Partial Least Squares Regression for Graph Mining

Hiroto Saigo
Max Planck Institute for Biological Cybernetics
Spemannstr. 38
Tübingen, Germany
hiroto.saigo@tuebingen.mpg.de

Nicole Krämer
TU Berlin
Franklinstr. 28/29, 10587
Berlin, Germany
nkraemer@cs.tu-berlin.de

Koji Tsuda
Max Planck Institute for Biological Cybernetics
Spemannstr. 38
Tübingen, Germany
koji.tsuda@tuebingen.mpg.de

ABSTRACT

Attributed graphs are increasingly more common in many application domains such as chemistry, biology and text processing. A central issue in graph mining is how to collect informative subgraph patterns for a given learning task. We propose an iterative mining method based on partial least squares regression (PLS). To apply PLS to graph data, a sparse version of PLS is developed first and then it is combined with a weighted pattern mining algorithm. The mining algorithm is iteratively called with different weight vectors, creating one latent component per one mining call. Our method, graph PLS, is efficient and easy to implement, because the weight vector is updated with elementary matrix calculations. In experiments, our graph PLS algorithm showed competitive prediction accuracies in many chemical datasets and its efficiency was significantly superior to graph boosting (gBoost) and the naive method based on frequent graph mining.

Categories and Subject Descriptors

I.5.2 [Pattern Recognition]: Design Methodology—Feature evaluation and selection; H.2.8 [Database Management]: Database Applications—Data mining

General Terms

Algorithms, Experimentation, Performance

Keywords

Partial least squares regression, graph mining, graph boosting, chemoinformatics

1. INTRODUCTION

As data mining and machine learning techniques continue to evolve and improve, the role of structure in the data becomes more and more important. Much of the real world data is represented not as vectors, but as graphs including sequences and trees, for example, biological sequences, semi-structured texts such as HTML and XML, chemical compounds, RNA secondary structures, and so forth. Like ordinary vectorial data, there are two kinds of learning tasks; unsupervised [30, 31] and supervised [22]. Among supervised learning tasks, graph regression and classification would be of wide interest. In graph regression, an attributed graph is given as an input, and a real-valued output variable is predicted. In classification, the output variable is binary.

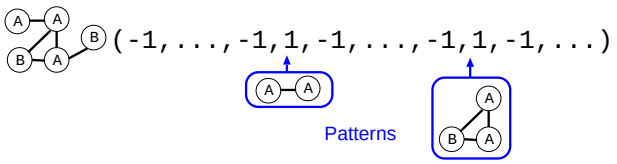

Figure 1: Feature space based on subgraph patterns. The feature vector consists of binary pattern indicators.

In learning from graph data, one can rely on the similarity measures derived from graph alignment [28] or graph kernels [10, 23, 6, 16]. However, one drawback is that the features used in learning are implicitly defined, and derived clusters are hard to interpret. Another approach is based on graph mining, where a set of small graphs (i.e., patterns) is used to represent a graph. Specifically, each graph is represented as a binary vector of pattern indicators (Figure 1). Graph mining is especially popular in chemoinformatics, where the task is to classify chemical compounds [11, 7]. When all possible subgraphs are used, the dimensionality of the feature space is too large for usual statistical methods. Therefore, feature collection is a central issue in graph mining algorithms [30, 1, 35].

To summarize the feature collection methods proposed so far, let us classify them into two categories: mine-at-once and iterative mining. In the first category, the whole feature space is built by one mining run before the subsequent machine learning algorithm is started. A naive approach is to use a frequent substructure mining algorithm such as AGM [9], gSpan [36] or Gaston [20] to collect frequently appearing patterns. This approach was employed by [7] and [11], where a linear support vector machine is used for classification. A more advanced approach is to mine informative patterns with high correlation to the output variable [19, 1]. However, salient patterns depend on the optimal parameters of the subsequent learning algorithm, and it is difficult to obtain a small number of features informative for any learning algorithm [12].

Among iterative mining methods, substructure boosting methods [15, 22, 26] have been successfully applied to many different domains such as images [22], videos [21], chemical compounds [26] and biological mutation sets [27]. The boosting algorithm calls a pattern mining algorithm repeatedly to incrementally form a feature space. In the first iteration, the patterns with high correlation with

KDD'08 August 24–27, 2008, Las Vegas, Nevada, USA.

the target variable are collected. In subsequent iterations, the algorithm updates the example weights such that more emphasis is put on mispredicted examples. It is reported that it creates less useless features compared to the mine-at-once methods [22]. In the very first paper by Kudo et al. [15], AdaBoost was used for updating the example weights. However, AdaBoost is not efficient in graph mining, because it takes too many iterations to finish. Thus recent papers use mathematical programming-based approaches such as linear programming boosting (LPBoost) [3, 24] and quadratic programming boosting (QPBoost) [26]. Furthermore, to reduce the number of iterations, several patterns are collected at the same time in one iteration by multiple pricing [22]. Nevertheless, substructure boosting can still be improved in term of efficiency, because the computation time for mathematical programming is substantially large. In itemset boosting [27], it is reported that the computational time for mathematical programming is much larger than that needed for mining. In particular, when solving a regression problem, one has to use a quadratic program that is computationally more demanding than a linear program.

We propose a new iterative mining method based on partial least squares regression (PLS) [33, 25, 34]. PLS is an iterative algorithm that extracts latent features iteratively from a high dimensional space. An attractive point of PLS is that it depends only on elementary matrix calculations (i.e., addition and multiplications). Therefore, it is more efficient than other methods depending on mathematical programming or eigen-decomposition. In gBoost, the transition from vectorial to graph data is achieved by replacing the feature selection step by a pattern mining algorithm [22]. In PLS, it is not so simple, because conventional algorithms for PLS such as NIPALS [33] require the deflation of the whole feature matrix. The feature matrix consists of the feature vectors of all training examples, and NIPALS substracts a dense matrix from the feature matrix in each iteration. It is possible only if the whole feature matrix is loaded to memory, which is not practical in graph mining.

In this paper, we develop a sparse version of non-deflation PLS such that each latent component depends on a limited number of subgraph patterns. Then, it is combined with a pattern mining algorithm to deal with graph data. We call our algorithm graph PLS or gPLS in short. gPLS collects informative patterns in a limited number of iterations, as it avoids the discovery of identical patterns by means of orthogonality constraints. Like gspan and gBoost, gPLS employs the DFS code tree [36] as the canonical search space of graph patterns. The criterion for pattern search is quite simple and it turns out that the pattern search algorithm in gBoost (i.e., weighted substructure mining) can be reused in gPLS as well.

This paper is organized as follows. In Section 2, we introduce the PLS regression and present its non-deflation version. Section 3 explains how PLS is applied to graph data. In Section 4, extensive experiments for various chemical datasets are presented. Section 5 discusses other possibilities in developing graph regression algorithms. Finally, we conclude the paper in Section 6.

2. PARTIAL LEAST SQUARES REGRESSION

This section reviews the partial least squares regression (PLS) algorithm for vectorial data. We first explain the conventional NIPALS algorithm and introduce a new non-deflation algorithm. The transition from vectorial to graph data will be discussed in the next section.

Let us assume n training examples $(x_1, y_1), \ldots, (x_n, y_n)$ where $x_i \in \Re^d$ and $y_i \in \Re$. The output y_i is assumed to be centralized $\sum_i y_i = 0$. Denote by X the design matrix, where each row corresponds to $x_i^\top$. Also denote by y the vector of all training outputs.

The regression function of PLS is linear, but the following special form,

$$f(x) = \sum_{i=1}^{m} \alpha_i w_i^\top x, \tag{1}$$

where w_i are weight vectors that reduce the dimensionality of x, satisfying the following orthogonality condition:

$$w_i^\top X^\top X w_j = \begin{cases} 1 & (i = j) \\ 0 & (i \neq j) \end{cases} \tag{2}$$

We need to determine two kind of parameters w_i and α_i. Basically, w_i are learned first, and the coefficients α_i are obtained by least squares regression without any regularization,

$$\alpha = \operatorname*{argmin}_{\alpha} \sum_{k=1}^{n} (y_k - \sum_{i=1}^{m} \alpha_i w_i^\top x_k)^2. \tag{3}$$

Due to the orthogonality conditions, this problem is easily solved as

$$\alpha_i = \sum_{k=1}^{n} y_k w_i^\top x_k. \tag{4}$$

The weight vectors are determined by the following greedy algorithm. The first vector is obtained by maximizing the covariance between the mapped feature Xw_1 and the output variable y,

$$w_1 = \operatorname*{argmax}_{w} \frac{(\sum_{k=1}^{n} y_k w^\top x_k)^2}{w^\top w} \tag{5}$$

subject to $w^\top X^\top X w = 1$. This problem is solved analytically as

$$w_1 = \frac{1}{\delta} X^\top y$$

where δ is the normalization factor

$$\delta = \sqrt{y^\top X X^\top X X^\top y}.$$

For the i-th weight vector, the same optimization problem is solved with additional constraints to keep orthogonality,

$$w_i = \operatorname*{argmax}_{w} \frac{(\sum_{k=1}^{n} y_k w^\top x_k)^2}{w^\top w} \tag{6}$$

subject to

$$w^\top X^\top X w = 1, \quad w^\top X^\top X w_j = 0, \ j = 1, \ldots, i-1.$$

The optimal solution of this problem cannot be obtained analytically. Since the regression of α is done without any regularization, it is important to choose the number of weight vectors appropriately. Typically, it is chosen to optimize the cross validation error, or other model selection criteria such as AIC and BIC [25].

2.1 NIPALS

Let us define the i-th latent component as $t_i = Xw_i$. The NIPALS algorithm [33] solves the optimization problem (6) in an indirect way, namely the optimal latent components t_i are obtained first and the corresponding w_i is obtained later. Let us define T_{i-1} as the matrix of latent components obtained so far,

$$T_{i-1} = (t_1, \ldots, t_{i-1})$$

and define a projection matrix as

$$P_{i-1} = T_{i-1}(T_{i-1}^\top T_{i-1})^{-1} T_{i-1}^\top = T_{i-1} T_{i-1}^\top. \tag{7}$$

The second equality is due to the orthogonal conditions (2). Then, a deflated design matrix $\tilde{X}$ is defined as

$$\tilde{X}_i = X - P_{i-1}X.$$

Now we solve the following problem based on the deflated matrix,

$$v_i = \underset{v}{\text{argmax}} \frac{(\sum_{k=1}^{n} y_k v^\top \tilde{x}_{ik})^2}{v^\top v}$$

where $v_i^\top \tilde{X}_i^\top \tilde{X}_i v_i = 1$. As in (5), the optimal solution has the form

$$v_i = \frac{1}{\eta}\tilde{X}_i^\top y. \tag{8}$$

where $\eta = \sqrt{y^\top \tilde{X}_i \tilde{X}_i^\top \tilde{X}_i \tilde{X}_i^\top y}$. In literature, we could not find appropriate terminology for v_i, but here we call it the i-th pre-weight vector, because it is used to create the "real" weight vector w_i. Based on v_i, the optimal latent component is obtained as

$$t_i = \tilde{X}_i v_i.$$

Finally, we have to recover the optimal weight vector w_i based on the following equation [8],

$$Xw_i = t_i = \tilde{X}_i v_i = Xv_i - P_{i-1}Xv_i.$$

Assuming the linear independence of rows of X, the equation is solved as

$$w_i = v_i - \sum_{j=1}^{i-1}(w_j^\top X^\top X v_i)w_j, \tag{9}$$

which corresponds to the optimal solution of (6).

The NIPALS algorithm consists of only elementary matrix computations and therefore is more efficient than solving (6) as an constrained quadratic program. The algorithm is summarized in Algorithm 1. Due to the following relationship,

$$\tilde{X}_i = \tilde{X}_{i-1} - t_{i-1}t_{i-1}^\top \tilde{X}_{i-1},$$

the deflated matrix is updated rather than recomputed in each iteration. However, for our purpose, the crucial drawback is that the sparseness of X is lost by deflation.

Algorithm 1 The NIPALS algorithm.

1: Initial: $\tilde{X}_1 = X$
2: for $i = 1, \ldots, m$ do
3: $\quad v_i = \tilde{X}_i^\top y/\eta$. ▷ Pre-weight vector
4: $\quad t_i = \tilde{X}_i v_i$ ▷ Latent components
5: $\quad \tilde{X}_{i+1} = \tilde{X}_i - t_i t_i^\top \tilde{X}_i$ ▷ Deflation
6: end for
7: Conversion of v_i to w_i for all i as (9)

2.2 Non-deflation sparse PLS

We now present an alternative derivation of PLS that avoids the deflation step and that is based on the connection of PLS the the Lanczos method and that uses recursive fitting of residuals [5, 13].

Substituting the definition of the projection matrix to the pre-weight vector (8), we obtain

$$v = \frac{1}{\eta}X^\top(I - T_{i-1}T_{i-1}^\top)y. \tag{10}$$

The NIPALS algorithm first computes the deflated matrix $X^\top(I - T_{i-1}T_{i-1}^\top)$ and then multiplies it with y. However, an obvious alternative way is to compute the residual vector

$$r_i = (I - T_{i-1}T_{i-1}^\top)y.$$

and then multiply it with $X^\top$. Following this idea, the NIPALS algorithm can be modified to a non-deflation version (Algorithm 2).

In graph mining, it is useful to have sparse weight vectors w_i such that only a limited number of patterns are used for prediction. To this aim, we modify the algorithm further by introducing sparseness to the pre-weight vectors v_i as follows:

$$v_{ij} = 0, \quad \text{if } |v_{ij}| \le \epsilon, \ j = 1, \ldots, d.$$

Due to the linear relationship between v_i and w_i, it is understood that w_i becomes sparse as well. The sparse weight vectors satisfy the orthogonality conditions (2). There are two alternative ways to determine the threshold ϵ: 1) Sort $|v_{ij}|$ in the descending order, take the top-k elements, and set all the other elements to zero. 2) Set ϵ to a fixed threshold. In the latter case, the number of non-zero elements in v_i may vary. In the experiments presented in this paper, we took the former top-k approach to avoid unbalanced weight vectors and to make efficiency comparisons easier.

It is worthwhile to notice that the residual of regression up to the $i-1$-th features,

$$r_{ik} = y_k - \sum_{j=1}^{i-1}\alpha_j w_j^\top x_k \tag{11}$$

is equal to the k-th element of r_i. It can be verified by substituting the definition of α_j (4) into (11). So in the non-deflation algorithm, the pre-weight vector v is obtained as the direction that maximizes the covariance with residues. This observation highlights the resemblance of PLS and boosting algorithms [3]. In boosting, example weights are iteratively altered such that the examples with high residues are weighted more. In this formulation of PLS, it is clearer that the residue vector plays a role similar to that of boosting's example weights. The connection between PLS and boosting is discussed in [17].

Algorithm 2 Non-deflation Sparse PLS algorithm.

1: for $i = 1, \ldots, m$ do
2: $\quad r_i = (I - T_{i-1}T_{i-1}^\top)y$ ▷ Residue
3: $\quad v_i = X^\top r_i/\eta$. ▷ Pre-weight vector
4: $\quad v_{ij} = 0$, if $|v_{ij}| \le \epsilon$, $j = 1, \ldots, d$ ▷ Sparsify
5: $\quad w_i = v_i - \sum_{j=1}^{i-1}(w_j^\top X^\top X v_i)w_j$ ▷ Weight vector
6: $\quad t_i = Xw_i$ ▷ Latent components
7: end for

3. GRAPH PLS (GPLS)

In this section, we discuss how to apply the non-deflation PLS algorithm to graph data. Here we deal with undirected, labeled and connected graphs. To be more precise, we define the graph and its subgraph as follows:

Definition 1 (Labeled connected graph). A labeled graph is represented in a 4-tuple $G = (V, E, \mathcal{L}, l)$, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, $\mathcal{L}$ is a set of labels, and $l \ \ V \cup E \to \mathcal{L}$ is a mapping that assigns labels to the vertices and edges. A labeled connected graph is a labeled graph such that there is a path between any pair of vertices.

Definition 2 (Subgraph). Let $G' = (V', E', \mathcal{L}', l')$ and $G = (V, E, \mathcal{L}, l)$ be labeled connected graphs. G' is a subgraph of G

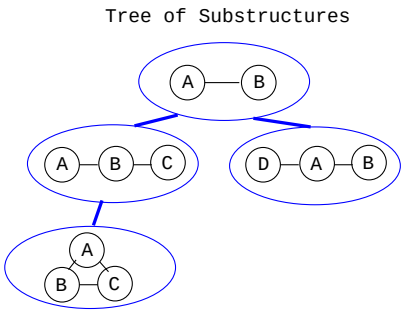

Figure 2: Schematic figure of the tree-shaped search space of graph patterns (i.e., the DFS code tree). To find the optimal pattern efficiently, the tree is systematically expanded by rightmost extensions.

($G' \subseteq G$) if the following conditions are satisfied: (1) $V' \subseteq V$, (2) $E' \subseteq E$, (3) $\mathcal{L}' \subseteq \mathcal{L}$, (4) $\forall v' \subseteq V', l(v') = l'(v')$ and (5) $\forall e' \subseteq E', l(e') = l'(e')$. If G' is a subgraph of G, then G is a supergraph of G'.

Our training set is represented as $(G_1, y_1), \ldots, (G_n, y_n)$ where G_i is a graph and $y_i \in \Re$ is a target value. Let p be a subgraph pattern in a graph, and $\mathcal{P}$ be the set of all patterns, i.e., the set of all subgraphs included in at least one graph. Then, the whole feature vector of each graph G_i is encoded as a $|\mathcal{P}|$-dimensional vector x_i,

$$x_{ip} = \begin{cases} 1 & \text{if } p \subseteq G_i, \\ -1 & \text{otherwise} \end{cases}$$

This feature space has already been illustrated in Figure 1. Since $|\mathcal{P}|$ is a huge number, we cannot keep the whole design matrix. So we need to set X as the empty matrix first, and grow the matrix as the iterations proceed. In each iteration, we obtain the set of patterns p whose pre-weight $|v_{ip}|$ is above the threshold, which can be written as

$$P_i = \{p \mid \left| \sum_{j=1}^{n} r_{ij} x_{jp} \right| \geq \epsilon\}. \tag{12}$$

Then, the design matrix is expanded to include newly introduced patterns. The pseudocode of gPLS is described in Algorithm 3. Most numerical computations are carried over from Algorithm 2 except that the residue vector is updated.

The pattern search problem (12) is exactly the same as the one solved in gBoost [22]. So we can reuse the same method to enumerate P_i. More specifically, it can be done by gspan function in the gBoost MATLAB toolbox[1]. However, we explain the pattern search algorithm briefly for the completeness of this paper.

Our search strategy is a branch-and-bound algorithm that requires a canonical search space in which a whole set of patterns are enumerated without duplication. As the search space, we adopt the DFS code tree [36]. The basic idea of the DFS code tree is to organize patterns as a tree, where a child node has a supergraph of the pattern in its parent node. (Figure 2). A pattern is represented as a text string called the DFS (depth first search) code. The patterns are enumerated by generating the tree from the root to leaves using a recursive algorithm. To avoid duplications, node generation is systematically done by rightmost extensions. Algorithm 4 shows the pseudo code for the recursive algorithm.

For efficient search, it is important to minimize the size of the search space. To this aim, tree pruning is crucially important [18, 15]. Let us define the gain function as $s(p) = \left| \sum_{j=1}^{n} r_{ij} x_{jp} \right|$. Suppose the search tree is generated up to the pattern p. If it is guaranteed that the gain of any supergraph p' is not larger than ϵ, we can avoid the generation of downstream nodes without losing the optimal pattern. Our pruning condition is described as follows.

[1] http://www.kyb.mpg.de/bs/people/nowozin/gboost/

Algorithm 3 gPLS

```
r_1 = y,  X = ∅
for i = 1, ..., m do
    P_i = {p | |Σ_{j=1}^n r_ij x_jp| ≥ ε}        ▷ Pattern search
    X_{P_i}: design matrix restricted to P_i
    X ← X ∪ X_{P_i}
    v_i = X^⊤ r_i / η                            ▷ Pre-weight vector
    w_i = v_i − Σ_{j=1}^{i−1} (w_j^⊤ X^⊤ X v_i) w_j   ▷ Weight vector
    t_i = X w_i                                  ▷ Latent component
    r_{i+1} = r_i − (y^⊤ t_i) t_i                ▷ Update residues
end for
```

Algorithm 4 Pattern search algorithm

```
procedure Pattern Search
    P ← ∅
    for p ∈ DFS codes with single nodes do
        project(p)
    end for
    return P
end procedure
function project(p)
    if p is not a minimum DFS code then
        return
    end if
    if pruning condition (13) holds then
        return
    end if
    if p satisfies the condition (12) then
        P ← P ∪ {p}
    end if
    for p' ∈ rightmost extensions of p do
        project(p')
    end for
end function
```

Theorem 1. Define $\tilde{y}_i = \mathrm{sgn}(r_i)$. For any pattern p' such that $p \subseteq p', s(p') < \epsilon$, if

$$\max\{s^+(p), s^-(p)\} < \epsilon, \tag{13}$$

where

$$s^+(p) = 2 \sum_{\{i \mid \tilde{y}_i = +1, x_{i,j} = 1\}} |r_i| - \sum_{i=1}^{n} r_i$$

$$s^-(p) = 2 \sum_{\{i \mid \tilde{y}_i = -1, x_{i,j} = 1\}} |r_i| + \sum_{i=1}^{n} r_i.$$

Other conditions such as the maximum size of pattern (maxpat) and the minimum support (minsup) can be used in combination with the pruning condition (13).

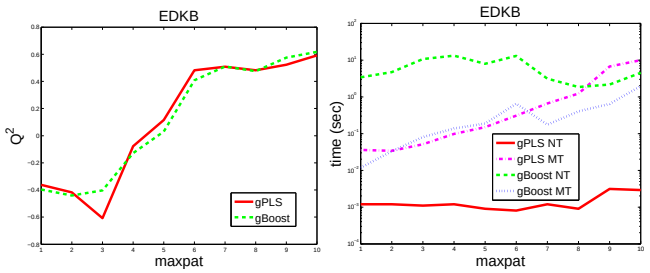

Figure 3: Regression accuracy (left) and computational time (right) against maximum pattern size (maxpat) in the EDKB dataset.

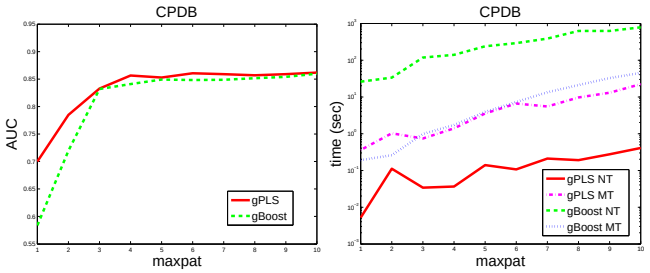

Figure 4: Classification accuracy (left) and computational time (right) against maximum pattern size (maxpat) in the CPDB dataset.

4. EXPERIMENTS

In this section, we evaluate our method using four publicly available chemical datasets: EDKB[2], CPDB[3], CAS[4] and AIDS[5]. Links to these datasets can be found in ChemDB [2]. Table 1 shows the summary of the datasets. Among them, the AIDS dataset [14, 4] is by far the largest both in the number of examples and the graph size. EDKB is a regression dataset, but the others are classification datasets. In gPLS, we solved classification problems by regressing the target values $+1, -1$. In gBoost, we employed the gBoost MATLAB toolbox for classification datasets so that the experimental results are easily reproducible. Since the toolbox does not offer regression solvers, we implemented a graph boosting regression algorithm based on quadratic programming. See Appendix for details.

We set minimum support parameter (minsup) to 2 for relatively small datasets (EDKB, CPDB and AIDS1), and to 10% of the number of positives for large datasets (CAS, AIDS2 and AIDS3). Throughout the experiments maximum pattern size (maxpat) is set to 10. We used AMD Opteron 2.2GHz system with at most 8GB memory for all experiments.

4.1 gPLS vs gBoost

GPLS is compared with gBoost in five fold cross validation experiments. In gPLS, there are two parameters to tune, namely the number of iterations m and the number of obtained patterns per search k. For each dataset, we exhaustively tried all combinations from $m = \{5, 10, 15, 20, 25, 30, 35\}$ and $k = \{5, 10, 15, 20, 25, 30, 35\}$. In the following, we always report the best test accuracy among all settings. Notice that, for AIDS datasets, the parameter values are changed as $m = \{10, 20, 30, 40, 50\}$, $k = \{10, 20, 30, 40, 50\}$ to cope with large-scale data. In gBoost, the regularization parameter was varied as $\nu = \{0.1, 0.2, \ldots, 0.9\}$ for classification, and $C = \{10, 50, 100, 150, 200, 1000\}$ for regression. The number of patterns to add per iteration is set to 50 for CAS and AIDS, and 10 for the other datasets. The accuracy is measured by Q^2 for regression and by the area under the ROC curve (AUC) for classification. The Q^2 score is defined as

$$Q^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - f(x_i))^2}{\sum_{i=1}^{n} \left(y_i - \frac{1}{n}\sum_{i=1}^{n} y_i\right)^2}$$

which is close to 1 when the regression function fits good, and is close to 0 when it does not. The interpretation is similar to that for the Pearson correlation coefficient.

The results of gPLS and gBoost are compared in Table 2. For EDKB and CPDB datasets, we performed more detailed experiments with different settings of maximum pattern size (Figure 3 and 4). In terms of accuracy, it is difficult to decide which method is better. GPLS was better in EDKB, CPDB and AIDS1 but gBoost was better in CAS. However, in terms of computational time, gPLS is clearly superior. In the table, we distinguish the computational time for pattern search (mining time, MT) and the numerical computations (numerical time, NT). The numerical time of gBoost was significantly larger than that of gPLS in all datasets, showing that gPLS's computational simplicity contributes to reduce the actual computational load. For large datasets (AIDS2 and AIDS3), gBoost did not finish in a reasonable amount of time.

Figure 5 shows the patterns selected by gPLS from the EDKB dataset. It is often observed that similar patterns are extracted together in the same component. This property makes PLS stable, because the regression function is less affected by small changes in graph data.

4.2 Efficiency gain by iterative mining

The main idea of iterative mining is to gain efficiency by means of adaptive example weights. We evaluated how large the efficiency gain is by comparing gPLS and a naive method that enumerates all patterns first and apply PLS afterwards. Table 3 summarizes the results for different maximum pattern sizes (maxpat). In the naive method, the number of patterns grow exponentially, hence the computational time for PLS grows rapidly as well. GPLS successfully keeps computational time small in all occasions.

5. DISCUSSION

So far, we have mainly focused on gPLS and gBoost. They are similar in that graph patterns are iteratively collected based on the weighted mining criterion. Obviously, they are not the only ones belonging to this family of algorithms. Other boosting methods and PLS-like methods could be applied to graph data in the same fashion.

However, it is important to recognize that there are two distinct classes in boosting algorithms, sequential update and totally corrective update. Both classes are based on linear classification function,

$$f(x) = \sum_{i=1} w_i x_i .$$

In sequential update algorithms including AdaBoost and its variants (GradientBoost, MadaBoost etc.), a new feature x_{i+1} is added in one iteration and the corresponding weight w_{i+1} is determined. However, previously fixed weights are never updated again. In these methods, numerical computation per iteration is very simple, but it

[2]http://edkb.fda.gof/databasedoor.html
[3]http://potency.berkeley.edu/cpdb.html
[4]http://www.chemoinformatics.org/datasets/bursi/
[5]http://dtp.nci.nih.gov/index.html

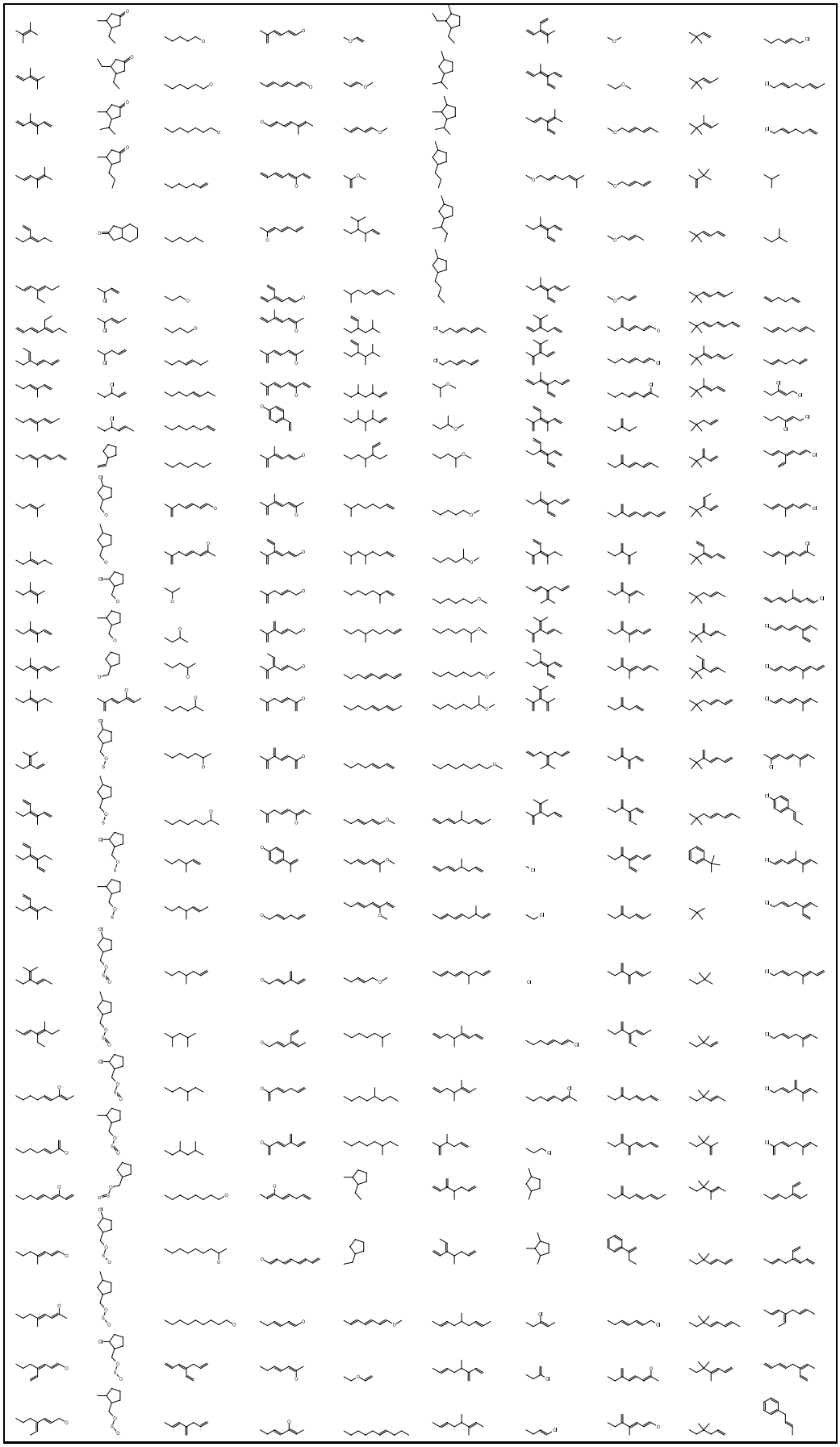

Figure 5: Patterns obtained by gPLS from the EDKB datasets. Each column corresponds to the patterns of a PLS component.

Table 1: Summary of datasets.

	label type	# data	# positives	# negatives	avg. atoms	avg. bonds
EDKB	real	59	-	-	18.5	20.1
CPDB	binary	684	342	343	14.1	14.6
CAS	binary	4337	2401	1936	29.9	30.9
AIDS1 (CA vs CM)	binary	1324	350	974	48.8	51.0
AIDS2 (CA CM vs CI)	binary	40939	1324	39615	42.7	44.6
AIDS3 (CA vs CI)	binary	39965	350	39615	42.7	44.5

Table 2: Results of gPLS and gBoost in various datasets. Values in the parentheses are optimal parameters achieving the best test accuracy. P: the average number of obtained patterns, MT: mining time, NT: numerical time, ITR: the number of iterations required until convergence.

	gPLS					gBoost					
	(m, r)*	P	MT	NT	AUC/Q^{2†}	$(\nu/C^{\dagger})$*	P	MT	NT	AUC/Q^{2†}	ITR
EDKB	(10, 30)	296	16.0	0.0025	$0.647^{\dagger} \pm 0.129$	$(100^{\dagger})$	216	15.6	83.3	$0.639^{\dagger} \pm 0.164$	9.2
CPDB	(20, 15)	258	26.8	0.474	0.862 ± 0.0214	(0.4)	260	22.8	344	0.862 ± 0.0316	18.6
CAS	(30, 10)	294	3570	14.1	0.870 ± 0.0098	(0.4)	503	8630	391	0.867 ± 0.000251	13.4
AIDS1	(10, 10)	99	290	0.0652	0.773 ± 0.0538	(0.4)	186	783	299	0.752 ± 0.138	19.6
AIDS2	(40, 10)	396	50300	167	0.747 ± 0.0266				over 24h		
AIDS3	(50, 20)	946	57100	509	0.883 ± 0.0541				over 24h		

takes many iterations to converge. In graph mining, each iteration involves pattern search, so sequential update algorithms are not efficient after all. On the other hand, totally corrective methods, such as gPLS, gBoost and TotalBoost [32], update all weights whenever new features are introduced. It requires more complicated numerical computation but the number of iterations can be by far smaller. Also it is possible to collect several patterns in each iteration, which substantially helps to reduce the number of iterations further. Graph LARS [29] could be considered as a totally corrective method as it updates all weights. However, since it is based on regularization path tracking, it is not clear how to collect more than one pattern by a mining call.

6. CONCLUSION

We presented a novel graph regression method based on partial least squares regression. Experiments showed that gPLS has better efficiency than gBoost. However, gPLS cannot completely replace gBoost, because gBoost has an advantage in its flexibility. With a little modification in mathematical programming formulation, gBoost can solve various machine learning problems, such as one-class SVM, ranking and the positive/negative unbalanced classification problem. In this paper, we used graph data only, but gPLS can be applied to subclasses of graphs such as trees, sequences and itemsets, simply by replacing graph mining with an appropriate mining algorithm.

Acknowledgements

The authors would like to thank Pierre Mahé for data preparation, Ichigaku Takigawa for figure preparation, and Sebastian Nowozin for preparation of MATLAB toolbox and proof reading.

7. REFERENCES

[1] B. Bringmann, A. Zimmermann, L. D. Raedt, and S. Nijssen. Don't be afraid of simpler patterns. In 10th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), pages 55–66. Sprinter, 2006.

[2] J. Chen, S. J. Swamidass, Y. Dou, J. Bruand, and P. Baldi. Chemdb: A public database of small molecules and related chemoinformatics resources. Bioinformatics, 21(22):4133–4139, 2005.

[3] A. Demiriz, K. Bennet, and J. Shawe-Taylor. Linear programming boosting via column generation. Machine Learning, 46(1-3):225–254, 2002.

[4] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. Frequent sub-structure-based approaches for classifying chemical compounds. IEEE Trans. Knowl. Data Eng., 17(8):1036–1050, 2005.

[5] L. Eldén. Partial least squares vs. lanczos bidiagonalization i: Analysis of a projection method for multiple regression. Computational Statistics and Data Analysis, 46(1):11–31, 2004.

[6] H. Fröhrich, J. Wegner, F. Sieker, and Z. Zell. Kernel functions for attributed molecular graphs - a new similarity based approach to ADME prediction in classification and regression. QSAR & Combinatorial Science, 25(4):317–326, 2006.

[7] C. Helma, T. Cramer, S. Kramer, and L. Raedt. Data mining and machine learning techniques for the identification of mutagenicity inducing substructures and structure activity relationships of noncongeneric compounds. J. Chem. Inf. Comput. Sci., 44:1402–1411, 2004.

[8] A. Höskuldsson. PLS Regression Methods. Journal of Chemometrics, 2:211–228, 1988.

[9] A. Inokuchi. Mining generalized substructures from a set of labeled graphs. In Proceedings of the 4th IEEE Internatinal Conference on Data Mining, pages 415–418. IEEE Computer Society, 2005.

[10] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In Proceedings of the 21st International Conference on Machine Learning, pages 321–328. AAAI Press, 2003.

Table 3: Frequent mining + PLS vs gPLS in the CPDB dataset

	frequent mining + PLS				gPLS			
maxpat	# patterns	mining time	numerical time	AUC	# patterns	mining time	numerical time	AUC
1	17	0.0927	0.038	0.696	15.2	0.308	0.038	0.700
2	61	0.148	0.0164	0.770	45.8	1.20	0.1169	0.782
3	182	0.212	0.0335	0.812	73.8	1.09	0.0573	0.833
4	515	0.282	0.0923	0.842	82.6	2.06	0.0488	0.857
5	1387	0.602	0.221	0.846	93.4	1.97	0.0296	0.844
6	3500	2.55	0.525	0.852	85.6	2.67	0.0222	0.833
7	8215	4.38	1.60	0.848	65.4	3.19	0.0146	0.837
8	18107	7.6	5.32	0.840	172	13.1	0.247	0.857
9	37719	17.9	7.42	0.840	209	12.7	0.282	0.859
10	74857	40.3	51.2	0.842	244	26.8	0.474	0.862
11	143006	70.3	92.8	0.835	244	35.4	0.375	0.862
12	out of memory				244	46.3	0.367	0.862
13	out of memory				244	52.4	0.549	0.861
∞	out of memory				244	66.3	0.586	0.861

[11] J. Kazius, S. Nijssen, J. Kok, and T. B. A. Ijzerman. Substructure mining using elaborate chemical representation. J. Chem. Inf. Model., 46:597–605, 2006.

[12] R. Kohavi and G. H. John. Wrappers for feature subset selection. Artificial Intelligence, 1-2:273–324, 1997.

[13] N. Krämer and M. Braun. Kernelizing partial least squares, degrees of freedom, and efficient model selection. In Proceedings of the 24th International Conference on Machine Learning, pages 441 – 448. AAAI Press, 2007.

[14] S. Kramer, L. Raedt, and C. Helma. Molecular feature mining in HIV data. In Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM Press, 2001.

[15] T. Kudo, E. Maeda, and Y. Matsumoto. An application of boosting to graph classification. In Advances in Neural Information Processing Systems 17, pages 729–736. MIT Press, 2005.

[16] P. Mahé, L. Ralaivola, V. Stoven, and J.-P. Vert. The pharmacophore kernel for virtual screening with support vector machines. J. Chem. Inf. Model., 46(5):2003–2014, 2006.

[17] M. Momma and K. Bennett. Constructing orthogonal latent features for arbitrary loss. Feature Extraction, Foundations and Applications. Springer, 2006.

[18] S. Morishita. Computing optimal hypotheses efficiently for boosting. In Discovery Science, pages 471–481. Springer, 2001.

[19] S. Morishita and J. Sese. Traversing itemset lattices with statistical metric learning. In Proceedings of ACM SIGACT-SIGMOD-SIGART Symposium on Database Systems (PODS), pages 226–236. ACM Press, 2000.

[20] S. Nijssen and J. Kok. A quickstart in frequent structure mining can make a difference. In Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 647–652. ACM Press, 2004.

[21] S. Nowozin, G. Bakir, and K. Tsuda. Discriminative subsequence mining for action classification. In Proceedings of the 11th IEEE International Conference on Computer Vision (ICCV 2007), pages 1919–1923. IEEE Computer Society, 2007.

[22] S. Nowozin, K. Tsuda, T. Uno, T. Kudo, and G. Bakir. Weighted substructure mining for image analysis. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, 2007.

[23] L. Ralaivola, S. Swamidass, H. Saigo, and P. Baldi. Graph kernels for chemical informatics. Neural Netw., 18(8):1093–1110, 2005.

[24] G. Rätsch, A. Demiriz, and K. Bennett. Sparse regression ensembles in infinite and finite hypothesis spaces. Machine Learning, 48(1-3):189–218, 2002.

[25] R. Rosipal and N. Krämer. Overview and recent advances in partial least squares. In Subspace, Latent Structure and Feature Selection Techniques, pages 34–51. Springer, 2006.

[26] H. Saigo, T. Kadowaki, and K. Tsuda. A linear programming approach for molecular QSAR analysis. In International Workshop on Mining and Learning with Graphs (MLG), pages 85–96, 2006.

[27] H. Saigo, T. Uno, and K. Tsuda. Mining complex genotypic features for predicting HIV-1 drug resistance. Bioinformatics, 23(18):2455–2462, 2007.

[28] A. Sanfeliu and K. Fu. A distance measure between attributed relational graphs for pattern recognition. IEEE Trans. Syst. Man Cybern., 13:353–362, 1983.

[29] K. Tsuda. Entire regularization paths for graph data. In Proceedings of the 24th International Conference on Machine Learning, pages 919–926, 2007.

[30] K. Tsuda and T. Kudo. Clustering graphs by weighted substructure mining. In Proceedings of the 23rd International Conference on Machine Learning, pages 953–960. ACM Press, 2006.

[31] K. Tsuda and K. Kurihara. Graph mining with variational dirichlet process mixture models. In SIAM Conference on Data Mining (SDM), 2008.

[32] M. Warmuth, J. Liao, and G. Rätsch. Totally corrective boosting algorithms that maximize the margin. In Proceedings of the 23rd International Conference on Machine Learning, pages 1001–1008, 2006.

[33] H. Wold. Path models with latent variables: The NIPALS approach. In Quantitative Sociology: International Perspectives on Mathematical and Statistical Model Building, pages 307–357. Academic Press, 1975.

[34] S. Wold, M. Sjöström, and L. Erikkson. PLS-regression: a basic tool of chemometrics. Chemometrics and intelligent laboratory systems, 58:109–130, 2001.

[35] X. Yan, H. Cheng, J. Han, and P. S. Yu. Mining significant graph patterns by leap search. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 433–444, 2008.

[36] X. Yan and J. Han. gSpan: graph-based substructure pattern mining. In Proceedings of the 2002 IEEE International Conference on Data Mining, pages 721–724. IEEE Computer Society, 2002.

APPENDIX

Here we briefly describe the gBoost regression algorithm. Boosting methods construct a linear combination of weak hypotheses to come up with a better prediction. In our case, a weak hypothesis corresponding to each subgraph pattern p is described as

$$x_{ip} = \begin{cases} 1 & \text{if } p \subseteq G_i, \\ -1 & \text{otherwise.} \end{cases}$$

The regression function is formulated as

$$f(x) = \sum_{i=1}^{n} \alpha_p x_{ip} + b,$$

where α, b are weight parameters to be learned. The learning problem is written as

$$\underset{\alpha,b}{\operatorname{argmin}} \sum_{p\in\mathcal{P}} |\alpha_p| + \frac{C}{2}\sum_{i=1}^{n}\Big(\sum_{p\in\mathcal{P}} \alpha_p x_{ip} + b - y_i\Big)^2,$$

where C is the regularization parameter to be adjusted. Using the L_1-norm regularizer (the first term), sparsity is enforced to the parameters.

The problem is rewritten as the following quadratic program.

$$\min_{\alpha,\xi,b} \quad \sum_{p\in\mathcal{P}} (\alpha_p^+ + \alpha_p^-) + \frac{C}{2}\sum_{i=1}^{n}\xi_i^2 \tag{14}$$

$$s.t. \quad \sum_{p\in\mathcal{P}} \alpha_p x_{ip} + b - y_i \le \xi_i, \quad i = 1,\ldots,n \tag{15}$$

$$y_i - \sum_{p\in\mathcal{P}} \alpha_p x_{ip} - b \le \xi_i, \quad i = 1,\ldots,n \tag{16}$$

$$\alpha^+, \alpha^- \ge 0, \quad \xi \ge 0, \tag{17}$$

where ξ_i is a slack variable, $\alpha_p = \alpha_p^+ - \alpha_p^-$. The above quadratic program has $|\mathcal{P}|$ variables and $2n$ constraints. Directly solving this primal problem is hard due to the large number of variables in α. Thus, we consider the dual problem:

$$\min_{\lambda} \quad \frac{1}{2C}\sum_{i=1}^{n}(\lambda_i^+ + \lambda_i^-)^2 - \sum_{i=1}^{n} y_i(\lambda_i^+ - \lambda_i^-) \tag{18}$$

$$s.t. \quad -1 \le \sum_{i=1}^{n}(\lambda_i^+ - \lambda_i^-)x_{ip} \le 1, \ \forall p \in \mathcal{P} \tag{19}$$

$$\sum_{i=1}^{n} \lambda_i^+ - \lambda_i^- = 0, \quad \lambda^+, \lambda^- \ge 0, \tag{20}$$

where λ_i^+ and λ_i^- are Lagrange multipliers for the constraints (15) and (16), respectively. Once the dual problem is solved, the primal solution α and b are recovered from the Lagrange multipliers of the dual problem. Though the dual problem has too many constraints, it can be efficiently solved by an iterative procedure called the column generation algorithm [3]. First of all, an initial solution of λ is obtained from the problem with no constraints (19). In each iteration, one finds the most violated constraint based on the current value of λ, and add the found constraint to the quadratic program. In our case, a constraint corresponds to a subgraph pattern, so we need to solve the following search problem,

$$\underset{p\in\mathcal{P}}{\operatorname{argmax}} \left|\sum_{i=1}^{n} \lambda_i x_{ip}\right|,$$

where $\lambda = \lambda^+ - \lambda^-$. This search problem coincides with that of gPLS (12), and can be solved using the same algorithm (Algorithm 4). In each iteration of the algorithm, a dual quadratic program with a limited number of constraints is solved and the obtained solution will be used in the next search. The iteration will be continued until the dual parameter λ converges.