of directed hypergraph was first introduced in (Gallo et al., 1993), who considered applications in propositional logic, analyzing dependency in relational database, and traffic analysis. On a high level, a directed hyperedge e consists of a *tail* set T_e pointing to a *head* set H_e such that a vertex in T_e labeled $+1$ implies that a vertex in H_e is more likely to be labeled $+1$. (Equivalently in terms of its contrapositive, a vertex in H_e labeled -1 implies that a vertex in T_e is more likely to be labeled -1.) In Section 2, we formally define the model and the corresponding potential function Φ. An additional advantage of our potential function is that there is no need to tune any parameters.

2. Confidence Interval for Unlabeled Vertices. Observe that the minimizer for our convex program might not be unique. In Section 3, we introduce the concept of *confidence interval* for each unlabeled vertex that can be useful for predicting its label. Furthermore, we provide an algorithm to calculate the confidence interval given an optimal solution.

3. Simpler Subgradient Method. Since the new potential function is not everywhere differentiable but still convex, we use the subgradient method (Shor et al., 1985) to obtain an estimated minimizer for label prediction. Inspired by the diffusion processes used for defining Laplacians in hypergraphs (Louis, 2015) and directed graphs (Yoshida, 2016), in Section 4, we define a simple *Markov* operator that returns a subgradient for Φ, which is used to solve the underlying convex program. We remark that our framework is very easy to understand, because it is a variation on the well-known gradient descent.

In contrast, the primal-dual approach in (Hein et al., 2013) considers the convex conjugate of the primal objective and involves complicated update operations on the primal and dual variables. The subgradient used in our approach gives the update direction, and we can actually solve exactly the same convex program with a much simpler method.

4. Experimental Results on Real-World Datasets. In Section 5, we revisit some datasets in the UCI Machine Learning Repository (Lichman, 2013), and experiments confirm that our prediction model based on confidence interval gives better accuracy than that in (Hein et al., 2013). Our simpler subgradient method takes more iterations than the primal-dual method (Hein et al., 2013), but each iteration is much faster. Experiments show that overall both methods have similar running times, and the subgradient method has an advantage when the number of vertices is much larger than the number of edges.

Moreover, using the DBLP dataset (Ley, 2009), our experiments also support that using directed hypergraphs to capture causal relationships can improve the prediction accuracy. The experiments for directed hypergraphs are described in the full version.

2. Preliminaries

We consider an edge-weighted directed hypergraph $H = (V, E, w)$ with vertex set V (with $n = |V|$), edge set E and weight function $w : E \to \mathbb{R}_+$. Each hyperedge $e \in E$ consists of a *tail* set $\mathsf{T}_e \subseteq V$ and a *head* set $\mathsf{H}_e \subseteq V$ (which are not necessarily disjoint); we use the convention that the direction is from tail to head. For $x \in \mathbb{R}$, we denote $[x]^+ := \max\{x, 0\}$.

In our application, each vertex $v \in V$ is supposed to have a label in $\{-1, +1\}$. Intuitively, the directed hypergraph attempts to capture the rule that for each edge $e \in E$, if there is a vertex in T_e having label $+1$, then it is more likely for vertices in H_e to receive label $+1$. In terms of its contrapositive, if there is a vertex in H_e having label -1, then it is more likely for vertices in T_e to receive label -1.

We use $f \in \mathbb{R}^V$ to denote a vector, where the coordi-

nates are labeled by vertices in V. For $U \subseteq V$, we use $f_U \in \mathbb{R}^U$ to denote the vector restricting f to coordinates in U. In semi-supervised learning, we consider a set $L \subseteq V$ of *labeled* vertices, which have labels $f^*_L \in \{-1,+1\}^L$. Typically, $|L| \ll |V|$ and the task is to assign a label in $\{-1,+1\}$ to each *unlabeled* vertex in $N := V \setminus L$, using information from the directed hypergraph H.

By relaxing labels to be in the interval $[-1,1]$, we consider the following regularization potential function $\Phi : \mathbb{R}^V \to \mathbb{R}$:

$$\Phi(f) = \frac{1}{2}\sum_{e\in E} w_e \cdot ([\Delta_e(f)]^+)^2,$$

where $\Delta_e(f) := \max_{(u,v)\in \mathsf{T}_e\times \mathsf{H}_e}(f_u - f_v) = \max_{u\in \mathsf{T}_e} f_u - \min_{v\in \mathsf{H}_e} f_v$.

In particular, there is a penalty due to edge e only if some vertex in T_e receives a label larger than that of some vertex in H_e. The convexity of Φ is proved in the full version.

Our approach is to consider the following convex program to obtain an estimated minimizer $f \in [-1,1]^V$, which can be rounded to an integer solution for labeling all vertices.

$$\begin{aligned} \min \quad & \Phi(f) && && \text{(CP1)}\\ \text{subject to} \quad & f_u \in [-1,1], && \forall u \in V\\ & f_u = f^*_u, && \forall u \in L \end{aligned}$$

Since the f values for the labeled vertices L are fixed in (CP1), we also view $\Phi : \mathbb{R}^N \to \mathbb{R}$ as a function on the f values of unlabeled vertices N. We use $\mathsf{OPT} \subset \mathbb{R}^V$ to denote the set of optimal solutions to (CP1).

Trivial Edges. An edge $e \in E$ is *trivial* if there exist vertices $u \in \mathsf{T}_e \cap L$ and $v \in \mathsf{H}_e \cap L$ such that $f^*_u = +1$ and $f^*_v = -1$. As trivial edges contribute constant towards the objective function Φ, we shall assume that there are no trivial edges in the convex program (CP1).

Special Cases. Our directed hypergraph model can capture other graph models as follows.

1. **Undirected Hypergraphs.** For each hyperedge e, we can set $\mathsf{T}_e = \mathsf{H}_e$ to the corresponding subset of vertices.
2. **Undirected Normal Graphs.** For each edge $e = \{u,v\}$, we can set $\mathsf{T}_e = \mathsf{H}_e = e$. Observe that in this case, the potential function becomes $\Phi(f) = \sum_{(u,v)\in E} w_{uv}(f_u - f_v)^2$, which is differentiable, and hence, (CP1) can be solved by standard techniques like gradient descent.

Soft Constraints. In (Hein et al., 2013), each labeled vertex $u \in L$ can also have some weight $\mu_u \in \mathbb{R}_+$, which can, for instance, indicate how trustworthy the label $f^*_u \in \{-1,+1\}$ is. The following relaxation is considered.

$$\begin{aligned} \min \quad & \widehat{\Phi}(f) := \Phi(f) + \frac{1}{2}\sum_{u\in L}\mu_u(f_u - f^*_u)^2 && \text{(CP2)}\\ \text{subject to} \quad & f_u \in [-1,1], \forall u \in V. \end{aligned}$$

Observe that (CP2) can also be expressed in the framework of (CP1). We simply consider an augmented hypergraph $\widehat{H}$ such that all vertices V are treated as unlabeled, and for each $u \in L$, we add a new vertex $\widehat{u}$ with label f^*_u and a new undirected edge $\{u,\widehat{u}\}$ with weight μ_u. Then, it follows that the convex program (CP1) for the augmented instance for $\widehat{H}$ is exactly the same as (CP2).

Challenges Ahead. We next outline how we resolve the encountered challenges when we use (CP1) for semi-supervised learning.

- Unlike the case for normal graphs, the set OPT can contain more than one optimal solution for (CP1). In Section 3, we prove some structural properties of the convex program, and illustrate that each $u \in N$ has some *confidence interval* from which we can predict its label.
- The function Φ is not everywhere differentiable. Hence, we use the subgradient method (Shor et al., 1985). In Section 4, we give a method to generate a subgradient, which is inspired by the continuous diffusion processes for hypergraphs (Louis, 2015) and directed graphs (Yoshida, 2016), and our method can in fact be viewed as a discretized version.

3. Confidence Interval for Semi-supervised Learning

In general, a minimizer for (CP1) might not be unique. Hence, we introduce the concept of confidence interval.

Definition 3.1 (Confidence Interval) *For each $u \in V$, we define its confidence interval to be $[m_u, M_u]$, where $m_u := \min_{f\in\mathsf{OPT}} f_u$ and $M_u := \max_{f\in\mathsf{OPT}} f_u$. The confidence intervals induce the lower and the upper confidence vectors, $\vec{m}$ and $\vec{M} \in \mathbb{R}^V$, respectively.*

In Section 3.1, we give the proof of the following lemma, which states that the confidence vectors $\vec{m}$ and $\vec{M}$ are optimal solutions, and so are their convex combinations.

Lemma 3.1 (Confidence Vectors Give Optimal Solutions) *For any $\lambda \in [0,1]$, the convex combination $\lambda\vec{m} + (1-\lambda)\vec{M} \in \mathsf{OPT}$ is optimal for (CP1).*

Semi-supervised Learning via Confidence Interval. Lemma 3.1 suggests what one can do when (CP1) has more than one optimal solution. Specifically, in Algorithm 1, the

average vector $\frac{1}{2}(\vec{m} + \vec{M}) \in \mathsf{OPT}$ can be used for label prediction. We show that the confidence vectors $\vec{m}$ and $\vec{M}$ can be recovered from any optimal solution $f \in \mathsf{OPT}$, which in turn can be estimated by the subgradient method described in Section 4.

Algorithm 1 Semi-Supervised Learning

1: **Input:** Directed hypergraph $H = (V, E, w)$, labels f^*_L for labeled vertices L
2: Compute (estimated) confidence vectors $(\vec{m}, \vec{M}) \in \mathbb{R}^N \times \mathbb{R}^N$, either by Algorithm 2 or 3.
3: Compute average vector $\overline{f}_N \leftarrow \frac{1}{2}(\vec{m} + \vec{M})$.
4: Compute threshold $\theta \leftarrow \frac{1}{|N|} \sum_{u \in N} \overline{f}_u$.
5: **for** each $u \in N$ **do**
6: **if** $\overline{f}_u \geq \theta$ **then**
7: $\widehat{f}_u \leftarrow +1$;
8: **else**
9: $\widehat{f}_u \leftarrow -1$;
10: **end if**
11: **end for**
12: **return** $\widehat{f}_N$

Fine-Tuning Parameters. In view of Lemma 3.1, one could further optimize the choice of $\lambda \in [0, 1]$ in defining $\overline{f}_N \leftarrow \lambda\vec{m} + (1-\lambda)\vec{M}$ in Line 3. Similarly, one could pick the threshold θ to be the ϑ-percentile of the sorted coordinates of $\overline{f}_N$, for some choice of $\vartheta \in [0, 1]$. The parameters λ and ϑ can be tuned using standard techniques like cross-validation. However, to illustrate our concepts, we keep the description simple without introducing too many free parameters.

3.1. Properties of Confidence Vectors

We derive some properties of the confidence vectors to prove Lemma 3.1. The full proofs of Lemma 3.2 and 3.3 are given in the full version.

Given a feasible solution $f \in \mathbb{R}^V$ to (CP1), we define the following:

1. $S_e(f) := \arg\max_{u \in \mathsf{T}_e} f_u \subseteq \mathsf{T}_e$ and $I_e(f) := \arg\min_{v \in \mathsf{H}_e} f_v \subseteq \mathsf{H}_e$.
2. $f(S_e) := \max_{u \in \mathsf{T}_e} f_u$ and $f(I_e) := \min_{v \in \mathsf{H}_e} f_v$. Hence, we have $\Delta_e(f) = f(S_e) - f(I_e)$.
3. The set of *active edges* with respect to f is $E(f) := \{e \in E : \Delta_e(f) > 0\}$.

The following lemma states even though a minimizer for (CP1) might not be unique, there are still some structural properties for any optimal solution.

Lemma 3.2 (Active Edges in an Optimal Solution) *Suppose f and g are optimal solutions to (CP1). Then, for all $e \in E$, $[\Delta_e(f)]^+ = [\Delta_e(g)]^+$. In particular, this implies that the set of active edges $E^* := E(f) = E(g)$ in any optimal solution is uniquely determined. Hence, for $e \in E^*$, we can define the corresponding $\Delta^*_e = \Delta_e(f)$.*

Definition 3.2 (Pinned Vertex) *An unlabeled vertex u is pinned in a solution $f \in \mathbb{R}^V$ if there exist active edges e and $e' \in E(f)$ such that $u \in S_e(f) \cap I_{e'}(f)$, in which case we say that the edges e and e' pin the vertex u under f.*

Lemma 3.3 (Extending an Active Edge) *Suppose edge $e \in E(f)$ is active in an optimal solution f. If H_e does not contain a vertex labeled with -1, then there exist $u \in I_e(f)$ and another active edge $e' \in E(f)$ such that the following holds.*

(a) The edges e and e' pin u under f, i.e., $u \in S_{e'}(f)$.
(b) If g is an optimal solution, then $I_e(f) \cap S_{e'}(f) = I_e(g) \cap S_{e'}(g)$ and $f_u = g_u$.

An analogous result holds when T_e does not contain any vertex labeled with $+1$.

In particular, for any active edge $e \in E^*$, the extremal values $f^*(S_e) := \max_{u \in \mathsf{T}_e} f_u$ and $f^*(I_e) := \min_{u \in \mathsf{H}_e} f_u$ are uniquely determined by any optimal solution f.

Corollary 3.1 (Pinned Vertices) *In any optimal solution, the set of pinned vertices is uniquely determined. We use L^* to denote the set of labeled or pinned vertices in an optimal solution. Then, for each $u \in L^*$, its value f^*_u in any optimal solution is also uniquely determined.*

From Corollary 3.1, the confidence interval for any $u \in L^*$ contains exactly one value, namely the unique value f^*_u in any optimal solution. The following lemma gives a characterization of an optimal solution.

Lemma 3.4 Characterization of Optimal Solutions *A solution f to* (CP1) *is optimal iff the following conditions hold.*

(a) For each $u \in L^*$, $f_u = f^*_u$.
(b) For each active edge $e \in E^*$, both the maximum $\max_{u \in \mathsf{T}_e} f_u$ and the minimum $\min_{v \in \mathsf{H}_e} f_v$ are attained by vertices in L^*.
(c) For each inactive edge $e \notin E^*$, for all $u \in \mathsf{T}_e$ and $v \in \mathsf{H}_e$, $f_u \leq f_v$.

Proof: We first observe that Corollary 3.1 states that the values of the vertices in L^* are uniquely determined in any optimal solution. Hence, any optimal solution must satisfy the three conditions. We next show that the three conditions implies that the objective value is optimal.

Once the values for vertices in L^* are fixed, Lemma 3.3 and condition (b) implies that the contribution of all active edges E^* are determined and are the same as any optimal solution.

Finally, condition (c) implies that edges not in E^* do not have any contribution towards the objective function. Hence, any solution satisfying the three conditions must be optimal. ■

Deriving Confidence Vectors. To prove Lemma 3.5, we define a procedure that returns a vector $\vec{m} \in V^R$ such that for any optimal $f \in \mathsf{OPT}$, we have $f \geq \vec{m}$. Moreover, we shall show that $\vec{m} \in \mathsf{OPT}$ and hence $\vec{m}$ is the lower confidence vector. The argument for the upper confidence vector $\vec{M}$ is similar. For the special case of undirected hypergraphs, the procedure can be simplified to Algorithm 2 in Section 3.2.

Lemma 3.5 (Confidence Vectors are Optimal: Proof of Lemma 3.1) *The confidence vectors $\vec{m}$ and $\vec{M}$ defined in Definition 3.1 are optimal solutions to* (CP1)*. This implies that any of their convex combination is also optimal.*

Proof: We give a procedure that returns a vector $\vec{m}$ such that at any moment during the procedure, the following invariant is maintained: for any $f \in \mathsf{OPT}$, $f \geq \vec{m}$.

The following steps correspond to maintaining the conditions in Lemma 3.4.

(a) Initialization. For $v \in L^*$, set $m_v := f^*_v$; for $v \notin L^*$, set $m_v := -1$. This satisfies the invariant, because for any $f \in \mathsf{OPT}$ and any $v \in L^*$, $f_v = f^*_v$.

(b) Preserving Active Edges. For each $v \notin L^*$, set $m_v \leftarrow \max\{m_v, \max_{e \in E^*: v \in \mathsf{H}_e} f^*(I_e)\}$. Observe that Lemma 3.4(b) implies that for any optimal $f \in \mathsf{OPT}$, any $e \in E^*$ and any $v \in \mathsf{H}_e$, $f_v \geq f^*(I_e)$. Hence, the invariant is maintained.

(c) Preserving Inactive Edges. While there is an inactive edge $e \notin E^*$ such that $u \in \mathsf{T}_e$, $v \in \mathsf{H}_e$ and $m_u > m_v$, set $m_v \leftarrow m_u$. We argue why each such update preserves the invariant. Consider any optimal $f \in \mathsf{OPT}$. Before this update, the invariant holds. Hence, we have $m_u \leq f_u$. Moreover, Lemma 3.4 implies that $f_u \leq f_v$. Therefore, after setting $m_v \leftarrow m_u$, we still have $m_v \leq f_v$.

Finally, observe that after step (b), the coordinates of $\vec{m}$ can take at most n distinct values. Moreover, after each update in step (c), one coordinate of $\vec{m}$ must increase strictly. Hence, this procedure will terminate.

We next argue that $\vec{m}$ is an optimal solution by checking that it satisfies the conditions in Lemma 3.4.

Condition (a). Observe that for each $v \in L^*$, m_v is initialized to f^*_v. Afterwards the value m_v could only be increased. However, because the invariant holds when the procedure terminates, it must be the case that $m_v = f^*_v$ at the end.

Condition (b). The procedure makes sure that at the end of step (b), for every active edge $e \in E^*$, $\min_{v \in \mathsf{H}_e} m_v$ can be attained by some vertex in L^*. Since only m_v for $v \notin L^*$ can be increased in step (c), it follows that in the end, the minimum can still be attained by some vertex in L^*.

Next, consider $u \in \mathsf{T}_e$, where $e \in E^*$. For any optimal solution f, Lemma 3.3 implies that $f_u \leq f^*(S_e)$. Hence, the invariant implies that $m_u \leq f_u \leq f^*(S_e)$. Since condition (a) holds, this means that $\max_{v \in \mathsf{T}_e} m_v$ can be attained by some vertex in L^*.

Condition (c). This is clearly satisfied because of the while-termination condition.

Therefore, we have $\vec{m} \in \mathsf{OPT}$, as required.

The proof for the upper confidence vector $\vec{M}$ is similar. We omit the detailed proof and just give the corresponding procedure to return $\vec{M}$.

(a) Initialization. For $v \in L^*$, set $M_v := f^*_v$; for $v \notin L^*$, set $M_v := +1$.

(b) Preserving Active Edges. For each $v \notin L^*$, set $M_v \leftarrow \min\{M_v, \min_{e \in E^*: v \in \mathsf{T}_e} f^*(S_e)\}$.

(c) Preserving Inactive Edges. While there is an inactive edge $e \notin E^*$ such that $u \in \mathsf{T}_e$, $v \in \mathsf{H}_e$ and $M_u > M_v$, set $M_u \leftarrow M_v$.

The same argument can show that for any optimal $f \in \mathsf{OPT}$, we have $f \leq \vec{M}$. Moreover, we also have $\vec{M} \in \mathsf{OPT}$. ■

3.2. Computing the Confidence Interval

As mentioned before, the proof of Lemma 3.5 implicitly gives a procedure to compute the confidence vectors from any optimal solution. For the special case of undirected hypergraphs, a simplified version of the procedure is given in Algorithm 2.

Alternatively, we can try to solve the convex program (CP1), for example using Algorithm 5 in Section 4, from two initial feasible solutions to heuristically estimate the confidence vectors. In Algorithm 3, one instance approaches an optimal solution from high f values and the other from low f values.

4. Subgradient Method via Markov Operator

Resolving Ties. Observe that $\Phi : \mathbb{R}^N \to \mathbb{R}$ is differentiable at $f_N \in \mathbb{R}^N$ that has distinct coordinates. For the purpose of computing a subgradient, we assume that there is some global ordering π on V to resolve ties among coordinates with the same value. In particular, the vertices in L having label $+1$ are the highest, and those in L labeled -1 are the lowest. Hence, in this section, we may assume that any $\arg\max$ or $\arg\min$ operator over a subset of vertices

Algorithm 2 Confidence Intervals for Undirected Hypergraphs

Input: Undirected hypergraph $H = (V, E, w)$, label vector f_L^* and tolerance $\epsilon \geq 0$.
Let f be a solution of (CP1), either by Algorithm 5 or by PDHG method (Hein et al., 2013)
For all $v \in V$, set $p(v) \leftarrow v$, $m_v \leftarrow -1$, $M_v \leftarrow +1$.
$\widehat{E} := \{e \in E : \Delta_e(f) \leq \epsilon\}$
while $\exists e_1 \neq e_2 \in \widehat{E}, e_1 \cap e_2 \neq \emptyset$ **do**
  $\widehat{E} \leftarrow (\widehat{E} \setminus \{e_1, e_2\}) \cup \{e_1 \cup e_2\}$
end while
for each $e \in \widehat{E}$ **do**
  $x \leftarrow$ an arbitrary vertex in e
  for each vertex $v \in e$ **do**
    $p(v) \leftarrow p(x)$
  end for
end for
for each vertex $v \in L$ **do**
  $m_{p(v)} \leftarrow f_v^*$, $M_{p(v)} \leftarrow f_v^*$
end for
for each edge $e \in E$ such that $\Delta_e(f) > \epsilon$ **do**
  for each vertex $v \in e$ **do**
    $m_{p(v)} \leftarrow \max\{m_{p(v)}, f(I_e)\}$
    $M_{p(v)} \leftarrow \min\{M_{p(v)}, f(S_e)\}$
  end for
end for
for each vertex $v \in V$ **do**
  $m_v \leftarrow m_{p(v)}$, $M_v \leftarrow M_{p(v)}$
end for
return vectors $(\vec{m}, \vec{M})$

will return a unique vertex.

We next define a Markov operator that is inspired from the diffusion processes on hypergraphs (Louis, 2015) and directed graphs (Yoshida, 2016) in the context of defining Laplacians. We denote the projection operator $\Pi_N : \mathbb{R}^V \to \mathbb{R}^N$ that takes $f \in \mathbb{R}^V$ and returns the restricted vector $f_N \in \mathbb{R}^N$.

Lemma 4.1 *For $f \in [-1, 1]^V$ that is feasible in (CP1), the Markov operator $\mathsf{M}f$ given in Algorithm 4 returns a subgradient of $\Phi : \mathbb{R}^N \to \mathbb{R}$ at f_N.*

Proof: (*Sketch*) Observe that if $f_N \in \mathbb{R}^N$ has distinct coordinates, then Φ is differentiable at f_N, and $\mathsf{M}f$ gives exactly the gradient (which is the only possible subgradient in this case). Observe that in our subgradient method application, we could imagine that at every iteration, infinitesimal perturbation is performed on the current solution to ensure that all coordinates are distinct, and ties are resolved according to our global ordering π.

Algorithm 3 Estimate confidence interval

Input: Directed hypergraph $H = (V, E, w)$, labels f_L^* for labeled vertices L
Construct feasible $f_N^{(0,+)} \leftarrow +\mathbf{1} \in \mathbb{R}^N$ with all entries being $+1$;
Construct feasible $f_N^{(0,-)} \leftarrow -\mathbf{1} \in \mathbb{R}^N$ with all entries being -1;
$\vec{M} \leftarrow \mathsf{SGM}(f_N^{(0,+)})$;
$\vec{m} \leftarrow \mathsf{SGM}(f_N^{(0,-)})$;
return the vectors $(\vec{m}, \vec{M})$

Algorithm 4 Markov Operator $\mathsf{M} : \mathbb{R}^V \to \mathbb{R}^N$

Input: Directed hypergraph $H = (V, E, w)$, feasible $f \in \mathbb{R}^V$ for (CP1)
Construct symmetric matrix $A \in \mathbb{R}^{V \times V}$; set $A \leftarrow \mathbf{0}$.
for each $e \in E$ such that $\Delta_e(f) > 0$ **do**
  $u \leftarrow \arg\max_{u \in \mathsf{T}_e} f_u$;
  $v \leftarrow \arg\min_{v \in \mathsf{H}_e} f_v$;
  $A_{uv} \leftarrow A_{uv} + w_e$;
  (The same is done for A_{vu} because A is symmetric.)
end for
Construct diagonal matrix $W \in \mathbb{R}^{N \times N}$; set $W \leftarrow \mathbf{0}$.
for each $u \in N$ **do**
  $W_{uu} \leftarrow \sum_{v \in V} A_{uv}$;
end for
return $(W\Pi_N - \Pi_N A)f$

Hence, as the magnitude of the perturbation tends to zero, if the global ordering π is preserved, then the gradient remains the same, which implies that the gradient is also the subgradient when the perturbation reaches 0. ■

Using the Markov operator M as a subroutine to generate a subgradient, we have the following subgradient method (SGM) (Shor et al., 1985).

Algorithm 5 Subgradient Method $\mathsf{SGM}(f_N^{(0)} \in \mathbb{R}^N)$

Input: Directed hypergraph $H = (V, E, w)$ with labels f_L^* for labeled vertices L, initial feasible solution $f_N^{(0)} \in \mathbb{R}^N$, step size $\{\eta_t := \frac{1}{t}\}_{t \geq 1}$
$t \leftarrow 1$;
(Throughout the algorithm, $f_L^{(t)} = f_L^*$ is given by the labeled vertices.)
while Solution $f_N^{(t)}$ has not "stabilized" **do**
  $g_N^{(t)} \leftarrow \mathsf{M}f^{(t-1)} \in \mathbb{R}^N$;
  $f_N^{(t)} = f_N^{(t-1)} - \eta_t \cdot \frac{g_N^{(t)}}{\left\| g_N^{(t)} \right\|_2}$;
  $t \leftarrow t + 1$;
end while
return $f^{(t)}$

Stabilizing Condition. Our experiments in Section 5 suggest that it suffices to run the solver for a short time, after which a better feasible solution f does not improve the prediction accuracy.

5. Experimental Results

Our experiments are run on a standard PC. In our graphs, each point refers to a sample mean, and the height of the vertical bar is the standard error of the mean.

5.1. Undirected Hypergraph: Comparing Accuracy of Prediction Methods

We show that our treatment of hypergraphs performs better than the previously best method in (Hein et al., 2013).

Hypergraph Model. We use three datasets from the UCI Machine Learning Repository (Lichman, 2013): mushroom, covertype45 and covertype67. As in (Hein et al., 2013), each dataset fits into the hypergraph learning model in the following way. Each entry in the dataset corresponds to a vertex, which is labeled either $+1$ or -1. Moreover, each entry has some categorical attributes. For each attribute and each realized value for that attribute, we form a unit-weight hyperedge containing all the vertices corresponding to entries having that attribute value. To summarize, below are the properties of the resulting hypergraphs.

Dataset	mushroom	covertype45	covertype67
$n = \lvert V\rvert$	8124	12240	37877
$m = \lvert E\rvert$	112	104	123
$k = \frac{\sum_{e\in E}\lvert e\rvert}{m}$	1523	1412	3695

Semi-supervised Learning Framework. We compare our semi-supervised learning framework with that in (Hein et al., 2013), which was previously the best (compared to (Zhou et al., 2006), for instance). Specifically, we compare the prediction accuracy of the following two prediction algorithms.

1. **Confidence Interval (CI).** We use hard constraints (CP1) and confidence intervals for prediction, as described in Algorithm 1 in Section 3.
2. **Hein et al.** We implement the method described in (Hein et al., 2013), which uses soft constraints (regularized version), plus 5-fold cross validation to determine the regularization parameter.

Testing Methodology. Since we focus on prediction accuracy, using either subgradient method or PDHG (Hein et al., 2013) for solving the underlying convex programs in each algorithm produces the same results. For each algorithm candidate, we try different sizes of labeled vertices L, where $l = |L|$ ranges from 20 to 200. For each size l of labeled vertices, we randomly pick l vertices from the dataset to form the set L and treat the rest as unlabeled vertices; we re-sample if only one label ($+1$ or -1) appears in L. For each size l, we perform 100 trials to report the average error rate together with its standard error.

Results. Our experiment can recover the results reported in (Hein et al., 2013). The test error for the two algorithms on the three datasets is presented in Figure 5.1, which shows that our CI method consistently has lower test error than the one in (Hein et al., 2013).

5.2. Comparing Running Times of Solvers

Different Solvers. We compare the running times of the following two convex program solvers:

- Subgradient Method (SG), proposed by us. Empirically, the step size $\eta_t := \frac{1}{(t+1)^{\min(\frac{0.16t}{10^5},1)}}$ gives good performance. For large t, η_t grows like $\frac{1}{t}$ and so the method converges; however, for small t, we would like a larger step size to speed up convergence.
- Primal-Dual Hybrid Gradient (PDHG), proposed in (Hein et al., 2013). We choose $\sigma = \tau = \frac{1}{\sqrt{1+d}}$, where d is the maximum degree.

Theoretical Analysis. Given a hypergraph with n vertices and m edges, where the average size of an edge is k, each vertex on average appears in $\frac{mk}{n}$ edges. For SG, we use a heap-based data structure to maintain the vertices within a hyperedge. Vertices attaining the maximum and the minimum value within a hyperedge can be retrieved in $O(1)$ time, and a value update takes $O(\log k)$ time. In each iteration, at most $2m$ vertices will have their values updated. Hence, in each iteration, SG takes time $2m \cdot \frac{mk}{n} \cdot O(\log k) = O(\frac{m^2k}{n}\log k)$. In the description of PDHG in (Hein et al., 2013), each iteration takes $O(mk \log k)$ time. Hence, when $n \gg m$, each iteration of SG will be significantly faster, although in general, the number of iterations required by the subgradient method can be larger than that for PDHG.

Testing Methodology. In each experiment, we consider the hypergraph from one of the above three datasets. We pick $l = 160$ vertices at random as the labeled vertices L, and form the corresponding convex program (CP1) for the two solvers, where the initial values for unlabeled vertices are chosen independently to be uniformly at random from $[-1, 1]$. To compare the performance, we run the two solvers on the same convex program, and record each trajectory of the objective value versus the time duration. According to experience, 100 seconds is good enough for either solver to reach an almost optimal solution, and we use the minimum value achieved by the two solvers after 100 seconds as an estimate for the true optimal value OPT. Then, we scan each trajectory, and for each *relative gap*

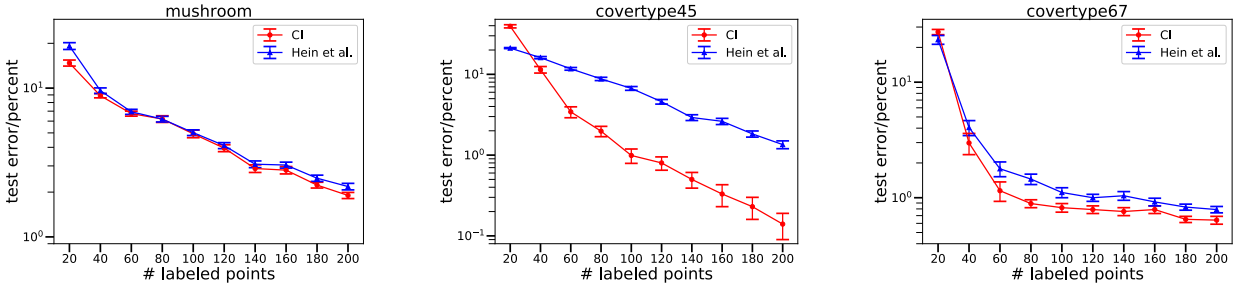

Figure 5.1. Prediction Accuracy of CI vs Hein .et al.

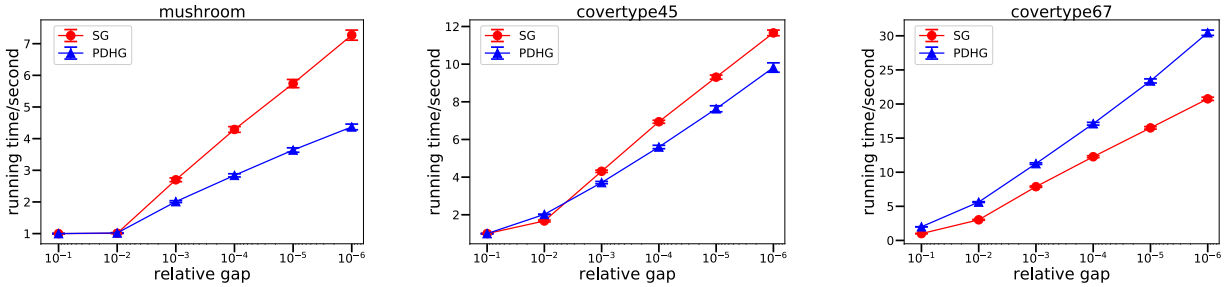

Figure 5.2. Comparing Running Times of the Two Solvers

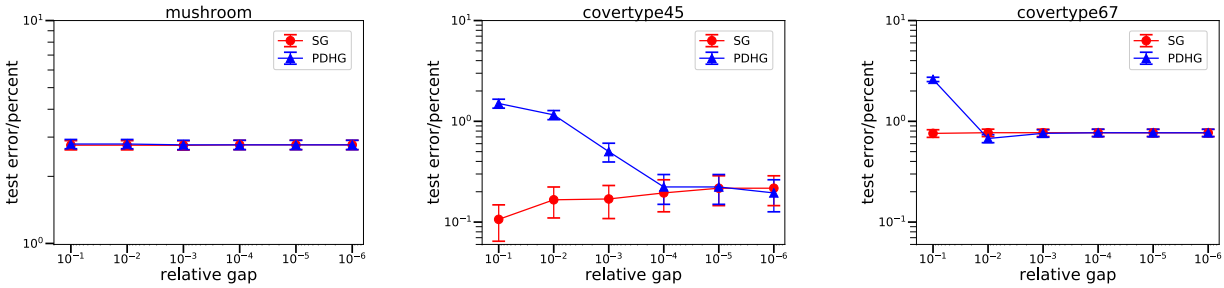

Figure 5.3. Test Error vs Relative Gap for the Two Solvers

$\epsilon \in \{10^{-i} : i = 1, 2, \ldots, 6\}$, we find the smallest time $T(\epsilon)$ after which the objective value is at most $\epsilon\mathsf{OPT}$ away from the estimate OPT. Each instance of the experiment is repeated 100 times (with different sets of labeled vertices) to obtain an average of those $T(\epsilon)$'s and their standard error. For each relative gap ϵ, we also report the test error for using a feasible solution that is $\epsilon\mathsf{OPT}$ away from the presumed optimal value OPT.

Results. Both solvers have similar performance. As predicted by our theoretical analysis, we see in Figure 5.2 that SG has an advantage when the number n of vertices is much larger than the number m of edges, which is the case for the the last dataset covertype67. Moreover, in Figure 5.3, we see that achieving a relative gap smaller than 10^{-4} has almost no effect on improving the prediction accuracy. Hence, we can conclude that for either solver, it takes roughly 10 to 20 seconds to produce a solution for the underlying convex program that can give good prediction accuracy.

5.3. Directed Hypergraph: More Powerful

DBLP Dataset. We use the DBLP (Ley, 2009) dataset. Each paper is represented by a vertex. We include papers from year 2000 to 2015 from conferences belonging to the following research areas to conduct our experiments:

- 7049 papers from machine learning (ML): NIPS, ICML
- 2539 papers from theoretical computer science (TCS): STOC, FOCS
- 3374 papers from database (DB): VLDB, SIGMOD

We perform the following prediction tasks: (a) ML (+1) vs TCS (-1), and (b) ML (+1) vs DB (-1).

The details of the experiment setup and the results are given in the full version.

References

Agarwal, Sameer, Branson, Kristin, and Belongie, Serge. Higher order learning with graphs. In *Proceedings of the 23rd international conference on Machine learning*, pp. 17–24. ACM, 2006.

Gallo, Giorgio, Longo, Giustino, Pallottino, Stefano, and Nguyen, Sang. Directed hypergraphs and applications. *Discrete applied mathematics*, 42(2):177–201, 1993.

Gao, Shenghua, Tsang, Ivor Wai-Hung, and Chia, Liang-Tien. Laplacian sparse coding, hypergraph laplacian sparse coding, and applications. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(1):92–104, 2013.

Gibson, David, Kleinberg, Jon, and Raghavan, Prabhakar. Clustering categorical data: An approach based on dynamical systems. *Databases*, 1, 1998.

Hein, Matthias, Setzer, Simon, Jost, Leonardo, and Rangapuram, Syama Sundar. The total variation on hypergraphs-learning on hypergraphs revisited. In *Advances in Neural Information Processing Systems*, pp. 2427–2435, 2013.

Huang, Yuchi, Liu, Qingshan, Zhang, Shaoting, and Metaxas, Dimitris N. Image retrieval via probabilistic hypergraph ranking. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 3376–3383. IEEE, 2010.

Ley, Michael. Dblp: some lessons learned. *Proceedings of the VLDB Endowment*, 2(2):1493–1500, 2009.

Lichman, M. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

Louis, Anand. Hypergraph markov operators, eigenvalues and approximation algorithms. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pp. 713–722. ACM, 2015.

Ricatte, Thomas, Gilleron, Rémi, and Tommasi, Marc. Hypernode graphs for spectral learning on binary relations over sets. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 662–677. Springer, 2014.

Shor, N. Z., Kiwiel, Krzysztof C., and Ruszcayǹski, Andrzej. *Minimization Methods for Non-differentiable Functions*. Springer-Verlag New York, Inc., New York, NY, USA, 1985. ISBN 0-387-12763-1.

Sun, Liang, Ji, Shuiwang, and Ye, Jieping. Hypergraph spectral learning for multi-label classification. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 668–676. ACM, 2008.

Tan, Shulong, Guan, Ziyu, Cai, Deng, Qin, Xuzhen, Bu, Jiajun, and Chen, Chun. Mapping users across networks by manifold alignment on hypergraph. In *AAAI*, volume 14, pp. 159–165, 2014.

Yoshida, Yuichi. Nonlinear laplacian for digraphs and its applications to network analysis. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pp. 483–492. ACM, 2016.

Yu, Jun, Tao, Dacheng, and Wang, Meng. Adaptive hypergraph learning and its application in image classification. *IEEE Transactions on Image Processing*, 21(7): 3262–3272, 2012.

Zhou, Dengyong, Huang, Jiayuan, and Schölkopf, Bernhard. Learning with hypergraphs: Clustering, classification, and embedding. In *Advances in neural information processing systems*, pp. 1601–1608, 2006.

Zhu, Xiaojin, Ghahramani, Zoubin, and Lafferty, John D. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*, pp. 912–919. AAAI Press, 2003.