ETH zürich

UQLab User Manual
Support Vector Machines For Classification

M. Moustapha, C. Lataniotis, S. Marelli, B. Sudret

Chair of Risk, Safety and Uncertainty Quantification
Stefano-Franscini-Platz 5
CH-8093 Zürich

Risk, Safety & Uncertainty Quantification

How to cite UQLab

S. Marelli, and B. Sudret, UQLab: A framework for uncertainty quantification in Matlab, Proc. 2nd Int. Conf. on Vulnerability, Risk Analysis and Management (ICVRAM2014), Liverpool, United Kingdom, 2014, 2554-2563.

How to cite this manual

M. Moustapha, C. Lataniotis, S. Marelli, B. Sudret, UQLab user manual – Support vector machines for classification, Report UQLab-V2.1-112, Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland, 2024

BibTeX entry

```
@TechReport{UQdoc_21_112,
author = {Moustapha, M. and Lataniotis, C. and Marelli, S. and Sudret, B.},
title = {{UQLab user manual -- Support vector machines for classification}},
institution = {Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich,
Switzerland},
year = {2024},
note = {Report UQLab-V2.1-112}
}
```

Document Data Sheet

Document Ref.	UQLAB-V2.1-112
Title:	UQLAB user manual – Support vector machines for classification
Authors:	M. Moustapha, C. Lataniotis, S. Marelli, B. Sudret Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland
Date:	15/04/2024

Doc. Version	Date	Comments
V2.1	15/04/2024	UQLAB V2.1 release
V2.0	01/02/2022	UQLAB V2.0 release
V1.4	01/02/2021	UQLAB V1.4 release
V1.3	19/09/2019	UQLAB V1.3 release
V1.2	22/02/2019	UQLAB V1.2 release • added automatic calculation of validation error if a validation set is provided
V1.1	05/07/2018	Initial release

Abstract

Support vector machines classification is a metamodelling technique which has been widely used in the machine learning community and brought more recently to the field of structural reliability. The UQLAB implementation of support vector machines for classification allows the user to easily define a classifier given a set of training points. An emphasis is given to the calibration of the hyperparameters of the models. Henceforth, numerous options are available to allow the user to tune the model.
This manual is divided into three parts:

- A brief introduction to the main concepts of SVM for classification together with the basic equations needed to build a model, which also includes relevant literature;
- A detailed usage of the UQLAB features for SVM classification throughout the implementation of basic examples;
- A comprehensive reference list of the available options for SVM classification in UQLAB.

Keywords: Machine learning, Surrogate modeling, Support vector machines, Classification

Contents

Chapter 1

Theory

1.1 Introduction

Support vector machines are a class of learning techniques developed to solve problems of learning from examples. Such problems occur when one needs to approximate a functional either because the latter is unknown, *i.e.* the underlying mapping is too complex to be modeled (*e.g.* meteorological phenomena) or because the functional is too expensive-to-evaluate for practical purposes (*e.g.* finite element simulation). Classification is the specific case of *supervised learning* which handles models whose outputs are discrete. Upon implementing the *structural risk minimization principle* (Vapnik, 1995), support vector machines classification achieves a high degree of accuracy while minimizing the risk of *overfitting*.
This manual considers support vector machines for binary classification, *i.e.* the particular case when only two classes are to be identified. In reliability analysis for instance, such classes may label the safety and the failure domains (Bourinet, 2018). The first chapter briefly presents the main concept and equations of support vector classification. For an in-depth introduction to support vector machines, the reader is referred to Vapnik (1995); Smola and Schölkopf (2004); Cherkassky and Mulier (2007). The second chapter details the usage of SVM within UQLAB while the last one presents a reference list of the associated commands.

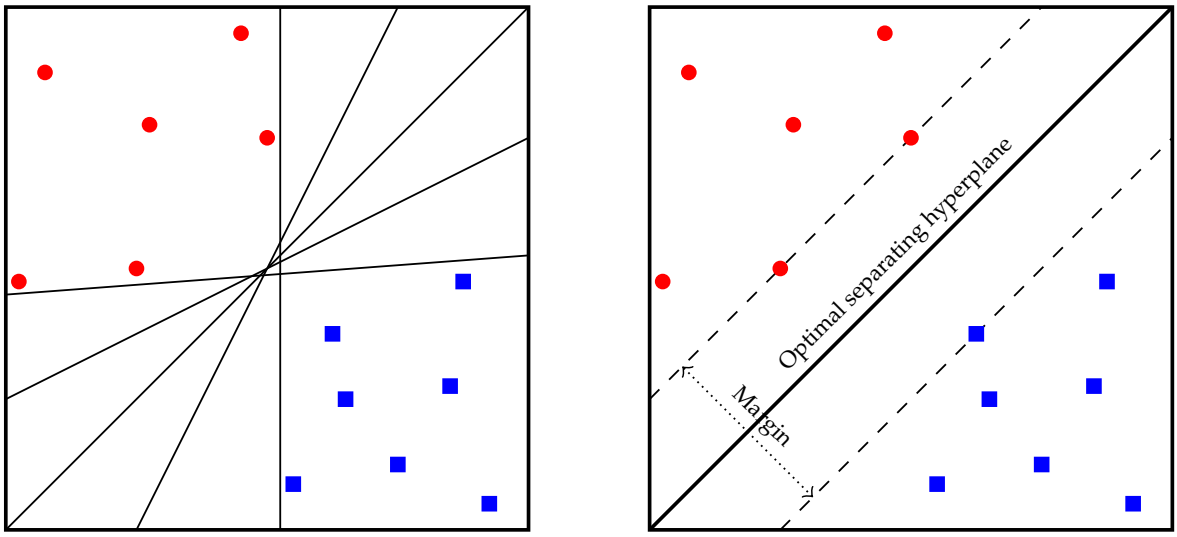
1.2 SVC basics: the optimal separating hyperplane

Let us consider a set of so-called *training* vectors $\mathcal{X} = \left\{\boldsymbol{x_i} \in \mathbb{R}^M, i = 1, \ldots, N\right\}$ and the associated labels $\mathcal{Y} = \{y_i = \{-1, 1\}, i = 1, \ldots, N\}$ which indicate the class of each training point. The task of learning consists in predicting which class a point belongs to, given these observations. Support vector machines achieve such binary classification by implementing two main ideas: mapping the data into a high-dimensional space and then constructing a linear classifier in this space.

1.2.1 Hard margin classification

For the sake of clarity, let us first consider the case when the data are linearly separable in the original input space. In such a case, one may find an infinite number of hyperplanes

that separate the data. Another constraint of the problem is however to find a classifier that generalizes well, *i.e.* that can predict with high accuracy on yet-to-be observed points. This is achieved in support vector machines by searching for the unique classifier with the largest distance to the closest training points. This distance is also known as the *margin*. The principle is illustrated in Figure 1: while many valid classifiers are represented in Figure 1a, the optimal separating hyperplane is shown in Figure 1b. Intuitively, the latter is more robust to noise in the training points.



(a) Valid separating lines

(b) The optimal separating line

Figure 1: Principle of the optimal separating hyperplane. In the left panel, some valid lines separate the two classes of data whereas in the right one, the unique and optimal separating hyperplane is shown, together with its margin (Figure from Moustapha (2016)).

Let us now consider the following canonical equation:

$$\mathcal{M}^{\text{SVC}}(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b, \tag{1.1}$$

and further assume that the separating hyperplane can be cast in the following form (Gunn, 1998):

$$\left\{\boldsymbol{x} \in \mathbb{R}^M : \boldsymbol{w}^T\boldsymbol{x} + b = 0\right\}, \tag{1.2}$$

where $\boldsymbol{w}$ and b are parameters to be set. The distance of any point $\boldsymbol{x}_i$ to this hyperplane reads:

$$d\left(\boldsymbol{x}_i, \mathcal{M}^{\text{SVC}}\right) = \frac{\left|\boldsymbol{w}^T\boldsymbol{x}_i + b\right|}{\|\boldsymbol{w}\|}. \tag{1.3}$$

It turns out that maximizing the margin is equivalent to minimizing the norm of $\boldsymbol{w}$ under some constraints.

Formally speaking, the optimization problem addressed by support vector machines reads

(Vapnik, 1995):

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2, \\ \text{subject to:} \quad & y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i + b\right) - 1 \geq 0, \quad i = \{1, \ldots, N\}, \end{aligned} \tag{1.4}$$

where the constraints ensure that all the training points lie outside the area covered by the margin. This problem is solved in its dual form thanks to its quadratic convex form. Upon introducing the Lagrange multipliers $\{\alpha_i, i = 1, \ldots, N\}$ and after some algebra, the final optimization problem reads:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j\, y_i y_j\, \boldsymbol{x}_i^T\boldsymbol{x}_j + \sum_{i=1}^{N}\alpha_i, \\ \text{subject to:} \quad & \sum_{i=1}^{N}\alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad i = \{1, \ldots, N\}. \end{aligned} \tag{1.5}$$

Solving this problem allows one to find the values of the coefficients $\{\alpha_i, i = 1, \ldots, N\}$ and that of the offset parameter b. The resulting values can then be plugged in the following expansion, hence allowing to express the prediction in terms of the training points only:

$$\mathcal{M}^{\text{SVC}}(\boldsymbol{x}) = \sum_{i=1}^{N}\alpha_i\, y_i\, \boldsymbol{x}_i^T\boldsymbol{x} + b. \tag{1.6}$$

As a final step, one decides the class of the new point by observing the sign of $\mathcal{M}^{\text{SVC}}(\boldsymbol{x})$, *i.e.*:

$$y(\boldsymbol{x}) = \text{sign}\left(\mathcal{M}^{\text{SVC}}(\boldsymbol{x})\right). \tag{1.7}$$

1.2.2 Soft margin classification

In some cases, the optimization problem introduced above may not be feasible. A turnaround is to allow for misclassification while keeping the number of errors minimal: this procedure is called *soft margin classification*. To this purpose, Cortes and Vapnik (1995) introduced the so-called *slack variables* ξ_i which measure the distance of a wrongly classified point to its actual class. The incurred penalty is expressed as a function of ξ_i. The final form of the problem to solve depends on the type of penalization. Standard formulations of SVM include the following two:

- Linear penalization

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi_i, \\ \text{subject to:} \quad & y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i + b\right) \geq 1 - \xi_i, \qquad i = \{1, \ldots, N\}. \end{aligned} \tag{1.8}$$

- Quadratic penalization

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{C}{2}\sum_{i=1}^{N}\xi_i^2, \\ \text{subject to:} \quad & y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i + b\right) \geq 1 - \xi_i, \qquad i = \{1, \ldots, N\}. \end{aligned} \tag{1.9}$$

1.2.3 Extension to non-linear cases

Another popular approach to deal with non linearly separable data is to map them into a higher dimensional space also known as the *feature space*. The idea is then to construct the optimal separating hyperplane in this space, which henceforth reads:

$$\mathcal{M}^{\text{SVC}}(\boldsymbol{x}) = \boldsymbol{w}^T\Phi(\boldsymbol{x}) + b = \sum_{i=1}^{N}\alpha_i y_i \Phi(\boldsymbol{x}_i)^T \Phi(\boldsymbol{x}) + b, \tag{1.10}$$

where $\Phi(\bullet)$ is the mapping function.
The main idea, which is in the core of support vector machines, is to notice that the expansion in Eq. (1.10) only depends on the inner product of the vectors that are images in the feature space, *i.e.* $\Phi(\boldsymbol{x}_i)^T \Phi(\boldsymbol{x})$. Therefore, if one is able to directly compute this inner product there is no need to carry out any operation in the high-dimensional feature space, a cumbersome task. This operation is known as the *kernel trick*. Indeed, it can be shown that this computation may be directly carried out through a *kernel* function. Such kernel functions are constructed following some rules so as to respect the so-called Mercer's conditions which guarantee the existence of an underlying mapping (Cherkassky and Mulier, 2007).
Once a kernel function k is chosen, the expansion in Eq. (1.10) may be simply recast as:

$$\mathcal{M}^{\text{SVC}}(\boldsymbol{x}) = \sum_{i=1}^{N}\alpha_i\, y_i\, k(\boldsymbol{x}_i, \boldsymbol{x}) + b. \tag{1.11}$$

We will show in the next section how the optimization problem in Eq. (1.5) is affected by the introduction of the kernel.

1.2.4 Kernel function families

Examples of valid kernel functions are presented in Vapnik (1995). Among the most widely used, the following ones are implemented in UQLAB:

- Non-stationary linear:

$$k_{\text{lin-ns}}\left(\boldsymbol{x}, \boldsymbol{x}'\right) = \boldsymbol{x}^T\boldsymbol{x}'; \tag{1.12}$$

- Polynomial:

$$k_{\text{poly}}\left(\boldsymbol{x}, \boldsymbol{x}'\right) = \left(\boldsymbol{x}^T\boldsymbol{x}' + d\right)^p, \tag{1.13}$$

  where $d \geq 0$ and $p \in \mathbb{N}^*$ are the kernel parameters;

- Sigmoid:

$$k_{\text{sigmoid}}\left(\boldsymbol{x},\boldsymbol{x}'\right)=\tanh\left(\frac{\boldsymbol{x}^T\boldsymbol{x}'}{a}+b\right), \tag{1.14}$$

  where $a>0$ and $b\leq 0$ are the kernel parameters;

- Stationary linear:

$$k_{\text{lin-s}}\left(\boldsymbol{x},\boldsymbol{x}'\right)=\max\left(0,1-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|}{\sigma}\right). \tag{1.15}$$

- Gaussian:

$$k_{\text{Gaussian}}\left(\boldsymbol{x},\boldsymbol{x}'\right)=\exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{2\sigma^2}\right); \tag{1.16}$$

- Exponential:

$$k_{\text{exp}}\left(\boldsymbol{x},\boldsymbol{x}'\right)=\exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|}{\sigma}\right); \tag{1.17}$$

- Matérn $3/2$ and Matérn $5/2$:

$$\begin{aligned} k_{3/2}\left(\boldsymbol{x},\boldsymbol{x}'\right)&=\left(1+\sqrt{3}\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|}{\sigma}\right)\exp\left(-\sqrt{3}\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|}{\sigma}\right),\\ k_{5/2}\left(\boldsymbol{x},\boldsymbol{x}'\right)&=\left(1+\sqrt{5}\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|}{\sigma}+\frac{5}{3}\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{\sigma^2}\right)\exp\left(-\sqrt{5}\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|}{\sigma}\right), \end{aligned} \tag{1.18}$$

  where $\sigma>0$ corresponds to the characteristic length-scale.

Note: The most popular kernel for support vector machines is the Gaussian kernel. Once a kernel is chosen, the most important step is to properly fit the hyperparameters.

1.2.5 Anisotropic kernel functions

The expressions of the different kernel functions that were given in Section 1.2.4 correspond to the *isotropic* case. Generally speaking, a kernel function is called *isotropic* when it has the same behavior over all dimensions. In that sense, for each type of kernel function, the same parameter is used in every dimension. Alternatively, one may need to define a different parameter in each dimension. In UQLAB, *anisotropy* is implemented for the stationary kernels, *i.e.* linear, Gaussian, exponential, Matérn $3/2$ and Matérn $5/2$. The following form is used:

$$k\left(\boldsymbol{x},\boldsymbol{x}'\right)=\prod_{i=1}^{M}k_1\left(x_i,{x'}_i;\,\sigma_i\right), \tag{1.19}$$

where $\{\sigma_i, i=1,\ldots,M\}$ are the kernel parameters in each dimension and $k\left(x_i,{x'}_i,\sigma_i\right)$ are the kernel functions as defined in Eqs. (1.15),(1.16),(1.17) and (1.18).

1.3 Solving the SVC problem

Two versions of SVC are implemented in UQLABusing the linear and quadratic penalization schemes introduced above. The associated formulations, which are closely related, are introduced in the sequel in a matrix form (Gunn, 1998).

1.3.1 Linear penalization

The optimization problem introduced in Eq. (1.5) may be adapted so as to account for soft margin and feature space. This problem may be cast in a matricial form as follows:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T\left(\boldsymbol{K}\boldsymbol{Y}\boldsymbol{Y}^T\right)\boldsymbol{\alpha} + \boldsymbol{c}^T\boldsymbol{\alpha} \\ \text{subject to:} \quad & \boldsymbol{\alpha}^T\boldsymbol{Y} = 0, \qquad 0 \leq \alpha_i \leq C, \qquad i = \{1,\ldots,N\}, \end{aligned} \tag{1.20}$$

where $\boldsymbol{K}$ is the so-called Gram matrix of size $N \times N$ whose components read $K_{ij} = k\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)$ and $\boldsymbol{c} = \{-1,\ldots,-1\}^T$ is a column vector of length N.

The solution to this problem may be given by general-purpose quadratic programming solvers. Decomposition methods taking advantage of the sparsity of SVM have been widely used (Platt, 1999; Chang and Lin, 2005). Here we consider MATLAB built-in algorithms such as the interior-point (IP) (Vanderbei, 1994), the *sequential minimal optimization* (SMO) (Platt, 1999) and the *iterative single data algorithm* (ISDA) (Kecman et al., 2005). An important feature of the first approach is that the bias term b may be directly retrieved as a by-product of the results (Bompard, 2011). The latter two are algorithms that have been developed specifically for SVM. For problems with medium to large datasets or problems where sparsity is a desirable property, these two algorithms are expected to be more efficient than IP and should be used. For extremely large datasets, IP may fail due to the large size of the dense kernel matrix that needs to be computed.

It is worth noting here that once the coefficients α_i are found, it is possible to group them into two distinct families: unbounded and bounded support vectors. The former are defined such that $0 < \alpha_i < C$ whereas the latter correspond to $\alpha_i = C$. The distinction may also be done geometrically as the unbounded SVs are inside the margin whereas the bounded ones belong to the margin's boundary. We will show in the sequel why this information is relevant for the calibration of the SVC model.

1.3.2 Quadratic penalization

The second formulation considered in UQLAB implements a quadratic penalization of the classification errors (Chapelle et al., 2002b). The quadratic optimization problem reads as follow:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T\left(\widetilde{\boldsymbol{K}}\boldsymbol{Y}\boldsymbol{Y}^T\right)\boldsymbol{\alpha} + \boldsymbol{c}^T\boldsymbol{\alpha} \\ \text{subject to:} \quad & \boldsymbol{\alpha}^T\boldsymbol{Y} = 0, \qquad \alpha_i \geq 0, \qquad i = \{1,\ldots,N\}, \end{aligned} \tag{1.21}$$

where $\widetilde{\boldsymbol{K}} = \boldsymbol{K} + 1/C\boldsymbol{I}_N$, with $\boldsymbol{I}_N$ being the identity matrix of size $N \times N$. The main difference with the linear penalization is that the coefficients α_i are no more upper bounded by C. Besides, the Gram matrix is modified by introducing a regularization term in its diagonal. The same algorithms as in the linear case can be used to solve this problem.

1.4 Error estimation

The generalization capacity of the metamodel is usually assessed through resampling techniques. The basic idea is to partition the experimental design into *training* and *validation* sets. A popular approach is *K-fold cross validation*, which consists in partitioning the data into K random sets then using $(K-1)$ sets for training and the remaining one for validation. The procedure is repeated K times by changing the validation set and the resulting error is averaged, thus yielding a measure of the model accuracy. The particular case when $K = N$, also known as *leave-one-out* cross-validation, reads:

$$e_{\text{LOO}} = \frac{1}{N}\sum_{i=1}^{N} \Psi\left(-y_i \mathcal{M}_{\sim i}^{\text{SVC}}\left(\boldsymbol{x}_i\right)\right), \tag{1.22}$$

where $\mathcal{M}_{\sim i}^{\text{SVC}}$ denotes the SVC model built by withdrawing the point $(\boldsymbol{x}_i, y_i)$ from the training set and Ψ is the step function defined such that $\Psi(x) = 1$ if $x > 0$ and $\Psi(x) = 0$ otherwise.
When N is large, the computational cost becomes too important as a large number of models $\mathcal{M}_{\sim i}^{\text{SVC}}$ need to be built. Conveniently, bounds and approximations of leave-one-out errors have been developed. They allow one to estimate the leave-one-out error solely by post-processing the results of the model $\mathcal{M}^{\text{SVC}}$ without actually having to build N models. Here we consider the span estimate of the leave-one out error as developed by Vapnik and Chapelle (2000).

1.4.1 Span estimate of the LOO error

The span estimate of the LOO error reads (Vapnik and Chapelle, 2000):

$$\widehat{e}_{\text{LOO}} = \frac{1}{N}\sum_{i=1}^{N} \Psi\left(\alpha_i S_i^2 - 1\right), \tag{1.23}$$

where S_i^2 is the so-called *span* of the support vectors. Its computation involves the inversion of a matrix of size $(N_{\text{USV}} + 1) \times (N_{\text{USV}} + 1)$ for the linear penalization and of size $(N_{\text{SV}} + 1) \times (N_{\text{SV}} + 1)$ for the quadratic one, where N_{USV} and N_{SV} are respectively the number of unbounded support vectors and support vectors.
For the *linear penalization*, the span reads (Chapelle et al., 2002a):

$$S_i^2 = \begin{cases} \dfrac{1}{\left(\widetilde{\boldsymbol{K}}_{\text{usv}}^{-1}\right)_{ii}} & \text{if} \quad 0 < \alpha_i < C \\ k\left(\boldsymbol{x}_i, \boldsymbol{x}_i\right) - \widetilde{v}_i^T \boldsymbol{K}_{\text{usv}}^{-1} \widetilde{v}_i & \text{if} \quad \alpha_i = C, \end{cases} \tag{1.24}$$

where $\widetilde{\boldsymbol{K}}_{\text{usv}} = \begin{bmatrix} \boldsymbol{K}_{\text{usv}} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix}$, $\boldsymbol{K}_{\text{usv}} = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j\in\mathcal{I}_{\text{usv}}}$ and $\widetilde{\boldsymbol{v}}_i$ is the i-th column of $\boldsymbol{K}_{\text{usv}}$ with $\mathcal{I}_{\text{usv}}$ being the set of unbounded support vectors.

For the *quadratic penalization* case, the computation of the span is similar and reads (Chapelle et al., 2002b):

$$S_i^2 = \frac{1}{\left(\widetilde{\boldsymbol{K}}_{\text{sv}}^{-1}\right)_{ii}}, \tag{1.25}$$

with $\widetilde{\boldsymbol{K}}_{\text{sv}} = \begin{bmatrix} \widetilde{\boldsymbol{K}} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix}$.

1.4.2 Smoothed span estimate of the LOO error

An alternative formulation of the span has been developed by Chapelle et al. (2002b) in order to facilitate the use of gradient-based techniques for optimizing the hyperparameters of the classifier in high dimensional problems. This formulation is a smoother version which reads:

$$\widetilde{S}_i^2 = \frac{1}{\left(\widetilde{\boldsymbol{K}}_{\text{sv}} + \widetilde{\boldsymbol{D}}\right)^{-1}_{ii}} - \boldsymbol{D}_{ii}, \tag{1.26}$$

where $\widetilde{\boldsymbol{D}} = \begin{bmatrix} \boldsymbol{D} & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix}$, with $\boldsymbol{D}$ being a $N_{\text{SV}} \times N_{\text{SV}}$ diagonal matrix whose elements read $\boldsymbol{D}_{ii} = \eta/\alpha_i$ and $\boldsymbol{D}_{ii} = 0$ for $i, j \neq 0$.

The modified leave-one-out error estimate which is associated to this span therefore reads:

$$\widehat{e}_{\text{LOO}} = \frac{1}{N}\sum_{i=1}^{N} \Psi\left(\alpha_i \widetilde{S}_i^2 - 1\right). \tag{1.27}$$

1.4.3 *A posteriori* error estimation

After the metamodel is set up, its predictive accuracy on new data can be assessed by using the so-called *validation error*. It is calculated as the relative generalization error on an independent set of inputs and outputs $[\boldsymbol{\mathcal{X}}_{Val},\ \mathcal{Y}_{Val} = \mathcal{M}(\boldsymbol{\mathcal{X}}_{Val})]$:

$$\epsilon_{Val} = \frac{N-1}{N}\left[\frac{\sum_{i=1}^{N}\left(\mathcal{M}(\boldsymbol{x}_{Val}^{(i)}) - \mathcal{M}^{\text{SVC}}(\boldsymbol{x}_{Val}^{(i)})\right)^2}{\sum_{i=1}^{N}\left(\mathcal{M}(\boldsymbol{x}_{Val}^{(i)}) - \hat{\mu}_{Y_{Val}}\right)^2}\right] \tag{1.28}$$

where $\hat{\mu}_{Y_{Val}} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{M}(\boldsymbol{x}_{Val}^{(i)})$ is the sample mean of the validation set response. This error measure is useful to compare the performance of different surrogate models when evaluated on the same validation set. This measure is not used in the optimization algorithms for the hyperparameter calibration.

1.5 Hyperparameters calibration

1.5.1 SVC hyperparameters

SVM classifiers are constructed so as to have *generalization* properties: they shall be capable of predicting the class of new points with high accuracy. A crucial step in this respect is to find the optimal hyperparameters of the classifier by minimizing a generalization error metrics, such as the LOO error introduced above. Two parameters are of interest in SVM classification, namely:

- The *penalty term* C: this parameter trades off margin maximization and error minimization. The larger it is, the less misclassification errors are tolerated on the training points.
- The *kernel parameters*: their number depends on the type of kernel function that has been selected. These parameters are defined in Section 1.2.4. In the sequel, they will be gathered in the vector $\boldsymbol{\theta}$ of size N_θ.

In this manual, the set of hyperparameters is denoted by the vector $\gamma = \{C, \boldsymbol{\theta}\}$ of size $N_\gamma = N_\theta + 1$.

1.5.2 Optimization algorithm

When considering the optimization of the hyperparameters, one peculiar feature of support vector machines for classification shall be underlined: the error is a step function, as it somehow represents the ratio of misclassifications in the cross-validation or leave-one-out procedure. Hence, a wide number of hyperparameters combinations will result in the exact same error. To differentiate them, we consider a heuristics which consists in privileging models with lower complexity which is less prone to overfit. Herein, when different models give the same error, the one with the lowest number of support vectors is preferred. This can be translated by the following modified LOO error:

$$e_{\text{mLOO}} = e_{\text{LOO}}\, N + N_{sv}/N \tag{1.29}$$

Five optimization techniques for SVC calibration are considered in UQLAB:

- the interior-point method which is gradient-based ;
- the cross-entropy method for optimization (CE) (Kroese et al., 2006) which is detailed in the sequel ;
- the covariance matrix adaptation - evolution scheme (CMA-ES), a derandomized evolution strategy developed by (Hansen and Ostermeier, 2001) ;
- the genetic algorithm (GA) as provided by MATLAB ;

- the grid search, where each parameter is discretized in its definition space and a grid obtained by tensorization. This method may only be efficient when a low number of hyperparameters are considered, say 2 or 3.

Note: For practical purposes, the optimization is carried out in the $\log_2$ space for the penalty term C.

1.5.2.1 Cross-entropy method for optimization

The cross-entropy method is a global search algorithm which consists in iteratively sampling points in the search space so as to converge to the optimal solution. The implementation in UQLAB considers a normal distribution for sampling candidates. The algorithm is summarized below (Bourinet, 2015; Moustapha, 2016):

1. Initialize the algorithm as follows:
   - Define the initial PDF parameters, *i.e.* the mean $\boldsymbol{\mu}_{\boldsymbol{\gamma}}^{(0)} = \{C_0, \boldsymbol{\theta}_0\}^T$ and the standard deviation $\boldsymbol{\sigma}_{\boldsymbol{\gamma}}^{(0)}$;
   - Set the bounds of the search space $\boldsymbol{\gamma}_{\min}$ and $\boldsymbol{\gamma}_{\max}$ and define the convergence criterion;
   - Set internal parameters of the algorithm such as the number of points per generation N_{pop}, the smoothing parameters α^{CE}, β^{CE} and q^{CE} and the number of points in the elite sample $N_{el} = \lfloor \rho N_{\text{pop}} \rfloor$, where $\lfloor\,\rfloor$ denotes the floor function and ρ is a coefficient such that $0 < \rho < 1$;
   - Set $t = 1$.
2. Sample N_{pop} points $\left\{\boldsymbol{\gamma}_1, \boldsymbol{\gamma}_2, \ldots, \boldsymbol{\gamma}_{N_{\text{pop}}}\right\}$ following a truncated normal distribution $\mathcal{N}_{[\boldsymbol{\gamma}_{\min}, \boldsymbol{\gamma}_{\max}]}\left(\boldsymbol{\mu}_{\boldsymbol{\gamma}}^{(t-1)}, \boldsymbol{\sigma}_{\boldsymbol{\gamma}}^{(t-}\right.$
3. Train N_{pop} SVC models with hyperparameters $\boldsymbol{\gamma}_i, i = \left\{1, \ldots, N_{\text{pop}}\right\}$ and evaluate the associated error.
4. Select the N_{el} best models with respect to the computed error metrics. The associated parameters are denoted $\boldsymbol{\gamma}_{(1)}, \ldots, \boldsymbol{\gamma}_{(N_{el})}$.
5. Compute the component-wise mean and standard deviations of the elite sample (the following equations should be understood component-wise for each optimized parameter $\gamma_{,i}$ of $\boldsymbol{\gamma}$):

$$\begin{aligned} \widetilde{\boldsymbol{\mu}}_{\boldsymbol{\gamma}}^{(t)} &= \frac{1}{N_{el}} \sum_{j=1}^{N_{el}} \boldsymbol{\gamma}_{(j)}, \\ \widetilde{\boldsymbol{\sigma}}_{\boldsymbol{\gamma}}^{(t)} &= \sqrt{\frac{1}{N_{el}} \sum_{j=1}^{N_{el}} \left(\boldsymbol{\gamma}_{(j)} - \widetilde{\boldsymbol{\mu}}_{\boldsymbol{\gamma}}^{(t)}\right)^2}. \end{aligned} \tag{1.30}$$

6. Update the parameters of the normal distribution as follows:

$$\begin{aligned}\boldsymbol{\mu}_{\boldsymbol{\gamma}}^{(t)} &= \alpha^{\text{CE}}\widetilde{\boldsymbol{\mu}}_{\boldsymbol{\gamma}}^{(t)} + (1-\alpha^{\text{CE}})\boldsymbol{\gamma}_{\text{best}},\\ \boldsymbol{\sigma}_{\boldsymbol{\gamma}}^{(t)} &= \beta_t^{\text{CE}}\widetilde{\boldsymbol{\sigma}}_{\boldsymbol{\gamma}}^{(t)} + (1-\beta_t^{\text{CE}})\boldsymbol{\gamma}_{\text{best}},\end{aligned} \tag{1.31}$$

where

$$\beta_t^{\text{CE}} = \beta^{\text{CE}} + \beta^{\text{CE}}\,(1-1/t)^{q^{\text{CE}}} \tag{1.32}$$

is a dynamic smoothing parameter and $\boldsymbol{\gamma}_{\text{best}}$ is the best set of hyperparameters found so far.

7. Stop if convergence is achieved, otherwise set $t \leftarrow t+1$ and go to 2. The following convergence criteria are considered:

   - Number of stall generations: The algorithm stops if the number of successive iterations without sampling a point that improves the current best solution reaches a given threshold;
   - Stagnation of the cost: The algorithm stops if the relative change in the cost function values within a given number of iterations is below a given threshold;
   - Stagnation of the solution: The algorithm stops if the possible change in the solution becomes extremely small, *i.e.* the current normal distribution can only sample points extremely close to its mean.
   - Number of function evaluations: The algorithm stops if the number of calls to the cost function reaches a given threshold.

All the parameters of this algorithm are tunable. It is however recommended to keep the default parameters given in Chapter 3.

1.5.2.2 Covariance Matrix adaptation - Evolution scheme (CMA-ES)

The Covariance Matrix Adaptation - Evolution Strategy (CMA-ES) is a derandomized stochastic search algorithm introduced by Hansen and Ostermeier (2001). It proceeds by adapting the covariance matrix of a normal distribution such that directions that have improved the cost in the recent past iterations are more likely to be sampled again.
In the so-called (μ, λ)-CMA-ES strategy, the candidate solutions of the next generation consist of μ points sampled using a normal distribution considering only the λ best points of the current generation (Hansen and Kern, 2004; Hansen, 2001). Such a strategy is considered in UQLAB, with $\mu = N_{\text{pop}}$ and $\lambda = N_{\text{el}}$. The important steps of the algorithm are presented in the sequel. The actual implementation is more complex, the user can refer to Hansen (2001) for more details.

1. Initialize the algorithm:

   - Set the initial hyperparameters mean $\boldsymbol{\mu}_{\boldsymbol{\gamma}}^{(0)} = \{C_0, \boldsymbol{\theta}_0\}^T$ and corresponding standard deviation $\boldsymbol{\sigma}_{\boldsymbol{\gamma}}^{(0)}$ which will be referred to as the *global step size*;

- Initialize the covariance matrix $\boldsymbol{C}^{(0)} = \boldsymbol{I}_{N_\gamma \times N_\gamma}$ where N_γ is the number of hyperparameters to optimize;
- Set the internal CMA-ES parameters, *i.e.* the weight coefficients $\{w_i,\, i = 1,\, \ldots\, , \lambda\}$, the so-called *evolution path* $\boldsymbol{p}_c$ and the coefficients c_σ, d_σ, c_c, c_{cov} and μ_{cov};

  Note: All these parameters have been fine-tuned for CMA-ES (Hansen and Ostermeier, 2001). It is not advised to modify their default values.

- Set $t = 1$.

2. Sample $\mu = N_{\text{pop}}$ points $\left\{\boldsymbol{\gamma}_1, \boldsymbol{\gamma}_2, \ldots, \boldsymbol{\gamma}_{N_{\text{pop}}}\right\}$ such that

$$\boldsymbol{\gamma}_i = \boldsymbol{\mu}_\gamma^{(t-1)} + \boldsymbol{\varepsilon}^{(t-1)}; \tag{1.33}$$

   where $\boldsymbol{\varepsilon}^{(t-1)} \sim \mathcal{N}\left(\boldsymbol{0}, \left(\boldsymbol{\sigma}_\gamma^{(t-1)}\right)^T \cdot \boldsymbol{C}^{(t-1)} \cdot \boldsymbol{\sigma}_\gamma^{(t-1)}\right)$

3. Train N_{pop} SVC models with hyperparameters $\boldsymbol{\gamma}_i, i = \left\{1, \ldots, N_{\text{pop}}\right\}$ and evaluate the associated error;

4. Select the $\lambda = N_{el}$ best models with respect to the computed error metrics. The associated parameters are denoted $\boldsymbol{\gamma}_{(1)}, \ldots, \boldsymbol{\gamma}_{(\lambda)}$;

5. Update the mean of the distribution as follows:

$$\boldsymbol{\mu}_\gamma^{(t)} = \sum_{i=1}^{\lambda} w_i \boldsymbol{\gamma}_{(i)}; \tag{1.34}$$

6. Update the global step size

$$\boldsymbol{\sigma}_\gamma^{(t)} = \boldsymbol{\sigma}_\gamma^{(t-1)} \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\boldsymbol{p}_c^{(t)}\|}{\sqrt{N_\gamma}\left(1 - \frac{1}{4N_\gamma} + \frac{1}{21N_\gamma^2}\right)} - 1\right)\right), \tag{1.35}$$

   where

$$\boldsymbol{p}_c^{(t)} = (1 - c_c)\boldsymbol{p}_c^{(t-1)} + \sqrt{c_\sigma\,(2 - c_\sigma)}\,\boldsymbol{B}^{(t-1)}\boldsymbol{D}^{(t-1)}\boldsymbol{B}^{(t-1)}\frac{\sqrt{\mu_{eff}}}{\boldsymbol{\sigma}_\gamma^{(t-1)}}\left(\boldsymbol{\mu}_\gamma^{(t)} - \boldsymbol{\mu}_\gamma^{(t-1)}\right). \tag{1.36}$$

   $\boldsymbol{B}^{(t-1)}$ and $\boldsymbol{D}^{(t-1)}$ are obtained using a principal component analysis, *i.e.* $\boldsymbol{C}^{(t-1)} = \left(\boldsymbol{B}^{(t-1)}\right)^T \left(\boldsymbol{D}^{(t-1)}\right)^2 \boldsymbol{B}^{(t-1)}$ and $\mu_{eff} = 1/\sum_{i=1}^{\mu} w_i^2$ is the so-called *variance effective selection mass*;

7. Update the covariance matrix

$$\begin{aligned}\boldsymbol{C}^{(t)} = {} & (1 - c_{cov})\,\boldsymbol{C}^{(t-1)} + \frac{c_{cov}}{\mu_{cov}}\boldsymbol{p}_c^{(t)}\left(\boldsymbol{p}_c^{(t)}\right)^T \\ & + c_{cov}\left(1 - \frac{1}{\mu_{cov}}\right)\sum_{i=1}^{\mu}\frac{w_i}{\left(\boldsymbol{\sigma}_\gamma^{(t-1)}\right)^2}\left(\boldsymbol{\gamma}_{(i)} - \boldsymbol{\mu}_\gamma^{(t-1)}\right)\left(\boldsymbol{\gamma}_{(i)} - \boldsymbol{\mu}_\gamma^{(t-1)}\right)^T.\end{aligned} \tag{1.37}$$

8. Stop if convergence is achieved, otherwise set $t \leftarrow t + 1$ and go to 2. The following convergence criteria are considered:

   - Number of stall generations: The algorithm stops if the number of successive iterations without sampling a point that improves the current best solution reaches a given threshold;
   - Stagnation of the cost: The algorithm stops if the relative change in the cost function values within a given number of iterations is below a given threshold;
   - Stagnation of the solution: The algorithm stops if the possible change in the solution becomes extremely small, *i.e.* the current normal distribution can only sample points extremely close to its mean.
   - Number of function evaluations: The algorithm stops if the number of calls to the cost function reaches a given threshold.

Chapter 2

Usage

In this section, a reference problem will be set up to showcase how each of the techniques in Chapter 1 can be deployed in UQLAB.

2.1 Reference problem

The *Fisher's Iris data set*, a multivariate set of data widely used to test various classification algorithms, is considered. The data set contains 50 samples of three different species of the iris flower with four features. In this example, only two species (namely *virginica* and *versicolor*) and two features (sepal length and width) are considered. These data are loaded from MATLAB. They are labeled $y = -1$ for virginica and $y = +1$ for versicolor where the input $\boldsymbol{x}$ belongs to $\mathcal{D}_{\boldsymbol{X}} = [3,\, 7] \times [1,\, 2.5]$. The data are illustrated in Figure 2 where *virginica* and *versicolor* are plotted in blue dots and red squares respectively.

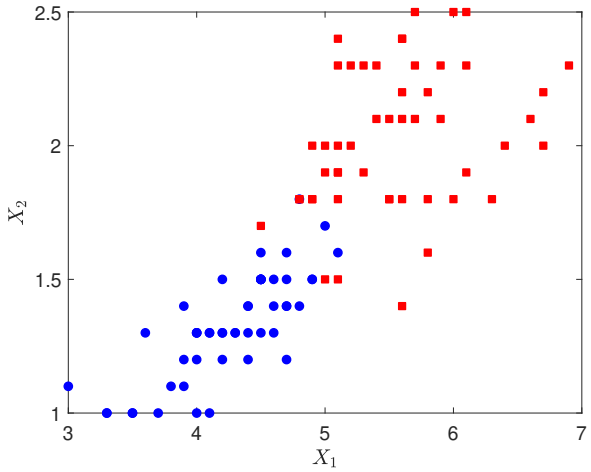


Figure 2: The Fisher's iris data with *virginica* and *versicolor* species respectively plotted with blue dots and red squares.

2.2 Problem set-up

The SVC module creates a MODEL object that can be used later on exactly as any other MODEL for classification. The basic options common to any SVC metamodel read:

```
MetaOpts.Type = 'Metamodel';
MetaOpts.MetaType = 'SVC';
```

As introduced in Eq. (1.11), the SVC prediction reads:

$$\mathcal{M}^{\text{SVC}}(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b, \tag{2.1}$$

where the coefficients α_i and the offset parameter b are solutions of the quadratic optimization problem given in Eq. (1.20) and Eq. (1.21) for linear and quadratic penalization respectively.
To solve these equations, the following minimal ingredients are needed:

- An experimental design $\mathcal{X}$ of size N, and the corresponding set of labels $\mathcal{Y} \in \{-1, 1\}^T$;
- A loss function;
- A kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$;
- Hyperparameters specification, *i.e.* the penalty term C and the kernel parameters $\boldsymbol{\theta}$;
- A solver for the quadratic optimization problem.

2.3 Fitting of the SVC metamodel

Default values are assigned within UQLAB to most of these parameters. In practice, only the experimental design is required. A minimal working example can be set up in UQLAB with the following code:

```
% Initialize the UQLab framework
uqlab;
% Retrieve training Iris Fisher's dataset X and Y
% The Fisher's iris data set is stored in a MAT-file
% in the following location:
FILELOCATION = fullfile(uq_rootPath, 'Examples', 'SimpleDataSets',...
'Fisher_Iris');
% Read the data set and store the contents in matrices:
load(fullfile(FILELOCATION,'fisher_iris_reduced.mat'), 'X','Y')

% Create the SVC metamodel
MetaOpts.Type = 'Metamodel';
MetaOpts.MetaType = 'SVC';
MetaOpts.ExpDesign.X = X;
MetaOpts.ExpDesign.Y = Y;

mySVC = uq_createModel(MetaOpts);
```

Note: When not specified by the user, the following default values are used:

- Loss function: Linear penalization;
- Kernel function: Isotropic Gaussian;
- Quadratic optimization solver: Interior-point (SMO if the size of the data set N is greater than 300);
- Error metric for hyperparameters calibration: Span estimate of the LOO ;
- Optimization algorithm for hyperparameters calibration: CMA-ES

Once the model is created, a summary of its main features can be printed using the command `uq_print`.

```
uq_print(mySVC);

%--------------- SVC metamodel ---------------%
Object Name:     Model 1
Input Dimension: 2

Experimental Design
Sampling:   User
X size:     [100x2]
Y size:     [100x1]

Loss function:      Linear
QP Solver:          Quadprog's IP
Kernel:             Gaussian

Hyperparameters
C:                    393.920139
Kernel params:        1.195202e+00
Estimation method:    Leave-one-out span estimate
Optim. method:        CMA-ES

Number of SVs (USV,BSV):    14   (10,4)
Leave-one-out error:        0.040000

%-------------------------------------------------%
```

For two-dimensional problems, the model can be illustrated using the `uq_display` command as shown in Figure 3. In this figure, the blue dots and red squares show respectively the training set with negative and positive labels. The contour $\left\{\boldsymbol{X} \in \mathcal{D}_{\boldsymbol{X}} : \mathcal{M}^{\text{SVC}}(\boldsymbol{X}) = 0\right\}$, which corresponds to the classifier, is shown by the black thick line. The margin corresponds to the area between the two dashed black lines. The support vectors which belong to this area, are shown by the green diamonds.

2.3.1 Accessing the results

The SVC model object is created using the options given by the user. When options are not provided, default parameters are set such that all the necessary information required to build the model is defined. The resulting parameters are structured in the SVC object, herein

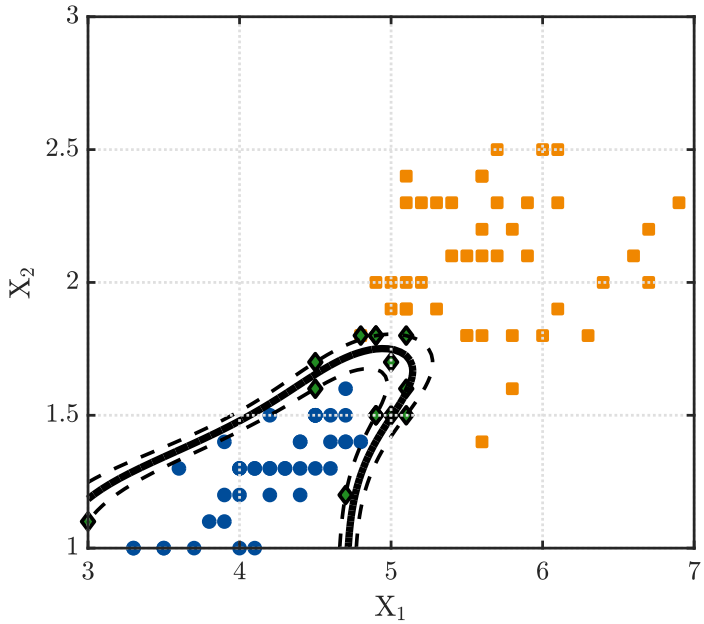


Figure 3: Illustration of the created SVC model for the Fisher's iris dataset: The hyperparameters are calibrated by minimizing the span estimate of the LOO error using CMA-ES.

`mySVC`, and can be accessed by the user.

SVC parameters

The basic results of the SVC metamodel, namely the hyperparameters values (Section Section 1.5.1) and the coefficients of the expansion in Eq. (1.11) are given in the field `mySVC.SVC`

```
mySVC.SVC

ans =

  struct with fields:

  Hyperparameters: [1x1 struct]
     Coefficients: [1x1 struct]
           Kernel: [1x1 struct]
```

The field `.Hyperparameters` contains the values of the penalty term C and kernel parameters:

```
mySVC.SVC.Hyperparameters

ans =

  struct with fields:

        C: 93.9201
    theta: 1.1952
```

The field `.Coefficients` contains the following information:

```
mySVC.SVC.Coefficients

ans =

  struct with fields:

     alpha: [100x1 double]
      beta: [100x1 double]
      bias: -8.3595
     SVidx: [14x1 double]
    USVidx:  [10x1 double]
    BSVidx: [4x1 double]
```

`alpha` is the vector containing the coefficients α_i of the model `bias` is the bias term b of the expansion, `beta` is an intermediate variable defined such that $\beta_i = \alpha_i\, y_i$, `SVidx`, `USVidx` and `BSVidx` are respectively the indices of the SVs, the unbounded SVs and the bounded SVs. Note that the latter two exist only when linear penalization is used (Section 1.3.1), otherwise no distinction is made between the support vectors.
Finally, `Kernel` is a structure containing the kernel handle, the kernel family and when applicable, the isotropy/anisotropy boolean. These values can be accessed as follows:

```
mySVC.SVC.Kernel

    Handle:  @uq_eval_Kernel
    Family: 'Gaussian'
    Isotropic: 1
```

Experimental design

The experimental design can be accessed in the substructure `mySVC.ExpDesign`. The following information is displayed:

```
mySVC.ExpDesign

ans =

         X: [100x2 double]
         Y: [100x1 double]
  Sampling: 'User'
  NSamples: 100
         U: [100x2 double]
```

The type and size of the experimental design are given respectively in the fields `ExpDesign.Sampling` and `ExpDesign.NSamples`. In the case when an INPUT object is used to create the experimental design, the related information is displayed in the field `ExpDesign.Input`. The remaining fields are:

- `ExpDesign.X`: the experimental design $\mathcal{X}$;
- `ExpDesign.Y`: the class labels corresponding to the inputs ;

- `ExpDesign.U`: The scaled experimental design. Refer to Section Section 2.9 for more information.

***A posteriori* error estimates**

The leave-one-out (LOO) error of the SVC model as estimated by Eq. (1.23) is stored in `mySVC.Error`:

```
mySVC.Error

  struct with fields:

LOO: 0.0400
```

2.4 Evaluating the model

To evaluate the model for a given set of points, the function `uq_evalModel` shall be used. In the following code, the built model is used to evaluate a set of validation data points `Xval`.

```
[Yclass, Yvalue] = uq_evalModel(mySVC,Xval);
```

The output `Yclass` is the label associated to a given input, *i.e.* $\text{sign}\left(\mathcal{M}^{\text{SVC}}(\boldsymbol{X})\right)$ whereas `Yvalue` is the SVC model prediction, *i.e.* $\mathcal{M}^{\text{SVC}}(\boldsymbol{X})$.
Note that the second output is optional. To obtain only the labels, the following command should be used:

```
Yclass = uq_evalModel(mySVC,Xval);
```

2.5 Set-up of the SVC metamodel

In the following subsections, the various configuration options of each of the ingredients of a SVC metamodel are presented.

2.5.1 Generation/Specification of the experimental design

2.5.1.1 Using already existing data

Depending on where the experimental design data is stored the following alternatives are offered:

- If data is stored in the MATLAB workspace, *e.g.* in the variables `X`, `Y`:

```
MetaOpts.ExpDesign.X = X;
MetaOpts.ExpDesign.Y = Y;
```

- If data is stored in a .mat file, *e.g.* `mydata.mat`:

```
MetaOpts.ExpDesign.DataFile = 'mydata.mat' ;
```

> **Note:** The experimental design that is contained in `mydata.mat` must be saved with variable names `X` and `Y` respectively.

The input and output dimensions M and N_{out} of the problem are automatically retrieved by the number of columns of `X` and `Y`, respectively.

2.5.1.2 Using INPUT and MODEL objects

In some applications, especially in uncertainty quantification, the user may be able to directly generate some data given their probability distribution. This is in opposition with the usual framework of machine learning where data are readily available. In the former case, UQLAB allows for the generation of data using a probabilistic INPUT vector. For instance, considering two-dimensional data uniformly distributed in the interval $[0,\ 10]$, an INPUT object can be created as follows:

```
IOpts.Marginals(1).Type = 'Uniform';
IOpts.Marginals(1).Parameters = [0 10];
IOpts.Marginals(2).Type = 'Uniform';
IOpts.Marginals(2).Parameters = [0 10];
myInput = uq_createInput(IOpts);
```

The input and output dimensions of the problem are automatically retrieved by the configuration of the INPUT object `myInput` and MODEL object `myModel`. For more information about the configuration options available for an INPUT and a MODEL object, please refer to the UQLAB User Manual – the INPUT module and the UQLAB User Manual – the MODEL module respectively.

Let us now assume that the classifier is defined as follows:

$$Y\left(\boldsymbol{X}\right) = \text{sign}\left(X_2 - X_1 \sin\left(X_1\right) - 1\right), \tag{2.2}$$

where the input $\boldsymbol{X} = \{X_1, X_2\}^T$ belongs to $\mathcal{D}_{\boldsymbol{X}} = [0, 10]^2$. Then the corresponding MODEL object can be created as follows:

```
MOpts.mString = 'sign(X_2 - X_1.*sin(X_1) - 1)' ;
myModel = uq_createModel(MOpts);
```

Now the user can include the INPUT and MODEL objects to the SVC metamodel configuration:

```
MetaOpts.Input = myInput;
MetaOpts.FullModel = myModel;
```

Finally, the user needs to define the size of the experimental design, *e.g.* for $N = 100$ samples:

```
MetaOpts.ExpDesign.NSamples = 100;
```

Note that, by default Monte Carlo sampling is used in order to obtain $\boldsymbol{X}$. However, other sampling strategies can be selected through the option `MetaOpts.ExpDesign.Sampling`, see Table 3 in Section 3.1.1 for a list of the available sampling strategies.

2.5.2 Penalization type

Two types of penalization are available in UQLAB, namely linear and quadratic. For instance, one can set the quadratic type with the following command:

```
MetaOpts.Penalization = 'quadratic' ;
```

Additional information is given in Table 2.

2.5.3 Kernel functions

2.5.3.1 Standard option

The main ingredients for specifying the kernel function are:

- **Family:** The most widely used families of kernel functions are implemented in UQLAB. An extensive list of the available kernel functions families can be found in Table 4. For instance, in order to use the Matérn-$5/2$ kernel, the following option is set:

  ```
  MetaOpts.Kernel.Family = 'Matern-5_2' ;
  ```

- **Isotropic:** This is a Boolean flag (with `true` or `false` value) that specifies whether the kernel function is isotropic or not. By default, the kernel function is considered isotropic, unless the following option is set:

  ```
  MetaOpts.Family.Isotropic = false ;
  ```

  Note that this option is only available for the radial-basis families of kernels, namely Gaussian, exponential, Matérn $3/2$ and Matérn $5/2$.

2.5.3.2 Advanced options

- **User-defined kernel family**

  Apart from the built-in kernel families described above, user-defined *one-dimensional* kernels can be used. They are expected to be of the form $k_1(x, x'; \theta)$ where x, x' are two input scalars and $\boldsymbol{\theta}$ is a vector of parameters. For example, to define the one-dimensional kernel family

  $$k_1(x, x'; \theta) = \exp\left[-\left(\frac{x - x'}{\theta}\right)^{3/2}\right],$$

  the following option should be provided:

  ```
  MetaOpts.Kernel.Family = @(x1,x2,th) exp(-((x1-x2)/th).^(3/2));
  ```

  Note: The user-defined kernel family is expected to be *vectorized*, *i.e.* it should be able to handle inputs `x1` and `x2` that are vectors with the same lengths.

- **User-specified routine to calculate the kernel matrix:** All the aforementioned options regarding the kernel function are handled by the function `uq_eval_Kernel`. If the user wants to use a fully custom kernel function in order to obtain the kernel matrix $\boldsymbol{K}$ the following rules need to be followed:

  - The function should return the kernel matrix $\boldsymbol{K}$ and accept x_1, x_2, θ and a structure of options like the following:

    ```
    function K = my_eval_K(x1,x2,theta,options)
    ...
    ```

  - Inputs `x1`, `x2` are matrices with arbitrary number of rows and M number of columns (where M is the input dimension).
  - When the custom kernel function `my_eval_K` is called, *e.g.* during the creation or usage of a SVC metamodel, the structure `options` will contain all options that were set inside `MetaOpts.Kernel`.
  - The input `theta` of the function `my_eval_K` corresponds to the hyperparameters $\boldsymbol{\theta}$ and it is expected to be a vector of arbitrary length. There is no limitation regarding its length but its dimension should be reflected in the choice of the initial value and/or optimization bounds (see Section 2.5.6.2 for more information about the optimization options).

  This user-defined function can be used for calculating the SVC metamodel as long as the following option has been set:

  ```
  MetaOpts.Kernel.Handle = @my_eval_K;
  ```

  An example with custom function using a mixed kernel function made of a linear combination of the Gaussian and polynomial families can be found in the example script `uq_Example_SVC_03_MixedKernel`.

2.5.4 Hyperparameters estimation methods

The hyperparameters estimation method affects the type of the objective function that is minimized to calibrate the hyperparameters. Currently, the leave-one-out and cross-validation methods are available for estimating the SVC parameters. They can be selected by setting `'SpanLOO'`, `'SmoothLOO'` or `'CV'` as the value of the field `MetaOpts.EstimMethod`.

Note: If not specified otherwise by the user, the span bound leave-one-out error (`'SpanLOO'`) is used by default.

2.5.4.1 Advanced Options

Smooth leave-one-out error

When the smooth leave-one-out error method is considered, it is used with its default

parameters. One can define a specific value for the regularization term η of Section 1.4.2 as follows:

```
MetaOpts.SmoothLOO.eta = eta_value;
```

where $\eta =$ `eta_value` is a positive real.

Cross-validation

The cross-validation option implements K-fold cross validation where the number of folds is by default $K = 3$. It can be set to leave-one-out or to any other arbitrary value of $K \geq 2$ with the following command:

```
MetaOpts.CV.Folds = K;
```

where K is an integer that should be smaller or equal to the size of the experimental design. The true leave-one-out error (in contrast to the span approximate) corresponds to $K = N$ where N is the experimental design size.

2.5.5 SVC quadratic problem solver

The coefficients of the SVC expansion are found by solving the quadratic optimization problem in Eq.(1.20) or (1.21). To set one algorithm, say SMO, one can use the following syntax:

```
MetaOpts.QPSolver = 'SMO';
```

Note: SMO and ISDA cannot be used with the quadratic penalization scheme. If they are selected, a warning will be issued and the program will proceed using interior-point convex algorithm.

Note: For large medium to large datasets ($N > 300$), the interior-point convex algorithm may be extremely slow or even fail to converge since it does not account for any sparsity and builds the Gram matrix at once using all the training points. It is advised to use SMO for medium to large dataset problems. This is set by default if the user does not specify the solver.

2.5.6 Hyperparameters calibration

2.5.6.1 Default values

Default values are given to the kernel parameters according to their types. For radial basis families, the default values depend on the experimental design and are computed as the average distance between the training points, which reads:

$$\sigma_0 = \frac{2}{N\,(N-1)} \sum_{j,k,j>k}^{N} d_{jk}, \tag{2.3}$$

where

$$d_{jk} = \|\boldsymbol{x}^{(j)} - \boldsymbol{x}^{(k)}\|, \quad j = 1,, \ldots ,, N, \quad k = 1,, \ldots ,, N, \tag{2.4}$$

Note: When the size of the training set is larger than 500, the average distance is computed on a random subset of the training set containing only 500 points

In case of anisotropy, the same parameter is computed independently for each dimension.
For the other hyperparameters, the default values are independent of the experimental design and are given values specified in Table 5.
These values are used to build a model when the user does not specify anything and does not carry out optimization of the hyperparameters (which can be done by setting the option `.Optim.Method` to `'none'` as will be shown in the sequel).
Instead of relying on the default values, one may also set values of the hyperparameters directly as follows:

```
MetaOpts.Hyperparameters.C = C_user;
MetaOpts.Hyperparameters.theta = theta_user;
```

For anisotropic kernels, the user may specify a scalar which will then be replicated in all dimensions, or a vector of size $1 \times M$ to define `theta_user`.
When using a polynomial kernel, the polynomial degree can be given using the following command:

```
MetaOpts.Hyperparameters.degree = 2;
```

If the value set is an array of integers, UQLAB will calibrate the model using each of the specified polynomial degrees and keep as final model the one with the lowest error. For instance, to calibrate a model with possible polynomial degrees going from 2 to 5, the following command can be used:

```
MetaOpts.Hyperparameters.degree = 2:5;
```

2.5.6.2 Optimization algorithms

Various configuration options are available regarding the method to solve the optimization problem used to calibrate the hyperparameters. The available optimization methods can be divided into three categories:

- **Gradient-based methods**

  Currently the BFGS method is available. In order to use this method the following syntax is used:

  ```
  MetaOpts.Optim.Method = 'BFGS';
  ```

  Additional options specific to BFGS can be set. For example, one can set the maximum number of iterations as follows:

```
MetaOpts.Optim.MaxIter = 10;
```

Note: It is not advised to use the BFGS algorithm with the current version of the LOO error which is a step function not suitable for gradient computation. BFGS can however be used when cross-validation is considered.

- **Global methods**

  Currently the Genetic Algorithm (GA), the cross-entropy method for optimization (CE), the covariance matrix adaptation - evolution scheme (CMA-ES) are available in UQLAB. To use one of these methods to solve the optimization of the hyperparameters, *e.g.* the cross-entropy method, one sets:

  ```
  MetaOpts.Optim.Method = 'CE';
  ```

  All the available methods are listed in Table Table 8. Additional options which are specific to each method exist. For example, when using CE, one might need to set a different value of stall generations, *e.g.* 20. This can be accomplished by setting:

  ```
  MetaOpts.Optim.CE.nStall = 20;
  ```

- **Grid search method**

  In this approach, a regular grid is created on the search space ($\log_2$-space for C). A SVC model is then built for each combination of hyperparameters and the one with the lowest error (according to the chosen estimation method) is selected as final model.

  To define the number of discretization points for each variable, one can use the following command:

  ```
  MetaOpts.Optim.GS.DiscPoints = 5;
  ```

  When a scalar is given, the same value is used in all directions. In contrast, the user can also provide a row vector, the size of which should be equal to the number of hyperparameters to optimize, in order to specify different values in each direction.

- **General options**

  Moreover, regardless of the method, the user can manually specify the following options:

  – The initial search point (not necessary for GA):

  ```
  MetaOpts.Optim.InitialValue.C = C_0 ;
  MetaOpts.Optim.InitialValue.theta = theta_0 ;
  ```

  – The number of iterations or generations (depending on the optimization method):

  ```
  MetaOpts.Optim.MaxIter = 100;
  ```

- The convergence tolerance:

```
MetaOpts.Optim.Tol = 1e-5;
```

- The verbosity of the optimization process, *e.g.* showing iteration information:

```
MetaOpts.Optim.Display = 'iter';
```

For a list of all the available options for each method, please refer to Table 8 in Section 3.1.5. A comparison of the resulting SVC models using different optimization (and estimation) methods can be found in the example script `uq_Example_SVC_02_VariousMethods`.

2.5.6.3 Default bounds of the search space

Most of the default hyperparameters values are independent of the experimental design and are given numeric values specified in Table 8. Only the radial-basis kernel functions have parameters which depend on the training set. They are defined by the following empirical rules:

- Lower bound:

$$\sigma_{\min} = \min_{j,k} d_{jk}/M, \tag{2.5}$$

  where d_{jk} is given by Eq. (2.4);

- Upper bound:

$$\sigma_{\max} = M \times \max_{j,k} d_{jk}, \tag{2.6}$$

  where d_{jk} is given by Eq. (2.4);

Note: For large data sets, the default settings of the UQLAB SVR module are modified in order to increase the calibration speed and favor sparsity. Hence when the experimental design is of size larger than 300 the following adaptations to the default values are made:

- the default QP solver becomes SMO;
- the upper bound of C becomes 2^{10}.

2.5.6.4 Specifying a subset of hyperparameters to calibrate

It is possible within UQLAB to calibrate a subset of hyperparameters while keeping the remaining ones to their user-given or default values. By default, all parameters are calibrated. To set off the calibration of one particular parameter, say the penalty term C, the following command should be used:

```
MetaOpts.Optim.Calibrate.C = false ;
```

2.6 Use of a validation set

If a validation set is provided (see Table 16 in Section 3.1.6), UQLAB automatically computes the validation error given in Eq. (1.28). To provide a validation set, the following command shall be used:

```
MetaOpts.ValidationSet.X = XVal;
MetaOpts.ValidationSet.Y = YVal;
```

The value of the validation error is stored in `mySVC.Error.Val` (see Table 23) and will also be displayed when typing `uq_print(mySVC)`.

2.7 SVC metamodels of vector-valued models

All the examples presented so far in this chapter dealt with scalar-valued models. In case the model (or the experimental design, if manually specified) produces multi-component outputs, UQLAB performs an independent SVC metamodel for each output component on the shared experimental design. No additional configuration is needed to enable this behaviour. A SVC example with multi-component outputs can be found in the UQLAB example script `uq_Example_SVC_04_MultipleOutputs`.

2.7.1 Accessing the results

Running a SVC calculation on a multi-component output model will result in a multi-component output structure. As an example, a model with 2 outputs will produce the following output structure:

```
mySVC.

  ans =

  1x2 struct array with fields:

    Hyperparameters
    Coefficients
    Kernel
```

Each element of the `SVC` structure is functionally identical to its scalar counterpart in Section 2.3.1. Similarly, the `mySVC.Error` structure becomes a multi-element structure:

```
mySVC.Error

  ans =

  1x2 struct array with fields:

  LOO
```

2.8 Using SVC with constant parameters

In some analyses, one may need to assign a constant value to one or more parameters. When this is the case, the SVC metamodel is designed as to internally remove the constant parameters from the inputs. This process is transparent to the users as they shall still evaluate the model using the full set of parameters (including those which were set constant). UQLAB will automatically and appropriately account for the set of input parameters which were declared constant.
To set a parameter to constant, the following command can be used (See UQLAB User Manual – the INPUT module):

```
inputOpts.Marginals.Type = 'Constant' ;
inputOpts.Marginals.Parameters = value;
```

Furthermore, when the standard deviation of a parameter is set to zero, UQLAB automatically sets this parameter's marginal to the type `Constant`. For example, the following uniformly distributed variable whose upper and lower bounds are identical is automatically set to a constant with value 1:

```
inputOpts.Marginals.Type = 'Uniform' ;
inputOpts.Marginals.Parameters = [1 1];
```

Note: A metamodel built with a given parameter set to constant shall not be evaluated using a different value of that parameter. It is however possible to build multiple metamodels by setting different constant values for some parameters.

2.9 Performing SVC on an Auxiliary space (Scaling)

The SVC module offers various options for scaling the experimental design before computing the metamodel.

2.9.1 Input scaling

Note: The SVC module uses by default the *scaled* experimental design $\mathcal{U}$ instead of the original values in $\mathcal{X}$.

Depending on the option `MetaOpts.Scaling` and whether a probabilistic input model has been defined (in `MetaOpts.Input`), the transformation $\mathcal{X} \mapsto \mathcal{U}$ varies. All the possible cases are summarised in Table 1.

Table 1: Available experimental design transformations		
`Input` / `Scaling`	No INPUT defined	INPUT defined
`1`	$\mathcal{U} = (\mathcal{X} - \mu_{\mathcal{X}})/\sigma_{\mathcal{X}}$	$\mathcal{U} = (\mathcal{X} - \mu_{\boldsymbol{X}})/\sigma_{\boldsymbol{X}}$
`0`	$\mathcal{U} = \mathcal{X}$	$\mathcal{U} = \mathcal{X}$
INPUT object	-	$\mathcal{U} = \mathcal{T}(\mathcal{X})$

By setting `MetaOpts.Scaling = 1`, the experimental design is scaled so that it has zero mean and unit variance. When no input distribution has been specified, $(\mu_{\mathcal{X}}, \sigma_{\mathcal{X}})$ are empirically estimated from the available data specified in `MetaOpts.ExpDesign.X`. When an input distribution has been specified via an INPUT object, say `myInput`, as follows:

```
MetaOpts.Input = myInput;
```

then $(\mu_{\boldsymbol{X}}, \sigma_{\boldsymbol{X}})$ are estimated from the moments of the marginal distribution of each component in $\boldsymbol{X}$.
When an additional INPUT object, say `myScaledInput`, is set as follows:

```
MetaOpts.Scaling = myScaledInput;
```

then $\mathcal{U} = \mathcal{T}(\mathcal{X})$ where $\mathcal{T}$ denotes the generalised *isoprobabilistic transformation* between the two probability spaces (for more information refer to the UQLAB User Manual – the INPUT module).

Chapter 3

Reference List

How to read the reference list

Structures play an important role throughout the UQLAB syntax. They offer a natural way to semantically group configuration options and output quantities. Due to the complexity of the algorithms implemented, it is not uncommon to employ nested structures to fine-tune the inputs and outputs. Throughout this reference guide, a table-based description of the configuration structures is adopted.
The simplest case is given when a field of the structure is a simple value or array of values:

Table X: `Input`			
●	`.Name`	String	A description of the field is put here

which corresponds to the following syntax:

```
Input.Name = 'My Input';
```

The columns, from left to right, correspond to the name, the data type and a brief description of each field. At the beginning of each row a symbol is given to inform as to whether the corresponding field is mandatory, optional, mutually exclusive, etc. The comprehensive list of symbols is given in the following table:

●	Mandatory
□	Optional
⊕	Mandatory, mutually exclusive (only one of the fields can be set)
⊞	Optional, mutually exclusive (one of them can be set, if at least one of the group is set, otherwise none is necessary)

When one of the fields of a structure is a nested structure, a link to a table that describes the available options is provided, as in the case of the `Options` field in the following example:

Table X: `Input`

●	.Name	String	Description
☐	.Options	Table Y	Description of the Options structure

Table Y: Input.Options			
●	.Field1	String	Description of Field1
☐	.Field2	Double	Description of Field2

In some cases, an option value gives the possibility to define further options related to that value. The general syntax would be:

```
Input.Option1 = 'VALUE1' ;
Input.VALUE1.Val1Opt1 = ...;
Input.VALUE1.Val1Opt2 = ...;
```

This is illustrated as follows:

Table X: Input			
●	.Option1	String	Short description
		'VALUE1'	Description of 'VALUE1'
		'VALUE2'	Description of 'VALUE2'
⊞	.VALUE1	Table Y	Options for 'VALUE1'
⊞	.VALUE2	Table Z	Options for 'VALUE2'

Table Y: Input.VALUE1			
☐	.Val1Opt1	String	Description
☐	.Val1Opt2	Double	Description

Table Z: Input.VALUE2			
☐	.Val2Opt1	String	Description
☐	.Val2Opt2	Double	Description

Note: In the sequel, double and doubles mean a real number represented in double precision and a set of such real numbers, respectively.

3.1 Create a support vector classification metamodel

Syntax

```
mySVC = uq_createModel(MetaOpts)
```

Input

The struct variable `MetaOpts` contains the configuration information for a support vector classification metamodel. The detailed list of available options is reported in Table 2.

Table 2: `MetaOpts`

●	`.Type`	`'Metamodel'`	Select the metamodeling tool
●	`.MetaType`	`'SVC'`	Select support vector machines for classification
☐	`.Name`	String	Unique identifier for the metamodel
☐	`.Display`	Sring default: `'standard'`	Level of information displayed by the methods.
		`'quiet'`	Minimum display level, displays nothing or very few information.
		`'standard'`	Default display level, shows the most important information.
		`'verbose'`	Maximum display level, shows all the information on runtime, like updates on iterations, etc.
☐	`.ExpDesign`	Table 3	Experimental design options
☐	`.Input`	String	The name of the INPUT object that describes the inputs of the metamodel
		INPUT object	The INPUT object that describes the inputs of the metamodel
☐	`.FullModel`	String	The name of the MODEL object that is used to calculate the output data
		MODEL object	The MODEL object that is used to calculate the output data
☐	`.Penalization`	Sring default: `'linear'`	Options of the soft margin classifier (Section 1.3)
		`'linear'`	Linear penalization
		`'quadratic'`	Quadratic penalization
☐	`.Kernel`	Table 4	Kernel function used to compute the Gram matrix (Section 1.2.4)

☐	.Hyperparameters	Table 5	Values of the hyperparameters (Section 1.5.1)
☐	.QPSolver	Sring default: 'IP'	Quadratic programming algorithm used to estimate the coefficients of the expansion (Eqs. (1.20) and (1.21))
		'IP'	Interior-point method
		'SMO'	Sequential minimal optimization using MATLAB's fitcsvm (becomes the default when the experimental design size $N > 300$)
		'ISDA'	Iterative single data algorithm using MATLAB's fitcsvm
☐	.Scaling	Logical / Integer default: true	An integer defining whether the input data should be scaled or not, see details in Table 1
		true / 1	Scale the input
		false / 0	Do not scale the input
☐	.EstimMethod	Sring default: 'SpanLOO'	Hyperparameters estimation method (Section 1.5.1)
		'SpanLOO'	Minimization of the span estimate of the leave-one-out error (Eq. (1.23))
		'SmoothLOO'	Minimization of a smoothed span estimate of the leave-one-out error (Eq. (1.27))
		'CV'	Minimization of a K-fold cross validation generalization error
☐	.SmoothLOO	Table 6	Smooth LOO span estimate related options
☐	.CV	Table 7	Cross-validation related options
☐	.Optim	Table 8	Hyperparameters optimization-related options (Section 1.5.2)
☐	.ValidationSet	Table 16	Validation set components (Section 2.6)

3.1.1 Experimental design options

Table 3: MetaOpts.ExpDesign

☐	.Sampling	Sring default: 'MC'	Sampling type
		'MC'	Monte Carlo sampling
		'LHS'	Latin Hypercube sampling

		`'Sobol'`	Sobol sequence sampling
		`'Halton'`	Halton sequence sampling
⊞	`.DataFile`	String	A string containing the name of the mat file that contains the Experimental design.
⊞	`.X`	$N \times M$ Double	User defined model input X.
⊞	`.Y`	$N \times O$ Double	User defined class labels Y.
☐	`.NSamples`	Integer	The number of samples to draw

3.1.2 Kernel function options

Table 4: `MetaOpts.Kernel`			
☐	`.Family`	String or function handle default: `'Gaussian'`	The kernel family, *i.e.* the elementary 1-D function that is used by the kernel function. For a description of each family, refer to Section 1.2.4
		`'Linear-NS'`	Non-stationary linear kernel function
		`'Polynomial'`	Polynomial kernel function
		`'Sigmoid'`	Sigmoid kernel function
		`'Linear'`	Stationary linear kernel function
		`'Exponential'`	Exponential kernel function.
		`'Gaussian'`	Gaussian kernel function
		`'Matern-3_2'`	Matérn-3/2 kernel function
		`'Matern-5_2'`	Matérn-5/2 kernel function
		function handle	The user defined kernel function handle
☐	`.Isotropic`	Logical default: `true`	Determines whether the kernel function is isotropic or anisotropic (only valid for radial basis function families of kernel)
☐	`.Handle`	Function handle default: `@uq_SVC_eval_K`	The handle of the function that is used to calculate the kernel matrix $\boldsymbol{K}$ (Section 2.5.3.2)

3.1.3 Hyperparameters specification

Table 5: MetaOpts.Hyperparameters			
☐	.C	Double default: 10	Penalty term
☐	.theta	Double	Kernel function parameter. • For stationary kernels, the default value depends on the experimental design (Section 2.5.6.1) • For the non-stationary linear kernel, there is no hyperparameter Eq. (1.12) • For the polynomial kernel, the defaults are $d = 1$ and $p = 2$ (Eq. (1.13)) • For the sigmoid kernel, the default is $a = 1$ and $b = -1$ (Eq. (1.14))

3.1.4 Estimation method options

Table 6: MetaOpts.SmoothLOO			
☐	.eta	Positive Double default: 0.1	Regularization term η for the computation of the span following Eq. (1.26)

Table 7: MetaOpts.CV			
☐	.Folds	Positive integer default: 3	• Number of folds for the cross-validation procedure. It can be any integer between 2 and N • The case with the value N corresponds to the well-known leave-one-out error

3.1.5 Hyperparameters optimization options

Table 8: MetaOpts.Optim			
☐	.Method	String default: 'CMAES'	Optimization algorithm for the hyperparameters calibration (Section 1.5.2)
		'none'	No selected optimization algorithm. Default or user-given values of the hyperparameters will be considered
		'CE'	Cross-entropy method for optimization

		`'CMAES'`	Covariance matrix adaptation - evolution scheme
		`'GS'`	Grid search
☐	`.InitialValue`	Table 9	A structure containing the initial values of the hyperparameters (See Section 1.5.1)
☐	`.MaxIter`	Integer default: 10	Maximum number of iterations allowed in the optimization algorithms
☐	`.Bounds`	Table 10	A structure containing the bounds of the search space for the calibration of the hyperparameters
☐	`.Calibrate`	Table 11	A structure containing flag specifying which hyperparameters should be calibrated (by default all parameters are calibrated)
☐	`.Display`	String default: Matches with global display level	Level of information displayed by the methods.
		`'none'`	Minimum display level, displays nothing.
		`'iter'`	Maximum display level, shows detailed information in each iteration of the algorithm
		`'final'`	Shows information regarding only the convergence of the algorithm, if any.
☐	`.Tol`	Double default: `1e-3`	Termination tolerance on the objective function Only applies when gradient-based algorithm is considered
⊞	`.CMAES`	Table 12	Options relevant to the CMA-ES optimization method
⊞	`.CE`	Table 14	Options relevant to the CE optimization method
⊞	`.GS`	Table 15	Options relevant to the grid-search algorithm

Table 9: `MetaOpts.Optim.InitialValue`

☐	`.C`	Positive Double	Initial value of the penalty term • For BFGS, the default value is given by Table 5 • For the other algorithms, the default value is the center of the search space

☐	`.theta`	$1 \times N_\theta$ Double	Initial values of the kernel parameters • For BFGS, the default value is given in Table 5 • For the other algorithms, the default value is the center of the search space

Table 10: `MetaOpts.Optim.Bounds`

☐	`.C`	2×1 Double default: `[1; 2^16]`	A vector in the form `[lower_C; upper_C]` • The default upper bound is reduced to 2^{10} when the sample size is larger than 300 (See Note in Section 2.5.6.3)
☐	`.theta`	$2 \times N_\theta$ Double	A vector in the form `[lower_theta; upper_theta]` • Defaults for radial basis families of kernel (σ): lower bound using Eq. (2.5), upper bound using Eq. (2.6) • Default for polynomial kernel: Lower d: 1, upper d: 10 (Eq. (1.13)) • Default for sigmoid kernel: Lower a: 10^{-1}, upper a: 10, lower b: -5 and upper b: 0 (Eq. (1.14))

Table 11: `MetaOpts.Optim.Calibrate`

☐	`.C`	Logical default: `true`	A logical defining whether the penalty term C should be calibrated or not
☐	`.theta`	Logical default: `true`	A logical defining whether the kernel parameters $\boldsymbol{\theta}$ should be calibrated or not

Table 12: `MetaOpts.Optim.CMAES`

☐	`.nPop`	Positive integer default: $5 \times (\lfloor 4 + 3 \log M \rfloor)$	The population size N_{pop} of each generation (See μ in Section 1.5.2.2)
☐	`.ParentNumber`	Positive integer default: $\lfloor N_{pop}/2 \rfloor$	Number of points in the current population selected to generate the offsprings (See λ in Section 1.5.2.2)

☐	`.sigma`	$1 \times N_\gamma$ Double default: $\boldsymbol{\sigma}_0 = (\boldsymbol{\gamma}_{\max} - \boldsymbol{\gamma}_{\min})/3$	Initial value of the global step size of the CMA-ES algorithm (See Section 1.5.2.2) • The given vector should concatenate the values for all the hyperparameters, *i.e.* in the form `[sigma_C, sigma_theta]` • This should be consistent with the actual parameters to optimize: if `.Calibrate.C = false`, then `.sigma` is in the form `sigma_theta`.
☐	`.nStall`	Positive integer default: $5 \times \lceil 30 N_\gamma / N_{pop} \rceil$	The maximum number of stall generations
☐	`.TolFun`	Double default: 1e-2	Termination tolerance on the cost function: The algorithm stops if the average relative change in the best cost value over a given number of generations is less than or equal to `.TolFun`
☐	`.TolX`	Double default: 1e-2	Termination tolerance on X: The algorithm stops if the average relative change in the best solution over a given number of generations is less than or equal to `.TolX`
☐	`.FvalMin`	Double default: 0	Termination tolerance on fitness function: The algorithm stops if the fitness function value in a given iteration is less or equal to `.FvalMin`

Table 13: `MetaOpts.Optim.GA`

☐	`.nPop`	Positive integer default: $\min(20, \lfloor 4 + 3 \log N_\gamma \rfloor)$	The population size of each generation
☐	`.nStall`	Positive integer default: 2	The maximum number of stall generations

Table 14: `MetaOpts.Optim.CE`

☐	`.nPop`	Positive integer default: 50	The population size N_{pop} of each generation (See Section 1.5.2.1)
☐	`.qElite`	Double (between 0 and 1) default: 0.05	Ratio of points in the current population selected to generate the offsprings (ρ in Section 1.5.2.1)

☐	`.sigma`	$1 \times N_\gamma$ Double default: $\boldsymbol{\sigma}_0 = (\boldsymbol{\gamma}_{\max} - \boldsymbol{\gamma}_{\min})/3$	Initial value of the global step size of the CE algorithm (See Section 1.5.2.1) • The given vector should concatenate the values for all the hyperparameters, *i.e.* in the form `[sigma_C, sigma_theta]` • This should be consistent with the actual parameters to optimize: if `.Calibrate.C = false`, then `.sigma` is in the form `sigma_theta`.
☐	`.nStall`	Positive integer default: 2	The maximum number of stall generations
☐	`.TolFun`	Double default: 1e-2	Termination tolerance on the fitness function: The algorithm stops if the average relative change in the best fitness function value over `.nStall` generations is less than or equal to `.TolFun`
☐	`.TolSigma`	Double or $1 \times N_\gamma$ Double default: 1e-2	Termination tolerance on the global step length: The algorithm stops if the ratio between the current global step length and the initial one (`.sigma`) is below or equal to `.TolFun`
☐	`.alpha`	Double default: 0.4	Smoothing parameter of the updating scheme (α^{CE} in Eq. (1.31))
☐	`.beta`	Double default: 0.4	Smoothing parameter of the updating scheme (β^{CE} in Eq. (1.32))
☐	`.q`	Double default: 10	Smoothing parameter of the updating scheme (q^{CE} in (1.32))

Table 15: `MetaOpts.Optim.GS`

☐	`.DiscPoints`	Integer or 1×Number of hyperparameters integer default: 5	The number of discretization points in each direction needed to build the grid • When a scalar is given, the same value is assumed in all directions

3.1.6 Validation Set

If a validation set is provided, UQLAB automatically calculates the validation error for the created SVC model. The required information is listed in Table 16.

Table 16: `MetaOpts.ValidationSet`

●	`.X`	$N \times M$ Double	User-specified validation set $\boldsymbol{\mathcal{X}}_{Val}$

●	`.Y`	$N \times N_{Out}$ Double	User-specified validation set response $\mathcal{Y}_{Val}$

3.2 Accessing the results

Syntax

```
mySVC = uq_createModel(MetaOpts) ;
```

Output

Regardless of the configuration options given at creation time in the `MetaOpts` structure, all SVC metamodels share the same output structure, given in Table 17.

Table 17: `mySVC`		
`.Name`	String	Unique name of the SVC metamodel
`.Options`	Table 2	Copy of the `MetaOpts` structure used to create the metamodel
`.SVC`	Table 18	Information about the final model
`.ExpDesign`	Table 22	Experimental design used for calculating the coefficients
`.Error`	Table 23	Error estimates of the metamodels's accuracy
`.Internal`	Table 24	Internal state of the MODEL object (useful for debug/diagnostics)

Table 18: `mySVC.SVC`		
`.Hyperparameters`	Table 19	The values the hyperparameters of γ (See Section 1.5.1)
`.Coefficients`	Table 20	The values of the different coefficients in the expansion in Eq. (1.11)
`.Kernel`	Table 21	Information about the kernel function

Table 19: `mySVC.SVC.Hyperparameters`		
`.C`	Double	The value of the penalty term C (See Section 1.5.1)
`.theta`	Variable size double	The value of the kernel hyperparameters (See Section 1.2.4)

Table 20: `mySVC.SVC.Coefficients`		
`.alpha`	$N \times 1$ Double	The value of the coefficients of the expansion in Eq. (1.11)

.beta	$N \times 1$ Double	The value of the vector $\boldsymbol{\alpha y}$ in Eq. (1.11)
.bias	Double	The value of the bias term in Eq. (1.11)
.SVidx	Variable size double	The index of the support vectors (non-zero coefficients)

Table 21: mySVC.SVC.Kernel		
.Handle	function handle	The user defined kernel family function handle (See Section 2.5.3.2)
.Family	String or function handle	The chosen kernel family, *i.e.* the elementary 1-D function that is used by the kernel function (See Section 1.2.4)
.Isotropic	Logical	A logical value showing whether the kernel is isotropic or not (only available for radial basis families of kernels)

Table 22: mySVC.ExpDesign		
.NSamples	Integer	The number of samples
.Sampling	String	The sampling method
.Input	INPUT object	The input module that was used to generate the experimental design (X)
.X	$N \times M$ Double	The experimental design values
.U	$N \times M$ Double	The experimental design values in the reduced space (See Section 2.9.1)
.Y	$N \times N_{out}$ Double	The output Y that corresponds to the input X

Table 23: mySVC.Error		
.LOO	Double	The Leave-One-Out error (Eq. (1.23))
.Val	Double	Validation error (see Eq. (1.28) and Section 2.6). Only available if a validation set is provided (see Table 16).

Note: In general the fields mySVC.SVC and mySVC.Error are structure arrays with length equal to the number of outputs N_{out} of the metamodel.

Table 24: mySVC.Internal		
.Runtime	Structure	Variables that are used during the calculation of the SVC metamodel
.SVC	Structure	All the parameters set and calculated to completely define the generated SVC model

`.AuxSpace`	INPUT object	Auxiliary space where SVC is performed (See Section 2.9.1)

Note: The internal fields of the SVC module are not intended to be accessed or changed for most typical usage scenarios of the module.

3.3 Evaluating the model

Syntax

```
Y_class = uq_evalModel(X)
Y_class = uq_evalModel(mySVC, X)
[Y_class, Y_val] = uq_evalModel(...)
```

Description

`Y_class = uq_evalModel(X)` returns the class of the SVC prediction ($N \times N_{\text{out}}$ Double) on the points `X` ($N \times M$ Double) using the last created SVC model.

> **Note:** by default, the *last created* model or surrogate model is the currently active model.

`Y_class = uq_evalModel(mySVC,X)` returns the class of the SVC prediction ($N \times N_{\text{out}}$ Double) on the points `X` ($N \times M$ Double) using the SVC model object `mySVC`.

`[Y_class, Y_val] = uq_evalModel(...)` additionally returns the value of the SVC predictor ($N \times N_{\text{out}}$ Double).

References

Bompard, M. (2011). *Modèles de substitution pour l'optimisation globale de forme en aérodynamique et méthode locale sans paramétrisation*. PhD thesis, Université de Nice Sophia-Antipolis, France. 6

Bourinet, J.-M. (2015). Reliability assessment with adaptive surrogates based on support vector machine regression. In M. Papadrakakis, V. Papadopoulos, G. S. e., editor, *Proc. 1st ECCOMAS Thematic Conference on Uncertainty Quantification in Computational Sciences and Engineering, Crete Island, Greece, May 25-27*. 10

Bourinet, J.-M. (2018). *Reliability analysis and optimal design under uncertainty - Focus on adaptive surrogate-based approaches*. Université Blaise Pascal, Clermont-Ferrand, France. Habilitation à diriger des recherches, 243 pages. 1

Chang, M.-W. and Lin, C.-J. (2005). Leave-one-out bounds for support vector regression model selection. *Neural Computation*, 17(5):1188–1222. 6

Chapelle, O., Vapnik, V., and Bengio, Y. (2002a). Model selection for small sample regression. 48(1):9–23. 7

Chapelle, O., Vapnik, V., Bousquet, O., and Mukherje, S. (2002b). Choosing multiple parameters for support vector machines. In Cristianini, N., editor, *Machine Learning*, volume 46, pages 131 – 159. 6, 8

Cherkassky, V. and Mulier, F. (2007). *Learning from data: concepts, theory, and methods*. Wiley. 1, 4

Cortes, C. and Vapnik, V. (1995). Support vector networks. In *Machine Learning*, pages 273–297. 3

Gunn, S. (1998). Support vector machines for classification and regression. Technical Report ISIS-1-98, Dpt. of Electronics and Computer Science, University of Southampton. 2, 6

Hansen, N. (2001). The CMA evolution strategy: A tutorial. Technical report, Institut National de Recherche en Informatique et en Automatique (INRIA), France. 11

Hansen, N. and Kern, S. (2004). Evaluating the CMA evolution strategy on multimodal test functions. *Evolutionary Computation*, 9(2):282–221. 11

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195. 9, 11, 12

Kecman, V., Huang, T.-M., and Vogt, M. (2005). *Iterative single data algorithm for training kernel machines from huge data sets: theory and performance*, pages 255–274. Springer Berlin Heidelberg. 6

Kroese, D. P., Porotsky, S., and Rubinstein, R. Y. (2006). The cross-entropy method for continuous multi-extremal optimization. *Methodology and Computing in Applied Probability*, 8:383–407. 9

Moustapha, M. (2016). *Adaptive surrogate models for the reliable lightweight design of automotive body structures*. PhD thesis, Université Blaise Pascal, Clermont-Ferrand, France. 2, 10

Platt, J. C. (1999). Fast training of support vector machines using sequential minimal optimization. In Schölkopf, B., Burges, C. J. C., and Smola, A. J., editors, *Advances in kernel methods*, pages 185–208. MIT Press. 6

Smola, A. and Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and Computing*, 14:199–222. 1

Vanderbei, R. J. (1994). LOQO: an interior point code for quadratic programming. Technical report, Optimization Methods and Software. 6

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag, New York. 1, 3, 4

Vapnik, V. and Chapelle, O. (2000). Bounds on error expectation for support vector machines. *Neural Computation*, 12(9):2013–2036. 7