Incremental Support Vector Machine Construction

Carlotta Domeniconi Dimitrios Gunopulos
Computer Science Department
University of California
Riverside, CA 92521
fax: 909-787-4643
{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. The training process, in fact, involves the solution of a quadratic programming problem. We overcome these limitations, and at the same time make SVMs suitable for learning with data streams, by constructing incremental learning algorithms.

Incremental learning approaches are needed in the following scenarios: (1) the example generation itself is time-dependent (data streams), e.g. time series data, networking data; (2) new data are obtained in chunks at interval, e.g. scientific research data; (3) the training data set is too large to fit in main memory.

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: customer click streams, telephone records, sales logs, large sets of web pages, multimedia data, social/economic/ financial 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. Hoverer, 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 [12], [2].

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. The learner is therefore incrementally trained as new batches of data are loaded into memory.

Support Vector Machines (SVMs) [14, 15] have been successfully used as a classification tool in a variety of areas [7, 1, 10], and the maximum margin boundary they provide has been proved to be optimal in a structural risk minimization sense. 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 [13, 9]. 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.

1.1 Our Contribution

In this paper we focus on classification problems and present different incremental learning techniques using SVMs.

- We present and experimentally evaluate new and existing incremental techniques for constructing SVMs. Our experiments indicate that incremental SVM construction is efficient and can achieve similar accuracy with the batch construction algorithm, for large datasets.
- We consider the problem of constructing SVM classifiers for streaming data. We concentrate on the *window model*, where the classifier has to maintain the representation of the most recent data. We apply the incremental techniques on data streams, and show that, although they allow loss of information, their performance comes very close to the batch algorithm in this setting.

The paper is organized as follows. In the following section we introduce the main concepts of SVMs, and in section 3 we briefly introduce the learning algorithm for SVMs. Section 4 introduces different incremental learning algorithms, and in section 5 we use them to deal with stream data. Section 6 describes the experimental results. Section 7 is a discussion of related work, and a concluding summary is given is section 8.

2 Support Vector Machines

In this section we introduce the main concepts and properties of SVMs. We are given l observations. Each observation consists of a pair: a vector $\mathbf{x}_i \in \Re^n$, $i = 1, \ldots, l$, and the associated class label y_i. It is assumed that there exists some unknown probability distribution $P(\mathbf{x}, y)$ from which these data are drawn. The task it to learn the set of parameters α in $f(\mathbf{x}, \alpha)$ so that f realizes the mapping $\mathbf{x}_i \rightarrow y_i$. A particular choice of α defines the corresponding trained machine $f(\mathbf{x}, \alpha)$.

The expectation of the test error for a trained machine is

$$R(\alpha) = \int \frac{1}{2}|y - f(\mathbf{x}, \alpha)| dP(\mathbf{x}, y).$$

The quantity $R(\alpha)$ is called the expected risk, or just the risk. It gives a nice way of writing the true mean error, but unless we have an estimate of what $P(\mathbf{x}, y)$ is, it is not very useful. The empirical risk $R_{emp}(\alpha)$ is then defined, as the mean error rate measured over the training set:

$$R_{emp}(\alpha) = \frac{1}{2l} \sum_{i=1}^{l} |y_i - f(\mathbf{x}_i, \alpha)|.$$

The following bound holds [14]:

$$R(\alpha) \leq R_{emp}(\alpha) + C(h),$$

where h is the Vapnik Chervonenkis (VC) dimension, and is a measure of the ability of the machine to learn any training set without error. The term $C(h)$ is called the VC confidence. Given a family of functions $f(\mathbf{x}, \alpha)$, it is desirable to choose the machine which gives the lowest upper bound on the risk. The first term, $R_{emp}(\alpha)$, represents the accuracy attained on a particular training set, whereas the second term $C(h)$ represents the ability of the machine to learn any training set without error. $R_{emp}(\alpha)$ and $C(h)$ respectively drive the bias and the variance of the generalization error. The best generalization error is achieved when the right balance between these two terms is attained. This gives a principled method for choosing a learning machine for a specific task, and is the essential idea of structural risk minimization.

Unlike traditional methods which minimize the empirical risk, a support vector machine aims at minimizing the above upper bound of the generalization error. It achieves this goal by learning the αs in $f(\mathbf{x}, \alpha)$ so that the resulting trained machine satisfies the maximum margin property, i.e. the decision boundary it represents has the maximum minimum distance from the closest training point.

The well developed theory that has motivated SVMs makes them an attractive learning machine for a variety of tasks. A SVM maps the data into a higher-dimensional space (feature space) and defines a separating hyperplane there. Translating the training set into a higher-dimensional space incurs both computational and learning-theoretic costs. SVMs avoid overfitting by choosing a particular hyperplane among the many that can separate the data in the feature space, specifically the maximum margin hyperplane.

The computational burden of explicitly representing the feature vectors is avoided by defining a function, called the kernel function, that plays the role of the dot product in feature

space. Therefore, a SVM can locate a separating hyperplane in feature space and classify points in that space without ever representing the space explicitly.

By choosing different functions as kernels, SVMs can realize Radial Basis Function (RBF), Polynomial and Multi-layer Perceptron classifiers. Compared with the traditional way of implementing such classifiers, SVMs have the advantage of automatically selecting both the optimal number and locations of the kernel function during training.

In addition to avoid overfitting, the use of the maximum margin hyperplane leads to a learning algorithm that can be reduced to a convex optimization problem. In order to train the SVM, the unique minimum of a convex function must be found. As a consequence, support vector machines do not suffer from the local minima problem that affects many learning schemes and, unlike the backpropagation learning algorithm for neural networks, a given SVM will always deterministically converge to the same solution for a given data set, regardless of the initial conditions.

Another appealing feature of SVMs is the sparseness representation of the decision boundary they provide. The location of the separating hyperplane in feature space is specified via real-values 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 receive non-zero weights. These training examples are called support vectors, since their removal would change the location of the separating hyperplane. The design of SVMs, in general, allows the number of support vectors to be small compared to the total number of training examples. This property allows the SVM to classify new examples efficiently, since the majority of the training examples will be safely ignored.

3 Learning with Support Vector Machines

In the simple case of two linearly separable classes, a support vector machine selects, among the infinite number of linear classifiers that separate the data, the classifier that minimizes an upper bound on the generalization error. The SVM achieves this goal by computing the classifier that satisfies the maximum margin property, i.e. the classifier whose decision boundary has the maximum minimum distance from the closest training point.

If the two classes are non-separable, the SVM looks for the hyperplane that maximizes the margin and that, at the same time, minimizes a quantity proportional to the number of misclassification errors. The trade-off between margin and misclassification error is driven by a positive constant C that has to be chosen beforehand. The corresponding decision function in then obtained by considering the $sign(f(\mathbf{x}))$, where $f(\mathbf{x}) = \sum_i \alpha_i y_i \mathbf{x}_i^T \cdot \mathbf{x} - b$, and the coefficients α_i are the solution of a convex quadratic problem, defined over the hypercube $[0, C]^l$. In general, the solution will have a number of coefficients α_i equal to zero, and since there is a coefficient α_i associated to each data point, only the data points corresponding to non-zero α_i will influence the solution. These points are the support vectors, i.e. the points that lie closest to the separating hyperplane. Intuitively, the support vectors are the data points that lie at the border between the two classes, and a small number of support vectors indicates that the two classes can be well separated.

This technique can be extended to allow for non-linear decision surfaces. This is done by mapping the input vectors into a higher dimensional feature space: $\phi : \Re^n \rightarrow \Re^N$, and by formulating the linear classification problem in the feature space. Therefore, $f(\mathbf{x})$ can be expressed as $f(\mathbf{x}) = \sum_i \alpha_i y_i \phi^T(\mathbf{x}_i) \cdot \phi(\mathbf{x}) - b$.

If one were given a function $K(\mathbf{x}, \mathbf{y}) = \phi^T(\mathbf{x}) \cdot \phi(\mathbf{y})$, one could learn and use the maximum margin hyperplane in feature space without having to compute explicitly the image of points in $\Re^N$. It has been proved (Mercer's Theorem) that for each continuous positive definite function $K(\mathbf{x}, \mathbf{y})$ there exists a mapping ϕ such that $K(\mathbf{x}, \mathbf{y}) = \phi^T(\mathbf{x}) \cdot \phi(\mathbf{y})$, $\forall \mathbf{x}, \mathbf{y} \in \Re^n$. By making use of such function K (*kernel function*), the equation for $f(\mathbf{x})$ can be rewritten as

$$f(\mathbf{x}) = \sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) - b. \tag{1}$$

4 Incremental Learning with Support Vector Machines

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:

- Error-driven;
- Fixed-partition;
- Exceeding-margin;
- Exceeding-margin+errors.

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.

4.1 Error-driven technique

This technique is a variation of the method introduced in [9], 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}.

4.2 Fixed-partition technique

This technique has been previously introduced in [13]. 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.

4.3 Exceeding-margin technique

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}.

4.4 Exceeding-margin+errors technique

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}.

5 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 [6], also used in [5], [4], [2]. A data stream is a sequence of items that can be seen only once, and in the same order it is generated. Networks produce increasing quantities of data in the form of data streams. These large volumes of data usually reside on secondary and tertiary storage, making multiple passes prohibitive.

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 ([4]). 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 [3], 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, ..., w batches. Thus, the window size is $W = wb$ example. The w models are trained incrementally as data becomes available. Let us call the models, at time t, SVM_1^t, SVM_2^t, ... 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, ..., SVM_{w-1}^t are incrementally updated to take into account the new batch of data, producing SVM_2^{t+1}, ..., 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. Another possibility is allowing the values of b and w to adapt over time, according to increases or decreases in prediction performances. Adaptive values of b and w should allow for an increased locality (i.e., smaller b and w values) when a change of distribution occurs over time. We intend to further explore this issue in our future work.

6 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 data (OQ, Breast, Pima) are all 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} [8], 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- 7 we report the best performances

obtained. 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 three different simulated data (Ringnorm, Twonorm, Noisy-crossed-norm), 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 tests of size 1000. We report average classification error rates and classifier sizes over successive steps in section 6.1.1. 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.

6.1 The Problems

1. **Ringnorm data**. This data set consists of $n = 20$ attributes and $J = 2$ classes. Data for one class are generated from a multivariate normal distribution with zero mean and covariance matrix equal four times the identity matrix. Data for the other class are generated from a normal distribution with unit covariance matrix and mean equal $1/\sqrt{20}$ along each dimension. Average results obtained over 10 independent training and testing sets of size 2000 each are shown in Table 1.

2. **Twonorm 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. Average results obtained over 10 independent training and testing sets of size 2000 each are shown in Table 2.

3. **Noisy-crossed-norm data**. The data set consists of $n = 10$ input features and $J = 2$ classes. Each class contains two spherical bivariate normal subclasses, having standard deviation 1. The mean vectors for one class are $(-3/4, -3)$ and $(3/4, 3)$; whereas for the other class are $(3, -3)$ and $(-3, 3)$. The remaining eight predictors have independent standard Gaussian distributions. They serve as noise. For each class, data are evenly drawn from each of the two normal subclasses. Table 3 shows the results for this problem. We generated 20000

data points, and performed 10-fold cross-validation with 10000 training data and 10000 testing data.

3. **Large-noisy-crossed-norm data.** The data for this problem are generated as in the previous example, but the training and test sets have larger sizes. We have generated 200,000 data points, and performed 5-fold cross-validation with 100,000 training data and 100,000 testing data. Table 4 shows the results for this problem. The last column lists the running times (in hours). Experiments were conducted on a 1.3 GHz machine with 1GB of RAM.

4. **OQ data.** This data set consists of $n = 16$ numerical attributes and $J = 2$ classes. The objective is to identify black-and-white rectangular pixel displays as one of the two capital letters "O" and "Q" in the English alphabet. There are $l = 1536$ instances in this data set. The character images were based on 20 different fonts, and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15. Results obtained over 10 independent runs are shown in Table 5. We performed 10-fold cross-validation with 1336 training data and 200 testing data.

5. **Wisconsin breast cancer data.** The data set consists of $n = 9$ medical input features that are used to make a binary decision on the medical condition: determining whether the cancer is malignant or benign. The data set contains 683 examples after removing missing values. Results for this problem are shown in Table 6. We performed 10-fold cross-validation with 483 training data and 200 testing data.

6. **Pima Indians Diabete data.** This data set consists of $n = 8$ numerical medical attributes and $J = 2$ classes (tested positive or negative for diabetes). There are $l = 768$ instances. Results are shown in Table 7. We again performed 10-fold cross-validation with 568 training data and 200 testing data.

6.1.1 Results

Tables 1-7 show that, for all the data sets we have tested, the performance obtained with the incremental techniques comes close to the performance given by the batch algorithm.

Table 1: Results for Ringnorm data.

	error (%)	std dev	#SVs	Cond. set (%)	cycles	batch size
BATCH	0.02	0.0003	52	2.6	-	-
ERROR-DRIVEN	0.02	0.0003	42	2.1	2	100
FIXED PART.	0.02	0.0003	47	2.4	10	100
EXCEED-MARGIN	0.07	0.0006	35	1.8	1	10
EXCEED-MARGIN+ERRORS	0.06	0.0005	30	1.5	2	10

Table 2: Results for Twonorm data.

	error (%)	std dev	#SVs	Cond. set (%)	cycles	batch size
BATCH	2.4	0.003	200	10	-	-
ERROR-DRIVEN	2.9	0.004	192	9.6	7	10
FIXED PART.	2.5	0.003	186	9.3	10	200
EXCEED-MARGIN	3.2	0.01	254	12.7	54	10
EXCEED-MARGIN+ERRORS	2.7	0.003	267	13.4	52	10

Table 3: Results for Noisy-crossed-norm data.

	error (%)	std dev	#SVs	Cond. set (%)	cycles	batch size
BATCH	3.1	0.17	1043	10.4	-	-
ERROR-DRIVEN	4.5	0.005	581	5.8	14	50
FIXED PART.	3.1	0.001	1129	11.3	99	100
EXCEED-MARGIN	3.4	0.003	835	8.4	35	50
EXCEED-MARGIN+ERRORS	3.3	0.002	993	9.9	36	50

Table 4: Results for Large-noisy-crossed-norm data.

	error (%)	std dev	#SVs	Cond. set (%)	cycles	batch size	time
BATCH	3.2	0.18	8321	8.3	-	-	14
ERROR-DRIVEN	9.1	0.05	4172	4.2	19	500	17
FIXED PART.	3.2	0.001	8452	8.5	201	500	20
EXCEED-MARGIN	4.5	0.02	1455	1.5	37	500	0.5
EXCEED-MARGIN+ERRORS	6.7	0.02	5308	5.3	48	500	22

Table 5: Results for OQ data.

	error (%)	std dev	#SVs	Cond. set (%)	cycles	batch size
BATCH	0.7	0.06	89	6.7	-	-
ERROR-DRIVEN	1.2	0.007	70	5.2	4	10
FIXED PART.	0.9	0.005	75	5.6	13	100
EXCEED-MARGIN	0.7	0.009	333	24.9	61	10
EXCEED-MARGIN+ERRORS	0.7	0.009	333	24.9	61	10

Table 6: Results for Breast data.

	error (%)	std dev	#SVs	Cond. set (%)	cycles	batch size
BATCH	2.7	0.16	55	11.4	-	-
ERROR-DRIVEN	3.6	0.01	82	17	2	10
FIXED PART.	2.8	0.01	51	10.6	30	10
EXCEED-MARGIN	2.7	0.01	42	8.7	12	10
EXCEED-MARGIN+ERRORS	2.9	0.009	44	9.1	12	10

Table 7: Results for Pima data.

	error (%)	std dev	#SVs	Cond. set (%)	cycles	batch size
BATCH	31.9	0.47	547	96	-	-
ERROR-DRIVEN	29.3	0.02	291	51.2	13	10
FIXED PART.	26.2	0.02	405	71.3	38	10
EXCEED-MARGIN	27.1	0.02	394	69.4	34	10
EXCEED-MARGIN+ERRORS	26.4	0.02	399	70.2	36	10

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 Figures 1-3, 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: 1617 and 1615 for the Ringnorm data; 437 and 452 for the Twonorm data; 418 and 430 for the Noisy-crossed-norm data. Since the data distribution is stationary , the performance and estimator size remain stable over time in all cases. We observe that, for each data set, 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. The results obtained for the Ringnorm data using the two methods (incremental and batch) are actually identical. 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.

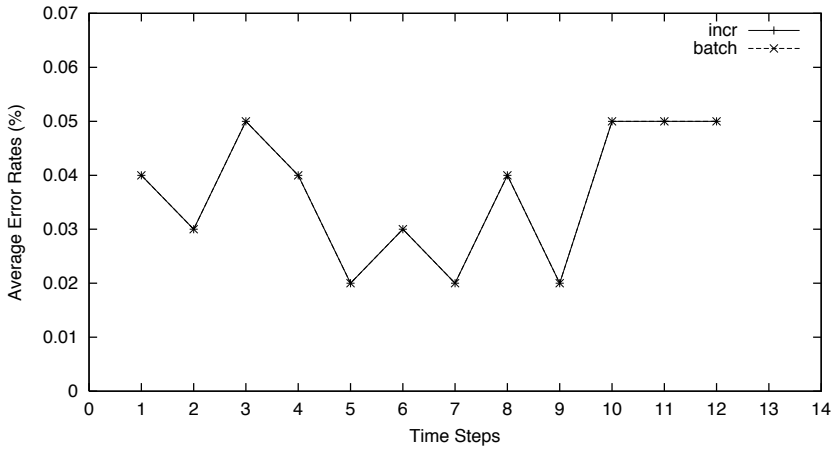

Figure 1: Ringnorm data: Average Error Rates of Fixed-partition and batch algorithms for consecutive time steps.

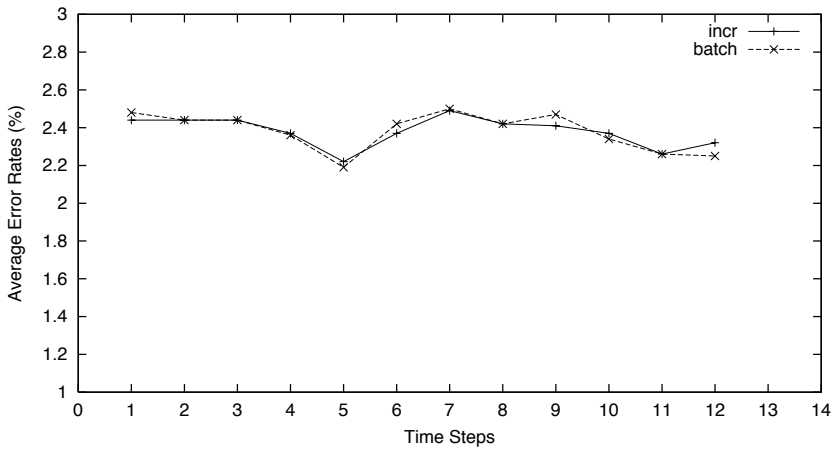

Figure 2: Twonorm data: Average Error Rates of Fixed-partition and batch algorithms for consecutive time steps.

7 Related Work

The incremental techniques discussed here can be viewed as approximations of the *chunking* technique employed to train SVMs [11]. 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

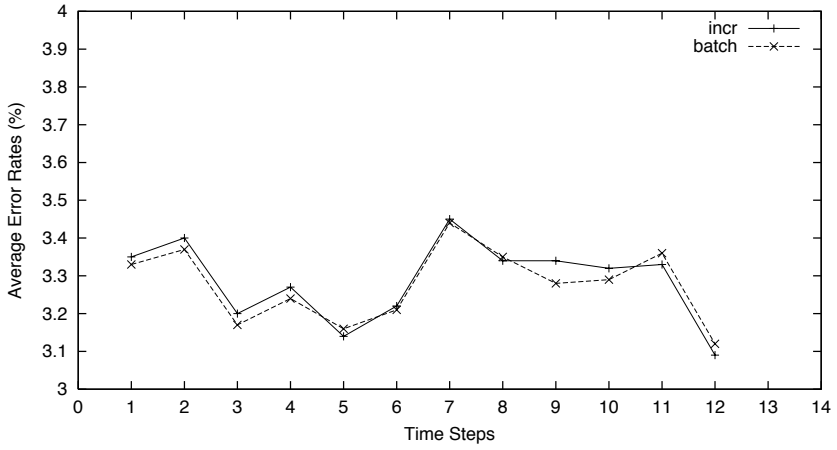

Figure 3: Noisy-crossed-norm data: Average Error Rates of Fixed-partition and batch algorithms for consecutive time steps.

to be employed within the data stream model also. Furthermore, the extensive experiments we have performed show that, although the incremental techniques allow loss of information, they are capable of achieving performance results similar to the batch algorithm.

8 Conclusions and Future Work

We have introduced and compared different 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 extend these approaches to consider the problem of constructing SVMs that accurately maintain a representation of a recent window of a data stream, and presented experimental results to show the efficiency and accuracy of the approach.

In our future work we intend to further explore the stream data model, by conducting more experiments with data sets changing distribution over time, and by developing adaptive techniques to set the b and w parameters accordingly.

References

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

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

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

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

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

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

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

[8] 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

[9] 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.

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

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

[12] 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.

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

[14] V. Vapnik, *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

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