ESANN'2007 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 25-27 April 2007, d-side publi., ISBN 2-930307-07-2.

Kernel-based online machine learning and support vector reduction

Sumeet Agarwal[1], V. Vijaya Saradhi[2] and Harish Karnick[2]

1- IBM India Research Lab, New Delhi, India.

2- Department of Computer Science and Engineering,
Indian Institute of Technology Kanpur, Kanpur, India.

Abstract. We apply kernel-based machine learning methods to online learning situations, and look at the related requirement of reducing the complexity of the learnt classifier. Online methods are particularly useful in situations which involve streaming data, such as medical or financial applications. We show that the concept of *span* of support vectors can be used to build a classifier that performs reasonably well while satisfying given space and time constraints, thus making it potentially suitable for such online situations.

1 Introduction

Kernel-based learning methods [6] have been successfully applied to a wide range of problems. The key idea behind these methods is to implicitly map the input data to a new feature space, and then find a suitable hypothesis in this space. The mapping to the new space is defined by a function called the kernel function.

This paper uses kernels to build a classifier that can be used in situations with streaming data. Such a classifier should consume limited space, and should be able to update itself each time a new point (which it fails to classify correctly) comes in. The number of support vectors (SVs henceforth), that determines classifier complexity, keeps increasing with increasing amounts of training data [7]. The problem is to reduce this complexity in order to meet the space constraints, while minimizing the loss in generalization error. We look to address this problem based on a concept known as support vector span. Our experiments show that often it is possible to reduce classifier complexity without significant loss in performance.

The rest of this paper is organized as follows: Section 2 looks at the concept of support vector span, which allows us to bound the generalization error. Section 3 looks at the basic problem of support vector reduction. In Section 4, we describe the online learning framework, and present our proposed span-based algorithm, comparing its performance with a standard method for this setting. Finally, Section 5 summarizes the paper and gives the overall conclusions.

2 Span of Support Vectors

The concept of support vector span was defined by Vapnik and Chapelle [8]. They showed that this could be used to obtain a tight bound on the generalization error of a given support vector machine (SVM henceforth). Suppose

that we have a set of n SVs denoted by $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)$ where $\mathbf{x}_i$ denotes a feature vector and y_i the corresponding label (± 1 for binary classification). Also, let $\alpha^0 = (\alpha_1^0, ..., \alpha_n^0)$ be the set of coefficients of the SVs, for the optimal classification function (hyperplane). For any SV $\mathbf{x}_p$, we define:

$$\Lambda_p = \left\{ \sum_{i=1, i\neq p}^{n} \lambda_i \mathbf{x}_i : \forall i \neq p,\ \alpha_i^0 + y_i y_p \alpha_p^0 \lambda_i \geq 0;\ \sum_{i=1, i\neq p}^{n} \lambda_i = 1 \right\}$$

as a constrained linear combination of the other SVs. Note that λ_i can be negative. We then define S_p, which we call the span of the support vector $\mathbf{x}_p$, as the distance between $\mathbf{x}_p$ and Λ_p: $S_p^2 = d^2(\mathbf{x}_p, \Lambda_p) = min_{\mathbf{x}\in\Lambda_p} \|\mathbf{x} - \mathbf{x}_p\|^2$ The maximal value of S_p over all the SVs is called the S-span:

$$S = max_{p\in\mathbb{N}_n} S_p \tag{1}$$

This definition of span holds for the case when the training data is linearly separable (in the feature space); a slightly modified expression can be used to account for non-separable data as well. Vapnik and Chapelle show that $S \leq D_{SV}$, where D_{SV} is the diameter of the smallest sphere enclosing all the SVs. Using this definition of span, they derive the following bound:

$$Ep_{error}^{l-1} \leq E\left(\frac{SD}{l\rho^2}\right) \tag{2}$$

This means that the expected value of generalization error for an SVM trained on a training set of size $l-1$ is bounded by the expression on the right hand side, where D is the diameter of the smallest sphere enclosing the training data, S is the S-span of the support vector set, and ρ is the margin. These quantities are all taken for training sets of size l; the difference in size of one element comes in due to the fact that the bound is derived via a leave-one-out procedure on the training data.

Vapnik and Chapelle also demonstrate via experiment that in practice, this is a tight bound. We will employ their results to justify our span-based approach to support vector reduction. For a proof of the stated bound see the original paper [8].

3 Support Vector Reduction

Although SVMs offer state-of-the-art classification performance, they have not been used much for applications where time efficiency is crucial, such as in streaming or online settings. This is due to the fact that for large data sets SVMs return a large number of SVs, and both the space and time requirements of an SVM classifier scale linearly with the number of SVs. In fact, a recent result by Steinwart [7] shows that the number of SVs is bounded from below by a linear function in the size of the training set.

However, intuitively we expect that the support vector set should saturate at some level. Once we have enough SVs to uniquely determine the separating hyperplane further additions should be redundant. Thus, it is worthwhile to explore whether the number of SVs can be substantially pruned without significant loss in generalization ability.

Burges [1] has demonstrated that this is feasible, though his method was too time-consuming. Downs et al. [3] have proposed leaving out SVs that are linear combinations of others by adjusting the coefficients of those that remain.

For streaming and online applications the data and consequently the size of the training set keep increasing, and as per [7], the number of SVs will also increase at least proportionately. Clearly, a method is needed to update the classifier continuously. If there is a size limit of N on the SV set of the SVM classifier, then for every subsequent misclassified data point we must decide which one of the existing SVs (if any) will be replaced by the new point.

4 Streaming, Online Learning Applications

To formalize, assume we have a memory limit of N SVs. Initially, the memory is empty and training data points keep streaming in. We cannot retrain the SVM from scratch for every new data point since it would be prohibitively expensive to do so. One approach is to use some kind of gradient-descent to adjust the old coefficient (α_i) values based on the extent to which the new point was misclassified (note that points classified correctly can be ignored). Simultaneously, we must also compute the multiplier for the new point. This procedure continues as long as the number of SVs is less than N. Once we have N SVs, we cannot add a point to the set without first discarding an existing SV. We propose to use the S-span to decide which point to remove.

Let X_{old} be the set (of size N) of SVs, S_{old} the corresponding S-span, x the new misclassified data point and $x_p \in X_{old}$ the point that leads to the *least* S-span, say S_{new}, when x replaces x_p in X_{old} (see algorithm below). We replace x_p by x if $S_{new} < S_{old}$. The basic objective is to not increase the bound on the predicted generalization error at each step. From (2), it is clear that the expected value of the generalization error depends on the value of S, the span of the set of SVs. We cannot really control the variation in D and ρ as they will follow the same trend irrespective of which vectors we throw out. However, the variation in span is not bound to follow any such trend and therefore becomes the key determining factor. So, our idea is to try and replace points in the set of SVs so as to maximally reduce the S-span of the resulting set. The simplest replacement strategy would be to leave out the oldest point in memory; this is essentially what was done in the algorithm proposed by Kivinen et al. [4]. Although this has the advantage of implicitly accounting for time-varying distributions, it also runs the risk of discarding a more informative point in favor of a less useful one. The span-based idea is unlikely to encounter this difficulty. We say unlikely since we do not yet have a proof that the method will always work optimally. However, our experiments with synthetic and natural data sets suggest that the

approach seems to do well. The algorithm is summarized below:

1. Let the current set of SVs in the memory be $X_{old} = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N))$, and let their multipliers be $(\alpha_1, ..., \alpha_N)$ (we assume the memory is full; otherwise, one can keep adding the incoming SVs until it is full). Let $(\mathbf{x}, y)$ be an incoming data point. We first compute: $f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x})$ Here k is the usual kernel function.

2. If $f(\mathbf{x}) = y$, we ignore it, return to step 1 and wait for the next point.

3. Compute S_{old}, the $S-$span of X_{old}. Remove support vector $\mathbf{x}_i$ from X_{old} and include the new data point $\mathbf{x}$. Let the new set of SVs be: $X^i_{new} = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_{i-1}, y_{i-1}), (\mathbf{x}_{i+1}, y_{i+1}), \cdots, (\mathbf{x}_N, y_N), (\mathbf{x}, y)\}$. Compute the span corresponding to this set of SVs; call it S_i.

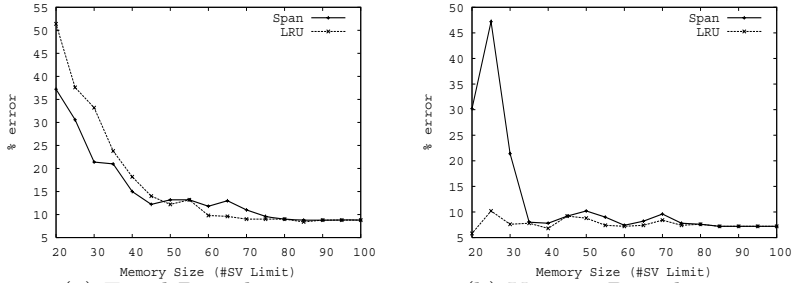
4. Repeat the above step for all N data points. Find the least $S-$span among the $S_i \; \forall \; i = 1, 2, \cdots, N$. Let this be S_p; the corresponding support vector removed is $\mathbf{x}_p$.

5. If $S_p < S_{old}$ then we remove x_p from the set by making $\alpha_p = 0$ while simultaneously adjusting the other coefficients in order to maintain the optimality conditions for the SVM. This is done via the Poggio-Cauwenberghs decremental unlearning procedure [2]. If $S_p \geq S_{old}$ do not include the new point in the SV set.

6. If $\mathbf{x}_p$ was removed in step 5, add the new point to the SV set, i.e. compute the appropriate multiplier for it. Once again the other α_i values must be adjusted so as to maintain optimality. Use the Poggio-Cauwenberghs incremental learning procedure [2] to carry out this adjustment.

7. We now have an updated set of N SVs, with the new point possibly replacing one of the earlier ones. Run the current classifier on an independent test set, to gauge its generalization performance. Return to step 1 and await the next point.

We experimented with both static and time-varying distributions of data. For the static case, a memory limit was fixed (smaller than the training set size), and all the training data was streamed in. The final classifier obtained was then tested on an independent test set drawn from the same distribution. This procedure was repeated for varying values of the memory limit. For the time-varying distribution case, the procedure used was similar, except that the training data's distribution varied gradually as it was streamed in. The test set was drawn from the current distribution, i.e. the distribution prevailing at the end of the training data stream. In this way, the ability of the algorithm to adapt to the variation could be gauged. We compared our algorithm's performance with the "Least Recently Used" algorithm (LRU henceforth), which leaves out the oldest point in memory whenever a new support vector comes in, and uses the same incremental learning and decremental unlearning procedures [2] for coefficient adjustment.



(a) Fixed Distribution (b) Varying Distribution

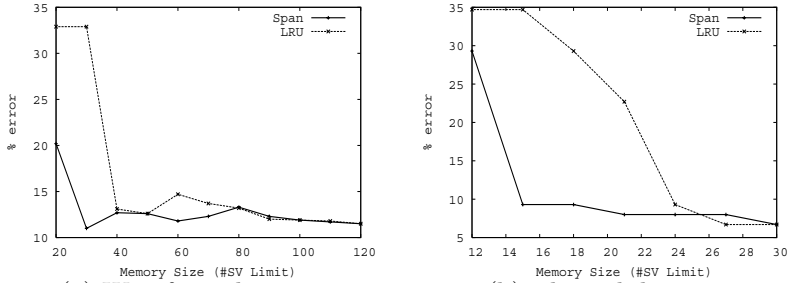
Fig. 1: In both (a) and (b), we have generated synthetic two-class data consisting of two overlapping 3-D Gaussian 'bells', with 5% Gaussian noise added in.

The results are shown in Figure 1. It is evident that for both methods, generalization error increases as the memory limit decreases; however, the span-based method is observed to be more consistent in its performance as the memory limit is reduced. For the static distribution the span-based method clearly outperforms LRU. For the time-varying distribution, the span method competes closely with LRU till a certain memory limit (around 35). When the memory limit is reduced even further the span method's performance breaks down. This is because the span-based bounds are derived based on the assumption that the data points are drawn from *independent and identical distributions*. The LRU method works better for lower memory limits, because the method is specifically designed to handle changing distributions. We have also experimented with the benchmark data sets thyroid and waveform [5]; these data sets were used as streaming data in our experiments. Results are given in Figure 2. In this case too the span-based algorithm is seen to perform better than LRU. On the whole, our proposed method gives encouraging results in an online setting. It is not as fast as the simple LRU approach; but for small memory sizes, it competes very well in terms of actual time taken.

The maximum number of SVs obtained in the case of the artificial data set is 89, when using the entire data set; using the span-based reduction technique, we got 35 to SVs without significantly increasing generalization error, a reduction of 61%. In the case of the real-world data sets, a maximum reduction of 71% of the SVs is achieved without noticeably affecting the error rate.

5 Conclusions and Future Work

This work proposes a new learning procedure for a kernel-based classifier for streaming, online data where the maximum size of the support vector set is fixed. We use the concept of the span of the set of SVs to decide which support vector to replace from the current set of support vectors when a new SV is



(a) Waveform data set (b) Thyroid data set

Fig. 2: Results on benchmark data sets: waveform (training size: 400; test size: 4600) and thyroid (training size: 140; test size: 75).

desired to be added. While we do not have a formal theoretical justification for the procedure, the experimental results are quite encouraging. Establishing a performance guarantee in the form of an error bound remains to be done.

References

[1] C. J. C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector machines. In *Neural Information Processing Systems*, 1997.

[2] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *Neural Information Processing Systems*, 2000.

[3] Tom Downs, Kevin E Gates, and Annette Masters. Exact simplification of support vector solutions. *Journal of Machine Learning Research*, 2:293–297, 2001.

[4] J. Kivinen, A. J. Smola, and R. C. Williamson. Online learning with kernels. *IEEE Transactions on Signal Processing*, 52(8):2165–2176, August, 2004.

[5] G. Rätsch. Benchmark repository. Technical report, Intelligent Data Analysis Group, Fraunhofer-FIRST, 2005.

[6] B. Schölkopf and A. J. Smola. *Learning with Kernels.* The MIT Press, Cambridge, MA, USA, 2002.

[7] I. Steinwart. Sparseness of support vector machines. *Journal of Machine Learning Research*, 4:1071–1105, 2003.

[8] V. Vapnik and O. Chapelle. Bounds on error expectation for SVM. *Neural Computation*, 12:2013–2036, 2000.