Incremental Sensitivity Analysis for Kernelized Models

Hadar Sivan[1](✉), Moshe Gabel[2], and Assaf Schuster[1]

[1] Technion - Israel Institute of Technology, Haifa 3200, Israel
{hadarsivan,assaf}@cs.technion.ac.il
[2] University of Toronto, Toronto, Canada
mgabel@cs.toronto.edu

Abstract. Despite their superior accuracy to simpler linear models, kernelized models can be prohibitively expensive in applications where the training set changes frequently, since training them is computationally intensive. We provide bounds for the changes in a kernelized model when its training set has changed, as well as bounds on the prediction of the new hypothetical model on new data. Our bounds support any kernelized model with L_2 regularization and convex, differentiable loss. The bounds can be computed incrementally as the data changes, much faster than re-computing the model. We apply our bounds to three applications: active learning, leave-one-out cross-validation, and online learning in the presence of concept drifts. We demonstrate empirically that the bounds are tight, and that the proposed algorithms can reduce costly model re-computations by up to 10 times, without harming accuracy.

1 Introduction

Supervised machine learning algorithms solve an optimization problem over a given training set, a process called *training*. Kernelized machine learning models are a common choice for handling non-linearity. However, training such models i.e., finding the optimal solution for an optimization problem over a training set, is often slow [31, 22], with run-time complexity being quadratic or cubic in the size of the training set [1].

Such steep run-times can be prohibitive for applications that require frequent retraining. For example, in *leave-one-out cross-validation* (LOOCV) we remove one sample at a time from the original training set, train a model, then test it on the removed sample. In *active learning*, we iteratively improve the training set by selectively choosing samples to label and then retraining the model. Conversely, in *online learning*, new samples arrive one by one, and are added to the training set; the model is recomputed as needed.

Existing approaches to this problem reduce the run-time by using a previously-computed model to speed up the computation of the next model, where the training sets of the models are only slightly different. Examples of such an approach include *incremental algorithms*, which update the model one sample at a time [3, 10, 18], and *warm start* approaches, which use the previous solution when initializing numerical optimization [30]. However, the computational cost of these algorithms is still very expensive reaching approximately $O(m^2)$ where m is the number of *support vectors*.

A more efficient alternative is *sensitivity analysis*, which bounds the change in the trained model or its predictions when the training set is changed, without actually computing the new model. Since the change in the training set is usually smaller than the size of the new training set, incremental bounds are efficient to compute. To the best of out knowledge, existing work on sensitivity analysis is limited to (a) linear models [24, 13] rather than kernelized models, and (b) to non-incremental bounds [29], which require computing over the full old and new training sets, and do not bound the change in prediction.

Our Contributions

We first present a novel bound for the change in any kernelized model as the training set changes, where this model is the solution for a kernelized machine learning optimization problem with convex and differentiable loss and L_2 regularization. Additionally, we derive bounds for the prediction of the updated model on new data. Our bounds support any convex differentiable loss, do not limit the change in the training set, and are applicable for both classification and regression. Finally, we provide an efficient procedure for incrementally evaluating the bounds as the training set changes, and analyze its runtime complexity.

We demonstrate our bounds in three different applications. First, we show that our general bounds can reduce model computations in LOOCV by 45% to 77% for parameter tuning, improving on state-of-the-art bounds for this application. Second, we provide a simple algorithm for active learning that learns faster than the original algorithm using a fine-grained selection of future training data. Third, we present an adaptive online learning algorithm that improves upon incremental learning and concept drift detectors in terms of accuracy and number of model computations by retraining models as needed. We also empirically test the tightness of the bound for model change, showing that for 95% of the test data the bound was no more than 1.2 times the actual change.

2 Background, Problem Definition, and Notations

Assume we have two datasets, $\mathcal{S}_1$ and $\mathcal{S}_2$, where $\mathcal{S}_2$ is more recent, for example $\mathcal{S}_2$ could be an updated version of $\mathcal{S}_1$. The datasets may overlap i.e., some samples in $\mathcal{S}_1$ also appear in $\mathcal{S}_2$. We focus on bounding the distance between the optimal solutions (i.e., models) for two identical learning problems over the two datasets $\mathcal{S}_1$ and $\mathcal{S}_2$. Formally, each dataset is a collection of samples (x_i, y_i), where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$ for classification problems or $y_i \in \mathbb{R}$ for regression problems, and we are interested in bounding the distance between the *kernelized* models trained on $\mathcal{S}_1$ and $\mathcal{S}_2$.

In many classification problems the data is not linearly separable, while in regression problems, the relationship between the predictors and the dependent variables is not always linear. In such cases, a feature map function $\Phi : \mathbb{R}^d \to \mathcal{H}$ can be applied to the samples to transform them from their original feature space to a new feature space $\mathcal{H}$, where the data separation (or relation between variables) is closer to linear [14].

Because $\Phi(x)$ might be infinitely dimensional, the *kernel trick* [27] is commonly used instead of computing and storing $\Phi(x)$ directly. $\beta \in \mathcal{H}$ is a hypothesis in a Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}$ with a positive definite kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ implementing the inner product $\langle \cdot, \cdot \rangle$. The inner product is defined so that it satisfies the reproducing property $\langle k(x, \cdot), \beta \rangle = \beta(x)$. For simplicity, we use the compact notation $\Phi(x)$ for $k(x, \cdot)$. The reproducing property of k implies in particular that $k(x, x') = \langle \Phi(x), \Phi(x') \rangle$.

The models β_1^* and β_2^* are the optimal solutions for the following optimization problems over the samples in the two datasets[1]:

$$\beta_1^* = \arg\min_{\beta \in \mathcal{H}} C_1 \sum_{i \in \mathcal{D}_1} \ell_i(\beta) + \frac{1}{2}\|\beta\|^2 \tag{1a}$$

$$\beta_2^* = \arg\min_{\beta \in \mathcal{H}} C_2 \sum_{i \in \mathcal{D}_2} \ell_i(\beta) + \frac{1}{2}\|\beta\|^2 \tag{1b}$$

for $C_1, C_2 > 0$, where $\mathcal{D}_1$ and $\mathcal{D}_2$ denote the set of indices of the samples in $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. The loss with respect to sample (x_i, y_i) is defined by $\ell_i(\beta) := \ell(y_i, z_i)$, where $z_i := \langle \Phi(x_i), \beta \rangle$ is the inner product between x_i and β. In this paper we focus on loss function $\ell(\cdot, \cdot)$ which is differentiable and convex function with respect to its second argument.

By the Representer theorem [29], if β_1^* is the solution of (1a), it can be expressed as the *dual form*:

$$\beta_1^* = \sum_{i \in \mathcal{D}_1} \alpha_i \Phi(x_i) \tag{2}$$

The coefficients α_i are obtained by solving the *dual problem* of (1a).

We also focus on bounding the prediction of the model β_2^* for a new sample x. The prediction of a model β^* for classification is $\hat{y} = \operatorname{sgn}(\langle \Phi(x), \beta^* \rangle)$, while for regression the prediction is $\hat{y} = \langle \Phi(x), \beta^* \rangle$. For example, predictions for model β_1^* can be computed by using the kernel trick: $\langle \Phi(x), \beta_1^* \rangle = \sum_{i \in \mathcal{D}_1} \alpha_i k(x_i, x)$.

3 Bounding Model Differences

This section details our main contributions. We first develop a *distance bound* – a bound that estimates the difference between the two models β_1^* and β_2^*, without actually computing β_2^*. We then extend it to *prediction bounds* which bound the predictions of β_2^* on new samples. Finally, we describe an incremental update scheme for the bounds. In Section 4 we show how using the bounds can significantly reduce number of model computations in different applications.

[1] Given an objective function of the form $a \sum_{i \in \mathcal{D}} \ell_i(\beta) + b\|\beta\|^2$, choosing $C = \frac{a}{2b}$ will bring it to the standard form (1).

3.1 Bounding the Distance Between Models

Let β_1^* and β_2^* be the models trained on the samples in the datasets $\mathcal{S}_1$ and $\mathcal{S}_2$. We define the difference between the two models as the Euclidean distance between the two model vectors: $\|\beta_1^* - \beta_2^*\|$. We propose a bound for this distance that can be evaluated without computing β_2^*.

Theorem 1 (Distance Bound). *Let β_1^* be the optimal solution in its dual form* (2) *of the optimization problem* (1a) *over the dataset $\mathcal{S}_1$ containing the labeled samples with indices $\mathcal{D}_1$. Let β_2^* be the optimal solution of the optimization problem* (1b) *over the updated dataset $\mathcal{S}_2$ containing the labeled samples with indices $\mathcal{D}_2$. Let $\mathcal{D}_\mathcal{A} = \mathcal{D}_2 \setminus \mathcal{D}_1$ be the set of indices of samples added in $\mathcal{S}_2$ and $\mathcal{D}_\mathcal{R} = \mathcal{D}_1 \setminus \mathcal{D}_2$ be the set of indices of samples removed in $\mathcal{S}_2$. Finally, let I be the indicator function and define $\gamma_i := (-1)^{I_{\{i\in\mathcal{D}_\mathcal{R}\}}} \partial_{z_i}\ell_i(\beta_1^*)$, where $\partial_{z_i}\ell_i(\beta_1^*)$ is the partial derivative of ℓ_i with respect to z_i at the point β_1^*.*

Then the distance between β_1^* and β_2^* is bounded by:

$$\|\beta_1^* - \beta_2^*\| \le 2\|r\|,$$

where

$$r = \sum_{i\in\mathcal{D}_1\cup\mathcal{D}_2} \tau_i \Phi(x_i) \tag{3}$$

and the coefficients τ_i are:

$$\tau_i = \begin{cases} \frac{1}{2}(1-\frac{C_2}{C_1})\alpha_i + \frac{C_2}{2}\gamma_i, & i \in \mathcal{D}_\mathcal{R} \\ \frac{1}{2}(1-\frac{C_2}{C_1})\alpha_i, & i \in \mathcal{D}_1 \cap \mathcal{D}_2 \\ \frac{C_2}{2}\gamma_i, & i \in \mathcal{D}_\mathcal{A} \end{cases} .$$

Discussion Theorem 1 bounds the difference between computed models for any convex differentiable loss, given the difference in their training sets. For example, we can apply Theorem 1 to L_2-regularized logistic regression, as defined in Table 1, $C_1 = C_2 = C$. In this case, $\tau_i = \frac{C}{2}\gamma_i$ for $i \in \mathcal{D}_\mathcal{A} \cup \mathcal{D}_\mathcal{R}$ and 0 otherwise. Assigning this in (3) gives $r = \sum_{i\in\mathcal{D}_\mathcal{A}\cup\mathcal{D}_\mathcal{R}} \frac{C}{2}\gamma_i\Phi(x_i)$. For L_2-regularized MSE loss, the constants C_1 and C_2 depend on the number of samples in the dataset; thus they may differ if the size of the datasets $\mathcal{S}_1$ and $\mathcal{S}_2$ is different. Table 1 shows how to compute r and γ_i for three common optimization problems. Note that α_i are the coefficients in the dual form of β_1^* and C_1, C_2 are determined by the objective function.

Proof. Let Δg be

$$\Delta g := \sum_{i\in\mathcal{D}_\mathcal{A}} \nabla_\beta \ell_i(\beta_1^*) - \sum_{i\in\mathcal{D}_\mathcal{R}} \nabla_\beta \ell_i(\beta_1^*).$$

By applying the chain rule for $\nabla_\beta \ell_i(\beta_1^*)$ we get:

$$\nabla_\beta \ell_i(\beta_1^*) = \partial_{z_i}\ell_i(\beta_1^*)\nabla_\beta z_i(\beta_1^*) = \partial_{z_i}\ell_i(\beta_1^*)\Phi(x_i)$$

Table 1. Objective functions, losses, gradients, and associated bound parameter r. $z_i = \langle \Phi(x_i), \beta \rangle$ can be computed using the kernel trick, and $\gamma_i = (-1)^{I\{i \in \mathcal{D}_\mathcal{R}\}} \partial_{z_i} \ell_i(\beta_1^*)$.

Model and Loss ℓ_i	Objective Function	$\partial_{z_i} \ell_i(\beta)$	r
Logistic Regression $\log(1+\exp(-y_i z_i))$	$C \sum_i \ell_i + \frac{1}{2}\|\beta\|^2$	$-y_i / (1+\exp(y_i z_i))$	$\sum_{i \in \mathcal{D}_\mathcal{A} \cup \mathcal{D}_\mathcal{R}} \frac{C}{2} \gamma_i \Phi(x_i)$
Squared Hinge SVM $(\max\{0, 1-y_i z_i\})^2$	$C \sum_i \ell_i + \frac{1}{2}\|\beta\|^2$	$-2y_i \max\{0, 1-y_i z_i\}$	$\sum_{i \in \mathcal{D}_\mathcal{A} \cup \mathcal{D}_\mathcal{R}} \frac{C}{2} \gamma_i \Phi(x_i)$
Ridge Regression $(y_i - z_i)^2$	$\sum_i \ell_i + \lambda \|\beta\|^2$	$-2(y_i - z_i)$	$\sum_{i \in \mathcal{D}_\mathcal{A} \cup \mathcal{D}_\mathcal{R}} \frac{1}{4\lambda} \gamma_i \Phi(x_i)$

We use this in the definition of Δg:

$$\begin{aligned}\Delta g &\triangleq \sum_{i \in \mathcal{D}_\mathcal{A}} \nabla_\beta \ell_i(\beta_1^*) - \sum_{i \in \mathcal{D}_\mathcal{R}} \nabla_\beta \ell_i(\beta_1^*) \\ &= \sum_{i \in \mathcal{D}_\mathcal{A} \cup \mathcal{D}_\mathcal{R}} \underbrace{(-1)^{I\{i \in \mathcal{D}_\mathcal{R}\}} \partial_{z_i} \ell_i(\beta_1^*)}_{\triangleq \gamma_i} \Phi(x_i) \\ &= \sum_{i \in \mathcal{D}_\mathcal{A} \cup \mathcal{D}_\mathcal{R}} \gamma_i \Phi(x_i), \end{aligned} \tag{4}$$

Note that $\partial_{z_i} \ell_i(\beta)$ is a scalar function with arguments y_i, z_i, which are also scalars. The computation of z_i at the point β_1^* is done with the kernel trick: $z_i = \sum_{j \in \mathcal{D}_1} \alpha_j k(x_j, x_i)$. Thus, γ_i can be computed using the kernel trick.

We can now use this kernelized form of Δg in the non-kernelized form of r from our previous work [28]. In previous work we proved that under the same conditions as in Theorem 1, but for linear models β_1^*, β_2^*

$$r = \frac{1}{2}\left(\beta_1^* - \frac{C_2}{C_1}\beta_1^* + C_2 \Delta g\right) \tag{5}$$

For completeness, we repeat some of the main steps from the original proof. See the Supplemental Material for the full proof. The proof proceeds in three steps:

(i) Use the convexity of the objective function to get a sphere Ω that contains β_2^*. The sphere Ω has center $m = \beta_1^* - r$ and radius vector $r = \frac{1}{2}\left(\beta_1^* + C_2 \sum_{i \in \mathcal{D}_2} \nabla_\beta \ell_i(\beta_1^*)\right)$.

(ii) Use the convexity of the objective function to express the sphere's radius as a function of Δg, and get (5).

(iii) Bound the distance between β_1^* and β_2^* using geometric arguments. We observe that β_1^* is on the surface of the sphere Ω and β_2^* is contained within the sphere. This implies that the maximum distance between β_1^* and β_2^* is obtained when β_1^* and β_2^* are at two opposite sides of the sphere's diameter, which has length $2\|r\|$. This yields the linear form bound in Theorem 1.

Note that this result cannot be applied directly for kernelized models: (5) cannot be evaluated for infinitely dimensional Φ such as the RBF kernel. To complete the proof for kernelized models, we show that the bound could be computed with the kernel trick when map function Φ is used on the data samples. We use the dual form of β_1^* (2) and the kernel form of Δg (4) in r (5):

$$\begin{aligned} r =&\frac{1}{2}\left(\beta_1^* - \frac{C_2}{C_1}\beta_1^* + C_2\Delta g\right) \\ =&\frac{1}{2}\left(1 - \frac{C_2}{C_1}\right)\sum_{i\in\mathcal{D}_1}\alpha_i\Phi(x_i) + \frac{C_2}{2}\sum_{i\in\mathcal{D}_\mathcal{A}\cup\mathcal{D}_\mathcal{R}}\gamma_i\Phi(x_i) = \sum_{i\in\mathcal{D}_1\cup\mathcal{D}_2}\tau_i\Phi(x_i)\ . \end{aligned}$$

The bound is then computed with the kernel trick:

$$2\|r\| = 2\sqrt{\sum_{i\in\mathcal{D}_1\cup\mathcal{D}_2}\sum_{j\in\mathcal{D}_1\cup\mathcal{D}_2}\tau_i\tau_j k(x_i, x_j)}\ , \tag{6}$$

which completes the proof. □

3.2 Bounding the Predictions of the New Model

We describe upper and lower bounds for the prediction of β_2^* for a new sample[2]. As before, using the predictions of β_1^* we can compute these bounds without computing β_2^*.

Using the observation from Section 3.1 that β_2^* is within a sphere Ω with center m and radius vector r, we can obtain lower and upper bounds for applying β_2^* to a new sample x:

Lemma 1 (Prediction Bounds). *Let β_1^*, β_2^*, and r be as in Theorem 1, and let x be a sample. Then the upper and lower bounds on the prediction of β_2^* for x are:*

$$L\left(\langle\Phi(x), \beta_2^*\rangle\right) =\langle\Phi(x), \beta_1^*\rangle - \langle\Phi(x), r\rangle - \|\Phi(x)\|\|r\| \tag{7a}$$

$$U\left(\langle\Phi(x), \beta_2^*\rangle\right) =\langle\Phi(x), \beta_1^*\rangle - \langle\Phi(x), r\rangle + \|\Phi(x)\|\|r\|. \tag{7b}$$

Proof. Every vector β in the sphere Ω can be represented as the sum of two vectors: the center of the sphere m, and an offset vector u that starts from the center of the sphere and whose magnitude is bounded by the sphere radius vector ($\|u\| \leq \|r\|$). Therefore, the dot product between β and a given $\Phi(x)$ is

$$\begin{aligned} \langle\Phi(x), \beta\rangle =&\langle\Phi(x), (m+u)\rangle = \langle\Phi(x), m\rangle + \langle\Phi(x), u\rangle \\ =&\langle\Phi(x), m\rangle + \|\Phi(x)\|\|u\|\cos\left(\angle(\Phi(x), u)\right). \end{aligned}$$

The minimum of the dot product $\langle\Phi(x), \beta\rangle$, with respect to u, is obtained when $\|u\| = \|r\|$ and $\cos\left(\angle(\Phi(x), u)\right) = -1$. In other words, u is a vector in the opposite

[2] Our previous work [28] includes similar bounds for linear models, but only provides a sketch of the proof. Here we provide bounds and the full proof for both kernelized and linear models.

direction of $\Phi(x)$ and with the maximum magnitude under the constraint that u is on the sphere. In this case, the lower bound is obtained, $L\left(\langle \Phi(x), \beta_2^* \rangle\right) = \langle \Phi(x), m \rangle - \|\Phi(x)\| \|r\|$. Using similar arguments, the maximum of the dot product $\langle \Phi(x), \beta \rangle$ is obtained when $\|u\| = \|r\|$ and $\cos\left(\angle(\Phi(x), u)\right) = 1$. This time u is in the same direction as $\Phi(x)$. In this case, the upper bound is obtained, $U\left(\langle \Phi(x), \beta_2^* \rangle\right) = \langle \Phi(x), m \rangle + \|\Phi(x)\| \|r\|$. By substituting $m = \beta_1^* - r$ (from the definition of Ω in Section 3.1) in the above expressions for L and U, we obtain (7). The computation of $\langle \Phi(x), \beta_1^* \rangle$ is done with the kernel trick, $\langle \Phi(x), \beta_1^* \rangle = \sum_{i \in \mathcal{D}_1} \alpha_i k(x_i, x)$. We now use the dual form of r (3) and get

$$\langle \Phi(x), r \rangle = \sum_{i \in \mathcal{D}_1 \cup \mathcal{D}_2} \tau_i k(x_i, x). \tag{8}$$

Finally, using (8), (6), and $\|\Phi(x)\| = \sqrt{k(x,x)}$ we obtain that all the elements in (7) can be computed using the kernel trick. □

3.3 Updating the Bounds Incrementally

Many applications require evaluating the bounds repeatedly (Section 4). We provide incremental update procedures for every sample that is added or removed from the training set, and show that when computed incrementally, the runtime of updating and evaluating our bounds is linear in the number of training examples. This compares favorably to the quadratic to cubic complexity of non-linear SVM solvers [1].

Any algorithm that uses the distance bound (Theorem 1) or the prediction bounds (7) can update the $\|r\|$ part in the bound incrementally with every sample that is added to or removed from either $\mathcal{A}$ or $\mathcal{R}$, where $\mathcal{A} = \mathcal{S}_2 \setminus \mathcal{S}_1$ is the set of samples added in $\mathcal{S}_2$ and $\mathcal{R} = \mathcal{S}_1 \setminus \mathcal{S}_2$ is the set of samples removed from $\mathcal{S}_1$. We limit the discussion of this incremental update and its computational complexity to the common case where $C = C_1 = C_2$, as in the examples in Table 1. In this case:

$$\|r\|^2 = \frac{C^2}{4} \sum_{i \in \mathcal{D}_\mathcal{A} \cup \mathcal{D}_\mathcal{R}} \sum_{j \in \mathcal{D}_\mathcal{A} \cup \mathcal{D}_\mathcal{R}} \gamma_i \gamma_j k(x_i, x_j) \tag{9}$$

$$\langle \Phi(x), r \rangle = \frac{C}{2} \sum_{i \in \mathcal{D}_\mathcal{A} \cup \mathcal{D}_\mathcal{R}} \gamma_i k(x_i, x) \ , \tag{10}$$

since $\tau_i = \frac{C}{2}\gamma_i$ for $i \in \mathcal{D}_\mathcal{A} \cup \mathcal{D}_\mathcal{R}$ and 0 otherwise. For every sample index $i \in \mathcal{D}_\mathcal{A} \cup \mathcal{D}_\mathcal{R}$ let

$$K_i := \frac{C^2}{4} \left(\sum_{\substack{j \in \mathcal{D}_\mathcal{A} \cup \mathcal{D}_\mathcal{R} \\ j \neq i}} 2\gamma_i \gamma_j k(x_i, x_j) + \gamma_i^2 k(x_i, x_i) \right) .$$

To update $\|r\|^2$ with a new sample (x_i, y_i), the sample is first added to $\mathcal{A}$ or $\mathcal{R}$; then this sample's K_i is computed and added to $\|r\|^2$. If the sample is removed,

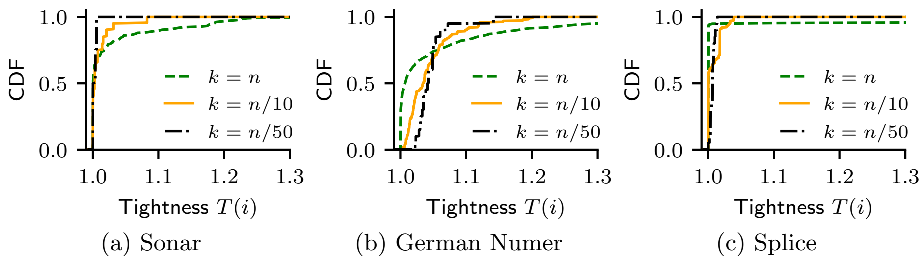


Fig. 1. Tightness of the distance bound. For more than 95% of the iterations, for every k value, the bound is less than 1.2 times the real distance between models.

K_i is first computed and subtracted from $\|r\|^2$. Only then is the sample removed from $\mathcal{A}$ or $\mathcal{R}$.

Computing K_i requires computing γ_i, which has complexity $O(n_1 d)$, where $n_1 = |\mathcal{S}_1|$ is the number of samples in $\mathcal{S}_1$, since a single kernel function evaluation is $O(d)$ and there are at most n_1 support vectors in β_1^* dual form. The complexity of computing $\sum_{j\in\mathcal{D}_\mathcal{A}\cup\mathcal{D}_\mathcal{R}} \gamma_j k(x_i, x_j)$ is $O((|\mathcal{A}|+|\mathcal{R}|)d)$. Given that $|\mathcal{R}| \leq n_1$, we have that the computation complexity of updating $\|r\|$ when a single sample is added or removed from either $\mathcal{A}$ or $\mathcal{R}$ is $O\left(\left(n_1 + |\mathcal{A}|\right) d\right)$. Similarly, given $\|r\|$, the runtime complexity of computing the prediction bounds (7) is $O\left(\left(n_1 + |\mathcal{A}|\right) d\right)$, since it requires evaluating $\langle \Phi(x), \beta_1^* \rangle$ and $\langle \Phi(x), r \rangle$ using (10).

4 Evaluation

In this section, we empirically evaluate the tightness of the distance bound. We also demonstrate the use of the bounds from Section 3 for specific applications by presenting improved variants of common algorithms. For clarity, we purposefully use simple algorithms. For example, in active learning we use only the mid-point of the upper and lower bounds and always select the same number of samples, while the online learning algorithm uses a static threshold T. We leave the detailed exploration and evaluation of improvements for these algorithms to future work. We used L_2 regularized squared hinge-loss (differentiable variant of SVM, Table 1) as the objective function with an RBF kernel $\exp(-\gamma\|x - x'\|^2)$ across all experiments. We used CVXPY [5] to solve the optimization problem.

4.1 Bound Tightness

We empirically evaluate the tightness of the distance bound (Theorem 1) by comparing the bound to the real distance between the models when we remove 1, 10, and 50 samples. Given a dataset of size n, we first train a model β_1^* on the full dataset. We then divide the dataset into k folds, where $k \in \{n, {}^{n}/_{10}, {}^{n}/_{50}\}$. In iteration i we exclude the i^{th} fold from the full training set, compute a model $\beta_2^*(i)$, and evaluate the bound $2\|r_i\|$. The tightness of the bounds for the samples

Table 2. Datasets and percentage of models computed for LOOCV.

Dataset:	Sonar	Breast cancer	Splice	German numer	w5a	EEG eye state	a7a
n	208	569	1000	1000	9888	14980	16100
d	60	30	60	24	300	13	123
Zhang %	55.19	55.50	60.06	57.02	23.26	46.98	40.74
Our %	**47.21**	**48.86**	**55.36**	**46.32**	**22.50**	**40.48**	**31.49**

in fold i, $T(i)$, is defined as the ratio of the bound to the true difference between the models: $T(i) = \frac{2\|r_i\|}{\|\beta_1^* - \beta_2^*(i)\|}$. The closer $T(i)$ is to 1, the tighter the bound.

We repeated this experiment for different datasets using an RBF kernel with $\gamma = 1, C = 1$. Figure 1 shows the CDFs of $T(i)$ for three datasets from the LIBSVM [4] dataset repository: Sonar with $n = 208$, German Numer and Splice with $n = 1000$. For more than 95% of the iterations the bound is less than 1.2 times the real distance between models, indicating that the bound is tight. Results on other datasets in Table 2 are similar, where for large datasets we randomly selected 1000 samples for the test.

4.2 Accelerated LOOCV

Leave-one-out cross-validation (LOOCV) is sometimes used for model selection or to evaluate the generalization error of a model β^* computed from the entire training set of size n. LOOCV works by iterating over the training set: in iteration t, the sample (x_t, y_t) is removed and a new model β_t^* is computed from the remaining $n-1$ samples. The model β_t^* is then used to predict the sample x_t, and these predictions are then aggregated. Although especially useful for small datasets or when low estimation bias is desired [23, 36], LOOCV is computationally expensive since it requires training n models. This is particularly true for non-linear models, which take longer to train than linear models.

We follow the procedure proposed by Okumura et al. [24] for linear models. Rather than computing a new model for each removed sample, we can use the prediction bounds (7) to predict the class of the sample x_t in the LOOCV process. We first compute a model β^* for all samples. Then, for every sample t, rather than computing the model β_t^*, we evaluate the upper and lower bounds (7) based on the original model β^* and the removed sample: $\beta_1^* = \beta^*$, $\beta_2^* = \beta_t^*$, $\mathcal{R} = \{(x_t, y_t)\}$, and $\mathcal{A} = \emptyset$. Note, the overlap between the models β^* and β_t^* is $n-1$ samples. If the signs of the bounds are the same, the classification of β^* for x_t is known and we can avoid computing β_t^*.

We compared our bounds to the bounds of Zhang [35], who proved that $\|\langle \Phi(x_t), \beta_t^* \rangle - \langle \Phi(x_t), \beta^* \rangle\| \leq |\alpha_t| K(x_t, x_t)$, where α_t is the coefficient of x_t in the dual form of β^* (2). From this bound, we obtain upper and lower bounds for the prediction of β_t^*: $\langle \Phi(x_t), \beta^* \rangle - |\alpha_t| K(x_t, x_t) \leq \langle \Phi(x_t), \beta_t^* \rangle \leq \langle \Phi(x_t), \beta^* \rangle + |\alpha_t| K(x_t, x_t)$. Thus, the lower bound for the prediction of β_t^* is $\langle \Phi(x_t), \beta^* \rangle - |\alpha_t| K(x_t, x_t)$ and the upper bound is $\langle \Phi(x_t), \beta^* \rangle + |\alpha_t| K(x_t, x_t)$.

Table 2 shows the results of using the bounds for LOOCV on seven different datasets from the LIBSVM [4] and the UCI dataset repositories [6]. For each dataset, we ran LOOCV five times to tune the γ parameter, selected from the values $\gamma \in \{0.001, 0.01, 0.1, 1, 10\}$. We measured the average percentage of the iterations in which the bounds disagree on the sign, over the five LOOCV runs. This is the percentage of iterations that required computing β_t^*.

We observe that the bounds reduce model computations by two to four times. This translates into significant resource savings. For example, solving the optimization problem for the w5a dataset on an Intel i7-7820HQ CPU running at 2.9 GHz takes 436 seconds. By avoiding model computations in 7663 LOOCV iterations on average in each run, the use of bounds saved over 193 days of compute time compared to standard LOOCV.

Furthermore, our bounds improve on previous work on LOOCV [35] in two ways. Empirically, they train fewer models across all tested datasets. Moreover, our bounds apply even when more than one sample is removed, and can therefore be used for k-fold cross-validation. We leave such improvements for future work.

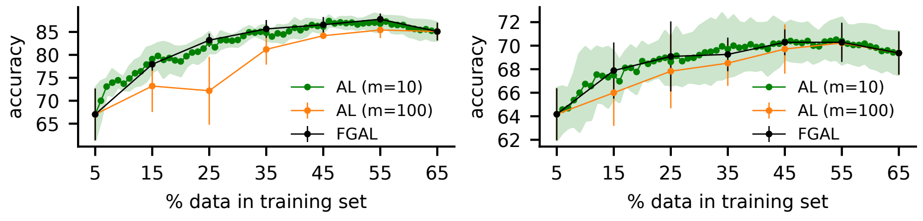
4.3 Fine-Grained Active Learning

In *pool-based active learning*, we are given an unlabeled dataset S of size n. We can ask for the label for any sample in the dataset and must make use of a limited budget for labeling.

The classic active learning algorithm (AL) begins with an initial model β_1^*, often trained on a small initial set of labeled samples. The algorithm iteratively improves the model by choosing more and more samples from the dataset. At each iteration t, the algorithm chooses a cohort of m previously unlabeled samples to be labeled and added to the labeled samples set; then, the model β_{t+1}^* is computed from all the labeled samples. The choice of which samples to add at every iteration is key. If samples are chosen wisely, the model can become accurate using fewer samples. A typical choice is the m samples that the current model is least certain about ($\langle \Phi(x), \beta_t^* \rangle$ closest to 0), since they are more likely to be misclassified.

While a smaller cohort m might be preferable, this comes at a cost of additional iterations, hence more model computations. We now describe FGAL (for Fine Grained Active Learner), a variant of the classic active learning algorithm that uses the prediction bounds to achieve a smaller cohort without the additional cost.

FGAL divides each standard AL iteration into m/p sub-iterations. It starts each iteration t with empty sets of the added and removed samples ($\mathcal{A}$ and $\mathcal{R}$). In every sub-iteration i, FGAL computes the prediction bounds for all the remaining unlabeled samples and chooses the ones that are more likely to be the closest to the separating hyperplane as the next p samples to label. FGAL chooses the samples with the lowest value for $|(U + L)/2|$, where U, L are the prediction bounds (7) of these samples (we leave exploring alternatives for future work). It then adds these samples to $\mathcal{A}$ and the process continues until m samples are chosen. Note, the first m/p samples in each iteration are chosen according



(a) Splice (b) German Numer

Fig. 2. Average model accuracy as a function of the training set size of the model. Each point represents 5 experiments with different seeds; the vertical lines represent the standard deviation. FGAL learns faster, achieving better accuracy with fewer samples.

to the prediction of the model from the last iteration, since $U = L = \langle \Phi(x), \beta_t^* \rangle$ when $\mathcal{A} = \mathcal{R} = \emptyset$. In practice, an efficient implementation can incrementally update $\|r\|$, as explained in Section 3.3, every time p samples are labeled. The pseudo-code for FGAL can be found in the Supplemental Material.

We empirically compared FGAL to standard Active Learning (AL) with same m, and to another AL instance with $m = p$. In our experiments, $m = 100$ and $p = 10$, with an initial labeled set of 50 randomly selected samples. We ran the experiments over two datasets from the LIBSVM dataset repository [4]: Splice with $n = 1000$ samples and $d = 60$ attributes, and the scaled version of German Numer with $n = 1000$ and $d = 24$. We used an RBF kernel with $\gamma = 0.1, C = 1$ for Splice and $\gamma = 0.5, C = 1$ for German Numer. We randomly selected 350 samples as a test set. Each run was repeated 5 times with different random seed values.

Figure 2 shows the accuracy on the test set of all the algorithms after each iteration. We observe that FGAL achieves faster learning than AL($m = 100$) using the same model computation budget. AL must select $m = 100$ samples without considering new information, while FGAL is able to incorporate information from each $p = 10$ samples during the sub-iteration (see Appendix A in the Supplemental Material). Alternatively, we can view FGAL as achieving similar learning performance as AL using a much smaller computational budget: FGAL's performance with $m = 100$ and $p = 10$ on the datasets is equivalent to that of AL with $m = 10$, using only 10% of the model computations. Finally, for FGAL and AL($m = 10$), the accuracy over the test set drops at the final iteration, possibly due to overfitting or label noise. If accuracy on the test set is the criteria, FGAL is able to achieve higher accuracy than AL($m = 100$).

4.4 Adaptive Online Learning

Consider an online learning application where samples are presented to the algorithm one by one: at time t the application is presented with the sample (x_t, y_t). Computing a new model every time a sample is added is wasteful since the underlying concept may have not changed. While incremental algorithms for kernelized models do exist [19, 25, 34], their performance can be sub-optimal due to strong reliance on individual samples and susceptibility to ill-conditioned problems [2, p. 467]. Instead, we propose an algorithm that can reduce computations while maintaining accurate models. The algorithm uses the distance bound to determine when the model has changed and should be re-computed.

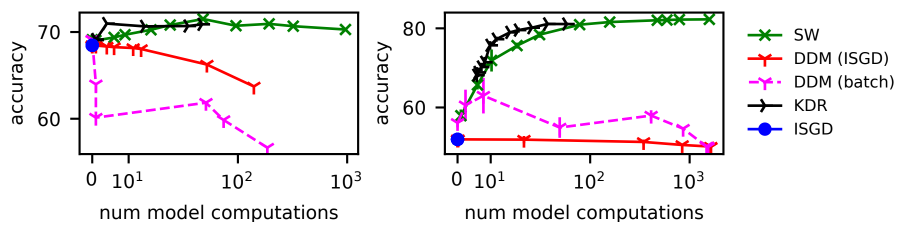
We present KDR (for Kernelized Drift detectoR), a sliding window algorithm that uses the distance bound (Theorem 1) to trade off a small number of batch model computations for better accuracy when learning a classification or regression model over a data stream with concept drifts. When the difference $\|\beta_1^* - \beta_2^*\|$ is too large, where β_1^* is the existing model and β_2^* is the hypothetical updated model, KDR re-computes β_1^*. We describe KDR in detail below. Its pseudo-code is also available in Appendix B of the Supplemental Material.

When a new sample (x_t, y_t) arrives, we incrementally update $\|r\|$ as described in Section 3.3. Note the window size is fixed, hence $C = C_1 = C_2$. We first add the new sample to $\mathcal{A}$ and then increment $\|r\|^2$ by K_t. Let (x_r, y_r) be the oldest sample in W. If the current window W overlaps with W_1 (the window at the time of β_1^* computation), then (x_r, y_r) came from W_1. We therefore first add it to the removed sample set $\mathcal{R}$ and then add K_r to $\|r\|^2$. Otherwise, (x_r, y_r) was never part of W_1, so we first subtract K_r from $\|r\|^2$ and then remove (x_r, y_r) from $\mathcal{A}$ (no change to $\mathcal{R}$). Along with the samples, we also store their γ coefficients in $\mathcal{A}$ and $\mathcal{R}$. Finally, we update the sliding window W by adding (x_t, y_t) and removing (x_r, y_r).

KDR uses β_1^* for predictions and monitors the average $\|r\|$ across the samples received since the last model re-computation. If this value is greater than a user-defined threshold T, we consider it a concept drift and re-compute the model β_1^* from the samples in the current window. For KDR, the overlap of the sliding window with the initial window determines the overlap of the training sets for β_1^* and the hypothetical β_2^*, and ranges from W to 0.

Evaluation We evaluate KDR on one real-world dataset and one synthetic dataset. **SensIT Vehicle** [7] is a real-world time-series collected from wireless distributed sensor networks (WDSN), with labeled classes indicating the vehicle type. We used the version from the OpenML repository [32] with 98,528 samples of 100 features each. We used a window size of 200 samples. To simulate fast concept drifts we "sped up" the data by only using every tenth sample (i.e., only 10% of the data). **Rotating Checkerboard**, proposed by Elwell and Polikar [9], is an artificial 2D time-series with examples sampled uniformly from the unit square and labeled in a 5×5 checkerboard pattern. We generated 9 abrupt concept drifts, where at each concept drift the checkerboard is rotated by an angle of $\pi/20$ radians. The time between drifts is drawn uniformly from 2000 to 8000 samples. The window size is set to 500 samples.

We compare KDR to three algorithms: (i) **SW**, a non-adaptive sliding window algorithm with a fixed period parameter that determines how often batch model re-computation is performed; (ii) **ISGD**, the incremental truncated SGD proposed by Kivinen et al. [19]; and (iii) **DDM** [11], a popular concept drift



(a) SensIT Vehicle (b) Checkerboard 5×5

Fig. 3. The tradeoff between accuracy and number of model computations in SensIT Vehicle and Rotating Checkerboard datasets, for different parameter configurations of each algorithm. KDR achieves a better tradeoff, showing equal or superior accuracy at lower computational cost than the other algorithms, across a range of configurations.

detector that monitors its base learner accuracy and decides when to update the model. We use both ISGD and the SW batch learner as the DDM base learner. All the algorithms have the same window size, which is also the size of the support vector set for ISGD. We use the first 1000 samples from each stream to tune the kernel parameters, learning rate, and regularization parameter C. Finally, we used prequential evaluation [12] to evaluate accuracy.

Since different algorithms have different parameters that control the tradeoff between accuracy and computational cost (i.e., the period for SW, the drift level for DDM, and the threshold T for KDR; there is no such parameter for ISGD), we use *tradeoff curves* to explore the computation-accuracy of the different algorithms. For each configuration of these parameters, we plot a point with the resulting accuracy as the Y coordinate and the number of model computations as the X coordinate. The resulting curve shows how the algorithm behaves as we change its parameters.

Figure 3(a) shows the tradeoff curves on the SensIT Vehicle dataset. Overall, the accuracy of sliding window algorithms that use batch learning (SW and KDR) is superior to that of the incremental learning algorithms (ISGD and DDM). KDR is able to achieve an accuracy equivalent to the non-adaptive SW algorithm using far fewer model computations.

Similarly, Figure 3(b) shows the computation-accuracy tradeoff for the Rotating Checkerboard dataset averaged across five runs with different random seeds (for every seed, all algorithms see the same data); error bars show standard deviation (in practice, it is very small). KDR achieves the same accuracy as SW with less computation since it can adapt to unpredictable concepts drifts. DDM and ISGD are unable to adapt quickly enough, despite substantial tuning.

5 Related Work

Existing work on general sensitivity analysis focuses on linear models without kernels. Okumura et al. [24], Hanada et al. [13] and Sivan et al. [28] all present bounds for the prediction of a linear model when the training set changes, as well as a bound for the Euclidean distance between the previously-computed model and the updated model. Okumura et al. suggest their prediction bounds could be kernelized for non-linear classification problems, but did not provide details. Steinwart and Christmann [29, Corollary 5.12] present a bound for the difference between two kernelized models. Their bound, however, is more limited. Our geometric proof allows us to obtain bounds for the prediction of a new model, which are critical for several applications (detailed below). Our bounds are incremental, and we show how to incrementally update kernel weights in linear time. In addition, the bounds are more general, supporting two optimization problems with different regularization parameters and number of samples.

For specific applications, there are specialized approaches that reduce kernelized model training.

For distributed learning, Kamp et al. [17] present a bound on the distance of a local kernelized model maintained by a local learner to the global average model of all distributed local models. We focus on bounding the differences in a single model when its training set changes.

For LOOCV, Jaakkola and Haussler [15], Joachims [16], Vapnik and Chapelle [33], and Zhang [35] provide upper bounds for the LOOCV error, which can be computed immediately after training the initial kernelized model from the entire training set. Zhang [35] also provides a bound for the distance between the prediction of two models whose training sets differ by exactly one sample. We adapt this bound in Section 4.2 to bound model prediction, and demonstrate empirically that our prediction bounds perform as well or better in LOOCV. Moreover, since our bounds support any change in the training set, they can be used for k-fold cross-validation.

For active learning, the simplest and most commonly used approach is *uncertainty sampling* [20]. In this approach, the model is used to estimate which unlabeled samples are most likely to be misclassified. We show that this approach can be improved using our bounds when more than one sample is chosen to be labeled simultaneously.

For online learning, incremental algorithms are often used to update the model one sample at a time. The challenge of kernelized incremental algorithms lies in limiting the size of the support vector set, which increases linearly with the size of the data stream. Kivinen et al. [19] study classical kernelized stochastic gradient descent (SGD) algorithms. They show that the oldest samples can be removed from the support vector set with only a small impact on the accuracy of the model. Orabona et al. [25] present the Projectron algorithm, which is based on the Perceptron algorithm but requires less memory. While the memory size of the Projectron algorithm is guaranteed to be bounded, it cannot be predicted in advance. Wang and Vucetic [34] propose Passive-Aggressive on a budget that maintains only a fixed number of support vectors, with several versions that trade optimality for runtime efficiency. While incremental model updates are relatively efficient, incremental algorithms can perform poorly on ill-conditioned problems [2, p. 467], and are less immune to outliers than batch learners (since only one sample at a time is used for the model update). Instead, we propose a simple online learning algorithm that uses our bounds to determine when batch computation should occur, similar to concept drift detection.

Another approach is to use *Influence Functions* (IF) to provide a probabilistic approximation for the error of model predictions [21, 29]. Conversely, we focus on providing a general and deterministic bound on both the difference between the models, as well as the prediction. Our work is also related to transfer learning [26]. While transfer learning deals with adapting the existing model to new data, in sensitivity analysis we bound the changes in the model, delaying the computation of the new model. We view our approach as a precursor for transfer learning: it determines when transfer learning should be applied.

6 Conclusions

We presented incremental sensitivity bounds for kernelized machine learning models that evaluate the change in a model and its predictions as the training set changes. Our bounds require only the already computed model and the difference in the training set, and can be evaluated in linear time. We empirically demonstrated the tightness of the bounds, as well as their effectiveness in three different applications: LOOCV, online learning, and active learning.

References

1. Bottou, L., Lin, C.J.: Support vector machine solvers. In: Large Scale Kernel Machines, pp. 301–320 (2007)
2. Boyd, S., Vandenberghe, L.: Convex Optimization (2004)
3. Cauwenberghs, G., Poggio, T.: Incremental and decremental support vector machine learning. pp. 388–394. NIPS '00 (2000)
4. Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology **2**, 27:1–27:27 (2011)
5. Diamond, S., Boyd, S.: CVXPY: A Python-embedded modeling language for convex optimization. JMLR **17**(83), 1–5 (2016)
6. Dua, D., Graff, C.: UCI machine learning repository (2017)
7. Duarte, M.F., Hu, Y.H.: Vehicle classification in distributed sensor networks. J. Parallel Distrib. Comput. **64**(7), 826–838 (Jul 2004)
8. Dunn, J.: Convexity, monotonicity, and gradient processes in Hilbert space. J. Math. Anal. Appl. **53**, 145–158 (01 1976)
9. Elwell, R., Polikar, R.: Incremental learning of concept drift in nonstationary environments. Neural Networks, IEEE Transactions on **22**, 1517 – 1531 (11 2011)
10. Fine, S., Scheinberg, K.: Incremental learning and selective sampling via parametric optimization framework for svm. p. 705–711. NIPS'01 (2001)
11. Gama, J., Medas, P., Castillo, G., Rodrigues, P.: Learning with drift detection. In: Brazilian Symposium on Artificial Intelligence. vol. 8, pp. 286–295 (09 2004)
12. Gama, J., Sebastião, R., Rodrigues, P.P.: On evaluating stream learning algorithms. Mach. Learn. **90**(3), 317–346 (Mar 2013)

13. Hanada, H., Shibagaki, A., Sakuma, J., Takeuchi, I.: Efficiently monitoring small data modification effect for large-scale learning in changing environment. In: AAAI (2018)
14. Hofmann, T., Schölkopf, B., Smola, A.: Kernel methods in machine learning. The Annals of Statistics **36** (01 2007)
15. Jaakkola, T.S., Haussler, D.: Probabilistic kernel regression models. In: In Proceedings of the 1999 Conference on AI and Statistics (1999)
16. Joachims, T.: Estimating the generalization performance of an SVM efficiently. pp. 431–438. ICML '00 (2000)
17. Kamp, M., Bothe, S., Boley, M., Mock, M.: Communication-efficient distributed online learning with kernels. In: Machine Learning and Knowledge Discovery in Databases. pp. 805–819 (2016)
18. Karasuyama, M., Takeuchi, I.: Multiple incremental decremental learning of support vector machines. p. 907–915. NIPS' 09 (2009)
19. Kivinen, J., Smola, A.J., Williamson, R.C.: Online learning with kernels. IEEE Transactions on Signal Processing **52**(8), 2165–2176 (Aug 2004)
20. Lewis, D.D., Gale, W.A.: A sequential algorithm for training text classifiers. In: Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 3–12. SIGIR '94 (1994)
21. Liu, Y., Jiang, S., Liao, S.: Efficient approximation of cross-validation for kernel methods using bouligand influence function. p. I–324–I–332. ICML'14 (2014)
22. Maalouf, M., Trafalis, T.B., Adrianto, I.: Kernel logistic regression using truncated Newton method. Computational Management Science **8**(4), 415–428 (Nov 2011)
23. Molinaro, A.M., Simon, R., Pfeiffer, R.M.: Prediction error estimation: a comparison of resampling methods. Bioinformatics **21**(15), 3301–3307 (05 2005)
24. Okumura, S., Suzuki, Y., Takeuchi, I.: Quick sensitivity analysis for incremental data modification and its application to leave-one-out CV in linear classification problems. pp. 885–894. KDD '15 (2015)
25. Orabona, F., Keshet, J., Caputo, B.: Bounded kernel-based online learning. J. Mach. Learn. Res. **10**, 2643–2666 (Dec 2009)
26. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering **22**(10), 1345–1359 (Oct 2010)
27. Schölkopf, B.: The kernel trick for distances. pp. 283–289. NIPS'00 (2000)
28. Sivan, H., Gabel, M., Schuster, A.: Online linear models for edge computing. In: ECML-PKDD (2019)
29. Steinwart, I., Christmann, A.: Support vector machines. Springer Science & Business Media (2008)
30. Tsai, C.H., Lin, C.Y., Lin, C.J.: Incremental and decremental training for linear classification. pp. 343–352. KDD '14 (2014)
31. Tsang, I.W., Kwok, J.T., Cheung, P.M.: Core vector machines: Fast SVM training on very large data sets. J. Mach. Learn. Res. **6**, 363–392 (Dec 2005)
32. Vanschoren, J., van Rijn, J.N., Bischl, B., Torgo, L.: OpenML: Networked science in machine learning. SIGKDD Explorations **15**(2), 49–60 (2013)
33. Vapnik, V., Chapelle, O.: Bounds on error expectation for support vector machines. Neural Comput. **12**(9), 2013–2036 (Sep 2000)
34. Wang, Z., Vucetic, S.: Online passive-aggressive algorithms on a budget. Proceedings of Machine Learning Research, vol. 9, pp. 908–915 (13–15 May 2010)
35. Zhang, T.: Leave-one-out bounds for kernel methods. Neural Comput. **15**(6), 1397–1437 (Jun 2003)
36. Zhang, Y., Yang, Y.: Cross-validation for selecting a model selection procedure. Journal of Econometrics **187**(1), 95 – 112 (2015)