Fast discriminative pattern mining using sparsity-inducing loss functions

No Institute Given

Keywords: discriminative pattern mining, graph mining, sparsity, support vectors, classification, regression

Abstract. Apriori-based mining algorithms enumerate frequent patterns efficiently, but the resulting large number of patterns makes it difficult to directly apply subsequent learning tasks. Recently, efficient iterative methods are proposed for mining discriminative patterns for classification and regression. These methods iteratively execute discriminative pattern mining algorithm and update example weights to emphasize on examples which received large errors in the previous iteration. In this paper, we study a family of loss functions that induces sparsity on example weights. Most of the resulting example weights become zeros, so we can eliminate those examples from discriminative pattern mining, leading to a significant decrease in search space and time. In computational experiments we compare and evaluate various loss functions in terms of the amount of sparsity induced and resulting speed-up obtained.

1 Introduction

Structured data is becoming increasingly popular in data mining and machine learning. Much of the worlds' interesting data are not vectorial (tabular) data, but structured data such as trees, sequences and graphs. Examples of such data includes HTML and RNA secondary structures as trees, time series data as sequence, chemical compounds and social networks as graphs. Influenced by the pioneering work of [1] for mining frequent association rules, various frequent pattern mining algorithms are developed for various class of structured data; such as LCM [17] for itemsets, TREEMINER [21] for trees, PrefixSpan [10] for sequences and gSpan [20] for graphs. These frequent structure enumeration algorithms give us a foundation to apply basic statistical learning tools on the obtained set of patterns. However, it is often argued that the number of frequent patterns is too large for the subsequent learning tasks, thus summarization of frequent patterns is necessary [19]. A common heuristic to overcome this difficulty is to set support (frequency of a pattern) high or maxpat (maximum pattern size) low to limit the number of resulting frequent patterns [18]. More advanced approaches attempt to mine discriminative graphs by using the labels of examples as external information source to prune the search space [2]. Correlation or Information gain are typically employed to estimate the informativeness of

patterns and prune uninteresting patterns. However, the set of patterns collected by such a two-step method is not optimal for different learning tasks.

More recently, substructure boosting approach has been successfully applied to different learning tasks on various kinds of data including RNA secondary structure clustering [16], video classification [9], and QSAR [12]. These methods combine statistical learning algorithms with pattern mining algorithms to directly mine discriminative patterns which are optimal for the subsequent learning task in an iterative fashion [7]. The basic strategy is similar to ordinary boosting where examples which received large errors in the previous iteration are intensively learned in the next iteration. In each iteration, one feature is added to the solution set, and the weights for all the previously found features are updated. The algorithm consists of two parts, namely, discriminative pattern mining part which searches for the most discriminative pattern, and the learning part which computes the example weights. In this paper, we study a family of loss functions that induce sparsity on example weights. The search space formed by

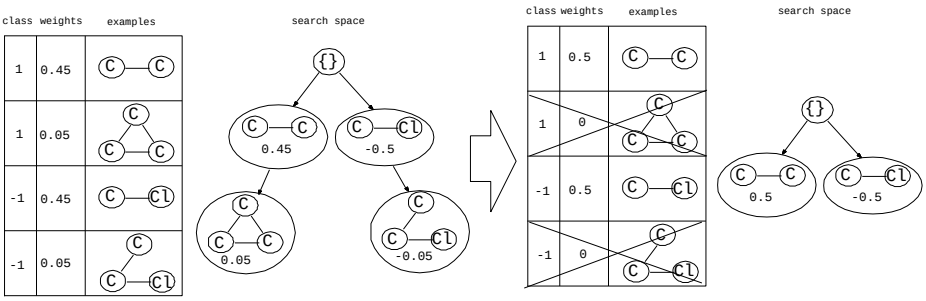

Fig. 1. Comparison of pattern mining with non-sparse example weights (left) to sparse example weights (right). If sparsity is enforced, example weights nearby zero (such as those of the second and the fourth example) on the left shrinks to zeros. Such examples can be eliminated from subsequent pattern mining, and the resulting search space for pattern mining shrinks from one on the left to the one on the right.

both non-sparse weights and sparse weights is illustrated in Figure 1. If sparsity is enforced, example weights nearby zero (such as those of the second and the fourth example) on the left shrinks to zeros. Such examples can be eliminated from subsequent pattern mining, and the resulting search space for pattern mining shrinks from one on the left to the one on the right. Therefore we can expect that pattern mining with sparse example weights is more efficient than one with non-sparse weights, and that we should fully exploit this property.

In machine learning literature, support vector machine (SVM) is known to achieve sparsity on the example weights, and only a small fraction of examples receives non-zero weights (so-called support vectors) [13]. We propose to take full advantage of sparsity of support vectors for speeding-up pattern mining. Lending

the knowledge from sparse kernel learning methods [13], we study different types of loss functions which induce sparsity.

This paper is organized as follows. In Section 2, we briefly review substructure boosting algorithm to understand what makes the example weight sparse, and give instances of loss functions which do not have sparse solutions. Section 3 considers regression methods in terms of ability to achieve sparsity on example weights. Section 4 shows computational results. Section 5 concludes the paper.

2 Review on substructure boosting for classification

This section briefly reviews substructure boosting algorithm. The substructure boosting algorithm constructs a linear model by progressively adding a feature at each iteration. Our feature vector is a binary indicator of patterns (Figure 2), and a label y_i is attached to each feature vector. We represent the presence or absence of the j-th pattern in the i-th graph by an indicator function which returns 1 if $x_{i,j} \in \boldsymbol{X}$, -1 otherwise, where $\boldsymbol{X}$ is a universe of patterns in a given dataset.

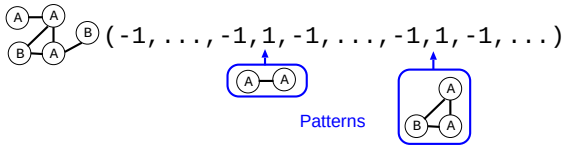

Fig. 2. Feature space based on subgraph patterns. The feature vector consists of binary pattern indicators.

Suppose for a moment that we solve classification problem, then our classifier is represented as a linear combination of patterns with corresponding weights;

$$y_i = \mathrm{sgn}(\sum_{j=1}^{p} x_{i,j}\beta_j),$$

where $y_i \in 0, 1$ is a binary target value of i-th graph, $\boldsymbol{x}_i$ is a length p vector corresponding pattern presence/absence in the i-th graph, and $\boldsymbol{\beta}$ is a length p weight vector to be learned. Note that the potential number of features p is quite large, so a large amount of memory is required when p is large. Therefore we regularize the weight vector $\boldsymbol{\beta}$ with respect to ℓ_1 norm so that most of patterns have zero weights. By employing hinge loss for classification (Figure 3), our objective function is written as

$$\min_{\boldsymbol{\beta}} \sum_{j=1}^{p} |\beta_j| + C \sum_{i=1}^{n} \left[1 - y_i \sum_{j=1}^{p} \beta_j x_{i,j} \right]_+ ,$$

where C is a regularization parameter and "+" indicates positive part. By introducing the slack variable ξ, we can formulate a linear programming problem corresponding to the above objective function.

$$\min_{\boldsymbol{\beta},\boldsymbol{\xi}} \sum_{j=1}^{p} |\beta_j| + C \sum_{i=1}^{n} \xi_i \tag{1}$$

$$\text{s.t. } y_i \sum_{j=1}^{p} x_{i,j}\beta_j + \xi_i \geq 1, \quad \xi_i \geq 0 \quad i = 1, \ldots, n. \tag{2}$$

Due to the high dimensionality of $\boldsymbol{\beta}$, solving the above primal problem is hard, thus we consider the equivalent dual problem;

$$\max_{\boldsymbol{u}} \sum_{i=1}^{n} u_i \tag{3}$$

$$\text{s.t. } \sum_{j=1}^{p} u_i x_{ij} y_i \leq 1, \quad j = 1, \ldots, p, \tag{4}$$

$$0 \leq u_i \leq C, \quad i = 1, \ldots, n. \tag{5}$$

This problem has a large number of constraints corresponding to equation (4), but column generation algorithm [4] can efficiently solve it by iteratively adding the mostly violated constraint. The constraint to be added is determined by solving the following column generation subproblem;

$$j^* = \operatorname*{argmax}_j \sum_{i=1}^{n} u_i x_{ij} y_i. \tag{6}$$

In our case this is equivalent to finding a pattern with the maximum absolute weighted support by discriminative pattern mining. For efficiently traversing the search space, pruning of the search space is crucial. We employ the following pruning condition that makes use of target labels y as extra information source,

Theorem 1. [8] Let us define

$$\mu(x_{(),j}) = \max\{2 \sum_{\{i|y_i=+1, x_{i,j}=1\}} u_i - \sum_{i=1}^{\ell} y_i u_i, 2 \sum_{\{i|y_i=-1, x_{i,j}=1\}} u_i + \sum_{i=1}^{\ell} y_i u_i\},$$

where $x_{(),j}$ denotes j-th pattern which appears at least once in a given data. If the following condition is satisfied,

$$g^* > \mu(x_{(),j}), \tag{7}$$

the inequality $g(x'_{(),j}) < g^*$ holds for any $x'_{(),j}$ such that $x_{(),j} \subseteq x'_{(),j}$. So we can safely prune the parent nodes of $x_{(),j}$ without losing the optimal pattern.

A pseudocode of this substructure boosting algorithm for classification is shown in Algorithm 1.

Algorithm 1 Substructure boosting algorithm

```
Initialization: X̂^(0) = ∅, u_i^(0) = 1/n, k = 0
loop
    Find the optimal pattern x* based on u^(k)
    if termination condition holds then
        break
    end if
    X̂ ← X̂ ∪ X_{j*}
    Solve the restricted dual problem (4 ) to obtain u^(k+1)
    k = k + 1
end loop
```

2.1 Sparsity on example weights

Notice that when $C \to \infty$, then solving (1) amounts to minimizing $\|\boldsymbol{\xi}\|_1$ while ignoring $\boldsymbol{\beta}$. This case is known as hard margin SVM which does not have regularization on $\boldsymbol{\beta}$ at all. In the dual, $C \to \infty$ corresponds to removing upperbound of u in equation (5);

$$\max_{\boldsymbol{u}} \sum_{j=1}^{p} u_j. \tag{8}$$

$$\text{s.t.} \sum_{j=1}^{p} u_i x_{ij} y_i \leq 1, \quad i = 1, \ldots, n, \tag{9}$$

The solution to this linear programming problem occurs at a vertex of a polyhedron, and most of the resulting $\boldsymbol{u}$ are zeros. Examples (data points) with nonzero weights $\boldsymbol{u}$ are known as support vectors in SVM literature [13]. Due to KKT condition, the following equations hold;

$$u_i \left(y_i \sum_j x_{ij} \beta_j - 1 + \xi_i \right) = 0, \quad u_i \geq 0, \quad y_i \sum_j x_{ij} \beta_j - 1 + \xi \geq 0,$$

that is, either $u_i = 0$ or $y_i \sum_j x_{ij} \beta_j - 1 + \xi_i = 0$ holds. In order to have more sparsity, more data points should satisfy $u_i = 0$. Geometrically speaking, such a region corresponds to a flat segment along the x axis in Figure (3).

The longer the segment, the more sparsity is induced. Among loss function for classification (Figure (3) left), only a hinge loss function turns out to induce sparsity. Inducing more sparsity is important in our case because examples with zero-weights can be eliminated from pattern mining. This effect is already illustrated in Figure 1. A figure on the left do not have sparsity on example weights, while one on the right has sparsity on example weights; the second and fourth examples can be eliminated since their examples weights are zero.

However, the problem of hard margin SVM is that it does not allow any error points during training, which is too restrictive in practice. Typically, the

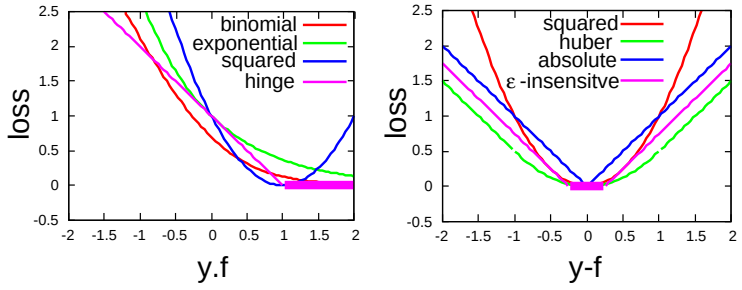

Fig. 3. Loss functions for binary classification (left). Loss functions for regression (right). Binomial deviance: $\log(1+\exp(-2yf))$, exponential loss: $\exp(-yf)$, squared loss $(y-f)^2$, hinge loss: $(1-yf)_+$, absolute loss: $|y-f|$, ϵ-insensitive loss $|y-f|_\epsilon$ and Huber's loss: $\frac{\delta}{2}(y-f)^2$ if $|y-f| \leq \delta$ and $\delta(|y-f|-\delta/2)$ otherwise.

trade-off between sparsity and the number of training errors is controlled by the regularization parameter C, which is found by a grid search from between 0 and ∞. ν-SVM or its linear programming version (ν-L1SVM) [4] provides us more sophisticated way of choosing regularization parameter. The primal problem of ν-L1SVM is written as follows;

$$\min_{\boldsymbol{\beta},\boldsymbol{\xi},\rho} \sum_{j=1}^{p} \beta_j + \frac{1}{n\nu} \sum_{i=1}^{n} \xi_i - \rho \tag{10}$$

$$\text{s.t.} \quad y_i \sum_{j=1}^{p} x_{i,j}\beta_j + \xi_i \geq \rho, \quad \xi_i \geq 0, \quad i = 1, \ldots, n, \tag{11}$$

where ν is a regularization parameter chosen from between 0 and 1. Equivalent dual problem is

$$\max_{\boldsymbol{u},\gamma} -\gamma \tag{12}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} y_i x_{ij} u_i \leq \gamma, \quad j = 1, \ldots, p, \tag{13}$$

$$\sum_{i=1}^{n} u_i = 1, \quad 0 \leq u_i \leq \frac{1}{n\nu}, \; i = 1, \ldots, n.$$

In the same way as in L1SVM, sparsity is enforced on example weights $\boldsymbol{u}$ as we set ν smaller, and recovers hard margin SVM in the limit $\nu \to 0$. Indeed ν controls the sparsity of the solution [14], and $n\nu$ is the lower bound of the number of support vectors [4]. Regarding the regularization parameter ν, the following statements hold:

Theorem 2 ([11]). Assume that the solution of (10) satisfies $\rho \geq 0$.

1. ν is an upperbound of the fraction of margin errors, i.e., the examples with

$$y_i \sum_{j=1}^{p} x_{i,j}\beta_j < \rho,\ i = 1, \ldots, n.$$

2. ν is a lowerbound of the fraction of the examples such that

$$y_i \sum_{j=1}^{p} x_{i,j}\beta_j < \rho,\ i = 1, \ldots, n.$$

Below, we abbreviate $\nu - L1SVM$ simply as L1SVM.

2.2 Non-sparse example weights of AdaBoost

In this subsection, we review AdaBoost [5] as an example of iterative learning algorithm which does not have sparsity on the example weights. AdaBoost iteratively generates a sequence of hypothesis functions to build a linear model that maximizes exponential loss (see Figure 3). The objective function of AdaBoost is as follows.

$$\begin{aligned} &\min_{\boldsymbol{\beta},\boldsymbol{\xi}} \ \exp\left(\sum_{i=1}^{n} \xi_i\right) \\ &s.t. \ \ y_i \sum_{j=1}^{p} x_{i,j}\beta_j \leq \xi_i, \quad \xi_i \geq 0 \quad i = 1, \ldots, n, \quad \beta_j \geq 0 \quad j = 1, \ldots, p \end{aligned}$$

The example weights of AdaBoost is updated by the following update rule [5].

$$u_i \leftarrow u_i \exp\left(-\log\left(\frac{1-err}{err}\right) \cdot I(y_i \neq f(x_i))\right), \tag{14}$$

where err stands for error rates of the current hypothesis. AdaBoost does not have a regularization on β, and the resulting β is has no structure such as sparsity.

3 Sparse substructure boosting for regression

This section deal with regression problem, so the target response value y takes real value. Without loss of generality, below we assume that $\boldsymbol{y}$ is center to zero. We compare two regression methods; LASSO (Least Absolute Shrinkage Operator) [15] that not induce sparsity and linear programming regression that induce sparsity on example weights.

LASSO employs least squared loss (Figure 3) and ℓ_1 regularization with respect to a parameter vector $\boldsymbol{\beta}$. The LASSO regression is formulated by the

quadratic programming problem as follows,

$$\min_{\boldsymbol{\beta},\xi} \sum_{j=1}^{p} \beta_j + \frac{C}{2} \sum_{i=1}^{n} \xi_i^2$$
$$s.t. \sum_{j=1}^{p} x_{i,j}\beta_j - y_i \leq \xi_i, \quad y_i - \sum_{j=1}^{p} x_{i,j}\beta_j \leq \xi_i, \quad \xi_i \geq 0 \quad i = 1, \ldots, n$$

where C is a regularization parameter. The dual of the LASSO is

$$\max_{\boldsymbol{u}} -\frac{1}{2C} \sum_{j=1}^{p} u_j^2 + \sum_{i=1}^{n} y_i u_i, \quad s.t. -1 \leq u_j \sum_{i=1}^{n} y_i x_{i,j} \leq 1, \quad j = 1, \ldots, p.$$

We can see that the example weights $\boldsymbol{u}$ is regularized with respect to ℓ_2-norm. ℓ_2-norm locates each u_j on the surface of an p-dimensional Euclid ball, but no one u_j becomes zero, so does not have sparsity. A sparse regression example is a linear programming regression(LPR) which employs ϵ-insensitive loss, and ℓ_1 norm on a parameter vector $\boldsymbol{\beta}$. The primal problem of LPR is as follows;

$$\min_{\boldsymbol{\beta},\boldsymbol{\xi}} \sum_{i=1}^{n} \beta_i + C \sum_{i=1}^{n} \xi_i + C\nu\epsilon$$
$$s.t. \sum_{j=1}^{p} x_{i,j}\beta_j - y_i \leq \epsilon + \xi_i, \quad y_i - \sum_{j=1}^{p} x_{i,j}\beta \leq \epsilon + \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, n$$

where C and ν are both regularization parameters. ν controls the ratio of support vectors inside the ϵ-tube, and C parameter controls the trade-off between overfitting and underfitting given ν. KKT condition tells us that either $u_i = 0$ or $\sum_j x_{ij}\beta_j - y - \epsilon + \xi_i = 0$ holds. Geometrically speaking, more data points should lie on a flat region of the ϵ-insensitive loss function (Figure 3) in order to have more sparsity. Sparsity and accuracy is trade-off [14], and controlled through C and ν.

4 Experiments

In this section, we compare several classification and regression methods in terms of induced sparsity and the resulting mining time. For the purpose of comparing different learning algorithms in a fair setting, we fix the mining algorithm to gSpan [20] for graph mining [1]. We show the basic statistics of the data used in Table 1.

[1] Graph mining toolbox, available from http://www.nowozin.net/sebastian/gboost/, is used for all the experiments.

Table 1. Datasets Summary. The number of positive data (POS) and negative data (NEG) are only provided for classification datasets. Average number of atoms (ATOM) and bonds (BOND) are shown for each dataset. TIME indicates the time in seconds for enumerating all the frequent patterns up to size 20.

	ALL	POS	NEG	ATOM	BOND	TIME
CPDB	684	341	343	14.1	14.6	7126
EDKB	146	-	-	19.5	21.1	2893

Table 2. Influence of the choice of ν parameter on L1SVM. Pat: the number of patterns with nonzero β, Itr: the number of iterations, ρ: the margin, Time: total time, SVs: mean ratio of support vectors over the iterations, Acc: the classification accuracy in the training set. We can observe that ν lowerbounds the number of support vectors.

ν	0.01	0.1	0.2	0.3	0.4
Itr	67	73	47	26	47
Pat	66	65	46	24	46
Time	1410	618	315	156	116
ρ	2.41e-11	0.130	0.0346	0.0809	0.143
SVs	0.537	0.549	0.572	0.585	0.745
Acc	0.993	0.973	0.938	0.892	0.839

The CPDB dataset is available from the supplementary information of [6], and used for classification experiments. The EDKB data is provided by National Center for Toxicological Research [2], and contains 146 molecules with activity levels in real number. This dataset is used for regression experiments. We used AMD Opteron 2.6GHz system with 32GB RAM for all the experiments. As a reference, frequent mining with minimum support 2 and maximum pattern size 20 was run, and it took 2893 seconds and 7126 seconds on EDKB dataset and CPDB dataset, respectively. The number of frequent subgraphs up to the size 20 were 4.4 million and 1 million for EDKB and CPDB dataset, respectively.

For classification problem, we compare L1SVM with AdaBoost in terms of induced sparsity on example weights and resulting running time. Convergence of L1SVM was checked using early stopping criterion $\sum y_i x_{ij} u_i \leq \gamma + \epsilon$, where ϵ is set to 0.05. AdaBoost was run 100 iterations.

Figure 4 shows the transition of example weights of AdaBoost(left) and L1SVM with ν set to 0.1(center) and 0.01(right). As expected, AdaBoost does not generates sparsity on examples weights (left). In contrast, examples weights of L1SVM become sparser as iteration proceeds (center, right). Setting $\nu = 0.1$ for L1SVM means that more than 10% of examples receive nonzero weights at each iteration. Behavior of L1SVM for various regularization parameter ν is summarized in Table 2.

We can observe that ν lowerbounds the number of support vectors.

[2] http://edkb.fda.gov/databasedoor.html

Figure 5 (left) shows the evolution of accuracy as a function of total time for L1SVM and AdaBoost. It is observed that L1SVM collect discriminative patterns and learns classifier much faster in total time than AdaBoost. Figure 5 (right) shows the mining time at each iteration. Due to the induced sparsity, mining time of L1SVM is shorter than that of AdaBoost except for last a few iterations. Mining time per iteration was 4.23 seconds and 16.4 seconds for AdaBoost and L1SVM, respectively. However, the last iterations of L1SVM did not contribute to the increase in accuracy, so one can stop it earlier by using validation set. Then the resulting mining time per iteration is shorter than that of AdaBoost as we can see in Figure 5 (right).

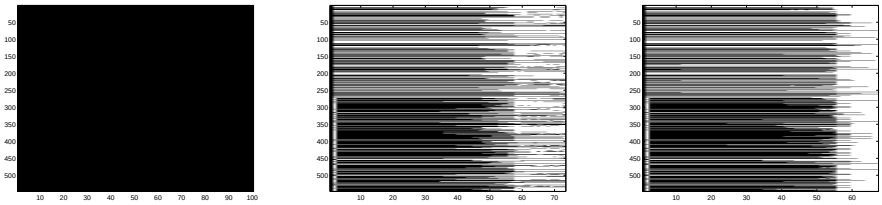

Fig. 4. Transition of example weights of AdaBoost (left), that of L1SVM ($\nu = 0.1$) (center) and that of L1SVM($\nu = 0.01$) (right). The vertical axis shows example ID, and the horizontal axis shows iterations. Nonzero weights are represented in black, and zero weights are represented in white. Notice that weights of L1SVM become increasingly sparser as iteration proceeds.

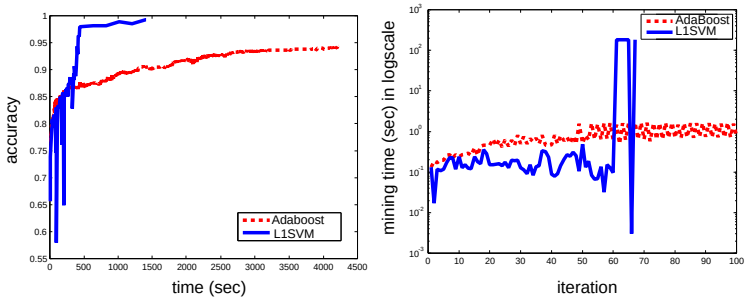

Fig. 5. (left) Evolution of Q2 for AdaBoost and L1SVM as a function of total time in seconds (total time = optimization time + mining time). (right) Mining time of AdaBoost and L1SVM at each iteration.

Figure 6 shows transition of percentage of support vectors and mining time for L1SVM and AdaBoost in details. As AdaBoost does not have sparsity on example weights, percentage of support vectors for AdaBoost is always 1.00. In contrast, percentage of support vectors for L1SVM keeps decreasing (right), which makes significant difference in mining time. In Figure 7, we can observe that mining is always more costly than optimization for AdaBoost (left). On the other hand, optimization is always more costly than mining for L1SVM except for last a few iterations. Notice that mining time for L1SVM is a magnitude shorter than that for AdaBoost, accounting for the effect of induced sparsity.

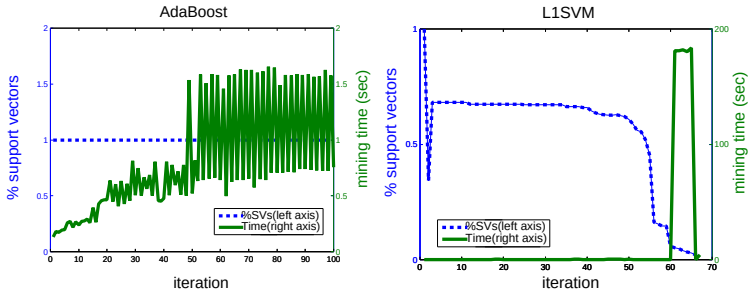

Fig. 6. Transition of percentage of SVs and mining time for AdaBoost (left) and L1SVM (right).

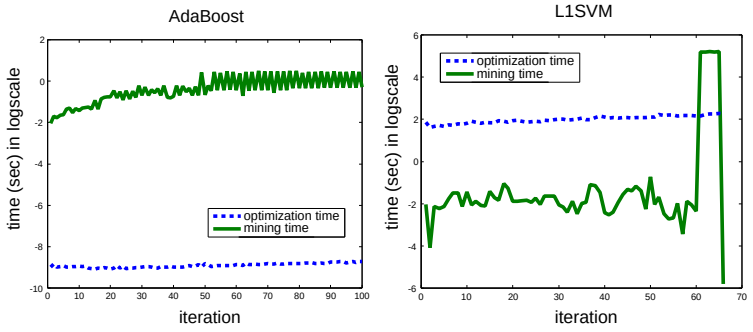

Fig. 7. Transition of optimization time and mining time for AdaBoost (left) L1SVM (right).

For regression we compare LASSO with LPR. Convergence of LASSO and LPR was checked using early stopping criterion $\sum_{i=1}^{n} x_{ij} u_i \leq 1 + \epsilon$, where ϵ was set to 0.05. Example weights for pattern mining at each iteration is shown in

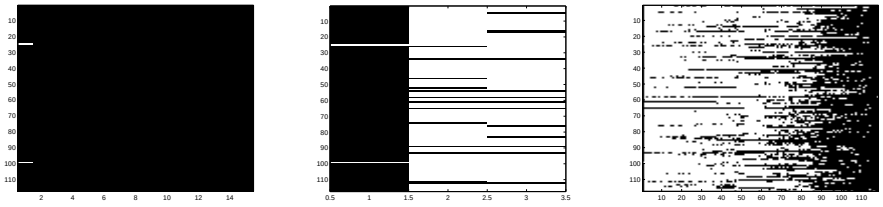

Fig. 8. Transition of example weights of LASSO (left), that of LPR ($C = 1000, \nu = 0.1$) (center) and that of LPR($C = 1000, \nu = 0.01$) (right). The vertical axis shows example ID, and the horizontal axis shows iterations. Nonzero weights are represented in black, and zero weights are represented in white. Example weights of LASSO are not sparse, in contrast to LPR.

Figure 8. As expected, example weights of LASSO is not sparse, in contrast to that of LPR. This makes difference in the size of search space and efficiency in pattern mining.

Table 3 shows the behavior of LPR when changing ν. We can observe that ν lowerbounds the number of support vectors. As we set ν larger, more examples (data points) become support vectors, and pattern mining becomes faster.

Table 3. Influence of the choice of ν parameter on LPR. Pat: the number of patterns with nonzero β, Itr: the number of iterations, ρ: the margin, Time: total time, ϵ: tube size automatic determined by ν, SVs: mean ratio of support vectors over the iterations, Q2: the regression Q2 in the training set. The tube size ϵ is recovered after solving the optimization problem [13].

ν	0.01	0.1	0.2	0.3	0.4
Itr	118	3	2	2	2
Pat	24	1	1	1	1
Time	2120	34.3	14.6	15.9	15.4
ϵ	7.23e-12	0.386	0.349	0.261	0.219
SVs	0.382	0.402	0.598	0.645	0.701
Q2	1.00	0.535	0.345	0.359	0.358

Figure 9 (left) shows the evolution of regression accuracy Q2 as a function of total time in seconds for LASSO and LPR. It clearly demonstrates faster learning of LPR compared with LASSO. $Q2$ of LPR is almost 1.0 around 700 seconds, but that of LASSO was still around 0.8, and was 0.85 after another 1800 seconds. Figure 9 (right) shows the mining time for LASSO and LPR at each iteration. This figure gives us an interesting observation; mining part of LASSO is slow, while that of LPR is fast and called many times. On average, mining time per iteration was 15.1 seconds for LPR, which was much faster than

LASSO that took 182 seconds on average. One interpretation of this observation is that LPR successfully split the original problem into many small parts, while LASSO tried to solve the hard original problem directly, which turned out to be more time consuming in this case.

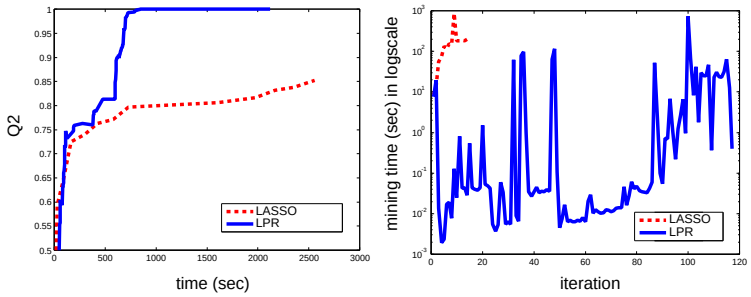

Fig. 9. (left) Evolution of Q2 for LASSO and LPR as a function of total time in seconds (total time = optimization time + mining time). (right) Mining time of LASSO and LPR at each iteration.

Figure 10 and Figure 11 give us more detailed information. Figure 10 shows percentage of support vectors for LASSO and LPR in details. As LASSO does not have sparsity on example weights, percentage of support vectors for LASSO is always 1.00 after the first iteration. In contrast, percentage of support vectors for LPR is less than 1.00 until last a few iterations, which makes significant difference in mining time. In Figure 11, we can observe that time used for mining is almost always shorter than time used for optimization in LPR, but vice verse in LASSO.

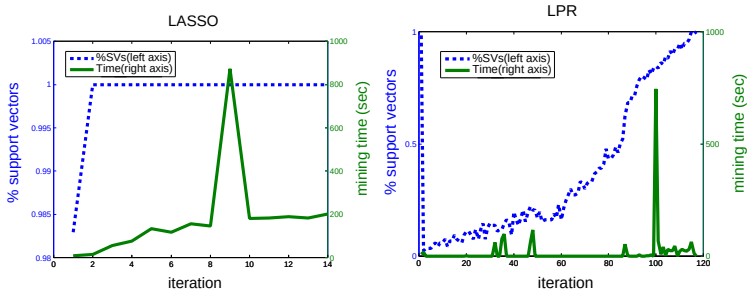

Fig. 10. Transition of % of SVs and mining time for LASSO (left) and LPR (right).

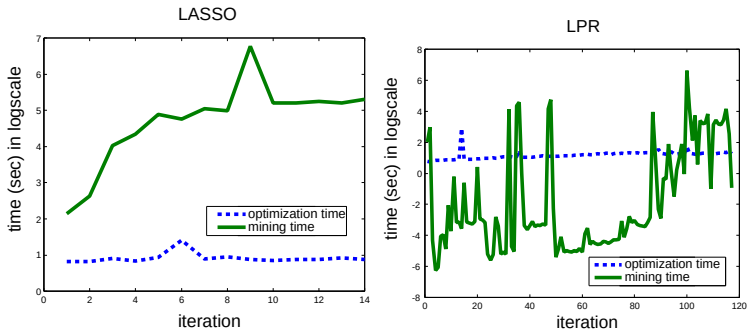

Fig. 11. Transition of optimization time and mining time for LASSO (left) LPR (right).

5 Conclusion

In this paper we proposed to use loss functions that induce sparsity on example weights for speeding-up discriminative pattern mining. We compared popular loss functions in classification and regression in terms of induced sparsity and resulting mining time. Computational experiments on real-world datasets showed that either exploiting sparsity or not makes large difference in pattern mining. It is worth noting that other iterative mining method for classification and regression can also benefit the claim of this paper and enjoy sparsity by carefully choosing loss functions. Resulting efficiency will be appreciated especially when mining problem is hard and time consuming such as the case of frequent graph mining. From an optimization point of view, one does not have to limit a loss function to convex one, but can employ, e.g., ramp loss function [3], which is not convex but induce improved sparsity, for solving large problems.

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In Proceedings of the 20th International Conference on Very Large Databases, pages 487–499, 1994.
2. 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.
3. R. Collobart, J. Weston, and L. Bottou. Trading convexity for scalability. In Proceedings of the 23rd International Conference on Machine Learning, pages 201–208, 2006.
4. A. Demiriz, K.P. Bennet, and J. Shawe-Taylor. Linear programming boosting via column generation. Machine Learning, 46(1-3):225–254, 2002.
5. Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 55(1):119–139, 1996.

6. C. Helma, T. Cramer, S. Kramer, and L.D. 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.
7. H. Kim, S. Kim, T. Weninger, J. Han, and T. Abdelzaher. Ndpmine: Efficiently mining discriminative numerical features for pattern-based classification. In European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), pages 35–50. Sprinter, 2010.
8. 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.
9. 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.
10. J. Pei, J. Han, B. Mortazavi-asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. IEEE Transactions on Knowledge and Data Engineering, 16(11):1424–1440, 2004.
11. G. Rätsch, S. Mika, B. Schölkopf, and K.-R. Müller. Constructing boosting algorithms from SVMs: an application to one-class classification. IEEE Trans. Patt. Anal. Mach. Intell., 24(9):1184–1199, 2002.
12. H. Saigo, S. Nowozin, T. Kadowaki, T. Kudo, and K. Tsuda. gBoost: A mathematical programming approach to graph classification and regression. Machine Learning, 75(1):69–89, 2009.
13. B. Schölkopf and A. J. Smola. Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, 2002.
14. I. Steinwart. Sparseness of support vector machines. Journal of Machine Learning Research, 4:1071–1105, 2003.
15. R. Tibshrani. Regression shrinkage and selection via the LASSO. J. Royal. Statist. Soc B., 58(1):267–288, 1996.
16. K. Tsuda and K. Kurihara. Graph mining with variational dirichlet process mixture models. In SIAM Conference on Data Mining (SDM), 2008.
17. T. Uno, M. Kiyomi, and H. Arimura. LCM ver.3: collaboration of array, bitmap and prefix tree for frequent itemset mining. In OSDM '05: Proceedings of the 1st international workshop on open source data mining, pages 77–86, 2005.
18. N. Wale and G. Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. In Proceedings of the 2006 IEEE International Conference on Data Mining, pages 678–689, 2006.
19. Y. Xiang, J. Ruoming, F. David, and F. F. Dragan. Succinct summarization of transactional databases: an overlapped hyperrectangle scheme. In KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 758–766, New York, NY, USA, 2008. ACM.
20. 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.
21. M. J. Zaki. Efficiently mining frequent trees in a forest: algorithms and applications. In IEEE Transactions on Knowledge and Data Engineering, pages 1021–1035, 2005.