Artificial Intelligence in Medicine 49 (2010) 79–91

Contents lists available at ScienceDirect

Artificial Intelligence in Medicine

journal homepage: www.elsevier.com/locate/aiim

Classification integration and reclassification using constraint databases

Peter Revesz [a,*], Thomas Triplet [b]

[a] Department of Computer Science and Engineering, University of Nebraska-Lincoln, 358 Avery Hall, Lincoln, NE 68588, USA
[b] Department of Computer Science, Concordia University, 1455 de Maisonneuve Bldv West, Montreal, Quebec, Canada, H3G 1M8

ARTICLE INFO

Article history:
Received 28 January 2010
Received in revised form 1 February 2010
Accepted 16 February 2010

Keywords:
Classification integration
Constraint database
Data integration
Decision tree
Reclassification
Support vector machine
Cardiology
Primary biliary cirrhosis

ABSTRACT

Objective: We propose *classification integration* as a new method for data integration from different sources. We also propose *reclassification* as a new method of combining existing medical classifications for different classes.
Background: In many problems the raw data are already classified according to a set of features but need to be reclassified. Data reclassification is usually achieved using *data integration* methods that require the raw data, which may not be available or sharable because of privacy and legal concerns.
Methodology: We introduce general *classification integration* and *reclassification* methods that create new classes by combining in a flexible way the existing classes without requiring access to the raw data. The flexibility is achieved by representing any linear classification in a constraint database.
Results: The experiments using support vector machines and decision trees on *heart disease diagnosis* and *primary biliary cirrhosis* data show that our *classification integration* method is more accurate than current *data integration* methods when there are many missing values in the data. The reclassification problem also can be solved using constraint databases without requiring access to the raw data.
Conclusions: The *classification integration* and the *reclassification* methods are applied to two particular data sets. Beside these particular cases, our general method is also appropriate for many other application areas and may yield similar accuracy improvements. These methods may be also extended to non-linear classifiers.

1. Introduction

Classifications are usually done by classifiers such as support vector machines [1], decision trees [2], or other machine learning algorithms. After being trained on some sample data, these classifiers can be used to classify new data. However, many challenging applications require the reuse of the old classifiers to derive a new classifier. This later problem emerges in various settings, such as, the following two cases.

(1) Classification integration: Consider three hospitals that keep separate records of their cardiology patients (see Fig. 1). Suppose that after each hospital analyzes its own patients, they find three different classifications of cardiology patients *for the same characteristics*, in this case, heart disease risk. Intuitively, the three hospitals could get a potentially better classification of cardiology patients if they could combine their databases. Combining their databases, called *data integration* [3,4], could be followed by running a new classification algorithm on the integrated data.

However, data integration often fails for several reasons. First, the hospitals may have tested different variables on their own patients. That would yield many missing values in the integrated data. Most classification algorithms perform poorly when there are many missing values in the data. Second, the hospitals may be restricted in sharing their data due to privacy and legal concerns.

In this paper, we propose *classification integration* as a new approach to address the above problems. Our key insight is to *let the hospitals share only their classifiers.* Then instead of the data the classifications can be integrated, which yields a new classifier that may be better than the individual classifiers.

(2) Reclassification: The problem may be further complicated if two classifiers classify *different characteristics*. For example, suppose that one classifier tests whether patients have disease A and another classifier tests whether they have disease B. If one wants to combine these two classifiers, then one would need a classifier for patients who (1) have both diseases, (2) have only disease A, (3) have only disease B, and (4) have neither disease. In general, when combining n classifiers, there are 2^n combinations to consider. Hence, many applications would be simplified if a single combined classifier could be found.

For example, Figs. 2 and 3 present two different ID3 decision tree classifiers for the status of patients and the efficiency of a drug. For simplicity, the status of patients is classified as *alive*, *dead*, or

* Corresponding author. Tel.: +1 402 472 3488; fax: +1 402 472 7767.
E-mail addresses: revesz@cse.unl.edu (P. Revesz), thomastriplet@gmail.com (T. Triplet).

0933-3657/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.artmed.2010.02.003

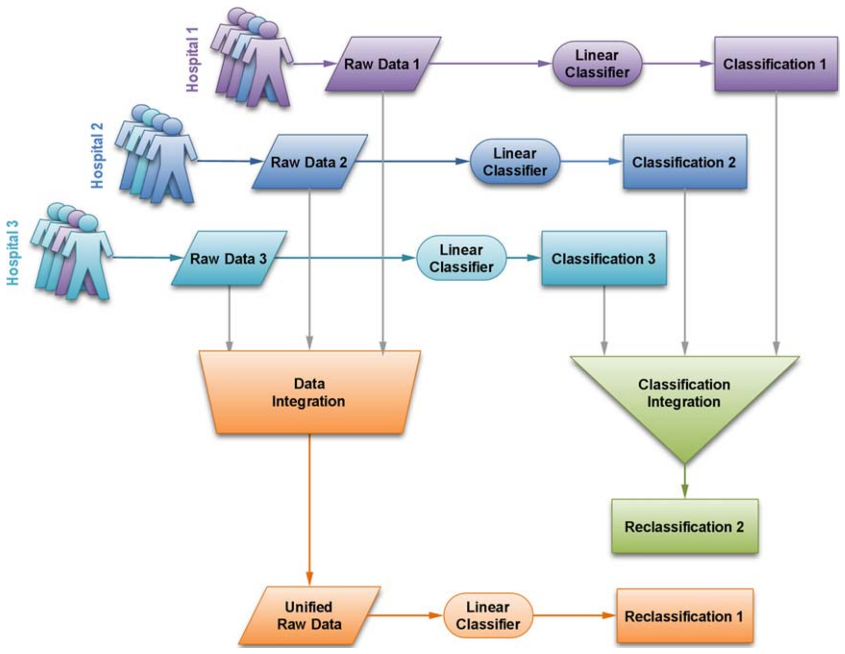


Figure 1. Comparison of data integration and classification integration methods.

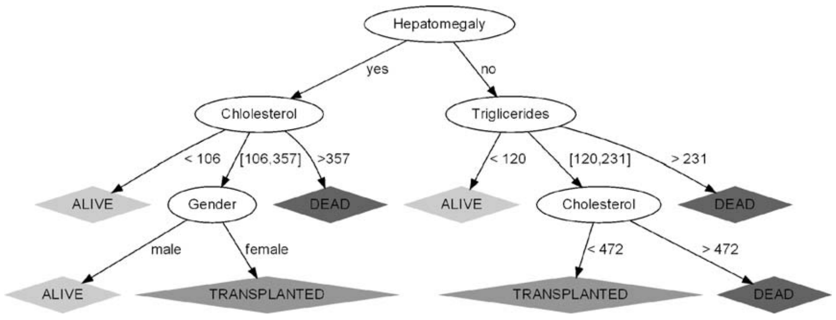


Figure 2. Decision tree for the prediction of the status of a patient using *primary biliary cirrhosis* data.

transplanted, and the drug efficiency is classified as *penicillamine* and *placebo*. Analysis of such decision trees is difficult. For instance, just by looking at these decision trees, it is hard to tell whether any patient died after taking a placebo, or whether any patient who used the drug is still alive. The problem with the above simple-looking queries is that no decision tree contains both *patient status* and *drug efficiency*. Finding a single decision tree that contains both *patient status* and *drug efficiency* would provide a convenient solution to the above queries. In Section 4, we propose a novel constraint database-based [5,6] reclassification method that results in a single classifier and enables to answer many challenging queries on medical data.

The rest of the paper is organized as follows. Section 2 presents a review of basic concepts and related work. Section 3 describes a novel *classification integration* method that relies on constraint databases. Section 4 presents a *reclassification with constraint databases* method, which extends with SVMs our preliminary work on reclassification [7]. Section 5 describes some computer experiments and discusses their significance. Finally, Section 6 gives some concluding remarks and open problems.

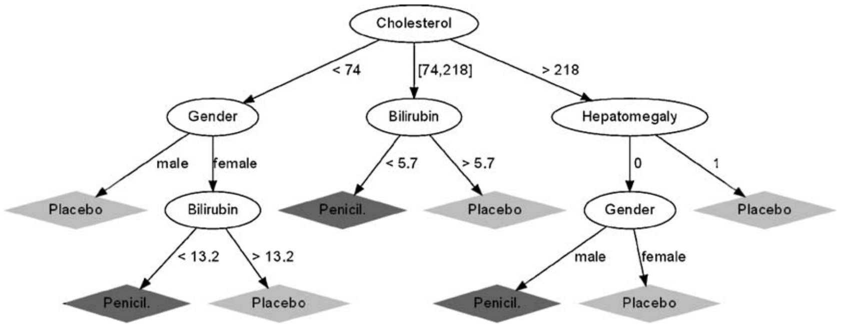


Figure 3. Decision tree for the prediction of the drug efficiency using *primary biliary cirrhosis* data.

2. Basic concepts and related work

Section 2.1 reviews classifiers, including support vector machines and decision trees. Section 2.2 reviews constraint databases and their prior applications for classifications.

2.1. Classifiers

In many problems, we need to classify items, that is, we need to predict some characteristics of an item based on several parameters of the item. Each parameter is represented by a variable which can take a numerical value. Each variable is called a *feature*, and the set of variables is called a *feature space*. The number of features is the *dimension* of the feature space. The characteristics of the item we want to predict is called the *label* or *class* of the item.

To make the predictions, we use *classifiers*. Each classifier maps a feature space X to a set of labels Y. The classifiers are found by various methods using a set of *training examples*, which are items where both the set of features and the set of labels are known.

A *linear classifier* maps a feature space X to a set of labels Y by a linear function. In general, a linear classifier $f(\vec{x})$ can be expressed as follows:

$$f(\vec{x}) = \langle \vec{w} \cdot \vec{x} \rangle + b = \sum_i w_i x_i + b \tag{1}$$

where $w_i \in \mathbb{R}$ are the *weights* of the classifiers and $b \in \mathbb{R}$ is a constant. The value of $f(\vec{x})$ for any item $\vec{x}$ directly determines the predicted label, usually by a simple rule. For example, in binary classifications if $f(\vec{x}) \geq 0$, then the label is $+1$, else the label is -1.

Example 2.1. Suppose that a disease is conditioned by antibodies A and B, the feature space is $X = \{Antibody\_A, Antibody\_B\}$ and the set of labels is $Y = \{Disease, No\_Disease\}$, where *Disease* corresponds to $+1$ and *No_Disease* corresponds to -1. Then, a linear classifier is:

$$f(\{Antibody\_A, Antibody\_B\}) = w_1 Antibody\_A + w_2 Antibody\_B + b$$

where $w_1, w_2 \in \mathbb{R}$ are constant weights, and $b \in \mathbb{R}$ is a constant. We can use the value of $f(\{Antibody\_A, Antibody\_B\})$ as follows:

- If $f(\{Antibody\_A, Antibody\_B\}) \geq 0$ then the patient has *Disease*.
- If $f(\{Antibody\_A, Antibody\_B\}) < 0$ then the patient has *No_Disease*.

2.1.1. Support vector machines

Suppose that numerical values can be assigned to each of the n features in the feature space. Let $\vec{x}_i \in \mathbb{R}^n$ with $i \in [1 \dots l]$ be a set of l training examples. Each training example $\vec{x}_i$ can be represented as a point in the n-dimensional feature space.

Support vector machines (SVMs) are popular classification tools. SVMs classify the items by constructing a hyperplane of dimension $n-1$ that will split all items into two classes +1 and -1. As shown in Fig. 4, several separating hyperplanes may be suitable to split correctly a set of training examples. In that case, an SVM will construct the *maximum-margin hyperplane*, that is, the hyperplane which maximizes the distance to the closest training examples. The maximum-margin hyperplane is defined in Eq. (2):

$$f(\vec{x}) = 0 \tag{2}$$

To find the hyperplane with the largest margin, $\vec{w}$ and b can be scaled so that the closest training examples $\vec{x}_i$ to the hyperplane satisfy $|f(\vec{x}_i)| = 1$. In this case, the margin equals $2/||\vec{w}||$ as shown in Fig. 5. Maximizing the margin is then equivalent to solving the following optimization problem:

$$\min_{\vec{w} \in \mathbb{R}^n, b \in \mathbb{R}} \frac{1}{2} ||\vec{w}||^2 \tag{3}$$

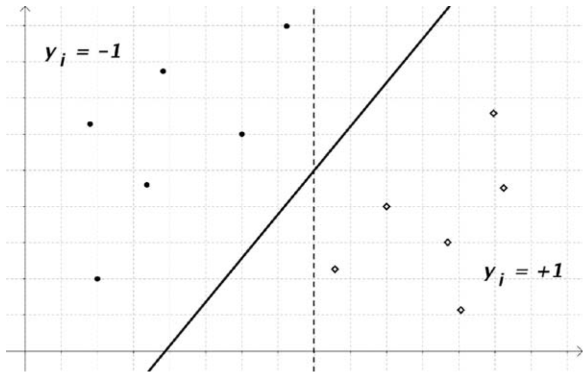


Figure 4. A set of training examples with labels $+1$ ($\diamond$) and -1 ($\bullet$). This set is linearly separable because a linear decision function in the form of a hyperplane can be found that classifies all examples without error. Two possible hyperplanes that both classify the training set without error are shown (solid and dashed lines). The solid line is expected to be a better classifier than the dashed line because it has a wider *margin*, which is the distance between the closest points and the hyperplane.

subject to:

$$|f(\vec{x}_i)| \geq 1$$

Eq. (3) leads to the following solution:

$$f(\vec{x}) = \sum_{i=1}^{l} \alpha_i y_i \langle \vec{x}_i \cdot \vec{x} \rangle + b \tag{4}$$

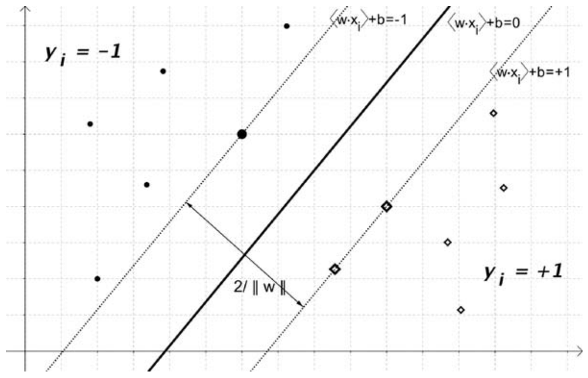
where α_i are positive real coefficients and $y_i \in \{+1, -1\}$ is the label of $\vec{x}_i$.

However, in practice, data are rarely linearly separable, and Eq. (4) may not be used. Vapnik [1] combined the technique described above with a mathematical method called the *kernel trick*. The kernel trick consists in a mapping function ϕ such that data, which are not linearly separable in the input feature space, may be linearly separable in a higher dimensional feature space. Using the kernel trick, Eq. (4) can be reformulated as:

$$f(\vec{x}) = \sum_{i=1}^{l} \alpha_i y_i \langle \phi(\vec{x}_i) \cdot \phi(\vec{x}) \rangle + b \tag{5}$$

We call the function $K(\vec{a}, \vec{b}) = \langle \phi(\vec{a}) \cdot \phi(\vec{b}) \rangle$ the *kernel* of the SVM. Various kernels are commonly used. In this paper, we are interested in reclassifying linear classifications. Hence, we choose a linear kernel defined as in Eq. (6):

$$K(\vec{a}, \vec{b}) = \langle \vec{a} \cdot \vec{b} \rangle = \sum_{j=1}^{n} a_j b_j \tag{6}$$



$\langle w \cdot x_i \rangle + b = -1$
$\langle w \cdot x_i \rangle + b = 0$
$\langle w \cdot x_i \rangle + b = +1$
$y_i = -1$
$y_i = +1$
$2/\|w\|$

Figure 5. Building the optimal – or maximum margin – hyperplane. The points nearest to the optimal separating hyperplane are called the "support vectors" (drawn larger than the other points). When w and b are properly scaled, the margin is $2/||w||$.

Let $\vec{x} = (x_1, x_2, \ldots, x_n)$. Combining Eqs. (5) and (6), $f(\vec{x})$ becomes:

$$f(\vec{x}) = \sum_{i=1}^{l}\left(\alpha_i y_i \sum_{j=1}^{n} x_{ij} x_j\right) + b$$
$$f(\vec{x}) = \sum_{j=1}^{n}\left(\sum_{i=1}^{l} \alpha_i y_i x_{ij}\right) x_j + b \tag{7}$$
$$f(\vec{x}) = \sum_{j=1}^{n} w_j x_j + b$$

with $w_j = \sum_{i=1}^{l} \alpha_i y_i x_{ij}$.

2.1.2. Decision trees: the ID3 algorithm

Decision trees were frequently used in the nineties by artificial intelligence experts because they can be easily implemented and they provide an *explanation* of the result. A decision tree is a tree with the following properties:

- Each internal node tests an attribute.
- Each branch corresponds to the value of the attribute.
- Each leaf assigns a classification.

The output of decision trees is a set of logical rules in the form of disjunctions of conjunctions. To train a decision tree, following the ID3 algorithm [2], we use the three steps below:

(1) The best attribute, A, is chosen for the next node. The best attribute maximizes the *information gain* which is defined as follows:

$$gain(S, A) = entropy(S) - \sum_{v \in (A)} \frac{|S_v|}{|S|} entropy(S_v) \tag{8}$$

where S is a sample of the training examples, S_v is the subset of S generated by v and A is a partition of the parameters. Like in thermodynamics, the entropy measures the *impurity* of S, purer subsets having a lower entropy:

$$entropy(S) = -\sum_{i=0}^{n} p_i log_2(p_i) \tag{9}$$

where S is a sample of the training examples, p_i is the proportion of *i-valued* examples in S, and n is the number of attributes.

(2) We create a descendant for each possible value of the attribute A.

(3) For each non-perfectly classified branch repeat steps (1) and (2).

ID3 is a greedy algorithm without backtracking. This means that the algorithm is sensitive to local optima. Furthermore, ID3 is inductively biased: the algorithm favors short trees and high information gain attributes near the root. At the end of the procedure, the decision tree perfectly suits the training data including noisy data. This leads to complex trees, which usually lead to problems in classifying new data. According to Occam's razor, shortest explanations should be preferred. To avoid overfitting, decision trees are usually pruned after the training stage by minimizing $size(tree) + error\_rate$, where $size(tree)$ is the number of leaves in the tree, and $error\_rate$ is the ratio of the number of misclassified instances and the total number of instances. Finally, *accuracy* is defined as $accuracy = 1 - error\_rate$.

It is interesting to compare decision trees with SVMs. The ID3 decision trees and SVMs are linear classifiers because their effects can be represented mathematically in the form of Eq. (1). Unlike decision trees, SVMs are not subject to local optima issues because SVMs are guaranted to find the global optimum. Hence SVMs are expected to generalize better than other machine learning algorithms such as decision trees. On the other hand, decision trees have the advantage of providing an explanation of their classifications, while SVMs do not provide any explanation.

Table 1
Constraint database representation of a linear classifier.

Antibody_A	Antibody_B	Label	
x_1	x_2	y	$w_1x_1 + w_2x_2 + b \geq 0$, y = 'Disease'
x_1	x_2	y	$w_1x_1 + w_2x_2 + b < 0$, y = 'No_Disease'

2.2. Constraint databases

Constraint databases [5,6] form an extension of relational databases [8] where the database can contain variables that are usually constrained by linear or polynomial equations.

Constraint databases can be queried similar to relational databases by both Datalog and SQL queries [9–11]. Constraint database systems include CCUBE [12], DEDALE [13], IRIS [14], and MLPQ [15].

Example 2.2. We can represent the linear classifier of Example 2.1 in a constraint database as in Table 1. Suppose a patient has 44 units of *Antibody_A* and 24 units of *Antibody_B*. Then the following SQL query in the MLPQ system can find whether the patient is sick.

```
SELECT Label
FROM Classifier
WHERE Antibody_A = 44 AND Antibody_B = 24
```

As an alternative, we can represent the linear classifier as in Table 2. Using this alternative representation, the above SQL query becomes:

```
SELECT ``Disease''
FROM Classifier
WHERE Antibody_A = 44 AND Antibody_B = 24 AND F ¿= 0
```

This query returns "Disease" if the patient is sick.

Constraint databases, which were initiated by the original article of Kanellakis et al. [16], have many applications ranging from spatial databases [17,18] through moving objects [19,20] to epidemiology [21]. The original constraint database model was extended recently to labelled object-relational constraint databases (LORCDB) by Gómez-López et al. [22] and applied to model-based diagnosis.

Geist [23] and Lakshmanan et al. [24] applied constraint databases to data mining and classsification problems and discussed the constraint representation of decision trees. In Section 4.3, we give a variant of this idea but using instead the ID3 algorithm to generate decision trees. Neither Geist [23] nor Lakshmanan et al. [24] considered the representation of support vector machines (SVMs) by constraint databases. They also did not consider applying constraint databases to the classification integration and the reclassification problems, which are the main issues addressed in this paper.

3. The classification problem

We first review the classification problem by giving an example in Section 3.1. Section 3.2 reviews the traditional approach of data integration-based classification. Section 3.3 introduces the new approach of *classification integration*.

Table 2
Alternative constraint database representation of a linear classifier.

Antibody_A	Antibody_B	F	
x_1	x_2	y	$w_1x_1 + w_2x_2 + b = y$

Table 3
Two heart patient data sets represented using relational databases.

Cleveland patients				
P	C	G	B	D
1	233	1	145	No
3	250	1	130	Yes
3	275	0	110	No
4	230	1	117	Yes
2	198	0	105	No
4	266	1	124	Yes
Budapest patients				
P	C	G	B	D
1	237	0	170	No
2	219	0	100	No
4	270	1	120	Yes
2	198	0	105	No
4	246	0	100	Yes
1	156	1	140	Yes
2	257	1	110	Yes

Table 4
Union of the patient data from Cleveland and Budapest using relational databases.

P	C	G	B	D
1	233	1	145	No
3	250	1	130	Yes
⋮	⋮	⋮	⋮	⋮
1	156	1	140	Yes
2	257	1	110	Yes

3.1. The classification problem with multiple sources

Example 3.1. A common problem in classification arises when we have several different sources of data. For example, the Cleveland Clinic Foundation has a heart disease data set, which is described in detail in Section 5.1. For simplicity, in this example, we can consider the following smaller version with only six randomly selected patients, where the features are: chest pain P, cholesterol C, gender G, and resting blood pressure B, and the label is disease D, which is 'Yes' or 'No' according to whether a patient has a heart disease.

Table 5
Linear constraint representation of the SVM trained on the integrated data from Cleveland and Budapest.

P	C	G	B	D	
p	c	g	b	d	$0.8696\,p - 0.0024c + 1.7055g + 0.0052b - 3.5154 = d$

The Institute of Cardiology in Budapest, Hungary collected another data set for heart disease patients. From this data set, we may select randomly seven patients. The Cleveland and the Budapest relations are shown in Table 3.

The main problem is how to combine the two data sources and use them together. We describe two different solutions in the following two sections.

3.2. Data integration

To solve the classification problem raised in Example 3.1, we can use *data integration* [3,4]. In data integration, we first take the union of the two data sets, yielding an integrated data set with thirteen patients (Table 4).

Second, we use the integrated data set to build a linear classification using a SVM. The D value 'No' is converted to -1 and 'Yes' is converted to $+1$ before the SVM algorithm is applied. The SVM algorithm yields a linear classifier, which can be represented by a linear constraint relation (see Table 5).

3.3. Classification integration

Considering the *Cleveland Patients* table, the Cleveland Clinic Foundation may build the linear classifier using a SVM shown in Table 6 (top). Similarly, the Institute of Cardiology in Budapest, Hungary may build based on its data set the linear classification shown in Table 6 (bottom).

As an alternative to data integration, we propose the use of *classification integration* by taking the natural join of the two linear constraint relations and then selecting the value $d = 0.5d_1 + 0.5d_2$ (see Table 7). Since the integrated classification is based on two data sets, it can be expected to be better than either the Cleveland classification or the Budapest classification in itself.

Table 6
Constraint database for the SVMs trained using Cleveland and Budapest patients.

Cleveland classification					
P	C	G	B	D_1	
p	c	g	b	d_1	$1.1568\,p - 0.0172c + 0.4278g + 0.0333b - 3.404 = d_1$
Budapest classification					
P	C	G	B	D_2	
p	c	g	b	d_2	$1.0213\,p - 0.0016c + 1.9091g + 0.015b - 4.2018 = d_2$

Table 7
Classification integration of the classifiers from Cleveland and Budapest using a join operator. The constraint relation may be simplified using a projection operator on the variables d_1 and d_2.

Integrated classification					
P	C	G	B	D	
Integrated classification					
p	c	g	b	d	$d = 0.5d_1 + 0.5d_2$, $1.1568\,p - 0.0172c + 0.4278g + 0.0333b - 3.404 = d_1$, $1.0213\,p - 0.0016c + 1.9091g + 0.015b - 4.2018 = d_2$
Simplified integrated classification					
P	C	G	B	D	
p	c	g	b	d	$1.0891\,p - 0.0094c + 1.1685g + 0.0242b - 3.8029 = d$

4. The reclassification problem

We describe the reclassification problem by giving an example in Section 4.1. Then we introduce several new reclassification methods. Section 4.2 describes the *Reclassification with an oracle* method. While oracle-based methods do not exist in practice, those methods give a theoretical limit to the best possible practical methods. Section 4.3 describes the practical *Reclassification with constraint databases* method.

4.1. The reclassification problem

The need for reclassification arises in many situations. Consider the following.

Example 4.1. One study found a classifier for the status of patients with *primary biliary cirrhosis* [25] using:

$X_1 = \{Cholesterol(C), Gender(G), Hepatomegaly(H), Triglicerides(T)\}$

and the class for patient status:

$S = \{Ali, Dead, Transplanted\}$

where cholesterol is the serum cholesterol in mg/dl, hepatomegaly is ‘1’ if the liver is enlarged and ‘0’ otherwise, gender is ‘0’ for male and ‘1’ for female, and triglicerides level is measured in mg/dl. Fig. 2 shows an example decision tree for the status of patients obtained after training by 50 random examples. A sample training data is shown in Table 8 (left).

Another study found a classifier to distinguish between a real drug and a placebo using:

$X_2 = \{Bilirubin(B), Cholesterol(C), Gender(G), Hepatomegaly(H)\}$

and the class for drug:

$D = \{Penicillamine, Placebo\}$

where the serum bilirubin of the patient is measured in mg/dl.

Fig. 3 shows an example of decision tree for the efficiency of a drug obtained after training by 50 random examples. A sample training data for the second study is shown in Table 8 (right).

Building a new classifier for $(X = X_1 \cup X_2, Y = S \times D)$ seems easy, but the problem is that there is no database for (X, Y). Finding such a database would require a new study with more data collection, which would take a considerable time. That motivates the need for reclassification. As Section 4.2 shows, a classifier for (X, Y) can be built by an efficient reclassification algorithm that uses only the already existing classifiers for (X_1, S) and (X_2, D).

Suppose we need to find a classifier for

$$\begin{aligned} X &= X_1 \cup X_2 \\ &= \{Bilirubin, Cholesterol, Gender, Hepatomegaly, Triglicerides\} \end{aligned}$$

and

$$\begin{aligned} Y = S \times D = \{&Ali\_Penicillamine, Ali\_Placebo, \\ &Dead\_Penicillamine, Dead\_Placebo, \\ &Transplanted\_Penicillamine, Transplanted\_Placebo\} \end{aligned}$$

The next two sections present two different solutions to this problem.

4.2. Reclassification with an oracle

In theoretical computer science, researchers study the computational complexity of algorithms in the presence of an oracle that tells some extra information that can be used by the algorithm. The computational complexity results derived using oracles can be useful in establishing theoretical limits to the computational complexity of the studied algorithms.

Similarly, in this section we study the reclassification problem with a special type of oracle. The oracle we allow can tell the value of a missing attribute of each record. That allows us to derive essentially a theoretical upper bound on the best reclassification that can be achieved. The reclassification with oracle method extends each of the original relations with the attributes that occur only in the other relation. Then one can take a union of the extended relations and apply any of the chosen classification algorithms. We illustrate the idea behind the *Reclassification with an oracle* method using an extension of Example 4.1 and ID3.

Example 4.2. The reclassification with an oracle method adds the missing bilirubin measure and drug class to each record of the *Patient Status* relation (Table 9, top) and triglicerides measure and patient status class to each record of the *Drug Efficiency* relation (Table 9, bottom). After the union of these two relations, we can train an ID3 decision tree to yield a reclassification as needed to complete Example 4.1.

4.3. Reclassification with constraint databases

The *Reclassification with constraint databases* method has two main steps:

(1) **Translation to constraint relations**: We translate the original linear classifiers to a constraint database representation. Our method does not depend on any particular linear classification method.
(2) **Join**: The linear constraint relations are joined together using a constraint database join operator [5,6].

Example 4.3. We give first an example to illustrate the *Translation to Constraint Relations*. Assume the following variables in the feature space:
• *b*, the bilirubin serum,
• *c*, the cholesterol rate,
• *g*, the gender of the patients,

Table 8
Example of medical records for the status of patients and the efficiency of penicillamine.

Patient status				
C	G	H	T	S
261	1	1	172	Alive
200	0	0	143	Transplanted
Drug efficiency				
B	C	G	H	D
3.6	244	0	0	Placebo
7.4	54	0	0	Penicillamine

Table 9
The reclassification with an oracle method adds the missing values.

Patient status						
B	C	G	H	T	S	D
14.1	261	1	1	172	Alive	Placebo
5.3	200	0	0	143	Transplanted	Penicillamine
Drug efficiency						
B	C	G	H	T	S	D
3.6	244	0	0	114	Alive	Placebo
7.4	54	0	0	189	Transplanted	Penicillamine

Table 10
Translation of the decision trees shown in Figs. 2 (bottom) and 3 (top).

Drug					
B	C	G	H	D	
b	c	g	h	d	$c<74, g=1, d$ = 'Placebo'
b	c	g	h	d	$c<74, g=0, b<13.2, d$ = 'Penicillamine'
b	c	g	h	d	$c<74, g=0, b>13.2, d$ = 'Placebo'
b	c	g	h	d	$c>74, c<218, b<5.7, d$ = 'Penicillamine'
b	c	g	h	d	$c>74, c<218, b>5.7, d$ = 'Placebo'
b	c	g	h	d	$c>218, h=0, g=1, d$ = 'Penicillamine'
b	c	g	h	d	$c>218, h=0, g=0, d$ = 'Placebo'
b	c	g	h	d	$c>218, h=1, d$ = 'Placebo'

Status					
C	G	H	T	S	
c	g	h	t	s	$h=1, c<106, s$ ='Alive'
c	g	h	t	s	$h=1, c>106, c<357, g=1, s$ ='Alive'
c	g	h	t	s	$h=1, c>106, c<357, g=0, s$ ='Transplanted'
c	g	h	t	s	$h=1, c>357, s$ ='Dead'
c	g	h	t	s	$h=0, t<120, s$ ='Alive'
c	g	h	t	s	$h=0, t>120, t<231, c<472, s$ ='Transplanted'
c	g	h	t	s	$h=0, t>120, t<231, c>472, s$ ='Dead'
c	g	h	t	s	$h=0, t>231, s$ ='Alive'

- h, the hepatomegaly,
- t, the triglicerides rate.

Suppose we try to predict the drug efficiency d on each patient. Then we use the decision tree shown in Fig. 3 to classify the efficiency of the drug. The decision may be translated into linear constraint as shown in Table 10 (top). We also try to predict the status s. Similarly, the decision tree represented in Fig. 2 is used to generate the constraint relations in Table 10 (bottom).

The reclassification problem can be solved by a constraint database join of the *Drug* and *Status* relations, expressed in non-recursive Datalog as follows:

```
Reclass(b,c,g,h,t,d,s) :- Drug(b,c,g,h,d), Status(c,g,h,t,s).
```

The evaluation of the above query requires a constraint database join, which is explained in detail, for example in the textbook [6]. The pseudocode below presents an outline of the database join. In the pseudocode, we use $R[i]$ for the ith constraint tuple of relation R. The pseudocode uses two functions, namely *combine_and_simplify* and *satisfiable*, which takes the conjunction and simplifies the constraint tuples and tests the satisfiablity of a constraint tuple, respectively.

```
Join ($R1, R2$)
input: R1 and R2 two constraint relations.
output: Relation $S$ containing the join of $R1$ and $R2$.
S := empty
for i := 1 to size(R1) do
for j := 1 to size(R2) do
Temp := combine_and_simplify ($R1[i], R2[j]$)
if satisfiable(Temp) then
S := S ∪ Temp
end-if
end-for
end-for
return (S)
```

To find the value of the *Reclass* relation in our present case, the constraint join operator will try to combine every constraint tuple of *Drug* with every constraint tuple of *Status*. If the combination of the two tuples yields an unsatisfiable set of constraints, then it is discarded. Only those combinations are kept that are satisfiable for some input values for the features. As there are eight constraint tuples in both *Drug* and *Status*, there are 64 possible combinations to consider. Table 11 shows the first five satisfiable constraint tuples of *Reclass*.

Table 11
First five satisfiable constraint tuples for the reclassification of the *Drug* and *Status* relations.

Reclass							
B	C	G	H	T	D	S	
b	c	g	h	t	d	s	$c<74, g=1, d$ = 'Placebo', $h=1, c<106, s$ ='Alive'
b	c	g	h	t	d	s	$c<74, g=1, d$ = 'Placebo' $h=0, t<120, s$ ='Alive'
b	c	g	h	t	d	s	$c<74, g=1, d$ = 'Placebo' $h=0, t>120, t<231, c<472, s$ ='Transplanted'
b	c	g	h	t	d	s	$c<74, g=1, d$ = 'Placebo' $h=0, t>231, s$ ='Alive'
b	c	g	h	t	d	s	$c<74, g=0, b<13.2, d$ = 'Penicillamine' $h=1, c<106, s$ ='Alive'
							⋮

In the *Reclass* relation, the first tuple is a combination of the first tuple of *Drug* with the first tuple of *Status*. A constraint database would usually not only check satisfiability but also simplify the constraints. That would mean deleting unnecessary constraints. For example, $c<106$ is unnecessary because it is already implied by $c<74$, which is the first constraint in the first tuple of *Reclass*.

Next, the first tuple of *Drug* and the second tuple of *Status* are combined. However, that is unsatisfiable because one contains $c<74$ while the other contains $c>106$. Hence this combination is not part of *Reclass*. Similarly, we also discard the combination of the first tuple of *Drug* with the third and fourth tuples of *Status*. Therefore, the next combination that is satisfiable is the first tuple of *Drug* and the fifth tuple of *Status*. This satisfiable combination is recorded as the second constraint tuple of *Reclass*. The other tuples are determined similarly. With n constraint tuples in both input relations, the overall time complexity of the constraint join operator is $O(n^2)$ [6], that is, it is as efficient as a relational database join operator.

Let *Patients(id,b,c,g,h,t)* be a relation that records the identification number and the features of each patient. Then the following non-recursive Datalog query can be used to predict the drug efficiency and the status for each patient:

```
Predict(d,s)   :-   Patients(id,b,c,g,h,t),
Reclass(b,c,g,h,t,d,s).
```

Instead of Datalog queries, one can use the logically equivalent SQL query:

```
CREATE VIEW Predict AS
SELECT R.d, R.s
FROM Patients as P, Reclass as R
WHERE  P.b = R.bANDP.c = R.cANDP.g = R.gAND
       P.h = R.hANDP.t = R.t
```

The SQL query is evaluated similarly to the Datalog query. It is largely a stylistic preference whether one uses SQL or Datalog queries.

5. Experimental results and discussion

Section 5.1 compares the *data integration* and the *classification integration* methods of Section 3, while Section 5.2 compares the *reclassification with an oracle* and the *reclassification with constraint databases* methods of Section 4.

We tested our methods using both SVMs and decision trees. However, our goal is not to compare SVMs with decision trees but to show the versatility of our methods. Note that each method can use either SVMs or decision trees, but comparing a method that uses SVM with another method that uses decision trees cannot decide which method is better because the difference may be due only to the differences between SVMs and decision trees.

5.1. Data integration vs. classification integration

We compare the *data integration* and the *classification integration* methods assuming that both methods use either SVMs or decision trees as original linear classifiers. In our experiments, we used the SVM implementation from SVMLib [26] and a heart disease diagnosis data set [25], which includes data from the following three hospital sources:

- Cleveland Clinic Foundation, USA (303 patients),
- Institute of Cardiology, Budapest, Hungary (294 patients),
- University Hospital of Zürich, Switzerland (123 patients).

Each hospital records the following ten features:

(1) **age**: years
(2) **gender**: (0 = female, 1 = male)
(3) **cp**: chest pain (1 = typical angina, 2 = atypical, 3 = non-anginal, 4 = asymptomatic)
(4) **trestbps**: resting blood pressure (in mmHg on admission to the hospital)
(5) **chol**: serum cholestoral in mg/dl
(6) **restecg**: resting electrocardiographic results
(7) **thalach**: maximum heart rate achieved
(8) **exang**: exercise induced angina (0 = no, 1 = yes)
(9) **oldpeak**: ST depression induced by exercise relative to rest
(10) **disease**: presence of heart disease (-1 = not present, 1 = present)

We used the first nine features to predict the last feature, using the following procedure, where n is a parameter controlling the size of the training set:

(1) Randomly select 60 patients from the three hospitals as a testing set.
(2) Randomly select n percent of the remaining patients as a training set.

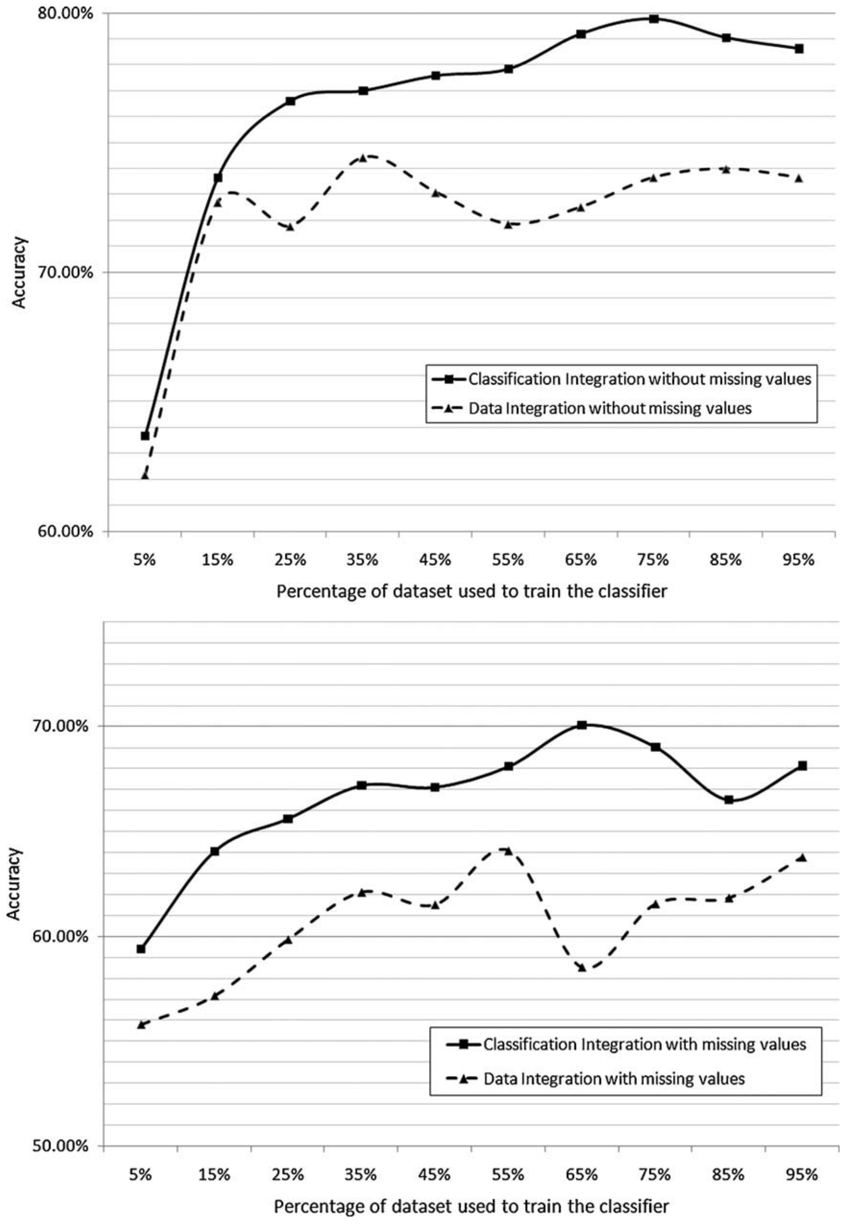


Figure 6. Comparison of data integration with classification integration using SVMs without (top) and with (bottom) missing values in the training set.

(3) Build a classification f_{123} on the union of the training data.
(4) Build a classification f_1, f_2, f_3 for each training data source.
(5) Integrate f_1, f_2, f_3 using *classification integration* to find f_{class_integ}.
(6) Test the accuracy of f_{123} and f_{class_integ} on the testing set.

Figs. 6 and 7 (top) report the average results of repeating the above procedure ten times for each n equal to $5, 15, 25, \ldots, 95$ when using SVMs and decision trees respectively as original classifiers.

We also simulated missing values by generating the following three subsets from the original data set by removing two features from each source:

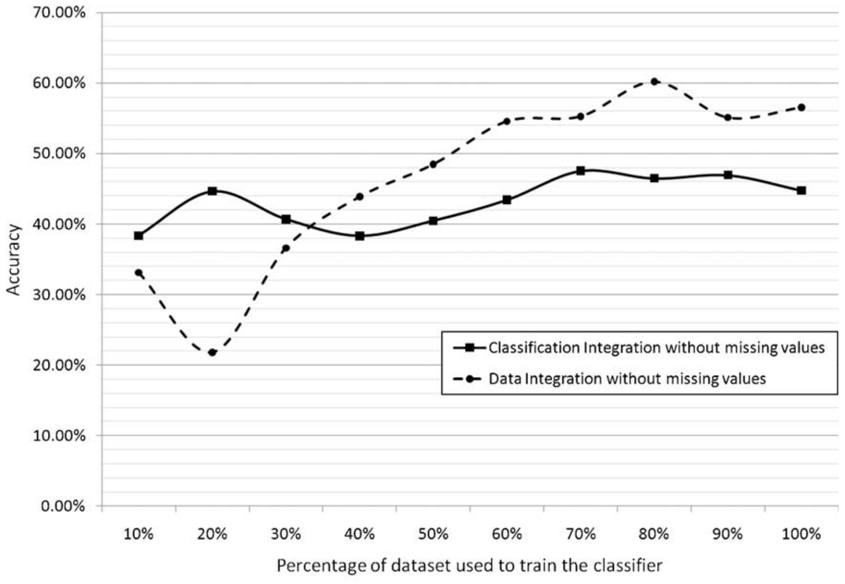
- BUDAPEST with features (1, 2, 3, 4, 5, 7, 9, 10)
- CLEVELAND with features (1, 2, 3, 5, 7, 8, 9, 10)
- ZÜRICH with features (1, 2, 3, 5, 6, 7, 9, 10)

In each subset, we used the first seven features to predict the last one. We used those three subsets in a similar manner as described in the above procedure.

Results: The *classification integration* method was more accurate than the *data integration* method when both used SVMs (Fig. 6) but less accurate when both used decision trees (Fig. 7). Surprisingly, classification integration improves in both cases when the training data set contains missing values.

Discussion: Classification integration and data integration can be compared for both accuracy and general applicability.

Accuracy: We can say that in general classification integration is more accurate than data integration when both use SVMs. Further, the experiments suggest that the increased accuracy is due to classification integration preserving better the relationships found by the original classifiers. As a hypothetical example, if chest pain is strongly related to heart attack, it will be recognized as such in a hospital where the chest pain of each patient is recorded. However, if the data from that hospital is aggregated with data from other hospitals that did not measure chest pain, then the relationship of chest pain and heart attack may be less clear and recognizable. In that case, a classifier used on the integrated data may miss some of the relationships. That seems to be the reason for the increase of the relative performance of the classification integration over data integration when there are missing values.



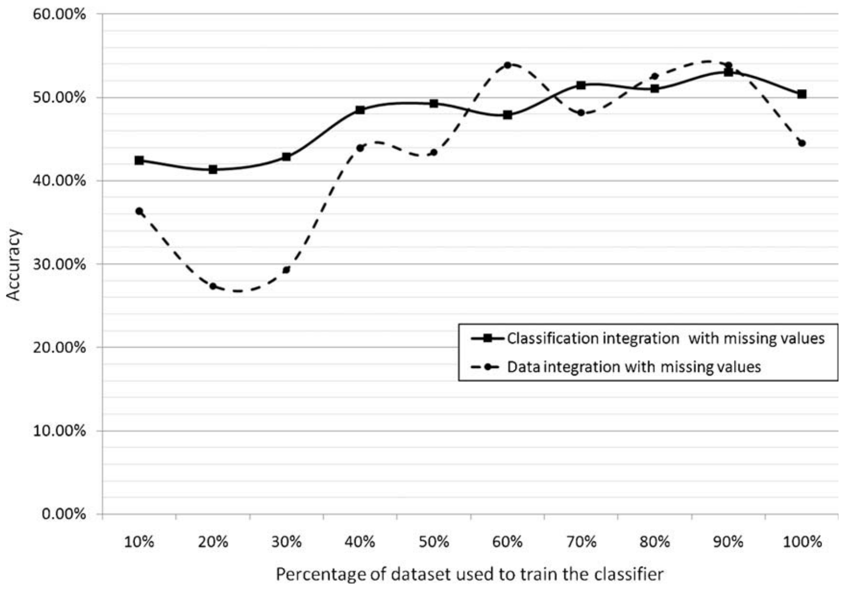


Figure 7. Comparison of data integration with classification integration using decision trees without (top) and with (bottom) missing values in the training set.

Based on the SVM experiments, our hypothesis is that as the amount of missing information increases, the classification integration will eventually outperform data integration in the case of other linear classification methods too. Since the experimental results will vary in accuracy according to the actual classifier used and the data, the exact degree of missing information needed for the classification integration to outperform data integration will also vary.

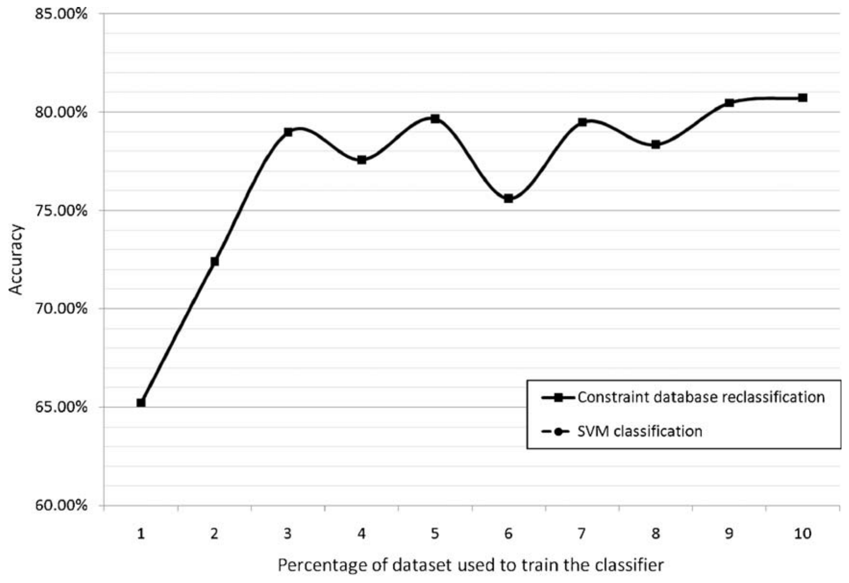
Applicability: Classification integration has a wider applicability than data integration has because a frequent issue in medicine is the lack of availability of raw data due to privacy and legal concerns. When raw data is not available, standard data integration methods cannot be applied but classification integration is still possible to apply.

5.2. Reclassification with an oracle vs. reclassification with constraint databases

In this section, we compare the *reclassification with an oracle* and the *reclassification with constraint databases* methods. We also compare *reclassification with constraint databases* with the original linear classification for the DISEASE and the DRUG classes.

We used a Mayo Clinic data set [27], which contains the medical records of 314 *primary biliary cirrhosis* patients with the following recorded features:

(1) case number,
(2) days between registration and earliest of death, transplantion, or study,
(3) age in days,
(4) gender (0 = male, 1 = female),
(5) asictes present (0 = no or 1 = yes),
(6) hepatomegaly present (0 = no or 1 = yes),
(7) spiders present (0 = no or 1 = yes),
(8) edema (0 = none, 0.5 = edema resolved, 1 = edema despite diuretics),
(9) serum bilirubin in mg/dl,
(10) serum cholesterol in mg/dl,
(11) albumin in mg/dl,
(12) urine copper in μg/day,
(13) alkaline phosphatase in Units/liter,
(14) SGOT in Units/ml,
(15) triglicerides in mg/dl,



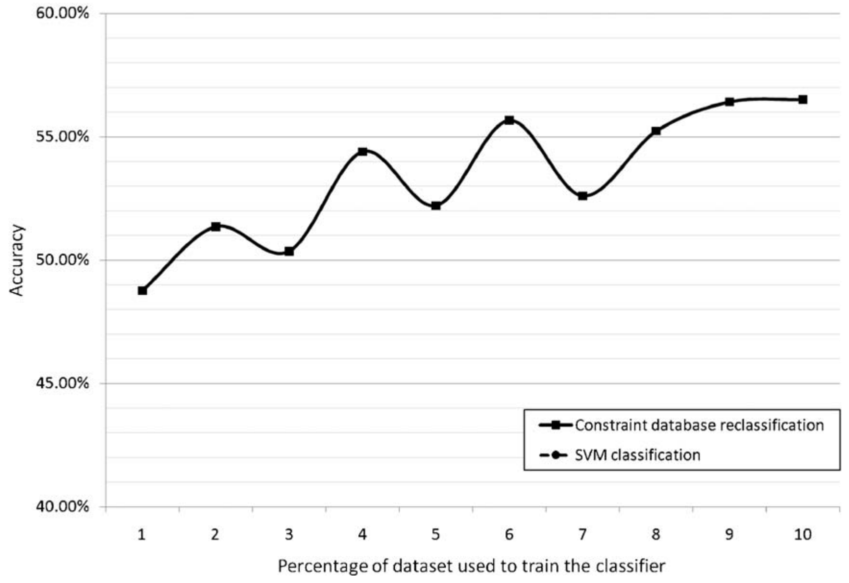


Figure 8. Prediction of DISEASE (top) of the patients and DRUG (bottom) from the primary biliary cirrhosis data using SVMs. Note that the SVMs and their constraint database representations are identical.

(16) platelets per cubic ml/1000,
(17) prothrombin time in seconds,
(18) status (0 = alive, 1 = transplant, or 2 = dead),
(19) drug (1 = D-penicillamine or 2 = placebo), and
(20) histologic stage of the disease (1, 2, 3, 4).

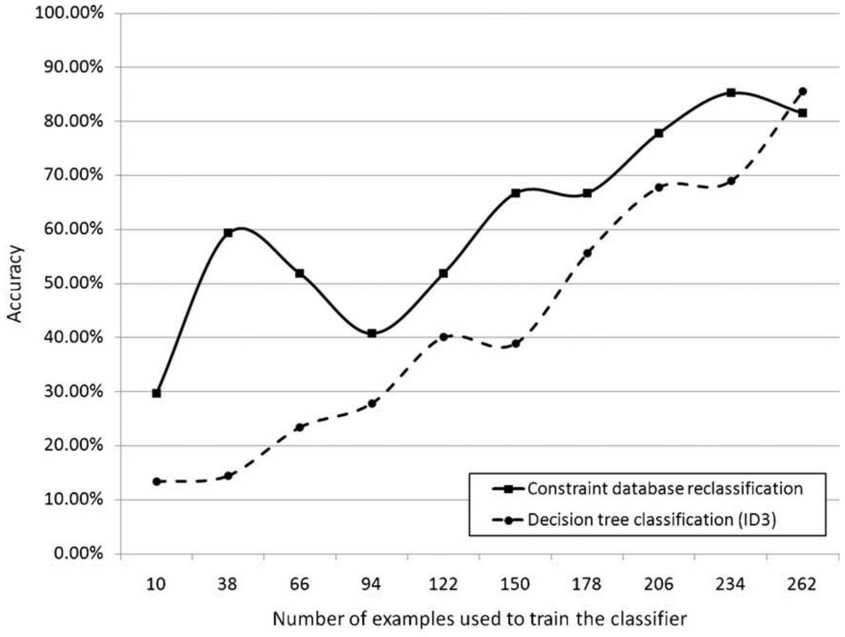
We generated the following two subsets from the original data set:

- DISEASE with features (3, 4, 5, 7, 8, 9, 10, 13, 14, 16, 17, 20)
- DRUG with features (3, 4, 6, 7, 8, 9, 10, 11, 13, 16, 17, 19)

Results: In both subsets, we used the first eleven features to predict the last feature. Fig. 8 shows the results of our experiments using SVMs as original classifications. The constraints defining the separating hyperplane of the SVMs can be exactly represented in the constraint database; hence the accuracies of the original SVMs and their constraint database representation are identical. Fig. 9 shows the results with ID3 decision trees. Constraint databases provide a more flexible representation of the original classifications. The experimental results show that constraint databases may significantly improve the accuracy of the classification depending on the choice of the original classifier.

Fig. 10 compares the accuracies of the reclassification with an oracle and with constraint databases for SVMs (top) and decision trees (bottom). The results show that the *reclassification with an oracle* and the *reclassification with constraint databases* perform similarly for both SVMs and decision trees.

Discussion: In any experiment, we consider the *theoretical limit* to be the maximum achievable with the use of the original linear classification algorithm. Fig. 10 proves that our practical *reclassification with constraint databases* method achieves the theoretical limit represented by the *reclassification with an oracle* method.



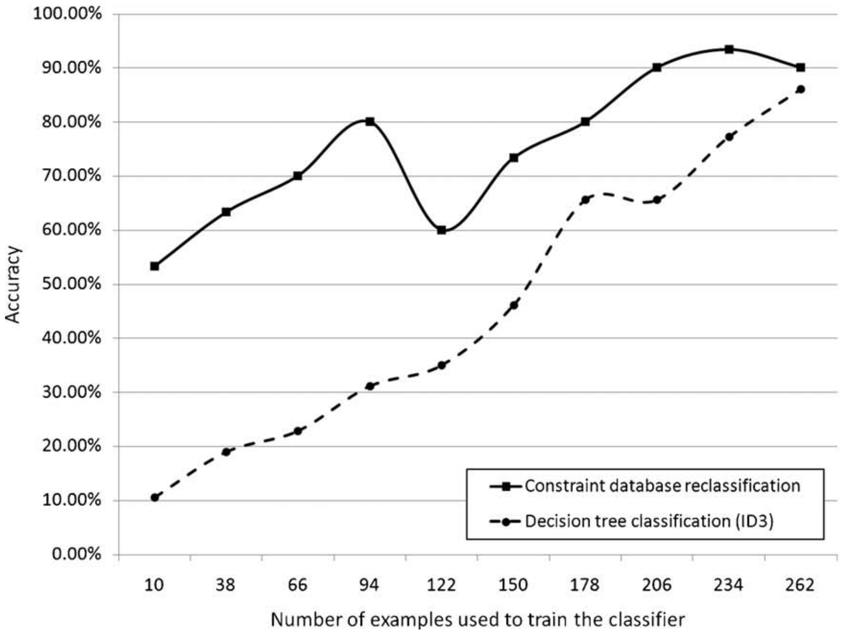
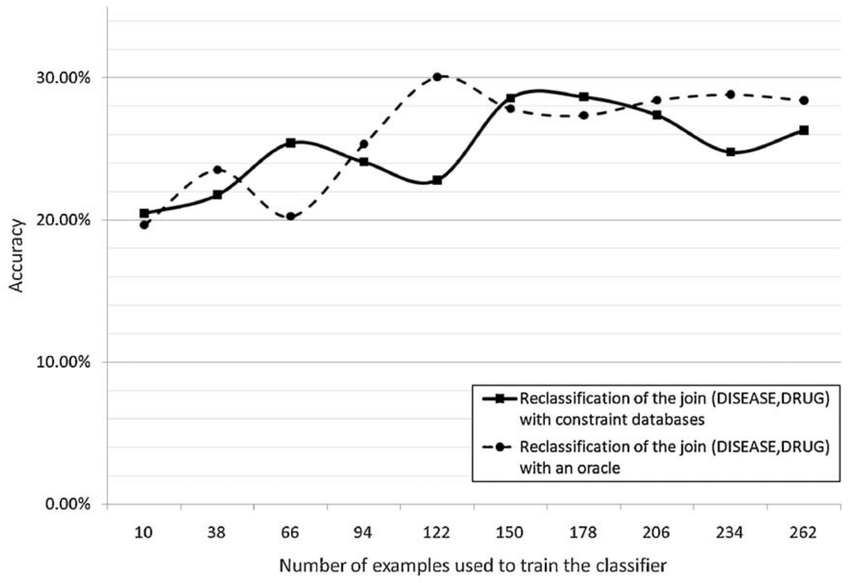


Figure 9. Prediction of DISEASE (top) of the patients and DRUG (bottom) from the primary biliary cirrhosis data using ID3 decision trees.



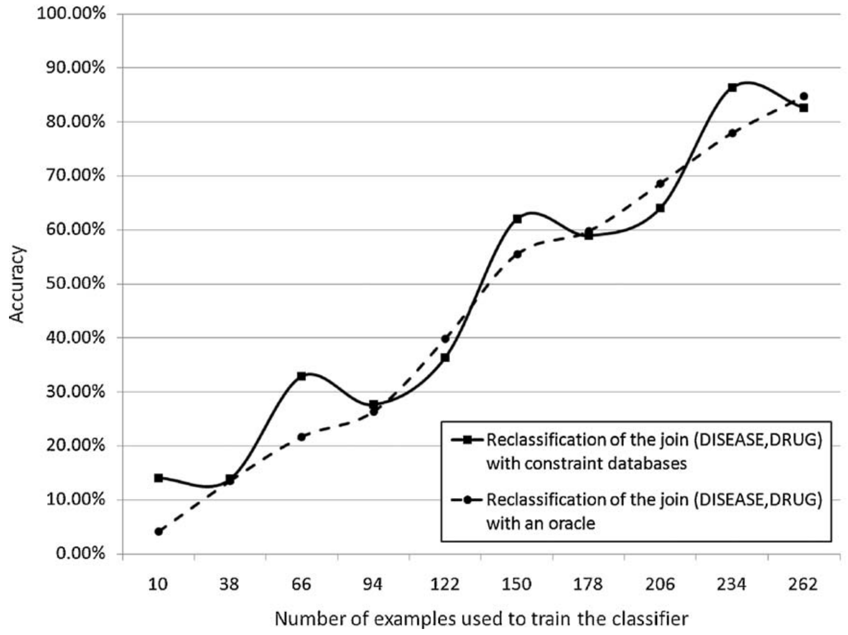


Figure 10. Prediction of DISEASE_DRUG with the primary biliary cirrhosis data using SVMs (top) and ID3 decision trees (bottom).

6. Conclusion

This paper presented the classification integration method and experimentally showed that when classification integration is built upon support vector machines and decision trees and there are many missing values in the data, then it is more accurate than current data integration methods. In addition the reclassification problem was also shown to be solvable using constraint databases without requiring access to the raw data.

In the future, we plan to experiment with other data sets and use non-linear classifiers in addition to SVMs and decision trees. Another interesting direction is to investigate extensions of our methods when the input data vary in time. For example, instead of having all the measurements of a patient at the same time, we have different pieces of measurements with different time instances. Constraint databases are already used for various spatio-temporal applications, and it would be interesting to explore the connections between these applications and the problem of dealing with temporally varying data.

Acknowledgement

The first author was supported in part by a J. William Fulbright U.S. scholarship and the second by a Milton E. Mohr fellowship.

References

[1] Vapnik V. The Nature of Statistical Learning Theory. New York: Springer-Verlag; 1995.
[2] Quinlan JR. Induction of decision trees. Machine Learning 1986;1(1):81–106.
[3] Louie B, Mork P, Martń-Sánchez F, Halevy AY, Tarczy-Hornoch P. Data integration and genomic medicine. Journal of Biomedical Informatics 2007;40(1):5–16.
[4] Calì A, Calvanese D, De Giacomo G, Lenzerini M. Data integration under integrity constraints. Information Systems 2004;29(2):147–63.
[5] Kuper GM, Libkin L, Paredaens J, editors. Constraint databases. Berlin: Springer-Verlag; 2000.
[6] Revesz P. Introduction to Constraint Databases. New York: Springer-Verlag; 2002.
[7] Revesz P, Triplet T. Reclassification of linearly classified data using constraint databases. In: Atzeni P, Caplinskas A, Jaakkola H, editors. 12th East European Conference on Advances of Databases and Information Systems, 2008.

[8] Codd EF. A relational model for large shared data banks. Communications of the ACM 1970;13(6):377–87.
[9] Abiteboul S, Hull R, Vianu V. Foundations of databases. Reading, MA: Addison-Wesley; 1995.
[10] Ramakrishnan R, Gehrke J. Database management systems, 3rd ed., New York: McGraw-Hill; 2002.
[11] Ullman JD. Principles of database and knowledge-base systems. New York: Computer Science Press; 1989.
[12] Brodsky A, Segal V, Chen J, Exarkhopoulo P. The CCUBE constraint object-oriented database system. Constraints 1997;2(3–4):245–77.
[13] Rigaux P, Scholl M, Segoufin L, Grumbach S. Building a constraint-based spatial database system: model, languages, and implementation. Information Systems 2003;28(6):563–95.
[14] Bishop B, Fischer F, Keller U, Steinmetz N, Fuchs C, Pressnig M. Integrated Rule Inference System, 2008. Software available at: http://www.irisreasoner.org. Accessed: 16 May 2009.
[15] Revesz P, Chen R, Kanjamala P, Li Y, Liu Y, Wang Y. The MLPQ/GIS constraint database system. In: Chen W, Naughton J, Bernstein PA, editors. Proceedings of the ACM SIGMOD International Conference on Management of Data, 2000.
[16] Kanellakis PC, Kuper GM, Revesz P. Constraint query languages. Journal of Computer and System Sciences 1995;51(1):26–52.
[17] Rigaux P, Scholl M, Voisard A. Introduction to spatial databases: applications to GIS. San Francisco: Morgan Kaufmann; 2000.
[18] Chomicki J, Haesevoets S, Kuijpers B, Revesz P. Classes of spatiotemporal objects and their closure properties. Annals of Mathematics and Artificial Intelligence 2003;39(4):431–61.
[19] Güting R, Schneider M. Moving Objects Databases. San Francisco: Morgan Kaufmann; 2005.
[20] Anderson S, Revesz P. Efficient MaxCount and threshold operators of moving objects. Geoinformatica 2009;13(4):355–96.
[21] Revesz P, Wu S. Spatiotemporal reasoning about epidemiological data. Artificial Intelligence in Medicine 2006;38(2):157–70.
[22] Gómez-López MT, Ceballos R, Gasca RM, Del Valle C. Developing a labelled object-relational constraint database architecture for theprojection operator. Data and Knowledge Engineering 2009;68(1):146–72.
[23] Geist I. A framework for data mining and KDD. In: Haddad H, Papadopoulos G, editors. Proceedings of the ACM Symposium on Applied Computing, 2002.
[24] Lakshmanan LVS, Leung CKS, Ng RT. Efficient dynamic mining of constrained frequent sets. ACM Transactions on Database Systems 2003;28(4):337–89.
[25] Asuncion A, Newman DJ. University of California Irvine, School of Information and Computer Science. Data available at: http://archive.ics.uci.edu/ml/. Last accessed: January 15, 2009.
[26] Chang CC, Lin CJ. LIBSVM: a library for support vector machines, 2001. Software available at: http://www.csie.ntu.edu.tw/cjlin/libsvm. Accessed: 16 May 2009.
[27] Fleming TR, Harrington DP. Counting Processes and Survival Analysis. New York: Wiley; 1991.