Incremental Support Vector Machine Construction

Carlotta Domeniconi Dimitrios Gunopulos
Computer Science Department
University of California
Riverside, CA 92521
{carlotta,dg}@cs.ucr.edu

Abstract

SVMs suffer from the problem of large memory requirement and CPU time when trained in batch mode on large data sets. We overcome these limitations, and at the same time make SVMs suitable for learning with data streams, by constructing incremental learning algorithms.

We first introduce and compare different incremental learning techniques, and show that they are capable of producing performance results similar to the batch algorithm, and in some cases superior condensation properties. We then consider the problem of training SVMs using stream data. Our objective is to maintain an updated representation of recent batches of data. We apply incremental schemes to the problem and show that their accuracy is comparable to the batch algorithm.

1. Introduction

Many applications that involve massive data sets are emerging. Examples are: telephone records, sales logs, multimedia data. When developing classifiers using learning methods, while a large number of training data can help reducing the generalization error, the learning process itself can get computationally intractable.

One would like to consider all training examples simultaneously, in order to accurately estimate the underlying class distributions. Howerer, these data sets are far too large to fit in main memory, and are typically stored in secondary storage devices, making their access particularly expensive. The fact that not all examples can be loaded into memory at once has two important consequences: the learning algorithm won't be able to see all data in one single batch, and is not allowed to "remember" too much of the data scanned in the past. As a consequence, scaling up classical learning algorithms to handle extremely large data sets and meet these requirements is an important research issue [15], [4].

One approach to satisfy these constraints is to consider incremental learning techniques, in which only a subset of the data is to be considered at each step of the learning process.

Support Vector Machines (SVMs) [17] have been successfully used as a classification tool in a variety of areas [9, 2, 12]. The solid theoretical foundations that have inspired SVMs convey desirable computational and learning theoretic properties to the SVM's learning algorithm. Another appealing feature of SVMs is the sparseness representation of the decision boundary they provide. The location of the separating hyperplane is specified via real-valued weights on the training examples. Those training examples that lie far away from the hyperplane do not participate in its specification and therefore receive zero weight. Only training examples that lie close to the decision boundary between the two classes (support vectors) receive non-zero weights.

Therefore, SVMs seem well suited to be trained according to an incremental learning fashion [16, 11]. In fact, since their design allows the number of support vectors to be small compared to the total number of training examples, they provide a compact representation of the data, to which new examples can be added as they become available.

New optimization approaches that specifically exploit the structure of the SVM have also been developed for scaling up the learning process. See [1, 14, 3].

2. Incremental Learning with SVMs

In order to make the SVM learning algorithm incremental, we can partition the data set in batches that fit into memory. Then, at each incremental step, the representation of the data seen so far is given by the set of support vectors describing the learned decision boundary (along with the corresponding weights). Such support vectors are incorporated with the new incoming batch of data to provide the training data for the next step. Since the design of SVMs allows the number of support vectors to be small compared to the total

number of training examples, this scheme should provide a compact representation of the data set.

It is reasonable to expect that the model incrementally built won't be too far from the model built with the complete data set at once (batch mode). This is because, at each incremental step, the SVM remembers the essential class boundary information regarding the seen data, and this information contributes properly to generate the classifier at the successive iteration.

Once a new batch of data is loaded into memory, there are different possibilities for the updating of the current model. Here we explore four different techniques. For all the techniques, at each step only the learned model from the previously seen data (preserved in form of support vectors) is kept in memory.

Error-driven technique (ED). This technique is a variation of the method introduced in [11], in which both a percentage of the misclassified and correctly classified data is retained for incremental training. The Error-driven technique, instead, keeps only the misclassified data. Given the model SVM_t at time t, new data are loaded into memory and classified using SVM_t. If the data is misclassified, it is kept, otherwise it is discarded. Once a given number n_e of misclassified data is collected, the update of SVM_t takes place: the support vectors of SVM_t, together with the n_e misclassified points, are used as training data to obtain the new model SVM_{t+1}.

Fixed-partition technique (FP). This technique has been previously introduced in [16]. The training data set is partitioned in batches of fixed size. When a new batch of data is loaded into memory, it is added to the current set of support vectors; the resulting set gives the training set used to train the new model. The support vectors obtained from this process are the new representation of the data seen so far, and they are kept in memory.

Exceeding-margin technique (EM). Given the model SVM_t at time t, new data $\{(\mathbf{x}_i, y_i)\}$ are loaded into memory. The algorithm checks if $(\mathbf{x}_i, y_i)$ exceeds the margin defined by SVM_t, i.e. if $y_i f_t((\mathbf{x}_i)) \leq 1$. If the condition is satisfied the point is kept, otherwise it is discarded. Once a given number n_e of data exceeding the margin is collected, the update of SVM_t takes place: the support vectors of SVM_t, together with the n_e points, are used as training data to obtain the new model SVM_{t+1}.

Exceeding-margin+errors technique (EM+E). Given the model SVM_t at time t, new data $\{(\mathbf{x}_i, y_i)\}$ are loaded into memory. The algorithm checks if $(\mathbf{x}_i, y_i)$ exceeds the margin defined by SVM_t, i.e. if $y_i f_t((\mathbf{x}_i)) \leq 1$. If the condition is satisfied the point is kept, otherwise it is classified using SVM_t: if misclassified it is kept, otherwise discarded. Once a given number n_e of data, either exceeding the margin or misclassified, is collected, the update of SVM_t takes place: the support vectors of SVM_t, together with the n_e points, are used as training data to obtain the new model SVM_{t+1}.

3. Training SVMs using Data Streams

We consider here the scenario in which the example generation is time dependent, and follow the data stream model presented in [8], also used in [7], [6], [4]. A data stream is a sequence of items that can be seen only once, and in the same order it is generated.

We seek algorithms for classification that maintain an updated representation of recent batches of data. The algorithm therefore must maintain an accurate representation of a *window* of recent data [6]. This model is useful in practice because the characteristics of the data may change with time, and so old examples may not be a good predictor for future points. The algorithm must perform only one pass over the stream data, and use a workspace that is smaller than the size of the input.

The incremental learning techniques we discussed are capable of achieving these objectives. Our approach is similar to [5], and works as follows: We consider the incoming data in batches of a given size b, and maintain in memory w models representative of the last $1, 2, \ldots, w$ batches. Thus, the window size is $W = wb$ examples. The w models are trained incrementally as data becomes available. Let us call the models, at time t, SVM_1^t, SVM_2^t, $\ldots SVM_w^t$ respectively. When a new batch of data comes in, at step $t+1$, SVM_w^t is discarded, the remaining SVM_1^t, $\ldots$, SVM_{w-1}^t are incrementally updated to take into account the new batch of data, producing $SVM_2^{t+1}, \ldots, SVM_w^{t+1}$ respectively. SVM_1^{t+1} is generated using the new batch of data only. At each step t, SVM_w^t gives the in-memory representation of the current distribution of data, and it is used to predict the class label of new data. Any of the discussed techniques can be employed for the incremental updates.

Besides the w SVM models, only b data points need to reside in memory at once. Both b and w can be set according to domain knowledge regarding locality properties of data distributions over time.

4. Experimental Evaluation

We compare the four incremental techniques and the SVM learning algorithm in batch mode, to verify their performances and sizes of resulting classifiers, i.e. number of resulting support vectors. We have tested the techniques on both simulated and real data. The real dataset (Pima) is taken from UCI Machine Learning Repository at http://www.cs.uci.edu/ ∼mlearn/MLRepository.html. We used, for both the incremental and batch algorithms, radial basis function kernels. We used SVM^{light} [10], and set

the value of γ in $K(\mathbf{x}_i, \mathbf{x}) = e^{-\gamma||\mathbf{x}_i - \mathbf{x}||^2}$ equal to the optimal one determined via cross-validation. Also the value of C for the soft-margin classifier is optimized via cross-validation. For the incremental techniques we have tested different batch sizes and n_e values. In Tables 1- 2 we report the best performances obtained (B is for the batch algorithm). We also report, besides the average classification error rates and standard deviations, the number of support vectors of the resulting classifier, the corresponding size of the condensed set (%), and the number of training cycles the SVM underwent.

To test the incremental techniques with stream data, we have used the Noisy-crossed-norm dataset (generated as the Large-noisy-crossed-norm dataset described below), and generated streams in batches of size $b = 1000$, and set $w = 3$. We have employed the Fixed-partition technique for the incremental updates. At each incremental step, we have tested the performance of the current model using 10 independent test sets of size 1000. We report average classification error rates and classifier sizes over successive steps. For comparison, we have also trained a SVM in batch mode over $w = 3$ consecutive batches of data over time, and report average classification error rates obtained at each step.

The Problems: *Large-noisy-crossed-norm data*. This data set consists of $n = 20$ attributes and $J = 2$ classes. Each class is drawn from a multivariate normal distribution with unit covariance matrix. One class has mean $2/\sqrt{20}$ along each dimension, and the other has mean $-2/\sqrt{20}$ along each dimension. We have generated 200,000 data points, and performed 5-fold cross-validation with 100,000 training data and 100,000 testing data. Table 1 shows the results. The last column lists the running times (in hours). Experiments were conducted on a 1.3 GHz machine with 1GB of RAM. *Pima Indians Diabete data*. This data set consists of $n = 8$ attributes, $J = 2$ classes, and $l = 768$ instances. Results are shown in Table 2. We performed 10-fold cross-validation with 568 training data and 200 testing data.

Results: Tables 1-2 show that, for both the data sets we have tested, the performance obtained with the incremental techniques comes close to the performance given by the batch algorithm. Moreover, for each problem considered, more than one incremental scheme provides a much smaller condensed set. In particular, it is quite remarkable the condensation power (1.5%) that the Exceed-margin technique shows for the Large-noisy-crossed-norm, while still performing close to the batch algorithm. The fact that the classifier is kept smaller allows for a much faster computation (30 minutes). The results obtained with the Pima data are also of interest. All four incremental techniques perform better than the batch algorithm and, at the same time, compute a smaller condensed set.

In Figure 1, we plot the results obtained with the stream data for 12 time steps. The average estimator size for the incremental and batch techniques, respectively, are 418 and 430. Since the data distribution is stationary, the performance and estimator size remain stable over time. We observe that the incremental technique employed (Fixed-partition) and the batch mode algorithm basically provide the same results, both in terms of performance and size of the model. These results provide clear evidence that, although the incremental techniques allow loss of information, they are capable of achieving accuracy results similar to the batch algorithm, while significantly improving training time.

Table 1. Results for Large-noisy-crossed-norm data.

	B	ED	FP	EM	EM+E
error (%)	3.2	9.1	3.2	4.5	6.7
std dev	0.18	0.05	0.001	0.02	0.02
#SVs	8321	4172	8452	1455	5308
Cond. set (%)	8.3	4.2	8.5	1.5	5.3
cycles	-	19	201	37	48
batch size	-	500	500	500	500
time	14	17	20	0.5	22

Table 2. Results for Pima data.

	B	ED	FP	EM	EM+E
error (%)	31.9	29.3	26.2	27.1	26.4
std dev	0.47	0.02	0.02	0.02	0.02
#SVs	547	291	405	394	399
Cond. set (%)	96	51.2	71.3	69.4	70.2
cycles	-	13	38	34	36
batch size	-	10	10	10	10

5. Related Work

The incremental techniques discussed here can be viewed as approximations of the *chunking* technique employed to train SVMs [13]. The chunking technique is an exact decomposition method that iterates through the training set to select the support vectors.

The incremental methods introduced here, instead, scan the training data only once, and, once discarded, data are not considered anymore. This property makes the methods suited to be employed within the data stream model also. Furthermore, the experiments we have performed show that,

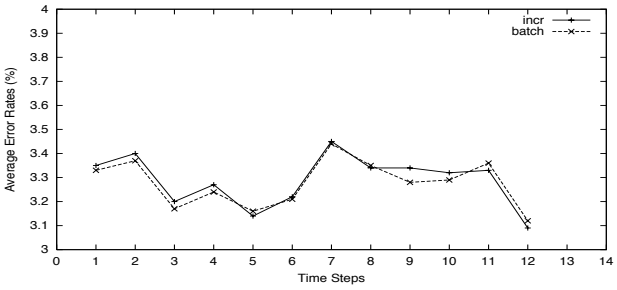

Figure 1. Noisy-crossed-norm data: Average Error Rates of Fixed-partition and batch algorithms for consecutive time steps.

although the incremental techniques allow loss of information, they are capable of achieving performance results similar to the batch algorithm.

6 Conclusions

We have introduced and compared new and existing incremental techniques for constructing SVMs. The experimental results presented show that incremental techniques are capable of achieving performance results similar to the batch algorithm, while improving the training time. We extended these approaches to work with stream data, and presented experimental results to show the efficiency and accuracy of the method.

Acknowledgments

This research has been supported by the National Science Foundation under grants NSF CAREER Award 9984729 and NSF IIS-9907477, by the US Department of Defense, and a research award from AT&T.

References

[1] J. C. Bennett, C. Campbel, "Support Vector Machines: Hype or Hallelujah?", *SIGKDD Explorations*, Vol. 2, No. 2, 1-13, 2000.

[2] M. Brown, W. Grundy, D. Lin, N. Cristianini, C. Sugnet, T. Furey, M. Ares, and D. Haussler, "Knowledge-based analysis of microarray gene expressions data using support vector machines," Tech. Report, University of California in Santa Cruz, 1999.

[3] G. Cauwenberghs and T. Poggio, "Incremental and Decremental Support Vector Machine Learning", *Advances in Neural Information Processing Systems*, 2000.

[4] Pedro Domingos, Geoff Hulten, "Mining high-speed data streams." *SIGKDD 2000*: 71-80, Boston, MA.

[5] Venkatesh Ganti, Johannes Gehrke, Raghu Ramakrishnan. "DEMON: Mining and Monitoring Evolving Data.", in *ICDE 2000*: 439-448, San Diego, CA.

[6] Sudipto Guha and Nick Koudas. "Data-Streams and Histograms.", In *Proc. STOC 2001*.

[7] S. Guha, N. Mishra, R. Motwani, L. O'Callaghan, "Clustering Data Stream", *IEEE Foundations of Computer Science*, 2000.

[8] M. R. Henzinger, P. Raghavan, and S. Rajagopalan, "Computing on data streams", *SRC Technical Note 1998-011*, Digital Research Center, May 26, 1998.

[9] T. Joachims, "Text categorization with support vector machines", *Proc. of European Conference on Machine Learning*, 1998.

[10] T. Joachims, "Making large-scale SVM learning practical" *Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf and C. Burger and A. Smola (ed.), MIT-Press, 1999. http://www-ai.cs.uni-dortmund.de/thorsten/svm_light.html

[11] P. Mitra, C. A. Murthy, and S. K. Pal, "Data Condensation in Large Databases by Incremental Learning with Support Vector Machines", *International Conference on Pattern Recognition*, 2000.

[12] E. Osuna, R. Freund, and F. Girosi, "Training support vector machines: An application to face detection", *Proc. of Computer Vision and Pattern Recognition*, 1997.

[13] E. Osuna, R. Freund, and F. Girosi, "An improved training algorithm for support vector machines", *Proceedings of IEEE NNSP'97*, 1997.

[14] J. C. Platt, "Fast Training of Support Vector Machines using Sequential Minimal Optimization", Advances in Kernel Methods, B. Schölf, C. J. C. Burges, and A. J. Smola (eds.), MIT Press, 185-208, 1999.

[15] F. J. Provost and V. Kolluri, "A survey of methods for scaling up inductive learning algorithms", *Technical Report ISL-97-3*, Intelligent Systems Lab., Department of Computer Science, University of Pittsburgh, 1997.

[16] N. A. Syed, H. Liu, and K. K. Sung, "Incremental Learning with Support Vector Machines", *International Joint Conference on Artificial Intelligence (IJCAI)*, 1999.

[17] V. Vapnik, *Statistical Learning Theory*. Wiley, 1998.