Multiple Incremental Decremental Learning of Support Vector Machines

Authored by:

Ichiro Takeuchi
Masayuki Karasuyama

Abstract

We propose a multiple incremental decremental algorithm of Support Vector Machine (SVM). Conventional single cremental decremental SVM can update the trained model efficiently when single data point is added to or removed from the training set. When we add and/or remove multiple data points, this algorithm is time-consuming because we need to repeatedly apply it to each data point. The roposed algorithm is computationally more efficient when multiple data points are added and/or removed simultaneously. The single incremental decremental algorithm is built on an optimization technique called parametric programming. We extend the idea and introduce multi-parametric programming for developing the proposed algorithm. Experimental results on synthetic and real data sets indicate that the proposed algorithm can significantly reduce the computational cost of multiple incremental decremental operation. Our approach is especially useful for online SVM learning in which we need to remove old data points and add new data points in a short amount of time.

1 Paper Body

Incremental decremental algorithm for online learning of Support Vector Machine (SVM) was previously proposed in [1], and the approach was adapted to other variants of kernel machines [2?4]. When a single data point is added and/or removed, these algorithms can efficiently update the trained model without re-training it from scratch. These algorithms are built on an optimization technique called parametric programming [5?7], in which one solves a series of optimization problems parametrized by a single parameter. In particular, one solves a solution path with respect to the coefficient parameter corresponding to the data point to be added or removed. When we add and/or remove multiple data points using these algorithms, one must repeat the updating operation for each single data point. It often requires too much computational cost to use it for real-time online learning. In what follows, we refer this conventional algorithm

as single incremental decremental algorithm or single update algorithm. In this paper, we develop a multiple incremental decremental algorithm of the SVM. The proposed algorithm can update the trained model more efficiently when multiple data points are added and/or removed simultaneously. We develop the algorithm by introducing multi-parametric programming [8] in the optimization literature. We consider a path-following problem in the multi-dimensional space spanned by the coefficient parameters corresponding to the set of data points to be added or removed. Later, we call our proposed algorithm as multiple incremental decremental algorithm or multiple update algorithm. The main computational cost of parametric programming is in solving a linear system at each breakpoint (see Section 3 for detail). Thus, the total computational cost of parametric programming is roughly proportional to the number of breakpoints on the solution path. In the repeated use of 1

single update algorithm for each data point, one follows the coordinate-wise solution path in the multi-dimensional coefficient parameter space. On the other hand, in multiple update algorithm, we establish a direction in the multi-dimensional coefficient parameter space so that the total length of the path becomes much shorter than the coordinate-wise one. Because the number of breakpoints in the shorter path followed by our algorithm is less than that in the longer coordinate-wise path, we can gain relative computational efficiency. Figure 2 in Section 3.4 schematically illustrates our main idea. This paper is organized as follows. Section 2 formulates the SVM and the KKT conditions. In Section 3, after briefly reviewing single update algorithm, we describe our multiple update algorithm. In section 4, we compare the computational cost of our multiple update algorithm with the single update algorithm and with the LIBSVM (the-state-of-the-art batch SVM solver based on SMO algorithm) in numerical experiments on synthetic and real data sets. We close in Section 5 with concluding remarks.

2

Support Vector Machine and KKT Conditions

Suppose we have a set of training data {(xi , yi)}ni=1 , where xi ? X ? Rd is the input and yi ? {?1, +1} is the output class label. Support Vector Machines (SVM) learn the following discriminant function: f (x) = wT ?(x) + b, where ?(x) denotes a fixed feature-space transformation. The model parameter w and b can be obtained by solving an optimization problem: n ? 1 min ——w——2 + C ?i 2 i=1 s.t. yi f (xi) ? 1 ? ?i , ?i ? 0, i = 1, ? ? ? , n, where C ? R is the regularization parameter. Introducing Lagrange ? multipliers ?i ? 0, the n optimal discriminant function f : X ? R can be formulated as f (x) = i=1 ?i yi K(x, xi) + b, where K(xi , xj) = ?(xi)T ?(xj) is a kernel function. From the Karush-Kuhn-Tucker (KKT) optimality conditions, we obtain the following relationships: yi f (xi) ¿ 1 ? ?i = 0, (1a) yi f (xi) = 1 ? ?i ? [0, C], (1b) yi f (xi) ¡ 1 ? ?i = C, (1c) n ? yi ?i = 0. (1d) +

i=1

Using (1a)-(1c), let us define the following index sets: O = {i — yi f (xi) ¿ 1, ?i = 0}, (2a) M = {i — yi f (xi) = 1, 0 ? ?i ? C}, (2b) I = {i — yi f (xi) ¡ 1, ?i = C}. (2c) In what follows, the subscription by an index set, such as v I

for a vector v ? Rn , indicates a subvector of v whose elements are indexed by I. Similarly, the subscription by two index sets, such as M M,O for a matrix M ? Rn?n , denotes a submatrix whose rows are indexed by M and columns are indexed by O. If the submatrix is the principal submatrix such as QM,M , we abbreviate as QM .

3 3.1

Incremental Decremental Learning for SVM Single Incremental Decremental SVM

In this section, we briefly review the conventional single incremental decremental SVM [1]. Using the SV sets (2b) and (2c), we can expand yi f (xi) as ? ? yi f (xi) = Qij ?j + Qij ?j + yi b, j?M

j?I

2

where Qij = yi yj K(xi , xj). When a new data point (xc , yc) is added, we increase the corresponding new parameter ?c from 0 while keeping the optimal conditions of the other parameters satisfied. Let us denote the amount of the change of each variable with an operator ?. To satisfy the equality conditions (1b) and (1d), we need ? Qic ??c + Qij ??j + yi ?b = 0, i ? M, j?M

yc ??c +

?

yj ??j

=

0.

j?M

Solving this linear system with respect to ??i , i ? M, and b, we obtain the update direction of the parameters. We maximize the ??c under the constraint that no element moves across M, I and O. After updating the index sets M, I and O, we repeat the process until the new data point satisfies the optimality condition. Decremental algorithm can be derived similarly, in which the target parameter moves toward 0. 3.2

Multiple Incremental Decremental SVM

Suppose we add m new data points and remove ? data points simultaneously. Let us denote the index set of new adding data points and removing data points as A = {n + 1, n + 2, ? ? ? , n + m} and R ? {1, ? ? ? , n}, respectively, where —R— = ?. We remove the elements of R from the sets M, I and O (i.e. M ? M R, I ? I R and O ? O R). Let us define y = [y1 , ? ? ? , yn+m]? , ? = [?1 , ? ? ? , ?n+m]? , and Q ? R(n+m)?(n+m) , where (i, j)-th entry of Q is Qij . When m = 1, ? = 0 or m = 0, ? = 1, our method corresponds to the conventional single incremental decremental algorithm. We initially set ?i = 0, ?i ? A. If we have yi f (xi) ¿ 1, i ? A, we can append these indices to O and remove them from A because these points already satisfy the optimality condition (1a). Similarly, we can append the indices {i — yi f (xi) = 1, i ? A} to M and remove them from A. In addition, we can remove the points {i — ?i = 0, i ? R} because they already have no influence on the model. Unlike single incremental decremental algorithm, we need to determine the directions of ??A

and ??R . These directions have a critical influence on the computational cost. For ??R , we simply trace the shortest path to 0, i.e., ??R = ???R ,

(3)

where ? is a step length. For ??A , we do not know the optimal value of ?A beforehand. To determine this direction, we may be able to use some optimization techniques (e.g. Newton method). However, such methods usually need additional computational burden. In this paper, we simply take ??A = ?(C1 ? ?A).

(4)

This would become the shortest path if ?i = C, ?i ? A, at optimality. When we move parameters ?i , ?i ? A ? R, the optimality conditions of the other parameters must be kept satisfied. From yi f (xi) = 1, i ? M, and the equality constraint (1d), we need ? ? ? Qij ??j + Qij ??j + Qij ??j + yi ?b = 0, i ? M, (5) j?A

j?R

?

j?A

?

yj ??j +

j?M

yj ??j +

j?R

[M=

yj ??j

=

0.

(6)

j?M

Using matrix notation, (5) and (6) can be written as [ ] [?b y? A M + ??M QM,A where

?

y? R QM,R

y? M QM

0 yM 3

] .

][

??A ??R

] = 0,

(7)

From the definitions of the index sets in (2a)-(2c), the following inequality constraints must also be satisfied: 0 ? ?i + ??i ? C, yi {f (xi) + ?f (xi)} ¿ 1, yi {f (xi) + ?f (xi)} ¡ 1,

i ? M, i ? O, i ? I.

(8a) (8b) (8c)

Since we removed the indices {i — f (xi) ? 1} from A, we obtain yi {f (xi) + ?f (xi)} ¡ 1,

i ? A.

(9)

During the process of moving ?i , i ? A, to C from 0, if the inequality (9) becomes equality for any i, we can append the point to M and remove it from A. On the other hand, if (9) holds until ?i becomes C, the point moves to I. In the path following literature [8], the region that satisfies (8) and (9) is called critical region (CR). We decide update direction by the linear system (7) while monitoring inequalities (8) and (9). Substituting (3) and (4) to (7), we obtain the update direction [ ] [ ][ ] ?b C1 ? ?A y? y? A R = ??, where ? = ?M ?1 . (10) ??M ??R QM,A QM,R To determine step length ?, we need to check inequalities (8) and (9). Using vector notation and the hadamard product ? (element-wise product [9]), we can write y ? ?f = ? ?, where ? = [y Q:,M] ? + Q:,A (C1 ? ?A) ? Q:,R ?R ,

(11)

and the subscription ?:? of Q denotes the index of all the elements {1, ? ? ? , n + m}. Since (10) and (11) are linear function of ?, we can calculate the set of the largest step length ?s for each i at which the inequalities (8) and (9) becomes equality for i. The size of such ?s is —M— ? 2 + —O— + —I— + —A— and we define this set as H. We determine the step length as follows: ? = min({? ? — ?? ? H, ?? ? 0} ? {1}). If ? becomes 1, we can terminate the algorithm because all the new data points in A and existing points in M, O and I satisfy the optimality conditions and ?R is 0. Once we decide ?, we can update ?M and b using (10), and ?A and ?R using (3) and (4). In the path-following literature, the points at which the size of linear system (7) is changed are called breakpoints. If the ith data point reaches bound of any one of the constraints (8) and (9) we need to update M, O and I. After updating, we re-calculate ?, ? to determine the next step length. 3.3

Empty Margin

We need to establish the way of dealing with the empty margin M. In such case, we can not obtain the bias from yi f (xi) = 1, i ? M. Then we can only obtain the interval of the bias from i ? O, i ? I ? A.

yi f (xi) ¿ 1, yi f (xi) ¡ 1,

To keep these inequality constraints, the bias term must be in max yi gi ? b ? min yi gi , i?L

where gi = 1 ?

?

(12)

i?U

?i Qij ?

i?I

? i?A

?i Qij ?

?

?i Qij ,

i?R

and L = {i — i ? O, yi = +1} ? {i — i ? I ? A, yi = ?1}, U = {i — i ? O, yi = ?1} ? {i — i ? I ? A, yi = +1}. If this empty margin happens during the path-following, we look for the new data points which re-enter the margin. When the set M is empty, equality constraint (6) becomes ? ? yi ??i + yi ??i = ??(?) = 0, (13) i?A

i?R

4

Figure 1: An illustration of the bias in empty margin case. Dotted lines represent yi (gi + ?gi (?)), for each i. Solid lines are the upper bound and the lower bound of the bias. The bias term is uniquely determined when u(?) and l(?) intersect. where ?(?) =

?

yi (C ? ?i) ?

i?A

?

yi ?i .

i?R

We take two different strategies depending on ?(?). First, if ?(?) ?= 0, we can not simply increase ? from 0 while keeping (13) satisfied. Then we need new margin data point m1 which enables equality constraint to be satisfied. The index m1 is either ilow = argmax yi gi or iup = argmax yi gi . i?L

i?U

If ilow , iup ? O ? I, we can update b and M as follows: ?(?) ¿ 0 ?(?) ¡ 0

? b = yiup giup , M = {iup }, ? b = yilow gilow , M = {ilow }.

By setting the bias terms as above, equality condition ??(?) + ym1 ??m1 = 0 is satisfied. If ilow ? A or iup ? A, we can put either of these points to margin. On the other hand, if ?(?) = 0, we can increase ? while keeping (13) satisfied. Then, we consider increasing ? until the upper bound and the lower bound of the bias (12) take the same value (the bias term can be uniquely determined). If we increase ?, gi changes linearly: ? ? ? { ? } ?gi (?) = ? ??j Qij ? ??j Qij = ? ? (C ? ?j)Qij + ?j Qij . j?A

j?R

j?A

j?R

Since each yi (gi + ?gi (?)) may intersect, we need to consider the following piece-wise linear boundaries: u(?)

=

max yi (gi + ?gi (?)),

l(?)

=

min yj (gj + ?gj (?)).

i?U

j?L

Figure 1 shows an illustration of these functions. We can trace the upper bound and the lower bound until two bounds become the same value. 3.4

The number of breakpoints

The main computational cost of incremental decremental algorithm is in solving the linear system (10) at each breakpoint (The cost is O(—M—2) because we use Cholesky factor update except the first step). Thus, the number of breakpoints is an important factor of the computational cost. To simplify the discussion, let us introduce the following assumptions: ? The number of breakpoints is proportional to the total length of the path. ? The path obtained by our algorithm is the shortest one. 5

initial

final

path

breakpoints borders of CR

(a) Adding 2 data points.

(b) Adding and Removing 1 data point

Figure 2: The schematic illustration of the difference of path length and the number of breakpoints. Each polygonal region enclosed by dashed lines represents the region in which M, I, O and A are constant (CR: critical region). The intersection of the path and the borders are the breakpoints. The update of matrices and vectors at the breakpoints are the main computational cost of pathfollowing. In the case of Figure 2(a), we add 2 data points. If optimal ?1 = ?2 = C, our proposed algorithm can trace shortest path to optimal point from the origin (left plot). On the other hand, single incremental algorithm moves one coordinate at a time (right plot). Figure 2(b) shows the case that we add and remove 1 data point, respectively. In this case, if ?2 = C, our algorithm can trace shortest path to ?1 = 0, ?2 = C (left plot), while single incremental algorithm again moves one coordinate at a time (right plot). The first assumption means that the breakpoints are uniformly distributed on the path. The second assumption holds for the removing parameters ?R because we know that we should move ?R to 0. On the other hand, for some of ?A , the second assumption does not necessarily hold because we do not know the optimal ?A beforehand. In particular, if the point i ? A which was located inside the margin before the update moved to M during the update (i.e. the equality (9) holds), the path with respect to this parameter is not really the shortest one. To simplify the discussion further, let us consider only the case of —A— = m ¿ 0 and —R— = 0 (the same discussion holds for other cases too). In this simplified scenario, the ratio of the number of breakpoints of multiple update algorithm to that of repeated use of single update algorithm is ??A ?2 : ??A ?1 , where ? ? ?2 is ?2 norm and ? ? ?1 is ?1 norm. Figure 2 illustrates the ? concept in the case of m = 2. If we consider only the case of ?i = C, ?i ? A, the ratio is simply m : m.

4

Experiments

We compared the computational cost of the proposed multiple incremental decremental algorithm (MID-SVM) with (repeated use of) single incremental decremental algorithm [1] (SID-SVM) and with the LIBSVM [10], the-state-of-the-art batch SVM solver based on sequential minimal optimization algorithm (SMO). In LIBSVM, we examined several tolerances for termination criterion:

? = 10?3 , 10?6 , 10?9 . When we use LIBSVM for online-learning, alpha seeding [11, 12] sometimes works well. The basic idea of alpha seeding is to use the parameters before the update as the initial parameter. In alpha seeding, we need to take care of the fact that the summation constraint ?? y = 0 may not be satisfied after removing ?s in R. In that case, we simply re-distribute ? ?= ?i yi i?R

to the in-bound ?i , i ? {i — 0 ¡ ?i ¡ C}, uniformly. If ? cannot be distributed to in-bound ?s, it is also distributed to other ?s. If we still can not distribute ? by this way, we did not use alpha-seeding. For kernel function, we used RBF kernel K(xi , xj) = exp(??——xi ? xj ——2). In this paper, we assume that the kernel matrix K is positive definite. If the kernel matrix happens to be singular, which typically arise when there are two or more identical data points in M, our algorithm may not work. As far as we know, this degeneracy problem is not fully solved in path-following literature. Many heuristics are proposed to circumvent the problem. In the experiments described below, we 6

4 3
x2
2 1 0 ?1 ?2 ?2
?1
0
1
x
2
3
1

Figure 3: Artificial data set. For graphical simplicity, we plot only a part of data points. The cross points are generated from a mixture of two Gaussian while the circle points come from a single Gaussian. Two classes have equal prior probabilities.

use one of them: adding small positive constant to the diagonal elements of kernel matrix. We set this constant as 10?6 . In the LIBSVM we can specify cache size of kernel matrix. We set this cache size enough large to store the entire matrix. 4.1

Artificial Data

First, we used simple artificial data set to see the computational cost for various number of adding and/or removing points. We generated data points (x, y) ? R2 ? {+1, ?1} using normal distributions. Figure 3 shows the generated data points. The size of initial data points is n = 500. As discussed, adding or removing the data points with ?i = 0 at optimal can be performed with almost no cost. Thus, to make clear comparison, we restrict the adding and/or removing points as those with ?i = C at optimal. Figure 4 shows the log plot of the CPU time. We examined several scenarios: (a) adding m ? {1, ? ? ? , 50} data points, (b) removing ? ? {1, ? ? ? , 50} data points, (c) adding m ? {1, ? ? ? , 25} data points and removing ? ? {1, ? ? ? , 25} data points simultaneously. The horizontal axis is the number of adding and/or removing data points. We see that MID-SVM is significantly faster than SID-SVM. When m = 1 or ? =

1, SID-SVM and MID-SVM are identical. The relative difference of SID-SVM and MID-SVM grows as the m and/or ? increase because MID-SVM can add or remove multiple data points simultaneously while SID-SVM merely iterates the algorithm m + ? times. In this experimental setting, the CPU time of SMO does not change largely because m and ? are relatively smaller than n. Figure 5 shows the number of breakpoints of SID-SVM and MID-SVM along with the theoretical number of breakpoints of the MID-SVM in ? Section 3.4 (e.g., for scenario (a), the number of breakpoints of SID-SVM multiplied by m/m). The results are very close to the theoretical one. 4.2

Application to Online Time Series Learning

We applied the proposed algorithm to a online time series learning problem, in which we update the model when some new observations arrive (adding the new ones and removing the obsolete ones). We used Fisher river data set in StatLib [13]. In this data set, the task is to predict whether the mean daily flow of the river increases or decreases using the previous 7 days temperature, precipitation and flow (xi ? R21). This data set contains the observations from Jan 1 1988 to Dec 31 1991. The size of the initial data points is n = 1423 and we set m = ? = 30 (about a month). Each dimension of x is normalized to [0, 1]. We add new m data points and remove the oldest ? data points. We investigate various settings of the regularization parameter C ? {10?1 , 100 , ? ? ? , 105 } and kernel parameter ? ? {10?3 , 10?2 , 10?1 , 100 }. Unlike previous experiments, we did not choose the adding or removing data points by its parameter. Figure 6 shows the elapsed CPU times and Figure 7 shows 10-fold cross-validation error of each setting. Each figure has 4 plots corresponding to different settings of kernel parameter ?. The horizontal axis denotes the regularization parameter C. Figure 6 shows that our algorithm is faster than the others, especially in large C. It is well known that the computational cost of SMO algorithm becomes large when C gets large [14]. Crossvalidation error in Figure 7 indicates that the relative computational cost of our proposed algorithm is especially low for the hyperparameters with good generalization performances in this application problem. 7

?1.2
?1.6
10
MID?SVM SID?SVM SMO(?=1e?3) SMO(?=1e?6) SMO(?=1e?9)
?1.4
10
MID?SVM SID?SVM SMO(?=1e?3) SMO(?=1e?6) SMO(?=1e?9)
?1.3
10 CPU time (sec)
?1.4
10
CPU time (sec)
CPU time (sec)
10
MID?SVM SID?SVM SMO(?=1e?3) SMO(?=1e?6) SMO(?=1e?9)

?1.2
10
?1.6
10
?1.5
10
?1.7
10
?1.8
?1.8
10
10
?1.9
10 0
10
20
m
30
40
50
0
10
20
30
l
40
50
0
5
10 15 m and l
20
25
(a) Adding m data points. (b) Removing ? data points. (c) Adding m data points and removing ? data points simultaneously (m = ?).

Figure 4: Log plot of the CPU time (artificial data set) 600
MID?SVM SID?SVM Theoretical
500 the number of breakpoints
the number of breakpoints
500
400
300
200
500
MID?SVM SID?SVM Theoretical
450 400 the number of breakpoints
600

400
300
200
100
MID?SVM SID?SVM Theoretical
350 300 250 200 150 100
100
50 0 0
10
20
30
40
0 0
50
10
20
30
m
40
0 0
50
5
10
15
l
20
25
m and l
(a) Adding m data points. (b) Removing ? data points. (c) Adding m data points and removing ? data points simultaneously (m = ?).
Figure 5: The number of breakpoints (artificial data set) 3
10
1
10
0
10
10 MID?SVM SID?SVM SMO(?=1e?3) SMO(?=1e?6) SMO(?=1e?9)
CPU time (sec)
CPU time (sec)
10
2
CPU time (sec)
2
2
10 MID?SVM SID?SVM SMO(?=1e?3) SMO(?=1e?6) SMO(?=1e?9)
1

10
0
1
10
0.1
MID?SVM SID?SVM SMO(?=1e?3) SMO(?=1e?6) SMO(?=1e?9)
10 CPU time (sec)
3
10
0
10
?0.1
10
MID?SVM SID?SVM SMO(?=1e?3) SMO(?=1e?6) SMO(?=1e?9)
?0.3
10
10
?0.5
?1
10 ?2 10
?1
0
10
2
10 C
(a) ? = 10
4
10
10 ?2 10
6
10
0
10
?1
0
10
2
10 C
(b) ? = 10
4
10
10 ?2 10
6
10
?1
0

10
2
10 C
(c) ? = 10
4
10
0
6
5
10
10
?2
10
C
(d) ? = 10
?3
Figure 6: Log plot of the CPU time (Fisher river data set) 0.42
0.46
0.46
0.46
0.44
0.44
0.41 0.44
0.37 0.36 0.35
0.4 0.38 0.36
Cross Validation Error
0.38
0.42
Cross Validation Error
Cross Validation Error
Cross Validation Error
0.4 0.39
0.42
0.4
0.38
0.42
0.4
0.38
0.34 0.36
0.34
0.36
0.33 0.32 ?2 10
0
10
2
10 C

4
10
(a) ? = 100
6
10
0.32 ?2 10
0
10
2
10 C
4
10
6
10
(b) ? = 10?1
0.34 ?2 10
0
10
2
10 C
4
10
6
10
(c) ? = 10?2
0.34 ?2 10
0
10
2
10 C
4
10
(d) ? = 10?3
Figure 7: Cross-validation error (Fisher river data set)
5
Conclusion

We proposed multiple incremental decremental algorithm of the SVM. Unlike single incremental decremental algorithm, our algorithm can efficiently work with simultaneous addition and/or removal of multiple data points. Our algorithm is built on multi-parametric programming in the optimization literature [8]. We previously proposed an approach to accelerate Support Vector Regression (SVR) cross-validation using similar technique [15]. These multi-parametric programming frameworks can be easily extended to other kernel machines. 8
6
10

2 References

[1] G. Cauwenberghs and T. Poggio, ?Incremental and decremental support vector machine learning,? in Advances in Neural Information Processing Systems (T. K. Leen, T. G. Dietterich, and V. Tresp, eds.), vol. 13, (Cambridge, Massachussetts), pp. 409?415, The MIT Press, 2001. [2] M. Martin, ?On-line support vector machines for function approximation,? tech. rep., Software Department, University Politecnica de Catalunya, 2002. [3] J. Ma and J. Theiler, ?Accurate online support vector regression,? Neural Computation, vol. 15, no. 11, pp. 2683?2703, 2003. [4] P. Laskov, C. Gehl, S. Kruger, and K.-R. Muller, ?Incremental support vector learning: Analysis, implementation and applications,? Journal of Machine Learning Research, vol. 7, pp. 1909?1936, 2006. [5] T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu, ?The entire regularization path for the support vector machine,? Journal of Machine Learning Research, vol. 5, pp. 1391?1415, 2004. [6] L. Gunter and J. Zhu, ?Efficient computation and model selection for the support vector regression,? Neural Computation, vol. 19, no. 6, pp. 1633?1655, 2007. [7] G. Wang, D.-Y. Yeung, and F. H. Lochovsky, ?A new solution path algorithm in support vector regression,? IEEE Transactions on Neural Networks, vol. 19, no. 10, pp. 1753?1767, 2008. [8] E. N. Pistikopoulos, M. C. Georgiadis, and V. Dua, Process Systems Engineering: Volume 1: MultiParametric Programming. WILEY-VCH, 2007. [9] J. R. Schott, Matrix Analysis For Statistics. Wiley-Interscience, 2005. [10] 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. [11] D. DeCoste and K. Wagstaff, ?Alpha seeding for support vector machines,? in Proceedings of the International Conference on Knowledge Discovery and Data Mining, pp. 345?359, 2000. [12] M. M. Lee, S. S. Keerthi, C. J. Ong, and D. DeCoste, ?An efficient method for computing leave-one-out error in support vector machines,? IEEE transaction on neural networks, vol. 15, no. 3, pp. 750?757, 2004. [13] M. Meyer, ?Statlib.? http://lib.stat.cmu.edu/index.php. [14] L. Bottou and C.-J. Lin, ?Support vector machine solvers,? in Large Scale Kernel Machines (L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, eds.), pp. 301?320, Cambridge, MA.: MIT Press, 2007. [15] M. Karasuyama, I. Takeuchi, and R.Nakano, ?Efficient leave-m-out cross-validation of support vector regression by generalizing decremental algorithm,? New Generation Computing, vol. 27, no. 4, Special Issue on Data-Mining and Statistical Science, pp. 307?318, 2009.

9