Ensemble of SVMs for Incremental Learning

Zeki Erdem[1,4], Robi Polikar[2], Fikret Gurgen[3], and Nejat Yumusak[4]

[1] TUBITAK Marmara Research Center, Information Technologies Institute,
41470 Gebze - Kocaeli, Turkey
zeki.erdem@bte.mam.gov.tr
[2] Rowan University, Electrical and Computer Engineering Department,
210 Mullica Hill Rd., Glassboro, NJ 08028, USA
polikar@rowan.edu
[3] Bogazici University, Computer Engineering Department,
Bebek, 80815 Istanbul, Turkey
gurgen@boun.edu.tr
[4] Sakarya University, Computer Engineering Department,
Esentepe, 54187 Sakarya, Turkey
nyumusak@sakarya.edu.tr

Abstract. Support Vector Machines (SVMs) have been successfully applied to solve a large number of classification and regression problems. However, SVMs suffer from the *catastrophic forgetting* phenomenon, which results in loss of previously learned information. Learn^{++} have recently been introduced as an incremental learning algorithm. The strength of Learn^{++} lies in its ability to learn new data without forgetting previously acquired knowledge and without requiring access to any of the previously seen data, even when the new data introduce new classes. To address the *catastrophic forgetting* problem and to add the incremental learning capability to SVMs, we propose using an ensemble of SVMs trained with Learn^{++}. Simulation results on real-world and benchmark datasets suggest that the proposed approach is promising.

1 Introduction

Support Vector Machines (SVMs) have enjoyed a remarkable success as effective and practical tools for a broad range of classification and regression applications [1-2]. As with any type of classifier, the performance and accuracy of SVMs rely on the availability of a representative set of training dataset. In many practical applications, however, acquisition of such a representative dataset is expensive and time consuming. Consequently, such data often become available in small and separate batches at different times. In such cases, a typical approach is combining new data with all previous data, and training a new classifier from scratch. In other words, such scenarios require a classifier to be trained and incrementally updated, where the classifier needs to learn the novel information provided by the new data without forgetting the knowledge previously acquired from the data seen earlier. Learning new information without forgetting previously acquired knowledge, however, raises the stability–plasticity dilemma [3]. A completely stable classifier can retain knowledge, but cannot learn

N.C. Oza et al. (Eds.): MCS 2005, LNCS 3541, pp. 246–256, 2005.

new information, whereas a completely plastic classifier can instantly learn new information, but cannot retain previous knowledge. The approach generally followed for learning from new data involves discarding the existing classifier, combining the old and the new data and training a new classifier from scratch using the aggregate data. This approach, however, results in *catastrophic forgetting* (also called unlearning) [4], which can be defined as the inability of the system to learn new patterns without forgetting previously learned ones. Methods to adress this problem include retraining the classifier on a selection of past or new data points generated from the problem space. However, this approach is unfeasible if previous data are no longer available.

Such problems can be best addressed through incremental learning, defined as the process of extracting new information without losing prior knowledge from an additional dataset that later becomes available. Various definitions and interpretations of incremental learning can be found in [6] and references within. For the purposes of this paper, we define an incremental learning algorithm as one that meets the following demanding criteria [5,6]:

1. be able to learn additional information from new data.
2. not require access to the original data used to train the existing classifier.
3. preserve previously acquired knowledge (that is, it should not suffer from catastrophic forgetting).
4. be able to accommodate new classes that may be introduced with new data.

In this paper we describe an ensemble of classifiers approach: ensemble systems have attracted a great deal of attention over the last decade due to their empirical success over single classifier systems on a variety of applications. An ensemble of classifiers system is a set of classifiers whose individual decisions are combined in some way to obtain a meta classifier. One of the most active areas of research in supervised learning has been to study methods for constructing good ensembles of classifiers. The main discovery is that ensembles are often more accurate than the individual classifiers that make them up. A rich collection of algorithms have been developed using multiple classifiers, such as AdaBoost [7] and its many variations, with the general goal of improving the generalization performance of the classification system. Using multiple classifiers for incremental learning, however, has been largely unexplored. Learn^{++}, in part inspired by AdaBoost, was developed in response to recognizing the potential feasibility of ensemble of classifiers in solving the incremental learning problem. Learn^{++} was initially introduced in [5] as an incremental learning algorithm for MLP type networks. A more versatile form of the algorithm was presented in [6] for all supervised classifiers.

Since SVMs are stable classifiers and use the global partitioning (global learning) technique, they are also susceptible to the catastrophic forgetting problem [8]. The SVMs optimise the positioning of the hyperplanes to achieve maximum distance from all data samples on both sides of the hyperplane through learning. Therefore, SVMs are unable to learn incrementally from new data. Since training of SVMs is usually present as a quadratic programming problem, it is a challenging task for the large data sets due to the high memory requirements and slow convergence. To overcome these

drawbacks, various methods have been proposed for incremental SVM learning in the literature [9-14]. On the other hand, some studies have also been presented to further improve classification performance and accuracy of SVMs with ensemble methods, such as boosting and bagging [15-18]. In this study, we consider the ensemble based incremental SVM approach. The purpose of this study was to investigate whether the incremental learning capability can be added to SVM classifiers through the Learn^{++} algorithm, while avoiding the catastrophic forgetting problem.

2 Learn^{++}

The strength of Learn^{++} as an ensemble of classifiers approach lies in its ability to incrementally learn additional information from new data. Specifically, for each dataset that becomes available, Learn^{++} generates an ensemble of classifiers, whose outputs are combined through weighted majority voting to obtain the final classification. Classifiers are trained based on a dynamically updated distribution over the training data instances, where the distribution is biased towards those novel instances that have not been properly learned or seen by the previous ensemble(s). The pseudocode for Learn^{++} is provided in Figure 1.

For each dataset $\boldsymbol{D}_k$, k=1,…,K that is submitted to Learn^{++}, the inputs to the algorithm are (i) $S_k = \{(x_i, y_i) | i = 1,...,m_k\}$, a sequence of m_k training data instances x_i along with their correct labels y_i, (ii) a classification algorithm **BaseClassifier** to generate hypotheses, and (iii) an integer T_k specifying the number of classifiers (hypotheses) to be generated for that dataset. The only requirement on the **BaseClassifier** is that it obtains a 50% correct classification performance on its own training dataset. **BaseClassifier** can be any supervised classifier such as a multilayer perceptron, radial basis function, a decision tree, or of course, a SVM.

Learn^{++} starts by initializing a set of weights for the training data, $\boldsymbol{w}$, and a distribution $\boldsymbol{D}$ obtained from $\boldsymbol{w}$, according to which a training subset TR_t and a test subset TE_t are drawn at the t^{th} iteration of the algorithm. Unless a priori information indicates otherwise, this distribution is initially set to be uniform, giving equal probability to each instance to be selected into the first training subset.

At each iteration t, the weights adjusted at iteration t-1 are normalized to ensure that a legitimate distribution, $\boldsymbol{D}_t$, is obtained (step 1). Training and test subsets are drawn according to $\boldsymbol{D}_t$ (step 2), and the base classifier is trained with the training subset (step 3). A hypothesis h_t is obtained as the t^{th} classifier, whose error ε_t is computed on the entire (current) dataset $S_k = TR_t + TE_t$, simply by adding the distribution weights of the misclassified instances (step 4).

$$\varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_t(i) \tag{1}$$

The error, as defined in Equation (1), is required to be less than ½ to ensure that a minimum reasonable performance can be expected from h_t. If this is the case, the hypothesis h_t is accepted and the error is normalized to obtain the normalized error

$$\beta_t = \varepsilon_t/(1-\varepsilon_t), \quad 0 < \beta_t < 1 \tag{2}$$

If $\varepsilon_t \geq$ ½, then the current hypothesis is discarded, and a new training subset is selected by returning to step 2. All hypotheses generated so far are then combined using the weighted majority voting to obtain the composite hypothesis H_t (step 5).

$$H_t = \arg\max_{y \in Y} \sum_{t:h_t(x)=y} \log(1/\beta_t) \tag{3}$$

The weighted majority voting scheme allows the algorithm to choose the class receiving the highest vote from all hypotheses, where the voting weight for each hypothesis is inversely proportional to its normalized error. Therefore, those hypotheses with good performances are awarded a higher voting weight. The error of the composite hypothesis is then computed in a similar fashion as the sum of distribution weights of the instances that are misclassified by H_t (step 6):

$$E_t = \sum_{i:H_t(x_i)\neq y_i} D_t(i) = \sum_{i=1}^{m} D_t(i)[\![H_t(x_i) \neq y_i]\!] \tag{4}$$

where [|·|] evaluates to 1, if the predicate holds true.

Input: For each dataset drawn from $\boldsymbol{D_k}$ $k=1,2,\ldots,K$

- Sequence of m examples $S_k=\{(x_i,y_i) \mid i=1,\ldots,m_k\}$.
- Learning algorithm **BaseClassifier**.
- Integer T_k, specifying the number of iterations.

Initialize $w_1(i) = D_1(i) = 1/m_k,\ \forall i,\ i = 1,2,\cdots,m_k$

Do for each $k=1,2,\ldots,K$:

Do for $t = 1,2,\ldots,T_k$:

1. Set $D_t = \mathbf{w}_t \Big/ \sum_{i=1}^{m} w_t(i)$ so that D_t is a distribution.
2. Choose training TR_t and testing TE_t subsets from D_t.
3. Call WeakLearn, providing it with TR_t.
4. Obtain a hypothesis $h_t : X \rightarrow Y$, and calculate the error of h_t: $\varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_t(i)$ on $S_k = TR_t + TE_t$. If $\varepsilon_t > ½$, set $t = t - 1$, discard h_t and go to step 2. Otherwise, compute normalized error as $\beta_t = \varepsilon_t / (1-\varepsilon_t)$.
5. Call weighted majority voting and obtain the composite hypothesis
$$H_t = \arg\max_{y \in Y} \sum_{t:h_t(x)=y} \log(1/\beta_t)$$
6. Compute the error of the composite hypothesis
$$E_t = \sum_{i:H_t(x_i)\neq y_i} D_t(i) = \sum_{i=1}^{m} D_t(i)[\![H_t(x_i) \neq y_i]\!]$$
7. Set $B_t = E_t/(1-E_t)$, and update the weights:
$$w_{t+1}(i) = w_t(i) \times \begin{cases} B_t, & if\ \ H_t(x_i) = y_i \\ 1, & otherwise \end{cases} = w_t(i) \times B_t^{1-[\![H_t(x_i)\neq y_i]\!]}$$

Call weighted majority voting and **Output** the final hypothesis:

$$H_{final}(x) = \arg\max_{y \in Y} \sum_{k=1}^{K} \sum_{t:h_t(x)=y} \log \frac{1}{\beta_t}$$

Fig. 1. The Learn^{++} Algorithm

The normalized composite error is then computed

$$B_t = E_t/(1 - E_t),\ \ 0 < B_t < 1 \tag{5}$$

to be used in the weight update rule (step 7) of Equation (6). This rule reduces the weights of those instances that are correctly classified by the composite hypothesis H_t, lowering their probability of being selected into the next training subset.

$$w_{t+1}(i) \quad = w_t(i) \times \begin{cases} B_t\ , & if\ \ H_t(x_i) = y_i \\ 1\ \ , & otherwise \end{cases} = w_t(i) \times B_t^{\,1-[|H_t(x_i) \neq y_i|]} \tag{6}$$

In effect, the weights of misclassified instances are increased relative to the rest of the dataset. We emphasize that, unlike AdaBoost and its variations, the weight update rule in Learn^{++} looks directly at the classification of the composite hypothesis (that is, the ensemble), not that of a specific hypothesis. This weight update procedure forces the algorithm to focus more and more on instances that have not been properly learned by the ensemble. When Learn^{++} is learning incrementally, the instances introduced by the new dataset (and in particular from new classes, if applicable) are precisely those not learned by the ensemble, and hence the algorithm quickly focuses on these instances. At any point, a final hypothesis H_{final} can be obtained by combining all hypotheses that have been generated so far using the weighted majority voting rule.

$$H_{final}(x) = \arg\max_{y \in Y} \sum_{k=1}^{K} \sum_{t:h_t(x)=y} \log \frac{1}{\beta_t} \tag{7}$$

Simulation results of Learn^{++} on incremental learning with MLPs used as base classifiers on a variety of datasets as well as comparisons to AdaBoost and other methods of incremental learning can be found in [6] and references within.

3 SVM Classifiers

Support vector machines (SVMs) have been successfully employed in a number of real world problems [1-2]. They directly implement the principle of structural risk minimization [1] and work by mapping the training points into a high dimensional feature space, where a separating hyperplane ($\boldsymbol{w}$, b) is found by maximizing the distance from the closest data points (boundary-optimization). Given a set of training samples $\boldsymbol{S}=\{(\boldsymbol{x}_i, y_i) \mid i=1,\ldots,m\}$, where $\boldsymbol{x}_i \in \boldsymbol{R}^n$ are input patterns, $y_i \in \{+1, -1\}$ are class labels for a 2-class problem, SVMs attempt to find a classifier $h(\boldsymbol{x})$, which minimizes the expected misclassification rate. A linear classifier $h(\boldsymbol{x})$ is a hyperplane, and can be represented as $h(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T\boldsymbol{x}+b)$. The optimal SVM classifier can then be found by solving a convex quadratic optimization problem:

$$\max_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \xi_i \quad \text{subject to } \ y_i\left(\langle w, x_i \rangle + b\right) \geq 1 - \xi_i \ \text{ and } \ \xi_i \geq 0 \tag{8}$$

where b is the bias, $\boldsymbol{w}$ is weight vector, and C is the regularization parameter, used to balance the classifier's complexity and classification accuracy on the training set S. Simply replacing the involved vector inner-product with a non-linear kernel function converts linear SVM into a more flexible non-linear classifier, which is the essence of the famous *kernel trick*. In this case, the quadratic problem is generally solved through its dual formulation:

$$L(w,b,\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\left(\sum_{i=1}^{m} y_i y_j \alpha_i \alpha_j K(x_i, x_j) \right) \text{ subject to } C \geq \alpha_i \geq 0 \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0 \tag{9}$$

where α_i are the coefficients that are maximized by Lagrangian. For training samples $\boldsymbol{x}_i$, for which the functional margin is one (and hence lie closest to the hyperplane), $\alpha_i > 0$. Only these instances are involved in the weight vector, and hence are called the *support vectors* [2]. The non-linear SVM classification function (optimum separating hyperplane) is then formulated in terms of these kernels as:

$$h(x) = sign\left(\sum_{i=1}^{m} \alpha_i y_i K(x_i, x_j) - b \right). \tag{10}$$

As mentioned earlier in Learn^{++} algorithm, incremental learning of SVMs is based on the following intuition: The ensemble is obtained by retraining a single SVM using strategically updated distributions of the training dataset, which ensures that examples that are misclassified by the current ensemble have a high probability of being resampled. The examples that have a high probability of error are precisely those that are unknown or that have not yet been used to train the previous classifiers. Distribution update rule is optimized for incremental learning of new data, in particular when the new data introduce new classes. After T_k classifiers are generated for each D_k, the final ensemble of SVMs is obtained by the weighted majority of all composite SVMs:

$$H_{final}(x) = \arg\max_{y \in Y} \sum_{k=1}^{K} \sum_{t:h_t(x)=y} \log \frac{1}{\beta_t}. \tag{11}$$

4 Simulation Results

Proposed incremental learning approach for SVMs using Learn^{++} has been tested on several datasets. For brevity, we will henceforth use the term SVMLearn^{++} for the proposed approach and present results on one benchmark dataset and one real-world application. The benchmark dataset is the Optical Character Recognition dataset from UCI machine learning repository, and the real world application is a gas identification problem for determining one of five volatile organic compounds based on chemical sensor data.

Two nonlinear SVM kernel functions were used in our experiments: Polynomial and Gaussian kernel functions.

Polynomial kernel: $$K(x_i, x_j) = \left(\langle x_i, x_j \rangle + 1 \right)^d \tag{12}$$

RBF kernel : $$K(x_i, x_j) = \exp\left(-\left\|x_i - x_j\right\|^2 / 2\sigma^2\right) \quad (13)$$

SVM classifier parameters are the regularization constant C, and the polynomial degree d (for the polynomial kernel) or the RBF width σ, (for the RBF kernel function). The choice of classifier parameters is a form of model selection. Although the machine learning community has extensively considered model selection with SVMs, optimal model parameters are generally domain-specific [19]. Therefore, kernel and regularization parameters were selected jointly to evaluate the best model for each dataset. We used the cross-validation technique with 5-folds to assess SVMs with given kernel parameter and regularization constants.

4.1 Optical Character Recognition Dataset

The Optical Character Recognition (OCR) data has 10 classes with digits 0-9 and 64 attributes. The dataset was split into four to create three training (**DS1, DS2, DS3**) and a test subsets (**Test**), whose distribution is given in Table 1. We evaluated the performance and the incremental learning ability of SVMs using Learn^{++} on a fixed number of classifiers rather than determining the number of classifiers via a validation set. SVMLearn^{++} was allowed to create seven classifiers with the addition of each dataset using both polynomial kernel (PolySVM) and kernel (RBFSVM), for a total of classifiers in three training sessions. The data distribution was deliberately made rather challenging, specifically designed to test the ability of proposed approach to learn multiple new classes at once with each additional dataset while retaining the knowledge of previously learned classes. In this incremental learning problem, instances from only six of the ten classes are present in each subsequent dataset resulting in a rather difficult problem.

Table 1. OCR data distribution

Class	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
DS1	250	250	250	0	0	250	250	250	0	0
DS2	150	0	150	250	0	150	0	150	250	0
DS3	0	150	0	150	400	0	150	0	150	400
Test	110	114	111	114	113	111	111	113	110	112

Results from this test are shown in Tables 2 and 3 based on an average of 20 trials. The last two columns are the average overall generalization performance (**Gen.**) on test data, and the standard deviation (**Std.**) of the generalization performances.

Poly SVMLearn^{++} was able to learn the new classes, 4 and 9, only poorly after they were first introduced in **DS2** but able to learn them rather well, when further trained with these classes in **DS3**. However, it performs rather well on classes 5 and 10 after they are first introduced in **DS3**. RBF SVMLearn^{++} was able to learn the classes 4 and 9, only poorly when they were introduced in **DS2** but able to learn them rather well, when further trained with these classes in **DS3**. Similarly, it performs rather poorly on classes 5 and 10 after they are first introduced in **DS3**, though it is reasonable to expect that it would do well on these classes with additional training.

Table 2. SVMLearn^{++} with polynomial kernel (*degree* = 3, C = 1) results on OCR dataset

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	Gen.	Std.
DS1	99%	100%	99%	-	-	100%	100%	99%	-	-	60%	0.07%
DS2	99%	100%	99%	17%	-	100%	100%	99%	21%	-	63%	2.37%
DS3	99%	100%	99%	94%	84%	100%	100%	99%	91%	94%	78%	2.32%

Table 3. SVMLearn^{++} with RBF kernel (σ = 0.1, C =1) results on OCR dataset

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	Gen.	Std.
DS1	99%	100%	100%	-	-	98%	100%	99%	-	-	60%	0.03%
DS2	99%	73%	100%	44%	-	98%	68%	99%	47%	-	63%	1.54%
DS3	99%	100%	100%	93%	14%	97%	100%	99%	90%	13%	80%	4.17%

The generalization performance of Poly and RBF SVMLearn^{++} is computed on the entire test data which included instances from all classes. This is why the generalization performance is only around 60% after the first training session, since the algorithms had seen only six of the 10 classes by that time. Both Poly and RBF SVMLean^{++} exhibit the ability of learning incrementally with a final overall generalization performance of 78-80% after new datasets are introduced.

4.2 Volatile Organic Compounds Dataset

The Volatile Organic Compounds (VOC) dataset is from a real world application that consists of 5 classes (toluene, xylene, hectane, octane and ketone) with 6 attributes coming from six (quartz crystal microbalance type) chemical gas sensors. The dataset was divided into three training and a test dataset. The distribution of the data is given in Table 4, where a new class was introduced with each dataset.

Table 4. VOC data distribution

Class	C1	C2	C3	C4	C5
DS1	20	0	20	0	40
DS2	10	25	10	0	10
DS3	10	15	10	40	10
Test	24	24	24	40	52

Again, SVMLearn^{++} was incrementally trained with three subsequent training datasets. In this experiment, Poly and RBF SVMLearn^{++} was allowed to generate as many classifiers as necessary to obtain their maximum performance. The number of classifiers generated were 5, 10, 18 (a total of 33 classifiers to achieve their best performance) for SVM classifiers with polynomial kernel (PolySVM) and RBF kernel (RBFSVM) in three training sessions. Results from this test are shown in Tables 5 and 6 based on average of 30 trials.

Table 5. SVMLearn^{++} with polynomial kernel (*degree* = 3, C = 100) results on VOC dataset

	C1	C2	C3	C4	C5	Gen.	Std
DS1	92%	-	88%	-	100%	58%	1.21%
DS2	98%	91%	94%	-	97%	72%	1.29%
DS3	96%	96%	98%	78%	76%	85%	7.29%

Table 6. SVMLearn^{++} with RBF kernel (σ = 3, C =100) results on VOC dataset

	C1	C2	C3	C4	C5	Gen.	Std.
DS1	91%	-	95%	-	99%	58%	1.62%
DS2	97%	91%	81%	-	95%	70%	1.84%
DS3	93%	99%	94%	68%	76%	83%	8.19%

The generalization performance of Poly and RBF SVMLearn^{++} on the test dataset gradually improved from 58% to 83-85% as new data were introduced, demonstrating its incremental learning capability even when instances of new classes are introduced in subsequent training sessions.

5 Conclusions

In this paper, we have shown that the SVM classifiers can in fact be equipped with the incremental learning capability, to address the catastrophic forgetting problem. SVM ensembles generated with Learn^{++} learning rule (SVMLearn^{++}) are capable of learning new information provided by subsequent datasets, including new knowledge provided by instances of previously unseen classes. Some knowledge is indeed forgotten while new information is being learned; however, this appears to be mild, as indicated by the steady improvement in the generalization performance. SVMLearn^{++} with two different kernel functions has been tested on one real world dataset and one benchmark dataset. The results demonstrate that SVMLearn^{++} work rather well in a variety of applications.

Learn^{++} suffers from the inherent "out-voting" problem when asked to learn new classes, which causes it to generate an unnecessarily large number of classifiers [20]. Therefore, in future work, we will test the modified version of Learn^{++}, called Learn^{++}.MT that attempts to reduce the number of classifiers generated.

Acknowledgements

This work is supported in part by the National Science Foundation under Grant No. ECS-0239090, "CAREER: An Ensemble of Classifiers Approach for Incremental Learning." Z.E. would like to thank Mr. Apostolos Topalis and Mr. Michael Muhlbaier graduate students at Rowan University, NJ, for their invaluable suggestions and assistance.

References

1. V. Vapnik, Statistical Learning Theory. New York: Wiley, 1998.
2. N. Cristianini, J. Shawe-Taylor, An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, Cambridge University Press, 2000.
3. S. Grossberg, "Nonlinear neural networks: principles, mechanisms and architectures," Neural Networks, Vol. 1, No. 1, pp. 17–61, 1988.
4. R. French, "Catastrophic forgetting in connectionist networks: Causes, Consequences and Solutions," Trends in Cognitive Sciences, vol. 3, no.4, pp. 128-135, 1999.
5. R. Polikar, L. Udpa, S. Udpa, V. Honavar, "Learn^{++}: An incremental learning algorithm for multilayer perceptrons." Proceedings of 25th. IEEE International Conference on Acoustics, Speech and Signal Processing, Vol. 6, pp: 3414-3417, Istanbul, Turkey, 2000.
6. R. Polikar, L. Udpa, S. Udpa, V. Honavar. "Learn^{++}: An incremental learning algorithm for supervised neural networks." IEEE Transactions on Systems, Man, and Cybernetics. Part C: Applications and Reviews, Vol. 31, No. 4, pp: 497-508, 2001.
7. Y. Freund, R. Schapire, "A decision theoretic generalization of on-line learning and an application to boosting," Computer and System Sciences, vol. 57, no. 1, pp. 119-139, 1997.
8. N. Kasabov, "Evolving Connectionist Systems: Methods and Applications in Bioinformatics, Brain Study and Intelligent Machines", Springer Verlag, 2002.
9. J. Platt, "Fast Training of Support Vector Machines using Sequential Minimal Optimization", Advances in Kernel Methods - Support Vector Learning, B. Schölkopf, C. Burges, and A. Smola, eds., MIT Press, 1998.
10. C. Domeniconi, D. Gunopulos, "Incremental Support Machine Construction", Proceedings of First IEEE Int. Conf. on Data Mining (ICDM 2001), pp. 589-592.
11. P. Mitra, C.A. Murthy, S.K. Pal, "Data condensation in large databases by incremental learning with support vector machines", Proceedings of 15th International Conference on Pattern Recognition, Vol.2, pp:708 – 711, 3-7 Sept 2000.
12. K. Li, H.-K. Huang, "Incremental learning proximal support vector machine classifiers", Proceedings of International Conference on Machine Learning and Cybernetics, vol. 3, pp:1635–1637, 4-5 November 2002.
13. J.-L. An, Z.-O. Wang, Z.-P. Ma, "An incremental learning algorithm for support vector machine", Proceedings of International Conference on Machine Learning and Cybernetics, Vol.2, pp:1153 – 1156, 2-5 November 2003.
14. Z.-W. Li; J.-P. Zhang, J. Yang, "A heuristic algorithm to incremental support vector machine learning", Proceedings of 2004 International Conference on Machine Learning and Cybernetics, Vol. 3, pp:1764–1767, 26-29 Aug. 2004.
15. D. Pavlov, J. Mao, and B. Dom, Scaling-up Support Vector Machines using The Boosting Algorithm, Proceedings of the International Conference on Pattern Recognition, Barcelona, Spain, September 3-7 2000, pp. 19-22.
16. G. Valentini, M. Muselli, and F. Ruffino, Cancer Recognition with Bagged Ensembles of Support Vector Machines, Neurocomputing 56(1), (2004), pp. 461-466.
17. G. Valentini, M. Muselli, F. Ruffino, Bagged Ensembles of SVMs for Gene Expression Data Analysis, Proceeding of the International Joint Conference on Neural Networks, Portland, OR, USA, July 20-24 2003, pp. 1844-1849.

18. H.-C. Kim, S. Pang, H.-M. Je, D. Kim, and S. Y. Bang, Constructing Support Vector Machine Ensemble, Pattern Recognition 36, (2003), pp. 2757-2767.
19. K. Duan, S.S. Keerthi, A.N. Poo, Evaluation of simple performance measures for tuning SVM hyperparameters, Neurocomputing, 51 (2003) 41-59.
20. M. Muhlbaier, A. Topalis, R. Polikar, Learn^{++}.MT: A New Approach to Incremental Learning, 5th Int. Workshop on Multiple Classifier Systems (MCS 2004), Springer LINS vol. 3077 , pp. 52-61, Cagliari, Italy, June 2004.