Fast Online Training of Ramp Loss Support Vector Machines

Zhuang Wang and Slobodan Vucetic
Department of Computer and Information Sciences
Temple University
Philadelphia, USA
zhuang@temple.edu and vucetic@ist.temple.edu

***Abstract*—A fast online algorithm OnlineSVM^R for training Ramp-Loss Support Vector Machines (SVM^{R}s) is proposed. It finds the optimal SVM^R for t+1 training examples using SVM^R built on t previous examples. The algorithm retains the Karush–Kuhn–Tucker conditions on all previously observed examples. This is achieved by an SMO-style incremental learning and decremental unlearning under the Concave-Convex Procedure framework. Further speedup of training time could be achieved by dropping the requirement of optimality. A variant, called OnlineASVM^R, is a greedy approach that approximately optimizes the SVM^R objective function and is suitable for online active learning. The proposed algorithms were comprehensively evaluated on 9 large benchmark data sets. The results demonstrate that OnlineSVM^R (1) has the similar computational cost as its offline counterpart; (2) outperforms IDSVM, its competing online algorithm that uses hinge-loss, in terms of accuracy, model sparsity and training time. The experiments on online active learning show that for a fixed number of label queries OnlineASVM^R (1) achieves consistently better accuracy than QueryAll and competitive accuracy to Greedy approach; (2) outperforms the active learning version of IDSVM.**

Keywords-SVM, CCCP, SMO, ramp loss, online learning

I. INTRODUCTION

Online learning is an important learning scenario in which a potentially unlimited stream of training data is presented one example at a time, and can only be seen in a single pass. This is opposed to offline learning where the whole collection of training examples is at hand. The objective of online learning is to learn a mapping $f: X \rightarrow R$ from a training stream $D = \{(x_t, y_t), t = 1, 2...\}$ that accurately predicts target variable y given an M-dimensional input vector $x \in X$. For online learning there is the anytime requirement that implies the ability to update the prediction model $f(x)$ after each new example and allows its instant use for prediction. Considering the potentially high stream rates, it becomes very important to update the model in a computationally efficient manner. Typically, online algorithms attempt to update the existing f without retraining it from scratch.

There have been many online algorithms [1, 2, 5] developed for optimization on a regularized *hinge loss* cost function. IDSVM [2] is one of the earliest successful solutions. It maintains the optimal Support Vector Machine (SVM) [16] solution after a new example is added. LASVM [1] and its improved version [8] have been proposed to speed-up online SVM training for large scale data by dropping the requirement of optimality. Similarly to their offline counterparts that use hinge-loss, their drawback is sensitivity to noise and outliers. This comes from the nature of the hinge loss cost function in which all the noisy examples become the Support Vectors (SVs) during training. Moreover, since the SVM training and prediction time grows with the number of SVs, the scalability of hinge-loss SVM is poor on noisy data.

To address drawbacks of hinge loss, recently, an efficient offline algorithm [4] based on ConCave Convex Procedure (CCCP) [17] has been proposed for SVMs by using a regularized *ramp loss* cost function. In this new Ramp-Loss SVM (SVM^R) optimization problem, the noisy examples (i.e. $y_i f(x_i) < -1$) no longer become SVs. .

In this paper, we propose a fast online modification of SVM^R for large-scale online learning tasks, called the online Ramp-Loss SVM (OnlineSVM^R). OnlineSVM^R produces the optimal solution for SVM^R on t+1 training data using the existing optimal solution on t previous examples and efficiently updating it given the new (t+1)-th example. The algorithm retains the Karush–Kuhn–Tucker (KKT) conditions on all previously observed examples using an SMO-style incremental learning and decremental unlearning approach under the CCCP framework. By only maintaining a fixed number of non-SVs closest to the decision boundary, the training time of OnlineSVM^R could be further improved while retaining competitive accuracy.

Applicability. Online learning algorithms [1, 2, 5] have been combined with other machine learning scenarios (e.g. active learning and budget learning) in many practical applications. Coupled with the higher accuracy and robustness to noise, faster training time, and sparser model, OnlineSVM^R could be used as an efficient alternative to these algorithms. To demonstrate the applicability of OnlineSVM^R, OnlineSVM^R with active learning is also proposed. Moreover, it could be shown that a greedy variant of OnlineSVM^R for active learning could be viewed as an approximate optimization process over a regularized ramp-loss cost function.

II. PRELIMINARIES

A. Zero-bias hinge loss SVM and Karush-Kuhn-Tucker conditions

We consider a linear function $f(x) = \mathbf{w}^T\Phi(x) + b$ with a bias threshold b fixed at 0, where $\mathbf{w}$ is the weight vector and

Φ is a nonlinear mapping of the attribute space. Zero bias does not lead to loss of generality [12], and can lead to the simplification of training [6, 11, 13]. From this point on we assume that form $f(x) = \mathbf{w}^T\Phi(x)$ is used.

The SVM [16] classifier $f(x)$ is trained from data set $D = \{(x_i, y_i),\ i = 1...N\}$, $x_i \in R$, $y_i \in \{-1,+1\}$ by optimizing the primal problem

$$\min P(\mathbf{w}) = \min\left(\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N} H(y_i, f(x_i))\right) \qquad (1)$$

where $H(y_i, f(x_i)) = \max(1 - y_i f(x_i), 0)$ is the *hinge loss* (see Figure 1) and C is a user specified parameter to balance the model structure and training loss.

In the dual formulation of the optimization, the primal problem is transformed to

$$\max D(\boldsymbol{\alpha}) = \max\left(\mathbf{1}^T\boldsymbol{\alpha} - \frac{1}{2}\boldsymbol{\alpha}^T\mathbf{Q}\boldsymbol{\alpha}\right) \qquad (2)$$
$$\text{s.t. } \mathbf{0} \le \boldsymbol{\alpha} \le \mathrm{C}\cdot\mathbf{1},$$

where $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_N)$ are the Lagrange multipliers associated with the primal problem, $Q_{ij} = y_i y_j k(x_i, x_j)$ is the element of the Gram kernel matrix $\mathbf{Q}$, and k is the positive definite kernel function satisfying $k(x_i, x_j) = \Phi(x_i)^T\Phi(x_j)$. Note that, unlike the standard SVM dual form, the linear constraint $\mathbf{y}^T\boldsymbol{\alpha} = 0$ vanished in (2) because of $b = 0$.

The Karush–Kuhn–Tucker (KKT) conditions are necessary and sufficient for the optimal solution of (2),

$$g_i = \frac{\partial D(\boldsymbol{\alpha})}{\partial \alpha_i} = 1 - y_i f(x_i) \begin{cases} < 0; & \alpha_i = 0 \\ = 0; & \alpha_i \in (0, C) \\ > 0; & \alpha_i = C. \end{cases} \qquad (3)$$

We call the examples not satisfying (3) the *KKT violators*. The resulting SVM classifier can be conveniently represented in the dual form as

$$f(x) = \sum\nolimits_i y_i \alpha_i k(x_i, x)\,. \qquad (4)$$

The training examples with $\alpha_i \neq 0$ are called Support Vectors (SVs).

B. Ramp loss SVM

One practical issue with SVM is its scalability. Recent results [15] show that the number of SVs scales linearly with the number of training examples. The SV scaling property is in fact a property of the hinge loss function. It could be shown through differentiating (1) (and assuming the hinge loss is differentiable at 1 with a smooth approximation) [4] that the optimal solution $\mathbf{w}$ must satisfy

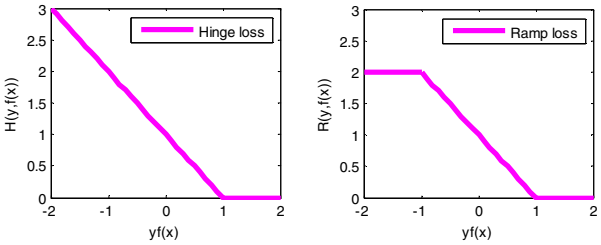


Figure 1. Hinge loss vs. Ramp loss

$$\mathbf{w} = -C\sum\nolimits_i y_i H'(y_i, y_i f(x_i))\Phi(x_i)\,. \qquad (5)$$

Thus, all examples with $y_i f(x_i) \le 1$ become SVs as their hinge-loss derivative is nonzero (i.e. $H'(y_i, f(x_i)) = 1$). This includes all noisy examples with $y_i f(x_i) < -1$.

To address this drawback of hinge loss, the non-convex *ramp loss*

$$R(y_i, f(x_i)) = \begin{cases} 0, & y_i f(x_i) > 1 \\ 1 - y_i f(x_i), & -1 \le y_i f(x_i) \le 1 \\ 2, & y_i f(x_i) < -1 \end{cases} \qquad (6)$$

(see Figure 1) was introduced in [4]. Replacing H by R in (1), it could be seen that the examples with $y_i f(x_i) < -1$ do not become SVs. The new ramp loss SVM ($\mathrm{SVM}^{\mathrm{R}}$) formulation then reads as

$$\min P^R(\mathbf{w}) = \min\left(\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N} R(y_i, f(x_i))\right). \qquad (7)$$

C. ConCave-Convex Procedure (CCCP)

To solve the non-convex objective function (7), CCCP [17] is used. CCCP optimizes the non-convex problem by solving a sequence of approximate convex problems and has no additional parameters to tune.

To optimize (7) using CCCP, the non-convex objective function is decomposed into convex $J_{vex}(\mathbf{w})$ and concave $J_{cave}(\mathbf{w})$ parts, which can be rewritten as

$$\min P^R(\mathbf{w}) = \min\left(\underbrace{\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N} H(y_i, y_i f(x_i))}_{J_{vex}} + \underbrace{C\sum_{i=1}^{N}\left(R(y_i, y_i f(x_i)) - H(y_i, y_i f(x_i))\right)}_{J_{cave}}\right). \qquad (8)$$

CCCP runs in iterations. First, the procedure sets $\mathbf{w}^{old}$ with an initial guess. In each iteration, the procedure approximates $J_{cave}(\mathbf{w})$ by its tangent $\mathbf{w}^{\mathrm{T}} J_{cave}'(\mathbf{w}^{old})$, where $\mathbf{w}^{old}$ indicates $\mathbf{w}$ in the previous iteration, and then optimizes the resulting objective function

$$\mathbf{w}^{new} = \arg\min_{w}\left(\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N} H(y_i, y_i f(x_i)) + \mathbf{w}^T \sum_{i\notin V} y_i \Phi(x_i)\right). \quad (9)$$

where V is the active subset of training examples with margin larger than -1, formally,

$$V = \{\forall i, y_i f(x_i) = y_i((\mathbf{w}^{old})^T \Phi(x_i)) \geq -1\}. \quad (10)$$

There is the convergence guarantee [17] that the procedure would decrease the non-convex objective function in each iteration.

The resulting optimization problem (9) is convex and can be reformulated by its dual form:

$$\max D(\boldsymbol{\alpha}_V) = \max\left(\mathbf{1}^T \boldsymbol{\alpha}_V - \frac{1}{2}\boldsymbol{\alpha}_V^T \mathbf{Q}_{VV} \boldsymbol{\alpha}_V\right) \quad \text{s.t. } \mathbf{0} \leq \boldsymbol{\alpha}_V \leq \mathrm{C}\cdot\mathbf{1}, \quad (11)$$

where $\boldsymbol{\alpha}_V$ and $\mathbf{Q}_{VV}$ correspond to the active set examples and $\boldsymbol{\alpha}$'s of examples in $\overline{V} = D - V$ is set to zero. It is worth noting that the optimization problem (11) is equivalent to training a standard SVM on V.

Using the CCCP algorithm, the procedure iteratively retrains SVMs on the dynamically updated V until V remains unchanged, and eventually has all the SVs within the ramp region (i.e. $\forall i, \alpha_i \neq 0 \Leftrightarrow |y_i f(x_i)| \leq 1$).

Cons. Frequently retraining SVM from scratch and reallocating V is computationally expensive even for offline training of SVM$^{\mathrm{R}}$. Directly applying the offline SVM$^{\mathrm{R}}$ algorithm [4] to online learning by retraining each time a new example is observed is impractical. This drawback leads us to the proposed efficient online algorithm that formulates the optimal solution for t+1 examples in terms of the existing solution for t previous examples and efficiently accounting for the new (t+1)-th example.

III. ONLINE RAMP LOSS SVMS (ONLINESVM$^{\mathrm{R}}$)

In this section, we describe our proposed online training algorithm for ramp loss SVM, called OnlineSVM$^{\mathrm{R}}$. OnlineSVM$^{\mathrm{R}}$ guarantees the optimal solution of (7) after every new training example is received.

An efficient online modification of CCCP serves as the core of OnlineSVM$^{\mathrm{R}}$. The key of this modification is to retain the KKT conditions on all previously seen examples, while iteratively adding or deleting an example from the active set V (see (10)). To achieve this, an SMO-style incremental learning and decemental unlearning method is proposed to guarantee the KKT conditions are maintained during the update. An efficient working set selection strategy is designed for this method that minimizes kernel computations and achieves faster convergence.

Algorithm 1: OnlineSVM$^{\mathrm{R}}$

0. $f(x)$=0, $I = \varnothing$, $V = \varnothing$
1. **repeat**
2. receive the next example (x_{t+1}, y_{t+1});
3. $I = I \cup \{t+1\}$, $\alpha_{t+1} = 0$, $g_{t+1} = 1 - y_{t+1} f(x_{t+1})$;
4. **if** $g_{t+1} \in [0,2]$
5. $V = V \cup \{t+1\}$;
6. calculate α_{t+1}^{new} using Eq. (14);
7. set $\Delta\alpha_{t+1} = \alpha_{t+1}^{new} - \alpha_{t+1}^{old}$;
8. $\forall i \in I$, update g_i using Eq. (15);
9. **repeat**
10. **while** ($\exists i \in V$ violates Eq. (3) or Eq.(13) wouldn't be improved much)
11. find a KKT violator i^* in V using Eq. (13);
12. calculate $\alpha_{i^*}^{new}$ using Eq. (14);
13. set $\Delta\alpha_{i^*} = \alpha_{i^*}^{new} - \alpha_{i^*}^{old}$;
14. $\forall i \in I$, update g_i using Eq. (15);
15. **endwhile**
16. **for** $\forall i \in I$
17. **if** $g_i \in (-\infty, 2]$ & $i \notin V$
18. $V = V \cup \{i\}$;
19. **elseif** $g_i \in (2, +\infty)$ & $i \in V$
20. $V = V - \{i\}$;
21. **endif**
22. **endfor**
23. **for** $\forall i$, just being removed from V
24. set $\alpha_i^{new} = 0$ and $\Delta\alpha_i = -\alpha_i^{old}$;
25. $\forall k \in I$, update g_k using Eq. (15);
26. **endfor**
27. **until** V is not changed
28. **endif**
29. **until** the end of the stream

A. Outline of OnlineSVM$^{\mathrm{R}}$ (*Algorithm 1*)

We outline the underlying structure of OnlineSVM$^{\mathrm{R}}$ in this section. The details are given in the following sections.

OnlineSVM$^{\mathrm{R}}$ starts by initializing a zero classifier and creating empty sets I and V. Here, I is a set of all the observed examples and V is a subset of I as defined in (10).

At time t+1, after adding the newly received example t+1 into I, initializing $\alpha_{t+1} = 0$, and calculating the gradient g_{t+1} (defined in (3)), the algorithm checks the KKT conditions for this initial assignment of t+1:

- If $g_{t+1} < 0$, then t+1 satisfies the KKT conditions of (11) and thus the existing solution of (7) from the first t examples is still optimal after its inclusion.

- If $g_{t+1} > 2$, then t+1 is a KKT violator and is at the noisy region of ramp loss; thus, it would not change the current V (according to the definition of V in (10)) and the current optimal solution of (7) is still valid.
- Otherwise, t+1 is within the linear region of ramp loss (i.e. $g_{t+1} \in [0,2]$), and thus the model is updated.

In the event of model update, (t+1)-st example is added to V, α_{t+1} is calculated and then the KKT conditions for *all the observed examples* (set I) are updated with respect to the obtained α_{t+1}. The above step (and, similarly, Steps 14 and 25) is important. The updating of KKT conditions for V guarantees the progress of optimization at this step, while the updating for the examples outside V is for a fast relocation of the new V in future steps. Following this, the algorithm updates the solution through online CCCP iterations (Steps 9 to 27). In the first part (Steps 10 to 15) of this iterative process, (11) is iteratively optimized on V until the stopping criteria is reached. After that, V is quickly reassigned with respect to Eq. (10) (Steps 16 to 22). If V is modified, the examples removed from the previous V are decrementally unlearned from the current solution and *all* KKT conditions are updated accordingly (Steps 19 to 22). This process (Steps 9 to 27) iterates until V stops changing. Thus, the optimal solution of (7) is maintained upon addition of a new example.

B. Online updating

In this section we give details about the online optimization used in Algorithm 1. As we already pointed out, the optimization of (11) is equivalent to training an SVM on V and setting α values of the examples outside V to zero. For the online updating, the goal is to maintain KKT conditions on all the previously seen examples. To achieve this, we propose an SMO-style online updating. SMO [14] is an iterative process where in each iteration the smallest set of examples is selected, and the dual problem is optimized with respect to them. This smallest set is called the *working set.*

In our case, the working set only involves a single example because the fixed threshold b removes the need for the linear constraint in the dual. Hence, the smallest sub-optimization problem when the example i is used as the working set reads as

$$\max D(\alpha_i) = \max\left((1 - \mathbf{Q}_{iU}\boldsymbol{\alpha}_U^{old})\alpha_i - \frac{1}{2}\alpha_i Q_{ii}\alpha_i \right) \qquad (12)$$

$$\text{s.t. } 0 \le \alpha_i \le C,$$

where $U = V - \{i\}$ denotes set of all the examples excluding the i-th example.

There are many ways to select i. The working set selection is closely related to the optimization progress and the cost of kernel computation. From a greedy view, the most efficient working set in each iteration should be the one that achieves the largest improvement of the dual objective function (12). Formally, the optimal selection is obtained as

$$\begin{aligned} i &= \arg\max_{j \in V} \left(D(\alpha_j^{new}) - D(\alpha_j^{old}) \right) \\ &= \arg\max_{j \in V} \left(\frac{1}{2} Q_{jj} \left((\alpha_j^{old})^2 - (\alpha_j^{new})^2 \right) \right. \\ &\quad \left. + (g_j^{old} + Q_{jj}\alpha_j^{old})(\alpha_j^{new} - \alpha_j^{old}) \right) \end{aligned} \qquad (13)$$

The computation of (13) is cheap, merely proportional to the number of KKT violators in V. An appealing part of this procedure over the other popular working set selection methods [7, 8] is that it is free of kernel computation and all the values can be directly read from memory if some specific kernels are applied (e.g. if RBF kernel is used, then Q_{ii} becomes constant).

The optimal solution of (12) leads to update

$$\alpha_i^{new} = \begin{cases} 0, & \text{if } \alpha_i^{old} + g_i / Q_{ii} < 0, \\ C, & \text{if } \alpha_i^{old} + g_i / Q_{ii} > C, \\ \alpha_i^{old} + g_i / Q_{ii}, & \text{otherwise.} \end{cases} \qquad (14)$$

After α_i is updated, the KKT conditions of *all training examples* are updated as well, with respect to the new α_i. Specifically,

$$\begin{aligned} g_k^{new} &= \frac{\partial D(\boldsymbol{\alpha}^{new})}{\partial \alpha_k} = 1 - y_k f^{new}(x_k) \\ &= 1 - y_k \left(f^{old}(x_k) + y_i \Delta\alpha_i k(x_i, x_k) \right) \\ &= g_k^{old} + y_k y_i \Delta\alpha_i k(x_i, x_k) \end{aligned} \qquad (15)$$

where $\Delta\alpha_i = \alpha_i^{new} - \alpha_i^{old}$. Given an SVM solution on the current V and a newly added example, iterative execution of Steps 11~14 in Algorithm 1 resolves one KKT violation at a time, thus increasing the dual objective (11) and guaranteeing convergence.

Decremental unlearning due to removal of an example from V can be implemented naturally using the same approach as above. For i to be unlearned, we first set $\alpha_i = 0$ and then update the remaining α values in set I using (14). We repeat this procedure for every example being removed from V (Steps 23~26 in Algorithm 1).

Stopping criteria. We use two different stopping criteria for the model optimization (the condition in Step 10 in Algorithm 1). The first one checks the KKT conditions (3). If all g_i are less than 10^{-3} from the desired value the procedure stops. This method is standard, and is also used in SMO [14]. The second criterion considers the optimization progress; if the dual objective function could not significantly improve (e.g. less than 10^{-5}), as measured by (13), then the iterations stop. Using (13) to estimate progress could significantly reduce the computation cost. These two criteria (with 10^{-3} and 10^{-5} thresholds) are the default setting for our algorithms.

C. Improving scalability

It is possible to modify Algorithm 2 to trade-off some accuracy for quicker update time and lower memory consumption. This can be achieved by only maintaining in I only a fixed number of the non-SVs that are closest to the

decision boundary (this can be directly measured by $|1 - g_i|$), and discarding the remaining ones. The non-SVs that are far from the decision boundary are not very likely to become SVs in the future. In practice, this removal step could be executed right after Step 28 in Algorithm 1 as described in Algorithm 2. There, removal criterion is triggered (when a predefined maximal number of non-SVs is reached).

Algorithm 2: The removal step

1. **if** $|X| > m$ /* $X = \{\forall i, \alpha_i = 0\}$, m is a predefined value*/
2. remove ($|X| - m$) non-SVs with the largest $|1 - g|$
3. **endif**

IV. ONLINESVM^R WITH ACTIVE LEARNING AS NON-CONVEX OPTIMIZATION

OnlineSVM^R can be modified for online active learning, where labels are queried only for some of the training examples from the stream. Instead of querying labels from all the observed examples, OnlineASVM^R (Algorithm 3) only queries examples within the ramp region (i.e. $|f(x)| \leq 1$). In other respects, it is identical to OnlineSVM^R. The justification comes from the fact that an example outside of the ramp region is not placed in V (it is placed in I though), its α value is set to zero, and the solution remains unchanged. Only if such an example finds itself within the ramp region during the training procedure, it is placed in V and its α value becomes nonzero, and thus the solution is improved. The farther away the new example is from the ramp region the smaller the chance that it will ever be used to update the solution.

More recently, a novel view of active learning as a non-convex optimization problem was suggested [9]. There, the perceptron algorithm with such label filtering step is viewed as a stochastic gradient descent over a ramp loss cost function. Following this view, we argue that *OnlineASVM^R could be considered as a greedy optimization approach that approximately optimizes the regularized ramp loss cost function (7)*.

Algorithm 3: OnlineSVM^R with active learning (OnlineASVM^R)

0. $f(x)=0$, $I = \varnothing$, $V = \varnothing$
1. **repeat**
2. receive the next unlabeled example x_{t+1};
3. **if** $|f(x_{t+1})| \leq 1$
4. query label y_{t+1};
5. $I = I \cup \{t+1\}$, $V = V \cup \{t+1\}$;
6. $g_{t+1} = 1 - y_{t+1} f(x_{t+1})$, $\alpha_{t+1} = 0$;
7. Steps 6 to 27 from Algorithm 1;
8. **endif**
9. **until** the end of the stream

V. EXPERIMENTS

A. Experimental setting

Datasets. We performed experiments on 9 benchmark binary classification data sets summarized in the first column of Table 1. The multi-class data sets were converted to two-class sets as follows. For the digit dataset *USPS* we converted the original 10-class problems to binary by representing digits 1, 2, 4, 5, 7 (non-round digits) as negative class and digits 3, 6, 8, 9, 0 (round digits) as positive class. For 3-class *DNA* data set class 3 was separated from the other 2 classes. Class 1 in the 3-class *Waveform* was treated as negative and the remaining two as positive. For *Covertype* data the class 2 was treated as positive and the remaining 6 classes as negative. *Adult*, *Banana*, *Checkerboard* and *Gauss* were originally 2-class data sets. *NCheckerboard* is a noisy version of *Checkerboard* where class assignment was switched for 15% of the randomly selected examples. For both data sets, we used the noise-free *Checkerboard* as the test set. Attributes in all data sets were scaled to mean 0 and standard deviation 1.

Algorithms. We used two online and one offline algorithm with hinge loss to compare with OnlineSVM^R and OnlineASVM^R:

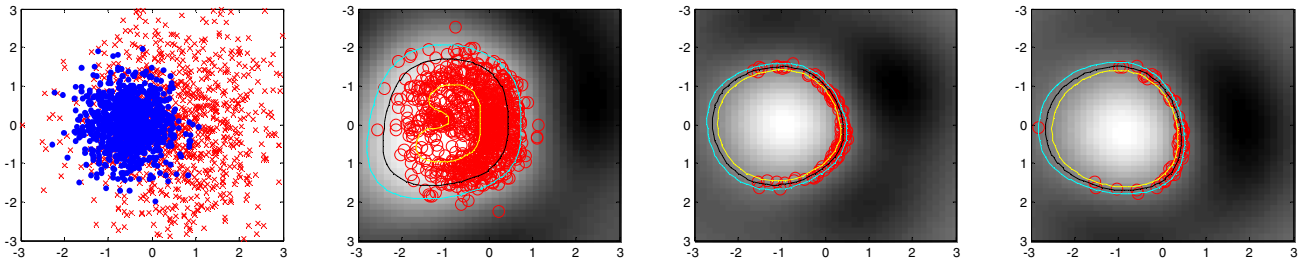
- IDSVM [2]: an online SVM algorithm which is guaranteed to achieve the optimal solution.
- Online Passive-Aggressive (PA) algorithm [5]: a popular online algorithm based on perceptron-style updating.
- LibSVM [3]: one of the most successful offline SVM algorithms with hinge loss.

Hyperparameter tuning. RBF kernel, $k(x_i, x_j) = \exp(-||x_i - x_j||^2 / 2\delta^2)$, was used in all experiments. We selected the best hyper-parameters C and δ^2 using cross-validation for all combinations of algorithm and data set. The considered values were $C = \{0.1, 1, 5, 10, 50, 100, 500\}$ and $\delta^2 = \{M/2^{-1}, M/2^0, M/2^1, M/2^2, M/2^4, M/2^6\}$, where M is data dimensionality.

B. Illustration on a 2-D dataset

Let us first illustrate the detailed comparison between the proposed and the previous algorithms on *Gauss* data set. *Gauss* is a noisy benchmark dataset [10] which consists of two overlapping 2-D Gaussian distributions (see Figure 2.a).

OnlineSVM^R & OnlineASVM^R vs IDSVM. In Figures 2.b-d the IDSVM, OnlineSVM^R, and OnlineASVM^R solutions are compared. It can be observed that OnlineSVM^R and OnlineASVM^R solutions are much sparser than that of IDSVM. As expected, their SVs are all inside the margins, while the SVs of the IDSVM solution are widely distributed and also include the noisy examples outside the margin. In Figure 3.a we show the accuracy as a function of the data stream size for the three algorithms (IDSVM was terminated after 5,000 seconds of training because it reached the resource limits of our PC). It can be seen that OnlineSVM^R consistently achieved higher accuracy than IDSVM, especially in the initial stages of training (see the initial points of two curves). The accuracy curve of OnlineASVM^R



(a) Data plot (b) IDSVM (c) OnlineSVM^R (d) OnlineASVM^R

Figure 2. Plots of solution on Gauss data set. Red circles are SVs, black line is decision boundary, and yellow and cyan lines are positive and negative margins.

closely follows that of OnlineSVM^R and reaches it after 10,000 observed examples. This is expected because, as the accuracy of OnlineASVM^R increases, there is less of a risk that an unlabeled point will end up within the ramp region. Figure 3.b illustrates that OnlineSVM^R and OnlineASVM^R learn sparser models than IDSVM.

Computational improvement. First, we show the computational cost for (1) OnlineSVM^R, (2) offline SVM^R [4] trained on demand, and (3) the naive online algorithm that repeatedly retrains offline SVM^R from scratch in an online setting. To reduce the influence of different implementation, the computational cost is expressed as total number of kernel computations. For offline SVM^R training, the initial model was trained on a subset of training data to speed-up computation, as suggested in [4]. From Figure 4.a, we can observe that the computational cost of OnlineSVM^R is significantly less than the naive approach of retraining SVM^R . Interestingly, total cost of OnlineSVM^R training is comparable to training a single offline SVM^R. This result clearly demonstrates the success of the proposed online solution. In Figure 4.b, we compare the training time of OnlineSVM^R and OnlineASVM^R with IDSVM. As can be seen, the proposed two algorithms lead to a large reduction in training time over IDSVM due to the sparser models.

C. *Results on benchmark datasets*

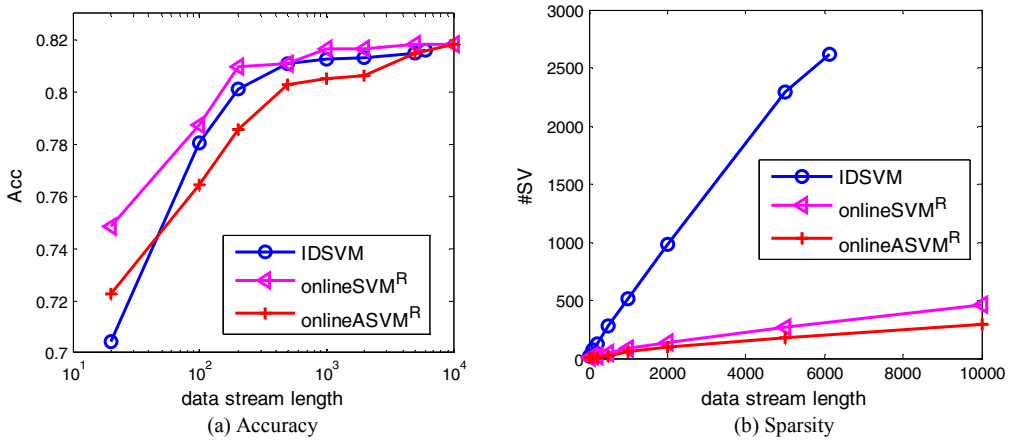
The results on 9 benchmark data sets are summarized in Table 1. Mean and standard deviation of 10 repeated experiments for accuracy and number of SVs are reported. If needed, we terminated IDSVM after 5,000 seconds of training.

Accuracy. On the accuracy side, we can observe that OnlineSVM^R is significantly more accurate than LibSVM on 5 out of 9 data sets and IDSVM on 4 out of 9 data sets. The largest accuracy improvement (up to 2%) happens on two noisy data sets *NCheckerboard* and *Covertype*. The accuracy of OnlineASVM^R is very competitive to both IDSVM, LibSVM and OnlineSVM^R and is usually within 1% of their accuracy. Finally, perceptron-based PA algorithm is less competitive and it is significantly less accurate than the others on 7 out of 9 data sets.

Sparsity. On the model sparsity side, OnlineASVM^R achieves the sparest model on 6 out of 9 data sets. OnlineSVM^R is as sparse as OnlineASVM^R in most cases, and it obtains the sparsest model on 2 data sets. Both SVM^R-based algorithms are much sparser than the hinge-loss based IDSVM, LibSVM and PA. In *Gauss* data set, OnlineSVM^R and OnlineASVM^R are about 10 times sparser than LibSVM. PA results in the densest models. In the extreme case, for *Cover* data set, PA includes all the training data as SVs.

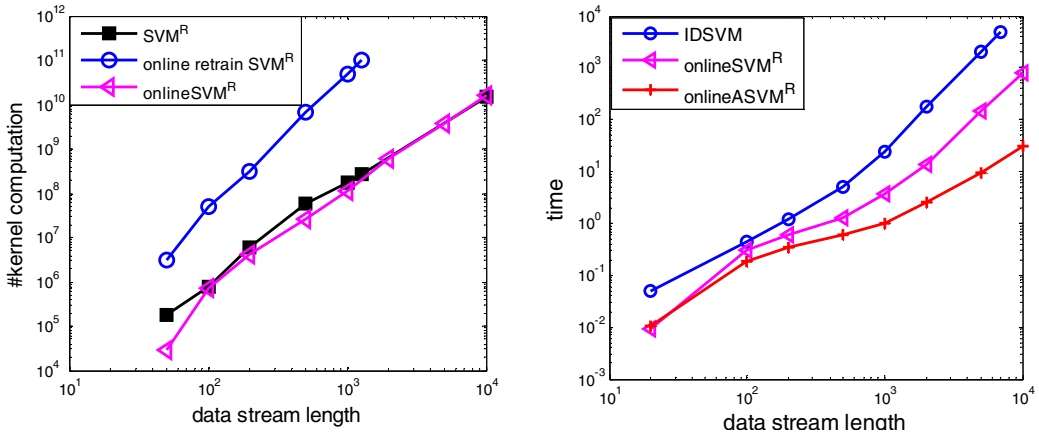
D. *Scalability*

In the scalability experiment, we evaluated OnlineSVM^R with the removal step (Algorithm 3). We explored how the



(a) Accuracy (b) Sparsity

Figure 3. Comparisons of accuracy and sparsity on *Gauss* data set



(a) OnlineSVM^R vs SVM^R & Naïve online retrain SVM^R (b) OnlineSVM^R & OnlineASVM^R vs IDSVM

Figure 4. Comparison of computational cost as a function of data stream length on *Gauss* data.

parameter m trades off accuracy and scalability. From Figures 5.a and c, it can be seen that the computational cost of OnlineSVM^R decreases with m. For aggressive removal with $m = 0$, the accuracy loss is significant and should not be recommended (see Figures 5.b and d). Choice of $m = 100$ is more appropriate, as it results in only a small accuracy loss that decreases with data stream size and still allows significant savings in computational time. If achieving the best accuracy is a critical requirement, $m = 1000$ seems to be a good choice because accuracy loss in negligible and computational savings are still significant.

E. Online active learning experiments

In sections 5.2 and 5.3 we have already observed that OnlineASVM^R achieves comparable accuracy and number of SVs to OnlineSVM^R and that it is somewhat faster. In this section, we give detailed results about growth in accuracy of OnlineASVM^R as a function of the labeling effort. To gain better insight, we compared its performance with 2 competing active learning approaches implemented on top of OnlineSVM^R and IDSVM algorithms:

- QueryAll: ask the label of every observed example. This is identical to the standard online learning and serves as the baseline approach.
- Greedy: ask label of the example with the largest uncertainty (i.e. $\arg\min(|f(x)|)$) from the h most recently observed examples. In our experiments, we used h=5 as proposed before [9]. Note that when h=1, Greedy is identical to QueryAll.

TABLE I. ACCURACY AND MODEL SPARSITY COMPARISON ON BENCHMARK DATA SETS.

Algorithms / Datasets		LibSVM [3]	IDSVM [2]	PA [5]	OnlineSVM^R	OnlineASVM^R
Adult	*Acc*	84.0±0.3	84.3±0.2	82.3±0.2	**84.5±0.2**	**84.5±0.2**
(10000×123)	*#SV*	4275±57	4093±50	7764±19	2631±68	**2497±54**
Banana	*Acc*	90.0±1.0	90.0±0.8	88.7±0.4	**90.8±1.0**	89.9±0.4
(4300×2)	*#SV*	926±17	995±14	1195±16	713±16	**303±11**
Checkerborad	*Acc*	99.2±0.1	99.1±0.2	97.1±0.3	**99.5±0.0**	99.0±0.2
(10000×2)	*#SV*	455±13	477±17	1990±29	492±2	**438±11**
NCheckerboard	*Acc*	96.5±0.3	96.6±0.5*	89.9±0.7	**98.6±0.2**	98.0±0.2
(10000×2)	*#SV*	4679±27	3196±48*	5605±66	**554±19**	987±31
Cover	*Acc*	79.8±0.4	80.8±0.4*	81.5±0.4	**81.8±0.4**	80.8±0.4
(10000×54)	*#SV*	5164±82	3529±22*	10000±0	3898±69	**2408±135**
DNA	*Acc*	95.0±0.0	**95.2±0.0**	93.8±0.2	95.1±0.1	**95.2±0.0**
(2000×180)	*#SV*	1080±1	817±0.4	1680±2	**796±4**	1097±4
Gauss	*Acc*	**81.9±0.3**	81.5±0.6*	76.4±5	81.8±0.2	81.8±0.2
(10000×123)	*#SV*	4183±74	2522±76*	4767±47	465±93	**304±27**
USPS	*Acc*	97.3±0.0	97.3±0.0	96.7±0.3	**97.5±0.1**	97.1±0.0
(7291×256)	*#SV*	**1126±0**	1180±0	2205±18	1166±10	1152±14
Waveform	*Acc*	88.5±0.2	**89.7±0.5**	89.0±0.4	89.4±0.3	89.5±0.4
(10000×21)	*#SV*	2629±45	2545±43	8273±19	1796±51	**1711±41**

The highest accuracy and the smallest #SV for each data set are in bold. * means early stopped after 5,000 seconds.

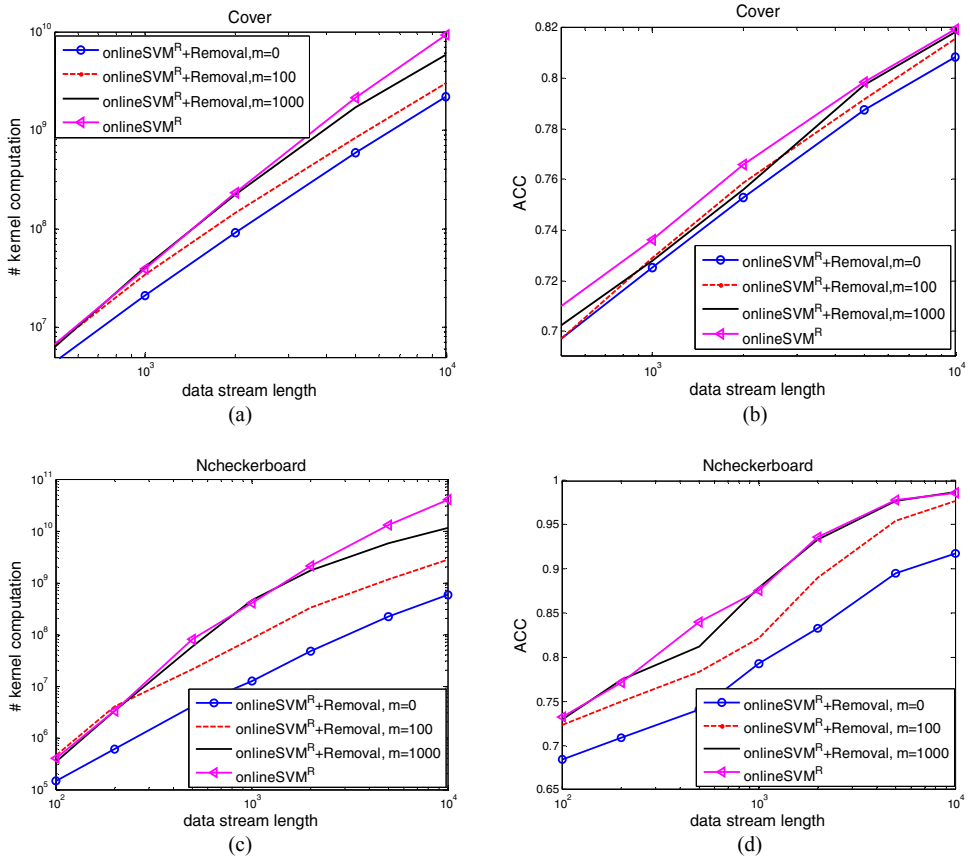


Figure 5. Influence of number of non-SVs on accuracy and computational cost on *Ncheckerboard* and *Cover* data sets.

Figure 6 shows results for each of the 9 data sets. The maximal number of labels in each plot is obtained when one of the 5 competing algorithms reaches the end of the data stream.

Let us first discuss performance of 3 active learning algorithms based on OnlineSVM^R. The QueryAll strategy is consistently and significantly less accurate than Greedy and OnlineASVM^R. OnlineASVM^R and Greedy appear to have strengths on different data sets. While Greedy is superior on *DNA*, *USPS*, and *Waveform* data sets, OnlineASVM^R is superior on *Gauss*, *Cover*, *Checkerboard*, and *NCheckerboard*. Their performance on the remaining 2 data set is similar. These results are interesting because Greedy focuses on examples near the decision boundary, while OnlineASVM^R is gentler and labels all examples within the margin. It would be interesting to explore in future work if the size of labeling region can be optimized to each individual data set.

Comparing ramp-loss OnlineSVM^R and hinge-loss IDSVM, it is evident that OnlineSVMR is superior on majority of data sets. This is another proof of the success of the ramp-loss measure.

VI. CONCLUSIONS

We presented a fast online algorithm for ramp loss SVM training. The proposed algorithm OnlineSVM^R significantly outperforms its competing algorithm IDSVM in terms of accuracy, model sparsity and training time. A variant of OnlineSVM^R for active learning could be viewed as an approximate optimization process over a regularized ramp loss cost function. The applicability of OnlineSVM^R in active learning scenario has been demonstrated, as it showed the advantages over the competing algorithm. The application of OnlineSVM^R to other machine learning scenarios (e.g. budget learning) could be further studied.

ACKNOWLEDGMENT

This work was supported by the U.S. National Science Foundation under Grant IIS-0546155.

REFERENCES

[1] Bordes, A., Ertekin, S., Wesdon,.J. and Bottou, L., "Fast kernel classifiers for online and active learning", *JMLR*, 2005.

[2] Cauwenberghs, G. and Poggio, T., "Incremental and decremental support vector machine learning", *NIPS*, 2001.

[3] Chang, C.-C. and Lin, C.-J., LIBSVM: a library for support vector machines, 2001.

[4] Collobert, R., Sinz, F., Weston, J. and Bottou, L., "Trading convexity for scalability", *ICML*, 2006.

[5] Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S. and Singer, Y. "Online passive-aggressive algorithms". *JMLR*, 2006.

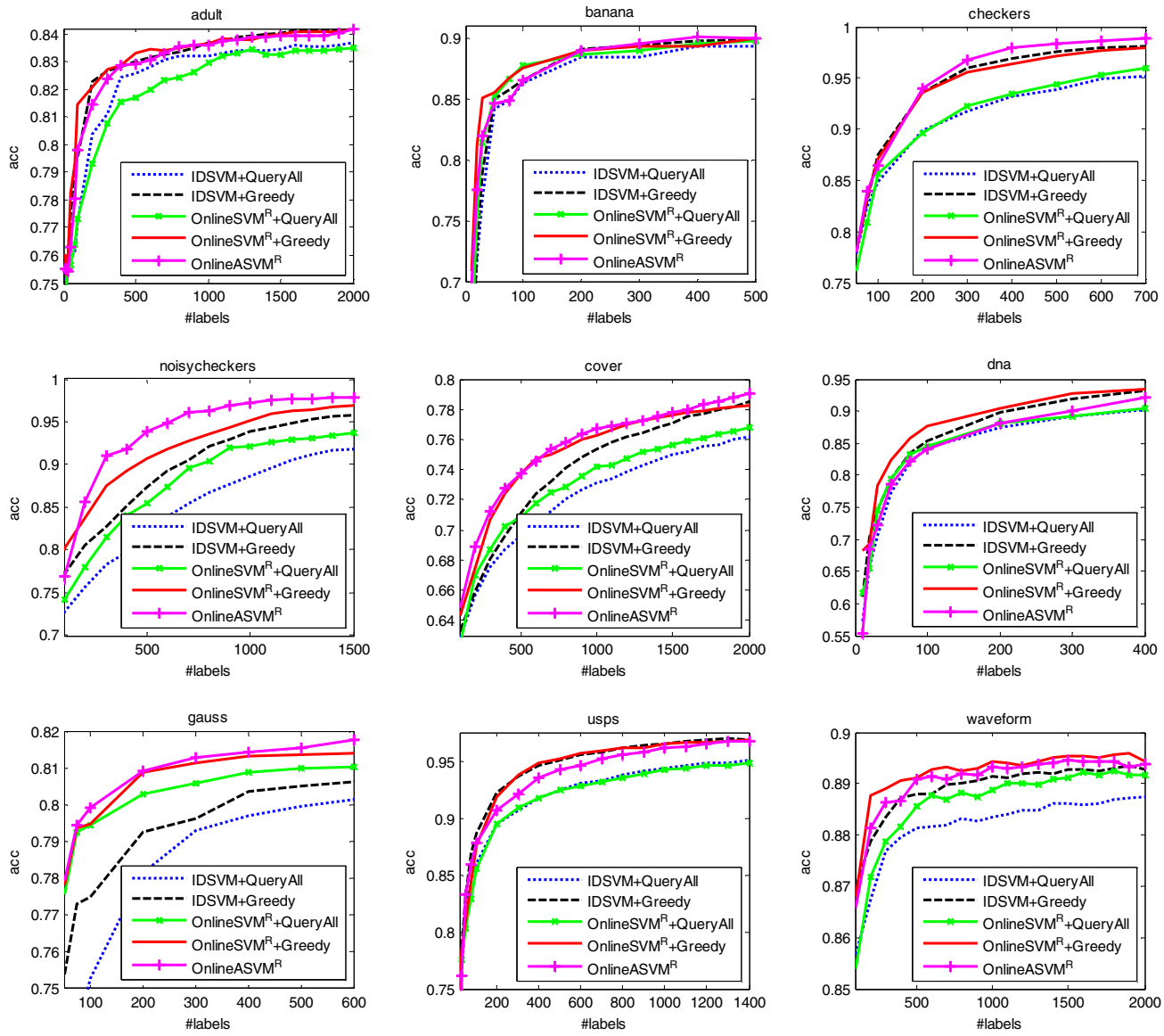


Figure 6. Accuracy vs. #queried labels for online active learning algorithms on 9 datasets

[6] Engel, Y., Mannor, S. and Meir, R., "Sparse online greedy support vector regression", *ECML*, 2002.

[7] Fan, R.-E., Chen, P.-H., and Lin, C.-J., "Working set selection using second order information for training SVM", *JMLR*, 2005.

[8] Glasmachers, T. and Igel, C., "Second order SMO improves SVM online and active Learning", *Neural Computation*, 2008.

[9] Guillory, A., Chastain, E. and Bilmes, J., "Active learning as non-convex optimization", *AISTATS*, 2009.

[10] Haykin, S., *Neural Networks*, Prentice Hall. Inc., 1999.

[11] Lee, C., Kim, H. and Jang, M., "Fixed-threshold SMO for joint constraint learning algorithm of structural SVM", *SIGIR*, 2008.

[12] Li, Y. and Long, P., "The relaxed online maximum margin algorithm", *Machine Learning*, 2002.

[13] Mangasarian, O. L. and Musicant, D. R., "Active support vector machine classification", *NIPS*, 2000.

[14] Platt, J. C., "Fast training of support vector machines using sequential minimal optimization", *Advances in Kernel Methods - Support Vector Learning*, MIT Press, 1998.

[15] Steinwart, I., "Sparseness of support vector machines", *JMLR*, 2003.

[16] Vapnik, V. N., *Statistical Learning Theory*, John Wiley Sons, Inc., 1998.

[17] Yuille, A. L. and Rangarajan, A., "The concave convex procedure (CCCP)", *NIPS*, 2002.