Chunking-Coordinated-Synthetic Approaches to Large-Scale Kernel Machines

Francisco J. González-Castaño and Robert R. Meyer[1]

Departamento de Ingeniería Telemática, Universidad de Vigo. ETSI Telecomunicación, Campus, 36200 Vigo, Spain. Phone: +34 986 813788. FAX: +34 986 812116.
javier@det.uvigo.es, rrm@cs.wisc.edu

Abstract. We consider a kernel-based approach to nonlinear classification that coordinates the generation of "synthetic" points (to be used in the kernel) with "chunking" (working with subsets of the data) in order to significantly reduce the size of the optimization problems required to construct classifiers for massive datasets. Rather than solving a single massive classification problem involving all points in the training set, we employ a series of problems that gradually increase in size and which consider kernels based on small numbers of synthetic points. These synthetic points are generated by solving and combining the results of relatively small nonlinear unconstrained optimization problems. In addition to greatly reducing optimization problem size, the procedure that we describe also has the advantage of being easily parallelized. Computational results show that our method efficiently generates high-performance simple classifiers on a problem involving a realistic dataset.

KEYWORDS: classification, support vector machines, kernel machines.

TRACK: Paper track

CONFERENCE TOPIC: Machine Learning, Knowledge Discovery and Data Mining

[1] Computer Sciences Department, University of Wisconsin – Madison. 1210 W. Dayton St., Madison WI 53706, USA. Phone: +1 608 262 1204. FAX: +1 608 262 9777.

1. Introduction

Suppose that the following classification problem is given:

A set of m training points in a n-dimensional space is given by the $m \times n$ matrix Y. These points are divided into two classes, types 1 and -1. A classifier is to be constructed using this data and a nonlinear kernel function K that is a mapping from $R^n \times R^n$ to R. We assume that construction of the corresponding classifier involves generating a function $g(x) = w K(C,x)$, where w is a row vector of size s and C is a set of s points in R^n and it is understood that, for such a set of points C, $K(C,x)$ is a column vector of size s whose entries are given by $K(c_i,x)$ for the rows c_i of C (in a further extension of this notation below, we will assume that for any two matrices A and B each of which has n columns, that $K(A,B)$ is a matrix whose (i,j) entry is $K(A_i, B_j)$ where A_i and B_j are the rows corresponding to indices i and j. In addition, we will use a juxtaposition of vectors and matrices to indicate a product operation, without introducing an extra symbol to indicate the transposition of the vector that may be required). The points used in a set C will be termed *classifier points*. While it is customary to set $C=Y$, one of the goals of this paper is to investigate alternative approaches to constructing C, particularly when Y is a large dataset. In conjunction with the determination of the weights w described above, a constant γ is also computed so that the resulting classifier designates (as correctly as possible) a point as type 1 if $g(x)>\gamma$ and a point as type -1 if $g(x)<\gamma$.

We assume the kernel *Mercer inner product property* [2] $K(y,x)= f(y)f(x)$, where f is a mapping from R^n to another space R^k, and $f(y)f(x)$ represents an inner product (for a simple example, suppose that n=2 and $K(y,x)=(yx)^2$. Then by taking $f(z)= (z_1^2, 2^{1/2} z_1 z_2, z_2^2)$, where z is a 2-vector, it is easy to verify that $K(y,x)= f(y)f(x)$ holds for any pair of vectors). The original space R^n is finite dimensional and is referred to as "input space" whereas R^k may have a much higher dimension (or even be infinite-dimensional) and is referred to as "feature space" (in the simple example given, n=2 and k=3, but note that even for this simple quadratic kernel the dimension of the feature space is proportional to the square of the dimension of the input space). Many commonly used nonlinear classifiers (such as Gaussians or homogeneous polynomials) have this Mercer property, and although evaluation of the corresponding function f itself is in general not computationally practical, the inner product representation $K(y,x)= f(y)f(x)$ allows classifier approximation problems to be formulated in an manner that is computationally tractable. In fact, by taking advantage of this Mercer property, we will be able to solve via a relatively small unconstrained nonlinear program (NLP), the problem of approximating a classifier with $C=Y$ (or subsets of Y) by classifiers based on very small numbers of "synthetic points" that are not necessarily in Y.

Classifiers are usually constructed by taking the set C to be the set Y of training points and then solving an optimization problem for the values of w and γ. However, for massive datasets this choice of C results in intractably large optimization problems. Moreover, it may be the case that even for medium-sized datasets, the classifiers using $C=Y$ can be outperformed by alternative choices of C. For example, if the type 1 points form a cluster about a center point z that is not in Y, and the type -1 points lie beyond this cluster, then an ideal classifier may be constructed using a Gaussian kernel involving the point z alone rather than any of the many points from Y. As observed in a noisy dataset below, the generalizability associated with alternative smaller choices of C may also be better than the choice $C=Y$, since the latter can lead to overtraining.

In the initialization step of our method we construct classifiers using C's that are small subsets of Y, and then in successor steps we approximate classifiers for successively larger subsets of the training set by using C's corresponding to even smaller sets of synthetic points (as opposed to using points from Y). We describe this procedure in the next section. It should be noted that this chunking strategy of considering successively larger subsets is one of the elements that differentiates this research from that of [2], which emphasizes synthetic point generation as an *a posteriori* procedure applied to simplify a classifier obtained in the standard manner from the full dataset. A second difference is our coordination of the results of parallel optimization problems for the synthetics.

The idea of simplifying classifiers by using synthetic points was introduced in [2]. Alternative approaches for generating synthetic points are described in [3,13]. Other approaches that seek to reduce kernel-classifier complexity are discussed in [11]. In the linear case, classifiers have also been simplified by means of feature selection via concave programming [1].

2. Chunking-Coordinated-Synthetic (CCS) approaches

Our approaches are based on a sequence of classifier generating problems interspersed with a sequence of classifier approximation problems (that generate synthetic points). The former we refer to as classifier problems and the latter, as approximation problems. We now define the format of the problems used in the specific implementation presented below.

LP(S, C) is the linear programming classifier generating problem:

$$\begin{aligned} &\text{Min}_{u,\gamma,E} \quad \nu\,||E||_1 + ||u||_1 \\ &\text{s.t.} \quad D(uK(C,S) - \gamma\mathbf{1}) + E \geq \mathbf{1},\ E \geq \mathbf{0}, \end{aligned} \qquad (\mathbf{1})$$

where ν is a weighting factor, $uK(C, S)$ is the vector obtained by applying the kernel corresponding to $uK(C, \bullet)$ to each element of the subset S of the training set, D is a diagonal matrix of ± 1's corresponding to the appropriate classification of the corresponding element of S**,** E is a vector of "errors" (with respect to the "soft" margin bounding surfaces corresponding to $u K(C, x)= \gamma \pm 1$), $\mathbf{1}$ is a vector of 1's, and $\mathbf{0}$ is a vector of 0's. Note that the standard classification problem in this framework would be obtained by setting $C=S=Y$, but this leads to a problem with $O(m)$ constraints and variables, which is intractable if m is large. Other classifier problems (such as classifiers with quadratic objective terms) may be substituted for the LP in (1). In particular, in future research we will experiment with further reductions in problem size via the use of *unconstrained* classifier models as described in section 4.

Using the notation $f(C)$ to denote the array obtained by applying f to each of the s points of C, consider the classifier term $u^* f(C)$, where u^* is obtained by solving (1), and generate an approximation (using $t < s$ points) to this term by considering the problem NLP(u^*,C,t), which is the unconstrained NLP:

$$\text{Min}_{w,Z} \quad d(u^* f(C),\, w\, f(Z)),$$

where $d()$ provides a measure of distance. As a particular case, in this research we used the 2-norm:

$$\text{Min}_{w,Z} \quad ||u^* f(C) - w\, f(Z)||_2^2 \qquad (\mathbf{2})$$

where w is a vector of size t, and Z is a set of t synthetic points, which may be distinct from points in C. Thus, the optimal solution $w^* f(Z^*)$ approximates the classifier term $u^* f(C)$. In order to avoid computation with f in feature space, this NLP is re-formulated by expanding the squared 2-norm and applying the Mercer inner product property to obtain an equivalent problem expressed in input space [2]. The expanded problem (2) is

$$\text{Min}_{w,Z} \; u^* f(C) f(C) \; u^* - 2\, u^* f(C) f(Z)\, w + w\, f(Z) f(Z)\, w,$$

so the corresponding problem in input space via the inner product property is

$$\text{Min}_{w,Z} \quad u^* K(C,C) \; u^* - 2\, u^* K(C,Z) \; w + wK(Z,Z)\, w.$$

Note that a numerical value of $K(y,x)$ for a specific pair (y,x) is computed by operations in input space (for example, we may evaluate $K(y,x)=(yx)^p$ where p is a positive integer and yx represents the inner product in input space. Evaluation of $K(y,x)$ via the expression $f(y)\,f(x)$ would be impractical for $p>1$ and large n).

The CCS strategy considered here allows different algorithmic formulations. We now describe the i-th stage of the multistage CCS algorithm that we used in this research. In our approach we perform *p independent runs of the i-th stage process*, using a different random subset of training points in each run; the results of these p runs are then combined as described below to provide the initial set of classifier points C_{i+1} for stage $i+1$. In our results we use the settings $p=10$, $t=10$ (number of points used in classifier approximation problems). See Fig. 1 for an overview of one of the independent runs of stage i.

At each successive stage, the size of the subset of the training set used to provide data for the classification problem is increased as described below until a classifier problem for the full training set is reached.

Stage *i* process (performed p times using p independent samples from the training set) :

a) *Let C_i be a set of s points in R^n, with $s=pt << m$ (C_0 is chosen as a subset of the training set; for $i>0$ the C_i are sets of synthetic points obtained from the preceding stage).*
b) *Let Y_i be a randomly chosen subset of Y such that $size(Y_i)>size(Y_{i-1})$; solve the* ***classifier problem*** *$LP(Y_i , C_i)$.*
c) *Letting Z_i, w_i be a set of initial values for a nonlinear optimization procedure, solve the* ***approximation problem*** *$NLP(u_i^*,C_i,t)$ to obtain* a set of synthetics Z^*_i .
d) *Solve the* ***classifier problem:*** *$LP(Y_i , Z^*_i)$.* **(3)**

Coordination Step (coordinates the p i-th stage processes): *For each run performed in stage $i+1$, C_{i+1} is the* ***union*** *of all Z^*_i for the p runs of the i-th stage (yielding a total of $s=pt$ potential classifier points for each of the initial classifier problems of step (b) in stage $i+1$).*

It should be noted that step (d) of this chunking strategy serves to validate the choices made for the strategic parameters p and t in the sense that the testing set correctness should be similar for steps (b) and (d) and correctness should stabilize as the subsets of Y approach the size of the original training set. This behavior was observed in the computational results that we now present.

Important remark: Alternative implementations of step (c) could employ other strategies to generate appropriate synthetic points. Some possible methods are described in [3,13,15].

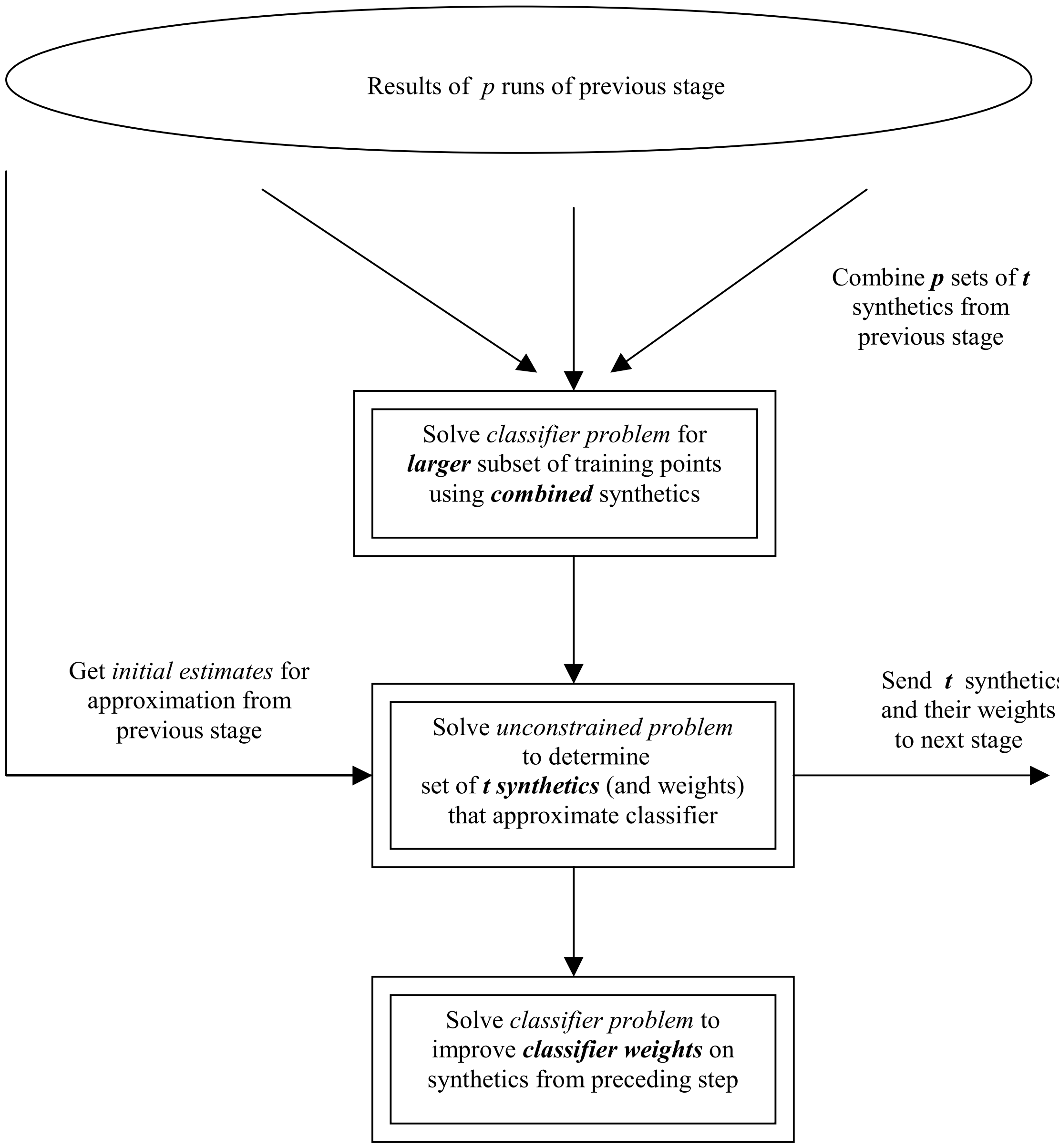
Results of *p* runs of previous stage
Combine ***p*** sets of ***t*** synthetics from previous stage
Solve *classifier problem* for ***larger*** subset of training points using ***combined*** synthetics
Get *initial estimates* for approximation from previous stage
Solve *unconstrained problem* to determine set of ***t synthetics*** (and weights) that approximate classifier
Send ***t*** synthetics and their weights to next stage
Solve *classifier problem* to improve ***classifier weights*** on synthetics from preceding step

Fig. 1. One of *p* runs of stage *i* process

3. Computational results, modified USPS problem

In order to evaluate the CCS algorithm, we considered a variant of the US Postal Service problem [14]. The USPS problem is composed of 9,298 patterns, 256 features each, corresponding to scanned handwritten numbers (0-9). Thus, it is a multi-category problem. Roughly, each category has the same number of representatives. To generate a (harder) modified version, inspired by [13], we add noise to a sample of USPS points in order to simulate a noisier dataset with less separability:

- We use a USPS sample as centers. In order to balance the problem, the sample is composed of 90 random '8' patterns and a random 10-pattern subset for each of the other nine digits.
- The resulting 180-center set was used as input for the NDC data generator [9] (thus, NDC centers are real data, instead of random points). The NDC generator typically assigns labels to centers by means of a random separating hyperplane, but in our case, this assignment method is not necessary, because labels are known *a priori*: 1 for '8' centers and -1 for the other centers and (as is usual in NDC) all points that are generated from a specific center are assigned the center's label. The overall result is a training set with approximately 4000 points of category 1 and 4000 of category –1. A Gaussian kernel $K(y,x)= exp(-|y-x|^2 /128)$ was used as in [13], where it provided good results.

All our runs took place on a node of a Sun Enterprise machine with an UltraSPARC II processor at 250 MHz. Step (b) and any other linear problems in our tests were solved using CPLEX 6.0. The nonlinear approximation problem in step (c) was solved by invoking CONOPT 2 via GAMS 2.50A (we chose an "off-the-shelf" NLP solver for this initial implementation, and found it to be quite effective. Other NLP techniques could, of course, be more efficient for this particular class of applications. With CONOPT 2 we found that the solution process was accelerated by imposing box constraints on the variables w. In particular, we impose the constraints: $-\alpha|u^*_i| \leq w \leq \alpha|u^*_i|$ for some $\alpha > 0$. The CONOPT 2 option `rtredg`, reduced gradient tolerance for optimality, was set to 1e-4). Both CPLEX and GAMS were called from a C program that implemented the whole algorithm.

All settings were NDC defaults, except for the number of points (8000 used here), number of features (256) and expansion factor (25) (this factor controls the expansion of point clouds around centers, using a random covariance matrix for each center). The expansion factor was selected to produce a separability in the range of 70% for the Gaussian kernel $K(y,x)=exp(-|y-x|^2 /128)$ in a standard 1-norm classification problem. The value ν=10.0 was found to yield the best training correctness for this dataset. Four stages were used, employing the following *sizes for the randomly selected subsets of the data:*

0. 1000 USPS/NDC points: 500 testing and 500 training (Y_{m0})
1. 2000 USPS/NDC points: 1000 testing and 1000 training (Y_{m1})
2. 4000 USPS/NDC points: 2000 testing and 2000 training (Y_{m2})
3. 8000 USPS/NDC points: 4000 testing and 4000 training (Y_{m3})

A point in Y_{mi} is termed a *support vector* if the dual variable of its classification constraint is nonzero at the solution, implying its classification constraint is active at the solution. Therefore, for similar testing correctness, the number of support vectors shown below is a measure of the robustness of the classifier in the sense that the support vectors include (in addition to "error" points) those points that are correctly classified, but lie in the margin and hence are not "widely separated" from the points of the other category.

The following implementation choices were made:

- α=1.25.
- The initial value set Z_0 for the initial approximation problem is a random subset of t Y_{m0} points with $u \neq 0$ after step (a) in stage 0. The w_0 are their u multipliers.

- For $i \geq 0$, the initial value set Z_{i+1} is the best Z^*_i obtained from an approximation problem in step (c), over p runs, in terms of testing set classification in step (c). The initial w_{i+1} are the corresponding w, also after step (c).

In order to evaluate the efficiency of the CCS algorithm, it is instructive to compare the quality of the solution at step (d) of stage i with the output of the **standard 1-norm classification problem** in which $C_i = Y_{mi}$:

$$\begin{aligned} &\text{Min}_{u,\gamma,E} \quad \nu\,||E||_1 + ||u||_1 \\ &\text{s.t. } D(uK(Y_{mi}, Y_{mi}) - \gamma\mathbf{1}) + E \geq \mathbf{1},\ E \geq \mathbf{0}, \end{aligned} \qquad \textbf{(4)}$$

Note that, in the particular case of stage 0, the problem (4) is the same as step (b). However, relative to stages 1, 2 and 3, the number of classifier points that are allowed and the corresponding number of variables grow significantly in the case of problem (4).

The results for **stage 0** were:

Step (b)	Step (d)
Avg. # of classifier points, p runs: 96.8	Max. # of classifier points for each run: 10
Avg. training correctness: 84.3%	Avg. training correctness: 84.38%
Avg. testing correctness: 72.12%	Avg. testing correctness: 72.42%
Avg. # of support vectors: 362.5 (out of 500)	Avg. # of support vectors: 212.5 (out of 500)

We see that the number of support vectors in step (d) is considerably less than in the standard 1-norm classification problem. Also, using only a small number of synthetic points does not degrade testing correctness, but instead yields a small improvement (similar results are obtained for the larger subsets considered below).

Step (c) yields an approximation of the classifier obtained in step (b). In order to evaluate the need of step (d), we evaluated the average quality of the step (c) classifier over all runs in stage 0 ("pure" synthetic-point classifier, without the re-classification step (d)) :

- Average testing correctness, step (c): 56.14%
- Average training correctness, step (c): 56.28%

Consequently, the benefit of the re-classification in step (d) is evident. Another question that may arise is the quality of trivial alternatives to the NLP synthetic point generation procedure of step (c). Thus, we considered a simple choice of classifier points: given the subset of C_0 such that $u \neq 0$ after step (b), we took a t-point subset whose u multipliers had largest absolute value, and used them for re-classification of the training points in stage 0, yielding:

- Average training correctness, modified step (d): 60.54%
- Average testing correctness, using approximate classifier from step (d): 58.16%

We observe again the considerable advantage of using synthetic points. Similar results were obtained when t-point random samples of Y_{m0} were used.

The behavior in stages 1 and 2 was similar:

Stage 1, step (b)	***Stage 1***, step (d)
Avg. # of classifier points, p runs: 30	Max. # of classifier points, p runs: 10
Avg. training correctness: 79.55%	Avg. training correctness: 79.34%
Avg. testing correctness: 77.17%	Avg. testing correctness: 77.13%
	Avg. # of support vectors: 515.7 (out of 1000)
Stage 2, step (b)	***Stage 2***, step (d)
Avg. # of classifier points, p runs: 13.1	Max. # of classifier points, p runs: 10
Avg. training correctness: 76.77%	Avg. training correctness: 76.71%
Avg. testing correctness: 76.76%	Avg. testing correctness: 76.62%
	Avg. # of support vectors: 1104 (out of 2000)

Standard 1-norm formulation in problem (4):

1000 training points, Y_{m1}	2000 training points, Y_{m2}
Avg. # of classifier points, p runs: 122.2	Avg. # of classifier points, p runs: 180.6
Avg. training correctness: 82.06%	Avg. training correctness: 81.94%
Avg. testing correctness: 73.3%	Avg. testing correctness: 75.47%
Avg. # of support vectors: 672.5 (out of 1000)	Avg. # of support vectors: 1181 (out of 2000)

Note that while *training correctness* is improved by allowing all 2000 points in the stage 2 training subset to be used to construct the classifier, the resulting *testing correctness* of 75.47% is actually worse than that obtained by using the classifier points from the much smaller synthetic sets C_i (whose 100 points yield 76.76% classification) or Z^*_i (whose 10 points yield 76.62%). This apparently surprising result that we already observed in stage 0 is analogous to results in [6], where random 1%-5% subsets of the Adult Dataset [8] were allowed in the Gaussian kernel classifier. Observe that if the solution quality is measured by the number of support vectors, then by this measure as well, the smaller classifier sets provide better quality solutions than the full training set. Similar results are obtained when the full dataset is used in stage 3, except that the standard 1-norm LP could not be solved in 48 hours, illustrating the scalability difficulties associated with the standard approach. Results are summarized in table 1 below.

We thus compared our results (as provided by the use of synthetic points in the classifier) with *random subset classifiers*:

1-norm classifier, 100 *random* points (10%) of Y_{m1}	1-norm classifier, 10 *random* points (1%) of Y_{m1}
Avg. # of classifier points, p runs: 51.9	Avg. # of classifier points, p runs: 10
Avg. training correctness: 74.15%	Avg. training correctness: 58.44%
Avg. testing correctness: 68.52%	Avg. testing correctness: 55.59%
	Avg. # of support vectors: 923.4 (out of 1000)
1-norm, 100 *random* points (5%) of Y_{m2}	1-norm, 10 *random* points (0.5%) of Y_{m2}
Avg. # of classifier points, p runs: 63.9	Avg. # of classifier points, p runs: 10
Avg. training correctness: 74.28%	Avg. training correctness: 59.18%
Avg. testing correctness: 70.83%	Avg. testing correctness: 57.74%
	Avg. # of support vectors: 1814.3 (out of 2000)

Observe that for this dataset, randomly chosen subsets of the training set produce average classifications that are significantly worse than those obtained with either the 1-norm classification with the full training set or our small sets of synthetic points. Selected results are summarized in table 1.

We close this section with a proposal of an *alternative CCS algorithm*. The most expensive step is the approximation problem of step (c), whose run-time depends on the size of the reduced set Z. A tiny reduced set may lead to a poor classifier. Hence, instead of increasing t (and thereby increasing the size of the approximation problem), we obtain another set of t synthetics by solving (c) starting from a different initial point (in our experiments, a random point - many other obvious and promising choices could also be explored). Then, we perform step (d) as

$$\begin{aligned} &\text{Min}_{w,\gamma,E} \quad \nu \|E\|_1 + \|w\|_1 \\ &\text{s.t. } D(wK(Z^*_i \, U \, Z^*_r,\, Y_i) - \gamma \mathbf{1}) + E \geq \mathbf{1}, \;\; E \geq \mathbf{0}, \end{aligned} \qquad (\mathbf{5})$$

where U is the set union operator and Z^*_r is the output of the second instance of step (c). This alternative CCS algorithm has potential for CCS parallel computation, since multiple approximation problems may be run in parallel.

In summary, for this dataset, the generalizability of *simple classifiers* obtained by solving optimization problems involving small numbers of synthetic points is better than that obtained via classifiers generated by much larger optimization problems that consider large sets of potential classifier points (see [6] for analogous observations based on the use of small randomly selected subsets of the training set as classifier points for a different collection of datasets).

4. Conclusions and directions for future research

In this paper, we have analyzed the fusion of synthetic point generators (small nonlinear programs) with different chunking algorithms to develop a chunking-coordinated-synthetic (CCS) approach that achieves good generalizability while greatly reducing the size of the optimization problems that are required to produce nonlinear classifiers. Our numerical results on a modified USPS dataset show that the classifiers obtained using very small numbers of synthetic points (*as few as 10*) not only yield good generalizability (in terms of good testing set classification in ten-fold cross-validation), but also, for the noisy data considered, actually yield classifiers with *better generalizability* than either various other choices of reduced sets of training points or even the full training set (the latter appears to result in over-training as was noted in [6] for other datasets and other reduced set approaches). Finally, since the CCS approach utilizes the solution of independent optimization problems at each stage, computation may be further accelerated by parallelization. One of our future directions for research will be the parallel implementation of CCS.

# training, # testing points	**# classifier points allowed**	**Source of classifier points**	**Training correctness %**	**# support vectors**	**Testing correctness %**
500 500	t=10	Z^*_0	84.38 (d)	213	72.42 (d)
1000 1000	t=10	Z^*_1	79.34 (d)	516	77.13 (d)
2000 2000	t=10	Z^*_2	76.71 (d)	1104	76.62 (d)
2000 2000	10 random	Y_{m2}	59.18	1814	57.74
2000 2000	s=100	C_2	76.77 (b)	1106	76.76 (b)
2000 2000	100 random	Y_{m2}	74.28	1409	70.83
2000 2000	Y_{m2}	Y_{m2}	81.94	1181	75.47
4000 4000	t=10	Z^*_3	75.58 (d)	2282	75.89 (d)
4000 4000	s=100	C_3	75.74 (b)	2280	76.02 (b)
4000 4000	100 random	Y_{m3}	73.78	2710	71.79

Table 1. Effect of classifier point set on generalizability (noisy USPS data; NLP:synthetic point generator)

The version of the CCS approach considered above reduces computation time by eliminating the computation of the very large matrix *K(Y,Y)* and by greatly reducing the number of terms (and corresponding optimization variables) used in the classifier. However, the use of the 1-norm to measure error leads to the introduction of large numbers of constraints when the corresponding LP is constructed. Hence, we will also consider alternative unconstrained classifier problems such as those discussed in [5,7,10] to determine if these replacements for 1-norm LP classifier problems can lead to improvements in scalability and computing times. As a variation of this latter approach, we will also investigate the utility of the unconstrained classifier problem obtained by inserting the "error" directly in the objective

$$\text{Min}_{u,\gamma} \quad \nu\, \mathbf{1}\, (\mathbf{1} - D(uK(C, Y_i) - \gamma \mathbf{1}))+ \; u\, u + \gamma^2. \qquad \mathbf{(6)}$$

The model (6) yields the usual penalty in the objective for points that are incorrectly classified, but assigns a "bonus" to points that are in the interior of the "correct" region beyond the margin. Allowing such bonuses may be particularly appropriate for noisy datasets and Gaussian kernels in which penalties and bonuses may be comparable. Note that (6) is an unconstrained quadratic problem whose solution may be written in closed form. In a more sophisticated variation of (6), the uniform weights on the error terms could be replaced by using a weight vector *Ω* after the initial classifier is generated:

$$\text{Min}_{u,\gamma} \quad \nu\, \Omega\, (\mathbf{1} - D(uK(C, Y_i) - \gamma \mathbf{1}))+ \; u\, u + \gamma^2. \qquad \mathbf{(7)}$$

These weights *Ω* could be generated in a pre-processing step by applying one or more of the preceding classifiers to the current training subset Y_i, and then setting the weight Ω_j on a point y_j to 1 if there is a positive error term associated with the point via one or more of those classifiers, and conversely setting the weight to 0 (or some small positive value) otherwise (see [12] for a related approach to weight adjustment). It is easy to verify via optimality conditions that if this process sets the weights *Ω* correctly, then the solution of (7) is the same as the solution of the constrained problem

$$\begin{aligned}&\text{Min}_{u,\gamma,E} \quad \nu\, ||E||_1 + \; u\, u + \gamma^2 \\ &\text{s.t. } D(uK(C, S) - \gamma \mathbf{1}) + E \geq \mathbf{1}, \;\; E \geq \mathbf{0}.\end{aligned}$$

Finally, although the emphasis of this research is on nonlinear kernels, similar ideas could be applied to linear kernels. For them, problem (2) is trivial, since *f* is the identity function and (2) is solved by taking *w*=1 and *Z*= *u** *C*. The interesting aspect of CCS in this case is the coordination step in which these single-point "ideal" classifiers for subsets are combined to produce a small classifier set for a larger subset. The key issue is whether these small classifier sets will produce good classifiers.

References

1. P.S. Bradley and O.L. Mangasarian. "Feature selection via concave minimization and support vector machines". In J. Shavlik, editor, Machine Learning Proceedings of the Fifteenth International Conference (ICML'98), pp. 82-90, San Francisco CA, 1998, Morgan Kaufmann.
2. C. J. C. Burges. "Simplified Support Vector Decision Rules". In L. Saitta, editor, Proceedings 13th Intl. Conf. on Machine Learning, pp. 71-77, San Mateo CA, 1996, Morgan Kaufmann.
3. C. J. C. Burges and B. Schölkopf. "Improving the Accuracy and Speed of Support Vector Machines". In M. Mozer, M. Jordan, and T. Petsche, editors, Advances in Neural Information Processing Systems 9, pages 375-381, Cambridge, MA, 1997. MIT Press.
4. T. Joachims. "Making large-scale SVM learning practical". In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, Advances in Kernel Methods - Support Vector Learning, pages 169-184, Cambridge, MA, 1999. MIT Press.

5. Y.-J. Lee and O. L. Mangasarian . SSVM: A Smooth Support Vector Machine for Classification. Data Mining Institute Technical Report 99-03, September 1999, Computational Optimization and Applications, to appear.
6. Y.-J. Lee and O. L. Mangasarian, "RSVM: Reduced Support Vector Machines". Data Mining Institute technical report 00-07, July 2000.
7. O. L. Mangasarian and David R. Musicant. "Lagrangian Support Vector Machines". Data Mining Institute Technical Report 00-06, June 2000.
8. P. M. Murphy and A. W. Aha. UCI repository of machine learning databases, 1992. www.ics.uci.edu/~mlearn/MLRepository.html.
9. D. R. Musicant. NDC: Normally Distributed Clustered datasets, 1998. www.cs.wisc.edu/~musicant/data/ndc.
10. D. R. Musicant. "Data Mining via Mathematical Programming and Machine Learning", Ph.D. Thesis, Computer Sciences Department, University of Wisconsin – Madison, 2000.
11. E. Osuna and F. Girosi. "Reducing the Run-time complexity of Support Vector Machines". In Proceedings of the 14th International Conference on Pattern Recognition, Brisbane, Australia, 1998.
12. Robert E. Schapire. "The Strength of Weak Learnability". Machine Learning, 5(2): 197-227,1990.
13. B. Schölkopf, S. Mika, C. J. C. Burges, P. Knirsch, K.-R. Müller, G. Rätsch and A. J. Smola. "Input Space vs. Feature Space in Kernel-Based Methods". IEEE Transactions on Neural Networks 10(5):1000-1017, 1999.
14. B. Schölkopf and A. J. Smola. Kernel machines page, 2000. www.kernel-machines.org.
15. D. DeCoste and B. Schölkopf. "Training Invariant Support Vector Machines". Machine Learning, in press.