Memory-efficient Large-scale Linear Support Vector Machine

Abdullah Alrajeh[ac], Akiko Takeda[b] and Mahesan Niranjan[c]

[a]CRI, King Abdulaziz City for Science and Technology, Saudi Arabia, asrajeh@kacst.edu.sa
[b]Mathematical Informatics, University of Tokyo, Japan, takeda@mist.i.u-tokyo.ac.jp
[c]ECS, University of Southampton, United Kingdom, {asar1a10, mn}@ecs.soton.ac.uk

ABSTRACT

Stochastic gradient descent has been advanced as a computationally efficient method for large-scale problems. In classification problems, many proposed linear support vector machines are very effective. However, they assume that the data is already in memory which might be not always the case. Recent work suggests a classical method that divides such a problem into smaller blocks then solves the sub-problems iteratively. We show that a simple modification of shrinking the dataset early will produce significant saving in computation and memory. We further find that on problems larger than previously considered, our approach is able to reach solutions on top-end desktop machines while competing methods cannot.

Keywords: Large-scale classification, linear SVM, stochastic gradient method, LIBLINEAR

1. INTRODUCTION

Support Vector Machines (SVMs), invented by Boser, Guyon and Vapnik [1], optimize classification boundaries to maximize the margin between classes. When used with kernels, the margin is imposed in a higher dimensional space implied by the chosen kernel function. Maximizing the margin restricts the complexity of the mapping between input and output spaces and acts as a regularization scheme. In this setting, the solution to the SVM problem reduces to a quadratic programming problem and is free from local minima, a particular attraction when compared to other machine learning methods such as neural networks.

Given a training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$ where $\mathbf{x}_i \in R^n$ and $y_i \in \{-1, +1\}$, the maximum margin hyperplane $(\mathbf{w}, b)$ requires solving the following optimization problem:

$$\underset{\mathbf{w}}{\text{minimize}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i \quad \text{subject to} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0\,, \forall i\,. \tag{1}$$

Here the slack variable ξ_i measures the margin violation by each data point $\mathbf{x}_i$ and $C \geq 0$ is a penalty parameter for this violation. The original SVM does not have slack variables which were introduced by [2]. The 1-norm soft margin version of SVM is very useful when the data is not lineally separable which is the case in many real-world problems. The optimization problem setting is called 1-norm soft margin because $\sum_i \xi_i$ can be replaced by $\|\boldsymbol{\xi}\|_1$ in (1). Introducing Lagrange multipliers $\boldsymbol{\alpha}$ to the primal form (1), the hyperplane can be found in the dual representation as follows:

$$\underset{\boldsymbol{\alpha}}{\text{maximize}} \quad \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^T\mathbf{x}_j \quad \text{subject to} \quad \sum_{i=1}^{l} y_i\alpha_i = 0 \text{ and } C \geq \alpha_i \geq 0\,, \forall i\,. \tag{2}$$

The solution $\mathbf{w}$ of the primal form (1) is realized by using the solution $\boldsymbol{\alpha}$ of the dual form (2) as follows: $\mathbf{w} = \sum_{i=1}^{l} y_i\alpha_i\mathbf{x}_i$. Note that in the dual form the data appears as an inner product between $\mathbf{x}_i$ and $\mathbf{x}_j$ without an explicit representation in the feature space. Hence, the problem complexity is independent from the feature dimension n. The remarkable advantage is to work on high dimensional feature space where the data might be lineally separable without having to compute the mapping explicitly. This is called the kernel trick where a function maps that space to the inner product as follows: $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$.

There are many nonlinear functions such as Gaussian kernel and polynomial kernel. Although nonlinear SVM could learn complex patterns but it is computationally expensive particularly for large-scale data since the optimization problem is a quadratic.

One of the early works for fast optimization is the kernel-Adatron algorithm based on a gradient ascent technique [3]. Decomposition methods are also good techniques to accelerate training and the idea is to update a subset of $\boldsymbol{\alpha}$ while keeping the remainder constant. Then a new α_i replaces an old one in the subset. These dual variables α_i that might change at each iteration are referred to as the *active set*. Sequential Minimal Optimization (SMO) is an extreme algorithm which only works on two points at each iteration [4].

In the case of linear kernel as in (2), the problem is still quadratic but one can take the advantage of accessing the feature space in order to accelerate the optimization. One of the techniques is a cutting plane solver SVM$^{\text{pref}}$ [5]. It is several times faster than decomposition methods such as SVM$^{\text{light}}$ proposed earlier by the same author. Pegasos is also a fast solver that alternates between stochastic gradient steps and projection steps to estimate the primal problem [6]. Another technique is a dual coordinate descent method with a shrinking heuristic [7].

All these techniques can successfully handle large-scale data sets when they are stored in the memory. This means that when the resources are limited, the algorithms will take a very long time due to severe disk-swapping. Recent work by Yu et al. [8] addressed this issue and proposed a block minimization method. The main idea is to divide the data set into m blocks where each block can fit in memory. Then exactly or approximately solve the sub-problem by any of the previous algorithms and update the variables. After going through all the blocks, the optimization procedure is repeated several times until stopping criterion is satisfied.

Our proposed method is closely related to [7,8]. We use the dual coordinate ascent method but with a harsh shrinking heuristic to reduce the data points as soon as possible from the first pass. Then the remaining active set is stored into a binary file and block optimization is carried out. At each iteration, the active set is reduced until convergence. The key difference between our method and the one referred to above [8] is the block optimization that is done after applying harsh shrinking in one pass over the data set. In many cases, the remaining active set is a small fraction of the original set (e.g. 10%) so the block optimization might not be necessary. Even if this is not the case, dividing the remaining active set is much cheaper than the whole data as we show in the discussion section. In addition to that, the number of blocks is dynamic and depends on the active set size at each iteration while they are fixed in Yu et al.'s study [8]. We have implemented a binary and a multiclass solver and present some promising results.

The remainder of this paper is organized as follows. Section 2 describes our proposed method for both binary and multiclass problems. Section 3 presents the experiments and compares them with three state-of-the-art linear SVM solvers. Section 4 discusses our findings then we end the paper with a summary of our conclusions.

2. STOCHASTIC GRADIENT METHOD

The gradient method is a simple optimization algorithm to find a local minimum or maximum of a function using its gradient. We start with initial values for $\boldsymbol{\alpha}$ and then move in the direction of the objective function's gradient. This steepest ascent algorithm will reach the maximum point if the learning rate was designed carefully. The partial derivative of (2) with respect to one variable is:

$$\frac{\partial W(\boldsymbol{\alpha})}{\partial \alpha_i} = 1 - y_i \sum_{j=1}^{l} \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j)\,. \tag{3}$$

A principled gradient ascent is to update the variables after one complete iteration over the whole data points. However, a fast strategy for convergence is to update them once they have been obtained which is called stochastic gradient ascent. Also a good practice is to present the data points in random order each iteration.

The constraints in the dual problem (2) restrict the feasible region and have not been considered by the previous approach. For the inequality constraints $C \geq \alpha_i \geq 0$, we ensure that α_i does not leave the box by [9]:

$$\alpha_i \leftarrow \min(C, \max(0, \alpha_i + \eta_i \frac{\partial W(\boldsymbol{\alpha})}{\partial \alpha_i}))\,. \tag{4}$$

The remaining linear constraints $\sum_{i=1}^{l} y_i \alpha_i = 0$ caused by the bias b can be ignored. The solution might not be optimal but we can represent the bias inside the input by adding one extra dimension to each data point, in which the new vector $\hat{\mathbf{x}} = (\mathbf{x}, \tau)$ and the kernel is computed as: $K(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j) = K(\mathbf{x}_i, \mathbf{x}_j) + \tau^2$. The value of τ can be any arbitrary number and some researchers choose $\tau = 1$ as [7] presented. Cristianini and Shawe-Taylor show in their book [9] an elegant proof that a safe choice of τ is the radius of a ball contains the whole data points which we can calculate as: $R = \max_{1 \leq i \leq l} \|\mathbf{x}_i\|$. They also show that the maximal gain during iterations is made by choosing $\eta_i = 1/K(\mathbf{x}_i, \mathbf{x}_i)$. The algorithm is known as kernel-Adatron and was introduced by Friess et al. [3].

2.1 Linear Kernel

Although kernel-Adatron is simple and fast but computing the gradient is expensive for large-scale data. It requires for one update sum over all nonzero α_i multiplied by the corresponding kernel. In the case of linear kernel, we have access to the feature space. Taking this advantage, we can rearrange the gradient as follows:

$$\frac{\partial W(\boldsymbol{\alpha})}{\partial \alpha_i} = 1 - \mathbf{x}_i y_i \sum_{j=1}^{l} \alpha_j y_j \mathbf{x}_j \,. \tag{5}$$

Now we can notice that the summation part remains the same after each update apart from one variable. A trick proposed by [7] is to store the summation in a vector $\mathbf{w}$ and when there an update we replaced the old α_i with the new one as shown below:

$$\mathbf{w}_{\text{new}} = \alpha_1 y_1 \mathbf{x}_1 + \ldots \alpha_i y_i \mathbf{x}_i \cdots = \alpha_1 y_1 \mathbf{x}_1 + \ldots \alpha_{\text{old}} y_i \mathbf{x}_i \cdots + (\alpha_i - \alpha_{\text{old}}) y_i \mathbf{x}_i = \mathbf{w}_{\text{old}} + (\alpha_i - \alpha_{\text{old}}) y_i \mathbf{x}_i \,. \tag{6}$$

This simple trick accelerates the kernel-Adatron algorithm for linear SVM and allowed for solving a large-scale problem [7] as shown in Algorithm 1.

Algorithm 1 Dual coordinate ascent method for 1-norm soft margin linear SVM

Require: training set S and $C \geq 0$
repeat
  Randomly shuffle S
  for $i = 1$ **to** l **do**
    $\alpha_{\text{old}} \leftarrow \alpha_i$
    $G = 1 - y_i \mathbf{w}^T \mathbf{x}_i$
    $\eta = 1/\mathbf{x}_i^T \mathbf{x}_i$
    $\alpha_i \leftarrow \alpha_i + \eta \cdot G$
    if $\alpha_i < 0$ **then**
      $\alpha_i \leftarrow 0$
    else if $\alpha_i > C$ **then**
      $\alpha_i \leftarrow C$
    end if
    $\mathbf{w} \leftarrow \mathbf{w} + (\alpha_i - \alpha_{\text{old}}) y_i \mathbf{x}_i$
  end for
until $\boldsymbol{\alpha}$ converge

2.2 Shrinking Heuristic

The dual coordinate ascent algorithm can converge faster by reducing the size of the problem (i.e. inner loop in Algorithm 1). Such a technique is known as decomposition or shrinking. When the data set is large, shrinking becomes important and it is vital when the problem requires more memory than available.

Hsieh et al. [7] proposed a shrinking technique depends on the old gradient of the objective function before removing any data point. They show that their algorithm terminates after finite iterations. In practice, the removing condition might not be satisfied in the early iterations. When the data cannot fit in memory, there is a need for disk-swapping.

We propose a shrinking heuristic from the first iteration. Less informative data points are removed from memory as soon as possible. Hence, there is only one pass over the whole data. Since the stochastic gradient method updates $\boldsymbol{\alpha}$ once they have been obtained, many variables $\alpha_i = 0$ or $\alpha_i = C$ have small chance to be changed. Therefore, the corresponding data point is removed from memory. This technique might be harsh but it

is practical for large-scale data. Surprisingly, the achieved accuracy is very close to state-of-the-art solvers as we will show in the experiments section. The remaining active set is stored in a binary file rather text for efficiency. Then we load part of the file determined by the available memory. Each loaded sub-problem is optimized for N inner iterations to reduce the outer iterations. Hence, the disk access times is limited. Our method is described in Algorithm 2.

Algorithm 2 Shrinking dual coordinate ascent method for 1-norm soft margin linear SVM

Require: training set S , $C \geq 0$ and $N \geq 1$
Open a binary file BF
for $i = 1$ **to** l **do**
  Do from line 4 to 13 in Algorithm 1
  if $\alpha_i > 0$ and $\alpha_i < C$ **then**
    Store $\mathbf{x}_i, y_i$ in BF
  end if
  Remove $\mathbf{x}_i, y_i$ from memory
end for
repeat
  $i \leftarrow 1$
  repeat
    if $\alpha_i > 0$ and $\alpha_i < C$ **then**
      Read $\mathbf{x}_i, y_i$ from BF
      $i \leftarrow i + 1$
    end if
    if memory is full or end of BF **then**
      Do lines 2-14 in Algorithm 1 for N times
      Remove all data points from memory
    end if
  until end of BF
until $\boldsymbol{\alpha}$ converge

2.3 Multiclass Problem

Multiclass problem can be solved as several binary problems using either one-versus-rest or one-versus-one strategies. The former requires K classifiers while the latter requires $K(K-1)/2$ classifiers. We choose the one-versus-rest strategy because it is simpler to implement and the number of the weight vectors $\mathbf{w}_k$ is smaller.

There are two comments on the multiclass problem. First, there is only one pass over the whole data to filter out easily classified points in each binary problem. This step saves both memory and time. Second, each binary problem is independent from the other. Hence, we might have K threads and run the algorithm in parallel. This step will speed up the process a lot significantly but our current implementation is based on a single thread.

3. EXPERIMENTS

We compare our proposed method with three state-of-the-art linear SVM solvers. Each one maximizes 1-norm soft margin SVM but approaches the problem with different method. The first solver is LIBLINEAR version 1.94, which is a dual coordinate descent method with a shrinking heuristic [7]. We used it with options '`-s 3 -B 1`'. The second solver is Pegasos which estimates the primal problem with stochastic sub-gradient descent algorithm [6]. The last one is SVM$^{\text{pref}}$ version 3.00, which is a cutting plane method solver [5].

Seven data sets have been used in our experiments; six of them are available at LIBSVM webpage. One of the data sets was introduced by Ma et al. [10] for detecting malicious web sites. Two data sets are from text classification (RCV1 [11] and WEBSPAM [12]). We also used an artificial data set (EPSILON) from PASCAL Challenge 2008 (http://largescale.ml.tu-berlin.de). Small and large handwritten digits sets were also tested (mnist [13] and mnist8m [14]). Last data set is at KDD archive (http://kdd.ics.uci.edu) for detecting legitimate connections in a computer network. Table 1 shows the training data statistics for each set. The test size for each one is: 239,613 (URL), 20,242 (RCV1), 50,000 (WEBSPAM), 100,000 (EPSILON), 10,000 (MNIST and MNIST8M) and 311,029 (KDD 1999).

All solvers are implemented in C/C++ including ours. The penalty parameter C is set to 1 following [7] (other values have no significant gain in our experiments). The equivalent setting for Pegasos parameter λ is $1/(Cl)$ and for SVM$^{\text{pref}}$ parameter C_{pref} is $0.01Cl$. The following relative difference function between $W(\boldsymbol{\alpha}*)$ (current iteration) and $W(\boldsymbol{\alpha})$ (previous iteration) is the stopping criterion defined as: $(W(\boldsymbol{\alpha}*) - W(\boldsymbol{\alpha}))/W(\boldsymbol{\alpha}*) \leq 0.01$.

There are three observations we can make about the results in Table 1. First, our solver (LLSVM) is always faster except in the first data set (URL) where Pegasos is faster. However, it has relatively low test accuracy

Table 1. Comparison between four solver on several data sets. In training data statistics, l is the number of instances and n is the number of features. The reported time is in minutes (m) seconds (s). Fields with brackets ( ) means the result is based on 25% of the data due to memory restriction in our machine (12 GB RAM). Some fields are not available (i.e. N/A) because the software does not solve multiclass problem internally.

Data Set	Training Data Statistics				LLSVM (our method)		LIBLINEAR [7]		Pegasos [6]		SVM$^{\text{pref}}$ [5]	
	class	l	n	# nonzeros	Time	Accuracy	Time	Accuracy	Time	Accuracy	Time	Accuracy
URL	2	2,156,517	3,231,961	249,347,283	2m19s	99.54%	6m21s	99.59%	1m56s	97.51%	15m4s	99.18%
RCV1	2	677,399	47,236	49,556,258	0m22s	97.67%	0m25s	97.78%	0m46s	96.03%	1m9s	97.72%
WEBSPAM	2	300,000	16,609,143	1,117,924,883	10m44s	99.00%	(2m43s)	(98.30%)	N/A	N/A	N/A	N/A
EPSILON	2	400,000	2,000	800,000,000	5m29s	89.75%	(1m48s)	(89.30%)	(4m10s)	(82.91%)	(3m5s)	(89.29%)
MNIST	10	60,000	784	8,994,156	0m33s	92.07%	0m41s	92.18%	N/A	N/A	N/A	N/A
MNIST8M	10	8,100,000	784	1,612,242,143	174m25s	90.86%	(50m28s)	(88.95%)	N/A	N/A	N/A	N/A
KDD 1999	2	4,898,431	41	61,138,555	0m30s	92.29%	0m37s	92.05%	0m41s	86.80%	1m31s	92.07%

(97.51%) compared to ours (99.53%). In fact, Pegasos always has the lowest accuracy and consumes more memory in our experiments. Second, we achieve comparable accuracy with state-of-the-art solvers and sometimes higher. Third, the proposed method could solve very large problems such as WEBSPAM, EPSILON and MNIST8M in reasonable time.

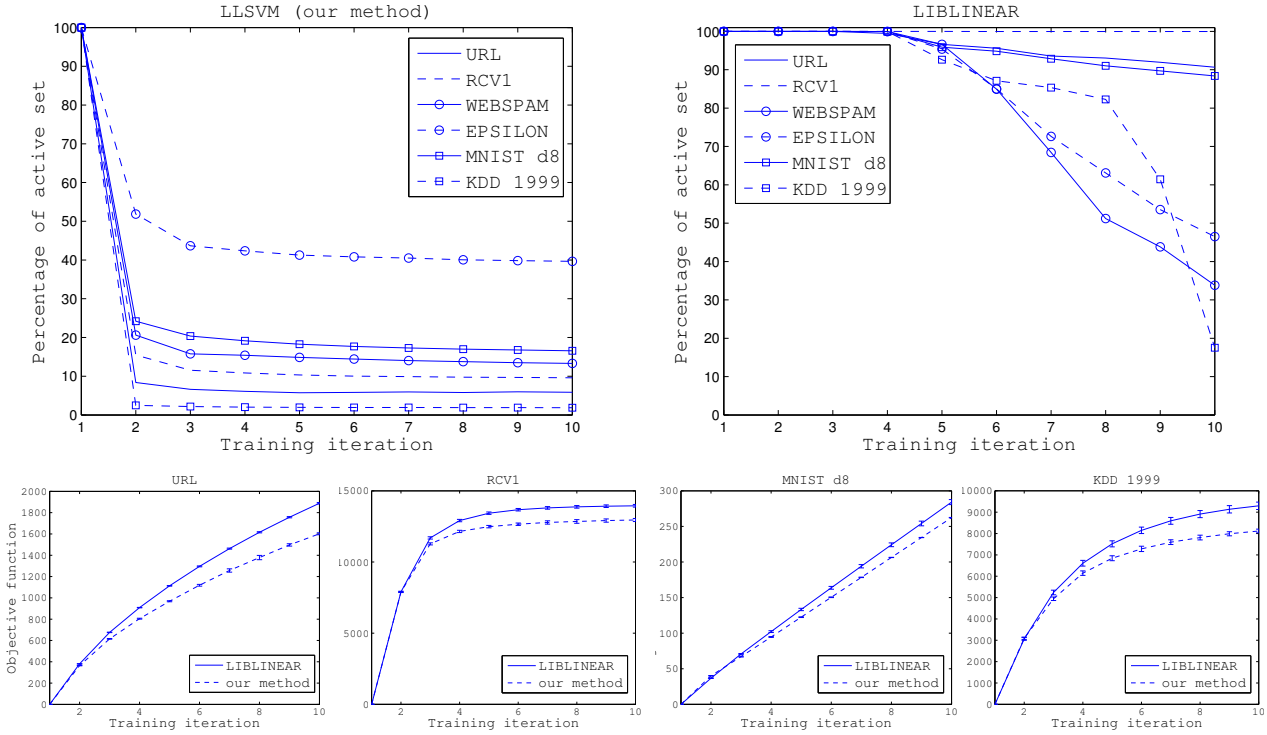

Figure 1. The active set (above) and the objective function (below) for 10 iterations in our solver and in LIBLINEAR.

Table 2. Non-active dual variables in our method compared to LIBLINEAR and the fraction of variables we mismatched.

Data Set	URL	RCV1	MNIST d8	KDD 1999
Our method	2,048,691	665,002	57,955	4,886,751
LIBLINEAR	2,018,715	662,902	55,910	4,893,975
Mismatch	0.0441	0.0183	0.0531	0.0023

Table 3. Block minimization method [8].

Data Set	WEBSPAM	EPSILON
Split	19m12s	14m10s
Train	4m48	3m26
Total Time	24m0s	17m36s
Accuracy	99.04 %	89.75 %

4. DISCUSSION

Shrinking the training data early is the main reason for the proposed method's achievement. Figure 1 shows the percentage of active set for the first 10 iterations in our solver compared with LIBLINEAR. The amount

of reduction for some data sets is huge due to the nature of the problem. For example, almost 99% of KDD is removed from the first iteration. Although the adopted harsh shrinking method reduced our objective function value (2) as illustrated in Figure 1, it seems to have little impact on the test accuracy while it largely saves memory and time. The non-active dual variables that we found from the early stage are surprisingly very similar to LIBLINEAR's solution which takes several iterations to discover the active set, as Table 2 shows.

Block minimization proposed by Yu et al. [8] is an alternative method to ours as discussed in the introduction. Table 3 reports training time and accuracy for two large-scale problems. It took 19 minutes to just split WEBSPAM and 14 minutes to split EPSILON while in Table 1 the whole training time is about 10 minutes (nearly half) and 5 minutes (nearly a third), respectively. Although we do block minimization when a data set after shrinking cannot fit in memory, dividing the remaining active set is much cheaper than the whole data.

Finally, we considered a safe choice $\tau = R$ in order to represent the bias in the input, $\hat{\mathbf{x}} = (\mathbf{x}, \tau)$, where R is the radius of a ball containing all data points. The importance of this choice rises when the data are not normalized. For example, we tried the original features in MNIST, which are scaled 0-255, to see how our solver behaves. We achieve 89.95% accuracy while LIBLINEAR achieves 85.21%.

5. CONCLUSION

We present a memory-efficient method for large-scale linear SVM. Experiments show the proposed method is comparable to state-of-the-art solvers but faster. Our shrinking heuristic could reduce up to 99% of some data sets from the first iteration. This allows the solver to process large amounts of data much faster than previous implementations. The current work focuses on two issues. First, we investigate more careful shrinking while retaining the performance. Second, we extend the approach to nonlinear SVM.

REFERENCES

[1] Boser, B. E., Guyon, I. M., and Vapnik, V. N., "A training algorithm for optimal margin classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, *COLT '92*, 144–152 (1992).

[2] Cortes, C. and Vapnik, V., "Support-vector networks," *Machine Learning* **20**, 273–297 (September 1995).

[3] Friess, T.-T., Cristianini, N., and Campbell, C., "The kernel-adatron algorithm: a fast and simple learning procedure for support vector machines," in *Mach Learn: Proc. 5th Int. Conf.*, Morgan Kaufmann (1998).

[4] Platt, J. C., "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods*, Schölkopf, B., Burges, C. J. C., and Smola, A. J., eds., 185–208, MIT Press (1999).

[5] Joachims, T., "Training linear svms in linear time," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *KDD '06*, 217–226 (2006).

[6] Shalev-Shwartz, S., Singer, Y., and Srebro, N., "Pegasos: Primal estimated sub-gradient solver for svm," in *Proceedings of the 24th International Conference on Machine Learning*, *ICML '07*, 807–814 (2007).

[7] Hsieh, C.-J., Chang, K.-W., Lin, C.-J., Keerthi, S. S., and Sundararajan, S., "A dual coordinate descent method for large-scale linear svm," in *Proc. 25th Int. Conf. on Mach. Learn.*, *ICML '08*, 408–415 (2008).

[8] Yu, H.-F., Hsieh, C.-J., Chang, K.-W., and Lin, C.-J., "Large linear classification when data cannot fit in memory," *ACM Trans. Knowl. Discov. Data* **5**, 23:1–23:23 (Feb. 2012).

[9] Cristianini, N. and Shawe-Taylor, J., *An Introduction to Support Vector Machines: And Other Kernel-based Learning Methods*, Cambridge University Press, New York, NY, USA (2000).

[10] Ma, J., Saul, L. K., Savage, S., and Voelker, G. M., "Identifying suspicious urls: An application of large-scale online learning," in *Proc. 26th Annu. Int. Conf. on Machine Learning*, *ICML '09*, 681–688 (2009).

[11] Lewis, D. D., Yang, Y., Rose, T. G., and Li, F., "Rcv1: A new benchmark collection for text categorization research," *J. Mach. Learn. Res.* **5**, 361–397 (Dec. 2004).

[12] Webb, S., "Introducing the webb spam corpus: Using email spam to identify web spam automatically," in *In Proceedings of the 3rd Conference on Email and AntiSpam (CEAS) (Mountain View*, (2006).

[13] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P., "Gradient-based learning applied to document recognition," *Proceedings of the IEEE* **86**, 2278–2324 (November 1998).

[14] Loosli, G., Canu, S., and Bottou, L., "Training invariant support vector machines using selective sampling," in *Large Scale Kernel Machines*, Bottou, L., Chapelle, O., DeCoste, D., and Weston, J., eds., 301–320, MIT Press, Cambridge, MA. (2007).