Incremental Learning with Support Vector Machines

Nadeem Ahmed Syed **Huan Liu** **Kah Kay Sung**
Program For Research In Intelligent Systems (PRIS)
School of Computing
National University of Singapore
Singapore 119260
{nadeem, liuh, sungkk}@comp.nus.edu.sg

Abstract

As real-world databases increase in size, there is a need to scale up inductive learning algorithms to handle more training data. Incremental learning techniques are one possible solution to the scalability problem, where data is processed in parts, and the result combined so as to use less memory. Support Vector Machines (SVMs) have worked well for the batch mode learning and have shown impressive performance in many practical applications. They also have nice properties for summarizing data in the form of support vectors. This suggests one might be able to incorporate them into certain *standard* frameworks for incremental learning. This paper proposes a framework for incremental learning with SVMs and evaluates its effectiveness using a set of proposed goodness criteria on some standard machine learning benchmark datasets.

1 Introduction

Example-based learning is an attractive framework for extracting knowledge from empirical data, with the goal of generalizing well on new input patterns. Many real-world processes can be formulated as a 2-way classification task that may be solved using example-based learning methods.

When developing classifiers using learning methods, while more training data can reduce the prediction error, the learning process can itself get computationally intractable. This issue is becoming more evident today, because there are complex classification problems waiting to be solved in many domains (e.g., scientific fields, sales logs, social/economic/financial studies), where large amounts of data are already available [Fayyad *et al.*, 1996]. Researchers in the machine learning and data mining community [Catlett, 1991a; 1991b; Provost and Kolluri, 1997] have therefore been trying to scale up classical inductive learning algorithms to handle extremely large data sets.

Ideally, it is desirable to be able to consider all the examples simultaneously, to get the best possible estimate of class distribution. On the other hand when the training set is large, all the examples cannot be loaded into memory at one go. One approach to overcome this constraint is to train the classifier using an incremental learning technique, whereby only subsets of the data are to be considered at any one time and results subsequently combined. Support Vector Machines have shown good results when used for batch learning, i.e. training with a batch of data. They also have nice properties for summarizing data which they achieve by preserving the support vectors. This suggests that they could extend well to fit in an incremental learning framework, as we shall further elaborate in section 2.2. The aim of this paper is to :

1. Present an incremental learning procedure using SVMs.
2. Propose criteria for evaluating the *goodness* of the incremental learning procedure.
3. Empirically demonstrate that the proposed incremental SVM procedure performs well along the above criteria, on some standard benchmarking datasets.

2 SVM and Incremental Learning

We briefly review the key ideas behind Support Vector Machines (SVMs) which make them so appealing to be adapted to an incremental form of learning.

2.1 The Support Vector Algorithm

Support Vector Machines (SVMs) are a general class of statistical learning architectures that perform *structural risk minimization* on a nested set structure of separating hyper-planes [Vapnik, 1995]. For a classification problem, given l data points $\{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x}_l, y_l)\}$, SVM training involves solving a quadratic programming problem and the optimal solution gives rise to a decision function of the following form:

$$f(\mathbf{x}) = sgn\left[\sum_{i=1}^{l} y_i \alpha_i (\mathbf{x} \cdot \mathbf{x_i}) + b\right]$$

To allow for decision surfaces other than hyper-planes, one can first non-linearly transform the set of input

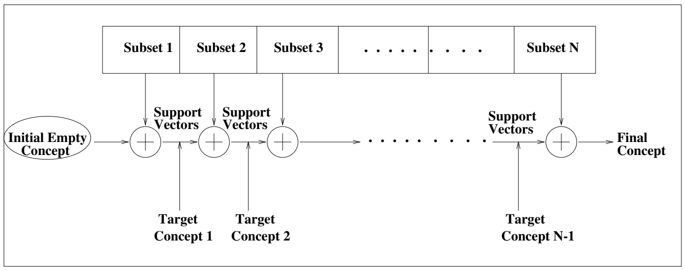


Figure 1: The Incremental Training Procedure

training vectors $\mathbf{x_1}, \ldots, \mathbf{x}_l$ into a higher-dimensional feature space using a map $\mathbf{\Phi}(\mathbf{x_i}) \mapsto \mathbf{z_i}$, and then do a linear separation there. This leads to decision functions of the form:

$$\begin{aligned} f(\mathbf{x}) &= sgn\left[\sum_{i=1}^{l} y_i \alpha_i (\mathbf{\Phi}(\mathbf{x}) \cdot \mathbf{\Phi}(\mathbf{x_i})) + b\right] \\ &= sgn\left[\sum_{i=1}^{m} y_i \alpha_i K(\mathbf{x}, \mathbf{x_i}) + b\right] \end{aligned}$$

where $K(\mathbf{x}, \mathbf{x_i}) = (\mathbf{\Phi}(\mathbf{x}) \cdot \mathbf{\Phi}(\mathbf{x_i}))$ is known as a *kernel*. Notice that the training and classification procedures now involve computing high dimensional dot products of the form $(\mathbf{\Phi}(\mathbf{x}) \cdot \mathbf{\Phi}(\mathbf{x_i}))$. To avoid these computationally expensive operations, we consider only those classes of $\mathbf{\Phi}$ that reduce to simple kernel functions K.

Often, only a small fraction of the α_i coefficients are non-zero. The corresponding pairs of $\mathbf{x_i}$ entries (known as *support vectors*) and y_i output labels fully define the decision function. Together with the α_i coefficients, they are preserved for use in the classification procedure. All remaining training examples that do not contribute to the decision function may be regarded as redundant.

2.2 Incremental Training

Given that only a small fraction of training examples end up as support vectors [Vapnik, 1995], the support vector algorithm is able to summarize the data space in a very concise manner.

This suggests that we could partition a huge database such that each partition fits into the main memory, and then incrementally train SVM classifier with the partitions. The training would preserve only the support vectors at each incremental step, and add them to the training set for the next step (Figure 1). The model obtained by this method should be the same or similar to what would have been obtained using all the data together to train. The reason for this is that the SVM algorithm will preserve the essential class boundary information seen so far in a very compact form as support vectors, which would contribute appropriately to deriving the new concept in the next iteration.

To evaluate the effectiveness of incremental training, we compare the performance of the incrementally trained model against the model trained with all the data so far in one batch. More specifically, we need to compare the following two cases :

Case 1 : The first case is where we have only the learned model from the previously seen data (preserved in the form of support vectors), and as new information comes in, how does the classification algorithm do. Does the performance deteriorate as we keep discarding the "redundant" examples at each successive incremental step, when the new examples come in.

Case 2: The second, in which we save all the previously seen data, and as new information arrives, how well does the classification algorithm (SVM) fare on unknown data. This is effectively batch learning using all the data seen so far.

3 Evaluation and Discussion

To analyze these two situations, the experiment was designed as follows. Let us call the training set TR, and the test set TE. In each iteration the training set was further split into 10 parts TR_i, where $i = 1, 2, 3, 4, ...10$, with each part containing mutually exclusive 10% of the data from TR. Now the following two types of incremental learning were carried out:

E1. We trained an SVM on TR_1. Then we took the support vectors chosen from TR_1, let us call these SV_1, and added to them TR_2. Again we ran the SVM training algorithm on $SV_1 + TR_2$. At each step, we used the trained machine to classify TE. Now the SVs chosen from the training of $SV_1 + TR_2$ were taken, let us call these SV_2, and TR_3 was added to them. Again the training and testing were done. This incremental step was repeated until all the TR_i are used.

E2. An SVM was trained using TR_1. Then the trained machine was used to classify the examples in TE, and classification rate (% correct classification) was recorded. Then TR_2 was added to TR_1, and $TR_1 + TR_2$ was used for training. Again the trained machine was used to classify TE, and results were recorded. Then TR_3 was added to $TR_1 + TR_2$, and so on, until all the data in the original training set was being used for training.

We arbitrarily use 10% of the data for each partition TR_i. While incrementally learning from large dataset, we can only use as much data at any step as the buffer can hold. So this partition could be 10% of the dataset or 20% of the dataset, or whatever suits the conditions.

The two steps above are repeated 10 times. Each time mutually exclusive 10% of the examples were used as test set TE, and the rest were used for the training set TR (on the lines of 10-fold cross validation.) The problem being handled in all these experiments is two-class classification problem.

Dataset	*# of Examples*	*No. of Attr.*
Australian	670	14
Diabetes	768	8
German	1000	24
Heart	270	13
Ionosphere	351	34
Liver-Disorders	345	6
Monks-1	432	6
Monks-2	432	6
Monks-3	432	6
Mushroom	8124	22
Promoter-Gene	106	57
Sonar	208	60

Table 1: The datasets used in experiments.

Dataset	*Batch method*	*Incr. method*
Australian	84.63	84.63
Diabetes	77.08	76.95
German	75.10	74.10
Heart	84.45	79.63
Ionosphere	94.57	93.14
Liver-Disorders	68.68	61.03
Monk-1	100	100
Monk-2	99.17	99.17
Monk-3	98.91	98.73
Mushroom	100	99.88
Promoter-Gene	93.55	92.55
Sonar	87.38	87.41
Average	88.87	87.27

Table 2: Final prediction accuracy of batch and incremental methods

The datasets used for carrying out the empirical study are summarized in Table 1. The datasets used are those that are standard among the machine learning community, for benchmarking purposes. All the datasets have been obtained from the UCI-machine learning repository [1]. We also obtained SVM^{light} [2] for our experiment.

The first step, E1, gives us a model obtained through incremental training as in Figure 1. With E2 we get a model trained with all the examples so far, i.e., $TR_1 \bigcup TR_2 \bigcup ... \bigcup TR_i$ in one batch. At each incremental step, comparing these two cases gives us an insight as to how the incremental method fares when compared with the batch method. Figure 2 shows the plot of the two cases for the various datasets as the learning progresses. The X axis is for the incremental step, and the Y-axis (the height of the bar) denotes the prediction accuracy of the models obtained at the end of the corresponding incremental step. The kernel used and the parameter setting is given within brackets next to the name of each dataset

It is clear from the plots that the prediction accuracy of the incrementally trained SVM method closely follows the performance of the SVM trained in one batch with all the data. This observation is important because it provides empirical evidence that the support vectors chosen by the SVM algorithm are able to remember the relevant parts of the data seen before, in a very concise and effective form. Table 2 shows the final prediction accuracy of the incremental and the batch training methods after training is completed through 10 incremental steps. The table clearly shows that there is negligible or no drop in prediction accuracy of the support vector machines when they are incrementally trained.

The very idea of support vectors is that they are sufficient to represent the data space, and they have previously been demonstrated to clearly do so [Vapnik, 1995], but all these results have been shown with batch training and testing. The novelty of our result is that it is the first empirical evidence we are aware of, that even after successive steps of discarding the redundant examples (as in the context of incremental learning), the successively collected set of support vectors still remain a representative set.

4 Succinctness of Support Vector Representation

Since our aim is to incrementally train SVM from a large dataset, it is desirable that the selected support support vector set is as small as possible. To investigate whether the selected set of support vectors are really a necessary set, we conducted another set of experiments. In these experiments our aim is to observe the effect of removing some examples from the support vector set on the representativeness of the remaining support vectors. To achieve this, the experiment was conducted as follows :

1. The SVM algorithm was run on the whole dataset D, and the support vector set SV were obtained.
2. SVM was trained on the obtained support vector set SV using 10-fold cross validation i.e. for 10-fold cross validation the training set TR and the test set TE were both obtained from the support vector set SV. The average prediction accuracy of the trained machine on the test set was recorded.

The effect of second step was that 10% of the examples from the support vector set SV were used to form the testing set TE and the rest used to form the training set TR. Further over the 10 folds of cross validation, different $1/10th$ of the support vectors were missing from the training set. If all the support vectors are necessary then each one of them will be non-redundant. As such, it is reasonable to expect that training SVM with one part TR of the SV and testing on another part TE should yield poor prediction accuracy.

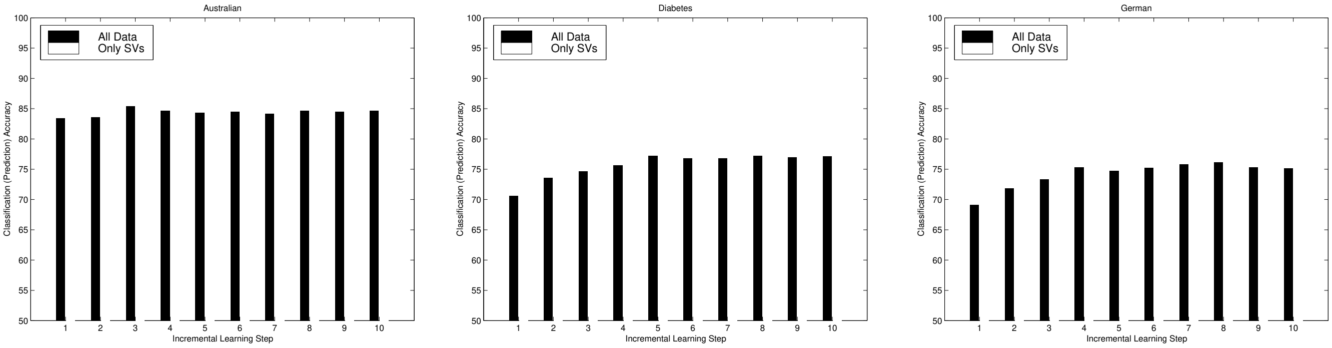
Table 3 provides the prediction accuracy on various datasets for this experiment. The first column shows the percentage of examples chosen as support vectors in step

---

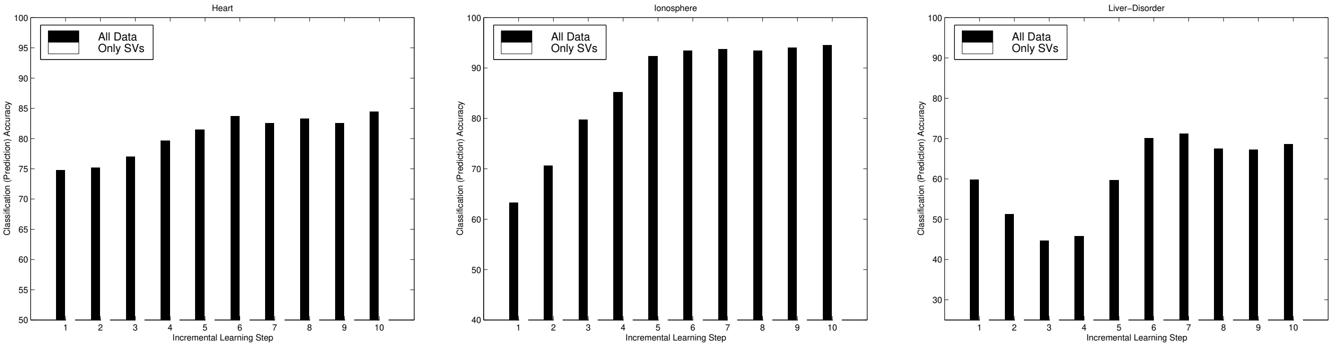
[1] http://www.ics.uci.edu/~mlearn/MLRepository.html

[2] http://www-ai.informatik.u ni-dortmund.de/ FORSCHUNG/VERFAHREN/SVM_LIGHT/ svm_light.eng.html

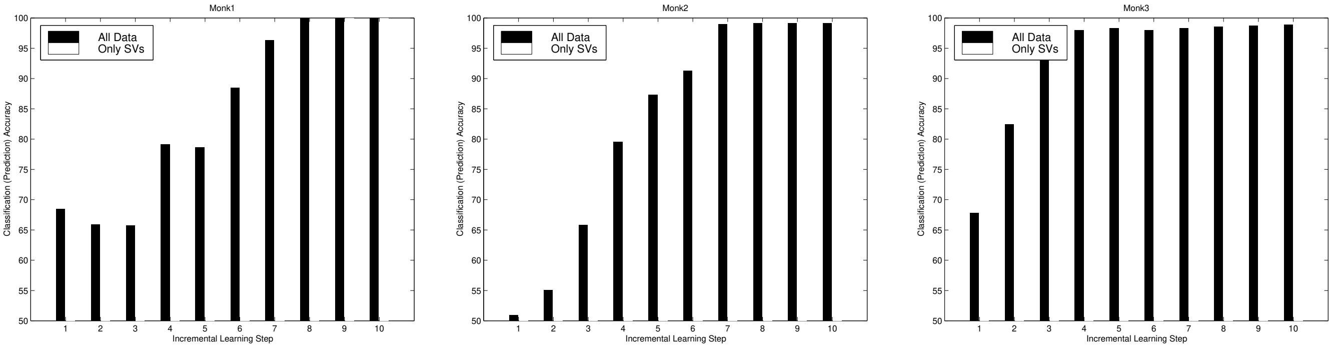
(1) Australian (gaussian $\sigma = 0.0005$) (2) Diabetes (gaussian $\sigma = 0.0005$) (3) German (gaussian $\sigma = 0.005$)



(4) Heart (gaussian $\sigma = 0.0005$) (5) Ionoshpere (gaussian $\sigma = 1.6$) (6) Liver Disorders (gaussian $\sigma = 0.1$)



(7) Monk-1 (gaussian $\sigma = 0.1$) (8) Monk-2 (gaussian $\sigma = 0.1$) (9) Monk-3 (gaussian $\sigma = 0.001$)



(10) Mushroom (linear) (11) Promoter-gene (linear) (12) Sonar (gaussian $\sigma = 0.4$)

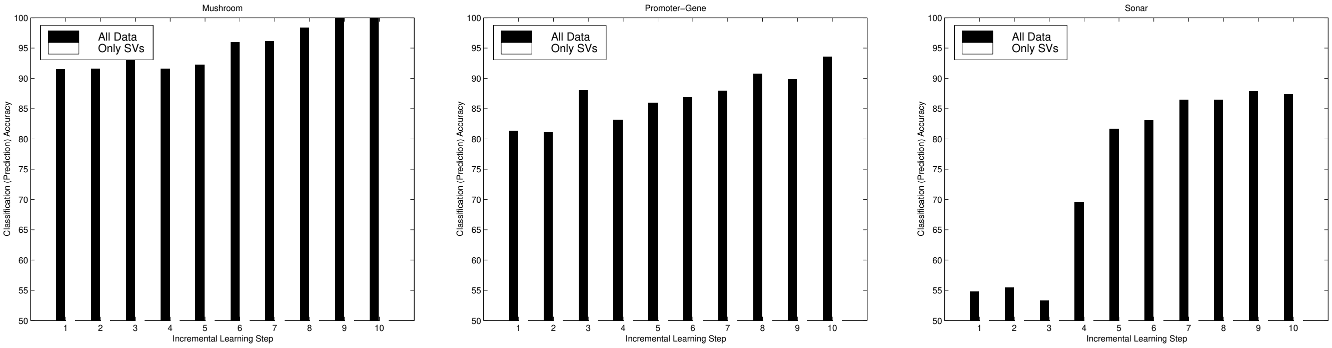


Figure 2: Concept Drift Results on benchmark datsets

Dataset	*% of Exs.* Selected	*%Accuracy* (All data)	*%Accuracy* (Selected Exs.)
Australian	33.76	84.63	48.40
Diabetes	58.03	77.08	51.68
German	54.16	75.10	45.35
Heart	35.16	84.45	33.31
Ion.	53.05	94.57	84.77
Liver.	67.64	68.68	21.25
Monk-1	82.61	100.0	100
Monk-2	78.76	99.17	98.75
Monk-3	23.54	98.91	94.04
Mush.	05.99	100.0	100
Gene	69.19	93.55	79.46
Sonar	67.38	87.38	58.81
Avg.	52.44	88.87	67.99

Table 3: Results illustrating the necessity of the selected samples

1 of the experiment, the second column shows the average prediction accuracy of the classifier when the 10 fold cross validation was carried out on the whole dataset D. The last column shows the prediction accuracy obtained in step 2 above.

It can be seen that the accuracy drops drastically when a training set is created from a subset (90%) of the selected support vectors. This means that removing even a small portion of the support vectors (10%) from the set SV affects the representation capability of the remaining set. This in turn implies that the support vector set chosen by the SVM algorithm is a minimal set and removing any more samples would result in the loss of vital information about the class distribution. These results clearly demonstrate that set of chosen support vectors forms a necessary set to retain the structure of the original data space. These results combined with the results of the previous section provide clear evidence that the support vector machine is very good at summarizing the data into a compact form, and that the selected support vectors form a minimal set. As a result of these properties SVMs readily fit into incremental framework, and perform just as well when incrementally trained.

5 Related Work

Our method can be considered similar in spirit to decomposition or "chunking" techniques employed to train SVMs, as in [Osuna *et al.*, 1997]. Our method differs in that unlike other methods our technique looks at the examples only once to determine if they will become support vectors. Once discarded, the vectors are not considered again. On the other hand, for example, Osuna's decomposition algorithm is an iterative method, which cycles through the training set a few times to select the support vectors. Our method can be considered as a kind of lossy approximation of the chunking methods. Consequently, our positive results show that such an approximation can be performed, without significantly deteriorating the performance of the algorithm.

6 Conclusion

We started by suggesting that support vector machines should be adaptable to incremental training, and provide the reasons behind our suggestions. To investigate and demonstrate the validity of our suggestion, we have empirically shown that support machines perform very well under incremental training. We also provide empirical evidence that the chosen support vector sets form a minimal set.

Because of these positive results for SVMs, we set out to study whether the support vector sets can be used to efficiently train other kinds of learning algorithms such as decision tree induction [Quinlan, 1993], neural networks [Rumelhart *et al.*, 1986; Aleksander and Morton, 1990], instance-based learning [Aha *et al.*, 1991; Cover and Hart, 1967], to attain our goal of scalable data mining.

References

[Aha *et al.*, 1991] D. W. Aha, D. Kilber, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[Aleksander and Morton, 1990] I. Aleksander and H. Morton. *An Introduction to Neural Computing.* Chapman and Hall, 1990.

[Catlett, 1991a] J. Catlett. *Megainduction : A Machine Learning on Very Large Databases.* PhD thesis, Department of Computer Science, University of Sydney, Australia, 1991.

[Catlett, 1991b] J. Catlett. Megainduction : A test flight. In *Proceedings of Eighth International Workshop on Machine Learning*, pages 596–599. Morgan Kaufmann, 1991.

[Cover and Hart, 1967] T. M. Cover and P. E. Hart. Nearest neighbour pattern classification. *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, 13:21–27, 1967.

[Fayyad *et al.*, 1996] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery:an overview. In *Advances in Knowledge Discovery and Data Mining.* MIT Press, 1996.

[Osuna *et al.*, 1997] E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In *Proceedings of IEEE NNSP'97*, Amelia Island, FL, 1997.

[Provost and Kolluri, 1997] 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.

[Quinlan, 1993] J. R. Quinlan. *C4.5: Programs For Machine Learning.* Morgan Kaufmann, San Mateo, CA., 1993.

[Rumelhart *et al.*, 1986] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart, D. E. McCleland, and "the PDP Research Gruop, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1, Foundations.*, pages 318–362. MIT Press, Cambridge MA, 1986.

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