BNL-211294-2019-JAAM

Visual Analytics of Heterogeneous Data Using Hypergraph Learning

C. Xie, W. Xu

To be published in "ACM TRANSACTIONS ON INTELLIGENT SYSTEMS AND TECHNOLOGY"

January 2019

Computational Science Initiative

Brookhaven National Laboratory

U.S. Department of Energy

USDOE Office of Science (SC), Advanced Scientific Computing Research (SC-21)

DISCLAIMER

Visual Analytics of Heterogeneous Data Using Hypergraph Learning

CONG XIE and WEN ZHONG, Stony Brook University
WEI XU, Brookhaven National Laboratory
KLAUS MUELLER, Stony Brook University

For real-world learning tasks (e.g., classification), graph-based models are commonly used to fuse the information distributed in diverse data sources, which can be heterogeneous, redundant, and incomplete. These models represent the relations in different datasets as pairwise links. However, these links cannot deal with high-order relations which connect multiple objects (e.g., in public health datasets, more than two patient groups admitted by the same hospital in 2014). In this article, we propose a visual analytics approach for the classification on heterogeneous datasets using the hypergraph model. The hypergraph is an extension to traditional graphs in which a hyperedge connects multiple vertices instead of just two. We model various high-order relations in heterogeneous datasets as hyperedges and fuse different datasets with a unified hypergraph structure. We use the hypergraph learning algorithm for predicting missing labels in the datasets. To allow users to inject their domain knowledge into the model-learning process, we augment the traditional learning algorithm in a number of ways. In addition, we also propose a set of visualizations which enable the user to construct the hypergraph structure and the parameters of the learning model interactively during the analysis. We demonstrate the capability of our approach via two real-world cases.

CCS Concepts: • **Human-centered computing → Visual analytics**;

Additional Key Words and Phrases: Hypergraph learning, data fusion, high-dimensional data

1 INTRODUCTION

Real-world learning tasks typically utilize information that is distributed across many disparate data sources. These data are often heterogeneous and contain different types of objects and various relations. Fusing these "real-world data" is frequently implemented via a graph structure [16] where an edge represents the pairwise relation of two objects in different datasets.

This research was partially supported by NSF grant IIS 1527200, BNL LDRD grant 16-041 and 18-009, the Exascale Computing Project (ECP) the Co-design center for Online Data Analysis and Reduction (CODAR) 17-SC-20-SC, and the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the "IT Consilience Creative Program (ITCCP)" (NIPA-2013-H0203-13-1001) supervised by NIPA.

Authors' addresses: C. Xie, W. Zhong, and K. Mueller, Department of Computer Science, Stony Brook University, Stony Brook, NY 11794; emails: {coxie, wezzhong, mueller}@cs.stonybrook.edu; W. Xu, Computational Science Initiative, Brookhaven National Laboratory, Upton, NY 11973; email: xuw@bnl.gov.

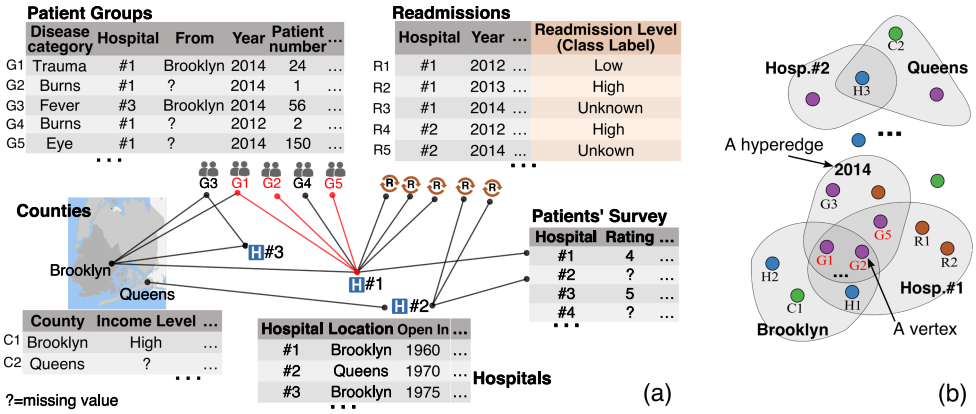

Fig. 1. (a) An example of predicting hospital readmission level with five heterogeneous tables. The readmission table is partially labeled as shown in the last column. Traditional graph-based methods use pairwise links to model pairwise relations among the tables. (b) In a hypergraph model, the hyperedges are able to encode higher-order relationships.

One application, in which these types of heterogeneous data typically occur, is hospital readmission prediction in public health. Readmission is the event when a patient checks back into a hospital within 30 days after discharge. Readmissions put a significant cost burden on the health care system (and cause stress for the patients), and therefore should be prevented [17]. We have been collaborating with a researcher in public health whose job is to predict readmission levels for specific target hospitals and patient groups. Here prediction can be considered as a classification problem where the readmission levels are the labels. The basis of this task is a set of disparate data tables related to readmission, encompassing factors like hospital ratings and past hospital readmissions (see the tables in Figure 1(a)). Via existing tools, he fuses these tables by connecting related objects (e.g., hospitals and patients) with the edges of a graph (see Figure 1(a)). Using the labeled nodes in the graph, classifying other unlabeled nodes can then be done using link mining methods [16]. For example, he can infer the readmission level of a specific patient group from the labeled readmission instances.

A shortcoming of the traditional graph model is that it has difficulties in dealing with higher-order relations. For example, in Figure 1(a), patient groups G1, G2 and G5 were all admitted by Hospital #1 in 2014. Such co-occurrences of multiple objects, however, cannot be easily viewed from the pairwise edges (red links in Figure 1(a)). This makes it difficult to efficiently recognize these type of joint relationships, especially at scale. Conversely, the hypergraph is able to represent these types of high-order relations. It is an extension to the ordinary graph, where a hyperedge can connect multiple vertices, and not just two. Figure 1(b) shows an example, where vertices G1, G2, and G5 are contained in the hyperedge "H osp.#1," but at the same time, they are also included in the hyperedge "2014." As a result, they become a part of the intersection of these two hyperedges, which conveys a possibly interesting relationship.

A crucial property of hyperedges is that they are defined based on the semantic meaning of the specific relation. For example, the hyperedge "2014" contains objects from both the patient and the readmission tables (purple and brown nodes in Figure 1(b), respectively). This makes them a natural representation for disparate heterogeneous data.

Our primary goal is to use the hypergraph to classify heterogeneous data given a partially labeling of it, such as "Readmission level" in the readmission table in Figure 1(a). Existing hypergraph learning methods [5, 15, 25] pre-define the structure of the hypergraph and the parameters

(e.g., the weights of the hyperedges). However, the construction process of a unified hypergraph principally is a design task and needs to be done manually during the analysis process. The user has to test different data sources for a proper hypergraph structure in a specific application. For example, a census dataset may not work for predicting the unknown readmission levels because it describes all the residents instead of only the patients. Furthermore, these users, who could be experienced domain experts, cannot lower or elevate the influences of certain factors by tuning their weights onto the model and its predictions, although this might be appropriate. In contrast, we have developed a visual analytics approach that enables such interactions. It allows a domain expert to assist the hypergraph learning process by applying his or her domain knowledge and intuition. In this way, a more realistic actionable model can be derived.

In our method, first a hypergraph structure for the heterogeneous data is constructed visually using a set of interactions defined on the hypergraph's incidence matrix. The hypergraph structure is then used for our improved hypergraph learning algorithm, which allows the user to adjust the parameters, such as hyperedge weights, during the learning process. After each run of the algorithm, the inspection, exploration, and validation of the learning results is enabled using a force-directed hypergraph visualization. Based on the feedback of the exploration, the user can go back to the previous steps to update the hypergraph structure or hyperedge weights, and examine the effects of the changes on the results.

The main contributions of this article include:

- We propose an improved hypergraph learning algorithm for heterogeneous data, which allows user input of domain knowledge.
- We propose a set of interactions based on an incidence matrix visualization, which enables the construction of a unified hypergraph structure on heterogeneous data.
- We extend the concept of force-directed visualization to hypergraphs, which enables the interactive verification, validation, and exploration of hypergraph learning results.

The remainder of this article is structured as follows. Section 2 reviews related work. Section 3 defines the problem and gives an overview of our approach. Section 4 introduces our improved hypergraph learning algorithm. Section 5, 6, and 7 describe our visual analytics approach for construction of the learning model, modulation of the learning process, and exploration of the learning results, respectively. Two real-world cases are used to validate our approach in Section 8. Section 9 ends with conclusions and future work.

2 RELATED WORK

A graph structure (e.g., Resource Description Framework [8]) is commonly used to model pairwise relations among different objects in heterogeneous datasets. Various algorithms have utilized graph models for link mining tasks [16]. For example, link-based classification [28] improves the classification accuracy by modeling the link distributions. On the other hand, a number of visualization methods have also been proposed to help the analysis of graph models on heterogeneous data, such as creating and designing visual representations [7, 27], searching heterogeneous networks [22], and evaluating hypotheses [2], and exploring multi-attribute data [9, 10, 11, 29, 41, 43]. However, as mentioned in Section 1, pairwise edges in a graph are unfit for high-order relations. Using ordinary graphs to model these relations can lead to unexpected information loss [44].

2.1 Hypergraph Learning on Heterogeneous Data

A hypergraph extends the ordinary graph to formulate complete high-order relations. An edge in a hypergraph can connect multiple vertices, called hyperedges (Figure 1(b)), as discussed. Existing hypergraph learning approaches [19, 20, 44] use the hyperedges to model the relations within

a table. For example, the bag-of-words model can be represented as a hypergraph in which a document is a hyperedge containing multiple words [38]. For heterogeneous data, hyperedges can be employed to connect related objects from different datasets, such as audiences and music tracks [5], readers and news [25], images and textual tags [15]. Most of these existing methods, however, pre-define the hypergraph structures according to their specific scenarios. They do not provide users with the means to modify the structure during the analysis (e.g., removing a biased dataset from the hypergraph). In addition, the parameters of the model are typically also pre-defined (e.g., hyperedge weights) in the hypergraph learning algorithms. However, as mentioned earlier, it is beneficial to endow the user with the ability to update the weights of the learning model during the analysis. Therefore, to deal with these two shortcomings, we improve the hypergraph learning algorithm [44] such that it accepts domain knowledge from the user. We embed the algorithm into a visual analytics framework that allows the interactive modulation of the hypergraph structure and parameters during the analysis process.

2.2 Hypergraph and Set Visualization

Visual exploration of the structure and the learning results of the hypergraph model can render useful insights for the analyst. Existing set visualization techniques [1, 40] can be used since a hyperedge can be viewed as a set of vertices. In this article, we adopt two common types of set visualizations: matrix-based and contour-based methods.

Matrix-based visualizations allow the user to easily get an overview of either the inclusions of elements in the set [23, 35], or the intersections of two sets [24]. Usually, those relations are encoded by the visual properties of cells in a matrix, such as color.

Contour-based visualizations can be more intuitive to encode the containment and the intersection relationships of multiple sets. Generally, the nodes inside a contour represent the membership of elements in a set, such as in Euler and Venn diagrams [37]. Some of these approaches (e.g., Bubble Sets [13], KelpFusion [31], and Data Context Map [12]) generate contours based on a pre-determined node layout. Other methods adopt existing layout algorithms (e.g., force-directed layout in Vizster [18]) to calculate the node positions.

Other types of set visualizations have been proposed, such as the Parallel Sets plot [3]. Most of these existing visualizations focus on the visual representation of the set data (e.g., encoding or layout), while a few of them exploit a set structure visualization to enhance learning tasks, such as classification. In this work, we integrate the set visualization approaches with the hypergraph learning model, which allows the visual analysis of the classification using the hypergraph structure.

3 PROBLEM DEFINITION AND APPROACH OVERVIEW

This article aims to address the classification problem [45] on a set of heterogeneous tables $\{V\}$. We use the data in Figure 1 as an example. Given a part of the readmissions labeled as "High" ($y(v) = +1$) or "Low" ($y(v) = -1$) (see the last column in the readmission table in Figure 1(a)), the user wants to predict the readmission levels $\boldsymbol{f}$ of all unlabeled instances, including the unlabeled hospitals, unlabeled patient groups, and the unknown labels in the readmission table (e.g., the readmission levels in rows 3 and 5 in Figure 1(a)). We will use this example in our article to illustrate our approach.

The classification problem is defined as follows: suppose $V_i = \{v_1^{(i)}, v_2^{(i)}, \ldots\}$ is the instance set of table V_i and $\mathcal{V} = V_1 \cup V_2 \cup \cdots$ is the set of all instances; given a partial labeling $\boldsymbol{y}$ of $\mathcal{V}$, assign labeling $\boldsymbol{f}$ to the rest of the unlabeled instances in $\mathcal{V}$. For binary classification, the vertex label $y(v)$ is $+1, -1$, and 0 for positive label, negative label, and unlabeled instance, respectively. The notations used in this paper are summarized in Table 1.

Table 1. Notations Used in Our Article

Notation	Description
$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{u}, \boldsymbol{w})$	The unified hypergraph $\mathcal{G}$.
v	A vertex in $\mathcal{G}$, which is also an instance in a table.
$\mathcal{V} = \{v_1, v_2, \ldots, v_{\lvert\mathcal{V}\rvert}\}$	Vertex set of $\mathcal{G}$.
V_i	Instance set of the ith table.
e	A hyperedge in $\mathcal{G}$, which is also a level of a categorical variable.
$\mathcal{E} = \{e_1, e_2, \ldots, e_{\lvert\mathcal{E}\rvert}\}$	Hyperedge set of $\mathcal{G}$.
E	A categorical variable in a table.
$u(v)$	The weight of vertex v.
$\boldsymbol{U} \in \mathbb{R}^{\lvert\mathcal{V}\rvert\times\lvert\mathcal{V}\rvert}$	The diagonal matrix form of $\boldsymbol{u}$.
$w(e)$	The weight of hyperedge e.
$\boldsymbol{W} \in \mathbb{R}^{\lvert\mathcal{E}\rvert\times\lvert\mathcal{E}\rvert}$	The diagonal matrix form of $\boldsymbol{w}$.
$d(v)$	The degree of the vertex v.
$\boldsymbol{D_v} \in \mathbb{R}^{\lvert\mathcal{V}\rvert\times\lvert\mathcal{V}\rvert}$	The diagonal matrix of $\boldsymbol{d}$.
$\delta(e)$	The degree of the hyperedge e.
$\boldsymbol{D_e} \in \mathbb{R}^{\lvert\mathcal{V}\rvert\times\lvert\mathcal{V}\rvert}$	The diagonal matrix form of $\boldsymbol{\delta}$.
$\boldsymbol{H} \in \mathbb{R}^{\lvert\mathcal{V}\rvert\times\lvert\mathcal{E}\rvert}$	The incidence matrix representation of $\mathcal{G}$.
$\boldsymbol{y} = [y_0, y_1, \ldots, y_{\lvert\mathcal{V}\rvert}]^T$	The given partial labeling vector of $\mathcal{V}$.
$\boldsymbol{f} = [f_0, f_1, \ldots, f_{\lvert\mathcal{V}\rvert}]^T$	The labeling vector of $\mathcal{V}$ to be learned.
$w_d(e)$	The prior weight of hyperedge e which is set by the user.

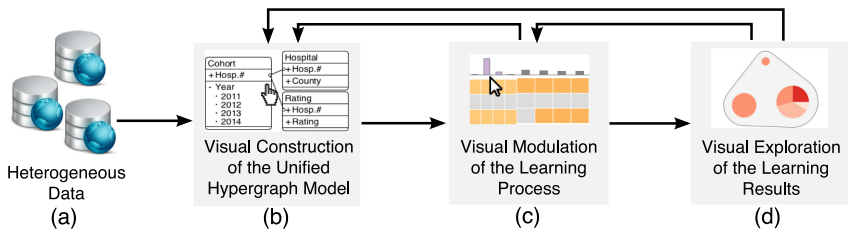

Fig. 2. Conceptual overview of our approach for classification on (a) heterogeneous data. The user is able to (b) construct, (c) modulate, and (d) validate the hypergraph model with visual interactions. The user can go back to the previous steps in this iterative process.

The classification of the heterogeneous data is accomplished by our visual analytics approach with the following steps (see Figure 2):

Step 1 Visual construction of the unified hypergraph model: The user can select tables of interest and present them in an initial incidence matrix visualization. He or she can then link, merge, include, and exclude the tables interactively within the matrix visualization to construct the hypergraph structure.

Step 2 Visual modulation of the learning process: With the constructed hypergraph in hand, our improved hypergraph learning algorithm is engaged to classify the unlabeled instances in the tables. The user is allowed to change the parameters (e.g., weights of hyperedges) interactively to update the learning results.

Step 3 Visual exploration and validation of the learning results: The user can select a subset of the interesting hyperedges from the matrix visualization. Then the structure and the

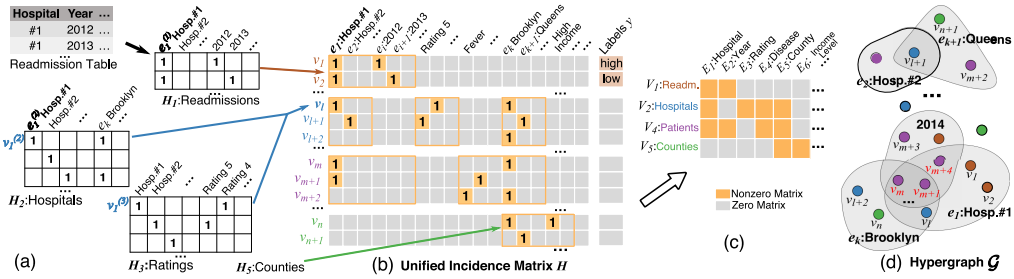

Fig. 3. The process of unifying all of the tables in Figure 1 with a hypergraph model. (a) The incidence matrices $\boldsymbol{H}_1$ - $\boldsymbol{H}_5$ are generated from the tables in Figure 1. The empty cells in the matrices represent zeros. By linking the variables and combining datasets, (b) a unified incidence matrix $\boldsymbol{H}$ can be constructed, which is the matrix form of the hypergraph $\mathcal{G}$. (c) $\boldsymbol{H}$ can be represented as a matrix of sub-matrices, the orange and gray cells represent nonzero and zero sub-matrices, respectively. (d) Hypergraph $\mathcal{G}$ can be regarded as a collection of sets, in which each hyperedge is a set of vertices.

learned labels of the hyperedge subset are shown in a force-directed hypergraph visualization for exploration and validation.

For each step, the user can return to the previous steps to change the structure or parameters of the hypergraph model until satisfying results are achieved (Figure 2).

4 HYPERGRAPH LEARNING

In this section, we first describe the definition of a hypergraph (Section 4.1) and the traditional hypergraph learning algorithm (Section 4.2), then introduce our improved algorithm for classification on heterogeneous tables (Section 4.3).

4.1 Formulation of Weighted Hypergraph

A hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{u}, \boldsymbol{w})$ is a generalization of a graph (Figure 3(d)). In detail, $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of hyperedges in $\mathcal{G}$. While a regular graph edge is a pair of nodes, a hyperedge $e \in \mathcal{E}$ connects a set of vertices $\{v\} \subseteq \mathcal{V}$. $\boldsymbol{w}$ represents the vector of weights of hyperedges $\mathcal{E}$. Different from the traditional hypergraph model [38, 44], we also add the weights $\boldsymbol{u}$ for the vertices $\mathcal{V}$. The motivation and the benefits are explained in Section 4.3 and Section 8.3.3. In another perspective, $\mathcal{G}$ can also be represented as a vertex-hyperedge incidence matrix $\boldsymbol{H} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{E}|}$, whose entry $h(v, e)$ is 1 if $v \in e$ and 0 otherwise (Figure 3(b)). Similar to the definition of the vertex degree in an ordinary graph, the hyperedge degree $\delta(e)$ and the vertex degree $d(v)$ is defined in Equation (1). Additionally, the diagonal matrix form for $\boldsymbol{w}$, $\boldsymbol{u}$, $\boldsymbol{\delta}$, and $\boldsymbol{d}$ are denoted as $\boldsymbol{W}$, $\boldsymbol{U}$, $\boldsymbol{D_e}$, and $\boldsymbol{D_v}$, respectively, which are used in the hypergraph learning algorithm in Section 4.3.

$$\delta(e) = \sum_{v\in\mathcal{V}} u(v)h(v,e) \text{ and } d(v) = \sum_{e\in\mathcal{E}} w(e)h(v,e) \tag{1}$$

Now that we have the definition of a hypergraph, given the categorical tables, a hypergraph can be built for each table by treating it as set data [44] (Figure 3 (a)). Specifically, let E be a categorical variable (e.g, "Location"), each of the possible values $e \in E$ is a level, such as "Brooklyn." Then the levels $\{e\}$ and the instances $\{v\}$ in the table are modeled as hyperedges and vertices, respectively. For example, a hospital v_l is a vertex and e_k = "Brooklyn" is a hyperedge. If the E value of v is e (e.g., the "Location" of hospital v_l is Brooklyn e_k in Figure 3 (a)), then $v \in e$ and $h(v, e) = 1$ in the incident matrix. In this way, a hypergraph and its incidence matrix can be constructed for each table, as shown in Figure 3 (a).

For different tables $\{V_i\}$, a unified hypergraph $\mathcal{G}$ can be constructed (Figure 3 (b)) by linking the variables in different datasets (Section 5.2.2) and combining the datasets (Section 5.2.3). The vertex set $\mathcal{V}$ and the hyperedges set $\mathcal{E}$ of $\mathcal{G}$ are the union of all instances $\mathcal{V} = V_1 \cup V_2 \cup \ldots$, and all variables $\mathcal{E} = E_1 \cup E_2 \cup \ldots$, respectively. Next, the constructed $\mathcal{G}$ can be used as the model for hypergraph learning.

4.2 Traditional Hypergraph Learning Framework

The hypergraph learning framework of Zhou et al. [44] calculates the labels $\boldsymbol{f}$ of the vertices $\mathcal{V}$ by minimizing the loss function in Equation (2).

$$\boldsymbol{f}^* = \arg\min_{\boldsymbol{f}} \Omega(\boldsymbol{f}) + \mu R_{emp}(\boldsymbol{f}) \tag{2}$$

The first term Ω in the loss function is defined in Equation (3). It aims to assign similar labels $f(v)$ to vertices v_i and v_j which are contained in many common hyperedges $\{e\}$. For example, if two patient groups are in the same age group and the same disease category, they may have the same readmission level (e.g., high readmission $f(v) = +1$). Hyperedges with high weights $w(e)$ in Equation (3) will dominate the label assignment. In our case, those hyperedges can be regarded as important factors of readmission.

$$\Omega(\boldsymbol{f}, \boldsymbol{w}) = \frac{1}{2} \sum_{i,j=1}^{|\mathcal{V}|} \sum_{e \in \mathcal{E}} \frac{1}{\delta(e)} \sum_{v_i, v_j \in e} w(e) u(v_i) u(v_j) \left\| \frac{f(v_i)}{\sqrt{d(v_i)}} - \frac{f(v_j)}{\sqrt{d(v_j)}} \right\|^2 \tag{3}$$

The second component $R_{emp}(\boldsymbol{f})$ in Equation (2) is defined in Equation (4). It measures the difference between the learned labels $\boldsymbol{f}$ and the pre-given labels $\boldsymbol{y}$ (see the last column $\boldsymbol{y}$ in Figure 3 (b)). The parameter μ in Equation (2) controls the relative importance of Ω and R_{emp}.

$$R_{emp}(\boldsymbol{f}) = \sum_{i=1}^{|\mathcal{V}|} u(v_i) \left\| f(v_i) - y(v_i) \right\|^2 = (\boldsymbol{f} - \boldsymbol{y})^T \boldsymbol{U} (\boldsymbol{f} - \boldsymbol{y}) \tag{4}$$

The traditional method [44] does not define the vertex weight $u(v)$, which is equivalent to setting all $u(v)$ to 1 in Equation (3) and Equation (4). The hyperedge weight $w(e)$ is usually assigned manually according to a specific case before the optimization process of Equation (2).

4.3 An Improved Hypergraph Learning Algorithm

To integrate the expert-provided knowledge into the analysis process, we augment the traditional hypergraph learning algorithm by allowing the user's initialization and modification of the weights.

In our problem, instances are aggregated in some of the tables. For example, the first patient group has 24 patients while the second patient group only contains 1 person (see patient group table in Figure 1(a)). Using uniform vertex weights will reduce the influences of the aggregated instances on the learning results. A reasonable solution is to set the weight of a vertex to its aggregation size.

For the hyperedge weights $\boldsymbol{w}$, pre-defined values are not sufficient since the user will wish to modify the weights according to some domain knowledge or feedback obtained from the learning results. We integrate the learning algorithm with the prior knowledge $\boldsymbol{w}_d$ provided by the user via adding an additional term $\Psi(\boldsymbol{w})$ to Equation (2). $\Psi(\boldsymbol{w})$ is defined in Equation (5).

$$\Psi(\boldsymbol{w}) = \sum_{e \in \mathcal{E}} \left\| w(e) - w_d(e) \right\|^2 \tag{5}$$

The augmented optimization is shown in Equation (6), where ρ sets the importance of $\Psi(\boldsymbol{w})$. In this way, injecting domain knowledge is enabled by changing the prior $\boldsymbol{w}_d$ during the analysis. For example, the user can set $w_d(e) = 0$ for all of the edges e (e.g., "Brooklyn") in "Location" E, if she or he knows that "Location" is not related to readmission. The reason that the user is prevented from setting $\boldsymbol{w}$ directly is that $\boldsymbol{w}$ needs to satisfy the vertex degree constraint in Equation (6).

$$(\boldsymbol{f}, \boldsymbol{w}) = arg\min_{\boldsymbol{f},\boldsymbol{w}} \Omega(\boldsymbol{f}, \boldsymbol{w}) + \mu R_{emp}(\boldsymbol{f}) + \rho\Psi(\boldsymbol{w})$$
$$s.t.\ d(v) = \sum_{e\in\mathcal{E}} w(e)h(v,e) \tag{6}$$

In Equation (6), it is difficult to optimize $\boldsymbol{f}$ and $\boldsymbol{w}$ at the same time since Ω may not be convex in $(\boldsymbol{f}, \boldsymbol{w})$. However, we find that if $\boldsymbol{f}$ and $\boldsymbol{w}$ are optimized independently, the local optimal solution of $(\boldsymbol{f}, \boldsymbol{w})$ can still be derived. We propose a two-step iterative method that alternatively finds the optimal $\boldsymbol{f}$ and $\boldsymbol{w}$ in Equation (6) as follows:

For the ith iteration in our method, we first fix $\boldsymbol{w} = \boldsymbol{w}^{(i-1)}$ in Equation (6). Because $\Psi(\boldsymbol{w}^{(i-1)})$ is a constant, we can get $\boldsymbol{f}^{(i)}$ using Equation (7). Because $\Omega \geq 0$ and $R_{emp} \geq 0$, the minimum value of $\Omega + \mu R_{emp}$ exists.

$$\boldsymbol{f}^{(\boldsymbol{i})} = \arg\min_{\boldsymbol{f}}\ \Omega(\boldsymbol{f}, \boldsymbol{w} = \boldsymbol{w}^{(\boldsymbol{i-1})}) + \mu R_{emp}(\boldsymbol{f}) \tag{7}$$

It can be proved[1] that $\Omega(\boldsymbol{f})$ can be written in the form: $\Omega(\boldsymbol{f}) = \boldsymbol{f}^T\Delta\boldsymbol{f}$, in which $\Delta = \boldsymbol{U} - \boldsymbol{U}\boldsymbol{D_v}^{-\frac{1}{2}}\boldsymbol{H}\boldsymbol{W}\boldsymbol{D_e}^{-1}\boldsymbol{H}^T\boldsymbol{D_v}^{-\frac{1}{2}}\boldsymbol{U}$. The optimal $\boldsymbol{f}^{(i)}$ of Equation (7) can be found by solving the system of linear equations in Equation (8).

$$\begin{aligned}&\frac{\partial(\Omega+\mu R_{emp})}{\partial f}\Big|_{f=f^{(i)}} = 0\\ \Rightarrow &2\Delta\boldsymbol{f}^{(\boldsymbol{i})} + 2\mu\boldsymbol{U}(\boldsymbol{f}^{(\boldsymbol{i})} - \boldsymbol{y}) = 0\\ \Rightarrow &(\Delta + \mu\boldsymbol{U})\boldsymbol{f}^{(i)} = \mu\boldsymbol{U}\boldsymbol{y}\end{aligned} \tag{8}$$

In the second step, $\boldsymbol{f}$ is fixed to $\boldsymbol{f}^{(i)}$, then $\Omega(\boldsymbol{f} = \boldsymbol{f}^{(\boldsymbol{i})}, \boldsymbol{w})$ is a linear function of $\boldsymbol{w}$ and $R_{emp}(\boldsymbol{f})$ is a constant. The optimal $\boldsymbol{w}^{(\boldsymbol{i})}$ is derived by solving the quadratic programming problem in Equation (9).

$$\boldsymbol{w}^{(\boldsymbol{i})} = \arg\min_{\boldsymbol{w}}\ \Omega(\boldsymbol{f} = \boldsymbol{f}^{(\boldsymbol{i})}, \boldsymbol{w}) + \rho\Psi(\boldsymbol{w})$$
$$s.t.\ d(v) = \sum_{e\in\mathcal{E}} w(e)h(v,e) \tag{9}$$

The above two steps are repeated until the loss function $\Omega + \mu R_{emp} + \rho\Psi$ in Equation (6) converges, as shown in Algorithm 1. For the learning results, a vertex v is labeled as positive if $f(v) > 0$, otherwise negative. The user can always change the prior $\boldsymbol{w}_d$ (Section 6) and run the algorithm for an updated $\boldsymbol{f}$.

For multi-class classification, let us suppose there are c classes. Vector $\boldsymbol{f}$ and $\boldsymbol{y}$ can be replaced with labeling matrices $\boldsymbol{F} = [\boldsymbol{f_1}, \boldsymbol{f_2}, \ldots, \boldsymbol{f_c}]$ and $Y = [\boldsymbol{y_1}, \boldsymbol{y_2}, \ldots, \boldsymbol{y_c}]$. $y_j(v) = 1$ indicates v is labeled as the jth class. $\Omega(\boldsymbol{f}, \boldsymbol{w})$ and $R_{emp}(\boldsymbol{f})$ in the learning algorithm is replaced by $\sum_{j=1}^{c}\Omega(\boldsymbol{f}_j, \boldsymbol{w})$ and $\sum_j^c R_{emp}(\boldsymbol{f}_j)$, respectively. Then v are predicted as the jth class if $f_j(v)$ is the largest in $[f_1(v), f_2(v), \ldots, f_c(v)]$.

[1]The proofs are provided in the supplementary file.

ALGORITHM 1: Iterative Hypergraph Learning with Prior Weights

Input: $\boldsymbol{w}^{(0)} = [1, 1, \ldots, 1]^T$, $\boldsymbol{f}^{(0)} = \boldsymbol{y}^{(0)}$, $i = 0$, $\boldsymbol{w}_d$, ϵ

1: **do**
2: $i = i + 1$
3: Solve $\boldsymbol{f}^{(i)}$ by fixing $\boldsymbol{w} = \boldsymbol{w}^{(i-1)}$ according Equation (8).
4: Optimize $\boldsymbol{w}^{(i)}$ by fixing $\boldsymbol{f} = \boldsymbol{f}^{(i)}$ according to Equation (9).
5: **while** (The loss function value in Equation (6) changes less than ϵ)

5 VISUAL CONSTRUCTION OF THE HYPERGRAPH MODEL

The first step of our visual analysis is to build a unified hypergraph $\boldsymbol{G}$ which fuses the heterogeneous tables. This is done by constructing the incidence matrix $\boldsymbol{H}$ of $\boldsymbol{G}$ (Figure 3(b)). As mentioned in Section 1, testing different hypergraph structures during the analysis is necessary to achieve the best classification performance. To allow this process, we visualize the incidence matrix $\boldsymbol{H}$ (Section 5.1) in our approach. Along with the visualization, several interactions for initializing and modifying the hypergraph structure are provided (Section 5.2). The capability of changing hyperedge weights during the analysis can also be incorporated in this visualization (Section 6).

5.1 Hierarchical Incidence Matrix Visualization

It is natural to present the incidence matrix $\boldsymbol{H}$ using a matrix-based visualization [24, 35]. As shown in Figure 3(b), each row represents an instance v and each column is a level e. The color of each cell is used to encode $h(v, e)$: a cell is filled with orange if $h(v, e) = 1$; otherwise, it is gray.

It is challenging to display a matrix with large numbers of rows or columns due to finite screen space. We therefore combine the rows $\{v\}$ from the same table into one row V, and aggregate all levels $\{e\}$ of a variable in one column E (Figure 3(c)). In the aggregated matrix, each cell represents a sub-matrix. The color of an aggregated cell is set to gray if it is a zero matrix, otherwise it is filled with orange. For an aggregated column, the user can expand it to check its level set $\{e\}$ (see "Nurse Rating," "Pain Management," and "C are Rating" in Figure 4(d)). To show its difference from an original column, an aggregated column is slightly wider and darker. The user still has the option to view the columns without aggregation.

5.2 Visual Construction of the Incidence Matrix

With our matrix visualization, we define four operations on $\boldsymbol{H}$ to interactively fuse the sub-incidence matrices of disparate tables into a unified matrix: inclusion, linking, merging, and exclusion.

5.2.1 Inclusion of Variables and Tables. At the beginning of the construction, the user can select a set of interesting tables $\mathcal{V} = \{V\}$ and their variables $\mathcal{E} = \{E\}$ to be included in the initial incidence matrix $\boldsymbol{H}$. Figure 4(a) shows that several disconnected tables (V_1, V_2, ...) and their variables (E_1, E_2, ...) are included in the initial $\boldsymbol{H}$. An aggregated row and an aggregated column in $\boldsymbol{H}$ represent a table V and a variable E, respectively.

5.2.2 Linking Variables in Different Tables. Two different tables may have a shared variable. For example, hospital ID occurs in the hospital and readmission tables. The common variable in the two tables can be connected using the linking operation.

Linking two variables E_k and E_l is performed by first combining their levels. Then the same levels of $e_i^{(k)} \in E_k$ and $e_i^{(l)} \in E_l$ are combined into one column e_j. For example, both column $e_1^{(1)}$ and $e_1^{(2)}$ are "H osp.#1" in the readmission and hospital tables (H_1 and H_2 in Figure 3(a)), so they are

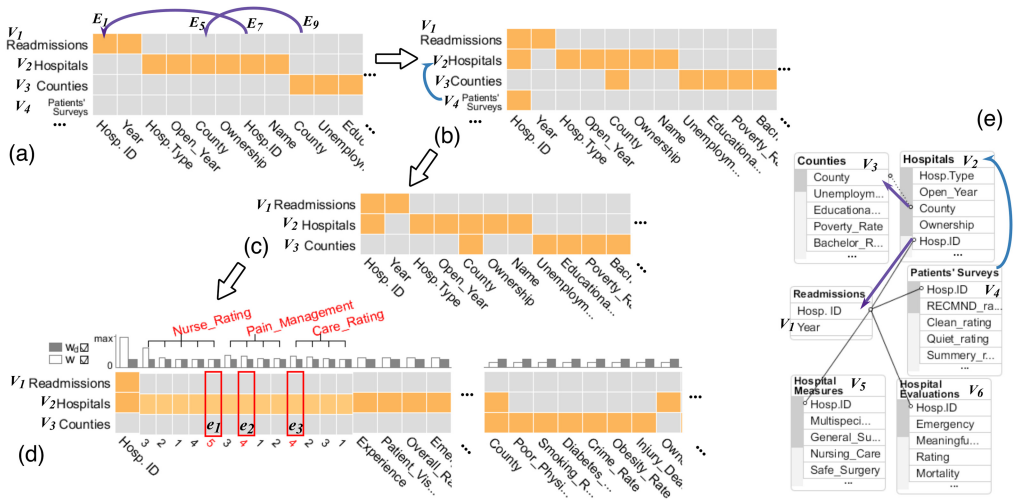

Fig. 4. (a) The initial incidence matrix visualization contains multiple disparate tables. An aggregated column and an aggregated row are a variable and a table, respectively. The orange cells mark the tables that contain the variables. Linking two variables can be done by dragging a column into another (purple lines). (b) The merging operation is performed by dragging a row into another (blue line). (c) The final matrix after the linking and merging operations. (d) The bar charts above the columns show the learned weights $\boldsymbol{w}$ and the prior weights $\boldsymbol{w_d}$. The columns of the matrix are reordered according to $\boldsymbol{w}$ of the variables. The user expands the aggregated columns of "Nurse Rating," "Pain Management," and "Care Rating" to view their levels. (e) The construction operations such as linking and merging are also supported in the dataset view.

combined into e_1 in H in Figure 3(b). Then, in the second step, we assign vertices $\{v\}$ to each combined edge e_j by setting $h(v, e_j)$ in H. $h(v, e_j)$ is set to 1 if $h(v, e_i^{(k)}) = 1$ or $h(v, e_i^{(l)}) = 1$; otherwise, $h(v, e_j) = 0$.

The user can perform a linking operation by dragging the aggregated column E_l into E_k (see the purple lines in Figure 4(a)). Figure 4(b) shows the result matrix after the linking operations.

5.2.3 Merging Instances in Different Tables. Two linked tables may describe the same set of objects. For example, the patient survey and hospital tables are linked by hospital ID, and they describe the ratings and the general information, respectively, of the same set of hospitals. Two such tables can be combined with a merging operation.

Merging two tables V_k and V_l is done by first merging their corresponding instances according to the linked variable E. For instances $v_i^{(k)} \in V_k$ and $v_j^{(l)} \in V_l$ with the same level $e \in E$, they are merged into one row v_m. For example, both $v_1^{(2)}$ and $v_1^{(3)}$ is "Hosp.#1" in the patients' survey and hospital tables (see H_2 and H_3 in Figure 3(a)), so they are combined into v_l in H in Figure 3(b). Then, in the second step, we assign each combined v_m to hyperedges $\{e\}$ by setting $h(v_m, e)$ in H. $h(v_m, e)$ is set to 1 if $h(v_i^{(k)}, e) = 1$ or $h(v_j^{(l)}, e) = 1$; otherwise, $h(v_m, e) = 0$.

This operation can be done by dragging an aggregated row V_k into another V_l (see blue line in Figure 4(b)). Figure 4(c) shows the result matrix after merging operations.

5.2.4 Exclusion of Variables and Tables. During the construction, the user can exclude an undesired variable E or a table V from $\boldsymbol{H}$ by deleting the corresponding column or row. The user can put the deleted tables or variables back into the incidence matrix by the inclusion operation.

5.2.5 Dataset View. When the number of variables or tables is large, it will be inconvenient to drag a column or a row in the matrix. A dataset view (Figure 4(e)) is provided which supports the same set of construction operations. The linking operation can be done by connecting two variables with a link (see purple lines in Figure 4(e)). The merging operation is performed by dragging a table into another (see blue line in Figure 4(e)).

Since it can still be labor-intensive to perform search and link operations among large amounts of variables, hints for linking operations will be user-friendly. Specifically, two variables whose level sets and names are identical may be a candidate pair for the linking operation. The user has the option to show these pairs with dotted lines in the dataset view, such as the dotted line between "County" of V_2 and V_3 in Figure 4(e).

Finally, the user is also able to undo the construction operations to restore a previous constructed hypergraph model.

6 VISUAL MODULATION OF HYPERGRAPH LEARNING

With the constructed hypergraph, the user is allowed to adjust the parameters of the model and run Algorithm 1 to see the learned labels $\boldsymbol{f}$. To be more specific, μ, ρ, and the prior hyperedge weights $\boldsymbol{w_d}$ (Equation (6)) can be modified interactively according to the domain knowledge or the feedback of the visual exploration. This process can be repeated until the user is satisfied with the learning results.

In this section, we first introduce the approach we designed for injecting the prior weights (Section 6.1), then we will discuss methods that deal with scalability and usability of the weight modification (Sections 6.2 and 6.3).

6.1 Setting the Prior Weights of the Hyperedges

For a better understanding of the hyperedge weights, we use bar charts above the columns to visualize the learned weights $\boldsymbol{w}$ and the prior weights $\boldsymbol{w}_d$, as shown in the white and gray bars, respectively, in Figure 4(d). For an aggregated column of a variable E, the weights are defined as the average $\overline{w_d}$ and $\overline{w}$ of its levels $e \in E$.

To support the modification of the prior weight $w_d(e)$, we initially designed an interaction on the bar charts, where the user could adjust the bar length of column e to set $w_d(e)$. However, in our experiments, we found that asking the user to set the exact value of $w_d(e)$ was not practical, since (s)he may not understand the meaning and effect of the exact value of a weight. But the user may have an idea about setting the change ratio of $w_d(e)$. As a result, we improve the interaction by providing a few popular options, such as half the w_d value, double the w_d value, or set w_d to 0 or the maximum value in $\boldsymbol{w_d}$. Alternatively, the user can also input a self-defined change ratio.

Each time the weights are changed, the learning results will be updated. Performance information such as test accuracy is provided so the user can evaluate the effects of the modification. The test accuracy also reveals whether the change caused overfitting. Since the user may want to try different $\boldsymbol{w_d}$ to find better learning results, there is an undo operation for the modification. Finally, our system can also be asked to return to a previous analysis step to view the learning results under a previous $\boldsymbol{w_d}$.

6.2 Reordering of the Hyperedges

Since there are large numbers of hyperedges, it will be time-consuming for the user to examine and modify all the weights. To deal with this scalability problem, we can focus on the hyperedges which have major influences in the model. According to Equation (4), hyperedges with higher weights w have more significant effects on the learning results. Therefore, it is preferable to inspect and

adjust the hyperedges with higher weights first. A simple reordering of the columns by $\boldsymbol{w}$ enables this.

Figure 4(d) shows the incidence matrix ordered by the variable weights. The variables related to hospital ratings (e.g., "Nurse Rating") have higher weights than those related to counties (e.g., "Smoking Rate"), which means hospital ratings are regarded as important predictors of readmission by the algorithm. Because the weights $\boldsymbol{w}$ are affected by their prior $\boldsymbol{w_d}$, ranking the columns by $\boldsymbol{w_d}$ is also available.

6.3 Recommending the Prior Weights

During our experiments, we found that setting the prior weights $\boldsymbol{w}_d$ can be impractical in the following two situations: (1) At the beginning of our algorithm, it is laborious to ask the user to set the initial w_d of the hyperedges one by one. (2) The user is usually interested in only a part of the hyperedges and may not have priors for others.

Because of the above reasons, a method for recommending $\boldsymbol{w_d}$ would be more user-friendly, essentially providing the user the option to set $\boldsymbol{w_d}$ to some recommended values during the analysis. We calculate the recommended $w_d(e)$ by making use of the Hellinger Distance [36], which measures the divergence of learned labels f in a hyperedge e (Equation (10)). For example, with the Hellinger Distance, a disease e which contains only patient groups with low readmission levels will have high $w_d(e)$, and e can be recognized as an indicator of low readmission. In Equation (10), f_e^+ and f_e^- represent the set of positive and negative instances in e, respectively; f^+ and f^- represent the set of positive and negative instances in all the datasets, respectively. Other weight measures can be adopted as well [15, 19] in our algorithm.

$$w_d(e) = \left(\sqrt{\frac{|f_e^+|}{|f^+|}} - \sqrt{\frac{|f_e^-|}{|f^-|}} \right)^2 \tag{10}$$

Via the recommended $\boldsymbol{w}_d$, the user is now free to only focus on the weights of those hyperedges of interest. For example, the user may halve the w_d of the hyperedge "Newborns" and keep the recommended values for the rest of the hyperedges.

The recommended weights bring several additional benefits. First, hyperedges e with high $w_d(e)$ will have low diverse labels, which can be a good predictor of labels. Sometimes $\boldsymbol{w}_d$ can be a better indicator than $\boldsymbol{w}$ since $\boldsymbol{w}$ needs to satisfy the vertex degree constraint in Equation (6), while the recommended $\boldsymbol{w_d}$ is only based on the result $\boldsymbol{f}$. In our system, ordering the matrix by either $\boldsymbol{w}$ or $\boldsymbol{w_d}$ is enabled for the user to explore potentially important hyperedges. Second, the user does not have to assign an initial value $\boldsymbol{w_d}$ for the algorithm. Instead, a default $\boldsymbol{w_d} = [1, 1, \ldots, 1]^T$ is used, which can be updated later by the recommended values based on the learning results.

7 HYPERGRAPH VISUALIZATION: EXPLORATION AND VALIDATION OF THE LEARNING RESULTS

Although the visual exploration of the learned labels $\boldsymbol{f}$ provides insights both into the model as well as into the effects of a user's modifications, this type of interaction is typically not available in most of the existing learning approaches. Specifically, the following tasks are usually performed by the user during the visual exploration and validation: **T1**: Examine the learning results of vertices or hyperedges of interest. **T2**: Examine the learned distributions of the intersections of different hyperedges. For example, the user may want to interactively query the learned readmission levels of the patient group from "Brooklyn" (e_1 in Figure 5) who suffer from "Fever" (e_2 in Figure 5). **T3**: Compare the learning results of different vertices in a hyperedge. **T4**: Examine the changes of the learning results after updating the model.

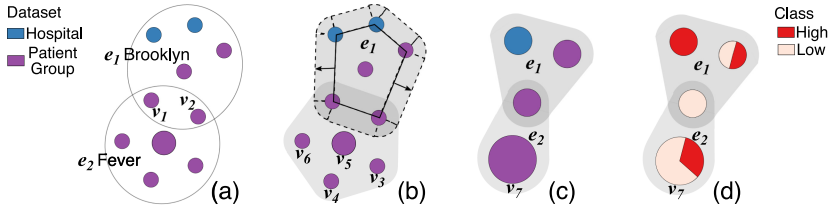

Fig. 5. (a) The force-directed layout of the vertices in two hyperedges e_1 and e_2. (b) For each hyperedge, the convex hull is calculated and the curved contour is generated by extending the convex hull. (c) Vertices from the same dataset can be aggregated. (d) The distribution of the learned labels of each aggregated vertex can be encoded with a pie chart.

7.1 Design Rationales

Learning result visualization is necessary to support the exploration tasks. One straightforward strategy is showing the labels as an additional column in the incidence matrix $\boldsymbol{H}$. However, there are some drawbacks to presenting the learning results in the matrix. First, the distributions of the learned labels in a hyperedge (**T1**) are not visualized. Second, the matrix can be large, which makes it time-consuming for the user to check the vertex labels (**T1**) row by row. Third, it is difficult to find the hyperedge-hyperedge (**T2**) or vertex-vertex (**T3**) relations in the matrix visualization.

As discussed in Section 2.2, contour visualizations allow the intuitive exploration of the learning results, especially for **T1** - **T3**. Some approaches (e.g., Bubble Sets [13]) are not applicable in our case since they require the input of a vertex layout. In contrast, Vizster [18] proposes an effective solution which generates contours based on the force-directed layout [14] of an ordinary graph. Following Vizster, our approach extends the force-directed layout for hypergraph visualization.

7.2 Force-directed Hypergraph Visualization

In initial studies with a prototype of our system we observed that during analysis users typically focused only on a small set of hyperedges at any one time. Hence, our system in normal operation only visualizes a subset E_I of interesting hyperedges which can be selected by the user (e.g, E_I = {e_1:"Brooklyn", e_2:"Fever"} in Figure 5(a)).

7.2.1 Force-directed Layout of Vertices. To minimize the visual clutter generated by unnecessary contour overlap, an optimized layout is expected to satisfy the following principles: (i) Vertices sharing more hyperedges should be closer, which keeps the area of a hyperedge compact. (ii) Vertices with no common hyperedge should be placed as far apart as possible, which reduces the possibility of an overlap of disjoint hyperedges.

Our algorithm positions the vertices by assigning forces among them. For two vertices $v_i, v_j \in V \subseteq E_I$, their mutual attractive force $f_a(v_i, v_j)$ and repulsive force $f_r(v_i, v_j)$ are defined according to principle 1 and 2, respectively (Equation (11)).

$$f_a(v_i, v_j) = \frac{m \cdot dis(v_i, v_j)^2}{k} \quad \text{and} \quad f_r(v_i, v_j) = -\frac{k^2}{dis(v_i, v_j)} \tag{11}$$

In Equation (11), k is a constant and $dis(v_i, v_j)$ is the distance of v_i and v_j. The difference between Equation (11) and the original force-directed algorithm [14] is that we multiply f_a by a factor m. m is defined as the number of common hyperedges which v_i and v_j are in: $m = \sum_{e \in E_I} h(v_i, e)h(v_j, e)$. For example, $m = 2$ for v_1 and v_2 in Figure 5(a) because they share 2 common hyperedges in E_I: e_1 and e_2. As a result, vertices that share more common hyperedges will be placed closer by f_a.

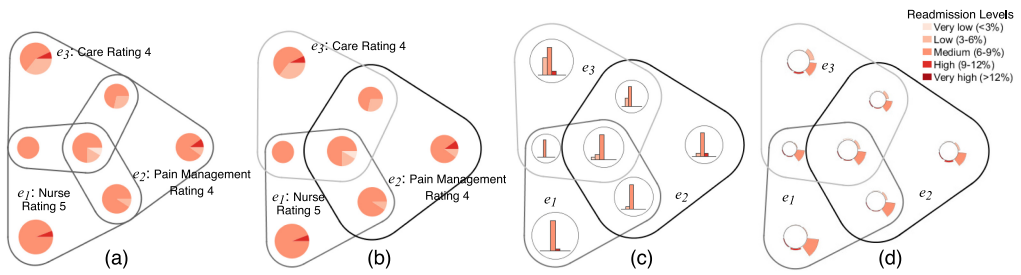

Fig. 6. A subset of interesting hyperedges $E_I = \{e_1, e_2, e_3\}$ are selected from Figure 4 (d) and presented in the contour visualization. The hyperedges contain the aggregated vertices of high-rating hospitals. The label distributions inside the vertices show the learning results of Algorithm 1. The readmission levels of these hospitals are mainly labeled as "Low" or "Medium". It suggests that high ratings could be good indicators of low hospital readmission. Different design options are provided for contour and vertex visualizations. (a) Contours with the same size and grayscale. (b) The contour borders are adjusted to different sizes and grayscales, which makes the contours more distinguishable. (c) The label distributions of vertices are visualized with histograms. (d) The distributions are shown in aster plots.

Then the regular force-directed layout algorithm is used to calculate the positions of the vertices (Figure 5(a)).

Each vertex is visualized as a circle, whose size is encoded with the vertex weight $u(v)$. For the color of a vertex, we provide different options to the user, such as the class label or the dataset type.

7.2.2 Contour Visualization of Hyperedges. Because the user will set E_I interactively during the exploration, the contours of the hyperedges should be generated in real time. Some approaches (e.g., Bubble Sets [13]) fail to meet this requirement due to their high complexities. Using the same method as Vizster [18], we create the contour of a hyperedge by calculating the convex hull of its vertices (see solid line in Figure 5(b)). The curved contour is computed by extending the convex hull outwards (see dashed line in Figure 5(b)). Other methods, such as quadratic splines, can also be used to get a smooth border [18]. The user is able to change the hypergraph layout manually by dragging the vertex. Then the contours will also be recalculated according to the updated vertex positions.

After some experiments, we decide to leave the contour areas uncolored. This keeps the vertex colors well visible and also avoids blending issues when differently colored hyperedges overlap. The omission of fill colors, however, can lead to ambiguities. Figure 6(a) shows an example of three overlapped hyperedges $e_1 - e_3$, which are selected from Figure 4(d). Initial testing with a prototype reveals that sometimes it is difficult to recognize the hyperedges (Figure 6(a)). In order to make the contours more distinguishable in the general case, we randomly resize the contours slightly to avoid any ambiguity of the contour lines (Figure 6(b)). Finally, for further disambiguation, we can also use different grayscales for the contour lines.

7.2.3 Vertex Aggregation. The calculation of the convex hull and the visualization of large numbers of vertices are both challenging when the number of vertices in E_I is large. We deal with this problem by aggregating vertices that are of the same data type and have exactly the same inclusion relations in E_I. For two vertices $v_i \in V$ and $v_j \in V$, we aggregate them if $\forall e \in E_I, h(v_i, e) = h(v_j, e)$. Figure 5(c) shows an example of a hyperedge visualization with aggregated vertices. Patient groups which are not in "Brooklyn" and have "Fever" (v_3 - v_6) are merged into vertex v_7.

Table 2. Visual Encodings of the Matrix Visualization and the Contour Visualization

Data	Notation	Matrix Visualization	Contour Visualization
An instance	A vertex v	A non-aggregated row	A non-aggregated vertex
A level of a variable	A hyperedge e	A non-aggregated column	A contour
A dataset	V	An aggregated row	Vertex color (optional) (Figure 5(a))
A variable	E	An aggregated column	-
The label of an instance	$f(v)$	-	Vertex color (optional) (Figure 5(d))

The aggregated vertex weight, which is encoded by the vertex size, is the sum of its original vertex weights. The data type of an aggregated vertex can be encoded with color. However, the class information cannot be represented by a single color, since the original vertices may have different labels. To show the class distribution in an aggregated vertex, we provide several design options, including pie chart (Figure 6(b)), histogram (Figure 6(c)), and aster plot (Figure 6(d)). For an aster plot, the size of the inside circle represents the weight and the lengths of the outside pie slices encode the label distribution. The users in our studies preferred the pie chart, as they found a label corresponding to a small magnitude (e.g., "High (9-12%)" in Figure 6) in the aster plot or histogram was too small to be seen. Later in the study, one of our users mentioned that comparing distributions of different vertices with a pie chart may not be as easy as with a histogram. We solve this problem by adding histogram visualizations of interesting vertices in a more detailed view (Section 7.3).

Each time the learned labels are updated, the distributions of the vertices will also change accordingly. The user can also switch to the visualization without vertex aggregation if she or he wants to see the details of the original hypergraph structure. The visual encodings of the visualizations proposed in this article are summarized in Table 2.

7.3 Visual Exploration

The user can query interesting results by setting the hyperedge subset E_I. This can be done by selecting/deselecting hyperedges of interest either in the dataset view or in the incidence matrix during the exploration (e.g., $E_I = \{e_4, e_5\}$ in Figure 7(a)). The vertex aggregation and the contours are recalculated upon change of E_I.

The hyperedge visualization as presented so far is incomplete since the learned class distribution of a hyperedge (**T1**) is not visualized, such as the overall label distribution of hyperedge e_5 in Figure 7(d). In addition, it is also difficult to discern the distribution in the pie chart of an aggregated vertex with small size. For these purposes, we provide a detail view to show the learned class distribution of an aggregated vertex or a hyperedge (Figure 7(e)). To help the user understand the effects of his or her modulation (e.g., updating w_d) on the learning results, both the results before and after the user's modulation are shown with gray and yellow histograms, respectively. The user can click on a vertex (e.g., the highlighted vertex in e_4 in Figure 7(d)) or a hyperedge (e.g., the highlighted hyperedge e_5 in Figure 7(d)) to view its distributions. Because the histograms are vertically aligned, this view also helps to compare the change patterns of different vertices and hyperedges (**T4**).

8 CASE STUDIES

We conducted two case studies, which focused on topics in different fields, with two scientific collaborators (call them SC1 and SC2). SC1 was a graduate student majoring in Biomedical Informatics who was interested in predicting the readmission levels of hospitals in New York State. However,

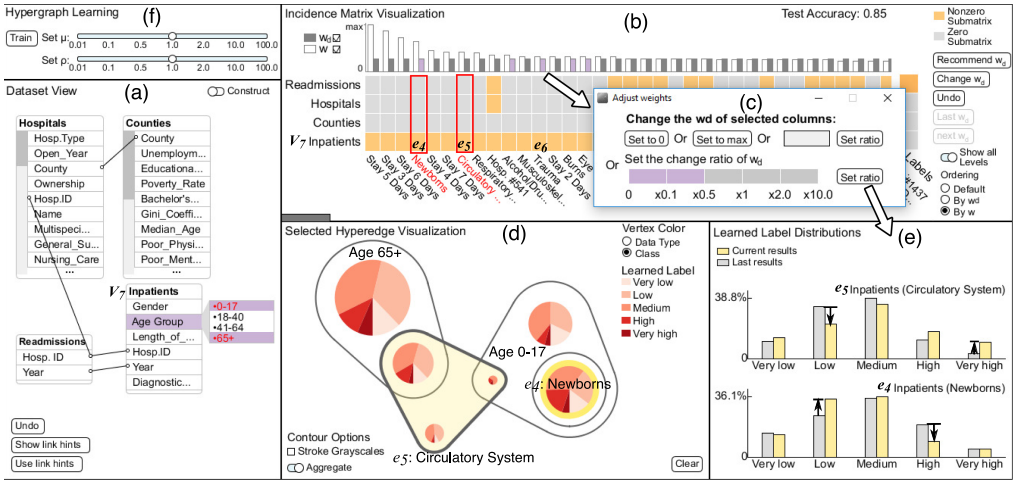

Fig. 7. The interface of our system. (a) The dataset view after adding the inpatient discharges dataset (V_7). (b) In the non-aggregated incidence matrix, all levels are ranked by their weights $\boldsymbol{w}$. (c) Reducing the prior weights w_d of the selected diagnosis categories, as shown in the purple bars above the matrix in (b). (d) Selecting four hyperedges of "Age 0-17," "Age 65+," "Newborns," and "Circulatory System" from the dataset view and the incidence matrix. The readmission levels of the patient groups in these hyperedges are shown in the contour visualization. Highlighting the hyperedge of e_5 and the vertex in e_4 to show their learned distributions. (e) There are different patterns of changes in the distributions of "Newborns" and "Circulatory System" after the modification of the weights $\boldsymbol{w_d}$. (f) Adjusting the hyper parameters of the learning algorithm.

he could not find a complete table including all the potential risk factors of readmission, and so our system offered him a welcome opportunity to make such prediction using disparate data sources. SC2 was a Political Science graduate student with a research focus on the public opinion of the recent US election. He wanted to analyze the 2016 election polls by comparing the real election results and the results predicted by the polls. Both users were not experts in visualization and had no prior knowledge about hypergraphs. The summaries of the two datasets are shown in Table 3.

Before the study, we had a number of thorough discussions with SC1 and SC2 on their problems. Then we searched online for additional sources to supplement their datasets. Each case study started with a training session to introduce our system. We then asked each participant to use the system, followed by an interview to gather evaluations and subjective feedback.

8.1 Case Study 1: Readmission Prediction of Hospitals in New York State

SC1, our public health collaborator, used our system to predict the readmission levels of hospitals using a collection of history records. He had a dataset of hospital Potential Preventable Readmission (PPR) rates (V_1) from the New York Health Department website [33]. We labeled each readmission instance as one of five levels ("Very low" to "Very high" in Figure 6) according to its PPR rate in the dataset. The readmissions from 2011 to 2013 were used as the labeled training data, and the readmissions in 2014 were the test data for our algorithm.

The readmission dataset contained only two variables "Year" and "Hosp. ID," but SC1 was interested in learning more potential readmission factors. We gathered supplementary information of hospitals from the Medicare Open Data [30], including their general information (V_2), structural measures (V_5), evaluations (V_6), and the patient satisfaction surveys (V_4). The inpatient discharge dataset (V_7) [33] was also obtained which described the details (e.g., age group and diagnostic

Table 3. A Summary of the Heterogeneous Datasets That Our Study Participants Used in the Readmission and Election Case

NY Hospital Readmission Case			
Datasets	Number of variables	Number of levels	Number of instances
V_1 Readmission (PPR)	3	219	874
V_2 Hospital information	6	556	250
V_3 County information	17	110	62
V_4 Patients' surveys	13	246	151
V_5 Hospital measures	9	217	155
V_6 Hospital evaluations	11	240	173
V_7 Inpatient discharges	7	272	9,750
US Presidential Election Case			
Datasets	Num. of variables	Num. of levels	Num. of instances
V_1 History election results by state	3	57	51
V_2 History election results by county	3	3,118	3,112
V_3 State facts	8	108	51
V_4 County facts	13	3,196	3,112
V_5 2016 polls by state	111	381	51

Some numerical values in the datasets were discretized into categorical levels.

category) of the inpatient groups of each hospital. To check if the location had effects on readmission, census information of counties in NY state (V_3) was collected [6]. We discretized the numerical variables (e.g., "Smoking Rate") in the county dataset into three levels: above, around, and below state average. The first table in Table 3 summarizes all datasets V and their detailed information.

8.1.1 Finding Hospital-level Risk Factors of Readmission. SC1 started his prediction of readmission with a focus on the hospital factors. He selected the readmissions (V_1), counties (V_3) and the hospital datasets (V_2, V_4 - V_6) in the dataset view (Figure 4(e)) to generate the initial incidence matrix (Figure 4(a)). By dragging the columns in the matrix, he linked the readmissions (V_1) and the hospital datasets (V_2, V_4 - V_6) by "Hosp. ID." The hospitals (V_2) were also linked to the counties (V_3) by the hospital locations (Figure 4(a)). Because the hospital datasets (V_2, V_4 - V_6) described the same set of hospital instances, SC1 merged V_4 - V_6 into V_2 by dragging their rows in the matrix (Figure 4(b)).

With the constructed hypergraph, SC1 trained the model using Algorithm 1 with the default $\boldsymbol{w_d} = [1, 1, \ldots, 1]^T$. The algorithm showed the test accuracy was about 0.80. To find out what factors of readmission were important according to the algorithm, SC1 ranked the variables by the learned weights $\boldsymbol{w}$ (Figure 4(d)). He discovered that the variables related to patients' surveys had the highest weights, such as "Nurse Rating," "Care Rating," and "Pain Management Rating." He expanded these variables and selected their levels of the highest rating (e.g., rating 4 of "Care Rating" in Figure 4(d)) for detailed inspection in the contour visualization (see e_1, e_2, and e_3 in Figure 6(b)). The hospitals were shown as the vertices in the contours. For example, the aggregated vertices in e_1 represented the hospitals whose "Nurse Rating" is 5. From the pie charts of the aggregated vertices, he found that the most hospitals contained in these hyperedges were labeled as "Low" to "Medium" by the algorithm. He confirmed that good quality of health care could solve the underlying problems and prevent unnecessary readmissions. Then, by exploring the remaining portion of the incidence matrix, he noticed that the weights w of county census variables were low, such as the "Smoking Rate" of each county (Figure 4(d)). This indicated that those variables were less important with respect to readmission. He suggested that this might be because the county

dataset described the census of the total population, while the hospital readmission levels were only decided by the inpatients.

8.1.2 Finding Risk Factors of Readmission in Patient Groups. To find factors of readmission in the inpatient information, SC1 modified the hypergraph by adding inpatient discharges (V_7) (Figure 7(a)). Then the updated model showed an accuracy of 0.85. He ranked all the levels by $\boldsymbol{w}$ (Figure 7(b)) in the incidence matrix, then started to examine the hyperedges with high weights w to find important predictors of readmission. He noticed that the levels in "Length of Stay" might be closely related with readmission because their weights were among the highest. He guessed that the length of stay at hospital indicated the severity of the illness, which had correlation with readmission.

He found that there are several diagnosis categories with high weights, such as "Newborns" (e_4) and "Circulatory System" (e_5). He visualized the readmission distributions of the patient groups in those diagnosis categories and different age groups (e.g., "Age 0-17" and "Age 65+"), as shown in Figure 7(d). From the pie charts of the vertices, he learned that the patient group of "Newborns" (e_4) had a higher proportion of "High Readmission" than others. He told us that "Newborns" was not supposed to be a factor of potential preventable readmission (PPR). According to his knowledge, "Newborns" admissions were rarely related to their previous admissions, and so few of them were regarded as re-admission [17]. He continued his examination on other diagnosis categories with high weights. He expressed that some categories such "Trauma" (e_6) were also usually excluded in potential preventable readmission rate calculations, since they were always not "preventable" [17]. After this inspection, he selected some of the diagnostic categories in the matrix (see the purple bars above the matrix in Figure 7(b)) and reduced their prior weights w_d by half (Figure 7(c)).

With the updated $\boldsymbol{w_d}$ in the model, he ran the learning algorithm again. The final accuracy was improved to around 0.89. To see the detailed changes of learned labels, he selected the hyperedges of "Circulatory System" and the vertex in "Newborns" (Figure 7(d)) to visualize their detailed label distributions. He found that more "Newborns" readmissions became "Low" after the modification of $\boldsymbol{w_d}$. While the overall distribution of "Circulatory System" had the opposite change pattern, it skewed to higher readmission levels (Figure 7(e)). He confirmed that some circulatory system diseases such as heart arrest had high possibilities of readmission.

8.2 Case Study 2: Bias of US Presidential Election Polls

The second participant (SC2) focused on the prediction of the 2016 US presidential election. He collected the historical presidential election results by state (V_1) and by county (V_2) [39] as the labeled data. We labeled the instances in V_1 and V_2 as "Republican," "Democrat," or "Others." The election results of 2004, 2008, and 2012 were used as the training data and the results by state in 2016 were used as the test data.

He also gathered the election polls by state in a month before the election date (V_5) [34], which contained more than 100 poll providers. In the poll table, the value of a poll in a state was assigned to "Clinton," "Trump," or "Others" according to the support rates in the poll, as shown in table V_5 in Figure 8(a). Since each poll provider surveyed a part of the states, there might be empty values in the poll table. For example, "CBS News/YouGov" did not survey "FL" (see table V_5 in Figure 8(a)).

To see if there were other factors related to the election results, we added the census datasets of the states (V_3) and the counties (V_4) in the US [6]. They contained the demographic information such as the population density and bachelor's degree rate. We discretized each numerical variable in these datasets into three levels: above, around, and below federal average. The summary of the datasets is shown in the second table in Table 3.

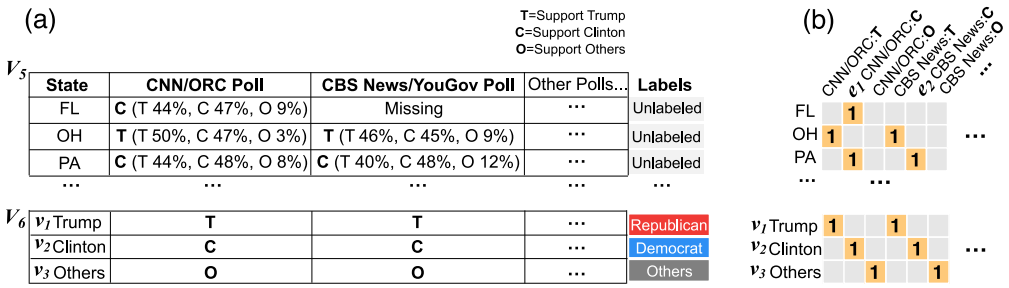

Fig. 8. (a) The top table shows the details of the poll dataset (V_5). Each poll is a categorical variable of three levels: "C," "T," and "O." We added the bottom candidate table (V_6) as the labeled data for training. (b) Incidence matrices of V_5 and V_6 respectively showed set relationships of the states and candidates to the polls.

8.2.1 Prediction by History Election Results. SC2 practiced using our system at the beginning of the study. He first constructed a hypergraph of the states, counties and their history election results (V_1 - V_4). He linked V_2, V_4 by variable "County Fips" (i.e., county ID) and V_1 - V_4 by "State." Then he trained the model with default $\boldsymbol{w_d}$, the result suggested that Democrat would win 347 out of 538 electoral votes. However, the Republicans won the majority of the electoral votes in the 2016 election. After checking the model, he admitted that using the history data might generate significant error because the election candidates of the parties were much different from the past.

8.2.2 Prediction by Election Polls. Because the history results did not help much and even misled the prediction, SC2 removed these two datasets (V_1 and V_2) and added the 2016 election polls (V_5) in the dataset view (Figure 9(a)) to update the hypergraph structure. Then we realized that there was no labeled data for training after the removal of the history results, so we added three candidates (V_6) as the labeled instances for the training of the model. To link them with the polls, we put the candidates into the corresponding levels of the polls (see table V_6 in Figure 8(a)). For example, Democratic party candidate Clinton (v_2) occurred in the "Clinton" hyperedges of all the polls, as shown in the set representation in Figure 8(b).

SC2 linked and merged the state (V_3) and poll (V_5) dataset by "State" in the dataset view (Figure 9(a)). Then he trained the model and found that 46 out of 51 states (50 states + D.C.) were correct (448/538 votes). However, Clinton still would win the majority votes according to the results. Because the model was still not satisfactory, he told us that he was interested in finding which poll providers had major influences on the learning results. He calculated $\boldsymbol{w_d}$ for all the variables with the Hellinger Distance (Section 6.3). By reordering the incidence matrix, he noticed that the polls of "CNN/ORC," "NBC/WSJ/Marist," and "CBS News/YouGov" had the highest $\boldsymbol{w_d}$ values (Figure 9(d)). He selected their levels of "Clinton" (see e_1, e_2, and e_3 in Figure 9(b)) for further validation in the contour visualization.

The states supporting Clinton in the selected poll were visualized in the hyperedge contours (Figure 9(c)). For example, e_1 contained the vertices of FL, PA, and NC because they supported Clinton according to the "CNN/ORC" poll (Figure 8). All the vertices were blue because our learning algorithm also labeled those states as "Democrat.'. SC2 told us that some of these states actually voted for Trump, such as Florida (FL) and North Carolina (NC). According to his knowledge, these polls were possibly biased. SC2 showed us some reports mentioning that media such as NBC, CBS, and WSJ had certain preferences for the Democratic party [42]. He selected these polls and set the change ratio of their prior weights w_d to 0.1 (Figure 9(d)). With the modified $\boldsymbol{w_d}$, he ran the algorithm again, and more states turned correct (49/51 states, 508/538

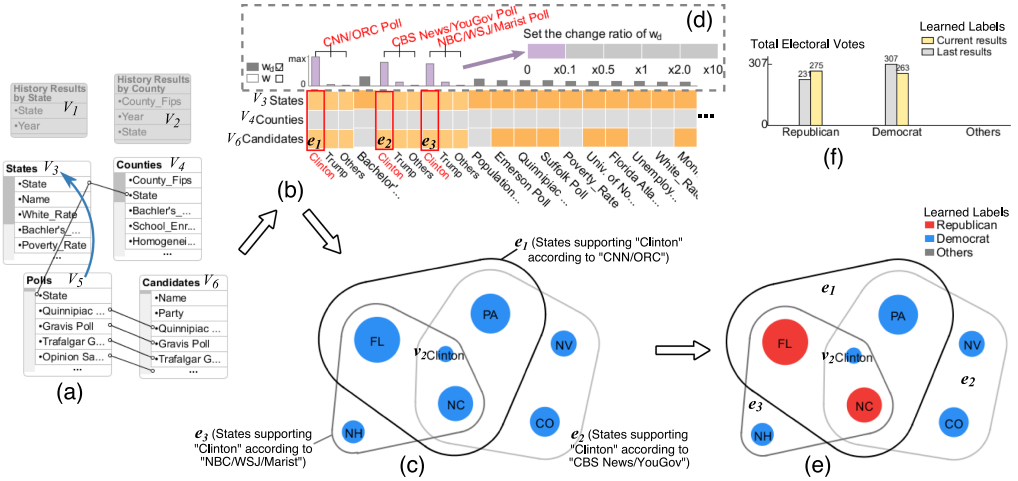

Fig. 9. (a) In the US presidential election case, the user constructed a hypergraph structure on four heterogeneous datasets (V_3 - V_6) in the dataset view. (b) The constructed structure was represented in an incidence matrix, which was ranked by $\boldsymbol{w_d}$. Some variables ("CNN/ORC Poll," "NBC/WSJ/Marist Poll," and "CBS News/YouGov Poll") had high w_d values, indicating they played important roles in the prediction. (c) Three hyperedges (e_1, e_2, and e_3) selected from the incidence matrix were visualized as contours. Each hyperedge contained states supporting "Clinton" in its poll (e.g., FL, NC, PA in e_1). The candidate vertex v_2 was also in the edges, as we added the candidates in the poll (see V_6 in Figure 8). The color of the vertices encoded the learned labels. (d) The user reduced the prior weights w_d of some polls since they might be biased. (e) After running the learning algorithm again with the updated $\boldsymbol{w_d}$, some of the learned labels were changed (e.g., the states FL and NC turned into Republican). (f) The histograms showed the label distributions before and after the change of weights.

votes), including Florida and North Carolina (Figure 9(e)). The total electoral vote distributions before and after changing $\boldsymbol{w_d}$ were shown in Figure 9(f). Our model failed to predict only two states: Pennsylvania and Wisconsin. SC2 checked the poll dataset (V_5) and found this was because almost all the polls predicted them as Clinton, but they actually voted for Trump.

During the examination of the poll providers, an interesting observation SC2 found was that most university polls showed Clinton would win, such as "Univ. of North Florida" and "Florida Atlantic Univ."

He continued his exploration and noticed that several variables about the state census had high weights, such as "Bachelor's Degree Rate" and "Population Density." He expanded these variables and found that it was because their levels of "Above federal average" had very high w_d values (see e_4 and e_5 in Figure 10(a)). To examine their detailed class distributions, he selected these two levels and visualized them in the contour visualization. From the pie charts of the aggregated vertices of states, he found that states with both high population densities and high bachelor's rates were all predicted as "Democrat," which suggested that the combination of these two hyperedges could be a good indicator of election results (see the blue vertex in the intersection of e_4 and e_5 in Figure 10(b)). He also noticed that the states that were not contained by e_4 and e_5 were mainly Republican.

8.3 Quantitative Analysis

Based on the datasets and settings of the two case studies, we evaluate the performance, the parameter selection, and the complexity of our algorithm.

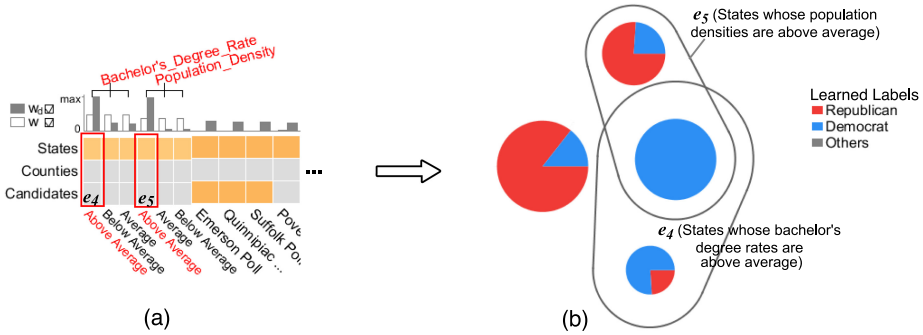

Fig. 10. (a) The user selected two hyperedges of "Above Average Population Density" and "Above Average Bachelor's Degree Rate" that had high weights. (b) The aggregated vertex of states in the intersection of the two hyperedges was blue, indicating these two hyperedges together were good predictors for the Democrat party, while the states outside both hyperedges were mainly Republican.

Table 4. Performance Comparison of Four Algorithms in Two Cases

Methods	Readmission Case			Election Case		
	Test Error	F1	AUROC	Test Error	F1	AUROC
TSVM	0.22	0.80	0.88	0.10	0.93	0.93
LBC	0.21	0.78	0.88	0.10	0.93	0.93
HG_T	0.15	0.85	0.89	0.10	0.93	0.94
Ours	**0.11**	**0.88**	**0.90**	**0.04**	**0.96**	**0.97**

Bold indicates the best performance.

8.3.1 Performance Comparison. We compare our algorithm with three baseline algorithms: Transductive Support Vector Machine [21], Link-based Classification [28], and traditional hypergraph learning [44], which are denoted as TSVM, LBC, and HG_T in Table 4, respectively. To train the TSVM with the complete datasets, we use the fused incidence matrix $\mathcal{H}$ constructed by each user as the set of input features. After testing different kernels, we choose the radial basis function (Gaussian kernel), which achieves the best performance for TSVM in our datasets. The ordinary graph structures of LBC are constructed manually with pairwise links on the datasets in the two cases. Logistic regression is employed as the local classifier in the iterative class assignment process of LBC. The hypergraph structures constructed in the two studies are adopted for both the traditional hypergraph and our algorithm. $\boldsymbol{w}$ in HG_T (Equation (2)) is set to $[1, 1, \ldots, 1]^T$, which is usually used as the default value [44]. For our algorithm, $\boldsymbol{w_d}$ set by the users in the studies are employed.

The test error, macro-average F1 score for multi-class, and macro-average area under the Receiver Operating Characteristic (AUROC) for multi-class are employed as the evaluation metrics. The results of the two cases are shown in Table 4. Our approach achieves the best performance among the compared methods. Especially, in the election case, our method is the only algorithm which predicts Trump's victory although other baseline methods also have high performances. The main reason for it may be that our algorithm allows the user to tune the parameters during visual analysis, which is critical for improving the learning model.

We perform 10-fold cross-validation to detect if there is overfitting in our approach. We use the $\boldsymbol{w_d}$ set by the users in the cases. The F1 scores of the cross-validation in the readmission and election case are 0.87 and 0.91, respectively. They are close to the F1 scores in the case studies (Table 4), indicating our models with the user-defined w_d do not overfit.

Table 5. The Average Convergence Iterations and the Average Test Errors of Algorithm 1 with Respect to Different μ and ρ

Settings	Iteration Number		Test Error	
	Readmission	Election	Readmission	Election
$\mu = 1, \rho = 0.01$	5.0	4.0	0.13	0.10
$\mu = 1, \rho = 0.1$	3.3	3.0	0.13	0.10
$\mu = 100, \rho = 1$	4.3	4.0	0.14	0.12
$\mu = 10, \rho = 1$	2.7	2.0	0.13	0.12
$\mu = 1, \rho = 1$	4.0	3.3	0.12	0.06
$\mu = 0.1, \rho = 1$	3.0	2.0	0.12	0.10
$\mu = 0.01, \rho = 1$	2.0	2.0	0.16	0.12
$\mu = 1, \rho = 10$	2.0	2.0	0.14	0.12
$\mu = 1, \rho = 100$	2.0	2.0	0.14	0.12

Our algorithm converges in constant time in both cases. The performance of our algorithm does not degrade much with different parameter settings.

8.3.2 Parameter Settings. To find the effects of different μ and ρ values on the learning results in Algorithm 1, we test different settings of (μ, ρ). We adopt the hypergraph models that the users constructed at the end of their studies. We ask the user to define a set of $\boldsymbol{w}_d$ and use them to test the average iteration steps for convergence and the average test errors. The threshold ϵ in Algorithm 1 is set to 0.1. The experiment results in the two cases are shown in Table 5.

Rows in Table 5 are ordered according to the value of ρ. When ρ increases, Algorithm 1 converges faster. We conclude that a larger ρ makes the weight regularization term Ψ more important in Equation (9). As a result, $\boldsymbol{w}$ will converge with less iterations, which leads to faster convergence of Algorithm 1.

Compared to ρ, μ values have less effects on the iteration numbers. By examining the learning results, we find that the learned labels $\boldsymbol{f}$ will become closer to $\boldsymbol{y}$ when μ is larger. Therefore, a very small or very large μ will cause underfitting or overfitting of the algorithm.

The default (μ, ρ) is set to $(1.0, 1.0)$, since Algorithm 1 has the lowest test error around $(1.0, 1.0)$ in our experiments (Table 5).

8.3.3 Complexity of the Algorithm. The complexity of Algorithm 1 is mainly decided by the complexity of Equation (8), Equation (9), and the number of iterations. The complexity of matrix solving in Equation (8) is $O(|\mathcal{V}|^3)$, $|\mathcal{V}|$ is the number of vertices. The quadratic programming in Equation (9) can be solved with interior-point algorithm [4] with $O(|\mathcal{E}|^3)$ complexity, $|\mathcal{E}|$ is the number of hyperedges.

According to Table 5, our algorithm converges within five iterations for both cases with different (μ, ρ). This indicates our iterative approach for solving optimization of Equation (6) will not increase the total complexity. Overall, the complexity of Algorithm 1 is $O(|\mathcal{V}|^3 + |\mathcal{E}|^3)$.

The complexity of the traditional hypergraph learning algorithm is $O(|\mathcal{V}|^3)$, since it presets the weights $\boldsymbol{w}$ and does not need to solve Equation (9). The complexity of our algorithm is not much worse than the traditional algorithm when $|\mathcal{E}|$ and $|\mathcal{V}|$ are of the same order.

When the size of the heterogeneous data is big, there are some strategies to accelerate our algorithm. First, because we introduce vertex weight $\boldsymbol{u}$, it becomes possible to aggregate the instances that have identical inclusion relationship in the hyperedges. The vertex weight u of an aggregated instance will be the sum of weights of the original instances. For example, an individual patient can be aggregated into a patient group with weight as the number of patients. Second, Liu et al. [26] provided a trade-off between time and accuracy. They dealt with large data by splitting the vertices

into subsets and classifying the subsets in multiple runs. Third, pre-computation is enabled if the prior $\boldsymbol{w_d}$ will not be modified.

8.4 Feedback and Discussion

We evaluated the learning cost and usability of our visual system in the training session and the interview session, respectively.

8.4.1 Learning Cost. In the training session, our instructor gave a 15-minute demo to explain the hypergraph learning algorithm and our system. Then SC1 and SC2 practiced our system with the help of the instructor. They were free to stop practicing when they felt ready. To evaluate the learning progress, each participant was given a test of six exercises. The exercises tested the users' understanding of the proposed visualizations and interactions based on the case scenario. For example, an exercise for SC1 was linking "Hosp. ID" in "Hospital Measures" and "Hospital Evaluation." We observed that neither participant practiced for more than 12 minutes. In the tests, the participants were able to respond to all exercises as expected without any help from our instructor.

When asked to rate the learning cost of our system (1 = very hard, 5 = very easy), SC1 and SC2 gave ratings of 4.5 and 4, respectively. SC1 mentioned that it was easy because the matrix-based and contour-based visualizations followed commonly used set visualizations. SC2 told us at first he had some trouble interpreting the contour visualization because he was unfamiliar with the concept of the hypergraph learning. However, he understood our visual representation as soon as he found that it basically showed the set memberships of the data instances and the levels.

8.4.2 Usability. In the interview session, both participants rated the usefulness of our system at 5 (1 = very useless, 5 = very useful). SC1 and SC2 were also required to compare our system with their usual tools. After that, we asked them to give detailed evaluations of the matrix construction interactions, weight modulation, and the contour-based visualization.

SC1 mentioned that he used to join the data by database queries and search for interesting variables in a trial-and-error process until a satisfied result was achieved. Likewise, SC2 said that he typically would also fuse the data manually, which prevented him from updating the model in an incremental manner. Both participants commented that they often struggled with the raw data and statistics due to the lack of an effective visualization of the data structures and learning results.

For the interactions related to the matrix construction, SC1 mentioned that it enabled him to test different datasets and variables in a visual interface to find the risk factors of readmission. He confirmed that the design of the linking and merging operations was natural because they were equivalent to the join operation in a database. SC2 said, "I realized that using the history election results for prediction was not suitable, so changing the model during the analysis was necessary." Then he expressed that the supported interactions for changing the incidence matrix of the election datasets were helpful and convenient.

SC1 told us that "Setting the weights in the matrix allowed me to exclude the unrelated factors of readmission, such as the diagnosis category of 'Newborns.'" When asked about the strategy to adjusting the prior $\boldsymbol{w_d}$ in the presence of a large number of hyperedges, SC1 mentioned that he was interested in finding the most important factors of readmission. As a result, he would only examine the hyperedges with high weights and adjust them according to his domain knowledge. SC2 also indicated that ranking was useful for finding the hyperedges with important effects on the predicted election results. He agreed that reordering helped to deal with the scalability problem of adjusting weights of large numbers of hyperedges.

For the contour-based visualization, both SC1 and SC2 confirmed that it supported their visual exploration tasks (**T1–T4**). SC1 mentioned that he was able to view the distributions for all interesting hospitals and patient groups at a glance (**T1**), which provided insights into the potential readmission factors. SC1 also stated that the detail view (Figure 7(e)) was an efficient supplement to the contour-based visualization because it informed him about the result changes caused by his interactions (**T4**). SC2 noted that he discovered that many election polls might be incomplete or biased from this contour visualization. He commented that it was very intuitive for visualizing the structure of interesting polls and the predicted election results at the same time (**T2**, **T3**).

8.4.3 Discussions and Limitations. The participants also gave us some suggestions that we implemented in our system, such as setting the change ratios instead of absolute values of the weights. SC1 recommended to use wider and darker cells for aggregated columns (Figure 9(b)). SC2 suggested changing the sizes of different hyperedge contour lines to distinguish them.

One participant asked if he can predict other missing values (e.g., missing "Income Level" values of counties in Figure 1(a)) in addition to the labels. To get good performance on predicting "Income Level," we have to set the known "Income Level" as the new labels and construct a different hypergraph structure on another set of tables related to the census information.

During the visual modification process, the risk of overfitting can arise from the tuning of the weights. This is most likely to happen when the user assigns extreme weights to the selected hyperedges. To prevent this, a widely used solution is to add ℓ_2 regularization $\|\boldsymbol{w} - \boldsymbol{w_0}\|^2$ as a penalty term in Equation (6). Here $\boldsymbol{w_0}$ is the default value of $\boldsymbol{w}$ (e.g., $[1, 1, \ldots, 1]^T$). On the other hand, cross-validation can be performed every time the weight is changed to prevent overfitting.

For the existing contour-based visualizations, one common limitation is scalability [1]. The contours can become heavily cluttered due to dense overlapping, especially for more than seven hyperedges [40]. Our visualization can potentially be better than existing approaches because it avoids the color blending of the overlapped regions. In addition, our force-directed layout reduces most unnecessary overlapping.

During the interview, both SC1 and SC2 agreed that our strategy of only visualizing interesting hyperedges helped them filter out the less important factors. They told us that they usually focused on less than seven hyperedges at any one time during the visual exploration. In fact, this bears a natural relation to the well-known "Magical Number" 7 (plus/minus 2) that has been found to form an upper bound to a human's working memory capacity [32]. Hence, the problem with the limited scalability of the contour visualization is less likely to arise. SC1 also mentioned that the hyperedges rarely would completely overlap, which limited the risk for visual clutter.

9 CONCLUSION

We described a visual analytics approach for classification on heterogeneous data. We used the hypergraph paradigm to model the high-order relations of various objects in the datasets. Interactive construction, modulation, and exploration of the hypergraph learning model were supported in the proposed visualizations. Two case studies showed that our approach is better than existing learning methods in both performance and usability.

For future work, we would like to set the structure and parameters of the hypergraph model by various forms of domain knowledge, such as the knowledge graph of the heterogeneous data.

REFERENCES

[1] Bilal Alsallakh, Luana Micallef, Wolfgang Aigner, Helwig Hauser, Silvia Miksch, and Peter Rodgers. 2014. Visualizing sets and set-typed data: State-of-the-art and future challenges. In *Proceedings of the Eurographics Conference on Visualization (EuroVis'14)*. 1–21.

[2] Paolo Angelelli, Steffen Oeltze, Judit Haász, Cagatay Turkay, Erlend Hodneland, et al. 2014. Interactive visual analysis of heterogeneous cohort-study data. *IEEE Computer Graphics and Applications* 34, 5 (2014), 70–82.
[3] Fabian Bendix, Robert Kosara, and Helwig Hauser. 2005. Parallel sets: Visual analysis of categorical data. In *Proceedings of the IEEE Symposium on Information Visualization (INFOVIS'05)*. IEEE, 133–140.
[4] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge university press.
[5] Jiajun Bu, Shulong Tan, Chun Chen, et al. 2010. Music recommendation by unified hypergraph: Combining social media information and music content. In *Proceedings of the 18th ACM International Conference on Multimedia*. 391–400.
[6] US Census Bureau. 2018. QuickFacts. Retrieved from https://www.census.gov/quickfacts/table/PST045216/00.
[7] Mike Cammarano, Xin Dong, Bryan Chan, Jeff Klingner, Justin Talbot, Alon Halevey, and Pat Hanrahan. 2007. Visualization of heterogeneous data. *IEEE Transactions on Visualization and Computer Graphics* 13, 6 (2007), 1200–1207.
[8] K. Selçuk Candan, Huan Liu, and Reshma Suvarna. 2001. Resource description framework: Metadata and its applications. *SIGKDD Explor. Newsl.* 3, 1 (July 2001), 6–19. DOI: https://doi.org/10.1145/507533.507536
[9] Haidong Chen, Song Zhang, Wei Chen, Honghui Mei, Jiawei Zhang, Andrew Mercer, Ronghua Liang, and Huamin Qu. 2015. Uncertainty-aware multidimensional ensemble data visualization and exploration. *IEEE Transactions on Visualization and Computer Graphics* 21, 9 (2015), 1072–1086.
[10] Wei Chen, Zhaosong Huang, Feiran Wu, Minfeng Zhu, Huihua Guan, and Ross Maciejewski. 2017. VAUD: A visual analysis approach for exploring spatio-temporal urban data. *IEEE Transactions on Visualization and Computer Graphics*1 (2017), 2636–2648.
[11] Shenghui Cheng and Klaus Mueller. 2015. Improving the fidelity of contextual data layouts using a generalized barycentric coordinates framework. In *Proceedings of the 2015 IEEE Pacific Visualization Symposium (PacificVis'15)*. IEEE, 295–302.
[12] Shenghui Cheng and Klaus Mueller. 2016. The data context map: Fusing data and attributes into a unified display. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (2016), 121–130.
[13] Christopher Collins, Gerald Penn, and Sheelagh Carpendale. 2009. Bubble sets: Revealing set relations with isocontours over existing visualizations. *IEEE Transactions on Visualization and Computer Graphics* 15, 6 (2009), 1009–1016.
[14] Thomas M. J. Fruchterman and Edward M. Reingold. 1991. Graph drawing by force-directed placement. *Software: Practice and Experience* 21, 11 (Nov. 1991), 1129–1164. DOI: https://doi.org/10.1002/spe.4380211102
[15] Yue Gao, Meng Wang, Huanbo Luan, Jialie Shen, et al. 2011. Tag-based social image search with visual-text joint hypergraph learning. In *Proceedings of the 19th ACM International Conference on Multimedia*. ACM, 1517–1520.
[16] Lise Getoor and Christopher P. Diehl. 2005. Link mining: A survey. *ACM SIGKDD Explorations Newsletter* 7, 2 (2005), 3–12.
[17] Norbert I. Goldfield, Elizabeth C. McCullough, John S. Hughes, Ana M. Tang, Beth Eastman, Lisa K. Rawlins, and Richard F. Averill. 2008. Identifying potentially preventable readmissions. *Health Care Financing Review* 30, 1 (2008), 75.
[18] Jeffrey Heer and Danah Boyd. 2005. Vizster: Visualizing online social networks. In *Proceedings of the IEEE Symposium on Information Visualization (INFOVIS'05)*. IEEE, 32–39.
[19] Sheng Huang, Ahmed Elgammal, and Dan Yang. 2017. On the effect of hyperedge weights on hypergraph learning. *Image and Vision Computing* 57 (2017), 89–101.
[20] TaeHyun Hwang, Ze Tian, Rui Kuangy, and Jean-Pierre Kocher. 2008. Learning on weighted hypergraphs to integrate protein interactions and gene expressions for cancer outcome prediction. In *Proceedings of the IEEE International Conference on Data Mining (ICDM'08)*. IEEE, 293–302.
[21] Thorsten Joachims. 1999. Transductive inference for text classification using support vector machines. In *Proceedings of the International Conference on Machine Learning (ICML'99)*.
[22] Sanjay Kairam, Nathalie Henry Riche, Steven Drucker, et al. 2015. Refinery: Visual exploration of large, heterogeneous networks through associative browsing. In *Computer Graphics Forum*, Vol. 34. 301–310.
[23] Bohyoung Kim, Bongshin Lee, and Jinwook Seo. 2007. Visualizing set concordance with permutation matrices and fan diagrams. *Interacting with Computers* 19, 5-6 (2007), 630–643.
[24] Alexander Lex, Nils Gehlenborg, Hendrik Strobelt, Romain Vuillemot, and Hanspeter Pfister. 2014. UpSet: Visualization of intersecting sets. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 1983–1992.
[25] Lei Li and Tao Li. 2013. News recommendation via hypergraph learning: Encapsulation of user behavior and news content. In *Proceedings of the 6th ACM International Conference on Web Search and Data Mining*. ACM, 305–314.
[26] Qingshan Liu, Yuchi Huang, and Dimitris N. Metaxas. 2011. Hypergraph with sampling for image retrieval. *Pattern Recognition* 44, 10 (2011), 2255–2262.
[27] Mona Hosseinkhani Loorak, Charles Perin, Christopher Collins, and Sheelagh Carpendale. 2017. Exploring the possibilities of embedding heterogeneous data attributes in familiar visualizations. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2017), 581–590.

[28] Qing Lu and Lise Getoor. 2003. Link-based classification. In *Proceedings of the 20th International Conference on Machine Learning (ICML'03)*. 496–503.
[29] Yuxin Ma, Jiayi Xu, Xiangyang Wu, Fei Wang, and Wei Chen. 2017. A visual analytical approach for transfer learning in classification. *Information Sciences* 390 (2017), 54–69.
[30] Medicare. 2018. Data.Medicare.gov. Retrieved March 1, 2018 from https://data.medicare.gov/.
[31] Wouter Meulemans, Nathalie Henry Riche, Bettina Speckmann, Basak Alper, and Tim Dwyer. 2013. KelpFusion: A hybrid set visualization technique. *IEEE Transactions on Visualization and Computer Graphics* 19, 11 (2013), 1846–1858.
[32] George A. Miller. 1956. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review* 63, 2 (1956), 81.
[33] New York State Department of Health. 2018. Open health data of State of New York. Retrieved March 1, 2018 from https://health.data.ny.gov/.
[34] RealClearPolitics. 2018. Election 2016 State Polls. Retrieved March 1, 2018 from http://www.realclearpolitics.com/epolls/latest_polls/state/.
[35] Ramik Sadana, Timothy Major, Alistair Dove, and John Stasko. 2014. Onset: A visualization technique for large-scale binary set data. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 1993–2002.
[36] Sai Nageswar Satchidanand, Harini Ananthapadmanaban, and Balaraman Ravindran. 2015. Extended discriminative random walk: A hypergraph approach to multi-view multi-relational transductive learning. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*. 3791–3797.
[37] Gem Stapleton, Peter Rodgers, John Howse, and Leishi Zhang. 2011. Inductively generating euler diagrams. *IEEE Transactions on Visualization and Computer Graphics* 17, 1 (2011), 88–100.
[38] Liang Sun, Shuiwang Ji, and Jieping Ye. 2008. Hypergraph spectral learning for multi-label classification. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 668–676.
[39] The New York Times. 2018. Presidential Election Results. Retrieved from http://www.nytimes.com/elections.
[40] Corinna Vehlow, Fabian Beck, and Daniel Weiskopf. 2015. The state of the art in visualizing group structures in graphs. In *Proceedings of the Eurographics Conference on Visualization (EuroVis)-STARs*, Vol. 2. The Eurographics Association.
[41] Xumeng Wang, Jia-Kai Chou, Wei Chen, Huihua Guan, Wenlong Chen, Tianyi Lao, and Kwan-Liu Ma. 2018. A utility-aware visual approach for anonymizing multi-attribute tabular data. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (2018), 351–360.
[42] Wikipedia. 2018. Media bias in the United States. Retrieved March 1, 2018 from https://en.wikipedia.org/wiki/Media_bias_in_the_United_States#Coverage_of_electoral_politics.
[43] Jing Xia, Wei Chen, Yumeng Hou, Wanqi Hu, Xinxin Huang, and David S. Ebertk. 2016. Dimscanner: A relation-based visual exploration approach towards data dimension inspection. In *Proceedings of the 2016 IEEE Conference on VAST*. IEEE, 81–90.
[44] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. 2006. Learning with hypergraphs: Clustering, classification, and embedding. In *Proceedings of the the Neural Information Processing Systems Conference (NIPS'06)*, Vol. 19. 1633–1640.
[45] Xiaojin Zhu. 2006. Semi-supervised learning literature survey. *Computer Science, University of Wisconsin-Madison* 2, 3 (2006), 4.

Received August 2017; revised March 2018; accepted March 2018