Parallelization of the Incremental Proximal Support Vector Machine Classifier using a Heap-based Tree Topology

Amund Tveit and Håvard Engum

Department of Computer and Information Science,
Norwegian University of Science and Technology,
N-7491 Trondheim, Norway
{amundt,havare}@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 two efficient parallelized algorithms based on heaps of processing nodes for classification with the incremental proximal SVM introduced by Fung and Mangasarian.

1 Introduction

Support Vector Machines (SVMs) is an exceptionally efficient data mining approach for classification, clustering and time series analysis [1–3]. This is primarily due to SVMs highly accurate results that are competitive with other data mining approaches, e.g. artificial neural networks (ANNs) and evolutionary algorithms (EAs). In recent years tremendous growth in the amount of data gathered (e.g. user clickstreams on the web, in e-commerce and in intrusion detection systems), has changed the focus of SVM classifier algorithms to not only provide accurate results, but to also enable online learning, i.e. incremental and decremental learning, in order to handle concept drift of classes [4, 5].

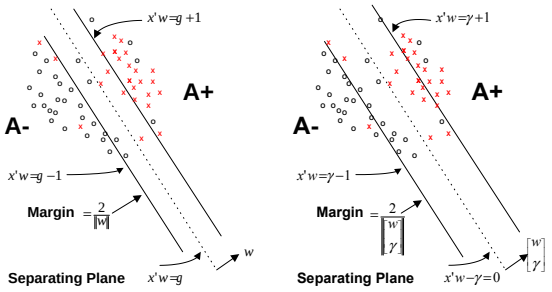
Fung and Mangasarian introduced the Incremental and Decremental Linear Proximal Support Vector Machine (PSVM) for binary classification [6], and showed that it was able to be trained extremely fast, i.e. with 1 billion examples (500 increments of 2 million) in 2 hours and 26 minutes on relatively low-end hardware (400 MHz Pentium II). This has later been extended to support efficient support of incremental multicategorical classification [7] and soft-decay decremental learning [8]. Proximal SVMs has also been shown to perform at a similar level of accuracy as regular SVMs and at the same time being significantly faster [9].

In this paper we propose and compare two parallelization approaches of the Incremental SVM classifier. The algorithms presented are based on heaps of CPUs represented as tree topologies.

2 Background Theory

The basic idea of Support Vector Machine classification is to find an optimal maximal margin separating hyperplane between two classes. Support Vector Machines uses an implicit nonlinear mapping from input-space to a higher dimensional feature-space using kernel-functions, in order to find a hyperplane of problems which are not linear separable in input-space [10, 11]. Classifying multiple classes is commonly performed by combining several binary SVM classifiers in a tournament manner, either one-against-all or one-against-one, the latter approach requiring substantial more computational effort [12].

The standard binary SVM classification problem with soft margin (allowing some errors) is shown visually in Fig. 1(a). 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

The standard SVM problem can be stated as a quadratic optimization problem with constraints, as shown in (1).

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

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

Fung and Mangasarian [13] 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:

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

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

Agarwal has showed that the Proximal SVM is directly transferable to a ridge regression expression [14]. 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.

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

where $d = De$

.

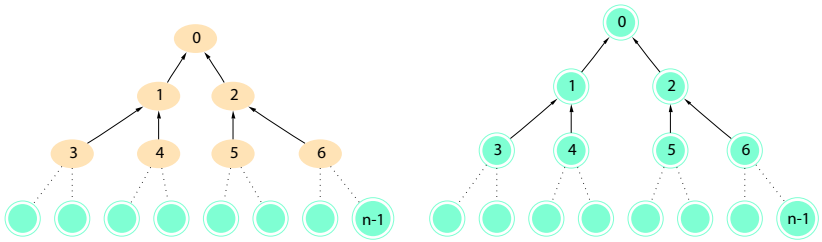
3 Parallelization of the Incremental Proximal SVM

The incremental capabilities of Proximal SVM allows us to efficiently calculate increments ($(E^i)'E^i$, $(E^i)'d^i$) in parallel.

The increments have the dimension numfeatures^2, compared to numfeatures·numexamples for E (feature dimension will also be limited due to the curse of dimensionality), in practice is numfeatures^2 << numfeatures · numexamples. The fact that these increments are relatively small makes them network-wise cheap to send to a parent node for accumulation into ($E'E$, $E'De$), before finally calculating $\mathcal{X}$ on the top node.

Most algorithms, both sequential and parallel, that involves a tree-based datastructure, perform computationally most efficiently when the tree is being balanced. Another wanted property is the capability of efficient calculation of adresses to child and parent nodes for each nodes in the tree. Heap-numbering of nodes solves both these issues.

In the first approach, as presented in figure 2(a) and algorithm 1, only the *leaf nodes* read increments of examples $((E^i)'E^i, (E^i)'d^i)$, in the second approach shown in figure 2(b) and algorithm 2 *all nodes* read examples.



(a) Read on leaf-nodes only (b) Read on all nodes

Fig. 2. Parallel heap-based tree topologies for Incremental PSVM

3.1 Description of trainParallelReadLeaf Algorithm

The basic flow for algorithm 1 is:

1. data is split (evenly distributed) between *the computational leaf nodes*
2. let the computational leaf nodes read training data from their respective storages.
3. calculate increments $((E^i)'E^i, (E^i)'d^i)$ on the leaf nodes before sending to parent nodes
4. parent nodes wait until they receive increment data from child node(s) (using the MPI receive method in the implementation)
5. parent nodes adds increments and send them to their respective parent nodes, repeated upward the tree
6. the top node finally gets the full incremental training data and is ready for classification
7. the top node (and intermediate nodes) are continously updated from increment data read at the leaf nodes

Algorithm 1 *trainParallelReadLeaf(thisNode, nNodes, incrementSize, nExamples)*

1: $topNode \Leftarrow 0$
2: $parentNode \Leftarrow ((thisNode + 1)/2) - 1$
3: $childNode1 \Leftarrow (thisNode + 1) * 2$
4: $childNode2 \Leftarrow (thisNode + 1) * 2 - 1$
5: $nIncrements \Leftarrow nExamples/incrementSize$
6: **for** $i = 1$ to $nIncrements$ **do**
7: **if** $childNode1 > nNodes - 1$ **then**
8: read $incrementSize$ training examples
9: calculate increment $((E^i)'E^i, (E^i)'d^i)$
10: **end if**
11: **if** $childNode1 \leq nNodes - 1$ **then**
12: (blocking) receive and accumulate increments $((E^i)'E^i, (E^i)'d^i)$ from childNode1
13: **end if**
14: **if** $childNode2 \leq nNodes - 1$ **then**
15: (blocking) receive and accumulate increments $((E^i)'E^i, (E^i)'d^i)$ from childNode2
16: **end if**
17: **if** $thisNode \neq topNode$ **then**
18: send (accumulated) increment $((E^i)'E^i, (E^i)'d^i)$ to $parentNode$
19: **else**
20: at the topnode, calculate $\mathcal{X}$ from total accumulated $E'E$, $E'd$
21: **end if**
22: **end for**

3.2 Description of trainParallelReadAll Algorithm

The basic flow for algorithm 2 is:

1. incoming data is split (evenly distributed) between *all the computational nodes*
2. all computational nodes read training data from their respective storages.
3. leaf nodes then calculates and sends increments to the parent nodes
4. non-leaf nodes calculates increments and wait for increments from their respective child nodes
5. non-leaf nodes adds increments and send them to their respective parent nodes, repeated upward the tree
6. the top node finally gets the full incremental training data (including its own) and is ready for classification
7. the nodes are continously updated from increment data read at all nodes and accumulated upward the tree

4 Empirical Results

Since the two algorithms proposed are exact parallalizations of the incremental proximal SVM, we have chosen to measure speedup (instead of accuracy) on

Algorithm 2 *trainParallelReadAll(thisNode, nNodes, incrementSize, nExamples)*

1: $topNode \Leftarrow 0$
2: $parentNode \Leftarrow ((thisNode + 1)/2) - 1$
3: $childNode1 \Leftarrow (thisNode + 1) * 2$
4: $childNode2 \Leftarrow (thisNode + 1) * 2 - 1$
5: $nIncrements \Leftarrow nExamples/incrementSize$
6: **for** $i = 1$ to $nIncrements$ **do**
7: read $incrementSize$ training examples
8: calculate increment E'E,E'De
9: **if** $childNode1 \leq nNodes - 1$ **then**
10: (blocking) receive and accumulate increments $((E^i)'E^i,\ (E^i)'d^i)$ from childNode1
11: **end if**
12: **if** $childNode2 \leq nNodes - 1$ **then**
13: (blocking) receive and accumulate increments $((E^i)'E^i,\ (E^i)'d^i)$ from childNode2
14: **end if**
15: **if** $thisNode \neq topNode$ **then**
16: send (accumulated) increment $((E^i)'E^i,\ (E^i)'d^i)$ to $parentNode$
17: **else**
18: calculate $\mathcal{X}$ from total accumulated $E'E$, $E'd$
19: **end if**
20: **end for**

various configurations (number of processing nodes and number of examples in increments). The example dataset is *Forest cover type*, 580012 training examples, 7 classes and 54 features (from UCI KDD Archive [15]). The results are based on average speedups from 10 runs of each configuration.

The parallel incremental proximal SVM has been implemented in C++ using the CLapack and ATLAS libraries for linear system solvers and MPI for communication between CPU nodes [16, 17]. The tests were run on a cluster with Athlon 1.46 GHz / 1 GB RAM nodes running Source Mage GNU/Linux (Linux kernel version 2.4.20).

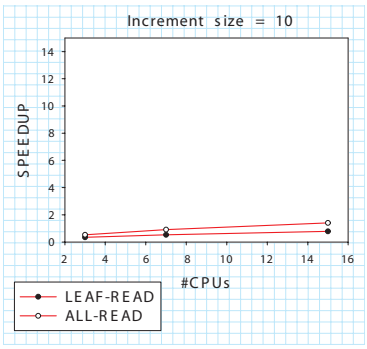
Acknowledgements

We would like to thank Professor Mihhail Matskin. This work is supported by the Norwegian Research Council.

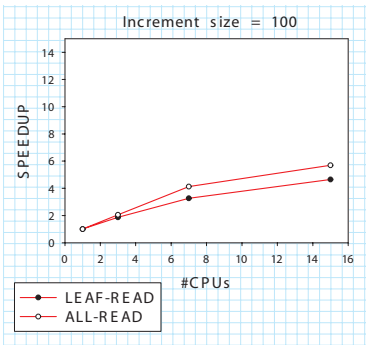
5 Discussion

In figure 3 and 4 the computational speedup of algorithm 1 and 2 compared to the sequential algorithm running on one processing node.

For a small increment size - 10 training examples per increment - the parallel version performs poorly, actually slower than the sequential version, the latter

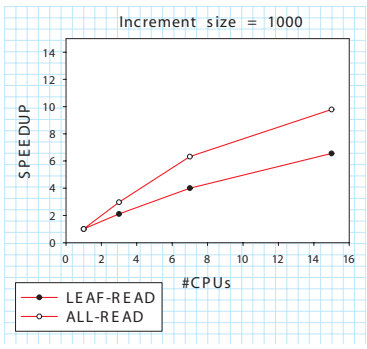


(a) increment size = 10

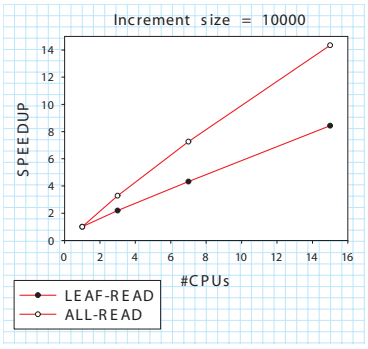


(b) increment size = 100

Fig. 3.



(a) increment size = 1000



(b) increment size = 10000

Fig. 4.

having the reference speedup value of 1. This is mainly caused by network IO (very high frequency of sending and receiving) and by disk IO (reading small increments, less than a disk block, is as time consuming as reading a full block).

Increasing the increment size to 100 improves the performance, and still there is still only little difference between algorithm 1 and 2, this is caused by that the computation is mainly network IO bound.

For an increment size of 1000 and 10000 the problem is becoming mainly disk IO bound, so letting *all* nodes read from disk in algorithm 2 significantly improves the performance compared to reading only with leaf-nodes in algorithm 1. With an increment size of 10000 algorithm 2 is close to being linearly scalable.

6 Conclusion and Future Work

We have introduced two algorithmic approaches for parallelizing the incremental proximal SVM and empirically shown that 1) they both have increasing speedup properties with increasing increment size, this is due to heavier processing per node and less network IO between nodes, and 2) reading on all nodes in the tree performs better than only reading at the leaf nodes.

Future work includes applying the approach on incremental classification and prediction problems. e.g. game usage mining [18]. Algorithmic improvements that needs to be done include 1) develop support for parallelized *decremental* PSVM, 2) add kernel support, 3) add incremental balancing mechanisms to improve accuracy for cases with many, potentially unbalanced, classes.

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. Ben-Hur, A., Horn, D., Siegelmann, H.T., Vapnik, V.: Support Vector Clustering. Journal of Machine Learning Research **2** (2001) 125–137
4. Ahmed, N., Liu, H., Sung, K.K.: Handling Concept Drifts in Incremental Learning with Support Vector Machines. In: Proceedings of the fifth International Conference on Knowledge Discovery and Data Mining, ACM Press (1999) 317–321
5. Klinkenberg, R., Joachims, T.: Detecting Concept Drift with Support Vector Machines. In Langley, P., ed.: Proceedings of the Seventeenth International Conference on Machine Learning (ICML), Morgan Kaufmann (2000)
6. 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
7. Tveit, A., Hetland, M.L.: Multicategory Incremental Proximal Support Vector Classifiers. In: Proceedings of the 7th International Conference on Knowledge-Based Information & Engineering Systems (KES'03, forthcoming). Lecture Notes in Artificial Intelligence, Springer-Verlag (2003)

8. Tveit, A., Hetland, M.L., Engum, H.: Incremental and Decremental Proximal Support Vector Classification using Decay Coefficients. In: Proceedings of the 5th International Conference on Data Warehousing and Knowledge Discovery (DaWaK'03, forthcoming). Lecture Notes in Artificial Intelligence, Springer-Verlag (2003)
9. Fung, G., Mangasarian, O.L.: Proximal support vector machine classifiers. In: Proceedings of the 7th ACM Conference on Knowledge Discovery and Data Mining, ACM (2001) 77–86
10. Christiani, N., Shawe-Taylor, J.: 6. In: An Introduction to Support Vector Machines and other kernel-based learning methods. 1st edn. Cambridge University Press (2000) 93–111
11. Vapnik, V.N.: 5. In: The Nature of Statistical Learning Theory. 2nd edn. Springer-Verlag (1999) 138–146
12. Hsu, C.W., Lin, C.J.: A Comparison of Methods for Multi-class Support Vector Machines. IEEE Transactions on Neural Networks **13** (2002) 415–425
13. Fung, G., Mangasarian, O.L.: Multicategory Proximal Support Vector Classifiers. Submitted to Machine Learning Journal (2001)
14. Agarwal, D.K.: Shrinkage Estimator Generalizations of Proximal Support Vector Machines. In: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM Press (2002) 173–182
15. Hettich, S., Bay, S.D.: The UCI KDD archive. `http://kdd.ics.uci.edu` (1999)
16. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK Users' Guide. Third edn. Society for Industrial and Applied Mathematics, Philadelphia, PA (1999)
17. Whaley, R.C., Petitet, A., Dongarra, J.J.: Automated Empirical Optimization of Software and the ATLAS Project". Parallel Computing **27** (2001) 3–25
18. Tveit, A., Tveit, G.B.: Game Usage Mining: Information Gathering for Knowledge Discovery in Massive Multiplayer Games. In: Proceedings of the International Conference on Internet Computing (IC'2002), session on Web Mining, CSREA Press (2002)