# Nearest Neighbor Prototyping for Sparse and Scalable Support Vector Machines

Technical Report CAL-2007-02

Bharath K. Sriperumbudur* and Gert Lanckriet

February 26, 2007



Computer Audition Laboratory (CAL) Technical reports are available on the CALs web page at http://cosmal.ucsd.edu/cal or you may contact us by mail at:

Prof. Gert Lanckriet
EBU1, Room 5604
University of California, San Diego
9500 Gilman Drive, Mail code 0407
La Jolla, CA 92093-0407

*Corresponding Author: bharathsv@ucsd.edu. The authors are with the Department of Electrical and Computer Engineering, University of California, San Diego, 9500 Gilman Drive, La Jolla, CA 92093.

**Abstract**

Training and evaluating support vector machines (SVMs) when large amounts of training data are available, is computationally expensive: for $n$ data points, exact optimization results in $O(n^3)$ time and $O(n^2)$ space complexity if general-purpose solvers are used, while prediction involves storing and accessing a significant subset of the training data, demanding time and space as well. Addressing these issues has been a topic of ongoing research. In this paper, we propose a novel approach, based on nearest neighbor prototyping techniques, to selectively remove patterns from the training set and perform SVM training with the reduced set. Empirical evidence shows that this simple approach sharply reduces the training complexity, while increasing the sparsity of the resulting SVM, without substantive loss in accuracy. The latter reduction in number of support vectors directly reduces the prediction complexity of the SVM.

# 1 Introduction

Support vector machines (SVMs) [1], currently a popular tool in machine learning, have attracted wide interest in classification and regression problems. In a standard two-class classification setting, given $n$ training patterns, $\boldsymbol{x}_i \in \mathbb{R}^d$, and their associated class labels, $y_i \in \{1, -1\}$ , the SVM decision function is given as

$$f(\boldsymbol{x}) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i^* y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b^*\right), \tag{1}$$

where $\alpha_i^*$ and $b^*$ are solutions to the following quadratic optimization problem (QP):

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & -\sum_{i=1}^{n} \alpha_i + \frac{1}{2}\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ \text{subject to} \quad & \sum_{i=1}^{n} y_i \alpha_i = 0, \qquad 0 \le \alpha_i \le C, \quad i = 1, \ldots, n. \end{aligned} \tag{2}$$

$\boldsymbol{\alpha}$ is a vector of $n$ variables, where each component $\alpha_i$ corresponds to a training pattern $(\boldsymbol{x}_i, y_i)$. The training patterns for which $0 < \alpha_i^* \le C$ are

called the support vectors (SVs). From Eq. (1), it is clear that only SVs determine the SVM decision function and the patterns with $\alpha_i^* = 0$ play no role for prediction, unlike a nearest neighbor (NN) rule [2] that requires all training patterns for prediction. $k(.,.)$ is the kernel function with $(\mathbf{Q})_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, where $\mathbf{Q}$ is the kernel matrix of size $n \times n$. Optimizing Eq. (2) using general-purpose interior-point solvers leads to $O(n^3)$ time and $O(n^2)$ space complexity, rendering SVMs impractical for large datasets, commonly encountered in data mining applications.

Apart from training complexity, the prediction complexity of Eq. (1), determined by the number of SVs, can easily become a bottleneck for systems that require predictions at high rate, e.g., in face recognition and/or tracking, check and ZIP code readers, etc. [3] indicates that the number of SVs increases linearly with the number $n$ of training patterns, making prediction complexity indeed an issue as well for large scale datasets.

In this paper, we propose a framework to reduce both training and prediction complexity of SVMs using NN prototyping techniques. This is based on two observations. First, training patterns in homogeneous regions, that can be recognized with high confidence, usually do not become SVs, while outliers and patterns close to the margin do. Second, although outliers will result in SVs, it is clear that patterns close to the margin play a more important role in determining the decision boundary. Therefore, we propose to use NN prototyping methods like *condensing, editing* and *replacement* to reduce the size of the training set by selectively removing patterns that are further away from the decision boundary. We empirically show that SVMs trained on the reduced training set provide fewer SVs than those obtained from the complete training set, without considerable loss in accuracy, thus reducing both the training and prediction complexity of SVMs.

# 2 Related work

Reducing training time complexity can be addressed by using efficient, special-purpose QP solvers and/or *data reduction methods*. Since our approach pertains to the latter category, we do not discuss related work in the former category. Data reduction methods can be categorized into two classes: *a priori reduction* and *a posteriori reduction* with respect to SVM training. A priori reduction deals with reducing the training data prior to training which benefits the QP problem in Eq. (2), whereas a posteriori reduction reduces

the number of SVs and thus the prediction complexity of Eq. (1).

## 2.1 A priori data reduction

One straightforward and ad-hoc way of reducing $n$ is to randomly down-sample the training dataset. [4] proposed the *reduced SVM* (RSVM), solving the primal SVM problem by randomly selecting a subset of the training data. The problem can be formulated as

$$\begin{aligned} \min_{\widetilde{\boldsymbol{\alpha}},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\left(\sum_{i,j=1}^{m}\widetilde{\alpha}_i\widetilde{\alpha}_j + b^2\right) + C\sum_{i=1}^{n}\xi_i^2 \\ \text{subject to} \quad & \sum_{j=1}^{m} y_i\widetilde{y}_j\widetilde{\alpha}_j k(\boldsymbol{x}_i,\widetilde{\boldsymbol{x}}_j) + y_i b \geq 1-\xi_i, \quad \xi_i > 0, \quad i=1,\ldots,n \end{aligned} \tag{3}$$

where $\widetilde{\alpha}_j$ is the Lagrange coefficient in the decision function for the reduced set of SVs, $\widetilde{\boldsymbol{x}}_j$, $j = 1, 2, \cdots, m$, with class label $\widetilde{y}_j$. $\xi_i$ is the slack variable for each training point, $\boldsymbol{x}_i$, $i = 1, 2, \cdots, n$. It has to be noted that the QP problem in Eq. (3) still has $n$ constraints. [5] studies four different implementations of RSVM and indicates the training time complexity to be $O(nm^2)$ per iteration. Using a kernel matrix $\widetilde{\mathbf{Q}}$ of size $n \times m$, for $m \ll n$, both training time and space complexity are very small. However, choosing $m$ is of course tricky. In most cases it is chosen to be 10% of the training data size. This is illogical as $m$ should somehow be dataset dependent in order to capture the number of classes, inter-class behavior, etc. In contrary to the results of [4], [5] shows the classification accuracy of SVM trained on all training data to be better than that of RSVM. Also, [4] shows the performance of RSVM using $k(\widetilde{\boldsymbol{x}}_i, \widetilde{\boldsymbol{x}}_j)$ instead of $k(\boldsymbol{x}_i, \widetilde{\boldsymbol{x}}_j)$ in Eq. (3), i.e., using an $m \times m$ kernel matrix, to be significantly worse than full SVM training. Training on randomly down-sampled data thus performs significantly worse than training on all training data.

Cascade SVM [6] is another down-sampling method wherein each randomly drawn subset of the original training set trains a SVM. This method involves a hierarchical training structure where, at each level of hierarchy, the SVs from the previous level are used as training patterns on the next level. This is repeated until the final level (with only one SVM classifier) is reached. The SVs of the last iteration are fed back to the leaf nodes thus completing a cycle. Although no single SVM has to deal with the entire training set,

multiple SVM training is required and if the global optimum is not reached, multiple passes over the hierarchy are required which is time-consuming.

Cross-training [7] is an empirical attempt to endow SVMs with the properties of the MULTIEDIT algorithm [8]. It creates $s$ subsets of the training set with $r$ patterns each and trains independent SVMs on each subset. The decision functions of these SVMs are then used to discard two types of training examples: those which are confidently recognized and those which are misclassified outliers. Overall, the removal procedure aims at creating a separable set of training examples without modifying the location of the decision boundary. Making the problem separable breaks the linear dependency between the number of SVs and the number of training patterns. However, similar to cascade SVMs, this method also involves multiple SVM training which might be time-consuming.

## 2.2 A posteriori data reduction

Several techniques aim to reduce the prediction complexity of SVMs by expressing Eq. (1) with a smaller kernel expansion. Burges [9] proposed a *reduced set method* to find a pre-defined number of *synthetic* vectors in order to define a compact approximation of the decision function in Eq. (1). This method can be stated as an unconstrained non-convex minimization problem and [9] shows that the use of the second degree homogeneous polynomial kernel allows a closed form solution to this minimization. However, this method is not applicable to arbitrary inputs such as graphs or strings. [10] suggests an $\ell_1$-penalization method for reduced set selection by sufficiently approximating the SVM decision function which leads to solving a QP.

In this paper, we propose an a priori data reduction scheme, which unlike ad-hoc random down-sampling, is motivated by the fact that the training patterns close to decision boundary are sufficient to define it while the rest can be removed without affecting the decision. Using NN prototyping methods, we capture such a set of points, which are then used to train a SVM. With this simple technique, we achieve significant reduction in both training data size and number of SVs without substantive loss in accuracy compared to using all training data.

# 3 Nearest neighbor prototyping

The NN classifier [2] is one of the most popular non-parametric supervised classification methods. Despite its simplicity, effectiveness and theoretical results on its asymptotic performance, practical use of this prediction rule has been historically limited due to the computational costs involved. Compared to SVMs, the NN rule does not have any training time complexity (as it is a lazy learning rule) but has high prediction complexity as all the training patterns are used. Since the patterns that lie closer to the decision boundary are sufficient to define the boundary, redundant patterns can be removed without affecting the NN prediction. The methods to find these patterns/prototypes can be categorized into *condensing*, *editing* and *replacement* rules.

## 3.1 Prototype condensing

Condensing rules reduce the size of training data by discarding the points that are further from the NN decision boundary. These rules can be classified as *training-set consistent* and *decision-boundary consistent*. Training-set consistent rules generate a subset of the training set that allows the original training set to be NN-classified with no error. On the other hand, decision-boundary consistent rules generate a subset of the training set such that the NN-boundary is not altered and are thus also training-set consistent. Finding decision-boundary consistent subsets is computationally more involved than finding training-set consistent subsets and both are not guaranteed to be minimal. We refer the interested reader to [11] for an in-depth overview on NN condensing rules.

Hart [12] proposed the first condensing scheme, named *Condensed NN rule* (CNN) which is training-set consistent. The algorithm is very simple. Let $\mathcal{C}$ denote the desired consistent subset. Initially $\mathcal{C}$ is empty. A random element from $\{\mathbf{X}, \boldsymbol{y}\} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_n, y_n)\}$ is transferred to $\mathcal{C}$ and $\mathcal{C}$ is used as a classifier with the 1-NN rule to classify all the remaining data in $\{\mathbf{X}, \boldsymbol{y}\}$. During this scan of $\{\mathbf{X}, \boldsymbol{y}\}$, whenever an element is incorrectly classified by $\mathcal{C}$, it is transferred from $\{\mathbf{X}, \boldsymbol{y}\}$ to $\mathcal{C}$. Scanning $\{\mathbf{X}, \boldsymbol{y}\}$ is repeated until no element is transferred from $\{\mathbf{X}, \boldsymbol{y}\}$ to $\mathcal{C}$ during a complete pass of the remaining data in $\{\mathbf{X}, \boldsymbol{y}\}$. The motivation for this heuristic is the intuition that data far from the decision boundary are not needed and that if an element is misclassified it must lie close to the decision boundary. With complexity of $O(n^2)$ per iteration, CNN is the simplest of all the condensing

rules proposed. Although this method works well for homogeneous clusters, it works poorly when there is a large overlap between the pattern distributions from different classes. But, we argue in our study that, although the condensed set might not capture the decision boundary sufficiently accurate for 1-NN prediction, it is accurate enough to give rise to SVM performance comparable to using all training data.

Toussaint *et al.* [13] proposed a decision-boundary consistent method of NN condensing called *Voronoi condensing*. Each point $\boldsymbol{x}_i$ in $\{\mathbf{X}, \boldsymbol{y}\}$ is marked if all its Voronoi neighbors have the same class label as $y_i$ and these marked points are discarded. The resulting set is decision-boundary consistent. Since the dual of a Voronoi diagram is Delaunay triangulation, proximity graphs [14] can be used to perform Voronoi condensing. This involves forming a proximity graph on $\{\mathbf{X}, \boldsymbol{y}\}$ and removing the vertices which have all of their neighbors belonging to their own class. Since computing a Delaunay graph is computationally intensive, a sub-graph, called Gabriel graph (GG), is used for condensing the training data. We show that, although GG condensed subsets provide good 1-NN performance, the condensation obtained is very small when the patterns are high dimensional. Also, with an average complexity of $O(dn^3)$, where $d$ is the pattern dimension, GG condensing is computationally more intensive than applying CNN.

## 3.2 Prototype editing

Editing methods aim at improving the recognition accuracy rather than at data reduction. The *Edited NN rule* [15] discards all training patterns that are misclassified when applying the 1-NN rule using all $n - 1$ remaining patterns of the training set. This simple editing rule is so powerful that the error rate of the 1-NN rule using the edited subset converges to the Bayes error. Devijer and Kittler [8] showed that the repeated application of Wilson's editing, called MULTIEDIT, will lead to the Bayes error rate. We show in our experiments that, although these methods alone are not suitable for the problem at hand, editing followed by condensing provides substantial reduction in training set size with minor loss in accuracy. Since editing rules are in essence data cleaning techniques, removing noisy patterns, they create more homogeneous classes, upon which condensing rules perform well.

## 3.3 Prototype replacement

As opposed to the aforementioned methods, that select a subset of the training data as the reduced set, prototype replacement methods generate new patterns that do not belong to the training set but form representative patterns. Chang [16] proposed the first such algorithm which repeatedly merges the two nearest neighbors of the same class as long as this merger does not increase the error rate of the training set. One drawback of Chang's method is that it may yield prototypes that do not characterize well the training set in terms of generalization. Another method is to learn a predefined number of prototypes by learning vector quantization (LVQ) [17], which is a competitive learning scheme. In this paper, we propose to use supervised clustering (SC) involving $k$-means to cluster the training data for prototype generation. Essentially, this is equivalent to modelling each of the class distributions as a mixture of Gaussians. We show that the prototypes from this simple method provide better reduction in training and prediction complexity of SVM without significant degradation in accuracy. This method works even better when applied on the edited data.

## 3.4 Empirical analysis of nearest neighbor prototyping methods

The NN prototyping methods are studied and evaluated on small benchmark datasets (2-class) from the UCI repository[1], before applying the most promising combination to the large datasets of interest. Australian (551/139/14), Ionosphere (280/71/34), Ringnorm (300/7100/20), Ripley (250/1000/2), Tic-Tac-Toe (765/193/9), Twonorm (300/7100/20), Vowel (528/462/10) (considering males vs females) and WDBC (454/115/30) are considered (with $n_{tr}/n_{te}/d$ representing the number of training patterns, $n_{tr}$, number of test patterns, $n_{te}$, and dimension, $d$). The training data is scaled to $[-1, 1]$ and the test data is scaled accordingly. Initially, a simple 1-NN rule involving Euclidean distance is used to classify the test patterns. For Wilson editing, the edited set is computed by using a $k$-NN rule with $k$ found by cross-validation (CV) over $k$ varying from 1 to 10. For prototype replacement by SC involving $k$-means, the value of $k$ has to be known beforehand. Instead of choosing some random value for $k$, we choose $k$ to be the reduced set size obtained

[1] ftp://ftp.ics.uci.edu/pub/machine-learning-databases

Table 1: Performance of nearest neighbor (NN), prototype condensing (condensed NN (CNN), Gabriel graph (GG)), prototype editing (Wilson edit, MULTIEDIT (M-Edit)) and prototype replacement (Chang method, supervised clustering (SC) by $k$-means) algorithms for various benchmark datasets. $E_p$ represents the test set error (%) using 1-NN and $l$ represents the size of the reduced training set (in italic). $n$ is the size of the training set.

| Dataset<br>$n$ | NN<br>$E_p, l$ | CNN<br>$E_p, l$ | GG<br>$E_p, l$ | Wilson<br>$E_p, l$ | M-Edit<br>$E_p, l$ | Chang<br>$E_p, l$ | SC<br>$E_p, l$ |
|---|---|---|---|---|---|---|---|
| Australian | 15.83 | 17.12 | 15.83 | 16.04 | 14.32 | 15.11 | 17.48 |
| *551* | *551* | *195* | *547* | *465* | *392* | *174* | *195* |
| Ionosphere | 16.90 | 17.04 | 16.90 | 19.86 | 24.79 | 14.09 | 19.30 |
| *280* | *280* | *65* | *229* | *239* | *177* | *63* | *65* |
| Ringnorm | 35.68 | 25.12 | 35.68 | 46.23 | 49.61 | 17.14 | 27.06 |
| *300* | *300* | *98* | *300* | *196* | *157* | *68* | *98* |
| Ripley | 14.50 | 16.65 | 15.40 | 11.19 | 9.35 | 15.30 | 14.02 |
| *250* | *250* | *66* | *91* | *215* | *184* | *60* | *66* |
| Tic-Tac-Toe | 0.00 | 4.72 | 0.00 | 0.00 | 20.46 | 6.22 | 2.75 |
| *765* | *765* | *108* | *765* | *751* | *511* | *88* | *108* |
| Twonorm | 6.44 | 11.03 | 6.44 | 5.70 | 4.97 | 9.39 | 4.44 |
| *300* | *300* | *74* | *299* | *275* | *231* | *55* | *74* |
| Vowel | 18.62 | 24.13 | 18.62 | 19.04 | 16.32 | 24.46 | 20.67 |
| *528* | *528* | *49* | *384* | *524* | *444* | *33* | *49* |
| WDBC | 3.48 | 7.04 | 3.48 | 4.52 | 6.35 | 7.83 | 6.26 |
| *454* | *454* | *62* | *284* | *435* | *392* | *40* | *62* |

from CNN. Although not an optimum value for $k$, it helps to compare the performance of CNN with SC. One might question the validity of SC as the generated prototypes often lie further away from the decision boundary. This would be an issue when the class distributions overlap significantly. However, it turns out that using SC on the edited set provides better generalization performance. We refer the interested reader to [18] for an in-depth study, comparing eleven methods of prototype selection.

Table 1 shows the performance (test set error using 1-NN and reduced training set) of various NN prototyping algorithms. It can be seen that CNN provides significant training set size reduction while GG performs similar to NN and provides a slight or no reduction in $n$. Since, in high dimensions, almost all patterns are neighbors to each other, GG selects all of them and the obtained reduction is small as is the loss in the NN performance. Except for the Ripley dataset, which is two-dimensional, GG indeed provides

negligible reduction. Therefore, we do not consider this method any further. Amongst the replacement methods, Chang's method provides good reduction, slightly better than SC, at comparable average test error. However, Chang's method has complexity $O(n^2)$ per iteration, while SC is cheaper as it only requires $O(nk)$ per iteration. Therefore, we prefer SC over Chang's method in generating prototypes.

As expected, editing rules don't achieve significant data reduction and are not directly useful by themselves. Table 1 shows Wilson editing performing similarly or better than NN as it generates homogeneous clusters by removing noisy patterns. Essentially, editing rules are data cleaning methods and therefore behave complementary to condensing rules. We expect the combination of these rules with editing followed by condensing to provide better data reduction. This approach will not work when combining MULTIEDIT with condensing rules because MULTIEDIT removes many points near the boundary making the classes completely separable. Subsequent application of a condensing rule would provide too few points to characterize the boundary resulting in poor performance.

Table 2 shows the performance of Wilson editing followed by CNN and SC on various benchmark datasets. Notice that these combo-methods provide better reduction in training set size than CNN or SC individually (see Table 1). This is because, as mentioned before, Wilson editing removes noisy outliers and makes the classes homogeneous (equivalent to truncating the class distributions), such that subsequent application of condensing rules effectively captures the boundary.

Based on the observations made so far, we consider CNN, SC, Wilson editing followed by CNN (W-CNN) and Wilson editing followed by SC (W-SC) as successful data reduction methods, which can now be applied to large-scale datasets to improve SVM scalability and sparsity.

# 4 Sparse and scalable support vector machines

As mentioned before, our final goal is to reduce the training and prediction complexity of SVMs by reducing the training set size using NN prototyping techniques rather than ad-hoc, random down-sampling (e.g., in RSVM). While [7] has a similar motivation, it involves training multiple SVMs. We propose to pre-process the training data (using condensing/replacement/editing), followed by a single SVM run. Although the results in Section 3.4 are

Table 2: Performance of Wilson editing followed by CNN (W-CNN) and Wilson editing followed by SC (W-SC) using 1-NN for testing; SVM on all training data, CNN (C-SVM), SC (SC-SVM), W-CNN (W-C-SVM) and W-SC (W-SC-SVM) using SVM for testing, for various benchmark datasets. $E_p$ represents test set error (%). $l$ and $q$ represent the size of the reduced training set and the number of support vectors (in italic), respectively. $n$ is the size of the training set.

| Dataset | W-CNN | W-SC | SVM | C-SVM | SC-SVM | W-C-SVM | W-SC-SVM |
|---|---|---|---|---|---|---|---|
| $n$ | $E_p, l$ | $E_p, l$ | $E_p, q$ | $E_p, q$ | $E_p, q$ | $E_p, q$ | $E_p, q$ |
| Australian | 15.97 | 14.68 | 12.23 | 13.70 | 13.38 | 13.70 | 13.31 |
| *551* | *28* | *28* | *216* | *189* | *131* | *26* | *17* |
| Ionosphere | 20.14 | 17.47 | 8.45 | 4.65 | 6.62 | 8.45 | 9.39 |
| *280* | *28* | *28* | *99* | *41* | *46* | *25* | *16* |
| Ringnorm | 27.94 | 36.2 | 1.96 | 2.87 | 2.82 | 3.15 | 3.22 |
| *300* | *23* | *23* | *243* | *42* | *90* | *17* | *14* |
| Ripley | 9.98 | 9.38 | 10.20 | 10.30 | 11.38 | 10.01 | 9.55 |
| *250* | *15* | *15* | *84* | *53* | *37* | *11* | *13* |
| Tic-Tac-Toe | 5.39 | 2.95 | 0.00 | 2.38 | 1.30 | 2.21 | 1.81 |
| *765* | *101* | *101* | *69* | *76* | *53* | *72* | *51* |
| Twonorm | 10.11 | 3.73 | 2.48 | 4.10 | 3.71 | 3.60 | 3.44 |
| *300* | *40* | *40* | *107* | *67* | *74* | *32* | *30* |
| Vowel | 25.95 | 20.76 | 25.11 | 25.37 | 22.53 | 26.06 | 21.30 |
| *528* | *41* | *41* | *200* | *48* | *45* | *41* | *40* |
| WDBC | 7.13 | 5.74 | 3.48 | 4.61 | 3.65 | 4.61 | 3.31 |
| *454* | *32* | *32* | *41* | *45* | *13* | *27* | *10* |

promising in terms of reduction, they assess the generalization performance based on 1-NN. Before addressing large-scale SVM problems, we validate the best prototyping techniques and their combinations for SVM classification and compare them to a SVM trained on all data.

We conduct SVM experiments on the datasets mentioned in Section 3.4. Since all the NN experiments are carried out in MATLAB, the SVM toolbox for MATLAB[2] is used. The parameter $C$ in Eq. (2) and $\gamma$ are obtained by CV over $C = \{2^{13}, 2^{12}, \cdots, 2^{-5}\}$ and $\gamma = \{2^5, 2^4, \cdots, 2^{-13}\}$, where $\gamma$ is the bandwidth parameter of the radial-basis kernel, $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\gamma||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}$. Ignoring the $b$ term in Eq. (1), it can seen that when $\gamma \rightarrow 0$, SVM implements 1-NN rule. If all training data are used to learn the optimal SVM, 5-fold

[2]http://ida.first.fraunhofer.de/ anton/software.html

Table 3: Performance of SVM, C-SVM, SC-SVM, W-C-SVM and W-SC-SVM on the Adult dataset. $E_p$ represents test set error (%). $l$ and $q$ represent the size of the reduced training set (in italic) and the number of support vectors (in bold), respectively. $n_{tr}$ and $n_{te}$ represent the size of the full training and test set, respectively.

| Adult Dataset | | SVM | C-SVM | SC-SVM | W-C-SVM | W-SC-SVM |
|---|---|---|---|---|---|---|
| $n_{tr}$ | $n_{te}$ | $E_p, (q)$ | $E_p, (l, q)$ | $E_p, (l, q)$ | $E_p, (l, q)$ | $E_p, (l, q)$ |
| 1444 | 161 | 21.74 (**926**) | 22.98 (*537*, **514**) | 21.12 (*537*, **358**) | 21.12 (*199*, **195**) | 22.36 (*199*, **90**) |
| 2037 | 228 | 19.30 (**1263**) | 18.86 (*766*, **740**) | 16.67 (*766*, **490**) | 19.74 (*216*, **211**) | 20.36 (*216*, **87**) |
| 2865 | 320 | 18.44 (**1635**) | 17.81 (*1072*, **994**) | 15.63 (*1072*, **641**) | 19.87 (*264*, **250**) | 20.94 (*264*, **96**) |
| 4302 | 479 | 16.70 (**2266**) | 17.33 (*1565*, **1408**) | 18.16 (*1565*, **945**) | 16.91 (*424*, **381**) | 20.04 (*424*, **139**) |
| 5772 | 642 | 21.50 (**3027**) | 20.41 (*2113*, **1866**) | 19.94 (*2113*, **1203**) | 19.78 (*566*, **481**) | 20.09 (*566*, **153**) |
| 10098 | 1123 | 17.36 (**4745**) | 17.81 (*3615*, **3072**) | 18.66 (*3615*, **1898**) | 19.41 (*904*, **753**) | 20.30 (*904*, **245**) |
| 14490 | 1611 | 17.26 (**6656**) | 16.51 (*5368*, **4370**) | 18.93 (*5368*, **3781**) | 18.25 (*1370*, **1091**) | 17.81 (*1370*, **317**) |

cross validation is used to compute the optimal $(C, \gamma)$. If the training set is reduced before training, the eliminated training data is used as validation set. Finally, the test data is classified after training a SVM for the optimal $(C, \gamma)$, on the full or reduced training set.

Table 2 shows the performance (test error (%) and number of support vectors) of a SVM trained on all data versus a reduced training set, obtained from CNN, SC, W-CNN and W-SC, on benchmark datasets. The results suggest that reduced-set training causes little performance loss (1.5% on the average) while yielding significantly less SVs, especially for the editing-condensation combination. The latter achieves around 90% data reduction before SVM training and requires only about 6% of the patterns as SVs, thus reducing both training and prediction complexity of the SVM.

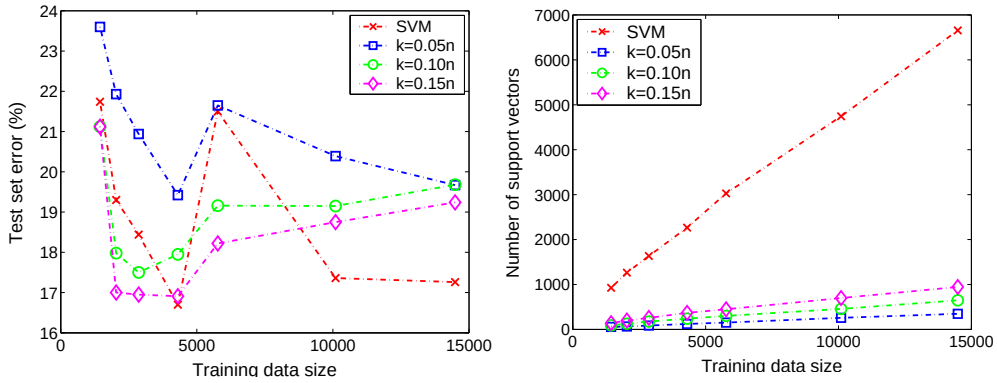
Table 3 evaluates the performance of aforementioned methods on a dataset of larger scale, the Adult[1] dataset. It can be seen that the C-SVM and SC-SVM error-rate is similar to SVM, using only 37% of the original training patterns (on average) and yielding 50% and 60% fewer SVs. The editing-condensing combination improves this further to about 90% reduction in

training data with 83% fewer SVs than a SVM trained on all data. The slight decrease in classification accuracy is permissible as, for even larger datasets, SVM training based on all data will become simply impossible. Moreover, the reduction in SVs makes the resulting predictor amenable for real-time systems that require fast prediction. Although a more efficient SVM implementation like LibSVM [19] would have allowed experiments on the full Adult dataset (32562 points), this in itself is not crucial to illustrate that NN prototyping algorithms show significant promise in reducing large training datasets, improving the scalability and sparsity of SVMs. Using a more efficient, special-purpose QP solver is complementary to the proposed pre-processing techniques, which will directly benefit it.

Throughout our study, we used the value of $k$, for SC, obtained from CNN (usually around 37% of $n_{tr}$, for the Adult dataset). We now investigate the effect of $k$ on the test error and the number of SVs, for subsets of the Adult dataset of varying size. $k$ is chosen to be 5%, 10% and 15% of $n_{tr}$ and the reduced training set is used for SVM training. Figure 1(a) and Figure 1(b) show plots of the test error (%) and the number of SVs, respectively, against dataset size, including results for different $k$ along with a SVM trained on all training data. Notice that the test set error decreases for increasing $k$ and $k = 10\%$ of $n_{tr}$ seems to provide a good trade-off between number of SVs and test error rate. Figure 1(b) shows the number of SVs (for SC-SVM) increases very slowly with the dataset size, unlike the linear increase for a SVM trained on all data. The drawback of SC is that the number of clusters need to be known beforehand unlike CNN or Wilson edit. So, the best way to use SC is in conjunction with CNN or Wilson edit.

# 5 Discussion & Conclusion

Using NN prototyping techniques we can significantly improve the training and prediction complexity of SVMs, by reducing the training set size and the number of support vectors. The proposed methods are validated on several datasets, and only result in a slight loss of accuracy. The proposed methods can directly be integrated with efficient, special-purpose SVM solvers. Even the most efficient QP solvers for SVMs will directly benefit from this approach as dataset sizes will soon outgrow the capacity of any such solver to deal with a given dataset in its entirety. We argue that the proposed methods are more appropriate than ad-hoc random pattern selection methods.




(a) (b)

Figure 1: Comparison of (a) test set error (%) and (b) number of SVs for subsets of varying size of the Adult dataset. Shown are results for SVM trained on complete data and SC-SVM (using $k = 5\%, 10\%$ and $15\%$ of $n = n_{tr}$).

**Acknowledgments**

This work was supported by the Fair Isaac Corporation and the University of California MICRO program.

# References

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

[2] E. Fix and J. Hodges. Discriminatory analysis, non-parametric discrimination: Consistency properties. Tech. Report 4, USAF School of Aviation Medicine, Randoph Field, Texas, 1951.

[3] I. Steinwart. Sparseness of support vector machines–some asymptotically sharp bounds. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, Cambridge, MA, 2004. MIT Press.

[4] Y.-J. Lee and O. L. Mangasarian. RSVM: Reduced support vector machines. In *Proc. of the First SIAM International Conference on Data Mining*, Chicago, IL, April 2001.

[5] K.-M. Lin and C.-J. Lin. A study on reduced support vector machines. *IEEE Trans. Neural Networks*, 14(6):1449–1459, 2003.

[6] H.P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik. Parallel support vector machines: The cascade svm. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 521–528, Cambridge, MA, 2005. MIT Press.

[7] G. H. Bakir, L. Bottou, and J. Weston. Breaking svm complexity with cross-training. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 81–88, Cambridge, MA, 2005. MIT Press.

[8] P. Devijer and J. Kittler. *Pattern Recognition: A Statistical Approach.* Prentice-Hall, NJ, 1982.

[9] C. Burges. Simplified support vector decision rules. In *Proc. of 13th International Conference on Machine Learning*, Bari, Italy, 1996.

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

[11] L. Devroye, L. Gyorfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition.* Springer-Verlag, New York, 1996.

[12] P. E. Hart. The condensed nearest neighbor rule. *IEEE Trans. Information Theory*, 14:515–516, 1968.

[13] G. T. Toussaint and R. S. Poulsen. Some new algorithms and software implementation methods for pattern recognition research. In *Proc. IEEE International Computer Software Applications Conference*, pages 55–63, Chicago, IL, 1979.

[14] J. W. Jaromczyk and G. T. Toussaint. Relative neighborhood graphs and their relatives. *Proc. of IEEE*, 80(9):1502–1517, 1992.

[15] D. L. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Trans. Systems, Man and Cybernetics*, 2:408–420, 1972.

[16] C. L. Chang. Finding prototypes for nearest neighbor classifiers. *IEEE Trans. Computers*, 23:1179–1184, 1974.

[17] T. Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, Berlin, Germany, 1989.

[18] J. C. Bezdek and L. Kuncheva. Nearest prototype classifier designs: An experimental study. *Int. J. Intell. Syst.*, 16(12):1445–1473, 2001.

[19] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.