Technometrics

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/utch20

# Fast and Exact Leave-One-Out Analysis of Large-Margin Classifiers

Boxiang Wang & Hui Zou

**To cite this article:** Boxiang Wang & Hui Zou (2022) Fast and Exact Leave-One-Out Analysis of Large-Margin Classifiers, Technometrics, 64:3, 291-298, DOI: 10.1080/00401706.2021.1967199

**To link to this article:** https://doi.org/10.1080/00401706.2021.1967199

TECHNOMETRICS
2022, VOL. 64, NO. 3, 291–298
https://doi.org/10.1080/00401706.2021.1967199

Taylor & Francis
Taylor & Francis Group



# Fast and Exact Leave-One-Out Analysis of Large-Margin Classifiers

Boxiang Wang[a] and Hui Zou[b]

[a]Department of Statistics, Actuarial Science at the University of Iowa, Iowa City, IA; [b]School of Statistics, the University of Minnesota, Minneapolis, MN

**ABSTRACT**

Motivated by the Golub–Heath–Wahba formula for ridge regression, we first present a new leave-one-out lemma for the kernel support vector machines (SVM) and related large-margin classifiers. We then use the lemma to design a novel and efficient algorithm, named "magicsvm," for training the kernel SVM and related large-margin classifiers and computing the exact leave-one-out cross-validation error. By "magicsvm," the computational cost of leave-one-out analysis is of the same order of fitting a single SVM on the training data. We show that "magicsvm" is much faster than the state-of-the-art SVM solvers based on extensive simulations and benchmark examples. The same idea is also used to boost the computation speed of the *V*-fold cross-validation of the kernel classifiers.

**ARTICLE HISTORY**
Received March 2021
Accepted August 2021

**KEYWORDS**
Cross validation; Kernel learning; Leave-one-out analysis; Support vector machines

## 1. Introduction

Among many existing classification methods, the kernel support vector machine (Cortes and Vapnik SVM, 1995; Vapnik SVM, 1995, 1999) is widely recognized as one of the most competitive classifiers. With extensive numerical studies, Fernández-Delgado et al. (2014) declared that the kernel SVM is one of the best among hundreds of popular classifiers, in the same league as random forest, boosting ensemble, and neural nets. The statistical view of the SVM reveals its connection to nonparametric function estimation in a reproducing kernel Hilbert space (Hastie, Tibshirani, and Friedman RKHS, 2009), which also suggests a unified derivation of many kernel classifiers based on a penalized loss formulation. Let $y = 1, -1$ denote the class label in a binary classification problem. Given a random sample $(y_i, \mathbf{x}_i)_{i=1}^n$, the kernel SVM can be defined as a function estimation problem

$$\min_{f\in\mathcal{H}_K}\left[\frac{1}{n}\sum_{i=1}^{n}\left(1-y_i f(\mathbf{x}_i)\right)_+ + \lambda\|f\|^2_{\mathcal{H}_K}\right], \tag{1}$$

where $(1-u)_+ \equiv \max(1-u, 0)$ is the so-called hinge loss and $f$ is found within an RKHS $\mathcal{H}_K$ with reproducing kernel $K$. The classification rule for $\mathbf{x}$ is $\operatorname{sgn}(f(\mathbf{x}))$. One can replace the hinge loss with other *classification calibrated margin-based loss functions* (Bartlett, Jordan, and McAuliffe 2006) in problem (1), and the resulting classifier is

$$\min_{f\in\mathcal{H}_K}\left[\frac{1}{n}\sum_{i=1}^{n}L\left(y_i f(\mathbf{x}_i)\right) + \lambda\|f\|^2_{\mathcal{H}_K}\right]. \tag{2}$$

Some popular margin-based loss functions in the literature include the logistic regression loss with $L(u) = \log(1+e^{-u})$ and the squared hinge loss with $L(u) = (\max(1-u), 0)^2$, among others. For ease of exposition, we do not include the intercept term throughout the article.

In this article, we consider the leave-one-out analysis of the kernel classifier defined in problem (2). Specifically, we use $\hat{f}_\lambda$ to denote the kernel classifier in problem (2), and use $\hat{f}_\lambda^{[-i]}$ to denote the leave-one-out classifier,

$$\hat{f}_\lambda^{[-i]} = \operatorname{argmin}_{f\in\mathcal{H}_K}\left[\frac{1}{n}\sum_{j\neq i}L\left(y_j f(\mathbf{x}_j)\right) + \lambda\|f\|^2_{\mathcal{H}_K}\right].$$

We repeat the above for $i = 1, 2, \ldots, n$. The leave-one-out analysis is closely tied to jackknife resampling that was proposed for bias and variance estimation of an estimator. Although bootstrap has replaced jackknife in statistical inference (Efron 1982), the leave-one-out analysis is still widely used in assessing the predictive accuracy of a model, that is, the cross-validation method. As early as 1969, it was shown that the leave-one-out cross-validation yields a nearly unbiased estimator of the predictor error (Luntz and Brailovsky 1969). In 1979, Golub, Heath, and Wahba (1979) studied the leave-one-out analysis of ridge regression. Their result directly motivated us to conduct the research in this article, so it is necessary to review their work. The ridge regression is $\hat{f}_\lambda = \operatorname{argmin}_f[\frac{1}{n}\sum_{i=1}^n\left(y_i - f(\mathbf{x}_i)\right)^2 + \lambda\|\boldsymbol{\beta}\|_2^2]$, where $f(\mathbf{x}_i) = \mathbf{x}_i^\top\boldsymbol{\beta}$. The solution is $\hat{f}_\lambda(\mathbf{x}_i) = \mathbf{H}_i^\top\mathbf{y}$, where $\mathbf{H} = \mathbf{X}(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top$, $\mathbf{H}_i$ is the $i$th column of $\mathbf{H}$, and $\mathbf{X}$ is the matrix whose $i$th row is $\mathbf{x}_i$. Consider the same ridge regression with the $i$th observation removed: $\hat{f}_\lambda^{[-i]} = \operatorname{argmin}_f[\frac{1}{n}\sum_{j\neq i}(y_j - f(\mathbf{x}_j))^2 + \lambda\|\boldsymbol{\beta}\|_2^2]$, then the mean-squared leave-one-out cross-validation error is $\frac{1}{n}\sum_{i=1}^n(y_i - \hat{f}_\lambda^{[-i]}(\mathbf{x}_i))^2$, which is denoted by $\widehat{\mathcal{E}}_\lambda^{\mathrm{LOOCV}}$. Golub, Heath, and Wahba (1979) showed the following Golub-Heath-Wahba formula:

**CONTACT** Hui Zou zouxx019@umn.edu School of Statistics, the University of Minnesota, Minneapolis, MN 55455.
Supplementary materials for this article are available online. Please go to www.tandfonline.com/r/TECH.

$$\widehat{\mathcal{E}}_{\lambda}^{\text{LOOCV}} = \frac{1}{n}\sum_{i=1}^{n}\frac{(y_i - \hat{f}_{\lambda}(\mathbf{x}_i))^2}{(1-h_{ii})^2}, \tag{3}$$

where $h_{ii}$ is the $i$th diagonal element of matrix $\mathbf{H}$. It is important to see that Equation (3) is directly from the following leave-one-out residual formula:

$$y_i - \hat{f}_{\lambda}^{[-i]}(\mathbf{x}_i) = \frac{y_i - \hat{f}_{\lambda}(\mathbf{x}_i)}{1-h_{ii}},$$

which can be extended to a family of linear smoothers including smoothing splines and kernel ridge regression (Wahba and Wold 1975; Wahba 1977; Craven and Wahba 1978; Hastie, Tibshirani, and Friedman 2009). In other words, the Golub-Heath-Wahba formula holds for a wide class of nonparametric regression methods as well. Equation (3) was recently used to improve the Mallow's $C_p$ (Rosset and Tibshirani 2020). The leave-one-out analysis was also used as a predictive inference tool in Barber et al. (2021).

The generalized cross-validation criterion was further proposed by using the average of $h_{ii}$ in place of each $h_{ii}$ in the Golub-Heath-Wahba formula, to make the criterion more stable and computationally easier. The generalized cross-validation criterion has been widely used to select the smoothing parameter in generalized additive model; see discussions in Hastie and Tibshirani (1990), Wahba (1990) and implementations in the R packages `gam` (Hastie 2020), `gss` (Gu 2014), and `mgcv` (Wood 2021), among others.

A close inspection of the derivation of the Golub-Heath-Wahba formula reveals that its proof critically depends on two facts: (a) the loss function is the squared error loss, and (b) the regression method is a special linear smoother with the form $\hat{\mathbf{y}} = \mathbf{S}_{\lambda}\mathbf{y}$, where $\mathbf{S}_{\lambda}$ is the *smoother matrix* that only depends on the prechosen parameter $\lambda$ and the predictors, and $\mathbf{S}_{\lambda}$ has a self-stable property (Fan et al. 2020). In fact, the Golub-Heath-Wahba formula does not hold for many powerful regression models such as random forest, gradient boosting and the SVM regression model because they are not self-stable linear smoothers. This is why for a long time the Golub-Heath-Wahba formula is considered as a special property of some linear smoothers in regression (and not even all linear smoothers). In the kernel classifier case, the squared error loss is further replaced with a margin-based loss like the SVM hinge loss. Wahba (1999) proposed a generalized approximate cross-validation (GACV) to estimate the leave-one-out cross-validation error of SVM; however, GACV's derivation is based on approximate Taylor's expansion and a smooth approximation of the hinge loss. Thus, it basically follows the derivation of GACV for regression methods. It remains an open problem how to extend the Golub-Heath-Wahba formula for the kernel classifiers.

In Section 2, we develop a new leave-one-out lemma that can essentially generalize the spirit of the Golub-Heath-Wahba formula to the kernel SVM and related classifiers. In Section 3, we apply the lemma to design a novel algorithm named *magicsvm* for computing the kernel classifier and its $n$ leave-one-out cross-validated variants in the same order of computations as the single kernel classifier on the whole training data. We present numerical examples in Section 4. The technical proofs are relegated to an appendix .

## 2. The Leave-One-Out Lemma

*Lemma 1.* Given a nonnegative convex loss function $L(\cdot)$, consider the corresponding margin-based classifier:

$$\hat{f}_{\lambda} = \text{argmin}_{f\in\mathcal{H}_K}\left[\frac{1}{n}\sum_{i=1}^{n}L\left(y_i f(\mathbf{x}_i)\right) + \lambda\|f\|_{\mathcal{H}_K}^2\right], \tag{4}$$

and its leave-one-out solution $\hat{f}_{\lambda}^{[-i]}$ is

$$\hat{f}_{\lambda}^{[-i]} = \text{argmin}_{f\in\mathcal{H}_K}\left[\frac{1}{n}\sum_{j\neq i}L\left(y_j f(\mathbf{x}_j)\right) + \lambda\|f\|_{\mathcal{H}_K}^2\right].$$

Create a response vector $\tilde{\mathbf{y}}^{[i]}$ by letting $\tilde{y}_i^{[i]} = 0$ and $\tilde{y}_j^{[i]} = y_j$ for all $j \neq i$. Then we have

$$\hat{f}_{\lambda}^{[-i]} = \text{argmin}_{f\in\mathcal{H}_K}\left[\frac{1}{n}\sum_{j=1}^{n}L\left(\tilde{y}_j^{[i]} f(\mathbf{x}_j)\right) + \lambda\|f\|_{\mathcal{H}_K}^2\right].$$

According to Lemma 1, $\hat{f}_{\lambda}^{[-i]}$ can be obtained by using Equation (4) and replacing $y_i$ with zero while keeping all other variables unchanged. Lemma 1 can be easily proven by using the fact that $L(0) = 0$, so the proof details are omitted. In the next section we explain how to exploit this property to design a new algorithm for doing leave-one-out analysis of the kernel classifier.

We further recognize that Lemma 1 can be naturally generalized to the leave-$m$-out validation, $m > 1$. It is worth noting that there is no leave-$m$-out generalization of the Golub-Heath-Wahba formula for ridge regression. So, this can be seen as an advantage of the leave-one-out lemma in this article.

*Lemma 2.* Given a nonnegative convex loss function $L(\cdot)$, consider the corresponding margin-based classifier $\hat{f}_{\lambda}$ that is defined in problem (4). Let $[v]$ denote a subset of the training data and there are $m$ observations. Let $\hat{f}_{\lambda}^{[-v]}$ denote the fitted margin-based classifier after deleting the set $[v]$ from the training set, that is,

$$\hat{f}_{\lambda}^{[-v]} = \text{argmin}_{f\in\mathcal{H}_K}\left[\frac{1}{n-m}\sum_{j\notin[v]}L\left(y_j f(\mathbf{x}_j)\right) + \lambda\|f\|_{\mathcal{H}_K}^2\right]. \tag{5}$$

Create a response vector $\tilde{\mathbf{y}}^{[v]}$ by letting $\tilde{y}_i^{[i]} = 0$ for $i \in v$ and $\tilde{y}_j^{[i]} = y_j$ for all $j \notin [v]$. Then we have

$$\hat{f}_{\lambda}^{[-v]} = \text{argmin}_{f\in\mathcal{H}_K}\left[\frac{1}{n-m}\sum_{j=1}^{n}L\left(\tilde{y}_j^{[v]} f(\mathbf{x}_j)\right) + \lambda\|f\|_{\mathcal{H}_K}^2\right].$$

Lemma 2 can be used for computing $V$-fold cross-validation. The details are given in the next section.

## 3. The Magic SVM Algorithm

### 3.1. Motivation

In this section, we show that the leave-one-out lemma enables us to further boost the computing efficiency of the kernel margin

classifiers including the kernel SVM. We derive a new algorithm for the kernel classifier with a smooth loss function and then extend it to handle the kernel SVM. Thus, our discussion mainly focuses on the SVM. Let $K(\cdot,\cdot)$ be the reproducing kernel function of $\mathcal{H}_K$. The representer theorem of reproducing kernels (Wahba 1990) indicates that the solution to problem (1) is $f(\mathbf{x}) = \sum_{i=1}^n \alpha_i^{\text{SVM}} K(\mathbf{x}_i, \mathbf{x})$ and

$$\boldsymbol{\alpha}^{\text{SVM}} = \text{argmin}_{\boldsymbol{\alpha}\in\mathbb{R}^n} \left[ \frac{1}{n}\sum_{i=1}^n \left(1 - y_i \mathbf{K}_i^\top \boldsymbol{\alpha}\right)_+ + \lambda \boldsymbol{\alpha}^\top \mathbf{K}\boldsymbol{\alpha} \right], \tag{6}$$

where $\mathbf{K}$ is the kernel matrix such that the $(i,j)$th element is $K(\mathbf{x}_i, \mathbf{x}_j)$ and $\mathbf{K}_i$ is the $i$th column of $\mathbf{K}$. The resulting classification rule for $\mathbf{x}_{\text{new}}$ is the sign of $\sum_{i=1}^n \hat{\alpha}_i K(\mathbf{x}_i, \mathbf{x}_{\text{new}})$. We assume the kernel matrix has full rank.

It is interesting to note that the current state-of-the-art algorithms typically solve the dual of problem (6) (Hastie, Tibshirani, and Friedman 2009). The dual problem is solved either by an interior point algorithm (Vanderbei 1999), which is implemented in an R package `kernlab` (Karatzoglou et al. 2004), or by being broken down into a series of smaller problems, namely sequential minimal optimization (Platt 1999; Fan, Chen, and Lin 2005), which is implemented in `libsvm` (Chang and Lin 2011) and interfaced in an R package `e1071` (Meyer et al. 2019).

The "standard" state-of-the-art approaches typically fit the kernel SVM on the training data separately with the leave-one-out variants. As a result, a "standard" approach of the leave-one-out analysis actually applies the same base algorithm $(n+1)$ times, and thus the whole computation time is roughly $(n+1)$ times as large as the time of a single fit. When $n$ is not small (e.g., $n \geq 50$), the "standard" approach is considered to be too expensive to be useful in practice. As a shortcut, people often do $V$-fold cross-validation with $V = 5$ or $V = 10$.

Based on the leave-one-out lemma, we integrate the training and tuning of the kernel SVM such that the whole computation time is of the same order of fitting one classifier on the training set. Therefore, the leave-one-out analysis is not computationally prohibitive to do for the kernel classifiers. Note that the current state-of-the-art algorithms cannot benefit from the leave-one-out lemma because they work in the dual space. To apply the lemma, we develop a new algorithm as explained in the following subsection.

### 3.2. Exact Finite Smoothing Principle for SVM

Finding an efficient algorithm for the kernel SVM is interesting as the kernel SVM is the most representing example among the margin-based classifiers. Computing the SVM directly from problem (6) is typically hard since it is nonsmooth. This is why the current state-of-the-art algorithms solve the dual problem, not problem (6). The leave-one-out lemma is on the primal form of the SVM. After a careful examination of problem (6), we prove that we can compute the exact SVM solution to problem (6) by solving a finite number of smoothed version of problem (6) followed by a projection step.

For any $\delta > 0$, we define a $\delta$-smoothed hinge loss function

$$L_\delta(u) = \begin{cases} 0 & u \geq 1+\delta, \\ \dfrac{1}{4\delta}[u-(1+\delta)]^2 & 1-\delta < u < 1+\delta, \\ 1-u & u \leq 1-\delta. \end{cases}$$

Treat $L_\delta(\cdot)$ as a new margin-based convex loss and the corresponding classifier in RKHS $\mathcal{H}_K$ is

$$\boldsymbol{\alpha}^\delta = \text{argmin}_{\boldsymbol{\alpha}\in\mathbb{R}^n} \left[ \frac{1}{n}\sum_{i=1}^n L_\delta\left(y_i \mathbf{K}_i^\top \boldsymbol{\alpha}\right) + \lambda \boldsymbol{\alpha}^\top \mathbf{K}\boldsymbol{\alpha} \right]. \tag{7}$$

*Proposition 1.* Let $Q(\boldsymbol{\alpha}) = \frac{1}{n}\sum_{i=1}^n L\left(y_i \mathbf{K}_i^\top \boldsymbol{\alpha}\right) + \lambda \boldsymbol{\alpha}^\top \mathbf{K}\boldsymbol{\alpha}$, then we have

$$Q(\boldsymbol{\alpha}^\delta) - \delta/4 \leq Q(\boldsymbol{\alpha}^{\text{SVM}}) \leq Q(\boldsymbol{\alpha}^\delta).$$

It is natural to expect that as $\delta$ approaches zero $L_\delta$ becomes closer and closer to the hinge loss, and consequently $\boldsymbol{\alpha}^\delta$ is approaching to $\boldsymbol{\alpha}^{\text{SVM}}$. Proposition 1 further quantifies the quality of $\boldsymbol{\alpha}^\delta$ as an approximate solution of the SVM in terms of the objective value. When $\delta/(4Q(\boldsymbol{\alpha}^\delta)) < \epsilon$, where $\epsilon$ is a prespecified small constant, say $\epsilon = 10^{-3}$, then the optimal SVM objective value is within the interval $Q(\boldsymbol{\alpha}^\delta)[1-\epsilon, 1]$. From a practical perspective, $Q(\boldsymbol{\alpha})$ reaches the optimal SVM objective at $\boldsymbol{\alpha}^\delta$.

Furthermore, we can do another step to obtain the exact SVM solution from $\boldsymbol{\alpha}^\delta$. For the discussion, we assume there exists some $i$ such that $y_i \mathbf{K}_i^\top \boldsymbol{\alpha}^{\text{SVM}} \neq 1$. The assumption basically says that not all the training points are support vectors. We can easily verify this assumption before doing any serious computation. If $y_i \mathbf{K}_i^\top \boldsymbol{\alpha}^{\text{SVM}} = 1$ for all $i$, then $\boldsymbol{\alpha}^{\text{SVM}}$ must be $\mathbf{K}^{-1}\mathbf{y}$ for a positive definite kernel. By the Karush–Kuhn–Tucker (KKT) conditions of problem (6), one can see this happens if and only if $y_i w_i \in [0, \frac{1}{2n\lambda}]$ for all $i$, where $w_i$ is the $i$th element of $\mathbf{K}^{-1}\mathbf{y}$. By this simple check step, we either know the SVM solution is indeed $\mathbf{K}^{-1}\mathbf{y}$ or the assumption holds. In the latter case, we continue to use the following procedure.

We first need to define several quantities. Define $E_0 = \{i : |1 - y_i \mathbf{K}_i^\top \boldsymbol{\alpha}^{\text{SVM}}| = 0\}$. Let $\delta_0 = \min_{i\notin E_0}\{|1 - y_i \mathbf{K}_i^\top \boldsymbol{\alpha}^{\text{SVM}}|\} > 0$ and $\mathcal{C}_{\delta_0/2}$ be the set $\left\{\boldsymbol{\alpha} : \|\mathbf{K}\boldsymbol{\alpha} - \mathbf{K}\boldsymbol{\alpha}^{\text{SVM}}\|_\infty \geq \delta_0/2\right\}$. Let $Q(\boldsymbol{\alpha})$ be the objective function of problem (6). Define $\eta = \inf_{\boldsymbol{\alpha}\in\mathcal{C}_{\delta_0/2}}[Q(\boldsymbol{\alpha}) - Q(\boldsymbol{\alpha}^{\text{SVM}})] > 0$ and

$$\delta^\star = \min\{\delta_0, 4\eta\}. \tag{8}$$

*Lemma 3.* For any $0 < \delta < \delta^\star$, define $\tilde{E}_0^\delta = \{i : |1 - y_i \mathbf{K}_i^\top \boldsymbol{\alpha}^\delta| < \delta\}$ and $\tilde{\boldsymbol{\alpha}}_0^\delta = \boldsymbol{\alpha}^\delta$. For $k = 1, 2, \ldots,$ let

$$\tilde{\boldsymbol{\alpha}}_k^\delta = \text{argmin}_{\boldsymbol{\alpha}\in\mathbb{R}^n} \left[ \frac{1}{n}\sum_{i=1}^n L_\delta\left(y_i \mathbf{K}_i^\top \boldsymbol{\alpha}\right) + \lambda \boldsymbol{\alpha}^\top \mathbf{K}\boldsymbol{\alpha} \right]$$
$$\text{subject to} \quad 1 = y_i \mathbf{K}_i^\top \boldsymbol{\alpha},\ \forall i \in \tilde{E}_{k-1}^\delta \tag{9}$$

and $\tilde{E}_k^\delta = \{i : |1 - y_i \mathbf{K}_i^\top \tilde{\boldsymbol{\alpha}}_k^\delta| < \delta\}$. Then there exists a finite $k^\star$ such that $\tilde{E}_{k^\star}^\delta = \tilde{E}_{k^\star-1}^\delta \subseteq E_0$ and $\tilde{\boldsymbol{\alpha}}_{k^\star+1}^\delta = \tilde{\boldsymbol{\alpha}}_{k^\star}^\delta$. We denote $\tilde{E}^\delta = \tilde{E}_{k^\star}^\delta$ and $\tilde{\boldsymbol{\alpha}}^\delta = \tilde{\boldsymbol{\alpha}}_{k^\star}^\delta$.

Lemma 4 is that $\tilde{\boldsymbol{\alpha}}^\delta$ will actually equal $\boldsymbol{\alpha}^{\text{SVM}}$ once $\delta$ is less than $\delta^\star$.

*Lemma 4.* Suppose there exists some $i$ such that $y_i \mathbf{K}_i^\top \boldsymbol{\alpha}^{\mathrm{SVM}} \neq 1$. It holds that $\tilde{\boldsymbol{\alpha}}^{\delta} = \boldsymbol{\alpha}^{\mathrm{SVM}}$ as long as $\delta < \delta^\star$, where $\delta^\star$ is given in equation (8).

The threshold in theory may well depend on the training set and is unknown before we have the SVM solution. This issue can be handled by the following computation procedure. We solve problem (7) on a decreasing sequence of $\delta$: $\delta_{(d)}$ with $\delta_{(d+1)} = \tau\delta_{(d)}$ and $0 < \tau < 1$. For example, $\tau = 1/8$ is used in our implementation. After obtaining $\hat{\boldsymbol{\alpha}}^{\delta_{(d)}}$ we solve problem (9) to get $\tilde{\boldsymbol{\alpha}}^{\delta_{(d)}}$ and then check if $\tilde{\boldsymbol{\alpha}}^{\delta_{(d)}}$ satisfies the Karush-Kuhn-Tucker (KKT) condition of the SVM problem (6). If so, then it is the exact SVM solution. If not, we consider the next $\delta$. Lemma 4 guarantees that the iterative process will terminate within a finite number of iterations. In our experiments, the iterative process stops after a few iterations. However, it still remains an open theoretical question to quantify the dependence of the number of iterations on the sample size.

We now develop an efficient algorithm for solving $\hat{\boldsymbol{\alpha}}^{\delta_{(d)}}$, we observe that the first-order derivative of $L_\delta(\cdot)$ is Lipschitz continuous:

$$|L'_\delta(t_1) - L'_\delta(t_2)| \leq |t_1 - t_2|/\kappa, \tag{10}$$

where $\kappa = 2\delta$. For a given $\delta$, we propose to solve problem (7) using the accelerated proximal gradient descent (Parikh and Boyd 2014). Define $\boldsymbol{\alpha}^{(1)}$ to be an initial value. For each $k = 1, 2, \ldots$, the proximal gradient method updates $\boldsymbol{\alpha}^{(k+1)}$ by

$$\boldsymbol{\alpha}^{(k+1)} = \mathrm{argmin}_{\boldsymbol{\alpha}\in\mathbb{R}^n} \left[ \lambda \boldsymbol{\alpha}^\top \mathbf{K}\boldsymbol{\alpha} + \frac{1}{2n\kappa} \left\| \mathbf{K}\boldsymbol{\alpha} - \left( \mathbf{K}\boldsymbol{\alpha}^{(k)} - n\kappa \mathbf{z}^{(k)} \right) \right\|_2^2 \right],$$

where $\mathbf{z}^{(k)}$ is an $n$-vector whose $i$th element is $y_i L'_\delta(y_i \mathbf{K}_i^\top \boldsymbol{\alpha}^{(k)})/n$. It is easy to see that

$$\boldsymbol{\alpha}^{(k+1)} - \boldsymbol{\alpha}^{(k)} = -\mathbf{P}_\lambda^{-1}(\mathbf{K}) \left( \mathbf{K}\mathbf{z}^{(k)} + 2\lambda \mathbf{K}\boldsymbol{\alpha}^{(k)} \right), \quad \text{where}$$
$$\mathbf{P}_\lambda(\mathbf{K}) = 2\lambda \mathbf{K} + \frac{1}{n\kappa}\mathbf{K}\mathbf{K}. \tag{11}$$

For a fixed sample size $n$ and kernel matrix $\mathbf{K}$, the proximal gradient method requires $\mathcal{O}(1/(\epsilon\kappa))$ times of the update (11) to achieve the prescribed precision $\epsilon$ with regard to the objective function.

We can further boost the convergence rate using the Nesterov's acceleration (Nesterov 1983, 2005, 2013; Beck and Teboulle 2009). Construct a sequence, $r_k$, such that $r_1 = 1$ and $r_{k+1} = (1 + \sqrt{1 + 4r_k^2})/2$. Define $\boldsymbol{\alpha}^{(0)}$ and $\boldsymbol{\alpha}^{(1)}$ as initial values. For each $k = 1, 2, \ldots$, we solve $\boldsymbol{\alpha}^{(k+1)}$ as

$$\boldsymbol{\alpha}^{(k+1)} = \mathrm{argmin}_{\boldsymbol{\alpha}\in\mathbb{R}^n} \left[ \lambda \boldsymbol{\alpha}^\top \mathbf{K}\boldsymbol{\alpha} + \frac{1}{2n\kappa} \left\| \mathbf{K}\boldsymbol{\alpha} - \left( \mathbf{K}\bar{\boldsymbol{\alpha}}^{(k)} - n\kappa \bar{\mathbf{z}}^{(k)} \right) \right\|_2^2 \right],$$

where

$$\bar{\boldsymbol{\alpha}}^{(k)} = \boldsymbol{\alpha}^{(k)} + \left( \frac{r_k - 1}{r_{k+1}} \right) \left( \boldsymbol{\alpha}^{(k)} - \boldsymbol{\alpha}^{(k-1)} \right).$$

and the $i$th element of $\bar{\mathbf{z}}^{(k)}$ is $y_i L'_\delta(y_i \mathbf{K}_i^\top \bar{\boldsymbol{\alpha}}^{(k)})/n$. The convergence rate of the Nesterov's accelerated algorithm is $\mathcal{O}((\epsilon\kappa)^{-1/2})$, which is quadratically faster than the algorithm without Nesterov's acceleration. We note the complexity of the update step (11) is $\mathcal{O}(n^2)$. The genuine bottleneck of the algorithm is the inversion of matrix $\mathbf{P}_\lambda(\mathbf{K})$, whose complexity is $\mathcal{O}(n^3)$.

The computation of $\tilde{\boldsymbol{\alpha}}^{\delta_{(d)}}$ can be obtained by using essentially the same proximal gradient descent algorithm in which we use an additional projection step (Parikh and Boyd 2014) to handle the equality constraints during the iteration procedure.

### 3.3. The Integrated Algorithm for Computing SVM and the Leave-One-Out Analysis

We have shown that the exact SVM solution of problem (6) can be obtained by solving a finite sequence of problem (7) with smooth losses. In this section, we shall show that, with the accelerated proximal gradient descent as the base algorithm, Lemma 1 enables us to drastically cut down the whole computation time of fitting SVM on the training set and its leave-one-out variants.

Let us consider the "standard" approach: the leave-$i$-out solution is

$$\hat{\boldsymbol{\alpha}}^{[-i]} = \mathrm{argmin}_{\boldsymbol{\alpha}\in\mathbb{R}^{n-1}} \left[ \frac{1}{n} \sum_{j\neq i} \left( 1 - y_j \boldsymbol{\alpha}^\top \mathbf{K}_j^{[-i]} \right)_+ + \lambda \boldsymbol{\alpha}^\top \mathbf{K}^{[-i]} \boldsymbol{\alpha} \right],$$

for each $i = 1, \ldots, n$, where $\mathbf{K}^{[-i]}$ is the kernel matrix with the $i$th row and column removed and $\mathbf{K}_j^{[-i]}$ is the $j$th column of $\mathbf{K}^{[-i]}$. As $\mathbf{K}^{[-i]}$ differs for each $i$, $\mathbf{P}_\lambda(\mathbf{K}^{[-i]})$ needs to be inverted individually, so each inversion requires $\mathcal{O}(n^3)$ operations.

Based on Lemma 1 we know that $\hat{\boldsymbol{\alpha}}^{[-i]}$ can be obtained via

$$\tilde{\boldsymbol{\alpha}}^{[-i]} = \mathrm{argmin}_{\boldsymbol{\alpha}\in\mathbb{R}^n} \left[ \frac{1}{n} \sum_{j=1}^{n} \left( 1 - \tilde{y}_j^{[i]} \mathbf{K}_j^\top \boldsymbol{\alpha} \right)_+ + \lambda \boldsymbol{\alpha}^\top \mathbf{K}\boldsymbol{\alpha} \right]. \tag{12}$$

Therefore, for each $i$, we construct the response $\tilde{\mathbf{y}}^{[i]}$ as instructed by Lemma 1, and we subsequently use the accelerated proximal gradient descent to solve problem (12). It is critically important to observe that in this process the same kernel matrix $\mathbf{K}$ appears in these slightly different versions of expression (12), namely the same $\mathbf{X}$ and slightly different $\tilde{\mathbf{y}}^{[i]}$. Thus, we only invert $\mathbf{P}_\lambda(\mathbf{K})$ once and store it, which avoids inverting an $(n-1)\times(n-1)$ matrix $n$ times.

Algorithm 1 summarizes the entire integrated algorithm for training and tuning SVM. We can also have a similar and even simpler procedure for other margin-based classifiers such as logistic regression and squared SVM. It is easy to verify that, in the Lipschitz condition (10), $\kappa = 4$ for logistic regression and $\kappa = 1/2$ for squared SVM. We use the same accelerated proximal gradient descent algorithm for computing the solution. See Algorithm 2 for details.

The computation of $V$-fold cross-validation can be likewise reduced by applying the leave-$m$-out formula in Lemma 2. For sake of space, we opt not to repeat the algorithm here. We have implemented the magic SVM algorithm in an R package `magicsvm`. The package handles the leave-one-out analysis as

**Algorithm 1** Magic SVM Procedure

**Require:** $\mathbf{y}$, $\mathbf{K}$, and $\lambda$.

Initialize $\delta$. Define the smooth function $L_\delta$. Let $\kappa = 2\delta$.
Initialize $\tilde{\boldsymbol{\alpha}}^{[-i]} = \mathbf{0}$ for $i = 1, \ldots, n$.
**repeat**
  Compute $\mathbf{P}_\lambda^{-1}(\mathbf{K}) = (2\lambda\mathbf{K} + \frac{1}{n\kappa}\mathbf{K}\mathbf{K})^{-1}$.
  **for** $i = 1, \ldots, n$ **do**
    Let $\tilde{y}_j^{[i]} = y_j$ if $j \neq i$, and $\tilde{y}_i^{[i]} = 0$.
    Initialize $\bar{\boldsymbol{\alpha}} = \tilde{\boldsymbol{\alpha}}^{[-i]}$ and $\boldsymbol{\alpha}' = \tilde{\boldsymbol{\alpha}}^{[-i]}$. Let $r = 1$.
    **repeat**
      Compute $r' = (1 + \sqrt{1 + 4r^2})/2$.
      Compute $\bar{\boldsymbol{\alpha}} = \tilde{\boldsymbol{\alpha}}^{[-i]} + \frac{r-1}{r'}(\tilde{\boldsymbol{\alpha}}^{[-i]} - \boldsymbol{\alpha}')$.
      Compute $\bar{\mathbf{z}} = (\bar{z}_1, \ldots, \bar{z}_n)^\top$, with $\bar{z}_j = \tilde{y}_j^{[i]} L_\delta'(\tilde{y}_j^{[i]} 2 \cdot \mathbf{K}_j^\top \bar{\boldsymbol{\alpha}})/n$.
      Update $\boldsymbol{\alpha}' \leftarrow \tilde{\boldsymbol{\alpha}}^{[-i]}$.
      Update $\tilde{\boldsymbol{\alpha}}^{[-i]} \leftarrow \bar{\boldsymbol{\alpha}} - \mathbf{P}_\lambda^{-1}(\mathbf{K})\left(\mathbf{K}\bar{\mathbf{z}} + 2\lambda\mathbf{K}\bar{\boldsymbol{\alpha}}^{[-i]}\right)$.
      Update $r \leftarrow r'$.
    **until** the convergence condition is met.
    Perform the projection step following Lemma 3.
  **end for**
  Update $\delta \leftarrow \delta/8$, update $L_\delta$, and update $\kappa$.
**until** the KKT conditions of all SVM models are satisfied.

**Algorithm 2** Magic LOOCV for large-margin classifiers with Lipschitz continuous loss functions

**Require:** $\mathbf{y}$, $\mathbf{K}$, $\lambda$, and $\kappa$ (the Lipschitz constant).

Compute $\mathbf{P}_\lambda^{-1}(\mathbf{K}) = (2\lambda\mathbf{K} + \frac{1}{n\kappa}\mathbf{K}\mathbf{K})^{-1}$.
**for** $i = 1, \ldots, n$ **do**
  Let $\tilde{y}_j^{[i]} = y_j$ if $j \neq i$, and $\tilde{y}_i^{[i]} = 0$.
  Initialize $\bar{\boldsymbol{\alpha}} = \tilde{\boldsymbol{\alpha}}^{[-i]}$ and $\boldsymbol{\alpha}' = \tilde{\boldsymbol{\alpha}}^{[-i]}$. Let $r = 1$.
  **repeat**
    Compute $r' = (1 + \sqrt{1 + 4r^2})/2$.
    Compute $\bar{\boldsymbol{\alpha}} = \tilde{\boldsymbol{\alpha}}^{[-i]} + \frac{r-1}{r'}(\tilde{\boldsymbol{\alpha}}^{[-i]} - \boldsymbol{\alpha}')$.
    Compute $\bar{\mathbf{z}} = (\bar{z}_1, \ldots, \bar{z}_n)^\top$, with $\bar{z}_j = \tilde{y}_j^{[i]} L'(\tilde{y}_j^{[i]} \cdot \mathbf{K}_j^\top \bar{\boldsymbol{\alpha}})/n$.
    Update $\boldsymbol{\alpha}' \leftarrow \tilde{\boldsymbol{\alpha}}^{[-i]}$.
    Update $\tilde{\boldsymbol{\alpha}}^{[-i]} \leftarrow \bar{\boldsymbol{\alpha}} - \mathbf{P}_\lambda^{-1}(\mathbf{K})\left(\mathbf{K}\bar{\mathbf{z}} + 2\lambda\mathbf{K}\bar{\boldsymbol{\alpha}}^{[-i]}\right)$.
    Update $r \leftarrow r'$.
  **until** the convergence condition is met.
**end for**

well as $V$-fold cross-validation for the SVM and related margin-based classifiers. The link to the R package is given in the supplementary file to this article.

## 4. Numerical Examples

### 4.1. Efficiency of `magicsvm`

For numerical experiments, we focus on the kernel SVM because of its prominent status among the kernel classifiers. We compare the magic SVM algorithm with `kernlab` and `libsvm`. In all examples, we used the Gaussian kernel.

We consider a simulation example in which data are generated from a mixture Gaussian model that is adopted from Hastie, Tibshirani, and Friedman (2009). Let $\boldsymbol{\mu}_+ = (\mu, \ldots, \mu, 0, \ldots, 0)$, $\boldsymbol{\mu}_- = (0, \ldots, 0, \mu, \ldots, \mu)$, where half of the coordinates are zeros. In each example, the positive class was generated from a mixture Gaussian distribution $\sum_{k=1}^{10} 0.1\mathrm{N}(\boldsymbol{\mu}_{k+}, \sigma\mathbf{I})$ with each $\boldsymbol{\mu}_{k+}$ drawn from $\mathrm{N}(\boldsymbol{\mu}_+, \mathbf{I})$, and likewise the negative class was assembled by $\sum_{k=1}^{10} 0.1\mathrm{N}(\boldsymbol{\mu}_{k-}, \sigma\mathbf{I})$ with each $\boldsymbol{\mu}_{k-}$ from $\mathrm{N}(\boldsymbol{\mu}_-, \mathbf{I})$. We set $\mu = 2$ and $\sigma = 4$. We included 12 examples, with the sample sizes $n$ varying as $\{200, 300, 400\}$ and the dimensions $p = 0.2n$ and $p = 0.5n$.

Table 1 compares the computation time of `magicsvm` with `kernlab` and `libsvm`. We selected $\lambda$ by leave-one-out cross-validation, and then computed the objective values in problem (6). We observe that the objective values of the three packages are exactly the same. We also see that `magicsvm` is consistently faster than the two competitors. For example, when $n = 400$ and $p = 200$, `libsvm` spent more than three hours to complete one time of leave-one-out cross-validation. The same results were obtained by `magicsvm` using about six minutes. The superiority of `magicsvm` is very clear in this example.

We further illustrate the performance of magic LOOCV algorithm for fitting and tuning large-margin classifiers with smooth loss functions. We considered kernel logistic regression and kernel-squared SVM. We used the same simulation data

**Table 1.** Simulated data: comparison of the three R solvers of kernel SVM: `magicsvm`, `kernlab`, and `libsvm`.

| $n$ | $p$ | Computation time (sec) | | | Objective value in (6) | | |
|---|---|---|---|---|---|---|---|
| | | `magicsvm` | `kernlab` | `libsvm` | `magicsvm` | `kernlab` | `libsvm` |
| 200 | 40 | 27.73 | 282.33 | 558.23 | 0.624 (0.007) | 0.624 (0.007) | 0.624 (0.007) |
| | 100 | 27.72 | 429.17 | 1047.60 | 0.561 (0.006) | 0.561 (0.006) | 0.561 (0.005) |
| 300 | 60 | 91.99 | 822.79 | 2003.62 | 0.598 (0.006) | 0.598 (0.006) | 0.598 (0.006) |
| | 150 | 94.20 | 1394.42 | 3958.16 | 0.562 (0.005) | 0.562 (0.005) | 0.562 (0.005) |
| 400 | 80 | 358.25 | 1936.39 | 5262.13 | 0.605 (0.004) | 0.605 (0.004) | 0.605 (0.004) |
| | 200 | 363.02 | 3843.65 | 11559.47 | 0.547 (0.003) | 0.547 (0.003) | 0.547 (0.003) |

NOTES: The run time includes the leave-one-out analysis. The run time and objective values are averaged over 50 independent runs, and the standard errors of the objective values are given in parentheses. Computations were conducted on an Intel Xeon CPU E5-2680 (2.40GHz) processor.

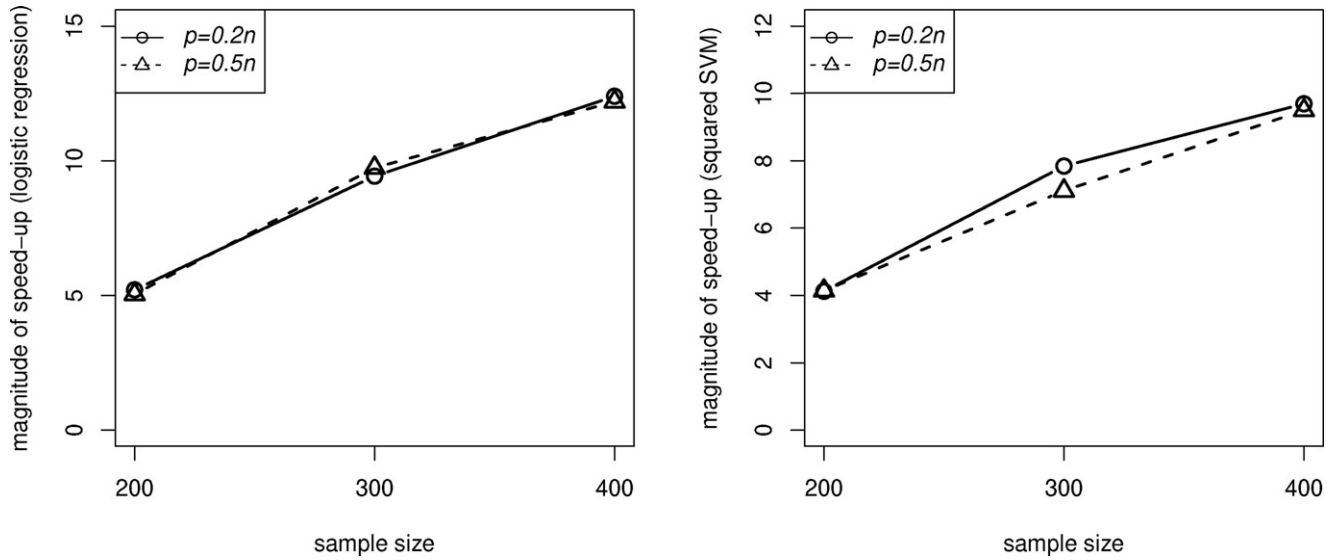


**Figure 1.** Magnitude of speed-up by using the magic LOOCV formula: ratios of the run time of the ordinary LOOCV approach and the run time of the integrated magic LOOCV approach. The left panel is for kernel logistic regression, and the right panel is for kernel squared SVM. The run time includes both fitting and tuning each method. The results are based on 20 independent runs. Computations were conducted on an Intel Xeon CPU E5-2680 (2.40GHz) processor.

**Table 2.** Comparisons of leave-one-out (LOO), 10-fold, 5-fold, and 2-fold CV as means for estimating the prediction error of the kernel SVM.

| *n* | criteria | LOO | 10 | 5 | 2 |
|---|---|---|---|---|---|
| 100 | bias | −0.08 | 0.23 | 1.19 | 4.16 |
| | $\sqrt{\text{var}}$ | 6.06 | 5.94 | 5.98 | 6.66 |
| | RMSE | 6.17 | 6.13 | 6.42 | 8.36 |
| 200 | bias | 0.01 | 0.53 | 1.34 | 4.75 |
| | $\sqrt{\text{var}}$ | 3.56 | 3.52 | 3.64 | 4.61 |
| | RMSE | 3.85 | 4.09 | 4.66 | 7.35 |
| 300 | bias | 0.44 | 1.17 | 1.95 | 5.29 |
| | $\sqrt{\text{var}}$ | 3.30 | 2.99 | 3.05 | 3.88 |
| | RMSE | 3.58 | 3.67 | 4.20 | 7.22 |
| 400 | bias | 0.18 | 0.82 | 1.67 | 5.08 |
| | $\sqrt{\text{var}}$ | 2.70 | 2.75 | 2.89 | 3.28 |
| | RMSE | 3.03 | 3.42 | 4.08 | 6.71 |

NOTES: Training data are simulated from the mixture Gaussian distribution. This table displays the bias, standard deviation ($\sqrt{\text{var}}$), and root mean squared error (RMSE) of the estimates of the generalization error. The numbers are the average quantities over ten different $\lambda$ values and over 50 independent runs.

for Table 1. We compared two approaches, (a) fitting each model using the APG algorithm and employing the ordinary LOOCV approach to select the tuning parameters, and (b) fitting and tuning each method using the integrated magic LOOCV algorithm introduced in Section 3.3. We plotted the ratio of the run time of the two approaches in Figure 1 to visualize the magnitude of speed-up. We observe that the magic LOOCV algorithm dramatically accelerates the ordinary LOOCV approach: for example, when $n = 400$, the magic LOOCV algorithm speeds up the ordinary LOOCV approach about 10 and 12 times faster. The rate of improvement roughly grows linearly with the sample size $n$.

### 4.2. Comparison Among Different V-Fold Cross-Validation

It is a popular claim that the leave-one-out cross-validation has very high variance compared with 5- or 10-fold cross-validation. If we consider the overall bias-variance tradeoff, one should prefer either 5- or 10-fold cross-validation over the leave-one-out. However, in the context of regression, many authors show such a claim is in fact false, for example in Burman (1989) and Zhang and Yang (2015). In the context of kernel learning, the comparison is missing, and it is largely because of the expensive computation brought by the standard leave-one-out analysis. Due to the leave-one-out lemma and magic SVM algorithm presented in this work, it is now feasible and desirable to actually compare the performance of $V$-fold cross-validation for the kernel SVM. We shall demonstrate that leave-one-out cross-validation is better than 10-, 5-, and also 2-fold cross-validation in kernel learning.

**Table 3.** Run time (in second) comparison of leave-one-out and 10-fold cross-validation on seven UCI benchmark data.

| data | *n* | *p* | LOOCV | | | 10-fold CV | | |
|---|---|---|---|---|---|---|---|---|
| | | | `magicsvm` | `kernlab` | `libsvm` | `magicsvm` | `kernlab` | `libsvm` |
| Arrhythmia | 452 | 191 | 48.53 | 866.73 | 2260.78 | 4.40 | 40.51 | 89.55 |
| Australian | 690 | 14 | 129.34 | 586.95 | 1085.17 | 13.60 | 19.19 | 28.64 |
| Hepatitis | 112 | 18 | 0.95 | 35.11 | 35.32 | 0.30 | 8.39 | 6.49 |
| LSVT | 126 | 309 | 1.66 | 146.67 | 359.45 | 0.40 | 26.75 | 55.87 |
| Musk | 476 | 166 | 50.71 | 853.66 | 2314.00 | 4.63 | 38.03 | 87.92 |
| Sonar | 208 | 60 | 5.84 | 107.88 | 192.44 | 0.93 | 12.92 | 17.70 |
| Valley | 606 | 100 | 131.34 | 1047.19 | 2861.66 | 8.44 | 36.76 | 83.72 |

NOTES: All the time are averaged over 50 independent runs. Computations were carried out on an Intel Xeon CPU E5-2680 (2.40GHz) processor.

We used the simulated data from the mixture Gaussian distribution mentioned earlier in this section. In Table 2 we evaluated the bias, variance and mean squared error of each $V$-fold cross-validation error as an estimator of the generalization error of the kernel SVM. It can be seen that the leave-one-out cross-validation is the best. It has the least bias (as expected) and also has similar variance as that of 5- or 10-fold cross-validation. It is interesting to observe that 2-fold cross-validation actually has the largest variance, which contradicts the belief than the variance of cross-validation increases with the number of folds.

### 4.3. Benchmark Data Applications

We further compared `magicsvm` with `kernlab` and `libsvm` on seven benchmark datasets which are available from the UCI machine learning repository (Dua and Graff 2019). The link to each data is provided in supplementary Materials. Those real data examples have various combinations of sample size and dimension. We randomly split each data into a training set and a test set with equal sizes. For sake of exposition, we only presented the computation time because the three packages give identical results except for small difference due to implementation. From the left panel of Table 3, we observe that `magicsvm` is superior over `kernlab` and `libsvm` for the leave-one-out analysis. Taking the dataset arrhythmia as an example, the computation time of `magicsvm` is 48.5 sec, which is more than 15 times faster than `kernlab` and 40 times faster than `libsvm`.

We further used the three R packages to compute 10-fold cross-validation error, and we observe that `magicsvm` is still much faster than the other two competitors. In addition, we notice that the run time of leave-one-out analysis using `magicsvm` is roughly on the same level of using `kernlab` or `libsvm` to compute 10-fold cross-validation. In other word, `magicsvm` enables us to conduct leave-one-out with the same computing resource that originally led to only 10-fold cross-validation.

## 5. Discussions

In this article, we have developed leave-one-out and leave-some-out formula for large-margin classifiers in RKHS, which leads to the design of a new, exact, and much faster algorithm `magicsvm` for training and tuning the kernel SVM and related classifiers. We have also shown that the LOOCV error, as an estimator of the generalization error of the SVM with a fixed regularization parameter, works as well as the 10-fold error or 5-fold CV error. Therefore, with `magicsvm` the computation time should not be a factor preventing us from using leave-one-out cross-validation to estimate the generalization error and select the regularization parameter of the kernel classifier. The numerical experiments have clearly demonstrated the advantage of our new algorithm.

## Supplementary Material

This file contains all the technical proofs and links to the R packages and benchmark data.

## Acknowledgments

We thank to the editor, AE and two referees for their helpful comments and suggestions.

## Funding

Zou's work is supported in part by NSF (grant nos. 1915-842 and 2015-120).

## References

Barber, R. F., Candes, E. J., Ramdas, A., and Tibshirani, R. J. (2021), "Predictive Inference With the Jackknife+," *Annals of Statistics*, 49, 486–507. [292]

Bartlett, P. L., Jordan, M. I., and McAuliffe, J. D. (2006), "Convexity, Classification, and Risk Bounds," *Journal of the American Statistical Association*, 101, 138–156. [291]

Beck, A., and Teboulle, M. (2009), "A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems," *SIAM Journal on Imaging Sciences*, 2, 183–202. [294]

Burman, P. (1989), "A Comparative Study of Ordinary Cross-Validation, $V$-Fold Cross-Validation and the Repeated Learning-Testing Methods," *Biometrika*, 76, 503–514. [296]

Chang, C.-C., and Lin, C.-J. (2011), "LIBSVM: A Library for Support Vector Machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2, 1–27. [293]

Cortes, C., and Vapnik, V. (1995), "Support-Vector Networks," *Machine Learning*, 20, 273–297. [291]

Craven, P., and Wahba, G. (1978), "Smoothing Noisy Data With Spline Functions," *Numerische Mathematik*, 31, 377–403. [292]

Dua, D., and Graff, C. (2019), "UCI Machine Learning Repository," available at *http://archive.ics.uci.edu/ml*. [297]

Efron, B. (1982), *The Jackknife, the Bootstrap and Other Resampling Plans*, Philadelphia, PA: SIAM. [291]

Fan, J., Li, R., Zhang, C.-H., and Zou, H. (2020), *Statistical Foundations of Data Science*, Boca Raton, FL: Chapman and Hall/CRC. [292]

Fan, R.-E., Chen, P.-H., and Lin, C.-J. (2005), "Working Set Selection Using Second Order Information for Training Support Vector Machines," *Journal of Machine Learning Research*, 6, 1889–1918. [293]

Fernández-Delgado, M., Cernadas, E., Barro, S., and Amorim, D. (2014), "Do We Need Hundreds of Classifiers to Solve Real World Classification Problems?" *Journal of Machine Learning Research*, 15, 3133–3181. [291]

Golub, G. H., Heath, M., and Wahba, G. (1979), "Generalized Cross-Validation as a Method for Choosing a Good Ridge Parameter," *Technometrics*, 21, 215–223. [291]

Gu, C. (2014), "Smoothing Spline ANOVA Models: R Package GSS," *Journal of Statistical Software*, 58, 1–25. [292]

Hastie, T. (2020), *gam: Generalized Additive Models*, R package version 1.20. [292]

Hastie, T., Tibshirani, R., and Friedman, J. (2009), *The Elements of Statistical Learning: Prediction, Inference and Data Mining*, New York: Springer-Verlag. [291,292,293,295]

Hastie, T. J., and Tibshirani, R. J. (1990), *Generalized Additive Models*, Boca Raton, FL: CRC Press. [292]

Karatzoglou, A., Smola, A., Hornik, K., and Zeileis, A. (2004), "kernlab-an S4 Package for Kernel Methods in R," *Journal of Statistical Software*, 11, 1–20. [293]

Luntz, A., and Brailovsky, V. (1969), "On Estimation of Characters Obtained in Statistical Procedure of Recognition," *Technicheskaya Kibernetica* (in Russian). [291]

Meyer, D., Dimitriado, E., Hornik, K., Weingessel, A., and Leisch, F. (2019), *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. R package version 1.7-3. [293]

Nesterov, Y. (2005), "Smooth Minimization of Non-Smooth Functions," *Mathematical Programming*, 103, 127–152. [294]

——— (2013), "Gradient Methods for Minimizing Composite Functions," *Mathematical Programming*, 140, 125–161. [294]

Nesterov, Y. E. (1983), "A Method for Solving the Convex Programming Problem With Convergence Rate $O(1/k^2)$," in *Dokl. akad. nauk Sssr*, Vol. 269. [294]

Parikh, N., and Boyd, S. (2014), "Proximal Algorithms," *Foundations and Trends in Optimization*, 1, 127–239. [294]

Platt, J. C. (1999), "Using Analytic QP and Sparseness to Speed Training of Support Vector Machines," in *Advances in Neural Information Processing Systems*, eds. M. Kearns, S. Solla, and D. Cohn, Cambridge, MA: The MIT Press. [293]

Rosset, S., and Tibshirani, R. J. (2020), "From Fixed-$X$ to Random-$X$ Regression: Bias-Variance Decompositions, Covariance Penalties, and Prediction Error Estimation," *Journal of the American Statistical Association*, 115, 138–151. [292]

Vanderbei, R. J. (1999), "LOQO: An Interior Point Code for Quadratic Programming," *Optimization Methods and Software*, 11, 451–484. [293]

Vapnik, V. (1995), *The Nature of Statistical Learning Theory*. Springer Science & Business Media. New York: Springer [291]

——— (1999), "An Overview of Statistical Learning Theory," *IEEE Transactions on Neural Networks*, 988–999. [291]

Wahba, G. (1977), "Practical Approximate Solutions to Linear Operator Equations When the Data Are Noisy," *SIAM Journal on Numerical Analysis*, 14, 651–667. [292]

——— (1990), *Spline Models for Observational Data*, Vol. 59. Philadelphia, PA: Society for Industrial and Applied Mathematics. [292,293]

——— (1999), "Support Vector Machines, Reproducing Kernel Hilbert Spaces and the Randomized GACV," *Advances in Kernel Methods-Support Vector Learning*, 6, 69–87. [292]

Wahba, G., and Wold, S. (1975), "Periodic Splines for Spectral Density Estimation: The Use of Cross Validation for Determining the Degree of Smoothing," *Communications in Statistics–Theory and Methods*, 4, 125–141. [292]

Wood, S. (2021), *mgcv: Mixed GAM Computation Vehicle with Automatic Smoothness Estimation* (R package version 1.8-36). [292]

Zhang, Y., and Yang, Y. (2015), "Cross-Validation for Selecting a Model Selection Procedure," *Journal of Econometrics*, 187, 95–112. [296]