Multicategory Incremental Proximal Support Vector Classifiers

Amund Tveit and Magnus Lie Hetland

Department of Computer and Information Science,
Norwegian University of Science and Technology,
N-7491 Trondheim, Norway
{amundt,mlh}@idi.ntnu.no

Abstract. Support Vector Machines (SVMs) are an efficient data mining approach for classification, clustering and time series analysis. In recent years, a tremendous growth in the amount of data gathered has changed the focus of SVM classifier algorithms from providing accurate results to enabling incremental (and decremental) learning with new data (or unlearning old data) without the need for computationally costly retraining with the old data. In this paper we propose an efficient algorithm for multicategory classification with the incremental proximal SVM introduced by Fung and Mangasarian.

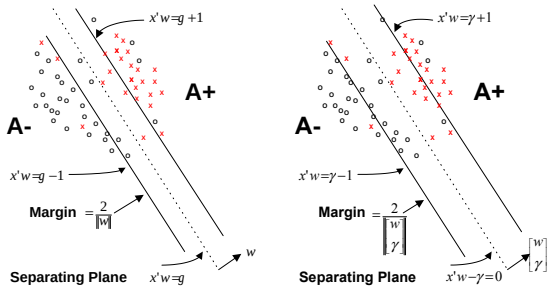
1 Introduction

Support Vector Machines (SVMs) are an efficient data mining approach for classification, clustering and time series analysis [1–3]. In recent years, a tremendous growth in the amount of data gathered (for example, in e-commerce and intrusion detection systems) has changed the focus of SVM classifier algorithms from providing accurate results to enabling incremental (and decremental) learning with new data (or unlearning old data) without the need for computationally costly retraining with the old data. Fung and Mangasarian [4] introduced the Incremental and Decremental Linear Proximal Support Vector Machine (PSVM) for binary classification and showed that it could be trained extremely efficiently, with one billion examples (500 increments of two million examples) in two hours and twenty-six minutes on relatively low-end hardware (400 MHz Pentium II).

In this paper we propose an efficient algorithm based on memoization, in order to support Multicategory Classification for the Incremental PSVM.

2 Background Theory

The standard binary SVM classification problem with soft margin (allowing some errors) is shown visually in Fig. 1(a) and as a constrained quadratic programming problem in (1). Intuitively, the problem is to maximize the margin between the solid planes and at the same time permit as few errors as possible, errors being positive class points on the negative side (of the solid line) or vice versa.



(a) Standard SVM classifier

(b) Proximal SVM classifier

Fig. 1. SVM and PSVM

$$\min_{(w,\gamma,y)\in\mathbb{R}^{n+1+m}} \{ve'y + \tfrac{1}{2}w'w\} \quad \text{s.t. } D(Aw - e\gamma) + y \geq e,\ y \geq 0 \tag{1}$$

$$A \in \mathbb{R}^{m\times n},\ D \in \{-1,+1\}^{m\times 1},\ e = 1^{m\times 1}$$

Fung and Mangasarian [5] replaced the inequality constraint in (1) with an equality constraint. This changed the binary classification problem, because the points in Fig. 1(b) are no longer bounded by the planes, but are clustered around them. By solving the equation for y and inserting the result into the expression to be minimized, one gets the following unconstrained optimization problem:

$$\min_{(w,\gamma)\in\mathbb{R}^{n+1+m}} f(w,\gamma) = \tfrac{\nu}{2}\|D(Aw - e\gamma) - e\|^2 + \tfrac{1}{2}(w'w + \gamma^2) \tag{2}$$

Setting $\nabla f = \left(\frac{\partial f}{\partial w}, \frac{\partial f}{\partial \gamma}\right) = \mathbf{0}$ one gets:

$$\underbrace{\begin{pmatrix} w \\ \gamma \end{pmatrix}}_{\mathcal{X}} = \begin{pmatrix} A'A + \frac{I}{\nu} & -A'e \\ -e'A & \frac{1}{\nu} + m \end{pmatrix}^{-1} \begin{pmatrix} A'De \\ -e'De \end{pmatrix} = \underbrace{\left(\frac{I}{\nu} + E'E\right)^{-1}}_{\mathcal{A}^{-1}} \underbrace{E'De}_{\mathcal{B}} \tag{3}$$

$$E = [A - e],\ E \in \mathbb{R}^{m\times(n+1)}$$

Fung and Mangasarian [6] later showed that (3) can be rewritten to handle increments (E^i, d^i) and decrements (E^d, d^d), as shown in (4). This decremental approach is based on time windows.

$$\mathcal{X} = \begin{pmatrix} w \\ \gamma \end{pmatrix}$$
$$= \left(\frac{I}{\nu} + E'E + (E^i)'E^i - (E^d)'E^d \right)^{-1} \left(E'd + (E^i)'d^i - (E^d)'d^d \right) \ , \quad (4)$$

where

$$d = De \ .$$

3 Incremental Proximal SVM for Multiple Classes

In the multicategorical classification case, the (incremental) class label vector d^i consists of m_i numeric labels in the range $\{0, \ldots, c-1\}$, where c is the number of classes, as shown in (5).

$$\mathcal{X} = \begin{pmatrix} w_0 \ \cdots \ w_{c-1} \\ \gamma_0 \ \cdots \ \gamma_{c-1} \end{pmatrix} = \mathcal{A}^{-1}\mathcal{B} \quad (5)$$

3.1 The Naive approach

In order to apply the proximal SVM classifier in a "one-against-the-rest" manner, the class labels must be transformed into vectors with $+1$ for the positive class and -1 for the rest of the classes, that is, $\Theta(cm_i)$ operations in total, and later $\Theta(cm_i n)$ for calculating $(E^i)'d$ for each class. The latter (column) vectors are collected in a matrix $\mathcal{B} \in \mathbb{R}^{(n+1)c}$. Because the training features represented by E^i are the same for all the classes, it is enough to calculate $\mathcal{A} \in \mathbb{R}^{(n+1)^2}$ once, giving $\Theta(m_i(n+1)^2 + (n+1)^2)$ operations for calculating $(E^i)'E^i$ and adding it to $\frac{I}{\nu} + E'E$. The specifics are shown in shown in Algorithm 1.

Theorem 1. *The running time complexity of Algorithm 1 is $\Theta(cm_{inc}n)$.*

Proof. The conditional statement in lines 3–7 takes $\Theta(1)$ time and is performed m_{inc} times (inner loop, lines 2–8) per iteration of *classId* (outer loop, line 1–10). Calculation of the matrix-vector product $\mathcal{B}[classId,]$ in line 9 takes $\Theta((n+1)m_{inc})$ per iteration of *classId*. This gives a total running time of

$$\Theta(c \cdot (m_{inc} + m_{inc}(n+1))) = \Theta(cm_{inc}n) \ .$$

□

Algorithm 1 *calcℬ_Naive*(E_{inc}, d_{inc})

Require: $E_{inc} \in \mathbb{R}^{m_{inc}x(n+1)}$, $d_{inc} \in \{0, \ldots, c-1\}^{m_{inc}}$ and $n, m_{inc} \in \mathbb{N}$
Ensure: $\mathcal{B} \in \mathbb{R}^{(n+1)xc}$
1: **for all** *classId* in $\{0, \ldots, c-1\}$ **do**
2: **for all** *idx* in $\{0, \ldots, m_{inc}-1\}$ **do**
3: **if** $d_{inc}[idx] = classId$ **then**
4: $d_{classId}[idx,] = +1$
5: **else**
6: $d_{classId}[idx,] = -1$
7: **end if**
8: **end for**
9: $\mathcal{B}[classId,] = E'_{inc} d_{classId}$
10: **end for**
11: return $\mathcal{B}$

3.2 The Memoization Approach

The $d_{classId}$ vectors, c in all, (in line 3 of Algorithm 1) are likely to be unbalanced, that is, have many more -1 values than $+1$ values. However, if there are more than two classes present in the increment d_i, the vectors will at least share one index position where the value is -1. With several classes present in the increment d_i, the matrix-vector products $(E^i)' d_{classId}$ actually perform duplicate calculations each time there exists two or more $d_{classId}$ vectors that have -1 values in the same position.

The basic idea for the memoization approach (Algorithm 3) is to only calculate the $+1$ positions for each vector $d_{classId}$ by first creating a vector $F = -[E^i_{.j}]$ (a vector with the negated sum of E's columns, equivalent to multiplying $E^{i'}$ with a vector filled with -1) and then to calculate the $d_{classId}$ vectors using F and only switching the -1 to a $+1$ by adding the row vector of E twice if the row in $d_{classId}$ is equal to $+1$. In order to do this efficiently, an index of d_i for each class ID has to be created (Algorithm 2).

Algorithm 2 *buildClassMap*(d_{inc})

Require: $d_{inc} \in \{0, \ldots, c-1\}^{m_{inc}}$ and $m_{inc} \in \mathbb{N}$
1: *classMap* = array of length c containing empty lists
2: **for all** $idx = 0$ to $m_{inc} - 1$ **do**
3: append *idx* to classMap$[d_{inc}[idx,]]$
4: **end for**
5: return *classMap*

Theorem 2. *The running time complexity of Algorithm 2 is* $\Theta(m_{inc})$.

Proof. Appending *idx* to a the tail of a linked list takes $\Theta(1)$ time, lookup of $classMap[d_{inc}[idx,]]$ in the directly addressable arrays *classMap* and d_{inc} also

takes $\Theta(1)$ time, giving a total for line 3 of $\Theta(1)$ time per iteration of *idx*. *idx* is iterated m_{inc} times, giving a total of $\Theta(m_{inc})$ time.

□

Algorithm 3 calc$\mathcal{B}$_Memo($E_{inc}, d_{inc}, E_{inc}{}' E_{inc}$)

Require: $E_{inc} \in \mathbb{R}^{m_{inc} \times (n+1)}$, $d_{inc} \in \{0, \ldots, c-1\}^{m_{inc}}$ and $n, m_{inc} \in \mathbb{N}$
Ensure: $\mathcal{B} \in \mathbb{R}^{(n+1)xc}$, $\mathcal{F} \in \mathbb{R}^{(n+1)}$
1: *classMap* = *buildClassMap*(d_{inc})
2: **for all** *classId* in $\{0, \ldots, c-1\}$ **do**
3: $\mathcal{B}[classId,] = E_{inc}{}' E_{inc}[n]$
4: **for all** *idx* in *classMap*[*classId*,] **do**
5: $\mathcal{B}[idx, classId,] = \mathcal{B}[idx, classId,] + 2 \cdot \sum_{j=0}^{n} E_{inc}[idx\,,\, j]$
6: **end for**
7: **end for**
8: return $\mathcal{B}$

Theorem 3. *The running time complexity of Algorithm 3 is* $\Theta(n(c + m_{inc})$.

Proof. Calculation of *classMap* (line 1) takes $\Theta(m_{inc})$ time (from Theorem 2). Line 3 takes $\Theta(n+1)$ time per iteration of $classId$, giving a total of $\Theta(c(n+1))$. Because *classMap* provides a complete indexing ($|\bigcup_{u=0}^{c-1} classMap[u]| = m_{inc}$) of the class labels in d_{inc}, and because there are no repeated occurrences of *idx* for different *classId*s ($\bigcap_{u=0}^{c-1} classMap[u] = \emptyset$), line 5 will run a total of m_{inc} times.

This gives a total running time of

$$\begin{aligned} &\Theta(m_{inc} + (n+1)m_{inc} + c(n+1) + m_{inc}) \\ &= \Theta(n(c + m_{inc})) \ . \end{aligned}$$

□

Corollary 1. *Algorithms 1 and 3 calculate the same* $\mathcal{B}$ *if provided with the same input.*

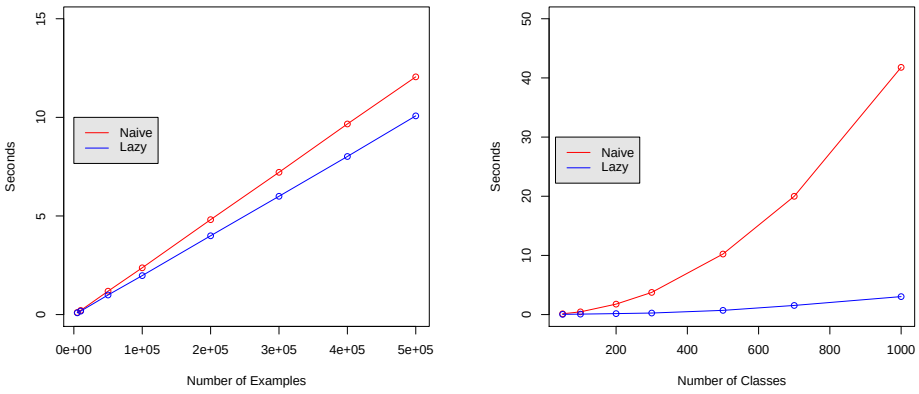
4 Empirical Results

In order to test and compare the computational performance of the incremental multicategory proximal SVMs with the naive and lazy algorithms, we have used three main types of data:

1. Forest cover type, 580012 training examples, 7 classes and 54 features (from UCI KDD Archive [7])
2. Synthetic datasets with a large number of classes (up to 1000 classes) and 30 features

3. Synthetic dataset with a large number of examples (10 million), 10 features and 10 classes

The results for the first two data sets are shown in Fig. 4; the average time from tenfold cross-validation is used. For the third data set, the average classifier training times were 18.62 s and 30.64 s with the lazy and naive algorithm, respectively (training time for 9 million examples, testing on 1 million).



(a) Runtime vs examples (cover type)

(b) Runtime vs classes (synthetic)

Fig. 2. Computational performance: training time

The incremental multicategory proximal SVM was been implemented in C++ using the CLapack and ATLAS libraries. The tests were run on an Athlon 1.53 GHz PC with 1 GB RAM running Red Hat Linux 2.4.18.

Acknowledgements

We would like to thank Professor Mihhail Matskin and Professor Arne Halaas. This work is supported by the Norwegian Research Council in the framework of the Distributed Information Technology Systems (DITS) program, and the ElComAg project.

5 Conclusion and Future Work

We have introduced the multiclass incremental proximal SVM and shown a computational improvement for training the multiclass incremental proximal SVM,

which works particularly well for classification problems with a large number of classes. Another contribution is the implementation of the system (available on request).

Future work includes applying the algorithm to demanding incremental classification problems, for example, web page prediction based on analysis of click streams or automatic text categorization. Algorithmic improvements that need to be done include (1) develop balancing mechanisms (in order to give hints for pivot elements to the applied linear system solver for reduction of numeric errors), (2) add support for decay coefficients for efficient decremental unlearning, (3) investigate the appropriateness of parallelized incremental proximal SVMs, (4) strengthen implementation with support for tuning set, kernels as well as one-against-one classifiers.

References

1. Burbidge, R., Buxton, B.F.: An introduction to support vector machines for data mining. In Sheppee, M., ed.: Keynote Papers, Young OR12, University of Nottingham, Operational Research Society, Operational Research Society (2001) 3–15
2. Huang, J., Shao, X., Wechsler, H.: Face pose discrimination using support vector machines (svm). In: Proceedings of 14th Int'l Conf. on Pattern Recognition (ICPR'98), IEEE (1998) 154–156
3. Muller, K.R., Smola, A.J., Ratsch, G., Scholkopf, B., Kohlmorgen, J., Vapnik, V.: Predicting time series with support vector machines. In: ICANN. (1997) 999–1004
4. Fung, G., Mangasarian, O.L.: Incremental support vector machine classification. In Grossman, R., Mannila, H., Motwani, R., eds.: Proceedings of the Second SIAM International Conference on Data Mining, SIAM (2002) 247–260
5. Fung, G., Mangasarian, O.L.: Multicategory Proximal Support Vector Classifiers. Submitted to Machine Learning Journal (2001)
6. Schwefel, H.P., Wegener, I., Weinert, K., eds.: 8. Natural Computing. In: Advances in Computational Intelligence: Theory and Practice. Springer-Verlag (2002)
7. Hettich, S., Bay, S.D.: The UCI KDD archive. `http://kdd.ics.uci.edu` (1999)