Least-squares Support Vector Regression for GSP-Phot

prepared by:	R. Andrae
reference:	GAIA-C8-TN-MPIA-RAN-024
issue:	1
revision:	2
date:	2016-02-05
status:	Issued

Abstract

We discuss the least-squares support vector regression as an alternative to the standard SVR used in GSP-Phot. We find that LSSVR is trivial to implement, more flexible in its choice of kernel functions, faster to train, has fewer hyperparameters and provides scientific parameter estimates that are competitive or even slightly better than the standard SVR currently implemented in GSP-Phot. Nevertheless, we conclude that the advantages of LSSVR are not decisive such that we will keep the currently implemented standard SVR for GSP-Phot and use the LSSVR only as a backup.

Document History

Issue	Revision	Date	Author	Comment
1	2	2016-02-05	RAN	Adding appendix using weights in training.
1	1	2015-03-02	RAN	First issue.
D	1	2015-03-02	RAN	Including comments from Coryn.
D	0	2015-02-13	RAN	First draft.

1 Introduction

GSP-Phot employs a Support Vector Regression (hereafter SVR) to obtain a first estimate of the stellar parameters from the given BP/RP spectra. The SVR result is then used as an initial guess for the other GSP-Phot algorithms, Ilium and Aeneas. Currently, this SVR is a standard SVR algorithm, as described in, e.g., Sect. 3.4 of Deng et al. (2012). While its scientific AP-estimation performance is very good, its implementation within the DPAC software is somewhat complicated. It employs the external `libsvm` library, which is then wrapped by the CU8 `classifier` package. This design has several disadvantages: First, maintainance support for `libsvm` or (less likely) the `classifier` package could cease in the future. Second, the `classifier` package stores the support vectors (BP/RP spectra) in double precision, whereas float precision would be sufficient and would reduce the memory usage by a factor of two.[1] Third, `libsvm` has a very limited choice of kernel functions and most of them are deactived by the `classifier` package.

In this TN, we discuss least-squares SVR as an alternative. We first present the algorithm and its mathematical details, deriving the fully analytic solution that is trivial to implement. We then apply least-squares SVR to BP/RP spectra, comparing its scientific AP-estimation performance to the current GSP-Phot implementation. Finally, in Appendix A, we discuss how to use weighted training data, e.g., to incorporate errors of training labels or to put more emphasis on some training examples.

2 Least-squares SVR

In this section, we introduce the least-squares SVR algorithm.

2.1 Least-squares SVR vs. standard SVR

As the name suggests, least-squares SVR employs the squared-error loss function, shown in Fig. 1c. Conversely, standard SVR uses the so-called ϵ-insensitive absolute-error loss function,

[1]Currently, SVR models are quite large and DPCC is worried about the memory consumption.

shown in Fig. 1b. This leads to some crucial differences between the standard SVR and least-squares SVR:

- Least-squares SVR loses the sparsity of the dual problem solution that makes standard SVR so special. The sparsity of standard SVR originates from the insensitivity of its loss function to any errors that are smaller than ϵ (c.f. Fig. 1b). In other words, while for standard SVR the target function is based on only a handful of support vectors, for least-squares SVR *all* training data enter into the target function.
- While the dual problem of standard SVR is a quadratic programming problem that requires an iterative solution due to the presence of inequality constraints, the least-squares SVR leads to a dual problem that does not have any inequality constraints, such that a one-shot analytic (i.e. non-iterative) solution exists. For standard SVR, the ϵ-insensitivity of the loss function, Fig. 1b, gives rise to a distinction of inequality-type cases whether residuals are larger than ϵ or smaller than $-\epsilon$.

Given its one-shot analytic solution, least-squares SVR may therefore be computationally more efficient for small training data sets. For large training data sets, however, the sparsity of the standard SVR formulation might eventually lead to lower computational cost.

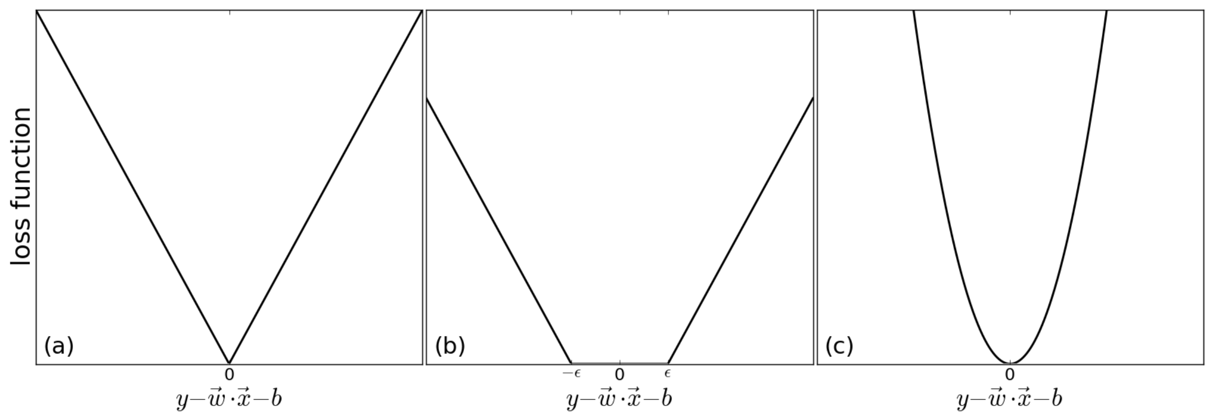

Figure 1: Comparison of loss functions: (a) The absolute-error loss funcion. (b) The ϵ-insensitive absolute-error loss function used by standard SVR. (c) The squared-error loss function used by least-squares SVR.

2.2 Formulating the primary problem

We are given N training data with input values $\{\boldsymbol{x}_n\}_{n=1}^N$ and output labels $\{y_n\}_{n=1}^N$. In the case of GSP-Phot, the $\boldsymbol{x}_n$ are the BP/RP spectra of the n-th training example, whereas y_n is, e.g., the effective temperature that we want to estimate from the spectrum. Leaving kernels aside for the moment, least-squares SVR then attempts to find a linear function,

$$f(\boldsymbol{x}) = \boldsymbol{w}^T \cdot \boldsymbol{x} + b\,, \tag{1}$$

to predict y from a given $\boldsymbol{x}$, where $\boldsymbol{w}$ and b are the parameters of the model. These parameters need to be fitted from the given training data by optimising some objective function. Additionally, there will also be some equality constraints, i.e., the objective function will be a Lagrangian, $L(\boldsymbol{w}, b, \boldsymbol{\alpha})$, where $\boldsymbol{\alpha}$ is the vector of Lagrange multipliers representing all equality constraints.[2] The optimisation process will be a minimisation, i.e., we seek estimates of the model parameters via

$$\hat{\boldsymbol{w}}, \hat{b} = \arg\min_{\boldsymbol{w},b} L(\boldsymbol{w}, b, \boldsymbol{\alpha})\,. \tag{2}$$

This is called the “primary problem”.

For least-squares SVR, Deng et al. (2012) (Sect. 8.2.1) define the following Lagrangian,

$$L(\boldsymbol{w}, b, \eta) = \frac{1}{2}||\boldsymbol{w}||^2 + \frac{C}{2}\sum_{n=1}^{N}\eta_n^2 + \sum_{n=1}^{N}\alpha_n\left(y_n - \boldsymbol{w}\cdot\boldsymbol{x}_n - b - \eta_n\right)\,, \tag{3}$$

where C is the cost parameter and η_n are the N residuals where the N equality constraints are $\eta_n = y_n - \boldsymbol{w}\cdot\boldsymbol{x}_n - b$. Leaving the constraints aside, this Lagrangian consists of two components: Minimisation of the first term, $\frac{1}{2}||\boldsymbol{w}||^2$, will maximise the margin width and thereby produce an SVR model that generalises well, i.e., which has a low expected error on test data it has *not* seen during its training process. Minimisation of the second term, $\sum_{n=1}^{N}\eta_n^2$, minimises the training error. Obviously, there is a trade-off here: We can attempt to perfectly fit the training data but then suffer from overfitting (a narrow margin, high variance, low bias). Alternatively, we can go for a large margin and vanishing generalisation error, at the expense of high training error (low variance, high bias). The cost parameter, C, governs exactly this trade-off. A large value of C will put more emphasis on the training error, whereas a small value of C will make the margin width more important. Furthermore, Eq. (3) contains the sum of squared residuals, $\sum_{n=1}^{N}\eta_n^2$, which obviously depends on the number N of given training examples. Therefore, the same value of C corresponds to different ratios of margin width vs. training error for training samples of different sizes N. Since we will choose the value of C through cross-validation, N will change and this could cause problems. We therefore modify Eq. (3) to become

$$L(\boldsymbol{w}, b, \eta) = \frac{1}{2}||\boldsymbol{w}||^2 + \frac{C}{2N}\sum_{n=1}^{N}\eta_n^2 + \sum_{n=1}^{N}\alpha_n\left(y_n - \boldsymbol{w}\cdot\boldsymbol{x}_n - b - \eta_n\right)\,, \tag{4}$$

such that the same value of C corresponds to exactly the same trade-off between margin width and training error, no matter how large the training set is.[3]

[2]For standard SVR, there are not only equality constraints but also inequality constraints, which are handled by a Lagrangian, too.

[3]Note that the last term in Eq. (4) does not require a factor of N because these are homologous constraints, i.e., they are numerically zero anyway.

2.3 Deriving the dual problem

We now solve the primary problem, i.e., we need to minimise the Lagrangian of Eq. (4) w.r.t. $\boldsymbol{w}$, b, and η_n, thereby removing any explicit dependence on these variables. We will then arrive at a Lagrangian that is only a function of the $\alpha_1, \ldots, \alpha_N$, which are the internal parameters of an SVM model. First, we take the derivative w.r.t. $\boldsymbol{w}$ and obtain:

$$\boldsymbol{\nabla}_w L = \boldsymbol{w} - \sum_{n=1}^{N} \alpha_n \boldsymbol{x}_n = 0 \quad \Leftrightarrow \quad \boldsymbol{w} = \sum_{n=1}^{N} \alpha_n \boldsymbol{x}_n \tag{5}$$

Second, we take the derivative w.r.t. b:

$$\frac{\partial L}{\partial b} = -\sum_{n=1}^{N} \alpha_n = 0 \quad \Leftrightarrow \quad \sum_{n=1}^{N} \alpha_n = 0 \tag{6}$$

Finally, we take the derivative w.r.t. η_n:

$$\frac{\partial L}{\partial \eta_n} = \frac{C}{N}\eta_n - \alpha_n = 0 \quad \Leftrightarrow \quad \eta_n = \frac{N}{C}\alpha_n \quad \forall n = 1, 2, \ldots, N \tag{7}$$

We now insert back into the original Lagrangian of Eq. (4). We first replace $\eta_n = \frac{N}{C}\alpha_n$ and obtain:

$$L(\boldsymbol{w}, b, \eta) = \frac{1}{2}||\boldsymbol{w}||^2 + \frac{N}{2C}\sum_{n=1}^{N} \alpha_n^2 + \sum_{n=1}^{N} \alpha_n \left(y_n - \boldsymbol{w} \cdot \boldsymbol{x}_n - b - \frac{N}{C}\alpha_n \right) \tag{8}$$

$$= \frac{1}{2}||\boldsymbol{w}||^2 + \frac{N}{2C}\sum_{n=1}^{N} \alpha_n^2 + \sum_{n=1}^{N} \alpha_n y_n - \sum_{n=1}^{N} \alpha_n \boldsymbol{w} \cdot \boldsymbol{x}_n - b\sum_{n=1}^{N} \alpha_n - \frac{N}{C}\sum_{n=1}^{N} \alpha_n^2 \tag{9}$$

$$= \frac{1}{2}||\boldsymbol{w}||^2 - \frac{N}{2C}\sum_{n=1}^{N} \alpha_n^2 + \sum_{n=1}^{N} \alpha_n y_n - \sum_{n=1}^{N} \alpha_n \boldsymbol{w} \cdot \boldsymbol{x}_n \tag{10}$$

In the last step, we already used $\sum_{n=1}^{N} \alpha_n = 0$ and if we now also insert $\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n \boldsymbol{x}_n$, we finally obtain:

$$L(\alpha) = \frac{1}{2}\sum_{m,n=1}^{N} \alpha_m \alpha_n \boldsymbol{x}_m \cdot \boldsymbol{x}_n - \frac{N}{2C}\sum_{n=1}^{N} \alpha_n^2 + \sum_{n=1}^{N} \alpha_n y_n - \sum_{m,n=1}^{N} \alpha_m \alpha_n \boldsymbol{x}_m \cdot \boldsymbol{x}_n \tag{11}$$

$$= -\frac{1}{2}\sum_{m,n=1}^{N} \alpha_m \alpha_n \boldsymbol{x}_m \cdot \boldsymbol{x}_n - \frac{N}{2C}\sum_{n=1}^{N} \alpha_n^2 + \sum_{n=1}^{N} \alpha_n y_n \tag{12}$$

$$= -\frac{1}{2}\sum_{m,n=1}^{N}\alpha_m\alpha_n\left(\boldsymbol{x}_m\cdot\boldsymbol{x}_n+\frac{N}{C}\delta_{mn}\right)+\sum_{n=1}^{N}\alpha_n y_n \tag{13}$$

Here, δ_{mn} denotes the Kronecker-delta. This objective function is subject to the constraint $\sum_{n=1}^{N}\alpha_n = 0$, such that the Lagrangian of the dual problem eventually reads:

$$L(\boldsymbol{\alpha}) = -\frac{1}{2}\sum_{m,n=1}^{N}\alpha_m\alpha_n\left(\boldsymbol{x}_m\cdot\boldsymbol{x}_n+\frac{N}{C}\delta_{mn}\right)+\sum_{n=1}^{N}\alpha_n y_n+\lambda\left(\sum_{n=1}^{N}\alpha_n\right) \tag{14}$$

2.4 Solving the dual problem

This dual problem involves only a single equality constraint but – unlike standard SVR – there are no inequality constraints. Consequently, the dual problem is a quadratic problem and it must have an analytic solution. Introducing the $N\times N$ matrix $M_{mn}=\boldsymbol{x}_m\cdot\boldsymbol{x}_n+\frac{N}{C}\delta_{mn}$ and the vector $\boldsymbol{e}=(1,1,\ldots,1)^T$, we can write Eq. (14) as:

$$L(\boldsymbol{\alpha}) = -\frac{1}{2}\boldsymbol{\alpha}^T\cdot M\cdot\boldsymbol{\alpha}+\boldsymbol{y}\cdot\boldsymbol{\alpha}+\lambda\left(\boldsymbol{e}\cdot\boldsymbol{\alpha}\right) \tag{15}$$

Taking the first derivative w.r.t. $\boldsymbol{\alpha}$, we obtain:

$$\boldsymbol{\nabla}_\alpha L(\boldsymbol{\alpha}) = -M\cdot\boldsymbol{\alpha}+\boldsymbol{y}+\lambda\boldsymbol{e}=0 \tag{16}$$

Assuming that the matrix M is invertible, which is almost guaranteed for a proper choice of $C>0$ that adds to M's diagonal elements, we can solve:

$$\boldsymbol{\alpha} = M^{-1}\cdot(\boldsymbol{y}+\lambda\boldsymbol{e}) \tag{17}$$

We still need to eliminate the Lagrange multiplier λ, though. Given $\frac{\partial L}{\partial\lambda}=\boldsymbol{e}\cdot\boldsymbol{\alpha}=0$, we can compute:

$$\boldsymbol{e}\cdot\boldsymbol{\alpha}=0=\boldsymbol{e}^T\cdot M^{-1}\cdot(\boldsymbol{y}+\lambda\boldsymbol{e}) \quad\Leftrightarrow\quad \lambda=-\frac{\boldsymbol{e}^T\cdot M^{-1}\cdot\boldsymbol{y}}{\boldsymbol{e}^T\cdot M^{-1}\cdot\boldsymbol{e}} \tag{18}$$

The analytic solution to the dual problem is therefore given by:

$$\boldsymbol{\alpha} = M^{-1}\cdot\boldsymbol{y}-\left(\frac{\boldsymbol{e}^T\cdot M^{-1}\cdot\boldsymbol{y}}{\boldsymbol{e}^T\cdot M^{-1}\cdot\boldsymbol{e}}\right)\left(M^{-1}\cdot\boldsymbol{e}\right) \tag{19}$$

2.5 Evaluating the target function

Once the analytic solution for $\boldsymbol{\alpha}$ has been obtained, we can calculate the target function given by Eq. (1), where $\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n \boldsymbol{x}_n$ and b is obtained by picking any n, say $n = 1$, and solving for the constraint $\eta_n = y_n - \boldsymbol{w} \cdot \boldsymbol{x}_n - b$ to obtain

$$b = y_1 - \eta_1 - \boldsymbol{w} \cdot \boldsymbol{x}_1 = y_1 - \frac{N}{C}\alpha_1 - \sum_{m=1}^{N} \alpha_m \boldsymbol{x}_m \cdot \boldsymbol{x}_1 \,. \tag{20}$$

We therefore obtain the target function

$$f(\boldsymbol{x}) = \sum_{n=1}^{N} \alpha_n \boldsymbol{x}_n \cdot \boldsymbol{x} + y_1 - \frac{N}{C}\alpha_1 - \sum_{m=1}^{N} \alpha_m \boldsymbol{x}_m \cdot \boldsymbol{x}_1 \,, \tag{21}$$

which predicts y for some new input vector $\boldsymbol{x}$.

2.6 From linear to nonlinear models

2.6.1 The kernel trick

So far, the model that we have considered in Eq. (1) had the mathematical form of a linear hyperplane. Real-world problems, such es estimating stellar parameters from BP/RP spectra, are usually not linear though. It is the so-called "kernel trick" that makes the SVR (standard or least-squares) applicable to nonlinear problems.

All the key equations of the problem involve the training data examples in the form of inner products $\boldsymbol{x}_m \cdot \boldsymbol{x}_n$, and never in any other form. This is true for the solution for b in Eq. (20) as well as for $f(\boldsymbol{x})$ which contains $\boldsymbol{w} \cdot \boldsymbol{x}$ where $\boldsymbol{w}$ is given by Eq. (5).

The kernel trick now works as follows: One maps $\boldsymbol{x}$ from the D-dimensional data space (for GSP-Phot, this is the space of 120 BP/RP pixels) into some other feature space via $\boldsymbol{z} = \Phi(\boldsymbol{x})$, where $\Phi(\cdot)$ is a nonlinear function. Then, one replaces any inner product $\boldsymbol{x}_m \cdot \boldsymbol{x}_n$ by $\boldsymbol{z}_m \cdot \boldsymbol{z}_n$ in all equations. What the mapping function $\Phi(\cdot)$ actually does is irrelevant because all we need are inner products, which we can write as,

$$\boldsymbol{z}_m \cdot \boldsymbol{z}_n = \Phi(\boldsymbol{x}_m) \cdot \Phi(\boldsymbol{x}_n) = K(\boldsymbol{x}_m, \boldsymbol{x}_n) \tag{22}$$

where $K(\cdot,\cdot)$ is the so-called kernel function that is symmetric in both arguments. All we need to do here, is specify the kernel function $K(\cdot,\cdot)$ but *not* the mapping $\Phi(\cdot)$ because we do not need it. As long as we are choosing a valid kernel function, it is enough to know that a mapping exists.

How does the kernel trick allow the SVM to handle nonlinear data? The equations of the SVM are still those of a linear hyperplane. However, the linear hyperplane no longer resides in the data space but instead it resides in the feature space that is created by the implicitely selected mapping. And because the mapping function $\Phi(\cdot)$ is nonlinear, a linear hyperplane in the kernel's feature space corresponds to a nonlinear boundary in the data space. Consequently, a kernel SVM can handle nonlinear data, provided that a suitable kernel is chosen.

2.6.2 Kernel functions

What is a valid choice for the kernel function? The kernel function is meant to represent an inner product in some feature space, which means that $K(\cdot,\cdot)$ has to be a symmetric and positive-semidefinite function, i.e., $K(\cdot,\cdot)$ must have the following two properties:

$$\text{symmetry:} \quad K(\boldsymbol{x},\boldsymbol{x}') = K(\boldsymbol{x}',\boldsymbol{x}) \qquad \forall \boldsymbol{x},\boldsymbol{x}' \tag{23}$$

$$\text{positive-semidefinite:} \quad K(\boldsymbol{x},\boldsymbol{x}) \geq 0 \qquad \forall \boldsymbol{x} \tag{24}$$

The two simplest kernel functions that satisfy these conditions are the constant kernel, $K(\boldsymbol{x}_n,\boldsymbol{x}_m) = 1$, and the linear kernel $K(\boldsymbol{x}_n,\boldsymbol{x}_m) = \boldsymbol{x}_n \cdot \boldsymbol{x}_m$. Furthermore, there are the following theorems:

1. Any summation of kernel functions, $K_1(\boldsymbol{x}_n,\boldsymbol{x}_m)+K_2(\boldsymbol{x}_n,\boldsymbol{x}_m)$, is a kernel function.
2. Any product of kernel functions, $K_1(\boldsymbol{x}_n,\boldsymbol{x}_m)\,K_2(\boldsymbol{x}_n,\boldsymbol{x}_m)$, is a kernel function.
3. Any convergent series of kernel functions, $\sum_{j=0}^{\infty} K_j(\boldsymbol{x}_n,\boldsymbol{x}_m)$, is a kernel function.

From these definitions it follows that any polynomial $p(\cdot)$ with argument $\boldsymbol{x}_n \cdot \boldsymbol{x}_m$ is a kernel function, e.g.,

$$K(\boldsymbol{x}_m,\boldsymbol{x}_n) = 1 + 2(\boldsymbol{x}_m \cdot \boldsymbol{x}_n) + \frac{1}{2}(\boldsymbol{x}_m \cdot \boldsymbol{x}_n)^2 \tag{25}$$

is a kernel function. Furthermore, any convergent polynomial series is a kernel function. This applies in particular to any convergent Taylor expansion. For instance,

$$K_{\text{exponential}}(\boldsymbol{x}_m,\boldsymbol{x}_n) = \sum_{j=0}^{\infty} \frac{1}{j!}(\boldsymbol{x}_m \cdot \boldsymbol{x}_n)^j = e^{\boldsymbol{x}_m \cdot \boldsymbol{x}_n} \tag{26}$$

$$K_{\text{Gaussian}}(\boldsymbol{x}_m,\boldsymbol{x}_n) = e^{-||\boldsymbol{x}_m||^2} e^{-||\boldsymbol{x}_n||^2} \sum_{j=0}^{\infty} \frac{(-2)^j}{j!}(\boldsymbol{x}_m \cdot \boldsymbol{x}_n)^j = e^{-||\boldsymbol{x}_m-\boldsymbol{x}_n||^2} \tag{27}$$

are both kernel functions. Furthermore, also the Cauchy function can act as a kernel:

$$K_{\text{Cauchy}}(\boldsymbol{x}_m,\boldsymbol{x}_n) = \frac{1}{1+||\boldsymbol{x}_m-\boldsymbol{x}_n||^2} = \sum_{p=0}^{\infty}(-1)^p||\boldsymbol{x}_m-\boldsymbol{x}_n||^{2p} \tag{28}$$

$$= \sum_{p=0}^{\infty} (-1)^p \left(||\boldsymbol{x}_m||^2 - 2\boldsymbol{x}_m \cdot \boldsymbol{x}_n + ||\boldsymbol{x}_n||^2 \right)^p \tag{29}$$

The advantage of the Cauchy kernel is that it does not involve any transcendental functions such as square-roots, logarithms or exponentials. As we are going to demonstrate in the next section, this makes the Cauchy kernel to have substantially lower computational cost than, e.g., the Gaussian kernel.

3 Scientific performance on BP/RP spectra

In order to get an impression of the scientific performance of such a least-squares SVR model, we apply it to cycle 8 BP/RP spectra. The Main Stellar Library chosen is PHOENIX at $G = 15$. The standard SVR used by GSP-Phot is trained on 10 000 training spectra. However, the standard SVR is sparse and at $G = 15$ it typically has 4 000 support vectors, whereas least-squares SVR is not sparse and will use *all* available training spectra as support vectors. Therefore, we restrict the training of the least-squares SVR to no more than 4 000 spectra. More precisely, we train least-squares SVR models using trainings sets containing 1 000, 2 000 and 4 000 spectra, which are randomly selected from the set of 12 000 training spectra available and whose composition is described in RAN-012.

Least-squares SVR has two hyperparameters – the cost C and the kernel width γ – whose values are found by minimising the two-fold cross-validation error on a two-dimensional brute-force grid search in C and γ. Figure 2 shows maps of the two-fold cross-validation error as a function of hyperparameters. Evidently, the maps are multimodal, which is the reason why we use a brute-force grid. If we used, e.g., a gradient-descent algorithm instead, we would simply get stuck in the nearest local minimum.[4] We use three different kernel functions, the Gaussian, the Cauchy kernel and the exponential kernel, to train least-squares SVR models for the parameters $\log_{10} T_{\rm eff}$, A_0, $\log g$ and [Fe/H].

Table 1 provides a detailed comparison. First and foremost, we see that, if trained on a similar number of 4 000 spectra, the Gaussian and the Cauchy kernel achieve AP estimation results that are competitive with the standard SVR currently implemented in GSP-Phot. If we train on fewer spectra, the results degrade. More precisely, least squares SVR using the Gaussian or the Cauchy kernel achieve almost identical results for $T_{\rm eff}$ and A_0, whereas they outperform the standard SVR when it comes to [Fe/H].[5] Finally, Table 1 also shows that while the Gaussian and the Cauchy kernel achieve very similar results, the exponential kernel fails hard to provide

[4]Standard SVR also suffers from such multimodalities when optimising the hyperparameters. In fact, the problem is even worse because standard SVR has a third hyperparameter, whereas LSSVR has only two.

[5]Their results for $\log g$ are also substantially better but here the current GSP-Phot implementation has some problem unrelated to SVR, so this is not a fair comparison.

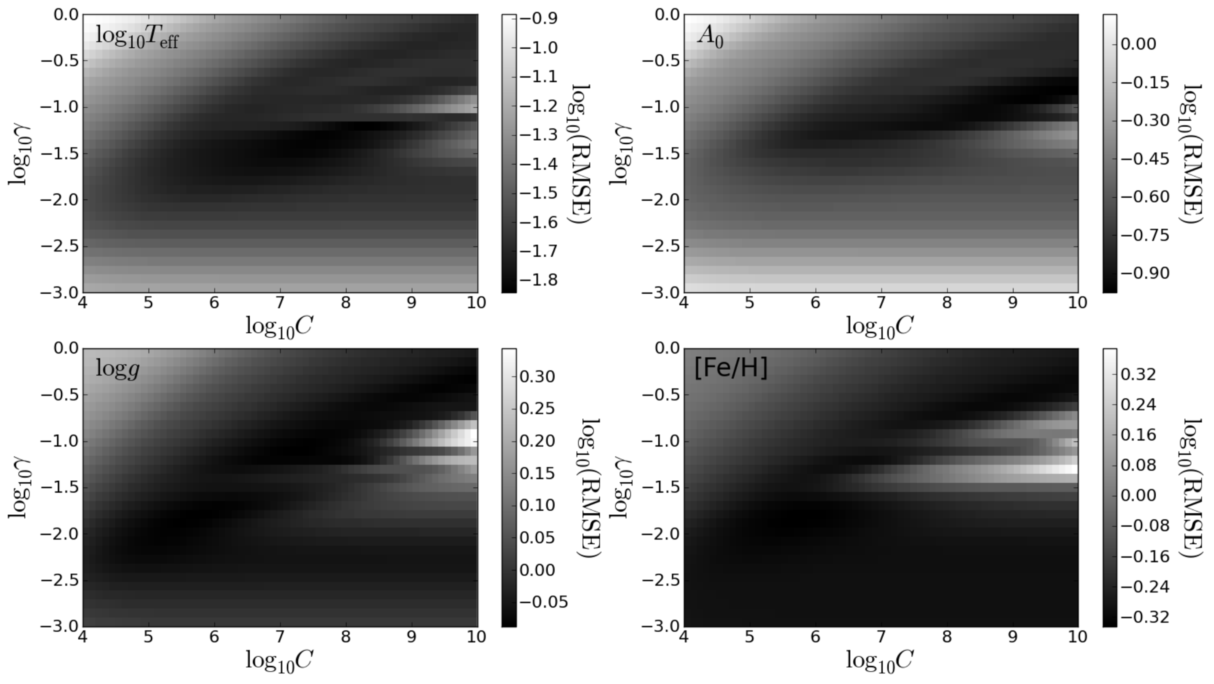

Figure 2: Maps of two-fold cross-validation RMS error as a function of hyperparameters C and γ for all four parameters ($\log_{10} T_{\rm eff}$, A_0, $\log g$, [Fe/H]). We used the Cauchy kernel and 4 000 BP/RP spectra for training.

Table 1: Scientific AP-estimation performance (RMS errors) of standard SVR and least-squares SVR for PHOENIX at $G = 15$. Least-squares SVR models are assessed using three different kernel functions (Gaussian, Cauchy and exponential) and for training sets of 1 000, 2 000 and 4 000 spectra, respectively. The standard SVR model was trained on a set of 12 000 spectra, whereof ca. 4 000 are support vectors.

SVR	standard	least-squares								
kernel	Gaussian	Gaussian			Cauchy			exponential		
training spectra	ca. 4 000	1 000	2 000	4 000	1 000	2 000	4 000	1 000	2 000	4 000
$T_{\rm eff}$ [K]	94	123	106	94	122	105	95	2314	1219	1006
A_0 [mag]	0.063	0.077	0.065	0.061	0.077	0.064	0.060	2.441	1.084	9.063
$\log g$ [dex]	0.709	0.490	0.486	0.396	0.472	0.479	0.396	0.813	1.304	14.262
[Fe/H] [dex]	0.278	0.306	0.296	0.218	0.301	0.295	0.215	0.744	1.495	1.350

any scientifically useful results. As we used exactly the same code implementation, this cannot be any software issue but it is a real failure of the exponential kernel.[6] We conclude that while seeking the perfect kernel function may be elusive, some care is still required because not all kernel functions are useful. Figure 3 directly compares the standard SVR currently implemented in GSP-Phot and the least-squares SVR using the Cauchy kernel.

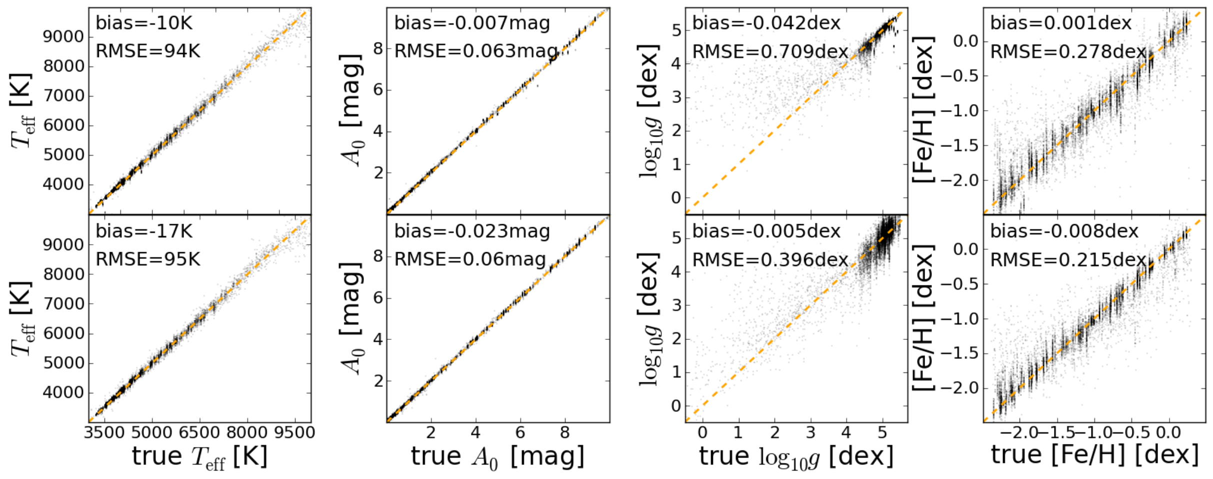

Figure 3: Scientific AP-estimation performance of the standard SVR used by GSP-Phot (top row) and least-squares SVR with Cauchy kernel trained on 4 000 spectra (bottom row). The LSSVR results here use the hyperparameter values that were inferred from Fig. 2.

As an aside, we also notice that the training process is ca. 30% faster if we use the Cauchy kernel instead of the Gaussian kernel. As mentioned before, this is because the Cauchy kernel does not involve any transcendental functions. Since training SVM models is a very time-consuming process, there is a real practical benefit of using the Cauchy kernel, even if it scientifically produces the same results as the Gaussian kernel.

4 Conclusions

We investigated least-squares SVR as an alternative to the standard SVR which is currently implemented in GSP-Phot. Here a summary of our comparison:

[6]We can only guess why the exponential kernel has such a poor performance. One potential reason is that the other kernels use distances $||\boldsymbol{x} - \boldsymbol{x}'||$ between spectra, which are somehow more informative than the projections $\boldsymbol{x} \cdot \boldsymbol{x}'$ used by the exponential kernel.

	standard SVR	least-squares SVR
implementation:	`libsvm` & `classifier`	from scratch
support vectors:	stored in double precision	stored in float precision
kernel function:	Gaussian only	whatever we want
hyperparameters:	C, γ, ϵ	C, γ
scientific results:	very good AP estimates	slightly better AP estimates

All in all, least-squares SVR is a viable alternative to the standard SVR that is currently implemented in GSP-Phot: Least-squares SVR is easier to maintain, uses less memory, is more flexible with regard to kernel functions, has less hyperparameters and is thus faster/easier to train, and, last but not least, even has a slightly better scientific AP-estimation performance. However, while least-squares SVR is one step ahead of the standard SVR in all respects, its advantages are still only minor. There is no striking argument that clearly calls for a replacement of the standard SVR by the least-squares SVR in GSP-Phot. We conclude that we will *not* change the SVR implementation of GSP-Phot at the moment. Nonetheless, the least-squares SVR investigated in this TN provides an excellent back-up in the case that any problems with `libsvm` or the `classifier` package occur in the future.

In future tests, we should also investigate whether SVR models perform better on CU5's spline coefficients instead of the pixels of the sampled spectra. Furthermore, we should investigate into multivariate SVR models, which estimate all four parameters ($T_{\rm eff}$, A_0, $\log g$ and [Fe/H]) in a single model instead of building four completely independent univariate models.

References

[RAN-012], Andrae, R., 2013, *Recent improvements and current performance of GSP-Phot*, `GAIA-C8-TN-MPIA-RAN-012`,
URL `http://www.rssd.esa.int/cs/livelink/open/3222048`

Deng, N., Tian, Y., Zhang, C., 2012, *Support Vector Machines: Optimization Based Theory, Algorithms, and Extensions*, Chapman & Hall/CRC, 1st edn.

A Training LSSVR on weighted labels

The LSSVR algorithm presented above are assuming that all training examples are of equal importance, or have no or identical label errors. In practice, one sometimes wants to drop this simplification and introduce a weighting. This can be accomplished by modifying Eq. (4) such that

$$L(\boldsymbol{w}, b, \eta) = \frac{1}{2}||\boldsymbol{w}||^2 + \frac{C}{2N}\sum_{n=1}^{N} W_n\eta_n^2 + \sum_{n=1}^{N}\alpha_n\left(y_n - \boldsymbol{w}\cdot\boldsymbol{x}_n - b - \eta_n\right) , \qquad (30)$$

where we have introduced the weights $W_1, W_2, \ldots, W_N$ for every training example. This is the only possible way to introduce weights. The first term, $\frac{1}{2}||\boldsymbol{w}||^2$ controls the margin and does not know about any training data. The last term introduces the Lagrange multipliers and is numerically zero. The middle term, on the other hand, describes the training error and it is thus the natural place to weigh up or down the impact of individual training examples. Note that we have effectively $C_n = W_n C$, i.e., every training example has its own cost parameter. However, we set the relative weights W_n a priori and then only have one global cost factor C.

Given this Lagrangian, we derive the secondary problem as before, where the solution is largely identical. In principle, we can use the substitution $C \to W_n C$ to directly jump to the solution of the LSSVR algorithm. Nevertheless, we want to give a proper derivation. First, we take the derivative w.r.t. $\boldsymbol{w}$ and obtain:

$$\boldsymbol{\nabla}_w L = \boldsymbol{w} - \sum_{n=1}^{N} \alpha_n \boldsymbol{x}_n = 0 \quad \Leftrightarrow \quad \boldsymbol{w} = \sum_{n=1}^{N} \alpha_n \boldsymbol{x}_n \tag{31}$$

Second, we take the derivative w.r.t. b:

$$\frac{\partial L}{\partial b} = -\sum_{n=1}^{N} \alpha_n = 0 \quad \Leftrightarrow \quad \sum_{n=1}^{N} \alpha_n = 0 \tag{32}$$

Finally, we take the derivative w.r.t. η_n:

$$\frac{\partial L}{\partial \eta_n} = \frac{C}{N} W_n \eta_n - \alpha_n = 0 \quad \Leftrightarrow \quad \eta_n = \frac{N}{W_n C}\alpha_n \quad \forall n = 1, 2, \ldots, N \tag{33}$$

We now insert back into the original Lagrangian of Eq. (4). We first replace $\eta_n = \frac{N}{C}\alpha_n$ and obtain:

$$L(\boldsymbol{\alpha}) = -\frac{1}{2}\sum_{m,n=1}^{N} \alpha_m \alpha_n \left(\boldsymbol{x}_m \cdot \boldsymbol{x}_n + \frac{N}{W_n C}\delta_{mn}\right) + \sum_{n=1}^{N} \alpha_n y_n + \lambda \left(\sum_{n=1}^{N} \alpha_n\right) \tag{34}$$

The Lagrangian is almost identical, only the diagonal elements of the matrix M are slightly modified, $M_{mn} = \boldsymbol{x}_m \cdot \boldsymbol{x}_n + \frac{N}{W_n C}\delta_{mn}$. Otherwise, the analytic solution for the Lagrange multipliers is still of the exact same form,

$$\boldsymbol{\alpha} = M^{-1} \cdot \boldsymbol{y} - \left(\frac{\boldsymbol{e}^T \cdot M^{-1} \cdot \boldsymbol{y}}{\boldsymbol{e}^T \cdot M^{-1} \cdot \boldsymbol{e}}\right) \left(M^{-1} \cdot \boldsymbol{e}\right) \tag{35}$$

However, when evaluating the target function, we need to pay attention to the newly introduced weighting one last time,

$$f(\boldsymbol{x}) = \sum_{n=1}^{N} \alpha_n \boldsymbol{x}_n \cdot \boldsymbol{x} + y_1 - \frac{N}{W_1 C}\alpha_1 - \sum_{m=1}^{N} \alpha_m \boldsymbol{x}_m \cdot \boldsymbol{x}_1 \,, \tag{36}$$

which predicts y for some new input vector $\boldsymbol{x}$.