Discrete Optimization 22 (2016) 183–194

Contents lists available at ScienceDirect

Discrete Optimization

www.elsevier.com/locate/disopt

Fast training of Support Vector Machines with Gaussian kernel

Matteo Fischetti

DEI, University of Padova, via Gradenigo 6/A, 35100 Padova, Italy

ARTICLE INFO

Article history:
Available online 1 April 2015

Keywords:
Support vector machine
Classification
Mixed-integer programming

ABSTRACT

Support Vector Machines (SVM's) are ubiquitous and attracted a huge interest in the last years. Their training involves the definition of a suitable optimization model with two main features: (1) its optimal solution estimates the a posteriori optimal SVM parameters in a reliable way and (2) it can be solved efficiently. Hinge-loss models, among others, have been used with remarkable success together with cross validation—the latter being instrumental to the success of the overall training, though it can become very time consuming. In this paper we propose a different model for SVM training, that seems particularly suited when the Gaussian kernel is adopted (as it is often the case). Our approach is to model the overall training problem as a whole, thus avoiding the need of cross validation. Though our basic model is an NP-hard Mixed-Integer Linear Program, some variants can be solved very efficiently by simple sorting algorithms. Computational results on test cases from the literature are presented, showing that our training method can lead to a classification accuracy comparable (or even slightly better) than the classical hinge-loss model, with a speedup of 2–3 orders of magnitude.

1. Introduction

Data classification is a very important task in machine learning. Supervised learning for data classification tries to infer a function from a given set of training examples. A supervised learning algorithm then analyzes the training data and produces a function to be used for the (as accurate as possible) classification of new examples.

In this paper we address Vapnik's [1] *Support Vector Machine* (SVM), and in particular its training when the so-called Gaussian kernel is adopted [2]. We assume that the reader has some familiarity with the basic ingredients of classification methods; see, e.g., Carrizosa and Romero Morales [3] for a recent review on mathematical optimization for supervised classification, and the book of Hastie, Tibshirani, and Friedman [4] for a more general treatment of statistical learning.

SVM training is typically done by solving an optimization problem whose objective function gives a tradeoff between classification margin (i.e., norm of the training parameters) and misclassification over the

E-mail address: matteo.fischetti@unipd.it.

http://dx.doi.org/10.1016/j.disopt.2015.03.002
1572-5286/© 2015 Elsevier B.V. All rights reserved.

training set. By fixing some parameters – notably, the tradeoff weight in the objective function and the constant(s) appearing in the kernel definition – the problem becomes convex and can therefore be solved efficiently. The fixed parameters can instead be tuned through an outer-loop optimization where candidate values are tested through cross validation over the training set. The overall approach is quite time consuming, as a series of convex optimization problems need to be solved. E.g., if the outer loop has just 2 parameters, each of which has 5 alternative values, and 5-fold cross validation is used, then $5 * 5 * 5 + 1 = 126$ convex problems need be solved for a single SVM training.

In the present paper we propose a new model that avoids (or, at least, greatly reduces) the need on the outer-loop optimization, by incorporating cross validation into the model. To be more specific, we propose a new Mixed-Integer Linear Programming (MILP) model that aims at minimizing the so-called *leave-one-out* estimate of misclassification probability—leave-one-out being the name of the most time-consuming k-fold validation arising where k is equal to the number of training points. Though NP-hard, we show that this model can heuristically be solved in a practically satisfactory way through a clever approach that applies a general-purpose MILP refining tool to an ad hoc warm-start solution. Even more importantly for practical applications, we show how the model can be simplified to be solvable very efficiently while retaining (or even improving) its accuracy on the test set.

In our view, the main contributions of our work are as follows:

1. We use a new interpretation of SVM's with Gaussian kernels in terms of telecommunication systems.
2. We present a new MILP model that uses a leave-one-out objective function to train the SVM, without the need of a time consuming outer-loop optimization through cross validation.
3. We investigate the overfitting phenomenon associated with our model, and introduce a new way to contrast it: instead of introducing quadratic penalty terms in the objective, we introduce a priori constraints in the model that limit the degree of freedom of the solutions and hence overfitting.
4. We present very fast algorithms for solving the constrained versions of our model, and computationally show that they allow for an SVM accuracy (on the test set) at least as good as a standard hinge-loss model, but run orders of magnitude faster.

The present paper is organized as follows. The basic SVM approach is outlined in Section 2, together with some basic optimization models used for its training. Our leave-one-out MILP model is introduced and discussed in Section 3, where we also illustrate the overfitting phenomenon on a sample case. Computational results on test cases from the literature are presented in Section 5. Conclusions and possible directions of research are finally illustrated in Section 6.

2. SVM training

In supervised classification, we are given a *training set* consisting of p pairs $(x_1, y_1), \cdots, (x_p, y_p)$ with $x_i \in \mathbb{R}^n$ and $y_i \in \{-1, +1\}$. The n components of each point x_i are called *features* and represent some known attributes of the point, whereas the corresponding y_i gives the known *classification* of the point as belonging to one of two given classes encoded by $+1$ and -1, respectively.

Given a point $x \in \mathbb{R}^n$ not in the training set with an (existing but) unknown classification $y_x \in \{-1, +1\}$, we want to estimate y_x with some degree of accuracy. An SVM is a function that estimates y_x by computing

$$y(x) := sign\left(\sum_{i=1}^{p} \alpha_i y_i K(x, x_i) + \beta_0\right) \tag{1}$$

where $K(x, x_i)$ is a given scalar *kernel* function, and $\alpha_1, \ldots, \alpha_p \geq 0$ and β_0 (the latter unconstrained in sign) are parameters of the SVM that do not depend on x but can preliminary be tuned by using the points in the training set.

The kernel function $K(x, x_i)$ is intended to measure the "similarity" between x and x_i (the larger the more similar). The two most widely-used such functions are

- linear kernel: $K(x, x_i) :=< x, x_i >$ (inner product between x and x_i)
- Gaussian kernel: $K(x, x_i) := e^{-\gamma\|x-x_i\|^2}$ (for a given parameter $\gamma > 0$)

The linear kernel gives a signed measure of the similarity between x and x_i, in the sense that the angle between the two points plays a role in determining how similar they are, and can lead to negative values of K. On the contrary, the Gaussian kernel only depends on the Euclidean distance between x and x_i, and is based on the assumption that similar points are close one to each other in the feature space (in terms of Euclidean distance). This latter assumption is very reasonable in many cases, hence the Gaussian kernel is often used in practice.

In what follows we will make use of the following "telecommunication" interpretation of (1), that we proposed together with Lucia Petterle in her master's thesis [5]. Each point x_i in the training set broadcasts its value y_i with an amplification level α_i, and the transmitted signal decay in the feature space with an exponential law $e^{-\gamma d^2}$ depending on the Euclidean distance d and on the power-decay parameter γ. A receiver sitting at the given measure point x computes the total signal received, namely $\sum_{i=1}^{p} \alpha_i y_i e^{-\gamma\|x-x_i\|^2}$, compares it with a threshold $-\beta_0$, and decides whether the signal in x is likely to be $+1$ or -1 according to the sign of the result.

The situation is illustrated in Fig. 1 for the classification of points in the plane that belong to a given spiralis. The $+1$ training points belong to the spiralis, whereas the -1 points are random samples on the plane. If all points x_j transmit with the same amplification level $\alpha_j = 1$, a careful choice of the power-decay parameter γ and of the threshold $-\beta_0$ defines a "narrow tube" around the spiralis where the received signal exceeds the threshold, thus allowing for a reliable classification of the points in the plane.

As already mentioned, the SVM parameters $\alpha = (\alpha_1, \ldots, \alpha_p)$ and β_0 (and γ in case of Gaussian kernel) have to be determined in a preliminary training phase. To this end, the SVM parameters are viewed as variables of a suitable mathematical model based on the training set data, and the optimal solution of the model determines the actual SVM parameters to be used to classify new points. The choice of the mathematical model is of course crucial, as being "too clever" on the training set almost invariably leads to an overfitting phenomenon. A widely-used model is the following convex quadratic problem:

$$\text{(HINGE)} \quad \min_{\alpha,\beta_0,\xi} \frac{1}{2}\sum_{i=1}^{p}\sum_{j=1}^{p} \alpha_i\alpha_j y_i y_j K(x_i, x_j) + C\sum_{j=1}^{p} \xi_j \tag{2}$$

$$y_j\left(\sum_{i=1}^{p} \alpha_i y_i K(x_j, x_i) + \beta_0\right) \geq 1 - \xi_j \quad \forall\, j = 1, \ldots, p \tag{3}$$

$$\alpha_j \geq 0, \quad \forall\, j = 1, \ldots, p \tag{4}$$

$$\xi_j \geq 0, \quad \forall\, j = 1, \ldots, p \tag{5}$$

where $C > 0$ is a parameter to be tuned (together with γ in case of Gaussian kernel) through an external cross validation procedure; see [4,3] for details.

3. Leave-one-out training models

At first glance, one would expect that the best choice for α and β_0 (and γ for Gaussian kernel) would be the one that minimizes the average number of misclassified points in the training set, computed by solving

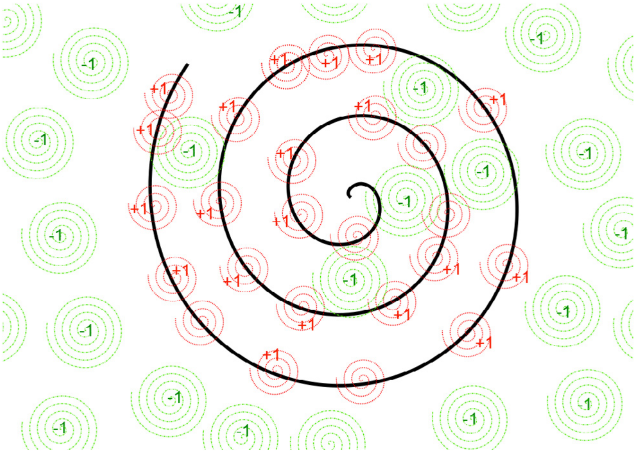

Fig. 1. Telecommunication interpretation of SVM with Gaussian kernel: the region where the received signal exceeds a sufficiently-large positive threshold defines a "narrow tube" around the spiralis.

the following MILP:

$$\text{(NAIVE)} \quad \min_{\alpha,\beta_0,z} \frac{1}{p}\sum_{j=1}^{p} z_j \tag{6}$$

$$y_j\left(\sum_{i=1}^{p} \alpha_i y_i K(x_j,x_i)+\beta_0\right) \geq \epsilon - M z_j \quad \forall\, j=1,\ldots,p \tag{7}$$

$$0 \leq \alpha_i \leq 1, \quad \forall\, i=1,\ldots,p \tag{8}$$

$$z_i \in \{0,1\}, \quad \forall\, i=1,\ldots,p \tag{9}$$

where $\epsilon > 0$ is a very small tolerance value used to make the null solution $(\alpha,\beta_0,z)=(0,0,0)$ infeasible, M is a very large "big-M" positive value, and $z_j = 1$ iff x_j is misclassified, i.e., iff y_j and $\sum_{i=1}^{p}\alpha_i y_i K(x_j,x_i)+\beta_0$ do not have the same sign. The normalization condition $\alpha_i \leq 1$ is also imposed for all i.

Training model NAIVE turns out to be very unsatisfactory in practice, as it tends to overfit the training points and typically leads to a high misclassification probability for the points x not belonging to the training set. In other words, the NAIVE objective function can be a very poor approximation of the real misclassification probability of the resulting SVM when applied to points x not in the training set.

In our view, a main modeling error is that NAIVE includes the contribution of x_j in its summation (7). By taking our telecommunication interpretation, this means that the model assumes that a transmitter is always present in the same place where we want to make a signal measure, which is of course unrealistic. For example, for a Gaussian kernel the NAIVE model would tend to choose a very large value for γ (so as to have an "almost impulsive" kernel with $K(x,x_j)\approx 1$ if $x=x_j$, $K(x,x_j)\approx 0$ otherwise), as setting $\alpha_1=\cdots=\alpha_p=1$ and $\beta_0=-0.5$ gives no misclassification at all in the training set—but this choice is completely useless as no significant signal is received outside the training set.

To avoid the above drawback, we propose a modified *leave-one-out (loo)* model where each point x_j is "left out" as a transmitter when the total signal seen at x_j is measured, namely:

$$\text{(LOO\_MILP)} \quad \min_{\alpha,\beta_0,z} \frac{1}{p}\sum_{j=1}^{p} z_j \tag{10}$$

$$y_j\left(\sum_{i:i\neq j} \alpha_i y_i K(x_j,x_i)+\beta_0\right) \geq \epsilon - M z_j \quad \forall\, j=1,\ldots,p \tag{11}$$

$$0 \leq \alpha_i \leq 1, \quad \forall\, i = 1, \ldots, p \tag{12}$$

$$z_i \in \{0, 1\}, \quad \forall\, i = 1, \ldots, p \tag{13}$$

where the quantity

$$\nu_j(\alpha) := \sum_{i:i \neq j} \alpha_i y_i K(x_j, x_i) \tag{14}$$

appearing in (11) will be called *net signal* measured at x_j, and does not take the signal transmitted by x_j into account as the *total signal* $\sum_{i=1}^{p} \alpha_i y_i K(x_j, x_i)$ in (7) does.

The "leave-one-out" name needs some clarifications. In statistical theory, term *leave-one-out* refers to a cross validation procedure that works as follows. Suppose to use a deterministic training method $\mathcal{M}(T)$ to define the optimal SVM parameters on a set T of training points. Let ALL $:= \{1, \ldots, p\}$ correspond to the overall training set. For each $j = 1, \ldots, p$: (i) temporarily remove x_j from the training set; (ii) train the SVM by applying $\mathcal{M}(\text{ALL} \setminus \{j\})$ and (iii) define $z_j = 1$ if x_j is misclassified by the trained SVM, $z_j = 0$ otherwise. At the end, the quantity $\sum_{j=1}^{p} z_j/p$ is the *leave-one-out estimate* of the probability that an SVM trained through $\mathcal{M}(\text{ALL})$ will misclassify a new point $x \in \mathbb{R}^n$. It is known that this estimate is quite reliable (both in theory and in practice), provided that the training method $\mathcal{M}(T)$ is completely "blind" with respect to the points not in T; see e.g. [4]. However, the leave-one-out method can be very time consuming, hence it is seldom used in practice—k-fold cross validation with $k = 5, 10$ being typically preferred for the choice of some model parameters such as C and γ.

In our LOO_MILP model (10)–(13), the objective function – because of (11) – plays the role of the leave-one-out estimate of the misclassification probability. However, the model (as stated) involves *all* training points for the optimal determination of the SVM parameters α_i's and β_0, so the blindness hypothesis fails and the method is still prone to overfitting. To contrast this drawback, we propose to limit a priori the valid choices for the SVM parameters, e.g., by imposing exactly one of the following additional sets of constraints:

$$\alpha_1 = \alpha_2 = \cdots = \alpha_p = 1 \quad \text{and} \quad \beta_0 = 0 \tag{15}$$

$$\alpha_1 = \alpha_2 = \cdots = \alpha_p = 1, \qquad \beta_0 \quad \text{free} \tag{16}$$

$$\alpha_i = \begin{cases} \alpha^+ \geq 0, & \forall\, i : y_i = +1 \\ \alpha^- \geq 0, & \forall\, i : y_i = -1 \end{cases}, \qquad \beta_0 \quad \text{free.} \tag{17}$$

Model LOO_1, i.e., model LOO_MILP amended by restriction (15), represents our extreme case where all SVM parameters α_i's and β_0 are fixed, so there is no degree of freedom and no optimization at all is needed—just set $z_j = 0$ if $y_j(\sum_{i \neq j} y_i K(x_j, x_i)) \geq \epsilon$, and $z_j = 1$ otherwise. In this case, the objective function models leave-one-out accuracy in an exact way, as no overtuning is (of course) incurred. Although apparently trivial, it was shown in Petterle [5] that the resulting SVM can perform reasonably well, mainly when the Gaussian kernel is considered. The explanation is that the Gaussian kernel itself often acts as a good classifier, even without any tuning of α and β_0—whose "too clever" determination can actually be counterproductive because of overfitting.

Model LOO_2, i.e., model LOO_MILP plus (16), represents a compromise where all α_i's are fixed a priori, and the training model has only one degree of freedom as it can decide the remaining variable β_0. Our intuition (confirmed by the computational tests) is that optimizing β_0 can significantly improve the optimal value of the objective function, that on the other hand remains a reliable approximation of the true leave-one-out estimate of the misclassification probability.

Model LOO_3, i.e., model LOO_MILP plus (17), has more degrees of freedom as it can optimize the three scalar variables α^+, α^-, and β_0. This is of course beneficial because a better optimal value than in the previous two models can be achieved, but the objective function itself can become a less reliable approximate of the true leave-one-out figure. Note that the α_i's (together with β_0) in model (10)–(13) can

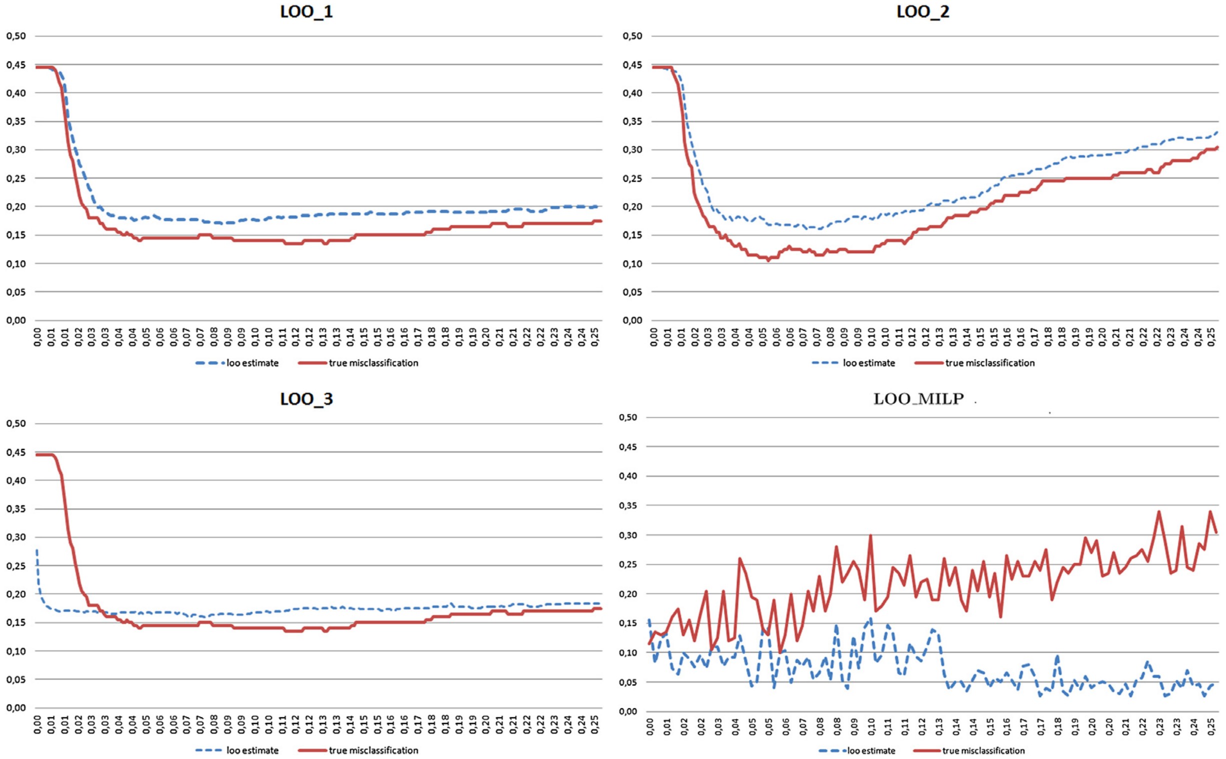

Fig. 2. Optimal model values ("loo estimate", in dashed blue) and "true misclassification" rate on the test set (in red) as a function of γ, for instance `Australian`. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

always be scaled, so we can without loss of generality impose the normalization condition $\alpha^+ + \alpha^- = 1$. Hence LOO_3 actually has only two degrees of freedom, as the only variables to optimize are $\alpha^+ \in [0, 1]$ and β_0.

An important feature of models LOO_1 and LOO_2 is that they allow for a very fast SVM training related to the leave-one-out estimate. In particular, model LOO_1 allows one to compute the exact leave-one-out estimate in just $O(p^2)$ time, assuming that the kernel function required $O(1)$ time, while model LOO_2 can also be implemented to run in $O(p^2)$ time – as shown in the next section – and produces a quite tight approximation of the corresponding leave-one-out estimate.

As to model LOO_3, in principle it can be solved as a MILP problem. In practice, however, one can just fix some guess values for α^+, e.g., $\alpha^+ \in \{0, \frac{1}{10}, \frac{2}{10}, \dots, 1\}$, define $\alpha^- = 1 - \alpha^+$, optimize β_0 through the fast procedure of the next section, and choose the option that gives the minimum objective function value. Besides its computational advantage over the MILP option, this procedure is beneficial in terms of overfitting in that it reduces the degrees of freedom of the training method.

The behavior of models LOO_MILP, LOO_1, LOO_2 and LOO_3 is illustrated in Fig. 2 on a sample case (instance `Australian` [6]). For each model, we plot two graphs in the corresponding subfigure, namely:

(a) the graph of the optimal value of the model, computed on the training set (called "loo estimate" in the figure)
(b) the graph of the a posteriori misclassification rate on the test set (called "true misclassification" in the figure), computed by the SVM corresponding to the optimal solution (α^*, β_0^*) of the model.

On the horizontal axis we consider different values for γ, ranging from 0 to 0.25.

There two conflicting requirements in the choice of the "best" training model.

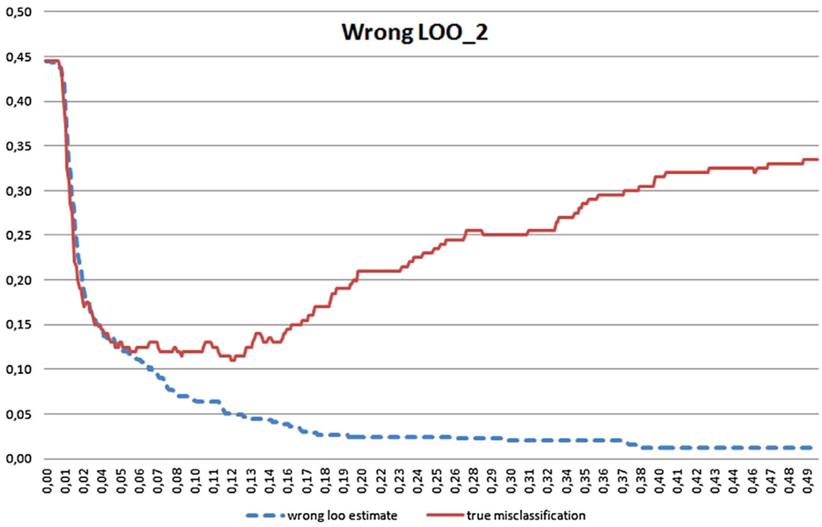

Fig. 3. Wrong LOO_2 optimal value with total instead of net signal (dashed blue) and “true” misclassification rate on the test set (red) as a function of γ, for instance `Australian`. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

On one hand, the “loo estimate” of the model should give a reliable approximation of the “true misclassification” over the test set, for all values of γ. If this is the case, it makes sense to choose the values (α^*, β_0^*) and γ^* that minimize the “loo estimate” on the test bed, and use them in the final SVM for the classification of unknown points. From this point of view, the fewer degrees of freedom in the model the better.

On the other hand, one is interested in attaining a small misclassification probability, i.e., a small value of both the “true misclassification” and “loo estimate” values. As the latter is in fact the optimal value of an optimization problem, one is tempted to favor models with a larger degree of freedom.

In this respect, model LOO_1 exhibits a very good similarity between the “loo estimate” and “true misclassification” graphs, which is not surprising as the optimal LOO_1 value coincides with the leave-one-out estimate—due to the very restrictive assumptions (10). However, the lack of any degree of optimization (besides γ) leads to an optimal LOO_1 value (i.e., estimated misclassification rate) that can hopefully be reduced. This is in fact what happens for models LOO_2 and LOO_3, where graph similarity is still very good – meaning that the model is still reliable – and the “loo estimate” values are smaller. The behavior of the LOO_MILP model (where no restrictions on α and β_0 are imposed) is quite interesting: the much larger degree of freedom leads to significantly better values of the “loo estimate”, but overfitting becomes large and makes this improvement useless as it does not translate into a better misclassification rate on the test set.

We finally observe that the nice similarity seen in Fig. 2 would be completely lost if the NAIVE model was used, i.e., if the net signal appearing in (11) were replaced by the standard total signal appearing in (7). Indeed, as already observed and clearly illustrated in Fig. 3, in this case the training objective function would favor an “impulsive” kernel and hence would lead to a very large γ value. This confirms the importance of the apparently minor change of using the net (as opposed to total) signal when counting misclassifications over the training set. In this respect, it is very important that training models such as NAIVE (and, to some extent, HINGE) do not pretend to optimize γ with the same model used for α and β_0, but resort to an external cross validation procedure.

4. Optimizing β_0 and γ

In this section we show how to optimize β_0 in the LOO models, assuming that all α_i's (as well as γ for Gaussian kernel) are fixed. The idea is very simple: according to (11) and by assuming a very small threshold

ϵ, a point x_j is classified correctly if and only if

$$y_j(\nu_j(\alpha) + \beta_0) > 0.$$

We start with $\beta_0 = -\infty$, for which only the -1 points in the training set are classified correctly, and initialize the number n_{ok} of correctly-classified points accordingly. Then we iteratively increase β_0, update n_{ok} and store the value of β_0 that maximizes n_{ok}. Of course, only the values $-\nu_j(\alpha)$ need to be considered explicitly when varying β_0: when β_0 meets $-\nu_j(\alpha)$, point x_j flips it classification status, and n_{ok} is increased (if $y_j = +1$) or decreased (if $y_j = -1$) by one unit. Therefore, to find the best β_0 it is enough to initially compute all $\nu_j(\alpha)$'s in $O(p^2)$ time, sort them in nondecreasing order in $O(p \log p)$ time, and then scan the $\nu_j(\alpha)$ values in $O(p)$ time. A pseudo-code of the whole procedure is given in Algorithm 1.

Algorithm 1: Fast leave-one-out optimization of β_0

```
input : the $p$ training points $(x_1, y_1), \cdots, (x_p, y_p)$ along with the associated $\alpha_i$'s, and the kernel
        function $K(\cdot, \cdot)$
output: the best parameter $\beta_0^*$ and the associated (approx.) leave-one-out estimate of the
        misclassification probability

$n_{ok} := 0$;
for $j := 1$ to $p$ do
    if $y_j = -1$ then $n_{ok} := n_{ok} + 1$;
    $\nu_j := 0$;
    for $i := 1$ to $p$ do
        if $i \neq j$ then $\nu_j := \nu_j + \alpha_i y_i K(x_j, x_i)$;
    end
end
Sort all $\nu_j$'s to get $\nu_{\sigma(1)} \leq \nu_{\sigma(2)} \leq \cdots \leq \nu_{\sigma(p)}$;
$\beta_0^* := -\infty$; $n_{ok}^* := n_{ok}$;
for $k := 1$ to $p$ do
    $j := \sigma(k)$;
    if $y_j = -1$ then
        $n_{ok} := n_{ok} - 1$;
    else
        $n_{ok} := n_{ok} + 1$;
        if $n_{ok} > n_{ok}^*$ then
            $n_{ok}^* := n_{ok}$;
            $\beta_0^* := -\nu_j$;
        end
    end
end
return $\beta_0^*$ and $(p - n_{ok}^*)/p$ ;
```

In case the Gaussian kernel is used, parameter γ needs to be carefully chosen as it affects the SVM in a very significant way. When the HINGE model is used, γ is not considered as a decision variable "inside the model" but it is chosen (together with C) by using an external k-fold validation procedure. When the LOO models are used, instead, one can think of optimizing γ by using the same LOO objective function, thus bringing γ inside the model itself.

Table 1
Characteristics of the instances in our testbed.

Instance	n. points	n. features
Australian	690	43
Breast	683	9
Bupa	345	6
German	1000	24
Heart	270	23
Ionosphere	351	34
Pima	768	8
Sonar	208	60
Wdbc	568	30
Wpbc	194	33

In particular, we have observed that the optimal value of the LOO_1–LOO_3 models typically behaves as an almost unimodal function of γ, i.e., this value is (almost) monotonically decreasing in the interval $(0, \gamma_{\min})$, and (almost) monotonically increasing in $(\gamma_{\min}, +\infty)$; see the dashed-blue graphs in Fig. 2 for an illustration. Therefore we can approximately minimize it through a simple (well known) bisection method that works as follows.

Let $V(\gamma)$ be the optimal LOO value for a given $\gamma \geq 0$, computed as explained in the previous section, and assume that it is exactly unimodal in the interval $[0, +\infty)$. At each iteration, we know the value of V for three points $\gamma_L < \gamma_C < \gamma_R$ (L-C-R for left-center-right), where $V(\gamma_C) < \min\{V(\gamma_L), V(\gamma_R)\}$ which ensures that the optimal γ belongs to the "uncertainty interval" (γ_L, γ_R). Then we choose the largest subinterval between $[\gamma_L, \gamma_C]$ and $[\gamma_C, \gamma_R]$, and compute $V(\gamma_M)$ for its middle point γ_M (say). Depending on the value of $V(\gamma_M)$ and because of the unimodal assumption, we can then exclude either γ_L or γ_R, rename the three remaining points, and repeat. The algorithm terminates when the uncertainty degree $\gamma_R - \gamma_L$ is sufficiently small, and the final γ_C is returned as an approximation of the optimal γ.

According to our computational experience, the above bisection method performs reasonably well even when the function is only approximately unimodal, provided that the first interval $[\gamma_L, \gamma_R]$ is sufficiently narrow and $V(\gamma_C)$ is significantly smaller than $\min\{V(\gamma_L), V(\gamma_R)\}$.

5. Computational tests

The methods presented in previous sections were implemented in C and run on a notebook with Intel Core i7@2.7 GHz with 8 GB RAM (single thread). The state-of-the-art IBM ILOG CPLEX 12.5 commercial solver (single thread) was used for solving our benchmark models HINGE and LOO_MILP, while all other codes are completely self-contained and do not need any external optimization tool.

Our test methodology followed closely that used in Brooks [6]. In particular, we addressed the same standard testbed. To be fair, we omitted instance `Adult` as it contains a very large number of points (30,157) and would be impossible to handle by our benchmark methods (HINGE and LOO_MILP) without using specific sampling techniques, as well as other artificial datasets. The main characteristics of the instances in our testbed are reported in Table 1. All instances can be downloaded in a unified format from http://www.dei.unipd.it/~fisch/datasetSVM.tar.gz.

For each instance in our testbed, 10 subinstances were constructed whose points were randomly assigned to the training set (with uniform probability of 70%) or to the test set (30% probability). As in [6], for each subinstance we applied the following normalization step: the average avg_f and standard deviation sd_f of each feature $f \in \{1, \ldots, n\}$ were computed over the training set, and all points x_i were normalized by replacing x_{if} by $(x_{if} - \text{avg}_f)/\text{sd}_f$; features f with $\text{sd}_f = 0$ are instead removed from all points (both in the training and test sets).

Table 2 (top) reports the outcome of our experiments for the Gaussian kernel, and compares the performance of the HINGE and of our four "LOO models" (namely, LOO_1, LOO_2, LOO_3, and LOO_MILP) in

Table 2
Percentage value and standard deviation (sd) of optimal LOO value (loo) and misclassification on the test set (mis, in boldface), and CPU time in seconds (t.); averages over 10 runs for each instance. For each instance, the best mis entry is underlined.

Instance	HINGE		LOO_1					LOO_2					LOO_3					LOO_MILP				
	mis	t.	loo	sd	mis	sd	t.	loo	sd	mis	sd	t.	loo	sd	mis	sd	t.	loo	sd	mis	sd	t.
Australian	**18.20**	728.3	16.04	1.13	**16.28**	2.22	0.4	14.74	1.15	**15.31**	2.78	0.8	14.64	1.17	**15.68**	2.59	10.6	2.51	0.88	**22.68**	2.94	435.1
Breast	**3.20**	627.5	3.22	0.45	**3.61**	0.99	0.4	2.97	0.29	**4.00**	0.98	0.8	2.41	0.35	**3.50**	1.04	9.8	0.04	0.08	**6.18**	1.80	41.4
Bupa	**32.70**	88.2	36.67	2.31	**38.68**	4.39	0.0	33.98	1.93	**39.00**	2.43	0.1	33.20	1.37	**39.61**	3.47	1.7	4.41	1.85	**39.08**	3.19	429.6
German	**27.97**	2453.2	27.19	0.95	**26.52**	3.18	1.0	26.57	1.20	**26.45**	3.16	2.2	24.09	0.84	**25.74**	3.27	22.9	19.81	2.14	**29.22**	3.21	462.8
Heart	**23.00**	47.9	17.88	1.02	**19.05**	3.26	0.0	17.12	0.84	**19.11**	3.19	0.1	16.47	0.98	**18.77**	2.21	1.0	1.29	0.43	**23.22**	6.17	141.9
Ionosphere	**6.30**	123.5	19.73	1.67	**23.38**	2.97	0.0	4.93	0.26	**6.66**	1.19	0.1	4.23	0.35	**6.34**	1.50	1.8	0.00	0.00	**6.41**	1.59	5.4
Pima	**25.70**	1231.0	25.24	1.42	**25.59**	2.52	0.6	24.39	1.23	**25.76**	2.76	1.2	23.37	1.01	**25.68**	2.95	14.8	11.82	4.10	**28.20**	3.20	450.4
Sonar	**18.00**	25.2	16.49	0.97	**17.03**	3.64	0.0	15.88	1.54	**17.93**	2.86	0.0	12.91	0.83	**11.44**	3.39	0.6	0.00	0.00	**15.56**	3.62	6.3
Wdbc	**4.50**	646.4	5.15	0.60	**5.66**	1.43	0.2	4.65	0.73	**5.29**	1.70	0.5	3.58	0.57	**4.63**	1.44	6.9	0.02	0.07	**4.82**	1.46	8.7
Wpbc	**23.70**	29.0	22.72	1.92	**21.59**	4.78	0.0	22.28	2.13	**22.67**	3.90	0.0	21.03	1.92	**23.14**	2.71	0.6	2.05	0.84	**32.66**	5.40	184.5
Average	**18.33**	600.0	19.03	1.24	**19.74**	2.94	0.3	16.75	1.13	**18.22**	2.50	0.6	15.59	0.94	**17.45**	2.46	7.1	4.20	1.04	**20.80**	3.26	216.6
Estim. err.			−3%					−8%					−10%					−79%				

terms of misclassification rates over the test set (in boldface to ease comparison) and computing times on our hardware.

HINGE misclassification figures are from Table 5.5 in Petterle [5]; note that the results in Table 3 of [6] are not comparable because they exploit four different kernels (linear, degree-2 polynomial, degree-9 polynomial, and Gaussian). As to HINGE computing times, they refer to the IBM ILOG CPLEX 12.5 barrier solver, and include time spent for choosing the best (γ, C) pair (out of $5*5$ options) through a 5-fold cross validation procedure—meaning that $5*5*5+1 = 126$ quadratic problems need be solved for each subinstance, as in [6]. We note here that specialized codes exist for solving HINGE (in its dual form), so the speedups reported have to be considered as just informative.

Columns LOO_1 to LOO_MILP refer to the models defined in Section 3. For all models, the choice of γ was performed through the bisection procedure of Section 4, starting from $\gamma_L = 0.01$, $\gamma_R = 1$, and $\gamma_C = (\gamma_L + \gamma_R)/2$, and ending when $\gamma_R - \gamma_L < 0.01$ (this typically required about 10 iterations). For LOO_2 and LOO_3, the best β_0 parameter was found through the efficient procedure of Section 4.

Model LOO_MILP was solved through the CPLEX's general-purpose MILP solver with a time limit of 30 s for each call, with default parameters except cut generation (all cuts were deactivated) and numerical tolerances `CPX_PARAM_EPINT` and `CPX_PARAM_EPRHS` set to 0.0 and 1e−9, respectively. The LOO_2 solution was provided on input to the solver as a warm start, switching to the so-called *polishing* refining heuristic after 10 s. In the LOO_MILP model, we set $\epsilon = 10^{-5}$ (which is 10 times larger than the numerical tolerance of 10^{-6} used for counting misclassifications in the test set) and $M = 10,000$. We stress again that model LOO_MILP was considered in our experiments mainly for benchmark purposes, as its performance (in terms of both speed and quality) is not competitive with the other LOO models.

For each instance and for each LOO model, five entries are reported in Table 2, each corresponding to averages over the 10 subinstances. The first two entries (`loo` and `sd`) give the percentage average and standard deviation of the optimal value of the LOO model (called "loo estimate" in Fig. 2), that we see as an estimate of the misclassification rate over the test set. The next two entries (`mis` and `sd`) give the percentage average and standard deviation of the misclassification rate over the test set (called "true misclassification" in Fig. 2). The 5-th entry (`t.`) reports the average computing time for the overall training, in CPU seconds on our hardware.

The last two rows of the table report percentage averages over the entire testbed (Average) and the percentage estimation error computed as $100 * ($`loo-mis`$)/$`mis` (Estim. err.), negative as `loo` tends to be a lower bound on `mis`.

Comparing the four `loo` columns confirms that having more degrees of freedom is beneficial in terms of optimal LOO value: on average, for LOO_1, LOO_2, LOO_3, and LOO_MILP the `loo` value is 19.03%, 16.75%, 15.59%, and 4.20%, respectively. However, these figures estimate the "true" misclassification rate with an increasing error of 3%, 8%, 10%, and 79%, so the improved `loo` values do not necessarily translate into better misclassifications rates, that in fact are 19.74%, 18.22%, 17.45%, and 20.80%, respectively.

Comparing "true" misclassification rates on the test set (in boldface) shows that the LOO models are competitive with HINGE, with the exception of LOO_1 that is heavily penalized by its bad performance on `Ionosphere`, and of LOO_MILP because of its very large overfitting. In particular, LOO_2 has a slightly better average misclassification rate than HINGE (18.22% vs. 18.33%), while LOO_3 with its 17.45% average misclassification rate qualifies as the best classifier.

The advantage of the LOO_1–LOO_3 models in terms of speed is quite impressive: on average, HINGE training required about 600 s, while LOO_1, LOO_2, and LOO_3 required just 0.3, 0.6, and 7.1 s, respectively, meaning a speedup of 2–3 orders of magnitude (at least, with respect to the general-purpose CPLEX solver) to achieve a comparable or better classification accuracy. The full LOO_MILP model, instead, appears less attractive also in terms of computing time, as it required 216.6 s on average.

6. Conclusions

We have addressed how to effectively train a Support Vector Machine with Gaussian kernel. We started with a naive Mixed-Integer Linear Programming model, and elaborated it to reduce its overfitting behavior. The resulting model aims at minimizing the leave-one-out estimate of the misclassification probability – a figure that we explicitly use as objective function – and does not require an external (time consuming) cross validation procedure. A main idea of our approach is to reduce overfitting on the training set by imposing additional constraints on the model—rather than penalizing risky solutions through the objective function, as it is done customary. For Gaussian kernel, a natural approach is to force all parameters α_i's to be equal, or to only depend on the $+1/-1$ class of the corresponding point.

Efficient algorithms based on sorting have been proposed for simplified versions of our model, that do not require any external optimization tool. According to our computational results, our method is competitive with classical hinge-loss training in terms of accuracy (at least, on our testbed and by using the Gaussian kernel), but runs much faster.

Future research should address how to allow for a controlled increase of the degrees of freedom in the training phase, that does not imply an unacceptable increase of overfitting. Indeed we conjecture that, at least for specific classes of problems, one could easily guess some a priori properties of the SVM parameters in a good classifier, and impose them as constraints in our training model. The adaptation of this approach to the linear kernel case is also an interesting research topic.

Acknowledgments

This research was supported by the University of Padova (Progetto di Ateneo "Exploiting randomness in Mixed Integer Linear Programming"), and by MiUR, Italy (PRIN project "Mixed-Integer Nonlinear Optimization: Approaches and Applications"). Thanks are due to two anonymous referees for their helpful comments.

References

[1] V. Vapnik, The support vector method, in: ICANN97, 1997, pp. 263–271.

[2] B. Schölkopf, K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio, V. Vapnik, Comparing support vector machines with gaussian kernels to radial basis function classifiers, IEEE Trans. Signal Process. 45 (1997) 2758–2765.

[3] E. Carrizosa, D. Romero Morales, Supervised classification and mathematical optimization, Comput. OR 40 (1) (2013) 150–165.

[4] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Data Mining, Inference and Prediction, Springer-Verlag, 2001.

[5] L. Petterle, A computational analysis of optimization models for support vector machines (Master's thesis), University of Padova, Engineering School, 2012, http://www.dei.unipd.it/~fisch/ricop/tesi/tesi_Petterle_2012.pdf.

[6] J.P. Brooks, Support vector machines with the ramp loss and the hard margin loss, Oper. Res. 59 (2) (2011) 467–479.