Semi-supervised learning on closed set lattices

Mahito Sugiyama\textsuperscript{a,b,*} and Akihiro Yamamoto\textsuperscript{a}
\textsuperscript{a}Graduate School of Informatics, Kyoto University, Yoshida Honmachi, Sakyoku, Kyoto, Japan
\textsuperscript{b}Japan Society for the Promotion of Science, Japan

Abstract. We propose a new approach for semi-supervised learning using closed set lattices, which have been recently used for frequent pattern mining within the framework of the data analysis technique of Formal Concept Analysis (FCA). We present a learning algorithm, called SELF (SEmi-supervised Learing via FCA), which performs as a multiclass classifier and a label ranker for mixed-type data containing both discrete and continuous variables, while only few learning algorithms such as the decision tree-based classifier can directly handle mixed-type data. From both labeled and unlabeled data, SELF constructs a closed set lattice, which is a partially ordered set of data clusters with respect to subset inclusion, via FCA together with discretizing continuous variables, followed by learning classification rules through finding maximal clusters on the lattice. Moreover, it can weight each classification rule using the lattice, which gives a partial order of preference over class labels. We illustrate experimentally the competitive performance of SELF in classification and ranking compared to other learning algorithms using UCI datasets.

Keywords: Semi-supervised learning, label ranking, mixed-type data, closed set lattice, formal concept analysis

1. Introduction

In various research domains from biology to economics, numerous mixed-type data including both discrete (binary or nominal) and continuous (real-valued) variables are collected by researchers. However, despite recent rapid development of many data analysis techniques in the fields of machine learning, data mining, and knowledge discovery, only few algorithms such as the decision tree-based classifier \cite{29} can directly handle such mixed-type data. In particular, to the best of our knowledge, no learning algorithm treats mixed-type data in a semi-supervised manner.

Semi-supervised learning is a special form of classification \cite{48,50}; a learning algorithm uses both labeled and unlabeled data to learn classification rules. In real tasks, it is often difficult to obtain enough labeled data since the task of labeling has a high cost in terms of time and money, while lots of unlabeled data can be collected easily. The goal of semi-supervised learning is to construct a better classifier using such large amount of unlabeled data together with labeled data in short supply.

\*Corresponding author: Mahito Sugiyama, Graduate School of Informatics, Kyoto University, Yoshida Honmachi, Sakyoku, 606-8501 Kyoto, Japan. Tel.: +81 75 753 5628; Fax: +81 75 753 5628; E-mail: mahito@iip.ist.i.kyoto-u.ac.jp; Present address: Max Planck Institute for Intelligent Systems, AG K. Borgwardt, Spemannstr. 38, 72076, Tübingen, Germany. Tel.: +49 7071 601 1789; E-mail: mahito.sugiyama@tuebingen.mpg.de.

1088-467X/13/$27.50 © 2013 – IOS Press and the authors. All rights reserved
To effectively use unlabeled mixed-type data for learning, we in this paper propose a novel semi-supervised learning algorithm, called SELF (SEmi-supervised Learning via Formal Concept Analysis), which can directly treat mixed-type data. SELF adopts a popular semi-supervised learning strategy, called cluster-and-label [6,9], where a clustering algorithm is first applied, followed by labeling each cluster using labeled data. One of the remarkable features of SELF is that it performs the clustering process using Formal Concept Analysis (FCA) [8,14], which is a mathematical theory for data analysis and knowledge representation introduced by Wille [45]. Recently, Pasquier et al. [30] proposed to use closed patterns (itemsets) obtained by FCA as condensed “lossless” representations of patterns. This new approach has been the subject of further research and extensions [1,31,36,46]. In SELF, the labeling process is performed on a closed set lattice, which is the result of FCA. Informally, this structure describes the maximally general classification rules that explain the training data, thus preventing overfitting. Moreover, each classification rule can be weighted using the closed set lattice by counting the number of clusters classified by the rule, resulting in the preference of class labels as a partial order of them for each unlabeled datum. Furthermore, FCA and closed set lattices enable us to naturally treat incomplete data including missing values.

To summarize, this paper provides contributions to the respective fields of:

1. To semi-supervised learning: we present a novel approach based on an algebraic framework without assuming any data distribution.
2. To FCA: we study a novel application, semi-supervised learning, using FCA and closed set lattices.

The behavior of SELF is outlined as a flowchart in Fig. 1, and this paper is organized along it after discussing about related work in Section 2. The data preprocessing phase to construct a context from a given dataset to apply FCA is explained in Section 3.1. Missing values are handled in this phase. The learning phase is described in Sections 3.2 and 3.3; Section 3.2 shows data clustering and making closed set lattices by FCA and Section 3.3 explains the training algorithm of SELF to learn classification rules. Classification by learned rules is considered in Section 3.4. Section 4 gives empirical evaluation of SELF and, finally, key points and future work are summarized in Section 5.

2. Related work

Many studies have used FCA for machine learning and knowledge discovery [26], such as classification [12,13], clustering [47], frequent pattern and association rule mining [20,30,41], and bioinformatics [2,23,25]. In particular, Ganter and Kuznetsov [13] investigated the problem of binary classification of real-valued data and proposed algorithms based on the JSM-method that produces hypotheses (classifiers) using positive and negative examples. Their idea of using the lattice structure derived by FCA for classification is similar to our approach, but the way of treating continuous variables is different. Their method discretizes continuous variables by inequations, called conceptual scaling [14], that are given a priori, while SELF automatically discretizes them along with the learning process and no background knowledge and assumption about data are needed.

On the other hand, in machine learning context, decision tree-based algorithms such as C4.5 [32,33] can treat mixed-type data by discretizing continuous variables, and there are several discretization techniques [10,27,38] to treat continuous variables in a discrete manner. Our approach is different from them since we integrate discretization process into learning process and avoid overfitting using closed set lattices effectively. SELF uses cluster-and-label, or called label propagation, which is a popular approach in semi-supervised learning as mentioned in Introduction [4,6,9,19,44]. First SELF makes clusters without label information by FCA, followed by giving preferences of class labels for each cluster. To date,
Fig. 1. A flowchart of the proposed SELF algorithm. It learns classification rules from training data and applies them to classify test data. Here we say that a concept is consistent if all labels contained in the concept are same.

most of such approaches are designed for only continuous variables and, to the best of our knowledge, no semi-supervised learning algorithm based on cluster-and-label can treat mixed-type data including discrete variables appropriately. Since SELF uses FCA for clustering, it needs no distance calculation and no data distribution, which is one of the remarkable features of SELF.

There exists a study by Kok and Domingos [24] which is related to the idea of putting original data on lattices. They proposed a learning algorithm via hypergraph lifting, which constructs clusters by hypergraphs and learns on them. Their idea is thus similar to ours since we also “lift” raw data to the space of a closed set lattice via FCA. However, it is difficult to treat continuous variables in their approach, thereby our approach can be more useful for machine learning and knowledge discovery from mixed-type data.

SELF achieves not only semi-supervised learning but also label ranking using the preference for each class label. Recently, the concept of preference has attracted more and more attention in artificial intelligence including machine learning, resulting in formalization of the research topic of preference learning [49]. In particular, label ranking [5,18,43] has been treated in preference learning as an extension
of traditional supervised classification, where the objective is to obtain a ranker which gives a (partial) order of labels for each datum. SELF is the first algorithm that treats label ranking of mixed-type data by weighting each classification rule through closed set lattices.

3. SELF algorithm

We present the SELF algorithm in this section, which is the main part of this paper. The behavior of SELF is illustrated in Fig. 1; first it performs data preprocessing to make a context from a given dataset, second it constructs concept lattices by FCA, and third it learns the preference for each class label. Notations used in this paper are summarized in Table 1.

<table>
<thead>
<tr>
<th>Notation used in this paper</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{R}$</td>
</tr>
<tr>
<td>$\tau = (H, X)$</td>
</tr>
<tr>
<td>$\text{set}(X)$</td>
</tr>
<tr>
<td>$h$</td>
</tr>
<tr>
<td>$x, y$</td>
</tr>
<tr>
<td>$\perp$</td>
</tr>
<tr>
<td>$n$</td>
</tr>
<tr>
<td>$d$</td>
</tr>
<tr>
<td>$\text{Dom}(h)$</td>
</tr>
<tr>
<td>$G$</td>
</tr>
<tr>
<td>$M$</td>
</tr>
<tr>
<td>$I$</td>
</tr>
<tr>
<td>$(G, M, I)$</td>
</tr>
<tr>
<td>$g$</td>
</tr>
<tr>
<td>$m$</td>
</tr>
<tr>
<td>$h.m$</td>
</tr>
<tr>
<td>$\psi(G, M, I)$</td>
</tr>
<tr>
<td>$\lambda$</td>
</tr>
<tr>
<td>$\Lambda(g)$</td>
</tr>
<tr>
<td>$\Gamma(G)$</td>
</tr>
<tr>
<td>$R$</td>
</tr>
<tr>
<td>$\omega(R)$</td>
</tr>
<tr>
<td>$\psi(\lambda)$</td>
</tr>
<tr>
<td>$\prec, \prec^*$</td>
</tr>
<tr>
<td>$\prec$</td>
</tr>
<tr>
<td>$\text{CR}(\prec, \prec^*)$</td>
</tr>
<tr>
<td>$\text{CP}(\prec)$</td>
</tr>
</tbody>
</table>

3.1. Data preprocessing

The aim of data preprocessing is to construct a (formal) context, a binary matrix specifying a set of objects and their attributes, to apply FCA to training data.

A dataset $\tau$ is given in the form of a table, or a relation [7,15,37]. Each table is a pair $\tau = (H, X)$ of a header $H$ and a body $X$. We always denote the header size and the body size by $d$ and $n$, respectively. An element of the header $h \in H$ is called a feature\(^1\) and the domain of $h$ is denoted by $\text{Dom}(h)$. The

\(^1\)It is usually called an attribute, but to avoid confusion with an attribute in a context, we use the word “feature”.

\[\begin{align*}
\mathbb{R} & \text{ The set of real numbers} \\
\tau = (H, X) & \text{ Table, pair of header } H \text{ and body } X \\
\text{set}(X) & \text{ The set of tuples of body } X \\
h & \text{ Feature (element in } H) \\
x, y & \text{ Tuple} \\
\perp & \text{ Missing value} \\
n & \text{ Number of data (objects)} \\
d & \text{ Number of features} \\
\text{Dom}(h) & \text{ Domain of the feature } h \\
G & \text{ The set of objects} \\
M & \text{ The set of attributes} \\
I & \text{ Binary relation between } G \text{ and } M \\
(G, M, I) & \text{ Context} \\
g & \text{ Object, identified with tuple} \\
m & \text{ Attribute} \\
h.m & \text{ Qualified attribute generated from feature } h \\
\psi(G, M, I) & \text{ Concept lattice} \\
\lambda & \text{ Label} \\
\Lambda(g) & \text{ Label of object } g \\
\Gamma(G) & \text{ Set of labeled objects in } G \\
R & \text{ Classification rule (pair of set of attributes and label)} \\
\omega(R) & \text{ Weight of classification rule } R \\
\psi(\lambda) & \text{ Preference of label } \lambda \\
\prec, \prec^* & \text{ True partial order} \\
\prec & \text{ Predicted partial order} \\
\text{CR}(\prec, \prec^*) & \text{ Correctness of } \prec \\
\text{CP}(\prec) & \text{ Completeness of } \prec
\end{align*}\]
body $X$ is a sequence of tuples $x_1, x_2, \ldots, x_n$, where each tuple $x_i$ is a total function from $H$ to $\text{Dom}(H) = \{ \text{Dom}(h) \mid h \in H \}$ such that $x_i(h) \in \text{Dom}(h)$ for all $h \in H$. Informally, each tuple corresponds to a data point. Missing values in $X$ are allowed and denoted by the special symbol $\bot$, that is, if the value $x_i(h)$ is missing, $x_i(h) = \bot$. In addition, we denote the body $X$ by set($X$) when we treat it as a set, that is, set($X$) = $\{x_1, x_2, \ldots, x_n\}$. Thus we do not take the order and multiplicity into account in set($X$). For each tuple $x$ and a subset $J$ of the header $H$, the projection of $x$ on $J$, denoted by $x|_J$, is exactly the same as the restriction of $x$ to $J$, i.e., the function from $J$ to $\text{Dom}(H)$ such that $x|_J(h) = x(h)$ for all $h \in J$.

We consider two types of variables, *discrete* and *continuous*, in this paper. If a feature $h \in H$ is discrete, $\text{Dom}(h) = S \cup \{\bot\}$ for some countable set $S$. For instance, $S = \{T, F\}$ if the feature $h$ is binary and $S$ is a (finite) set of symbols if $j$ is nominal (categorical). If $h$ is continuous, $\text{Dom}(h) = \mathbb{R} \cup \{\bot\}$, where $\mathbb{R}$ is the set of real numbers.

In FCA, we call a triplet $(G, M, I)$ context. Here $G$ and $M$ are sets and $I \subseteq G \times M$ is a binary relation between $G$ and $M$. The elements in $G$ are called objects, and those in $M$ are called attributes. For a given table $\tau = (H, X)$, we identify the set of objects $G$ with set($X$) = $\{x_1, x_2, \ldots, x_n\}$.

In the data preprocessing, for each feature $h \in H$ of a table $\tau$, we independently construct a context $(G, M_h, I_h)$ and combine them into a context $(G, M, I)$. For this process, we always *qualify* attributes to be disjoint by denoting each element $m$ of the attribute $M_h$ by $h.m$ following the notations used in the database systems literature [15].

First, we focus on preprocessing for discrete variables. Since a context is also a discrete representation of a dataset, this process is directly achieved as follows: For each feature $h$, the set of attributes $M_h = \{h.m \mid m \in \text{Dom}(h) \setminus \{\bot\}\}$ and, for each value $x_i(h)$, $(x_i, h.m) \in I_h$ if and only if $x_i(h) = m$. In this way, discrete values are translated into a context and missing values are naturally treated. Algorithm 1 performs this translation.

### Algorithm 1: Data preprocessing for discrete variables

**Input:** Table $\tau = (H, X)$ whose variables are discrete  
**Output:** Context $(G, M_D, I_D)$  

function CONTEXTD$(\tau)$

1. $G \leftarrow \text{set}(X)$  
2. for each $h \in H$
3.  $M_h \leftarrow \{h.m \mid m \in \text{Dom}(h) \setminus \{\bot\}\}$
4.  $I_h \leftarrow \{(x, h.x(h)) \mid x \in G$ and $x(h) \neq \bot\}$
5. end for  
6. combine all $(G, M_h, I_h)$ with $h \in H$ into $(G, M_D, I_D)$  
7. return $(G, M_D, I_D)$

---

**Example 1** Given a table $\tau = (H, X)$ with $H = \{1, 2, 3\}$ and $X = x_1, x_2$ such that  

$(x_1(1), x_1(2), x_1(3)) = (T, \bot, C)$,  

$(x_2(1), x_2(2), x_2(3)) = (F, F, \bot)$.

This table can be represented in the following manner.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>T</td>
<td>$\bot$</td>
<td>C</td>
</tr>
<tr>
<td>$x_2$</td>
<td>F</td>
<td>F</td>
<td>$\bot$</td>
</tr>
</tbody>
</table>
The domains are given as Dom (1) = Dom (2) = \{T, F\} and Dom (3) = \{A, B, C\}. Here we have
\[
G = \{x_1, x_2\},
\]
\[
(M_1, I_1) = (\{1.T, 1.F\}, \{(x_1, 1.T), (x_2, 1.F)\}),
\]
\[
(M_2, I_2) = (\{2.T, 2.F\}, \{(x_2, 2.F)\}),
\]
\[
\]

Thus we have the context \((G, M, I)\) such that
\[
\]
\[
I = I_1 \cup I_2 \cup I_3 = \{(x_1, 1.T), (x_1, 3.C), (x_2, 1.F), (x_2, 2.F)\}.
\]

It is visualized as a cross-table as follows:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>\</td>
<td>x1</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\</td>
<td>x2</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Second, we make a context from continuous variables using discretization. This process is embedded in the learning process (see Fig. 1) and discretizing resolution increases along with the process. The degree of resolution is denoted by a natural number \(k\), called discretization level and, in the following, we explain how to discretize continuous variables at fixed level \(k\). First we use min-max normalization [17] so that every datum is in the closed interval \([0, 1]\). For every feature \(h\), each value \(x(h)\) is mapped to a value \(y(h)\) such that
\[
y(h) = \frac{x(h) - \min_{x \in \text{set}(x)} x(h)}{\max_{x \in \text{set}(x)} x(h) - \min_{x \in \text{set}(x)} x(h)}.
\]

Next, we discretize values in \([0, 1]\) and make a context using the binary encoding of real numbers, following the approach we have used [39]. At discretization level \(k\), \(M_h\) for a feature \(h \in H\) is always the set \(\{h.1, h.2, \ldots, h.2^k\}\). For each value \(x_i(h)\) of \(x_i(h) = 0\), then \((x_i, h.1) \in I_h\). Otherwise if \(x_i(h) \neq 0\), then \((x_i, h.a) \in I_h\) if and only if
\[
\frac{a - 1}{2^k} < x_i(h) \leq \frac{a}{2^k}.
\]

If \(x_i(h) = \bot\), then \((x_i, m) \notin I_h\) for all \(m \in M_h\). This means that if we encode the value \(x_i(h)\) as an infinite sequence \(p = p_0 p_1 p_2 \ldots\), a context at level \(k\) is decided by the first \(k\) bits \(p_0 p_1 \ldots p_{k-1}\). Each value is converted to exactly one relation of a context if it is not missing. Algorithm 2 shows the above process for making a context from continuous variables.

**Example 2** Given a table \(\tau = (H, X)\) with \(H = \{1, 2, 3, 4\}\) and \(X = x_1, x_2\) such that
\[
(x_1(1), x_1(2), x_1(3)) = (T, C, 0.35, 0.78),
\]
\[
(x_2(1), x_2(2), x_2(3)) = (\bot, \bot, 0.813, \bot).
\]
Thus we have the context and construct closed set lattices by FCA. First we summarize FCA (see the articles [8,14] for detail). We

3.2. Clustering and making lattices by FCA

It can be represented as follows:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>x1</td>
<td>T</td>
<td>C</td>
<td>0.35</td>
</tr>
<tr>
<td></td>
<td>x2</td>
<td>⊥</td>
<td>⊥</td>
<td>0.813</td>
</tr>
</tbody>
</table>

where the first and second features are discrete with Dom(1) = \{T,F\} and Dom(2) = \{A,B,C\}, and the third and forth are continuous. Assume that this discretization level \(k = 1\). We have

\[
G = \{x_1, x_2\},
\]

\[
(M_1, I_1) = (\{(1, T), (1, F)\}, \{(x_1, 1, T)\}).
\]

\[
(M_2, I_2) = (\{(2, A), (2, B), (2, C)\}, \{(x_1, 2, C)\}).
\]

\[
(M_3, I_3) = (\{3.1, 3.2\}, \{(x_1, 3.1), (x_2, 3.2)\}).
\]

\[
(M_4, I_4) = (\{4.1, 4.2\}, \{(x_1, 4.2)\}).
\]

Thus we have the context \((G, M, I)\) such that \(M = M_1 \cup M_2 \cup M_3 \cup M_4\) and \(I = I_1 \cup I_2 \cup I_3 \cup I_4\), which is visualized as a cross-table as follows:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x2</td>
<td></td>
<td></td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.2. Clustering and making lattices by FCA

From a context obtained by the data preprocessing, we generate closed sets as clusters of data points and construct closed set lattices by FCA. First we summarize FCA (see the articles [8,14] for detail). We always assume that a given table \(\tau\) is converted into a context \((G, M, I)\) by Algorithms 1 and 2.

For subsets \(A \subseteq G\) and \(B \subseteq M\), we define

\[
A' := \{m \in M \mid (g, m) \in I \text{ for all } g \in A\},
\]

\[
B' := \{g \in G \mid (g, m) \in I \text{ for all } m \in B\}.
\]
Fig. 2. The closed set lattice (concept lattice) constructed from the context given in Example 3.2. In this diagram, each dot
represents a concept, which are treated as a cluster in SELF.

Using these mappings, we define a concept as follows: a pair \((A, B)\) with \(A \subseteq G\) and \(B \subseteq M\) is called a concept of a context \((G, M, I)\) if \(A' = B\) and \(A = B'\). The set \(A\) is called an extent and \(B\) an intent. Each operator \(\prime\) is a Galois connection between the power set lattices on \(G\) and \(M\), respectively, hence the mapping \(\prime\) becomes a closure operator on the context \((G, M, I)\). This means that, for each concept \((A, B)\), \(A\) and \(B\) are (algebraic) closed sets. Note that a subset \(A \subseteq G\) (resp. \(B \subseteq M\)) is the extent (resp. intent) of some concept if and only if \(A'' = A\) (resp. \(B'' = B\)). Thus a set of objects \(A \subseteq G\) forms a cluster if and only if \(A'' = A\). Each object usually belongs to more than one cluster, hence this method is not “crisp” clustering.

The set of concepts over \((G, M, I)\) is written by \(\mathcal{B}(G, M, I)\) and called the concept lattice. If we focus on either one of the set of objects or attributes, this lattice is called the closed set lattice. In particular, in the context of frequent pattern mining, a set of attributes corresponds to an itemset and the lattice is called the closed itemset lattice. For a pair of concepts \((A_1, B_1) \in \mathcal{B}(G, M, I)\) and \((A_2, B_2) \in \mathcal{B}(G, M, I)\), we write \((A_1, B_1) \leq (A_2, B_2)\) if \(A_1 \subseteq A_2\). Then \((A_1, B_1) \leq (A_2, B_2)\) holds if and only if \(A_1 \subseteq A_2\) (and if and only if \(B_1 \supseteq B_2\)). This relation \(\leq\) becomes an order on \(\mathcal{B}(G, M, I)\) in the mathematical sense and \((\mathcal{B}(G, M, I), \leq)\) becomes a complete lattice. Let \(\mathcal{C} \subseteq \mathcal{B}(G, M, I)\). A concept \((A, B) \in \mathcal{C}\) is a maximal element of \(\mathcal{C}\) if \((A, B) \leq (X, Y)\) and \((X, Y) \in \mathcal{C}\) imply \((A, B) = (X, Y)\) for all \((X, Y) \in \mathcal{C}\). We write the set of maximal elements of \(\mathcal{C}\) by \(\text{Max}\mathcal{C}\).

Many algorithms are available for constructing closed set lattices, or concept lattices, and the algorithm proposed by Makino and Uno [28] is known to be one of the fastest algorithms. Their algorithm enumerates all maximal bipartite cliques in a bipartite graph with \(O(\Delta^3)\) delay, where \(\Delta\) is the maximum degree of the given bipartite graph, that is,

\[
\Delta = \max \left\{ \#J \mid J \subseteq I, \text{ where } g = h \text{ for all } (g, m), (h, l) \in J, \text{ or } m = l \text{ for all } (g, m), (h, l) \in J \right\}
\]
(#J is the number of elements in J) in the FCA context. A concept coincides with a bipartite graph, hence we can use their algorithm directly. For empirical experiments, we use the program LCM [40] provided by the authors to enumerate all concepts and construct the closed set lattice.

**Example 3** Given the following context:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>x₁</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x₂</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x₃</td>
<td></td>
<td>×</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x₄</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

There exist eight concepts in total; \(\emptyset, \{1, 2, 3, 4, 5\}\), \(\{x₁\}\), \(\{1, 2, 4\}\), \(\{x₂\}, \{2, 4, 5\}\), \(\{x₃\}, \{3\}\), \(\{x₁, x₂\}\), \(\{2, 4\}\), \(\{x₂, x₄\}, \{2, 5\}\), \(\{x₁, x₂, x₄\}, \{2\}\), and \(\{x₁, x₂, x₃, x₄\}, \emptyset\), and \(\Delta = 3\). We show the closed set lattice in Fig. 2. Let \(\mathcal{C} = \{\emptyset, \{1, 2, 3, 4, 5\}\), \(\{x₁\}, \{1, 2, 4\}\), \(\{x₂\}, \{2, 4, 5\}\), \(\{x₁, x₂\}, \{2, 4\}\), \(\{x₂, x₄\}, \{2, 5\}\}\}. Then \(\text{Max}_{\mathcal{C}} = \{\{x₁\}, \{1, 2, 4\}\}, \{\{x₁, x₂\}, \{2, 4\}\}, \{\{x₂, x₄\}, \{2, 5\}\}\}.

### 3.3. Learning classification rules

Here we present the main learning algorithm of SELF in Algorithm 3, which obtains a set of classification rules from a table \(\tau\) for training. In this paper, a classification rule is a pair of a set of attributes and a label. Intuitively, every unlabeled tuple (datum) is classified to the associated label if it has the same attributes. SELF generates a set of classification rules at each discretization level. We give the precise algorithm of classification in the next subsection.

<table>
<thead>
<tr>
<th>Algorithm 3: Main learning algorithm of SELF; learning classification rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> Table (\tau = (H, X))</td>
</tr>
<tr>
<td><strong>Output:</strong> Classification rules (R_1, R_2, \ldots, R_k)</td>
</tr>
<tr>
<td><strong>function</strong> <strong>MAIN</strong>(τ)</td>
</tr>
<tr>
<td>1: Divide (\tau) vertically into two tables (\tau_D) and (\tau_C), where (\tau_D) contains all discrete variables in (\tau) and (\tau_C) contains all continuous variables in (\tau)</td>
</tr>
<tr>
<td>2: ((G, M_D, I_D) \leftarrow \text{CONTEXT}(\tau_D))</td>
</tr>
<tr>
<td>// make a context from discrete variables of (\tau) (see Section 3.1)</td>
</tr>
<tr>
<td>3: (k \leftarrow 1) (// k) is discretization level</td>
</tr>
<tr>
<td>4: (\text{LEARNING}(\tau_C, G, M_D, I_D, k)) (// use this function recursively)</td>
</tr>
<tr>
<td><strong>function</strong> <strong>LEARNING</strong>(τₖ, G, M, Iₖ)</td>
</tr>
<tr>
<td>1: ((G, M_C, I_C) \leftarrow \text{CONTEXT}(\tau_C))</td>
</tr>
<tr>
<td>// make a context from continuous variables of (\tau) at level (k) (see Section 3.1)</td>
</tr>
<tr>
<td>3: construct the concept lattice (\mathcal{B}(G, M, I)) from ((G, M, I)) (see Section 3.2)</td>
</tr>
<tr>
<td>4: (\mathcal{C} \leftarrow {(A, B) \in \mathcal{B}(G, M, I)\mid \text{(A, B)} is consistent})</td>
</tr>
<tr>
<td>5: (R_k \leftarrow {(B, \Lambda(g))\mid (A, B) \in \text{(\text{Max}_{\mathcal{C}})}\text{(\and\text{(g \in \Gamma(A))}})</td>
</tr>
<tr>
<td>6: output (R_k)</td>
</tr>
<tr>
<td>7: (G \leftarrow {g \mid g \in A \text{ for some } (A, B) \in \mathcal{C}})</td>
</tr>
<tr>
<td>8: remove corresponding attributes and relations from (M_0) and (I_0), respectively</td>
</tr>
<tr>
<td>9: remove corresponding tuples from (\tau_C)</td>
</tr>
<tr>
<td>10: if (\Gamma(G) = \emptyset) then halt</td>
</tr>
<tr>
<td>11: else (\text{LEARNING}(\tau_C, G, M_D, I_D, k + 1))</td>
</tr>
<tr>
<td>12: end if</td>
</tr>
</tbody>
</table>

We introduce some notations. For each object \(g \in G\), we denote a label, an identifier of a class, of \(g\) by \(\Lambda(g)\), and if \(g\) is unlabeled; i.e., the label information is missing, we write \(\Lambda(g) = \perp\). Moreover, we
define \( \Gamma(G) := \{ g \in G \mid \Lambda(g) \neq \perp \} \), hence objects in \( \Gamma(G) \) are labeled objects, and those in \( G \setminus \Gamma(G) \) are unlabeled objects. For a concept \((A, B) \in \mathcal{B}(G, M, I)\), we say that it is consistent if \( \Gamma(A) \neq \emptyset \) and \( \Lambda(g) = \Lambda(h) \) for all \( g, h \in \Gamma(A) \). Note that a concept with \( \Gamma(A) = \emptyset \) (all labels are missing) is not consistent.

First SELF performs data preprocessing and makes the context \((G, M, I)\) from a given table at each discretization level \( k \) using the algorithms given in Section 3.1. Second it constructs the concept lattice \( \mathcal{B}(G, M, I) \) using both labeled and unlabeled tuples and finds consistent concepts using labeled tuples (objects). Third it outputs the sets of classification rules such that

\[
\mathcal{R}_k = \{ (B, \lambda) \mid (A, B) \in \text{Max}\mathcal{C}_k \text{ and } \lambda = \Lambda(g) \text{ with } g \in \Gamma(A) \}, \text{ where}
\]

\[
\mathcal{C}_k = \{ (A, B) \in \mathcal{B}(G, M, I) \mid (A, B) \text{ is consistent} \}
\]

at discretization level \( k \). The lattice enables us to avoid overfitting since, informally, attributes of maximal concepts correspond to the most general classification rules. If some objects that are not contained in consistent concepts remains, it refines discretization; i.e., increases discretization level, and repeats the above procedure for the remaining objects.

Moreover, SELF weights each classification rule. For a classification rule \( R = (B, \lambda) \), the weight \( \omega(R) \) is defined as follows:

\[
\omega(R) := \#\{ (C, D) \in \mathcal{B}(G, M, I) \mid D \supseteq B \}.
\]

Intuitively, the weight of a rule \( R \) means its importance since it is the number of clusters classified by the rule. Using the weight of rules, label ranking is realized (see the next subsection).

**Example 4** Given a dataset \( \tau = (H, X) \) and its labels as follows:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>T</td>
<td>C</td>
<td>0.28</td>
<td>1</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>F</td>
<td>A</td>
<td>0.54</td>
<td>1</td>
</tr>
<tr>
<td>( x_3 )</td>
<td>T</td>
<td>B</td>
<td>\perp</td>
<td>\perp</td>
</tr>
<tr>
<td>( x_4 )</td>
<td>F</td>
<td>A</td>
<td>0.79</td>
<td>2</td>
</tr>
<tr>
<td>( x_5 )</td>
<td>T</td>
<td>C</td>
<td>0.81</td>
<td>\perp</td>
</tr>
</tbody>
</table>

where \( \text{Dom}(1) = \{ T, F \} \cup \{ \perp \} \), \( \text{Dom}(2) = \{ A, B, C \} \cup \{ \perp \} \), and \( \text{Dom}(3) = \mathbb{R} \cup \{ \perp \} \). At discretization level 1, we have the following context:

<table>
<thead>
<tr>
<th>( 1.\mathbf{T} )</th>
<th>( 1.\mathbf{F} )</th>
<th>( 2.\mathbf{A} )</th>
<th>( 2.\mathbf{B} )</th>
<th>( 2.\mathbf{C} )</th>
<th>( 3.1 )</th>
<th>( 3.2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>( x_3 )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>( x_4 )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>( x_5 )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
</tbody>
</table>

We show the closed set lattice in the left-hand side in Fig. 3. By SELF, we obtain \( \mathcal{R}_1 = \{ \{ 1.\mathbf{T} \} \} \) since the concept \( \{ x_1, x_3, x_5 \}, \{ 1.\mathbf{T} \} \) is the maximal consistent concept, and there is no consistent concept that contains \( x_2 \) or \( x_4 \). This classification rule means “For a tuple \( x \), if \( x(1) = \mathbf{T} \), then \( x \) is classified to the class 1”. The weight is calculated as \( \omega(\{ 1.\mathbf{T} \}, 1) = 6. \) SELF removes objects \( x_1, x_3, \) and \( x_5 \) contained in the consistent concepts and proceeds to the next level. At discretization level 2, we have the following context:
Fig. 3. The closed set lattices (concept lattices) at discretization levels 1 and 2 constructed during the learning phase in Example 4. In these diagrams, each black dot denotes the maximal consistent concept in the set of concepts covered by the dotted line.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x_2</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x_4</td>
<td></td>
<td></td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The right-hand side in Fig. 3 shows the closed set lattice of the above context, and we obtain \( R_2 = \{ (\{1.F, 2.A, 3.3\}, 1), (\{2.A, 3.4\}, 2) \} \). For instance, the first rule means “For a tuple \( x \), if \( x(1) = F, x(2) = A \), and \( 0.5 < x(3) \leq 0.75 \), its class label is 1”. The weight are 2 for both rules.

We show that SELF always stops in finite time if there are no conflicting objects. Namely, for a table \( \tau = (H, X) \), if there is no pair \( x, y \in \text{set}(X) \) such that \( \Lambda(x) \neq \Lambda(y) \) and \( x(h) = y(h) \) for all \( h \in H \), Algorithm 3 stops in finite time. This statement is proved in the following way: if discretization level \( k \) is large enough, we have the concept lattice \( B(G_k, M_k, I_k) \), where for every object \( x \in G \), there exists a concept \( (A, B) \) such that \( A = \{ x \} \) since there is no pair \( x, y \in G \) satisfying \( x(h) = y(h) \) for all \( h \in H \). Thus each object \( x \) with \( \Lambda(x) \neq \bot \) must be contained in some consistent concept, and the algorithm stops. Note that the algorithm works even if \( \Gamma(G) = G \); i.e., all objects have labels, hence it also can be viewed as a supervised classification method.

The time complexity of learning by SELF is \( O(nd) + O(\Delta^3N) \) such that

\[
N = \max_{k \in \{1,2,\ldots,K\}} \#B(G_k, M_k, I_k),
\]

where \( (G_k, M_k, I_k) \) is the context at discretization level \( k \) and \( K \) is the level where SELF stops since data preprocessing takes \( O(nd) \), making a concept lattice takes less than \( O(\Delta^3N) \), and obtaining classification rules takes less than \( O(N) \).

### 3.4. Classification

Now we have sets of classification rules \( R_1, R_2, \ldots, R_K \) for each discretization level from training mixed-type data including labeled and unlabeled data using Algorithms 1, 2, and 3. In this section, we show how to classify a new unlabeled datum using the rules. We assume that such a new datum is given as a table \( \upsilon = (H, y) \), where the body \( y \) consists of only one tuple.

Algorithm 4 performs classification using the obtained rules \( R_1, R_2, \ldots, R_K \). The algorithm is levelwise; i.e., at each level \( k \), it makes a context \( (G, M, I) \) from the table \( \upsilon = (H, y) \) and apply the set of
Table 2

Statistics for UCI datasets used for experiments

<table>
<thead>
<tr>
<th>Name</th>
<th># Data</th>
<th># Classes</th>
<th># Features</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Discrete</td>
</tr>
<tr>
<td>Abalone</td>
<td>4177</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>Allbp</td>
<td>2800</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Anneal</td>
<td>798</td>
<td>5</td>
<td>28</td>
</tr>
<tr>
<td>Arrhythmia</td>
<td>452</td>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>Australian</td>
<td>690</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>Crx</td>
<td>690</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>Echocardiogram</td>
<td>131</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Heart</td>
<td>270</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>155</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>Horse colic</td>
<td>368</td>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>

Algorithm 4: Classification

Input: Classification rules \( \mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_K \), table \( \upsilon = (H, y) \), and the set of labels \( \mathcal{L} \)
Output: Preference of each label

\[ \text{CLASSIFY}(\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_K, \upsilon) \]

1: Divide \( \upsilon \) vertically into two tables \( \upsilon_D \) and \( \upsilon_C \), where \( \upsilon_D \) contains all discrete variables in \( \upsilon \) and \( \upsilon_C \) contains all continuous variables in \( \upsilon \)

2: \((G, M_D, I_D) \leftarrow \text{CONTEXT}_D(\upsilon_D)\)
   // make a context from discrete values of \( \upsilon \)

3: for each \( \lambda \in \mathcal{L} \)

4: \( \psi(\lambda) \leftarrow 0 \)

5: for each \( k \in \{1, 2, \ldots, K\} \)

6: \((G, M_C, I_C) \leftarrow \text{CONTEXT}_C(\upsilon_C, k)\)
   // make a context from continuous values of \( \upsilon \) at level \( k \)


8: \( \psi(\lambda) \leftarrow \psi(\lambda) + \sum_{R \in \mathcal{D}} \omega(R) \), where \( \mathcal{D} = \{(B, \lambda) \in \mathcal{R}_k \mid (y, b) \in I \text{ for all } b \in B \} \)

9: end for

10: output \( \psi(\lambda) \)

11: end for

rules \( \mathcal{R}_k \) to it. Let \( \mathcal{L}' \) be the domain of class labels. It checks all rules in \( \mathcal{R}_k \) and, for each label \( \lambda \in \mathcal{L} \), it outputs the preference of the label \( \lambda \), which is defined as

\[
\psi(\lambda) := \sum_{k=1}^{K} \sum_{R \in \mathcal{D}} \omega(R), \text{ where } \mathcal{D} = \{(B, \lambda) \in \mathcal{R}_k \mid (y, b) \in I \text{ for all } b \in B \},
\]

by summing up weights of rules. Note that the set \( G \) is always a singleton \{y\} in the classification phase. The result means that if \( \psi(\lambda) > \psi(\lambda') \) for labels \( \lambda \) and \( \lambda' \), \( \lambda \) is preferable than \( \lambda' \), and vice versa, and if \( \psi(\lambda) = \psi(\lambda') \), the preference of \( \lambda \) and \( \lambda' \) are same, resulting in the partial order over the set of labels \( \mathcal{L}' \). Thus the task of label ranking is achieved by the preference \( \psi \). Moreover, if we pick up the label \( \lambda \in \arg\max_{\lambda \in \mathcal{L}'} \psi(\lambda) \), (1) multiclass classification is also performed directly.

Example 5 Let us consider the case discussed in Example 4. A tuple \( y \) such that

\[(y(1), y(2), y(3)) = (T, B, 0.45)\]
We empirically evaluate SELF. Our experiments consist of two parts: one is about multiclass classification, and the other is about label ranking.
4.1. Methods

4.1.1. Environment

SELF was implemented in R version 2.12.1 [34] and all experiments were performed in the R environment. For enumeration of all concepts and construction of a closed set lattice from a context, we used LCM² distributed by Uno [40], which was implemented in C.

²http://research.nii.ac.jp/uno/codes.htm.

Fig. 5. Experimental results of accuracy for ten mixed-type datasets from UCI repository with varying the number of features. The number of labeled data was fixed at 10 in each experiment. We performed SELF using both labeled and unlabeled data (SELF) and using only labeled data (SELF (w/o)), and compared them to the decision tree-based classifier (Tree), SVM with the RBF kernel (SVM), and the k-nearest neighbor algorithm (1-NN, 5-NN). Data show mean ± s.e.m.
4.1.2. Datasets

We collected ten mixed-type datasets from UCI Machine Learning Repository [11]: abalone, allbp, anneal, arrhythmia, australian, crx, echocardiogram, heart, hepatitis, and horse colic. Their basic statistics are summarized in Table 2. Datasets allbp, anneal, arrhythmia, australian, crx, echocardiogram, hepatitis, and horse colic includes missing values, which were directly treated in SELF. In other learning algorithms, we ignored all tuples which have missing values since they cannot treat such datasets appropriately.

In label ranking, we used four datasets: abalone, allbp, anneal, and arrhythmia, which have more than three classes. The other datasets had only two classes and could not be used for label ranking evaluation.
414  M. Sugiyama and A. Yamamoto / Semi-supervised learning on closed set lattices

Fig. 7. Experimental results of correctness (should be maximized) for four mixed-type datasets from UCI repository with varying the number of labeled data. We performed SELF using both labeled and unlabeled data (SELF) and using only labeled data (SELF (w/o)). Data show mean ± s.e.m.

4.1.3. Control learning algorithms

In multiclass classification, three learning algorithms were adopted: the decision tree-based classifier implemented in R supplied in the tree package [35], SVM with the RBF kernel ($C = 5$ and $\gamma = 0.05$) in the kernlab package [21], and the $k$ nearest neighbor algorithm ($k = 1$ and 5) in the class package. Notice that only the decision tree-based algorithm can treat mixed-type data directly, which is one of typical such learning algorithms. All discrete values were treated as continuous in SVM and $k$NN.

4.1.4. Evaluation

In classification, for each dataset, the following procedure was repeated 20 times and the mean and s.e.m. (standard error of the mean) of accuracy was obtained: 1) the number of labeled data or features was fixed, where the range was from 10 to 100 and 2 to 10, respectively, 2) labeled training data were sampled randomly, 3) labels of the remaining data were predicted by respective learning algorithms, and 4) the accuracy was obtained.

The Eq. (1) was used to determine the most preferable label for each unlabeled datum. If there exists more than two such labels, we chose the smallest one.

We adopted two criteria: correctness and completeness, used in the literature [5] to evaluate partial orders of labels in label ranking. Correctness coincides with the gamma rank correlation [16], which is the normalized difference between the number of correctly ranked pairs and that of incorrectly ranked pairs. Let $\mathcal{L}$ be the set of class labels and we denote by $\prec_*$ the ground truth of the partial order over the set of labels $\mathcal{L}$. Assume that $\prec$ is a predicated partial order. Here we define

$$ C := \#\{(\lambda, \lambda') \in \mathcal{L} \times \mathcal{L} \mid \lambda \prec \lambda' \text{ and } \lambda \prec_*, \lambda'\}, $$
Fig. 8. Experimental results of completeness (should be maximized) for four mixed-type datasets from UCI repository with varying the number of labeled data. We performed SELF using both labeled and unlabeled data (SELF) and using only labeled data (SELF (w/o)). Data show mean ± s.e.m.

\[ D := \# \{ (\lambda, \lambda') \in \mathcal{L} \times \mathcal{L} \mid \lambda \prec \lambda' \text{ and } \lambda' \prec^* \lambda \} . \]

Then, the correctness is defined by

\[ \text{CR}(\prec, \prec^*) := \frac{C - D}{C + D} . \]

Trivially, the correctness takes a value in \([-1, 1]\), and \(\text{CR}(\prec, \prec^*) = 1\) if \(\prec = \prec^*\) and \(\text{CR}(\prec, \prec^*) = -1\) if \(\prec\) is the inversion of \(\prec^*\). Thus the correctness should be maximized. Moreover, to evaluate the degree of completeness of a predicted partial order, we use the completeness defined as follows:

\[ \text{CP}(\prec) := \frac{C + D}{\# \{ (\lambda, \lambda') \in \mathcal{L} \times \mathcal{L} \mid \lambda \prec \lambda' \text{ or } \lambda' \prec^* \lambda \} . \]

The completeness takes a value in \([0, 1]\) and should be maximized.

4.2. Results

4.2.1. Multiclass classification

We evaluated SELF in multiclass classification. Specifically, we examined SELF’s behavior with respect to the number of labeled data and the number of features; the number of labeled data was varied from 10 to 100, and the number of features from 2 to 10. When we fixed the number of labeled data,
we used all features for abalone, anneal, australian, crx, echocardiogram, heart, and hepatitis, and only used features 1, 2, 3, 18, 20 in allbp, 1, 2, ..., 6, 22, 22, ..., 25 in arrhythmia, and 1, 2, 4, 5, ... 11 in horse colic, since we could not finish experiments in reasonable time for such dense datasets. The above features seem to be representative for each dataset. Otherwise if we fixed the number of features, we examined two cases in which the number of labeled data for training were 10 or 100. Such small amount of labeled data is typical in semi-supervised learning; for example, the numbers 10 and 100 were adopted in benchmarks in the literature\(^3\) [50, §21].

To analyze effectivity of unlabeled data in the semi-supervised manner, we trained SELF in two ways; one is using both labeled and the remaining all unlabeled data for training, and the other is using only labeled data for training without any unlabeled data. In the following, we denote “SELF” in the former case and “SELF (w/o)” in the latter case. All experiments were carried out in the transductive setting [42], that is, test data coincide with the unlabeled training data. This setting is common in empirical evaluation of semi-supervised learning methods [50, §21].

For control, three learning algorithms were adopted: the decision tree-based classifier, SVM with the RBF kernel, and the \(k\) nearest neighbor algorithm (\(k = 1\) and 5). All the above algorithms are typical for supervised learning and hence did not use unlabeled data in training.

Figures 4 and 5, 6 show the accuracy with respect to changes in the number of labeled data and the number of features, respectively. In every case, the accuracy of SELF was much better than that of

\(^3\)This content is available at http://olivier.chapelle.cc/ssl-book/benchmarks.pdf.
Fig. 10. Experimental results of completeness (should be maximized) for mixed-type datasets from UCI repository with varying the number of features. The number of labeled data was fixed at 10 in each experiment. We performed SELF using both labeled and unlabeled data (SELF) and using only labeled data (SELF (w/o)). Data show mean ± s.e.m.

SELF (w/o), and the accuracy was getting better according as the number of labeled data increases. Moreover, SELF’s performance is getting better with increase in the number of features. SELF therefore can effectively use unlabeled data and features for learning.

In comparison with the tree algorithm which can treat mixed-type data directly, SELF showed better performance in all datasets in Fig. 4. Moreover, compared to other learning algorithms of SVM and kNN, SELF also achieved the best performance in abalone, anneal, and horse colic. When the number of labeled data is small (about 10–40), SELF outperformed other learning algorithms in all datasets except allbp, as shown in Figs 4 and 5.

4.2.2. Label ranking
We examined effectivity of SELF for label ranking. In consideration of the lack of benchmark data for label ranking, we adopted the following procedure for label ranking: We trained SELF using all labeled data on the respective dataset and obtained the ranking for each datum, and used them as the ground truth. Literature [5,18] which studied label ranking used the naïve Bayes classifier to make the ground truth of rankings from datasets. However, the mathematical theory is totally different from those of SELF, hence their approach is not appropriate to our case.

Figures 7 and 8 show the results of label ranking by SELF with varying the number of labeled data, and Figs 9–12 show those with respect to the number of features, where the number of labeled data is 10 for Figs 9 and 10, and 100 for Figs 11 and 12. The correctness of SELF is better than SELF (w/o) in abalone, and is similar between them in the other datasets for all conditions. In contrast, the
Fig. 11. Experimental results of correctness (should be maximized) for mixed-type datasets from UCI repository with varying the number of features. The number of labeled data was fixed at 100 in each experiment. We performed SELF using both labeled and unlabeled data (SELF) and using only labeled data (SELF (w/o)). Data show mean ± s.e.m.

completeness of SELF is much higher than that of SELF (w/o) in most cases. The main reason might be that lots of data are not classified to any class in SELF (w/o).

4.3. Discussion

Our experiments about classification (Figs 4, 5, 6) show that SELF has competitive performance compared to other machine learning algorithms, where unlabeled data can be used effectively in training. This result means that data clustering using the closed set lattices works well for semi-supervised learning of mixed-type data. Moreover, SELF can explicitly produce classification rules like the decision tree-based algorithm, hence SELF’s results can be easily interpreted. Furthermore, in label ranking (Figs 7–12), SELF outperformed SELF (w/o) in most cases in terms of completeness, and the performance got higher with increase of the number of labeled data. Our results therefore show that unlabeled data are also effectively used in SELF in the task of label ranking.

5. Conclusion

We have proposed a novel semi-supervised learning method, called SELF, for mixed-type data including both discrete and continuous variables, and experimentally showed its competitive performance. The key strategy is data clustering with closed set lattices using FCA, and the present study shows the effectivity of the lattices in semi-supervised learning. To our best knowledge, this approach is the first
Fig. 12. Experimental results of completeness (should be maximized) for mixed-type datasets from UCI repository with varying the number of features. The number of labeled data was fixed at 100 in each experiment. We performed SELF using both labeled and unlabeled data (SELF) and using only labeled data (SELF (w/o)). Data show mean ± s.e.m.

direct semi-supervised method for mixed-type data, and also the first one to exploit closed set lattices in semi-supervised learning. Moreover, we can directly treat missing values on SELF, meaning that SELF can be used for various practical datasets. To date, many semi-supervised learning methods use data distribution and probabilities, whereas SELF uses only the algebraic structure of data without any background knowledge. Our results with lattice-based data analysis provide new insight to machine learning and knowledge discovery.

There are two future works; one is analysis of SELF from FCA point of view. Refinement of discretization of continuous variables must have some connection with reduction of a context [14] since if we extend a context by refining real-valued variables, the original attributes are removed by reduction. Thereby analysis of mathematical connection between them is a future work. The other is theoretical analysis in the computational learning theory context. de Brecht and Yamamoto [3] have proposed Alexandrov concept space for learning from positive data. Our proposed method might be an instance of the study, since the concept lattice is similar to the Alexandrov space. Thus theoretical analysis of our framework is also a future work.

Acknowledgments

We would like to thank Marco Cuturi for his helpful comments. This work was partly supported by Grant-in-Aid for Scientific Research (A) 22240010 and for JSPS Fellows 22 - 5714.
References


