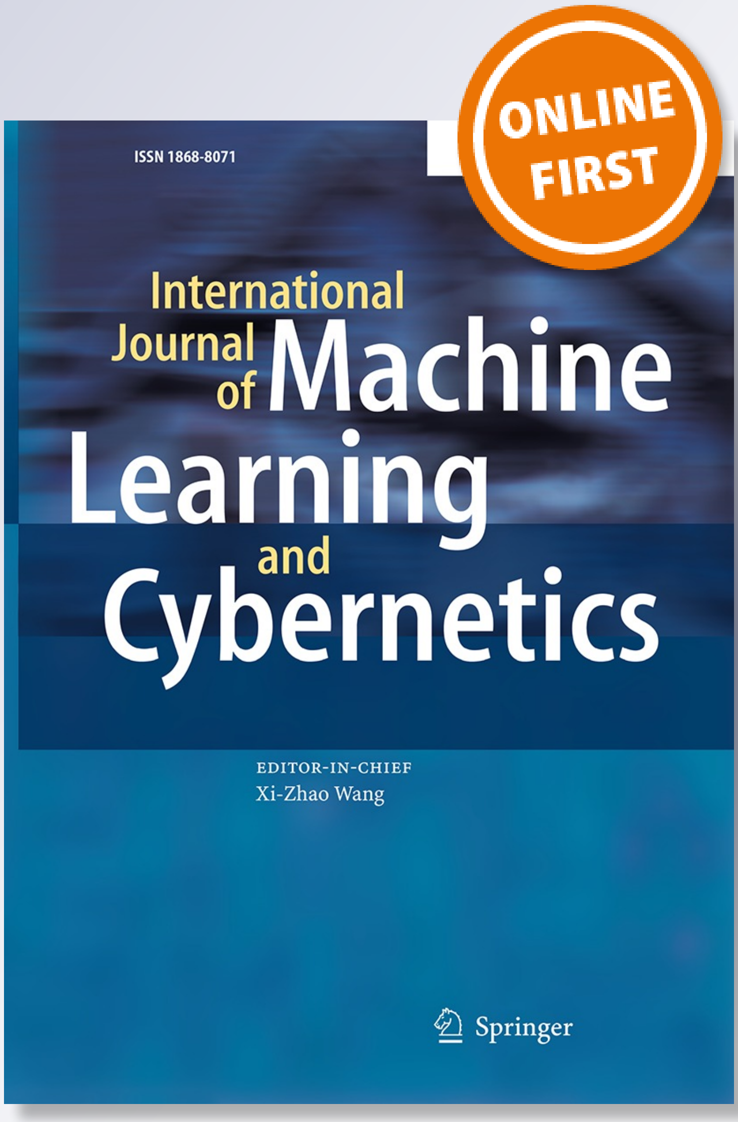
# *A nonlinear least squares quasi-Newton strategy for LP-SVR hyper-parameters selection*

**Pablo Rivas-Perea, Juan Cota-Ruiz & Jose-Gerardo Rosiles**

**International Journal of Machine Learning and Cybernetics**

ISSN 1868-8071

Int. J. Mach. Learn. & Cyber.
DOI 10.1007/s13042-013-0153-9



Int. J. Mach. Learn. & Cyber.
DOI 10.1007/s13042-013-0153-9

ORIGINAL ARTICLE

# A nonlinear least squares quasi-Newton strategy for LP-SVR hyper-parameters selection

**Pablo Rivas-Perea · Juan Cota-Ruiz · Jose-Gerardo Rosiles**

Received: 7 December 2011 / Accepted: 29 December 2012

**Abstract** This paper studies the problem of hyper-parameters selection for a linear programming-based support vector machine for regression (LP-SVR). The proposed model is a generalized method that minimizes a linear-least squares problem using a globalization strategy, inexact computation of first order information, and an existing analytical method for estimating the initial point in the hyper-parameters space. The minimization problem consists of finding the set of hyper-parameters that minimizes any generalization error function for different problems. Particularly, this research explores the case of two-class, multi-class, and regression problems. Simulation results among standard data sets suggest that the algorithm achieves statistically insignificant variability when measuring the residual error; and when compared to other methods for hyper-parameters search, the proposed method produces the lowest root mean squared error in most cases. Experimental analysis suggests that the proposed approach is better suited for large-scale applications for the particular case of an LP-SVR. Moreover, due to its mathematical formulation, the proposed method can be extended in order to estimate any number of hyper-parameters.

**Keywords** Support vector regression · Hyper-parameters · Large-scale LP-SVR

This work was supported in part by NASA Goddard Space Flight Center's GSSP 2009 program and by the National Council for Science and Technology (CONACyT), Mexico, under Grant 193324/303732.

P. Rivas-Perea (✉)
Department of Computer Science, Baylor University,
One Bear Place #97356, Waco, TX 76798-7356, USA
e-mail: Pablo_Rivas_Perea@Baylor.edu
URL: http://baylor.academia.edu/prp

J. Cota-Ruiz
Department of Electrical and Computer Engineering,
Autonomous University of Ciudad Juarez (UACJ),
Ave. del Charro #450 Nte. C. P. 32310,
Ciudad Juarez, Chihuahua, México
e-mail: jcota@uacj.mx

J.-G. Rosiles
Science Applications International Corp., 7400 Viscount Blvd,
El Paso, TX 79925, USA
e-mail: gerardo_rosiles@yahoo.com

## 1 Introduction

Support vector machines (SVMs) are considered part of the supervised learning methods [19]. In general, SVMs take a given data set, $\mathcal{T} = \{\mathbf{x}_i, y_i\}_{i=1}^{N}$, consisting of $N$ input patterns, $\mathbf{x} \in \Re^M$, and a desired output class value $y \in \Re$, and aims to find the minimum number of input patterns that are relevant and necessary for solving a classification or regression problem. Such reduced number of input patterns are called "support vectors" [20, 62]. SVM-based learning methods exhibit great generalization capabilities in both classification and regression problems [42, 62]. SVMs possess the highest performance rankings in many different machine learning problems and applications in cybernetics [30, 36, 57, 65, 69, 71]. However, SVM-based methods' robustness dramatically depends on the the set of parameters necessary for its implementation [45]. This set of parameters are known as "hyper-parameters", hereafter denoted as $\boldsymbol{\theta}$ [59]. Both original models for SVMs and support vector machines for regression (SVRs) take the data available to construct an SVM/SVR model and project it into a higher-dimensional space using a kernel function that typically requires one hyper-parameter, $\theta_1 = \gamma$; during the construction of the model, *i.e.*, training of an

SVM/SVR, every classification or regression mistake is penalized using a loss function that also requires a second hyper-parameter, $\theta_2 = \epsilon$; and the third hyper-parameter is a regularization parameter, $\theta_3 = C$, which is applicable only in soft-margin SVM formulations and it is necessary for the regularization of the SVM/SVR optimization problem [54], without regularization, an SVM would be incapable of working with non-separable data. The number of hyper-parameters depends directly from the type of SVM/SVR model the researcher chooses [56]. Therefore, hyper-parameters estimation is currently one of the general open problems in SVM and SVR learning methods [27, 59].

Broadly speaking, one tries to find a set of $n$ hyper-parameters, $\boldsymbol{\theta} = \{\boldsymbol{\theta_1}, \boldsymbol{\theta_2}, \ldots, \boldsymbol{\theta_n}\}$, minimizing the generalization error of an SVM or SVR learning machine; the generalization error is typically understood as the likelihood of a learning machine to produce an error for any given input [18]. Thus, it makes sense to find a set of hyper-parameters that minimize the generalization error of the learning machine; however, finding the true generalization error is not a trivial problem. In fact, Anguita et al. [1], claim that "the estimation of the generalization error of a learning machine is still the holy grail [sic] of the research community". The previous statement implies it is virtually impossible to find the true generalization error; at best we only can attempt to find good estimates of it. And this is important because if one can find a good generalization error estimate, then, perhaps, it could be feasible to develop a method to find the set of hyper-parameters that minimizes such generalization error estimate. However, a similar problem exists with respect to the methods that try to minimize the generalization error: they are either computationally expensive but good in minimizing the error or have faster convergence and poor minimization of the generalization error. Consequently, it could be appropriate to divide the problem of finding the hyper-parameters in two parts: (i) finding a function that produces a "good" generalization error estimate, and (ii) finding a feasible method to minimize that function. A brief discussion of existing methods in both categories is introduced next.

### 1.1 On generalization error estimators

The most popular methods to estimate the true generalization error include techniques of $\mathcal{K}$-fold cross validation [12], leave-one-out cross validation [1], maximal discrepancy [2], bootstrapping [22], and compression bound [1, 63]. Particularly, the compression bound provides an estimate of the generalization error considering only those elements of the data set that are not support vectors. That is, the authors follow the well known concept of dividing the data set into a training set $\mathcal{T}$ and a testing set $\mathcal{D}$ [19], where the training set is used to teach (or train) the machine learning model, and the testing set is used to analyze the final quality of the trained model. Then, the authors use the "non-support vectors" as testing set to provide estimates of the generalization error [1].

The maximal discrepancy method, in contrast, trains an SVM model as normal, and then re-trains another model but this time perturbing the "target" outputs to observe the changes produced by both models. In fact, half of the target output values are actually flipped. Then an estimate of the bounds of the generalization error is given as a function of the observed changes [2].

Bootstrapping consists of diving the training set $\mathcal{T}$ into a new, smaller, training set and a validation set. The selection is performed based on the statistical properties of the data; several groups can be formed and the data in each group could be repeated several times as necessary [22]. An estimate of the generalization error is given by averaging the error produced by the different realizations of the method.

Similarly, the $\mathcal{K}$-fold cross validation method divides the training set into a given number of groups, $|\mathcal{K}|$, of (ideally) equal size. These groups are known as folds or slices [19]. The learning machine is trained with all the slices except for one of them, which is then used as a validation set. The procedure is repeated for each group and the results are averaged and given as an estimate of the generalization error [12]. The leave-one-out (LOO) cross validation method is the extreme case of a $\mathcal{K}$-fold cross validation, when the number of samples in the training set is also the number of folds [1], *i.e.*, $|\mathcal{K}| = N$.

The common drawback of most of these methods is that they are problem dependent [6]. This statement is confirmed by Anguita et al. [1]; they performed a comprehensive study on the above techniques and ranked such techniques according to their ability to estimate the true generalization error. The result concluded that most of the methods they evaluated either underestimate or overestimate the true generalization error. Nonetheless, their research suggests that the $\mathcal{K}$-fold cross validation technique is one of the less risky techniques for estimating the true generalization error. Cawley's research [6] also demonstrates that LOO and $\mathcal{K}$-fold cross validation methods work much better than other methods. Furthermore, the research of Smets et al. [58] on the different methods for estimating the true generalization error also confirmed that a $\mathcal{K}$-fold cross validation provides a smoother surface for hyper-parameters estimation, suggesting that gradient-based methods could be safely applied, *e.g.*, quasi-newton methods [7]. Therefore, this research proposes the usage of a $\mathcal{K}$-fold cross validation technique to estimate the true test generalization error along with a gradient-based method that minimizes such an error. Current methods for minimizing the generalization error are discussed next.

## 1.2 On methods that find a set of hyper-parameters

There are a number of existing methods to find a set of hyper-parameters. These could be categorized in two main groups: (i) methods that exploit well established optimization methods, and (ii) those who use heuristics approaches to find a set of hyper-parameters. From the optimization point of view, the solution to the problem of finding the set of hyper-parameters is clearly non-trivial. To illustrate this, Fig. 1 is presented.

The figure shows the root mean squared error (RMSE) surface of a linear programming support vector machine for regression (LP-SVR) as a function of its hyper-parameters $[C, \gamma] = \boldsymbol{\theta}$. Given that the problem is non-convex, we can observe how the error surface is non-smooth and has many local minima. Therefore, "smoother" error functions ought to be considered as well as new approaches for finding the global minimum or at least a "good" local minimum. From such approaches is also desired a fast rate of convergence.

The most popular approach for finding the hyper-parameters is a heuristic called "grid search" which could be thought as a "brute-force" method. It searches all over the hyper-parameters space in equally spaced intervals in a logarithmic or linear scale. When finished evaluating all points in the grid, a new grid search can be employed to refine the search for the best set of hyper-parameters. This method is computationally intense and expensive, yet it is still considered an alternative in some cases [45].

Another popular approach that uses a fast heuristic approach to find a set of hyper-parameters is the one

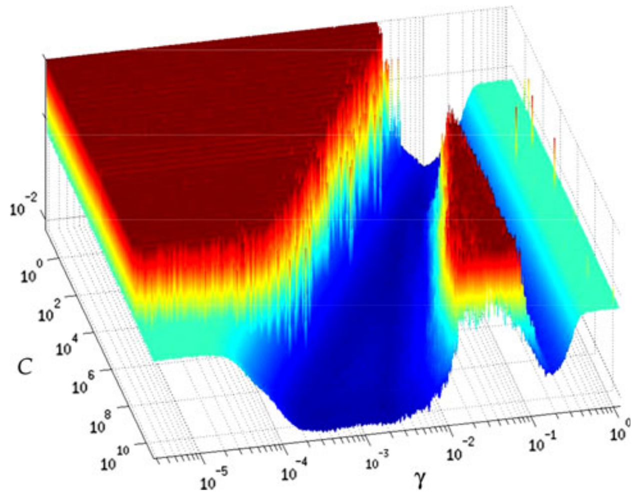


**Fig. 1** Response of the root mean squared error as a function of $\boldsymbol{\theta} = [\boldsymbol{C}, \boldsymbol{\gamma}]$ using the *Bodyfat* data set [48]. Note the non-smooth surface error, and how it has at least two valleys with many possible local minima. Here $C$ is evaluated over the interval $[10^0, \ldots, 10^{10}]$ and $\gamma$ is evaluated over the interval $[10^0, \ldots, 10^5]$.

introduced by Momma et al. [41]. The authors introduced the "Pattern Search" method for SVR, showing that the method could achieve fairly good, though not the best, results in fewer number of function calls. The authors propose an algorithm that starts from a randomly chosen point, in the hyper-parameters space, evaluating its own cost according to a quality function that depends on the mean squared error and the variance of the error at that point. The estimation of the error at any given point is based on the LOO technique. The algorithm seeks to explore other points in the hyper-parameters space guided by a fixed pattern, typically corresponding to the surrounding neighbors. In doing this, the algorithm avoids the estimation of a gradient, thus, reducing the number function calls.

Later, Ito et al. [23] proposed a method that uses the "minimum cross validation" technique to determine the direction of the hyper-parameters that produce a descent in the mean squared error function using LOO. They claim that their optimization method is faster than the traditional grid search and that it also produces a lower mean squared error with LOO than the grid search method. This demonstrated that when optimization techniques are used, lower errors can be achieved. However, the authors did not provide any reference as to how the derivatives (in the Hessian) are computed.

Cherkassky et al. [8] proposed a method to determine the values for the hyper-parameters using solely the training set with no slices. The authors demonstrated that statistical properties of the training set provide sufficient insight to produce an educated guess of a good set of hyper-parameters. Their method is evidently less expensive in terms of function calls, e.g., when compared to cross validation-based techniques. However, it does not provide the best set of hyper-parameters. Cherkassky's method will be revisited again in this document when describing the proposed optimization algorithm.

Not much time after Cherkassky's research, Kobayashi et al. [32] improved Ito's method [23]; the authors noticed that the algorithm for estimating the hyper-parameters introduced by Ito et al., could take advantage of the properties of SVR itself, i.e., they realized that the elements of the data-set that are not SVs could be discarded in the computation of the derivatives. They started with a given set of hyper-parameters and then solved the whole problem with such, which helps in identifying the SVs. The elements that were not SVs were discarded in any subsequent computation. Based on the same concept, they went further and discarded more SVs while estimating the Hessian. Although the total CPU time was reduced, in comparison with Ito's original work, no insight was given as to how the initial hyper-parameters were determined; the computational cost of the Hessian and any information concerning the computation of the Hessian was very obscure.

Karasuyama et al. [26] noticed the problems in the models of Kobayashi et al. [32] and of Ito et al. [23]. The authors realized that at the moment of inverting the Hessian matrix, their inverse estimation algorithm yielded inconsistencies. Then they proposed a different method for estimating the gradient, which the authors claimed was free of inconsistencies, i.e., it had guaranteed non-singularity. The authors claim that this new method reduces the CPU time of previous versions by more than one half. However, their method has contradictory findings to that of Cherkassky et al. [8], who demonstrated that when $\epsilon = 0$ (a loss function hyper-parameter), it falls in the case of the "least-module", where all data-points are SVs. Therefore, the computation time should be increasing as Karasuyama's algorithm reaches $\epsilon = 0$. Furthermore, the author compares his results to that of an $\ell_1$-SVR, which is known to be more efficient [70], and provides no insight regarding what parameters where used in this kind of SVR.

Later, Karasuyama et al. [27] improved his own method by implementing a technique called "accurate online"-SVR (or AOSVR) [38]. Such method reduces the SVR solution time by removing samples from the training data, solving, and then observing the behavior of the SVR after removal. Depending of what the relevancy of the data was, it could be either removed from the training set or preserved for further function calls. The authors claim that this procedure accelerates their own previously revised model [26], without loosing generalization capabilities.

Ortiz-García et al. [45] have recently proposed upper bounds for the hyper-parameters with the purpose of reducing the search space and consequently reducing the total estimation time. The authors studied the relationship and relevancy of the hyper-parameters of an SVR and demonstrated that the proposed boundaries can be estimated only from training data, which is similar to Cherkassky's approach [8].

Other researchers have also studied the SVM/SVR hyper-parameters estimation problem from other perspectives. For instance, Yuan et al. [66] presented a method based on chaos optimization theory, and Ren et al. [51] used Genetic Algorithms and Particle Swarm optimization to obtain a good set of hyper-parameters. However, the methodology particularly proposed by Ren et al. seems to contradict the findings of Smets et al. [58] in the sense that Ren claims the surface of a $\mathcal{K}$-fold cross validation does not provide a smooth surface; however, Smets provides with more experiments and a more sound theoretical background to support his claim, while Ren falls short in this sense. Even Duan's research [12] demonstrates that $\mathcal{K}$-fold cross validation is one of the best generalization error estimators; this situation suggests a lack of agreement among researchers.

The methods briefly discussed above show the variety of heuristic and optimization-based approaches for hyper-parameters estimations. Some findings seem to contradict others and no agreement has been achieved thus far [45]. Perhaps one can argue that such differences rely on the fact that those methods use different SVR/SVM models, with different error functions, and with different generalization error estimators. For example, Cherkassky et al. [8], Momma et al. [41], Ortiz-García et al. [45], and Ren et al. [51] use the standard SVR with an $\epsilon$-insensitive loss function; while Ito et al. [23], Karasuyama et al. [26], Kobayashi et al. [32], and Smets et al. [58] use the standard SVR but with a quadratic $\epsilon$-insensitive loss function; and Cawley [6] uses a least squares SVM (LS-SVM).

The lack of agreement cries out for more research in this field and demands the exploration of different SVR models and methodologies apart from the ones discussed above. Therefore, the aim of this research is to provide a different perspective to the problem of hyper-parameters estimation. This research explores the usage of a quasi-Newton-based nonlinear least squares method to minimize the generalization error estimated with a $\mathcal{K}$-fold cross validation method, thus, providing a novel alternative to the problem of finding the hyper-parameters. Furthermore, this research explores an LP-SVR formulation that was originally introduced in [54] and that has not been reported in the literature to the best of the authors knowledge.

This document is organized as follows: Sect. 2 defines and discusses a general nonlinear least squares method that uses a set of individual error functions to minimize the overall generalization error of any machine learning problem. Section 3 introduces the LP-SVR formulation that is used in this research, which requires a particular number of hyper-parameters; such hyper-parameters are addressed in Sect. 4 along with specific error functions chosen to solve the optimization problem. Section 5 discusses the implementation details and other considerations applicable to the proposed method. The experimental results are discussed in Sect. 6, and conclusions are drawn in Sect. 7.

## 2 Error function minimization using nonlinear least squares

Let $r(\boldsymbol{\theta}|\mathcal{T}), \boldsymbol{r}: \Re^{\boldsymbol{n}} \mapsto \Re$ be a real function representing some estimate of error; where $\boldsymbol{\theta} \in \Re^{\boldsymbol{n}}$ is a vector of parameters, and $\mathcal{T} = \{\mathbf{x}_i, y_i\}_{i=1}^{N}$ defines a training set given by $N$ samples of the $M$-dimensional data vector $\mathbf{x} \in \Re^M$, and a desired output class value $y \in \Re$. Then, let $\mathbf{R} : \Re^n \mapsto \Re^m$ be denoted as

$$\mathbf{R}(\boldsymbol{\theta}|\mathcal{T}) = \begin{bmatrix} \boldsymbol{r_1}(\boldsymbol{\theta}|\mathcal{T}) \\ \boldsymbol{r_2}(\boldsymbol{\theta}|\mathcal{T}) \\ \vdots \\ \boldsymbol{r_m}(\boldsymbol{\theta}|\mathcal{T}) \end{bmatrix}_{\boldsymbol{m\times 1}}. \tag{1}$$

The vector $\mathbf{R}(\boldsymbol{\theta}|\mathcal{T})$ represents $m$ different measures of error provided a vector of hyper-parameters $\boldsymbol{\theta} = [\boldsymbol{\theta_1}, \boldsymbol{\theta_2}, \ldots, \boldsymbol{\theta_n}]$, and training data $\mathcal{T}$. This research aims to minimize $\mathbf{R}(\boldsymbol{\theta}|\mathcal{T})$, which reduces the “residual error”. Here, the term “generalization error” is equivalent to the term “residual error”; however, the latter term is used following the standard optimization terminology [11]. Indeed, this particular kind of minimization problems are known as “nonlinear least squares problems” [11, 44]; and following the nonlinear least squares model, let $f(\boldsymbol{\theta}|\mathcal{T})$ denote the following function:

$$f(\boldsymbol{\theta}|\mathcal{T}) = \frac{\boldsymbol{1}}{\boldsymbol{2}}||\mathbf{R}(\boldsymbol{\theta}|\mathcal{T})||_2^2, \tag{2}$$

where $||\cdot||_2$ denotes the $\ell_2$-norm (a.k.a. euclidean norm). Then, since $f : \Re^n \mapsto \Re$, then its derivatives can be expressed in terms of the Jacobian given by

$$\mathbf{J}_f(\boldsymbol{\theta}|\mathcal{T}) = \left[\frac{\partial r_j}{\partial \theta_i|\mathcal{T}}\right] \begin{matrix} 1,2,\ldots,m \\ 1,2,\ldots,n \end{matrix} = \cdots \begin{bmatrix} \nabla r_1(\boldsymbol{\theta}|\mathcal{T})^{\boldsymbol{T}} \\ \nabla r_2(\boldsymbol{\theta}|\mathcal{T})^{\boldsymbol{T}} \\ \vdots \\ \nabla r_n(\boldsymbol{\theta}|\mathcal{T})^{\boldsymbol{T}} \end{bmatrix}_{n\times n}, \tag{3}$$

where $\nabla r_n(\boldsymbol{\theta}|\mathcal{T})$ denotes the gradient of the $n$-th function, given by

$$\nabla r_n(\boldsymbol{\theta}|\mathcal{T})^{\boldsymbol{T}} = \begin{bmatrix} \frac{\partial \boldsymbol{r_n}}{\partial \boldsymbol{\theta_1}} & \frac{\partial \boldsymbol{r_n}}{\partial \boldsymbol{\theta_2}} & \cdots & \frac{\partial \boldsymbol{r_n}}{\partial \boldsymbol{\theta_n}} \end{bmatrix}_{\boldsymbol{1\times n}}. \tag{4}$$

Therefore, the gradient $\nabla f(\boldsymbol{\theta}|\mathcal{T})$ and Hessian $\nabla^2 f(\boldsymbol{\theta}|\mathcal{T})$ are defined as follows:

$$\nabla f(\boldsymbol{\theta}|\mathcal{T}) = \mathbf{J}_{\boldsymbol{f}}(\boldsymbol{\theta}|\mathcal{T})^{\boldsymbol{T}}\mathbf{R}(\boldsymbol{\theta}|\mathcal{T}), \tag{5}$$

$$\nabla^2 f(\boldsymbol{\theta}|\mathcal{T}) = \mathbf{J}_{\boldsymbol{f}}(\boldsymbol{\theta}|\mathcal{T})^{\boldsymbol{T}}\mathbf{J}_{\boldsymbol{f}}(\boldsymbol{\theta}|\mathcal{T}) + \mathbf{Z}(\boldsymbol{\theta}|\mathcal{T}), \tag{6}$$

where $Z(\boldsymbol{\theta}|\mathcal{T}) = \sum_{\boldsymbol{j=1}}^{\boldsymbol{m}} \boldsymbol{r_j}(\boldsymbol{\theta}|\mathcal{T})\nabla^{\boldsymbol{2}}\boldsymbol{r_j}(\boldsymbol{\theta}|\mathcal{T})$.

The solution to the problem, indicated with a star ($\bigstar$), is the vector of parameters $\boldsymbol{\theta}^\bigstar$ which, given a training set $\mathcal{T}$, produces minimal error functions, i.e., $f(\boldsymbol{\theta}^\bigstar|\mathcal{T})$ is minimum; then Newton's method can be used since $f(\boldsymbol{\theta}^\bigstar|\mathcal{T})$ is continuously differentiable on $\Re^n$.

The method of Newton is well known from basic calculus and optimization courses [11], and it appears summarized in Algorithm 1.

**Algorithm 1** Newton method to find $\boldsymbol{\theta}^\star|\mathcal{T}$ that minimizes $f(\boldsymbol{\theta}^\star|\mathcal{T}) = \frac{1}{2}\mathbf{R}(\boldsymbol{\theta}^\star|\mathcal{T})^T\mathbf{R}(\boldsymbol{\theta}^\star|\mathcal{T})$.

```
Require: $f$ to be continuously differentiable on $\Re^n$
Require: A close initial point $\boldsymbol{\theta}_0|\mathcal{T}$.
1: for $t = 0, 1, 2, \ldots,$ until convergence do
2:     Solve for $\Delta\boldsymbol{\theta}_t|\mathcal{T}$ in:                    ▷ Newton direction
           $\left[\mathbf{J}_f(\boldsymbol{\theta}_t|\mathcal{T})^T\mathbf{J}_f(\boldsymbol{\theta}_t|\mathcal{T}) + Z(\boldsymbol{\theta}_t|\mathcal{T})\right]\Delta\boldsymbol{\theta}_t|\mathcal{T}\cdots$
           $= -\mathbf{J}_f(\boldsymbol{\theta}_t|\mathcal{T})^T\mathbf{R}(\boldsymbol{\theta}_t|\mathcal{T})$   or       (7)
           $\Delta\boldsymbol{\theta}_t|\mathcal{T} = -\left[\mathbf{J}_f(\boldsymbol{\theta}_t|\mathcal{T})^T\mathbf{J}_f(\boldsymbol{\theta}_t|\mathcal{T})\cdots\right.$
           $\left.+Z(\boldsymbol{\theta}_t|\mathcal{T})\right]^{-1}\mathbf{J}_f(\boldsymbol{\theta}_t|\mathcal{T})^T\mathbf{R}(\boldsymbol{\theta}_t|\mathcal{T})$       (8)
3:     Update:
           $\boldsymbol{\theta}_{t+1}|\mathcal{T} = \boldsymbol{\theta}_t|\mathcal{T} + \Delta\boldsymbol{\theta}_t|\mathcal{T}$       (9)
4: end for
```

The method of Newton updates the value of a given variable, $\boldsymbol{\theta}|\mathcal{T}$, according to the direction and magnitude of the derivative of the function that characterizes such variable, $f(\boldsymbol{\theta}|\mathcal{T})$. The idea of using derivatives greatly rewarded the method with a fast rate of convergence. More precisely, Newton's method is known because it has a $q$-quadratic rate of convergence, finding a solution $\boldsymbol{\theta}^\bigstar|\mathcal{T}$ in very few iterations, minimizing $f(\boldsymbol{\theta}^\bigstar|\mathcal{T})$, if such a solution exists.

This method is also known for one of its main disadvantages: it is a local method. Therefore, one needs to have in advance a vector of parameters $\boldsymbol{\theta_0}|\mathcal{T}$ that is close to an acceptable solution; otherwise the Newton step $\boldsymbol{\Delta\theta_t}|\mathcal{T}$, though in the correct descent direction, could be very large and move far away from the solution. To overcome this difficulty one needs to consider using a globalization strategy, also known as “line search”. For the proposed model, the globalization strategy could use the following merit function:

$$M_f(\boldsymbol{\theta}|\mathcal{T}) = \frac{\boldsymbol{1}}{\boldsymbol{2}}||\mathbf{R}(\boldsymbol{\theta}|\mathcal{T})||_2^2. \tag{10}$$

Then, using the merit function (10) one can define the following property which establishes that there is a descent direction for the proposed function minimization problem.

**Property 1** *$\boldsymbol{\Delta\theta}|\mathcal{T}$ is a descent direction for $M_f(\boldsymbol{\theta}|\mathcal{T})$. That is, $\boldsymbol{\Delta\theta}|\mathcal{T} \le \mathbf{0}$ in the system given by*

$$\nabla M_f(\boldsymbol{\theta}|\mathcal{T})^{\boldsymbol{T}}\boldsymbol{\Delta\theta}|\mathcal{T} = -\boldsymbol{M_f}(\boldsymbol{\theta}|\mathcal{T}). \tag{11}$$

*Proof* Let $\nabla M_f(\boldsymbol{\theta}|\mathcal{T})$ be the gradient of the merit function (10) denoted as:

$$\nabla M_f(\boldsymbol{\theta}|\mathcal{T}) = \frac{\boldsymbol{1}}{\boldsymbol{2}}\mathbf{J}_{\boldsymbol{f}}(\boldsymbol{\theta}|\mathcal{T})^{\boldsymbol{T}}\mathbf{R}(\boldsymbol{\theta}|\mathcal{T}). \tag{12}$$

Then, substituting (10) and (12) into (11) results in

$$\frac{1}{2}\mathbf{R}(\boldsymbol{\theta}|\mathcal{T})^{\boldsymbol{T}}\mathbf{J}_{\boldsymbol{f}}(\boldsymbol{\theta}|\mathcal{T})\boldsymbol{\Delta\theta}|\mathcal{T} = -\frac{\boldsymbol{1}}{\boldsymbol{2}}||\mathbf{R}(\boldsymbol{\theta}|\mathcal{T})||_2^2, \tag{13}$$

which making use of algebraic operations reduces to

$$\Delta\boldsymbol{\theta}|\mathcal{T} = -\left[\mathbf{J}_f(\boldsymbol{\theta}|\mathcal{T})\right]^{-1}\mathbf{R}(\boldsymbol{\theta}|\mathcal{T}) \leq \mathbf{0}. \tag{14}$$

Hence, $\Delta\boldsymbol{\theta}|\mathcal{T} \leq \mathbf{0}$.

The next step is to establish a globalization strategy. Since the merit function (10) is a valid function that guarantees a descent at every iterate, the globalization strategy can be defined by the following property.

**Property 2** *If $\Delta\boldsymbol{\theta}|\mathcal{T}$ is a descent direction of $M_f(\boldsymbol{\theta}|\mathcal{T})$, then, there exists an $\beta_1 \in (0, 1]$ such that*

$$M_f(\boldsymbol{\theta}|\mathcal{T} + \boldsymbol{\beta_1}\Delta\boldsymbol{\theta}|\mathcal{T}) \leq \boldsymbol{M_f}(\boldsymbol{\theta}|\mathcal{T}) + \boldsymbol{\beta_1}\boldsymbol{\beta_2}\nabla \boldsymbol{M_f}(\boldsymbol{\theta}|\mathcal{T})^{\boldsymbol{T}}\Delta\boldsymbol{\theta}|\mathcal{T}, \tag{15}$$

*where $\beta_2$ is a fixed parameter that limits the speed of the descent.*

*Proof* The proof is given by Dennis et al. [11] (see Theorems 6.3.2. and 6.3.3. pp. 120–123).

Thus, substituting (10) into (15) results in the following:

$$\begin{aligned}\frac{1}{2}||\mathbf{R}(\boldsymbol{\theta}|\mathcal{T} + \boldsymbol{\beta_1}\Delta\boldsymbol{\theta}|\mathcal{T})||_2^2 &\leq \frac{\boldsymbol{1}}{\boldsymbol{2}}||\mathbf{R}(\boldsymbol{\theta}|\mathcal{T})||_2^2 \\ &+ \beta_1\beta_2\mathbf{R}(\boldsymbol{\theta}|\mathcal{T})^{\boldsymbol{T}}\mathbf{J}_f(\boldsymbol{\theta}|\mathcal{T})\Delta\boldsymbol{\theta}|\mathcal{T},\end{aligned} \tag{16}$$

which, using (14) on the right hand side and applying algebraic operations, reduces to

$$||\mathbf{R}(\boldsymbol{\theta}|\mathcal{T} + \boldsymbol{\beta_1}\Delta\boldsymbol{\theta}|\mathcal{T})||_2 \leq ||\mathbf{R}(\boldsymbol{\theta}|\mathcal{T})||_2 + \sqrt{\boldsymbol{1} - \boldsymbol{2}\boldsymbol{\beta_1}\boldsymbol{\beta_2}}, \tag{17}$$

where $\beta_2$ is a parameter controlling the speed of the line search. Typically $\beta_2 = 1 \times 10^{-4}$ [11].

Using the line-search globalization strategy, Newton's method can be modified to include a sufficient decrease condition (a.k.a. Armijo's condition). This condition is of extreme importance for a successful gradient descent (for a detailed explanation see [4]). The Globalized Newton method is as shown in Algorithm 2.

**Algorithm 2** Globalized Newton method to find $\boldsymbol{\theta}^\star|\mathcal{T}$ that minimizes $f(\boldsymbol{\theta}^\star|\mathcal{T}) = \frac{1}{2}\mathbf{R}(\boldsymbol{\theta}^\star|\mathcal{T})^T\mathbf{R}(\boldsymbol{\theta}^\star|\mathcal{T})$.

```
Require: F to be continuously differentiable on $\Re^n$
Require: An initial point $\boldsymbol{\theta}_0|\mathcal{T}$.
1: for $t = 0, 1, 2, \ldots,$ until convergence do
2:     Solve for $\Delta\boldsymbol{\theta}_t|\mathcal{T}$ in:                          ▷ Newton direction
           $\Delta\boldsymbol{\theta}_t|\mathcal{T} = -\left[\mathbf{J}_f(\boldsymbol{\theta}_t|\mathcal{T})^T\mathbf{J}_f(\boldsymbol{\theta}_t|\mathcal{T}) \cdots$
               $+Z(\boldsymbol{\theta}_t|\mathcal{T})\right]^{-1}\mathbf{J}_f(\boldsymbol{\theta}_t|\mathcal{T})^T\mathbf{R}(\boldsymbol{\theta}_t|\mathcal{T})$
3:     Sufficient decrease:                                         ▷ Armijo's condition
           Find $\beta_1$ that satisfies:
           $||\mathbf{R}(\boldsymbol{\theta}_t|\mathcal{T} + \beta_1\Delta\boldsymbol{\theta}_t|\mathcal{T})||_2 \leq ||\mathbf{R}(\boldsymbol{\theta}_t|\mathcal{T})||_2 \cdots$
               $+\sqrt{1 - 2\beta_1\beta_2}$
4:     Update:
           $\boldsymbol{\theta}_{t+1}|\mathcal{T} = \boldsymbol{\theta}_t|\mathcal{T} + \beta_1\Delta\boldsymbol{\theta}_t|\mathcal{T}$            (18)
5: end for
```

Notice the new update step (18) that considers the sufficient decrease condition. This will produce an acceptable step towards the solution parameters $\boldsymbol{\theta}_t^\star$. Section 5 shows how to find the hyper-parameters from the LP-SVR model introduced next.

## 3 LP-SVR formulation and hyper-parameters

This research aims to explore finding the hyper-parameters of a linear programming support vector regression (LP-SVR) approach, that uses an infeasible primal-dual interior point method to solve the optimization problem [3]. But in order to describe the LP-SVR formulation, it is necessary to depart from the $\ell_1$-SVR, having a training data set $\mathcal{T} = \{\mathbf{x}_i, y_i\}_{i=1}^N$ of $N$ samples of $M$-dimensional data, i.e., $\mathbf{x} \in \Re^M$, and a desired output class value $y \in \Re$. The formulation of an $\ell_1$-SVR (*i.e.* norm-1-SVR) problem is as follows:

$$\begin{array}{lc} \min\limits_{\boldsymbol{\alpha},\boldsymbol{\xi}} & ||\boldsymbol{\alpha}||_{\boldsymbol{1}} + \boldsymbol{2C}\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}}\boldsymbol{\xi_i} \\ \text{s.t.} & \begin{cases} y_j - \sum_{i=1}^N \alpha_i K(\mathbf{x}_j, \mathbf{x}_i) - b \leq \epsilon + \xi_j \\ \sum_{i=1}^N \alpha_i K(\mathbf{x}_j, \mathbf{x}_i) + b - y_j \leq \epsilon + \xi_j \\ \boldsymbol{\xi} \geq \mathbf{0} \end{cases} \\ \text{for} & j = 1, 2, \ldots, N, \end{array} \tag{19}$$

where $\boldsymbol{\alpha}$ is the vector of Lagrange multipliers associated with the support vectors (SVs); the summation in the cost function accounts for the $\epsilon$-insensitive training error, which forms a tube where the solution is allowed to be defined without penalization; $C > 0$ is a constant usually called "regularization parameter" that describes the trade off between the training error and the penalizing term $||\boldsymbol{\alpha}||_{\boldsymbol{1}}$; the variable $\xi_i$ is a nonnegative slack variable that describes the $\epsilon$-insensitive loss function; $y$ is the desired output in response to the input vector $\mathbf{x}$; the variable $b$ is a bias; $K(\cdot,\cdot)$ is any valid kernel function (see [10, 40]). The parameter vector $\boldsymbol{\alpha}$ and the bias $b$ are the unknowns and can take on any real value as long as the inequalities given in the constraints are satisfied.

Then, since the requirement of an LP-SVR (and of any linear programming problem) is to have the unknowns greater than or equal to zero, such variables are typically decomposed in their positive and negative parts. Therefore, one can denote $\boldsymbol{\alpha} = \boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-$, and $b = b^+ - b^-$. Then, in order to pose the problem as a linear programming problem in its canonical form and in order to use an interior point method solver, problem (19) must have no inequalities; thus, a slack variable $\mathbf{u}$ is required, which all-together results on the following problem:

$$\begin{array}{ll}\min\limits_{\boldsymbol{\alpha}^{\pm},\boldsymbol{b}^{\pm},\boldsymbol{\xi},\mathbf{u}} & \sum_{i=1}^{N}\left(\alpha_i^+ + \alpha_i^- + 2C\xi_i\right) \\ \text{s.t.} & \begin{cases} -\sum_{i=1}^{N}(\alpha_i^+ - \alpha_i^-)K(\mathbf{x}_j,\mathbf{x}_i)\cdots \\ \quad -b^+ + b^- - \xi_j + u_j = \epsilon - y_j \\ \sum_{i=1}^{N}(\alpha_i^+ - \alpha_i^-)K(\mathbf{x}_j,\mathbf{x}_i)\cdots \\ \quad +b^+ - b^- - \xi_j + u_j = \epsilon + y_j \\ \quad \alpha_j^+,\alpha_j^-,b^+,b^-,\xi_j,u_j \geq 0 \end{cases} \\ \text{for} & \quad j = 1,2,\cdots,N, \end{array} \tag{20}$$

which is the formulation used in the analysis presented in this paper, along with an interior point solver and a radial-basis function (RBF) kernel with parameter $\gamma$. Specifically, the RBF kernel used is of the form $e^{-\gamma\|\mathbf{x}_j-\mathbf{x}_i\|_2^2}$. The choice of the kernel was made because RBF kernels have demonstrated to be more robust in general cases, that is, they represent the best choice whenever the researcher does not have any (useful) prior information about the data used [54]. Using this kernel, the following equation defines the regression function for this problem:

$$F(\mathbf{x}) = \sum_{i=1}^{N}(\alpha_i^+ - \alpha_i^-)K(\mathbf{x},\mathbf{x}_i) + b^+ - b^-. \tag{21}$$

Note that (20) allows to define the following equalities:

$$\mathbf{A} = \begin{bmatrix} -\mathbf{K} & \mathbf{K} & -1 & 1 & -\mathbf{I} & \mathbf{I} \\ \mathbf{K} & -\mathbf{K} & 1 & -1 & -\mathbf{I} & \mathbf{I} \end{bmatrix}_{2N\times 4N+2}, \tag{22}$$

$$\mathbf{b} = \begin{bmatrix} \mathbf{1}\epsilon - \mathbf{y} \\ \mathbf{1}\epsilon + \mathbf{y} \end{bmatrix}_{2N\times 1}, \tag{23}$$

$$\mathbf{z} = [\boldsymbol{\alpha}^+ \quad \boldsymbol{\alpha}^- \quad b^+ \quad b^- \quad \boldsymbol{\xi} \quad \mathbf{u}]^T_{1\times 4N+2}, \tag{24}$$

$$\mathbf{c} = [\mathbf{1} \quad \mathbf{1} \quad 0 \quad 0 \quad \mathbf{2C} \quad \mathbf{0}]^T_{1\times 4N+2}, \tag{25}$$

which is an acceptable linear program of the form:

$$\begin{array}{ll}\min_{\mathbf{z}} & \mathbf{c}^T\mathbf{z} \\ \text{s.t.} & \begin{cases} \mathbf{Az} & = & \mathbf{b} \\ \mathbf{z} & \geq & \mathbf{0}. \end{cases}\end{array} \tag{26}$$

Note that this problem has $(4N+2)$ variables and $2N$ constraints.

This is a more suitable definition than the one described in late 2009 by Lu et.al. [37] for interior point methods; and also it is an extension of the LP-SVM work presented in early 2009 by Torii et al. [61] and in early 2010 by Zhang [70]. The main advantage of the proposed LP-SVR is a sparser solution, meaning fewer support vectors, a more efficient computation of the optimization problem, and a very fast rate of convergence. However, note that since the LP-SVR definition suffers from an increase in dimensionality, the approach presented in [55] is recommended (and used here) for large scale problems.

## 4 Model hyper-parameters selection criteria: error functions

In machine learning one typically wants to minimize the true generalization error, implying that the deviation from the expected output has to be measured in some way. Indeed, the measurement of the generalization error is model-dependent [6], suggesting that one machine learning problem may have a particular method that performs better than others, and a different problem may have a different method that also works well; however, cross-validation-based methods will perform well in the average case, as discussed earlier. Nevertheless, cross-validation-based and other methods for the estimation of the true generalization error go together with error functions that measure and inform the researcher on what basis the model is "good" or "bad". The method proposed in this research allows for any error function to be used, as long as it is relevant or useful for the application subject of research. As an example of how the proposed model works, the following paragraphs discuss error metrics chosen particularly for classification and regression problems based on the LP-SVR model described in (20).

### 4.1 Multi-class and two-class problems

To particularize, this research aims to estimate the model vector of hyper-parameters $\boldsymbol{\theta} = [\boldsymbol{C}, \gamma]$. The error functions proposed for multi-class problems are two: a modified estimate of scaled error rate (ESER) and the balanced error rate (BER) [6]. The ESER metric is given by

$$r_1(\boldsymbol{\theta}|\mathcal{T}) = \sum_{i=1}^{N}\zeta\Psi\{(F(\mathbf{x}_i) - y_i) - 0.5\}, \tag{27}$$

where $F(\mathbf{x})$ is the actual output of the LP-SVR classifier when the input vector $\mathbf{x}$ is presented as its input, as in (21); $\zeta$ is a scaling factor used only to match the ESER to a desired range of of values; and the $\Psi\{\cdot\}$ function is given by

$$\Psi\{x\} = \frac{1}{1+e^{-\tau x}}, \tag{28}$$

which is an approximation to the well known ideal unit step function. The quality of the approximation is given by the parameter $\tau$ as illustrated in Fig. 2. Throughout this research $\zeta$ is fixed to $\frac{1}{N}$. If $\zeta = \frac{1}{N}$, then $r_1$ has values only within the interval $r_1 \in [0,1]$ which is desirable. The smaller the ESER is, the better the hyper-parameters are.

In some special cases, the ESER may become biased towards false positive counts, especially if there is a large

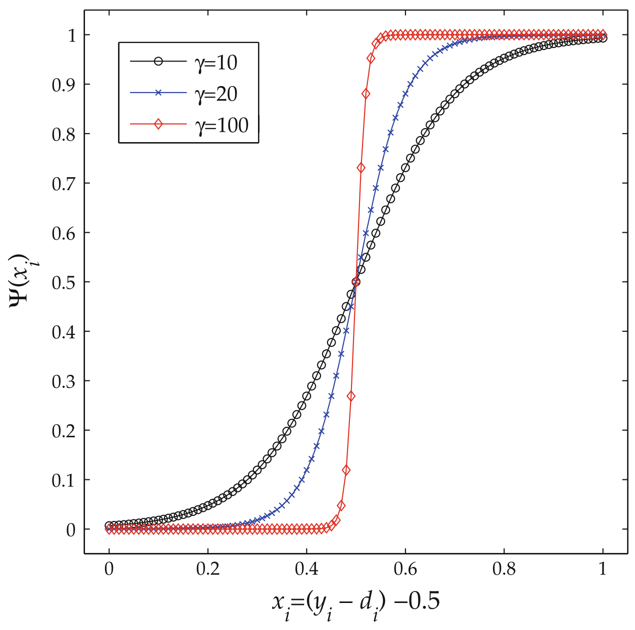


**Fig. 2** Unit step function approximation. A function $\Psi\{\cdot\}$ was used for convenience in easing computations

number of unbalanced class examples. To alleviate the problem, the BER is used, which is defined as follows:

$$r_2(\boldsymbol{\theta}|\mathcal{T}) = \frac{1}{2}\left(\frac{\sum FP}{\sum TN + \sum FP} + \frac{\sum FN}{\sum FN + \sum TP}\right), \quad (29)$$

where *TP* stands for "True Positive," *FP* "False Positive," *TN* "True Negative," and *FN* "False Negative." The domain of the BER falls within the interval $r_2 \in [0, 1]$, having that a small BER represents a good set of hyper-parameters.

Clearly, the BER meets the classical misclassification rate if there are equal number of positive and negative examples, that is the case in which $\sum TN + \sum FP = \sum FN + \sum TP$ (see [6]).

In a two class approach, it is more convenient to use the area under the receiver operating characteristic (ROC) curve, as well as the BER metric. It is well known that maximizing the area under the ROC curve (AUC) leads to better classifiers, and therefore, it is desirable to find ways to maximize the AUC during the training steps of supervised classifiers [13]. The AUC is estimated by means of adding successive areas of trapezoids. For a complete treatment of the ROC concept and AUC algorithm, please consult [13]. Let us define the function $r_1$ for the two class approach as follows:

$$r_1(\boldsymbol{\theta}|\mathcal{T}) = 1 - AUC(\cdot)_{\boldsymbol{\theta},\mathcal{T}}, \quad (30)$$

where the $AUC(\cdot)$ is computed using Algorithms 1, 2, and 3 from [13]; and its interval is $r_1 \in [0, 1]$. Let us recall that, essentially, it is desired to have an $r_1(\boldsymbol{\theta}|\mathcal{T}) = \boldsymbol{0}$, which evidently in (30) means a maximization of the AUC. The function $r_2$ for the two-class approach is the same BER as in (29).

The metrics for regression problems are quite different, as explained in the following section.

### 4.2 Regression problems

In regression, different measures of error are required to make a significant and meaningful progress towards a solution. The typical measure of error in regression problems is the mean squared error (MSE). However, for the sake of investigating other options, this research uses two different error functions for regression: sum of squared error (SSE) [6], and a statistics-based metric, which from hereafter will be referred to as "STAT" (this is not an acronym, but a name that the authors made up for a *STATistics*-based metric). Note that there is no actual need to have two metrics, indeed the problem could be solved with only one, nonetheless, this research uses two error functions to show that the algorithm performs well in complicated scenarios, in which two or more functions are used. The STAT function is an original contribution of the authors of this research; and the SSE metric, however, is given by

$$r_1(\boldsymbol{\theta}|\mathcal{T}) = \sum_{i=1}^{N}(F(\mathbf{x}_i) - y_i)^2, \quad (31)$$

where $F(\mathbf{x})$ is the actual output of the classifier LP-SVR when the input vector $\mathbf{x}$ is presented at its input. The domain of the SSE is in the interval $r_1 \in [0, \Re^+]$, and the desired response is as near as zero as possible.

The second metric is based on the statistical properties of the residual error given by the difference $F(\mathbf{x}_i) - y_i$. From estimation theory it is known that if the expected value of the residual error is equal to zero and the variance is unitary, we have achieved the ordinary least squares solution to the regression problem, assuming that the training set has zero mean and unit variance [60]. Furthermore, it is also understood that as the variance of the residual error approaches zero, the regression problem is better solved [29]. Let us denote the expected value of the residual error as

$$\mu = E[F(\mathbf{x}_i) - y_i] = \frac{1}{N}\sum_{i=1}^{N} F(\mathbf{x}_i) - y_i, \quad (32)$$

and the variance of the residual error as follows:

$$\sigma^2 = E[F(\mathbf{x}_i) - y_i - \mu]^2 = \frac{1}{N-1}\sum_{i=1}^{N}(F(\mathbf{x}_i) - y_i - \mu)^2, \quad (33)$$

from where it is desired that $\mu, \sigma^2 \to 0$. Hence, the second error metric is defined as

$$r_2(\boldsymbol{\theta}|\mathcal{T}) = \boldsymbol{\sigma}^2 + \sqrt{\boldsymbol{\mu}^2}, \tag{34}$$

where the term $\sqrt{\mu^2}$ has the meaning of the absolute value of the mean, and was chosen for convenience since $|\mu| \equiv \sqrt{\mu^2}$ is easier to handle in optimization problems. The domain of STAT falls within the interval $r_2 \in [0, \Re^+]$.

Note that when the two functions are used, the SSE metric is likely to weight more during the early iterations of the optimization process, this is mainly because its scalar value would be very large; however, as the method comes closer to the solution, the SSE metric is likely to have small values and the STAT metric is likely to weight more towards the minimization of the function, refining the current solution which was obtained under the lead (the weighing) of the SSE.

## 5 Implementation and considerations

This section takes the general method presented in Sect. 2 and particularizes it to the hyper-parameters of the LP-SVR model presented in Sect. 3, using the generalization error metrics discussed in Sect. 4 to find a good set of hyper-parameters. This particularization or implementation is discussed next.

### 5.1 Nonlinear least squares quasi-Newton implementation

Particularizing (1) for the cases presented in Sects. 3–4, the formulation of $\mathbf{F}$ simply becomes

$$\mathbf{R}(\boldsymbol{\theta}|\mathcal{T}) = \begin{bmatrix} \boldsymbol{r_1}(\boldsymbol{\theta}|\mathcal{T}) \\ \boldsymbol{r_2}(\boldsymbol{\theta}|\mathcal{T}) \end{bmatrix}_{2\times 1}, \tag{35}$$

where clearly $\mathbf{R}: \Re^2 \mapsto \Re^2$ since $\boldsymbol{\theta} \in \Re^2$. The typical challenge is to compute (3), the Jacobian matrix $\mathbf{J}_f(\boldsymbol{\theta}|\mathcal{T})$, since not all the error functions are analytically differentiable, i.e. (29) or (30). Then, the classical approaches are to estimate $\mathbf{J}_f(\boldsymbol{\theta}|\mathcal{T})$ via finite difference approximation, or secant approximation. For its robustness and precision over the secant method [11], this research uses the finite difference approximation. In this case, $\widetilde{\mathbf{J}}_f(\boldsymbol{\theta}|\mathcal{T})$ corresponds to a finite difference derivate approximation which solves (3) using (4) where

$$\frac{\partial r_n}{\partial \theta_1} \approx \frac{r_n(\theta_1 + h, \theta_2) - r_n(\theta_1, \theta_2)}{h}, \tag{36}$$

$$\frac{\partial r_n}{\partial \theta_2} \approx \frac{r_n(\theta_1, \theta_2 + h) - r_n(\theta_1, \theta_2)}{h}, \tag{37}$$

allowing $h$ to be sufficiently small, as appropriate.

### 5.2 Possible upper bounds and an initial set of hyper-parameters

Although the globalization strategy presented in Algorithm 2 prevents Newton method from going far away from a solution, and though it guarantees a decrease of the generalization error at each iteration, its success relies upon a "good" initial set of hyper-parameters. A common approach to find a "good" initial set of hyper-parameters consists of varying $C$ and $\gamma$ in a grid of equally spaced intervals and observing the pair of hyper-parameters that is producing the minimum error. Typically, this is achieved by varying $C$ in the interval $C \in \{2^{-5}, 2^{-3}, \ldots, 2^{13}, 2^{15}\}$ and $\gamma \in \{2^{-7}, 2^{-6}, \ldots, 2^{1}, 2^{2}\}$; this method is known as "grid search" [1, 12, 45]. This approach, in spite of being very powerful to find a good starting set of hyper-parameters, requires a loop of 110 iterations (for the example shown above), which is very costly, specially in large machine learning problems. This problem gave the opportunity for researchers to work in ways to reduce the number of iterations required.

Recent findings by Ortiz et al. [45], show that it is possible to establish upper bounds to the search space of the hyper-parameters; evidently, with a reduction of the search space, the total search time may drop considerably when the data set is large. Ortiz et al., studied the relationship and relevancy of the hyper-parameters of a traditional SVR model, and although the proposed research works with a different SVR model, i.e. an LP-SVR, their findings can also be extended to this research with the next minor adaptations. The following equation defines the upper bound for $C$:

$$C \le \frac{y_i^{\max} - 2y_i^{\min} - \epsilon}{1 - \frac{1}{N-1}\sum_{j=1,\, j\neq i}^{N} K(\mathbf{x}_j, \mathbf{x}_i)}, \tag{38}$$

where the input vector $\mathbf{x}_i$ is set to the input vector corresponding to the highest output $y_i^{\max}$; $y_i^{\min}$ corresponds to the lowest output; and $\epsilon$ corresponds to the loss function parameter used in problem (20).

Similarly, the following equation defines the upper bound for $\gamma$:

$$\gamma \le -\frac{\ln(0.001)}{\left(\frac{1}{N}\sum_{j=1}^{N} \min_{j,i\neq j} d(\mathbf{x}_j, \mathbf{x}_i)\right)^2}, \tag{39}$$

where $\min_{i\neq j} d(\mathbf{x}_j, \mathbf{x}_i)$ is the minimum distance between two vectors in the training set. The denominator of (39) can be interpreted as the average of the minimum distances between the vectors of the training set, while the numerator is related to the amplitude of a Gaussian function that has negligible influence in the SVs [45]. Both, the upper bound for $C$ in (38) and for $\gamma$ in (39), restrict the search space and may also reduce the time of the grid search strategy; therefore, this could be a good alternative when a grid-search method is used or when constrained optimization problems are used; however, since this research uses an unconstrained optimization method *i.e.*, a nonlinear least squares method with a quasi-Newton implementation, this bound may not be helpful, and, as it will be shown later, it does not perform well for all problems. Thus, other alternatives were indeed explored.

In 2003, Cherkassky et al. [8] developed a method called "Practical Selection" that estimated the hyper-parameters of an SVR. The authors used statistical properties of the training set to provide an educated guess for each of the hyper-parameters. Given the simplicity of the estimation and since no iterative process is required, this research uses Cherkassky's estimates as initial hyper-parameters for all the proposed algorithms. The following equation defines the initial value for the regularization parameter $C$:

$$C = \max\left\{|E[y] + 3\sigma_y|, |E[y] - 3\sigma_y|\right\}, \tag{40}$$

where $E[y]$ is the expected value of the desired output $y$ of the training data set; and $\sigma_y$ is the standard deviation of $y$. On the other hand, the kernel parameter $\gamma = \frac{1}{2\sigma^2}$ can be computed using the following definition [8, 26, 27]:

$$\sigma = 0.3(\mathbf{x}_{\max} - \mathbf{x}_{\min}), \tag{41}$$

where $\mathbf{x}_{\max}$ and $\mathbf{x}_{\min}$ are maximum and minimum values of the training set input values respectively; which is equivalent to $\sigma = 0.3 * range(\mathbf{x})$.

As it can be seen, the estimation of the hyper-parameters using (40) and (41) is fairly simple and does not involve any kind of iterative process or SVR training. Therefore, this research suggest the usage of both (40) and (41) to estimate the initial values for $C$ and $\gamma$ respectively. With respect to the parameter $\epsilon$, the research presented in this paper opted to leave $\epsilon$ as a free parameter. This parameter corresponds to the loss function and controls the width of the $\epsilon$-tube; the authors of this research are convinced that this parameter has much to do with the tolerance or over-fit that the researcher wants to allow in every particular problem, i.e., $\epsilon$ is directly dependent of, and unique to, every problem and application; and ultimately it has been demonstrated that the most relevant parameter is $C$ [45]; however, the fact that a poor selection of $\epsilon$ impacts the value of $C$, obligates the researcher to observe carefully how the value of $\epsilon$ is used. Therefore, if the reader desires to estimate an initial $\epsilon$ parameter for the loss function, it can also be obtained with the following equation [8]:

$$\epsilon = 3\sigma_\epsilon\sqrt{\frac{\ln N}{N}}, \tag{42}$$

where $N$ is the number of samples and $\sigma_\epsilon$ can be estimated from the following equality:

$$\sigma_\epsilon^2 = \frac{N}{N-d}\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \tag{43}$$

where $d$ is the degree of a polynomial that is trying to fit $y$, and $\hat{y}$ is the output given by the polynomial function of degree $d$.

## 5.3 $\mathcal{K}$-fold cross validation

Every time (1), or particularly (35), are evaluated, true generalization error estimates become necessary for the success of the hyper-parameters estimation. "True generalization error estimates" refers to training the LP-SVR, observing the actual output and comparing it to the desired output obtaining an amount of error that ideally should be as close as possible to the amount of error obtained using a test set.

Estimating the true generalization error given a training set $\mathcal{T}$ is not a trivial task. The best two approaches to solve this problem are: $\mathcal{K}$-fold and leave-one-out (LOO) cross validation. The former is the best for moderate to large-scale applications, while the latter is excellent for very small applications since it is known to be close to an unbiased estimator [58].

This research uses the $\mathcal{K}$-fold cross validation approach for all the experiments that involve the final algorithm. In this approach, the training data set is divided in a $|\mathcal{K}|$ number of slices or folds. Then it follows to define a rule to find a reasonable number of partitions in $\mathcal{K}$, that is:

$$|\mathcal{K}| = \max\left\{5, \left\lfloor\frac{N}{N-q}\right\rfloor\right\}, \tag{44}$$

where $q$ represents the maximum number of samples that make (26) computationally tractable with traditional training methods; the function $\lfloor\cdot\rfloor$ represents a round down operation; and $|\mathcal{K}|$ is the number of slices in $\mathcal{K}$. This equation can be justified in two ways: (i) when the problem is, computationally-speaking, small, a 5-fold cross validation suffices and it is proven to be one of the best estimators [58]; (ii) with a large-scale problem, it makes sense to use large-scale SVR training methods which usually require cutting the problem in pieces that are computationally tractable [54], therefore, using a similar approach, this research proposes to increase the number of slices (or folds) in relation to the number of samples that are computationally tractable with a conventional SVR training method. It is known [58] that as the number of slices or folds increase, the less biased the error estimation becomes. E.g., consider the LOO cross validation which is the case when $|\mathcal{K}| = N$. Thus, the main advantage of using (44) is a better estimation of the true error and a possible reduction of the total training time; while at the same time, for small problems, Eq. (44) chooses the minimum best number of slices i.e., five.

The slices in the set $\mathcal{K}$ are denoted as

$$\mathcal{K} = \{s_1, s_2, \ldots, s_k\}, \tag{45}$$

where $k = |\mathcal{K}|$, and $s_k$ denotes the $k$-th slice and contains the indexes of those training data points in $\mathcal{T} = \{\mathbf{x}_i, y_i\}_{i=1}^{N}$. Therefore it is clear that $s_k \subset \{1, 2, \ldots, N\}$ and $\mathcal{K} \subseteqq \{1, 2, \ldots, N\}$.

The general concept is to partition the training set $\mathcal{T}$ in $|\mathcal{K}|$ groups of data (ideally of equal size), then train the LP-SVR classifier with $|\mathcal{K}| - 1$ and use the remaining data as validation set. The process is repeated for all the partitions $s_k$ and the error is averaged as follows

$$\widetilde{\mathbf{R}}(\boldsymbol{\theta}|\mathcal{T}_{\mathcal{K}}) = \frac{\mathit{1}}{|\mathcal{K}|}\sum_{k=\mathit{1}}^{|\mathcal{K}|} \mathbf{R}(\boldsymbol{\theta}|\mathcal{T}_{s_k}), \tag{46}$$

where $\mathbf{R}(\boldsymbol{\theta}|\mathcal{T}_{s_k})$ is the error obtained for the $k$-th portion; $\widetilde{\mathbf{R}}(\boldsymbol{\theta}|\mathcal{T}_{\mathcal{K}})$ is an estimate of the true generalization error; $\mathcal{T}_{s_k} = \{\mathbf{x}_i, y_i\}_{i \in s_k}$ and $\mathcal{T}_{\mathcal{K}} = \{\mathbf{x}_i, y_i\}_{i \in \mathcal{K}}$.

## 5.4 Final algorithm

**Algorithm 3** Nonlinear least squares quasi-Newton method to minimize $\frac{1}{2}||\mathbf{R}(\boldsymbol{\theta}^{\star}|\mathcal{T}_{\mathcal{K}})||_2^2$ and find the hyper-parameters $\boldsymbol{\theta}^{\star}|\mathcal{T}_{\mathcal{K}}$ for an LP-SVR model.

```
Require: Cross validation indexes $\mathcal{K}$.                          ▷ Sec. 5.3.
Require: Training set $\mathcal{T}$.
Require: Initial parameters $\boldsymbol{\theta}_0 = [C_0, \gamma_0]$.     ▷ Uses (40) and (41) respectively.
 1: Initial point $\boldsymbol{\theta}_0|\mathcal{T}_{\mathcal{K}}$.        ▷ Iteratively. Sec. 5.2.
 2: for $t = 0, 1, 2, \ldots,$ until convergence do
 3:     Solving for $\Delta\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}}$  ▷ Sec. 5.1, and 5.3.
        a) $\mu_\lambda = \min\ \mathrm{eig}\left[\widetilde{\mathbf{J}}_{\mathbf{F}}(\boldsymbol{\theta}_t|\mathcal{T}_S)\right]$
        b) Solution to the linear system
        if $\mu_\lambda > 0$ then
            $\Delta\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}} = -\left[\widetilde{\mathbf{J}}_f(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}})^T\widetilde{\mathbf{J}}_f(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}})\right]^{-1}\cdots$
            $\widetilde{\mathbf{J}}_f(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}})^T\widetilde{\mathbf{R}}(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}})$      (47)
        else
            $\Delta\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}} = -\left[\widetilde{\mathbf{J}}_f(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}})^T\widetilde{\mathbf{J}}_f(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}})\cdots\right.$
            $\left.+(\mu_\lambda+\delta)\,\mathbf{I}\right]^{-1}\widetilde{\mathbf{J}}_f(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}})^T\widetilde{\mathbf{R}}(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}})$      (48)
        end
 4:     Sufficient decrease                                                ▷ Armijo's condition.
        Find $\beta_1$ that satisfies:
            $||\widetilde{\mathbf{R}}(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}} + \beta_1\Delta\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}})||_2 \le \cdots$
            $||\widetilde{\mathbf{R}}(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}})||_2 + \sqrt{1-2\beta_1\beta_2}$
 5:     Update:
            $\boldsymbol{\theta}_{t+1}|\mathcal{T}_{\mathcal{K}} = \boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}} + \beta_1\Delta\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}}$      (49)
 6: end for
Ensure: Hyper-parameters estimate $\widetilde{\boldsymbol{\theta}}^{\star} = \boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}}$.
```

Algorithm 3 shows the final methodology after certain considerations that are explained here. To review this final algorithm let us proceed to briefly revisiting each step. The inputs of the algorithm are (i) the indexes $\mathcal{K}$ corresponding to the cross validation indexes and (ii) the training set $\mathcal{T}$ from which the LP-SVR hyper-parameters $\boldsymbol{\theta}^{\star}$ are to be estimated. The notation that uses a "star", e.g., $\boldsymbol{\theta}^{\star}$, is used to denote a solution, while the notation with a "star and tilde", e.g., $\widetilde{\boldsymbol{\theta}}^{\star}$, means a very good approximation to a solution using an inexact method, which in this case corresponds to a quasi-Newton method that uses an finite-differences estimate of the partial derivatives in the Jacobian matrix. In Step 1, the algorithm requires a set of initial parameters $\boldsymbol{\theta}_{\mathit{0}} = [\boldsymbol{C}_{\mathit{0}}, \boldsymbol{\gamma}_{\mathit{0}}]$; these are given by (40) and (41) respectively. In Step 3, the algorithm proceeds using the approximation to the true Jacobian (35)–(37) as shown in Sect. 5.1. This is the most expensive part of the algorithm since every single function call of (35) requires cross validation, as explained in Sect. 5.3. However, by "recycling" previously computed partial derivatives, all unnecessary additional computations could be avoided. That said, notice that the linear system was expected to be

$$\begin{aligned}\Delta\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}} = -\Big[&\widetilde{\mathbf{J}}_f(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}})^{\boldsymbol{T}}\widetilde{\mathbf{J}}_f(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}}) \\ &+ Z(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}})\Big]^{-\mathit{1}}\widetilde{\mathbf{J}}_f(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}})^{\boldsymbol{T}}\widetilde{\mathbf{R}}(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}}),\end{aligned} \tag{50}$$

which follows from Algorithm 2, but instead it now is

$$\begin{aligned}\Delta\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}} = -\Big[&\widetilde{\mathbf{J}}_f(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}})^{\boldsymbol{T}}\widetilde{\mathbf{J}}_f(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}}). \\ &+(\mu_\lambda + \delta)\mathbf{I}\Big]^{-1}\widetilde{\mathbf{J}}_f(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}})^{\boldsymbol{T}}\widetilde{\mathbf{R}}(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}}).\end{aligned}$$

This change is justified in the following manner. In (50), the term $Z(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}})$ corresponds to the second order information shown in (6). But in most cases, second derivatives are not available or are expensive to obtain and implausible, if not impossible. Fortunately, the impact of $Z$ in the equation is almost negligible and the community allows its removal because the consensus is that the Jacobian provides the most relevant information [44]. That is,

$$\mathbf{J}_f(\boldsymbol{\theta}|\mathcal{T})^{\boldsymbol{T}}\mathbf{J}_f(\boldsymbol{\theta}|\mathcal{T}) + \mathbf{Z}(\boldsymbol{\theta}|\mathcal{T}) \approx \mathbf{J}_f(\boldsymbol{\theta}|\mathcal{T})^{\boldsymbol{T}}\mathbf{J}_f(\boldsymbol{\theta}|\mathcal{T}), \tag{51}$$

and therefore (48) drops out Z. This is expressed as in (47) and is known as the G'auss-Newton method; for more details see [11]. Since one of the problems of this method is when the Jacobian is ill-posed, this research adds a constant that is proportional to the minimum eigenvalue of the Jacobian to itself in order to alleviate the issue. Thus, the algorithm includes the following verification procedure (Step 3.a): if the minimum eigenvalue, $\mu_\lambda$, is strictly positive, then, proceed with the typical Gauss-Newton linear system (47); otherwise, add a constant proportional to the minimum eigenvalue and then, solve the linear system (48). The constant $\delta > 0$ is sufficiently small so that it cannot be interpreted as zero, and $\mathbf{I}$ is the identity matrix of identical size to the Jacobian. In (48) the typical choice is $\delta = 1 \times 10^{-8}$. Finally, in the process of matrix inversion this research suggests the use of the well known direct approach called

*LU*-factorization [44] for all large-scale cases; or also an indirect approach such as the classic conjugate gradient algorithm by Hestennes [21] may be used as well.

The remaining steps are the Armijo's condition and update, which did not suffer any changes. At "the end", the final algorithm returns the set of estimated hyper-parameters, $\widetilde{\boldsymbol{\theta}}^{\star}$, that minimize the generalization error with $\mathcal{K}$-fold cross validation. The term "the end" of the algorithm here means that the algorithm stops and returns a solution. For this reason, this paper defines four options to stop the algorithm. The first option is the natural stop, when it is said that the algorithm "converged", which in this case is the fulfillment of the following inequality:

$$||\widetilde{\mathbf{R}}(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}})||_2 \leq \varepsilon_1, \tag{52}$$

where $\varepsilon_1$ is chosen by the researcher. Ideally the left side term of (52) would be zero, or would use $\varepsilon_1 = 1 \times 10^{-4}$, or another value sufficiently small. In practice, it is rare that the problem is minimized to zero; however, in any linearly-separable or non-linearly separable problems (speaking of classification) a zero residual can be achieved, but not so in regression, unless the problem is trivial. Clearly, the minimum of (2) is also the minimum of the left side of (52). Then, another option to stop the algorithm is to monitor every set of hyper-parameters produced at each iteration and measure its change between iterations as follows:

$$||\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t||_2 \leq \varepsilon_2, \tag{53}$$

where $\varepsilon_2$ is typically set to a very small value; this parameter is given by the researcher. Condition (53) states a stopping criteria if the algorithm has no variability in terms of the updates at each iteration. However, it may happen that the algorithm is indeed changing the solution $\boldsymbol{\theta}_t$ at each iterate, but indeed this represent no significant progress towards a solution. In such case, the following stopping criteria is used:

$$\left| ||\widetilde{\mathbf{R}}(\boldsymbol{\theta}_t|\mathcal{T}_{\mathcal{K}})||_2 - ||\widetilde{\mathbf{R}}(\boldsymbol{\theta}_{t+1}|\mathcal{T}_{\mathcal{K}})||_2 \right| \leq \varepsilon_3, \tag{54}$$

where $\varepsilon_3$ is sufficiently small. Therefore, even though the algorithm makes progress updating its hyper-parameters, if the new set does not produce a sufficiently "better" set of hyper-parameters, then it should stop. Finally, the last stopping criteria monitors a number of maximum iterations. The criteria is simply

$$t \leq \varepsilon_4, \tag{55}$$

where $\varepsilon_4$ is the maximum number of iterations permitted; this parameter is also given by the researcher. Every detail explained up to this point completes the analytical and numerical considerations of the algorithm; the next section introduces the experiments conducted over different data sets and analyzes the results obtained.

## 6 Experimental results

To show the effectiveness and efficiency of the proposed hyper-parameters selection method, this research carried simulations over different data sets. The summary of the properties of these data sets are shown in Table 1. The next section presents a brief description of the data sets; and then follows a discussion of the methodology followed in this research.

### 6.1 Data sets

The first data set is the well-known *Ripley* problem [46, 52] which consists of two classes where the data for each class have been generated by a mixture of two Gaussian distributions. The *Sonar* data set corresponds to the classification of sonar returns for two targets under the sea level: a metal cylinder and a similarly shaped rock [17]. The *Wine* data set [15, 31] contains results of wine chemical analysis within the Italy region but was derived from three different vines. The analysis consists of 13 attributes of two different groups of wine. This data set is also part of the UCI machine-learning repository [16]. The *Spiral* data set is a

**Table 1** Summary of the dimensions and properties of the data sets used for the proposed experiments. The simulations include classification and regression examples

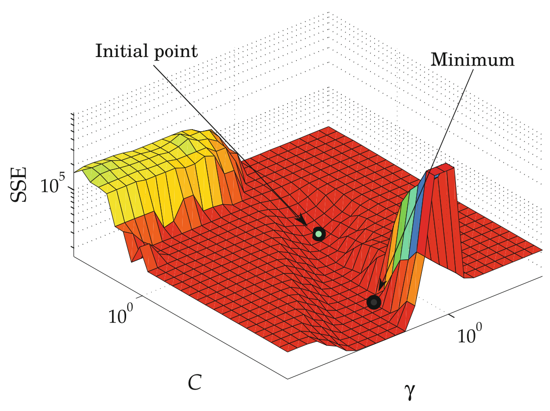
| Data set | Classes | Features $M$ | Training $N$ | Testing $N^*$ | References |
|---|---|---|---|---|---|
| Ripley | 2 | 2 | 250 | 1,000 | [46, 52] |
| Sonar | 2 | 60 | 104 | 104 | [17] |
| Wine | 2 | 13 | 110 | 20 | [15, 31] |
| Spiral | 2 | 2 | 200 | 101 | [34, 67] |
| ADA | 2 | 48 | 4,147 | 415 | [33] |
| GINA | 2 | 970 | 3,153 | 315 | [35] |
| HIVA | 2 | 1,617 | 3,845 | 384 | [6] |
| NOVA | 2 | 16,969 | 1,754 | 175 | [16, 24] |
| SYLVA | 2 | 216 | 13,086 | 1,308 | [5, 9] |
| Iris | 3 | 4 | 130 | 20 | [14, 18] |
| MODIS | 4 | 4 | 374,566 | 85 + mil | [53] |
| SincF | $\Re$ | 1 | 200 | 200 | [47] |
| LoadP | $\Re$ | 8 | 35,064 | 8,784 | [28] |
| MortPollution | $\Re$ | 15 | 48 | 12 | [39] |
| Bodyfat | $\Re$ | 13 | 201 | 52 | [48] |
| Betaplasma | $\Re$ | 12 | 252 | 63 | [43] |
| Retplasma | $\Re$ | 12 | 252 | 63 | [43] |
| Autompg | $\Re$ | 7 | 313 | 79 | [50] |
| Housing | $\Re$ | 13 | 404 | 102 | [50] |
| Concrete | $\Re$ | 16 | 824 | 206 | [68] |
| Abalone | $\Re$ | 8 | 3,341 | 836 | [64] |

synthetic data set that consists of a two class problem with an extremely non-linear decision surface and is typically used to test the ability of classifiers in finding such difficult decision functions [67]. *ADA* is a marketing- related data set [33]. The goal of ADA is to discover high revenue people from census data. This is a two-class classification problem. The raw data from the census bureau is also known as the Adult database [25, 49] in the UCI machine-learning repository [16]. *GINA* is a digit recognition-related data set that is commonly known as the MNIST database of handwritten digits [35]. GINA aims to provide features for handwritten digit recognition [9, 49]. *HIVA* is a data set related to HIV infections. The goal of HIVA is to provide features for prediction of active compounds within an AIDS HIV infection [6]. *NOVA* is a text classification data set. The data of NOVA comes from the UCI repository [16] which is also known as Twenty-Newsgroup data set [24]. *SYLVA* is an ecology-related data set that is also part of the UCI repository [16] under the name of covertype data set [5, 9]. The SYLVA data set aims to provide features for forest cover type classification. The *Iris* data set is perhaps the best known database to be found in the machine learning literature and is also part of the UCI data set [16]. The data set contains three classes of 50 instances, where each class refers to a type of Iris plant [14, 18]. The *MODIS* data set was obtained from a dust storm detection project developed by the principal author of this paper while at NASA Goddard Space Flight Center [53].

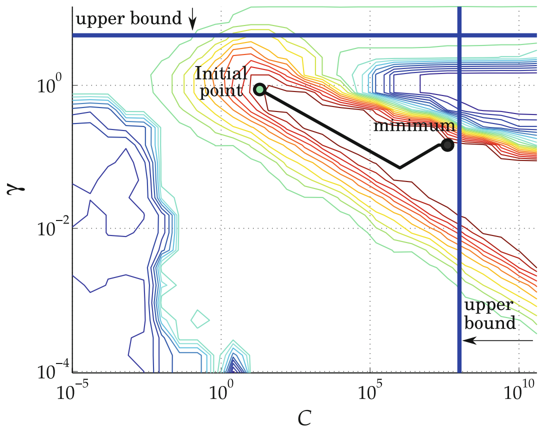
The *SincF* data set aims to fit the "sinc" function, which is a typical function to approximate [47]. The *SincF* data set consists of unevenly sampled points from the sinc function, $f(x) = \text{sinc}(x)$. The *LoadP* data set is a relatively new data set extracted from New England's power network data [28, 54]; it consists of different physical measurements used to predict the electric power load for any given hour of the day. *MortPollution* uses pollution data to analyze the mortality rate in relationship with the age [39]. *Bodyfat* determines the amount of fat in the body based on a number of different measurements [48]. *Betaplasma* and *Retplasma* analyze the risk of developing cancer using factors such as plasma and retinol concentration [43]. The *Autompg* data set studies the fuel consumption of a vehicle within a city considering different factors; and the *Housing* data set deals with the value of property in the Boston area [50]. *Concrete* aims to model the compressive strength of the concrete material based on the components mixture used for that particular kind of concrete [68]. And finally, the *Abalone* data set models the age of abalones given their physical properties [64]. Note that although Abalone is a multi-class problem, it will be addressed as a regression problem in order to compare results with [45]. Such results will be explained in the next paragraphs.

## 6.2 Methodology and results

The data sets were divided in the amounts indicated in Table 1; the training samples were selected in the exact order as they appear in the original data sets. The initial values for the hyper-parameters and the upper bounds were estimated from the training set $\mathcal{T}$. In most instances the initial values were close to the solution and reduced the number of iterations; similarly, the upper bounds in most cases contained a global minimum. To exemplify this situation, Fig. 3b depicts the contour plot produced by metric SSE for the *Abalone* data set; the figure shows the initial point and upper bounds obtained with the methods discussed in Sect. 5.2 and the global minimum found with the proposed algorithm. In this case, the upper bounds contain the global minimum; however, consider the case of Fig. 4b which shows a contour plot of the STAT error measure for

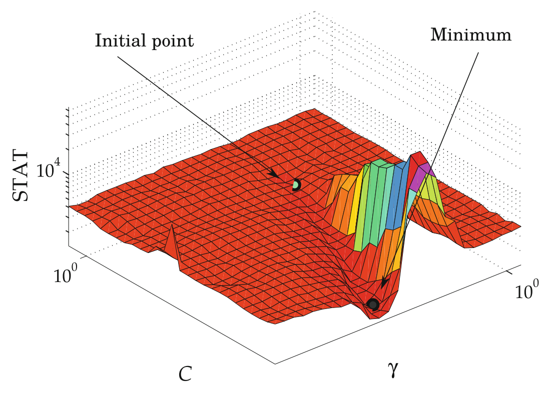


**(a)** Error surface with the SSE metric.

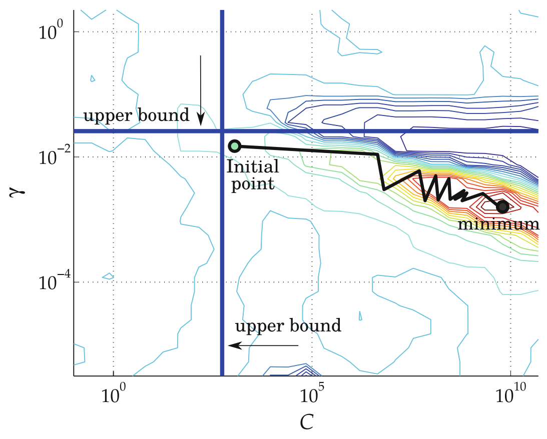


**(b)** Contour plot of the SSE metric and path towards solution.

**Fig. 3** Abalone data set analysis. The surface plot (a) shows a main valley where the initial point is located for the advantage of the proposed algorithm. The contour plot (b) depicts the upper bounds that can be used to reduce the search space of search-based algorithms



**(a)** Error surface with the STAT metric.



**(b)** Contour plot of the STAT metric and path towards solution.

**Fig. 4** MortPollution data set analysis. The surface plot shown in (a) demonstrates the complexity of the problem as it has many local minima; however, the initial point is placed in a position that allows the proposed algorithm to make progress towards the solution. The contour plot in (b) shows the upper bounds that fail to contain either the initial point or the minimum found with the proposed algorithm

the *MortPollution* data set; the figure shows the initial point, upper bounds, and the global minimum. Clearly, the upper bounds obtained using Ortiz et al. method [45] fail to contain the estimated initial point nor the global minimum; therefore, any constrained search-based or optimization-based algorithms may fail to obtain a "good" set of hyper-parameters. Consequently, this research only relies on the initial hyper-parameters previously described, which are supplied as inputs for Algorithm 3. Note that although this research avoids using the upper bounds, they were included in this paper to encourage and motivate the reader to advance the research in this area.

When Algorithm 3 is finished processing each data set using the training set $\mathcal{T}$, the resulting estimated hyper-parameters are those shown in Table 2. The second column of Table 2 shows the total number of iterations at the point where the algorithm stopped; in average the total number of iterations is around seven, which is one of the most important properties of the method. Column three and four of Table 2 show the hyper-parameters found; while the fifth column reports the $\ell_2$-norm of the residual error produced by the algorithm at the last iteration. Note how variable is this value depending on the data set, specially for regression since the error functions to minimize vary and have different output domains. Finally, the sixth column shows the criteria that made the algorithm stop; it is clear that the most common is the criteria $\varepsilon_3$ described in (54). This latter statement suggests that the algorithm stopped because no further progress was being made towards the solution, i.e., a minimum has been reached. An example of this is the case presented in Fig. 4; the algorithm proceeds iteratively making progress towards a minimum as indicated by the gradient and its direction. As indicated in Table 3 and depicted in Fig. 4, the algorithm used 15 iterations until there was no significant difference between the residual error in the current iteration and in the previous iteration, that is, the algorithm stopped using criteria $\varepsilon_3$. The second most common stopping criteria was that of $\varepsilon_2$ in which the actual set of hyper-parameters in the current iteration was not significantly different than the set given by the previous iteration; this is precisely the case shown in Fig. 3 where the algorithm stops after 2013_153 three iterations. In rare cases, mostly when the data set is separable (in classification) or easy to fit (in regression), the algorithm stops by finding a minimum to the linear least squares problem.

Up to this point the experiments only involve the training set $\mathcal{T}$. The second part of the experiments involves the testing set in order to analyze the generalization capabilities and the quality of the hyper-parameters found i.e., the set of hyper-parameters $C$ and $\gamma$. For this purpose, let us define $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{N^*}$ as the testing set, where $N^*$ is the number of samples available for testing. The value for $N^*$ comes directly from Table 1. Let us remark that the testing set $\mathcal{D}$ has never been shown to the LP-SVR model before nor it has been used for the estimation of the initial hyper-parameters. Thus, the experiments consists of using the testing set $\mathcal{D}$ as inputs for the following regression function:

$$F(\mathbf{x}) = \sum_{i \in SV} (\alpha_i^+ - \alpha_i^-) K(\mathbf{x}, \mathbf{x}_i) + b^+ - b^-, \tag{56}$$

where $SV$ denotes the set of indexes corresponding to the support vectors found during the training phase, i.e., the indexes corresponding to those $\alpha^+$ and $\alpha^-$ greater than zero [62]; $b$ is also given after training with the corresponding hyper-parameters shown Table 2. Note that (56) is simply a particular case of (21) which corresponds to the LP-SVR

**Table 2** Algorithm results for each data set: number of iterations, regularization parameter, kernel parameter, $\ell_2$-norm of the residual function, and stopping criteria

| Data set | $\varepsilon_1 = 1 \times 10^{-4}, \varepsilon_2, \varepsilon_3 = 1 \times 10^{-3}$, and $\varepsilon_4 = 100$ | | | | |
|---|---|---|---|---|---|
| | $t$ | $C_t$ | $\sqrt{\frac{1}{2\gamma_t}}$ | $\lVert\widetilde{\mathbf{R}}(\boldsymbol{\theta}_t \mid \mathcal{T}_K)\rVert_2$ | Stop Crit. |
| Ripley | 5 | 5.047 | 0.2501 | 0.0021 | $\varepsilon_3$ |
| Sonar | 5 | 5.413 | 1.5041 | 0.0013 | $\varepsilon_3$ |
| Wine | 2 | 0.031 | 0.2500 | 0.0001 | $\varepsilon_1$ |
| Spiral | 4 | 8.287 | 0.2515 | 0.0001 | $\varepsilon_1$ |
| ADA | 6 | 0.460 | 117.82 | 0.9813 | $\varepsilon_3$ |
| GINA | 9 | 0.125 | 157.49 | 0.1658 | $\varepsilon_2$ |
| HIVA | 10 | 0.500 | 8.0730 | 0.0954 | $\varepsilon_3$ |
| NOVA | 14 | 2.004 | 4.7045 | 0.0313 | $\varepsilon_2$ |
| SYLVA | 4 | 1.125 | 4,096.1 | 0.1512 | $\varepsilon_2$ |
| Iris | 3 | 11.46 | 1.7726 | 0.0019 | $\varepsilon_3$ |
| MODIS | 7 | 0.501 | 0.1249 | 0.0813 | $\varepsilon_3$ |
| SincF | 5 | 959.1 | 0.9978 | 0.0011 | $\varepsilon_1$ |
| LoadP | 4 | 0.499 | 0.1254 | 0.0210 | $\varepsilon_2$ |
| MortPollution | 16 | 548.1 | 748.28 | $7.6488 \times 10^4$ | $\varepsilon_3$ |
| Bodyfat | 8 | 1.112 | 104.12 | 0.0002 | $\varepsilon_1$ |
| Betaplasma | 5 | 733.6 | 2413.8 | $4.0432 \times 10^{13}$ | $\varepsilon_3$ |
| Retplasma | 6 | 1179 | 2413.8 | $6.2076 \times 10^{13}$ | $\varepsilon_3$ |
| Autompg | 4 | 41.41 | 1541.7 | $1.2210 \times 10^6$ | $\varepsilon_3$ |
| Housing | 7 | 51.98 | 199.8 | $7.6078 \times 10^6$ | $\varepsilon_3$ |
| Concrete | 11 | 89.21 | 343.5 | $2.1263 \times 10^9$ | $\varepsilon_2$ |
| Abalone | 3 | 630.9 | 194,984 | $2.1886 \times 10^8$ | $\varepsilon_2$ |
| | Average 6.6 | – | – | – | Mode: $\varepsilon_3$ |

model in (20). Then the output $F(\mathbf{x}_i)$ is observed for all $i \in \mathcal{D}$ and the different error metrics presented in Sect. 4 are computed with the purpose of assessing the quality of the final solution. This analysis is shown in Table 3, which presents the result of the $n$-th function (or error criteria), $r_n(\widetilde{\boldsymbol{\theta}}^\star \mid \mathcal{D})$, evaluated at the approximate solution $\widetilde{\boldsymbol{\theta}}^\star$ using only the testing set $\mathcal{D}$. These results are shown in columns two through six. In column number two is shown the modified estimate of scaled error rate (27), which was used with parameters $\zeta = \frac{1}{N^*}$ and $\tau = 100$. The parameter $\zeta$ was chosen by convenience in order to have an error within the interval [0, 1]. The third column displays results for when the balanced error rate (29) was utilized. The area under the ROC curve (30) shown in the fourth column also produces a result within the same interval as the BER. Note that classification error functions in average are very close to zero for practical purposes, which is desirable.

For regression problems, the error functions used are shown in the fifth and sixth column; the metrics have a wide interval that is always positive, i.e., the sum of squared errors (31) and the statistical properties (34) fall into the interval $[0, \Re^+]$. These large outputs in the SSE and STAT metrics are expected since neither the target, $y$, nor the output data, $F(\mathbf{x})$, have been passed through any pre-processing technique, e.g., normalization to zero mean and unit variance; however, for a qualitative analysis in the statistical sense, this research includes, in Table 3 columns six through seven, the statistical properties of the residuals given by $(F(\mathbf{x}) - y)_{\mathcal{D}}$. This residual is acquired by showing the testing set $\mathcal{D}$ to the LP-SVR model with hyper-parameters $\widetilde{\boldsymbol{\theta}}^\star$ and measuring the output $F(\mathbf{x})$. However, this time, the standard normalization procedure is followed [45], that is, the desired output $y_i$, for all $i \in \mathcal{D}$, is normalized to have zero mean and unit variance, using the exact same shifting and scaling factors to shift and scale $F(\mathbf{x})$. Ideally, it is desired for the average of the residuals to be zero, as well as their standard deviation. As the table shows, the proposed set of hyper-parameters produces near zero mean and small variance residuals in the case of both classification and regression, which typically indicates good classification and regression models and low error predictions. For two and multi-class classification problems, the table shows low errors, meaning good generalization capabilities.

## 6.3 Comparison with other methods

Finally, this research introduces Table 4, which shows a comparison of the proposed method for finding SVR hyper-parameters with other algorithms. The first method is known as "Grid Search" [45], which is typically the "brute force" way to determine a "good" set of hyper-parameters. The Grid Search method can be very costly, in terms of computational expense, depending on the level of refinement desired. The second method of comparison is the "Pattern Search" method [41], which searches for a solution in the hyper-parameters space neighborhood until it finds a neighbor location that produces a smaller error than in the current location. The third comparison method involves a Genetic Algorithm [63] that performs the search for hyper-parameters based on mutation and reproduction theories. The new generation of hyper-parameters may reduce the error of the current generation of hyper-parameters. The last method of comparison is known as "Practical Selection" [8] which is simply a heuristic used to directly estimate a "good" set of hyper-parameters. The experimental results carried by Ortiz et al. [45] are included in this table. These results involve measuring the root mean squared error (RMSE), and the total training time of each algorithm. The code number for the first column regarding the data sets is as follows: 1=*MortPollution*, 2=*Bodyfat*, 3=*Betaplasma*, 4=*Retplasma*, 5=*Autompg*, 6=*Housing*, 7=*Concrete*, and 8=*Abalone*. Notice that in this experiment, the three hyper-parameters $C$, $\gamma$, and $\epsilon$ are

**Table 3** Analysis of the final quality of the proposed methodology

| Data set | $r_n(\widetilde{\boldsymbol{\theta}}^{\star}\vert\mathcal{D})$ | | | | | $(F(\mathbf{x})-y)_{\mathcal{D}}$ | |
|---|---|---|---|---|---|---|---|
| | ESER | BER | 1-AUC | SSE | STAT | $\mu_{\mathcal{D}}$ | $\sigma_{\mathcal{D}}$ |
| Ripley | – | 0.0852 | 0.0259 | – | – | −0.0660 | 0.4023 |
| Sonar | – | 0.0506 | 0.0631 | – | – | 0.0385 | 0.2760 |
| Wine | – | 0.0000 | 0.0000 | – | – | 0.0008 | 0.0007 |
| Spiral | – | 0.0000 | 0.0000 | – | – | 0.0001 | 0.0008 |
| ADA | – | 0.1486 | 0.0669 | – | – | 0.0012 | 0.2001 |
| GINA | – | 0.0027 | 0.0000 | – | – | 0.0069 | 0.0397 |
| HIVA | – | 0.1714 | 0.0452 | – | – | 0.0270 | 0.1609 |
| NOVA | – | 0.0000 | 0.0000 | – | – | 0.0003 | 0.0189 |
| SYLVA | – | 0.0056 | 0.0000 | – | – | −0.0016 | 0.1980 |
| Iris | 0.0008 | 0.0001 | – | – | – | −0.0001 | 0.0022 |
| MODIS | 0.0069 | 0.0317 | – | – | – | 0.0704 | 0.3109 |
| SincF | – | – | – | 0.0003 | 0.0012 | 0.0001 | 0.0008 |
| LoadP | – | – | – | 0.0097 | 0.1151 | −0.0175 | 0.1112 |
| MortPollution | – | – | – | 18,707 | 1,693.5 | −0.0493 | 0.6609 |
| Bodyfat | – | – | – | 0.0058 | 0.0009 | 0.0412 | 0.5709 |
| Betaplasma | – | – | – | 1,355,801 | 21,710 | −0.0708 | 0.8049 |
| Retplasma | – | – | – | 3,568,079 | 49,342 | −0.4326 | 1.0624 |
| Autompg | – | – | – | 1,112.7 | 14.575 | −0.0186 | 0.4867 |
| Housing | – | – | – | 1,331.9 | 13.314 | −0.1082 | 0.3816 |
| Concrete | – | – | – | 8,829.7 | 43.313 | 0.0245 | 0.3921 |
| Abalone | – | – | – | 2,593.9 | 3.3164 | 0.0931 | 0.5387 |
| Average | 0.0039 | 0.0450 | 0.0223 | – | – | −0.0219 | 0.3153 |

The testing set, $\mathcal{D}$, is presented as input for the trained LP-SVR models using the hyper-parameters found, $\widetilde{\boldsymbol{\theta}}^{\star}$, and the different error metrics, $r_n$, are computed for the final assessment of the generalization error for each metric, corresponding to each column

being calculated following the exact same guidelines as in [45]. However, they are not optimizing $\mathbf{R}(\boldsymbol{\theta}|\mathcal{T})$, i.e. authors use their own method to obtain the hyper-parameters.

As Table 4 shows, for most cases the proposed method finds a better solution in terms of the RMSE which is the primary goal of this research. However, even though this research is not primarily focused on reducing the computational time, it should be noted that as the size of the problem increases, less time is required for training. Also, two things should be noted in regard to the reduction in training time. First, the difference in the training methodology from the experiments of Ortiz et al. [45] and the experiments of this research is that this research uses a training methodology that is more efficient in large-scale settings since it provides sparser solutions [54]; therefore, table shows that for the proposed method the time becomes less as the scale of the problem increases, but the converse is also true, the smaller the problem the more it takes to finish the training in comparison with the other methods. Second, the relatively high training time for the smaller problems can be safely attributed to the inherent computational expense of the line-search strategy of the proposed quasi-Newton method and the unpredictability of the error surface as it leads or misleads the optimization process. Consequently, this research finds its best applicability in large scale problems which is currently one of the major issues in the area [54]. In fact Ortiz et al. pointed out that it is "impracticable" to use direct methods in large-scale SVR problems, and that their search space reduction method could help in reducing training time [45]. However, the alternative presented in this research clearly is better shaped for medium to large-scale applications. The authors of this research leave as future work, the study of different or more error functions.

As a final remark, note that the concepts discussed in this paper can also be applied to other SVM/SVR-based learning machines with little or no modification. And one of the most important properties of the proposed method is that, being a nonlinear least squares problem, one may choose any number of error functions to minimize, and any number of hyper-parameters to estimate. The recommendation is to keep it simple, one or two error functions and two or three hyper-parameters should suffice; it is highly recommended for the number of error functions to always

**Table 4** Comparison of the proposed algorithm with the following methods: Grid Search [45], Pattern Search [41], Genetic Algorithms [63], and Practical Selection [8]. The name of the data set corresponding to the number given in the first column is given by the following relationship: 1=*MortPollution*, 2=*Bodyfat*, 3=*Betaplasma*, 4=*Retplasma*, 5=*Autompg*, 6=*Housing*, 7=*Concrete*, and 8=*Abalone*

| Data set | Grid S. [45] | | Pattern S. [41] | | Genetic A. [63] | | Practical S. [8] | | Proposed | |
|---|---|---|---|---|---|---|---|---|---|---|
| | RMSE | Time | RMSE | Time | RMSE | Time | RMSE | Time | RMSE | Time |
| 1 | 51.09 | 2.18 s | 48.22 | 15.4 s | 49.04 | 6.1 s | 63.85 | 0.01 s | 39.48 | 91 s |
| 2 | 0.011 | 24.1 s | 0.011 | 81.8 s | 0.011 | 109 s | 0.017 | 0.02 s | 0.011 | 74 s |
| 3 | 172.3 | 33.2 s | 184.7 | 644 s | 169.7 | 182 s | 185.3 | 0.01 s | 146.7 | 241 s |
| 4 | 254.9 | 32.3 s | 259.0 | 68.5 s | 261.9 | 182 s | 251.5 | 0.02 s | 238 | 77 s |
| 5 | 4.543 | 72.0 s | 4.625 | 122 s | 4.657 | 410 s | 4.740 | 0.02 s | 3.777 | 115 s |
| 6 | 4.989 | 157 s | 3.681 | 698 s | 5.032 | 469 s | 9.339 | 0.07 s | 3.632 | 313 s |
| 7 | 28.50 | 885 s | 28.56 | 2.96 h | 28.55 | 27 m | 28.74 | 0.33 s | 6.547 | 23 m |
| 8 | 1.777 | 1.53 h | 1.792 | 5.28 h | 1.772 | 3.9 h | 1.819 | 3.13 s | 1.761 | 59 m |

exceed or equal the number of hyper-parameters to be estimated.

## 7 Conclusions

This paper discussed an algorithm for LP-SVR hyper-parameters selection. This research proposes a quasi-Newton method for a minimization problem known as nonlinear least squares problem, that uses a Gauss-Newton gradient, a globalization strategy, and an inexact computation of first order information, that is, the Jacobian is computed via finite differences. The need for a "good" initial point in the hyper-parameters space is alleviated using existing methods that proved to be crucial in the rapid convergence of the proposed algorithm.

This research explored the cases of two and multi-class problems including regression; experimental results suggest that the algorithm achieves negligible variability when analyzing the statistical properties of the residual error. Simulations included mostly standard benchmark data sets from real-life applications, a small number of synthetic data sets, and also, comparisons with other state-of-the-art search methods. When the method was compared to other approaches, it was found that in terms of the root mean squared error, the proposed approach produces the lowest error in most cases.

The method proposed in this paper represents a particularization of the generalized method introduced at the beginning. This generalized method can be used to train other types of support vector machines. Furthermore, this method is not limited to a particular number of error functions or to a particular number of hyper-parameters. Moreover, experimental results suggest that the proposed method, if used along with a linear programming support vector regression approach and a large-scale training method, it produces its best results in large-scale applications.

This research significantly advances the natural problem of hyper-parameters selection in most of today's SVR/SVR-based methods. It does so by exploring a more efficient LP-SVR formulation along with a nonlinear least squares quasi-Newton strategy to minimize an estimate of the true generalization error.

**Acknowledgments** The author P. R. P. performed part of this work while at NASA Goddard Space Flight Center as part of the Graduate Student Summer Program (GSSP 2009) under the supervision of Dr. James C. Tilton. This work was supported in part by the National Council for Science and Technology (CONACyT), Mexico, under Grant 193324/303732 and mentored by Dr. Greg Hamerly who is with the department of Computer Science at Baylor University. Finally, the authors acknowledge the support of the Large–Scale Multispectral Multidimensional Analysis (LSMMA) Laboratory (www.lsmmalab.com).

## References

1. Anguita D, Boni A, Ridella S, Rivieccio F, Sterpi D (2005) Theoretical and practical model selection methods for support vector classifiers. In: Support vector machines: theory and applications, Springer, Berlin, pp 159–179
2. Anguita D, Ridella S, Rivieccio F, Zunino R (2003) Hyperparameter design criteria for support vector classifiers. Neurocomputing 55(1–2):109–134
3. Argáez M, Velázquez L (2003) A new infeasible interior-point algorithm for linear programming. In: Proceedings of the 2003 conference on diversity in computing, TAPIA '03, ACM, New York, pp 12–14. doi:10.1145/948542.948545
4. Armijo L (1966) Minimization of functions having lipschitz continuous first partial derivatives. Pac J Math 16(1):1–3
5. Blackard J, Dean D (1999) Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. Comput Electr Agric 24(3): 131–151
6. Cawley G (2006) Leave-one-out cross-validation based model selection criteria for weighted ls-svms. In: Proceedings of the

IEEE international joint conference on neural networks, IJCNN'06, pp 1661–1668. doi:10.1109/IJCNN.2006.246634
7. Chang M, Lin C (2005) Leave-one-out bounds for support vector regression model selection. Neural Comput 17(5):1188–1222
8. Cherkassky V, Ma Y (2004) Practical selection of svm parameters and noise estimation for svm regression. Neural Netw 17(1): 113–126
9. Collobert R, Bengio S (2001) Svmtorch: support vector machines for large-scale regression problems. J Mach Learn Res 1:143–160. doi:10.1162/15324430152733142
10. Courant R, Hilbert D (1966) Methods of mathematical physics. Interscience, New York
11. Dennis J, Schnabel R (1996) Numerical methods for unconstrained optimization and nonlinear equations. Society for Industrial Mathematics, Philadelphia
12. Duan K, Keerthi S, Poo A (2003) Evaluation of simple performance measures for tuning SVM hyperparameters. Neurocomputing 51:41–59
13. Fawcett T (2004) Roc graphs: notes and practical considerations for researchers. Mach Learn 31:1–38
14. Fisher R (1936) The use of multiple measurements in taxonomic problems. Ann Eugen 7(2):179–188
15. Forina M, Leardi R, Armanino C, Lanteri S (1998) PARVUS: an extendable package of programs for data exploration, classification and correlation. Institute of Pharmaceutical and Food Analysis Technologies, Genoa, Italy
16. Frank A, Asuncion A (2010) UCI machine learning repository. http://archive.ics.uci.edu/ml
17. Gorman R, Sejnowski T (1988) Analysis of hidden units in a layered network trained to classify sonar targets. Neural Netw 1(1):75–89
18. Hart P, Duda R, Stork D (2001) Pattern classification. Wiley, New York
19. Haykin SS (2009) Neural networks and learning machines. Prentice Hall, Upper Saddle River
20. He Q, Wu C (2011) Separating theorem of samples in banach space for support vector machine learning. Int J Mach Learn Cybern 2(1):49–54
21. Hestenes M (1975) Pseudoinversus and conjugate gradients. Commun ACM 18(1):40–43
22. Hui-ren Z, Pi-e Z (2008) Method for selecting parameters of least squares support vector machines based on GA and bootstrap. J Syst Simul 12:58. doi:cnki:sun:xtfz.0.2008-12-058
23. Ito K, Nakano R (2003) Optimizing support vector regression hyperparameters based on cross-validation. In: Proceedings of the IEEE international Joint Conference on neural networks, vol 3, pp 2077–2082
24. Joachims T (1998) Text categorization with support vector machines: learning with many relevant features. Machine learning ECML-98, Computer Science Department, University of Dortmund, pp 137–142
25. Joachims T (1999) Making large-scale support vector machine learning practical. In: Advances in kernel methods, MIT Press, Cambridge, pp 169–184
26. Karasuyama M, Kitakoshi D, Nakano R (2006) Revised optimizer of svr hyperparameters minimizing cross-validation error. In: Proceedings of the IEEE international joint conference on neural networks, IJCNN'06, pp 319–326
27. Karasuyama M, Nakano R (2007) Optimizing svr hyperparameters via fast cross-validation using aosvr. In: Proceedings of the IEEE international joint conference on neural networks, IJCNN 2007, pp 1186–1191
28. Karsaz A, Mashhadi H, Mirsalehi M (2010) Market clearing price and load forecasting using cooperative co-evolutionary approach. Int J Electr Power Energy Syst 32(5):408–415
29. Kay S (2006) Intuitive probability and random processes using MATLAB, 1st edn. Springer, Berlin. doi:10.1007/b104645
30. Khemchandani R, Karpatne A, Chandra S (2012) Twin support vector regression for the simultaneous learning of a function and its derivatives. Int J Mach Learn Cybern, Springer, pp 1–13. doi: 10.1007/s13042-012-0072-1
31. Kinzett D, Zhang M, Johnston M (2008) Using numerical simplification to control bloat in genetic programming. Simul Evol Learn 5361:493–502. doi:10.1007/978-3-540-89694-4_50
32. Kobayashi K, Kitakoshi D, Nakano R (2005) Yet faster method to optimize svr hyperparameters based on minimizing cross-validation error. In: Proceedings of the 2005 IEEE international joint conference on neural networks, IJCNN'05, vol 2, pp 871–876. doi:10.1109/IJCNN.2005.1555967
33. Kohavi R (1996) Scaling up the accuracy of naive-Bayes classifiers: a decision-tree hybrid. In: Proceedings of the second international conference on knowledge discovery and data mining, vol 7. Menlo Park, AAAI Press, USA
34. Lang K, Witbrock M (1988) Learning to tell two spirals apart. In: Proceedings of the 1988 connectionist models summer school, pp 52–59 (M. Kaufmann)
35. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11): 2278–2324. doi:10.1109/5.726791
36. Liu Z, Wu Q, Zhang Y, Philip Chen C (2011) Adaptive least squares support vector machines filter for hand tremor canceling in microsurgery. Int J Mach Learn Cybern 2(1):37–47. doi:10.1007/s13042-011-0012-5
37. Lu Z, Sun J, Butts KR (2009) Linear programming support vector regression with wavelet kernel: a new approach to nonlinear dynamical systems identification. Math Comput Simul 79(7): 2051–2063. doi:10.1016/j.matcom.2008.10.011
38. Ma J, Theiler J, Perkins S (2003) Accurate on-line support vector regression. Neural Comput 15(11):2683–2703. doi:10.1162/089976603322385117
39. McDonald G, Schwing R (1973) Instabilities of regression estimates relating air pollution to mortality. Technometrics 15(3): 463–481. doi:10.2307/1266852
40. Mercer J (1909) Functions of positive and negative type, and their connection with the theory of integral equations. Philos Trans R Soc Lond Ser A (containing papers of a mathematical or physical character) 209:415–446. doi:10.1098/rsta.1909.0016
41. Momma M, Bennett K (2002) A pattern search method for model selection of support vector regression. In: Proceedings of the SIAM international conference on data mining, SIAM, Philadelphia, pp 261–274
42. Musa A (2012) Comparative study on classification performance between support vector machine and logistic regression. Int J Mach Learn Cybern, 1–12. doi:10.1007/s13042-012-0068-x
43. Nierenberg D, Stukel T, Baron J, Dain B, Greenberg E (1989) Determinants of plasma levels of beta-carotene and retinol. Skin cancer prevention study group. Am J Epidemiol 130(3):511–521
44. Nocedal J, Wright S (1999) Numerical optimization. Springer, Berlin. doi:10.1007/b98874
45. Ortiz-García E, Salcedo-Sanz S, Pérez-Bellido Á, Portilla-Figueras J (2009) Improving the training time of support vector regression algorithms through novel hyper-parameters search space reductions. Neurocomputing 72(16):3683–3691. doi:10.1016/j.neucom.2009.07.009
46. Osuna E, Castro O (2002) Convex hull in feature space for support vector machines. In: Advances in artificial intelligence IBERAMIA 2002, lecture notes in computer science, vol 2527, Springer, Berlin, pp 411–419. doi:10.1007/3-540-36131-6_42
47. Peng X (2010) Tsvr: an efficient twin support vector machine for regression. Neural Netw 23(3):365–372. doi:10.1016/j.neunet.2009.07.002

48. Penrose K, Nelson A, Fisher A (1985) Generalized body composition prediction equation for men using simple measurement techniques. Med Sci Sports Exerc 2(17):189
49. Platt J (1999) Using analytic qp and sparseness to speed training of support vector machines. In: Proceedings of the 1998 conference on Advances in neural information processing systems II, MIT Press, Cambridge, MA, USA, pp 557–563
50. Quinlan J (1993) Combining instance-based and model-based learning. In: Proceedings of the 10th international conference on machine learning, pp 236–243
51. Ren Y, Bai G (2010) Determination of optimal svm parameters by using ga/pso. J Comput 5(8):1160–1168. doi:10.4304/jcp.5.8.1160-1168
52. Ripley B (2008) Pattern recognition and neural networks, 1st edn. Cambridge University Press, Cambridge
53. Rivas-Perea P (2009) Southwestern US and northwestern mexico dust storm modeling trough moderate resolution imaging spectroradiometer data: a machine learning perspective. Technical report: NASA/UMBC/GEST graduate student summer program. http://gest.umbc.edu/student_opp/2009_gssp_reports.html
54. Rivas Perea P (2011) Algorithms for training large-scale linear programming support vector regression and classification. PhD thesis, The University of Texas at El Paso
55. Rivas-Perea P, Cota-Ruiz J (2012) An algorithm for training a large scale support vector machine for regression based on linear programming and decomposition methods. Pattern Recogn Lett (In Press). doi:10.1016/j.patrec.2012.10.026
56. Schölkopf B, Smola A, Williamson R, Bartlett P (2000) New support vector algorithms. Neural Comput 12(5):1207–1245. doi:10.1162/089976600300015565
57. Small K, Roth D (2010) Margin-based active learning for structured predictions. Int J Mach Learn Cybern 1(1–4):3–25. doi:10.1007/s13042-010-0003-y
58. Smets K, Verdonk B, Jordaan E (2007) Evaluation of performance measures for svr hyperparameter selection. In: Proceedings of the IEEE international joint conference on neural networks, IJCNN 2007, pp. 637–642. doi:10.1109/IJCNN.2007.4371031
59. Smola AJ, Schölkopf B (2004) A tutorial on support vector regression. Stat Comput 14(3):199–222. doi:10.1023/B:STCO.0000035301.49549.88
60. Stark H, Woods J (2001) Probability and random processes with applications to signal processing, 3rd edn. Prentice-Hall, Upper Saddle River
61. Torii Y, Abe S (2009) Decomposition techniques for training linear programming support vector machines. Neurocomputing 72(4-6):973–984. doi:10.1016/j.neucom.2008.04.008
62. Vapnik V, Golowich S, Smola A (1997) Support vector method for function approximation, regression estimation, and signal processing. Adv Neural Inf Process Syst 9:281–287
63. Wang L (2005) Support vector machines: theory and applications, studies in fuzziness and soft computing, vol 177, Springer, Berlin
64. Waugh S (1995) Extending and benchmarking cascade-correlation. PhD thesis, University of Tasmania, Tasmania
65. Xiao JZ, Wang HR, Yang XC, Gao Z (2012) Multiple faults diagnosis in motion system based on svm. Int J Mach Learn Cybern 3(1):77–82. doi:10.1007/s13042-011-0035-y
66. Xiaofang Y, Yaonan W (2008) Parameter selection of support vector machine for function approximation based on chaos optimization. J Syst Eng Electr 19(1):191–197. doi:10.1016/S1004-4132(08)60066-3
67. Xu Z, Huang K, Zhu J, King I, Lyu MR (2009) A novel kernel-based maximum a posteriori classification method. Neural Netw 22(7):977–987. doi:10.1016/j.neunet.2008.11.005
68. Yeh I (1998) Modeling of strength of high-performance concrete using artificial neural networks. Cement and Concrete research 28(12):1797–1808. doi:10.1016/S0008-8846(98)00165-3
69. Zhang JP, Li ZW, Yang J (2005) A parallel svm training algorithm on large-scale classification problems. In: Proceedings of the 2005 international conference on machine learning and cybernetics, vol 3, pp 1637–1641. doi:10.1109/icmlc.2005.1527207
70. Zhang L, Zhou W (2010) On the sparseness of 1-norm support vector machines. Neural Netw 23(3):373–385. doi:10.1016/j.neunet.2009.11.012
71. Zhang XQ, Gu CH (2007) Ch-svm based network anomaly detection. In: Proceedings of the 2007 international conference on machine learning and cybernetics, vol 6, pp 3261 –3266. doi:10.1109/icmlc.2007.4370710