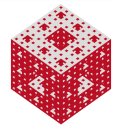
MATH 590: Meshfree Methods

Machine Learning

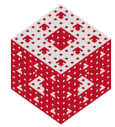
Greg Fasshauer

Department of Applied Mathematics
Illinois Institute of Technology

Fall 2014



Outline



Outline

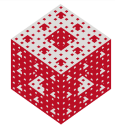
Until now

Given data

$$\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N, \qquad \boldsymbol{x}_i \in \Omega \subset \mathbb{R}^d, \quad y_i \in \mathbb{R},$$

find a function s that predicts, for a previously unobserved $\boldsymbol{x}$ value,

$$s(\boldsymbol{x}) \approx y.$$



Until now

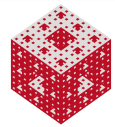
Given data

$$\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N, \qquad \boldsymbol{x}_i \in \Omega \subset \mathbb{R}^d, \quad y_i \in \mathbb{R},$$

find a function s that predicts, for a previously unobserved $\boldsymbol{x}$ value,

$$s(\boldsymbol{x}) \approx y.$$

Scattered data interpolation: Construct s as a linear combination of "shifts" of kernels such that $\|\boldsymbol{s} - \boldsymbol{y}\| = 0$, where $\boldsymbol{y} = (y_1, \ldots, y_N)^T$ and $\boldsymbol{s} = (s(\boldsymbol{x}_1), \ldots, s(\boldsymbol{x}_N))^T$.



Until now

Given data

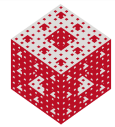
$$\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N, \qquad \boldsymbol{x}_i \in \Omega \subset \mathbb{R}^d, \quad y_i \in \mathbb{R},$$

find a function s that predicts, for a previously unobserved $\boldsymbol{x}$ value,

$$s(\boldsymbol{x}) \approx y.$$

Scattered data interpolation: Construct s as a linear combination of "shifts" of kernels such that $\|\boldsymbol{s} - \boldsymbol{y}\| = 0$, where $\boldsymbol{y} = (y_1, \ldots, y_N)^T$ and $\boldsymbol{s} = (s(\boldsymbol{x}_1), \ldots, s(\boldsymbol{x}_N))^T$.

Kriging: Construct s such that $s(\boldsymbol{x}) = \hat{y}_{\boldsymbol{x}} = \mathbb{E}[Y_{\boldsymbol{x}} | \boldsymbol{Y} = \boldsymbol{y}]$, where the data is realized by a Gaussian random field Y with specified covariance K.



Until now

Given data

$$\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N, \qquad \boldsymbol{x}_i \in \Omega \subset \mathbb{R}^d, \quad y_i \in \mathbb{R},$$

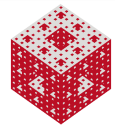
find a function s that predicts, for a previously unobserved $\boldsymbol{x}$ value,

$$s(\boldsymbol{x}) \approx y.$$

Scattered data interpolation: Construct s as a linear combination of "shifts" of kernels such that $\|\boldsymbol{s} - \boldsymbol{y}\| = 0$, where $\boldsymbol{y} = (y_1, \ldots, y_N)^T$ and $\boldsymbol{s} = (s(\boldsymbol{x}_1), \ldots, s(\boldsymbol{x}_N))^T$.

Kriging: Construct s such that $s(\boldsymbol{x}) = \hat{y}_{\boldsymbol{x}} = \mathbb{E}[Y_{\boldsymbol{x}} | \boldsymbol{Y} = \boldsymbol{y}]$, where the data is realized by a Gaussian random field Y with specified covariance K.

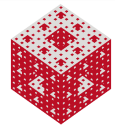
Truncated Mercer series: Construct s as a linear combination of only $M < N$ eigenfunctions such that $\min_{\boldsymbol{s}} \|\boldsymbol{s} - \boldsymbol{y}\|$.



We now consider ill-posed problems

A problem may be ill-posed if, e.g.,

- we don't have enough data to capture the complexity of the model,
- we don't have enough complexity in our model to match the data,
- we don't want to match all the complexity of the data because some of it might be due to measurement errors.



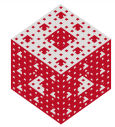
We now consider ill-posed problems

A problem may be ill-posed if, e.g.,

- we don't have enough data to capture the complexity of the model,
- we don't have enough complexity in our model to match the data,
- we don't want to match all the complexity of the data because some of it might be due to measurement errors.

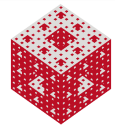
In such cases our earlier approaches need to be modified and one typically solves the data fitting problem via a regularization approach. We now

- give a overview to such a general regularization strategy to fitting data,
- look at RBF network regression,
- support vector machine (SVM) classification and
- SVM regression.



Each of our algorithms will involve its own particular

- loss function
- coupled with an appropriate regularization term with the help of a regularization parameter $\mu > 0$.

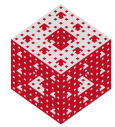


Each of our algorithms will involve its own particular

- loss function
- coupled with an appropriate regularization term with the help of a regularization parameter $\mu > 0$.

The discussion below is quite brief.

Many more details can be found in specialized books or survey papers on machine learning or statistical learning such as, e.g., [EPP00, HTF09, RW06, SS02, STC04, SC08].



Loss function

A typical loss function L depends on

- an input measurement $\boldsymbol{x}$,
- its associated value y
- and a value $s(\boldsymbol{x})$ predicted by the learning algorithm.

Loss function

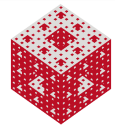
A typical loss function L depends on

- an input measurement $\boldsymbol{x}$,
- its associated value y
- and a value $s(\boldsymbol{x})$ predicted by the learning algorithm.

The goal of the training phase of the machine learning algorithm is to determine the predictor s such that the empirical risk

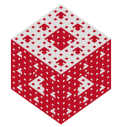
$$R_L = \frac{1}{N} \sum_{i=1}^{N} L\left(y_i, s(\boldsymbol{x}_i)\right)$$

is minimized.



Regularization

The regularization functional frequently measures the smoothness of the predictor of s.

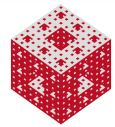


Regularization

The regularization functional frequently measures the smoothness of the predictor of s.

Remark

The regularization term can also be interpreted as a measure of the complexity of the model (think of an eigenfunction expansion of a smooth function s with rapidly decaying eigenvalues so that high-frequency eigenfunctions contribute very little to s, i.e., s is not very complex).



Regularization

The regularization functional frequently measures the smoothness of the predictor of s.

Remark

The regularization term can also be interpreted as a measure of the complexity of the model (think of an eigenfunction expansion of a smooth function s with rapidly decaying eigenvalues so that high-frequency eigenfunctions contribute very little to s, i.e., s is not very complex).

Example

The quadratic loss $L(\boldsymbol{y}, \boldsymbol{s}) = \|\boldsymbol{y} - \boldsymbol{s}\|^2$ coupled with a quadratic regularization functional leads to spline smoothing or penalized least squares.
This is also what we use for learning via RBF networks.

Regularization Theory in RKHSs

If $\mathcal{H}_K(\Omega)$ is a RPHS with reproducing kernel K we can consider $\boldsymbol{c}^T \mathrm{K} \boldsymbol{c}$, the square of the native space norm of s, as the associated regularization term.

Theorem (Representer Theorem [KW71])

The optimal predictor s in $\mathcal{H}_K(\Omega)$ characterized by

$$\min_{\boldsymbol{c}} \left[L(\boldsymbol{y}, \mathrm{K}\boldsymbol{c}) + \mu \boldsymbol{c}^T \mathrm{K} \boldsymbol{c} \right],$$

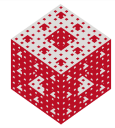
can be expressed as a linear combination of kernel functions, i.e.,

$$s(\boldsymbol{x}) = \sum_{j=1}^{N} c_j K(\boldsymbol{x}, \boldsymbol{x}_j).$$

Here K *is our usual kernel matrix and* $\boldsymbol{y} = (y_1, \ldots, y_N)^T$.

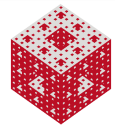
Remark

- *For general L the optimal predictor characterized in the representer theorem requires the solution of a non-trivial nonlinear optimization problem.*



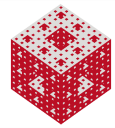
Remark

- *For general L the optimal predictor characterized in the representer theorem requires the solution of a non-trivial nonlinear optimization problem.*
- *If L is squared loss and we have a square system matrix* K *then the optimal solution is obtained by simply solving the linear system* $\mathrm{K}\boldsymbol{c} = \boldsymbol{y}$*, i.e., the empirical risk is zero and the native space norm of s is automatically minimized (see also [Fas07, Chapter 19], or the discussion below).*



Remark

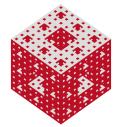
Question: *Why are we willing to solve an optimization problem rather than a simple interpolation/regression problem?*



Remark

Question: *Why are we willing to solve an optimization problem rather than a simple interpolation/regression problem?*

Answer: *Machine learning applications generally deal with data contaminated by significant errors (perhaps on the order of 10% error). This often means that the observed data look rough, as though generated by a nonsmooth function.*

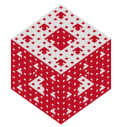


Remark

Question: *Why are we willing to solve an optimization problem rather than a simple interpolation/regression problem?*

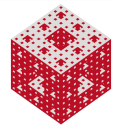
Answer: *Machine learning applications generally deal with data contaminated by significant errors (perhaps on the order of 10% error). This often means that the observed data look rough, as though generated by a nonsmooth function.*

A common assumption is that the data were generated by a smooth function, but that the observations were corrupted by a nonsmooth error term.



Remark

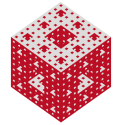
This suggests we should use smooth basis functions for the approximation, but should also do some balancing to avoid fitting the nonsmooth errors.



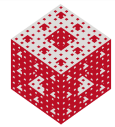
Remark

This suggests we should use smooth basis functions for the approximation, but should also do some balancing to avoid fitting the nonsmooth errors.

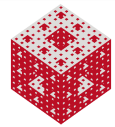
- *Choosing $\mu > 0$ forces $\mathbf{c}$ to not grow too large (and so prevents the wild oscillations which would be needed to exactly fit data from a nonsmooth function).*
  - *A larger μ will demand smaller c_i values and care less about fitting the data.*
  - *A smaller μ will more closely fit the observed data at the cost of an approximation which is more susceptible to errors in the observations, i.e., overfitting.*



Outline

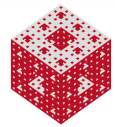


Machine learning networks follow a supervised learning strategy according to which the algorithm learns patterns that exist in a given set of inputs/outputs $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$.



Machine learning networks follow a supervised learning strategy according to which the algorithm learns patterns that exist in a given set of inputs/outputs $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$.

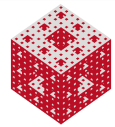
In its simplest form, the pattern is approximated by a linear combination of basis functions.



Machine learning networks follow a supervised learning strategy according to which the algorithm learns patterns that exist in a given set of inputs/outputs $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$.

In its simplest form, the pattern is approximated by a linear combination of basis functions.

Common choices of basis functions include the Gaussian, trigonometric functions and sigmoids, which have an "on/off" behavior and are used to reflect the behavior of neurons in a human brain [HTF09, Orr96].

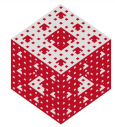


Machine learning networks follow a supervised learning strategy according to which the algorithm learns patterns that exist in a given set of inputs/outputs $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$.

In its simplest form, the pattern is approximated by a linear combination of basis functions.

Common choices of basis functions include the Gaussian, trigonometric functions and sigmoids, which have an "on/off" behavior and are used to reflect the behavior of neurons in a human brain [HTF09, Orr96].

We will consider only so-called single layer learning algorithms using shifts of the Gaussian kernel (often called the RBF-kernel in the learning literature).

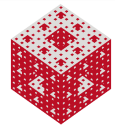


We construct s using a linear combination of basis functions analogously to our earlier approaches:

We construct s using a linear combination of basis functions analogously to our earlier approaches:

- If $M \leq N$ copies of K are used with centers at locations $\{\boldsymbol{z}_j\}_{j=1}^M$ then

$$s(\boldsymbol{x}) = \sum_{j=1}^{M} c_j K(\boldsymbol{x}, \boldsymbol{z}_j).$$

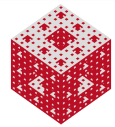


We construct s using a linear combination of basis functions analogously to our earlier approaches:

- If $M \leq N$ copies of K are used with centers at locations $\{\boldsymbol{z}_j\}_{j=1}^M$ then

$$s(\boldsymbol{x}) = \sum_{j=1}^{M} c_j K(\boldsymbol{x}, \boldsymbol{z}_j).$$

- If $M = N$ and $\boldsymbol{z}_i = \boldsymbol{x}_i$ for $1 \leq i \leq N$ then s is an interpolant.

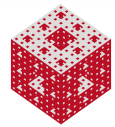


We construct s using a linear combination of basis functions analogously to our earlier approaches:

- If $M \leq N$ copies of K are used with centers at locations $\{\boldsymbol{z}_j\}_{j=1}^{M}$ then

$$s(\boldsymbol{x}) = \sum_{j=1}^{M} c_j K(\boldsymbol{x}, \boldsymbol{z}_j).$$

- If $M = N$ and $\boldsymbol{z}_i = \boldsymbol{x}_i$ for $1 \leq i \leq N$ then s is an interpolant.
- Otherwise s is an approximation/regression (or smoothing spline in statistics).



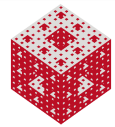
We construct s using a linear combination of basis functions analogously to our earlier approaches:

- If $M \leq N$ copies of K are used with centers at locations $\{\boldsymbol{z}_j\}_{j=1}^M$ then

$$s(\boldsymbol{x}) = \sum_{j=1}^{M} c_j K(\boldsymbol{x}, \boldsymbol{z}_j).$$

- If $M = N$ and $\boldsymbol{z}_i = \boldsymbol{x}_i$ for $1 \leq i \leq N$ then s is an interpolant.
- Otherwise s is an approximation/regression (or smoothing spline in statistics).

Either way, the coefficients $\boldsymbol{c}$ are defined as the vector which minimizes $\|\mathrm{K}\boldsymbol{c} - \boldsymbol{y}\|$ or a regularized version.



We construct s using a linear combination of basis functions analogously to our earlier approaches:

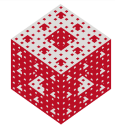
- If $M \leq N$ copies of K are used with centers at locations $\{\boldsymbol{z}_j\}_{j=1}^M$ then

$$s(\boldsymbol{x}) = \sum_{j=1}^{M} c_j K(\boldsymbol{x}, \boldsymbol{z}_j).$$

- If $M = N$ and $\boldsymbol{z}_i = \boldsymbol{x}_i$ for $1 \leq i \leq N$ then s is an interpolant.
- Otherwise s is an approximation/regression (or smoothing spline in statistics).

Either way, the coefficients $\boldsymbol{c}$ are defined as the vector which minimizes $\|\mathrm{K}\boldsymbol{c} - \boldsymbol{y}\|$ or a regularized version.

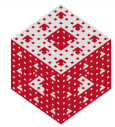
In the interpolation setting this is zero for $\boldsymbol{c} = \mathrm{K}^{-1}\boldsymbol{y}$.



We commonly find $\boldsymbol{c}$ by solving a minimization problem with squared loss of the form

$$\begin{aligned}\boldsymbol{c} &= \underset{\boldsymbol{c}\in\mathbb{R}^M}{\operatorname{argmin}} \sum_{i=1}^{N}\left(y_i - \sum_{j=1}^{M} c_j K(\boldsymbol{x}_i, \boldsymbol{z}_j)\right)^2 + \mu \sum_{j=1}^{M} c_j^2 \\ &= \underset{\boldsymbol{c}\in\mathbb{R}^M}{\operatorname{argmin}} \|\boldsymbol{y} - \mathrm{K}\boldsymbol{c}\|^2 + \mu\|\boldsymbol{c}\|^2,\end{aligned}$$

where we specify the kernel K and regularization parameter μ beforehand.



We commonly find $\boldsymbol{c}$ by solving a minimization problem with squared loss of the form

$$\begin{aligned}\boldsymbol{c} &= \underset{\boldsymbol{c}\in\mathbb{R}^M}{\operatorname{argmin}} \sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{M} c_j K(\boldsymbol{x}_i, \boldsymbol{z}_j) \right)^2 + \mu \sum_{j=1}^{M} c_j^2 \\ &= \underset{\boldsymbol{c}\in\mathbb{R}^M}{\operatorname{argmin}} \|\boldsymbol{y} - \mathrm{K}\boldsymbol{c}\|^2 + \mu\|\boldsymbol{c}\|^2,\end{aligned}$$

where we specify the kernel K and regularization parameter μ beforehand.

Remark

Since this is a convex minimization problem, the necessary condition obtained by the standard strategy of differentiating and setting the result equal to zero is also sufficient.

We see that

$$\nabla_{\boldsymbol{c}} \left[(\boldsymbol{y} - \mathrm{K}\boldsymbol{c})^T (\boldsymbol{y} - \mathrm{K}\boldsymbol{c}) + \mu \boldsymbol{c}^T \boldsymbol{c} \right] = 0$$

We see that

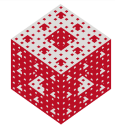
$$\nabla_{\boldsymbol{c}} \left[(\boldsymbol{y} - \mathrm{K}\boldsymbol{c})^T (\boldsymbol{y} - \mathrm{K}\boldsymbol{c}) + \mu \boldsymbol{c}^T \boldsymbol{c} \right] = 0$$

$$\iff \quad \nabla_{\boldsymbol{c}} \left[\boldsymbol{y}^T \boldsymbol{y} - \boldsymbol{c}^T \mathrm{K}^T \boldsymbol{y} - \boldsymbol{y}^T \mathrm{K}\boldsymbol{c} + \boldsymbol{c}^T \mathrm{K}^T \mathrm{K}\boldsymbol{c} + \mu \boldsymbol{c}^T \boldsymbol{c} \right] = 0$$

We see that

$$\begin{aligned} & \nabla_{\boldsymbol{c}} \left[(\boldsymbol{y} - \mathrm{K}\boldsymbol{c})^T (\boldsymbol{y} - \mathrm{K}\boldsymbol{c}) + \mu \boldsymbol{c}^T \boldsymbol{c} \right] = 0 \\ \iff \quad & \nabla_{\boldsymbol{c}} \left[\boldsymbol{y}^T \boldsymbol{y} - \boldsymbol{c}^T \mathrm{K}^T \boldsymbol{y} - \boldsymbol{y}^T \mathrm{K}\boldsymbol{c} + \boldsymbol{c}^T \mathrm{K}^T \mathrm{K}\boldsymbol{c} + \mu \boldsymbol{c}^T \boldsymbol{c} \right] = 0 \\ \iff \quad & 0 - \mathrm{K}^T \boldsymbol{y} - \mathrm{K}^T \boldsymbol{y} + 2\mathrm{K}^T \mathrm{K}\boldsymbol{c} + 2\mu \boldsymbol{c} = 0, \end{aligned}$$

where we've used $\boldsymbol{y}^T \mathrm{K}\boldsymbol{c} = \boldsymbol{c}^T \mathrm{K}^T \boldsymbol{y}$ since it is just a scalar.



We see that

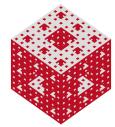
$$\begin{aligned} & \nabla_{\boldsymbol{c}} \left[(\boldsymbol{y} - \mathrm{K}\boldsymbol{c})^T (\boldsymbol{y} - \mathrm{K}\boldsymbol{c}) + \mu \boldsymbol{c}^T \boldsymbol{c} \right] = 0 \\ \iff \quad & \nabla_{\boldsymbol{c}} \left[\boldsymbol{y}^T \boldsymbol{y} - \boldsymbol{c}^T \mathrm{K}^T \boldsymbol{y} - \boldsymbol{y}^T \mathrm{K}\boldsymbol{c} + \boldsymbol{c}^T \mathrm{K}^T \mathrm{K}\boldsymbol{c} + \mu \boldsymbol{c}^T \boldsymbol{c} \right] = 0 \\ \iff \quad & 0 - \mathrm{K}^T \boldsymbol{y} - \mathrm{K}^T \boldsymbol{y} + 2\mathrm{K}^T \mathrm{K}\boldsymbol{c} + 2\mu \boldsymbol{c} = 0, \end{aligned}$$

where we've used $\boldsymbol{y}^T \mathrm{K}\boldsymbol{c} = \boldsymbol{c}^T \mathrm{K}^T \boldsymbol{y}$ since it is just a scalar.

Thus we can solve the optimization problem by solving the linear system

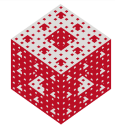
$$(\mathrm{K}^T \mathrm{K} + \mu \mathrm{I}_M)\boldsymbol{c} = \mathrm{K}^T \boldsymbol{y}, \tag{1}$$

which is guaranteed to be well-defined (i.e., the inverse exists) for any $\mu > 0$.



Picking an "optimal" μ via GCV

Treating μ as a free parameter gives the potential to improve the quality of the prediciton but also the potential to cripple it, making an appropriate choice vital as it was in Chapter 13.

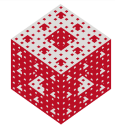


Picking an "optimal" μ via GCV

Treating μ as a free parameter gives the potential to improve the quality of the prediciton but also the potential to cripple it, making an appropriate choice vital as it was in Chapter 13.

Generalized cross validation (GCV) is popular in the machine learning literature [CW79, GHW79, GVM97]. For our RBF networks it can be computed as [AC10, Orr96]

$$C_{GCV} = \boldsymbol{y}^T P^2 \boldsymbol{y} \frac{N}{(\text{trace}\, P)^2}, \qquad P = I_N - K(K^T K + \mu I_M)^{-1} K^T.$$



Picking an "optimal" μ via GCV

Treating μ as a free parameter gives the potential to improve the quality of the prediciton but also the potential to cripple it, making an appropriate choice vital as it was in Chapter 13.

Generalized cross validation (GCV) is popular in the machine learning literature [CW79, GHW79, GVM97]. For our RBF networks it can be computed as [AC10, Orr96]

$$C_{GCV} = \boldsymbol{y}^T \mathsf{P}^2 \boldsymbol{y} \frac{N}{(\operatorname{trace} \mathsf{P})^2}, \qquad \mathsf{P} = \mathsf{I}_N - \mathsf{K}(\mathsf{K}^T\mathsf{K} + \mu \mathsf{I}_M)^{-1}\mathsf{K}^T.$$

Remark

C_{GCV} *is not an error in the true sense, though it is related to the residual:*

$$\mathsf{P}\boldsymbol{y} = \boldsymbol{y} - \mathsf{K}\boldsymbol{c}, \quad \textit{i.e.,} \quad \boldsymbol{y}^T\mathsf{P}^2\boldsymbol{y} = \|\boldsymbol{y} - \mathsf{K}\boldsymbol{c}\|_2^2.$$

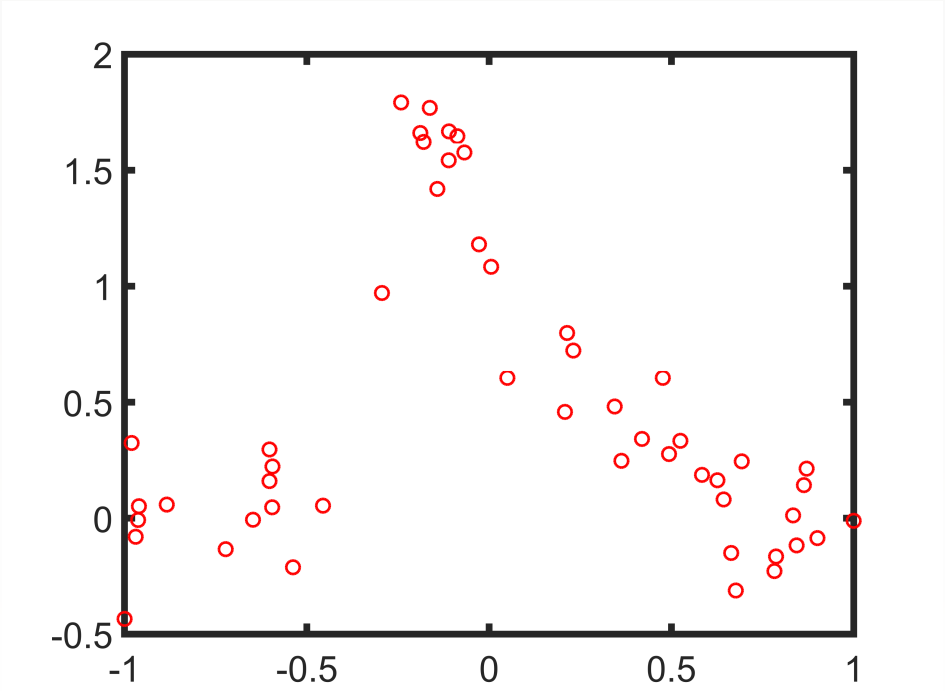
Example: Effects of μ

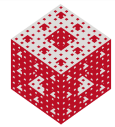
We take $N = 50$ data sampled randomly on $[-1, 1]$ from the function

$$f(x) = (1 - 4x + 32x^2)e^{-16x^2}$$

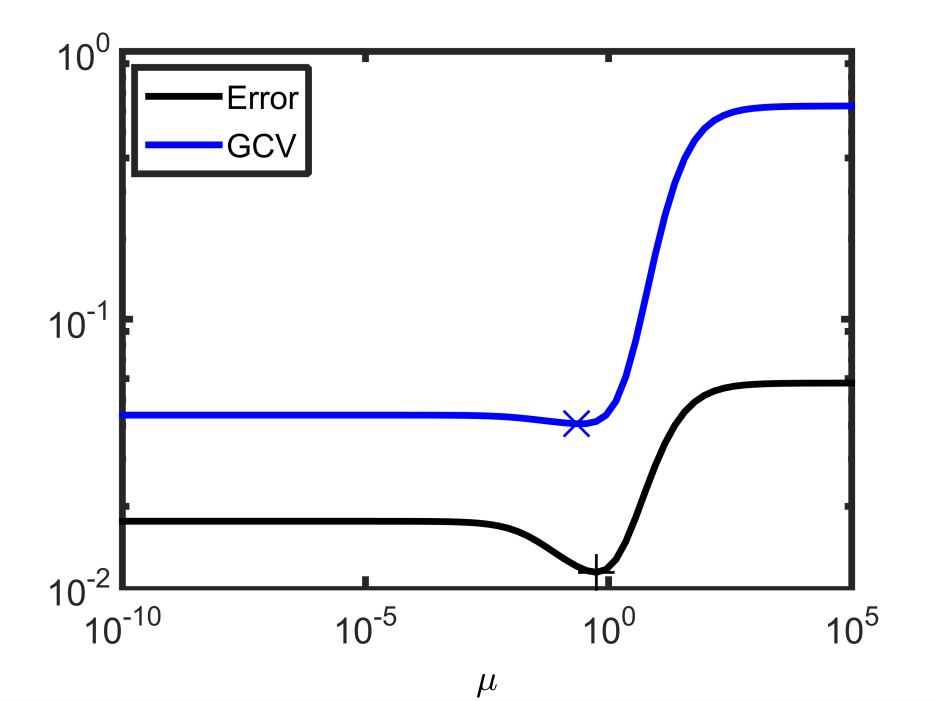
with added normally distributed noise with zero mean and standard deviation 0.2.

MATLAB code for this example is provided in `RBFNetwork1.m`.

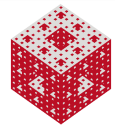


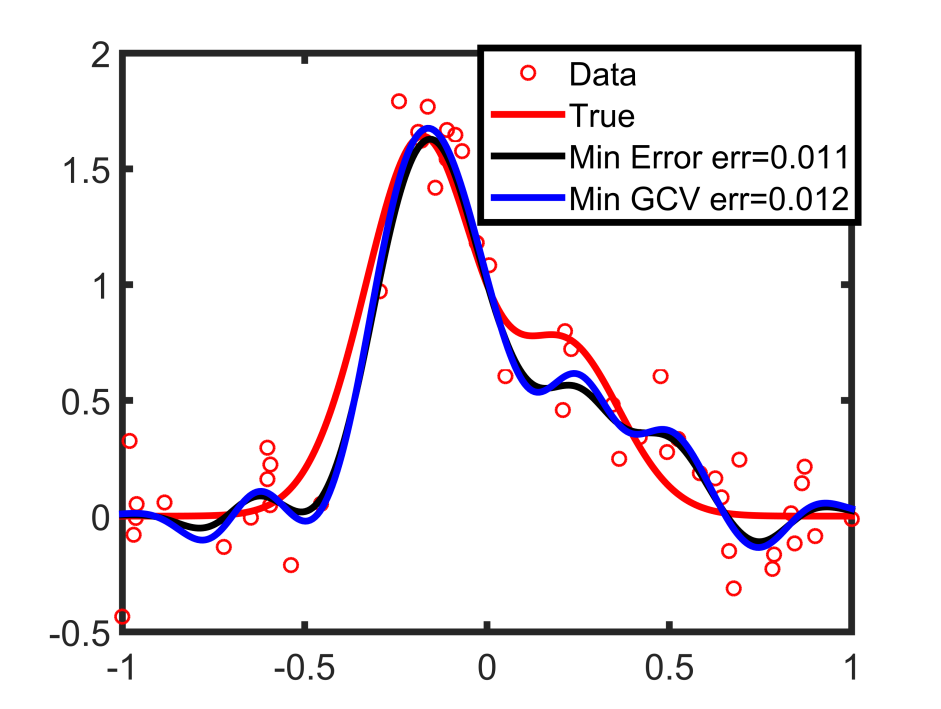


We form an RBF network using Gaussian kernels with $\varepsilon = 8$ centered at $M = 15$ evenly spaced points in $[-1, 1]$.
To smooth out the noise, we consider regularization parameters μ in $[10^{-10}, 10^{5}]$.



The RMS relative error is computed at 300 evenly spaced points in $[-1, 1]$.





- "Optimal" μ from the error plot: $\mu \approx 0.56$
- "Optimal" μ from GCV: $\mu \approx 0.22$

Remark

The network based on a larger μ oscillates less than the one produced with smaller μ.

Remark

- *The smoothing parameter μ can affect the quality of the prediction in an understandable (larger μ yields less oscillation) but unpredictable (the optimal amount of oscillation is unknown a priori) way, much as the shape parameter ε works in unpredictable ways.*

Remark

- *The smoothing parameter μ can affect the quality of the prediction in an understandable (larger μ yields less oscillation) but unpredictable (the optimal amount of oscillation is unknown a priori) way, much as the shape parameter ε works in unpredictable ways.*
- *Due to the ill-conditioning of $\mathrm{K}^T\mathrm{K}$ as $\varepsilon \to 0$, for small ε values, μ may actually improve the condition of the solution.*

Remark

- *The smoothing parameter μ can affect the quality of the prediction in an understandable (larger μ yields less oscillation) but unpredictable (the optimal amount of oscillation is unknown a priori) way, much as the shape parameter ε works in unpredictable ways.*
- *Due to the ill-conditioning of $\mathrm{K}^T\mathrm{K}$ as $\varepsilon \to 0$, for small ε values, μ may actually improve the condition of the solution.*
- *There is a complicated relationship between μ and ε. For some kernels μ may improve the accuracy through smoothing out noise and for others it may improve the quality by reducing ill-conditioning.*

Remark

- *The smoothing parameter μ can affect the quality of the prediction in an understandable (larger μ yields less oscillation) but unpredictable (the optimal amount of oscillation is unknown a priori) way, much as the shape parameter ε works in unpredictable ways.*
- *Due to the ill-conditioning of $\mathrm{K}^T\mathrm{K}$ as $\varepsilon \to 0$, for small ε values, μ may actually improve the condition of the solution.*
- *There is a complicated relationship between μ and ε. For some kernels μ may improve the accuracy through smoothing out noise and for others it may improve the quality by reducing ill-conditioning.*
- *Although μ affects the quality of the prediction s, it does not change the native space $\mathcal{H}_K$ (it pushes the interpolant toward a set of functions which try to minimize $\|\boldsymbol{c}\|_2$ rather than $\|s\|_{\mathcal{H}_K}$). In contrast, changing ε changes K and, therefore, $\mathcal{H}_K$.*

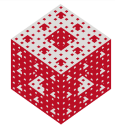
Example: Effects of stable basis

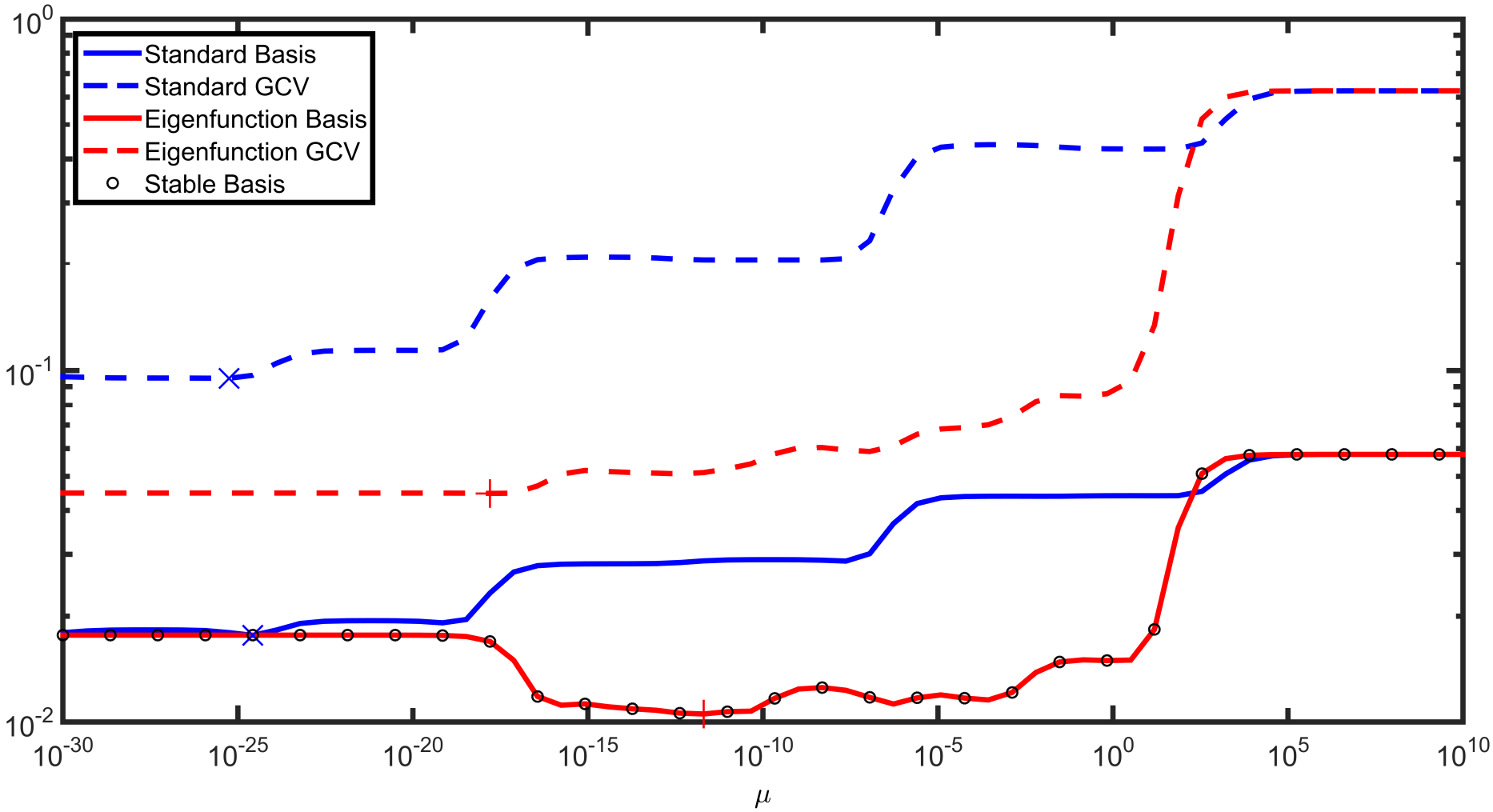
We use the same data as before, but now "flat" Gaussian kernels with $\varepsilon = 0.01$ so that $K^T K$ is severely ill-conditioned.

We compare three alternative approaches:

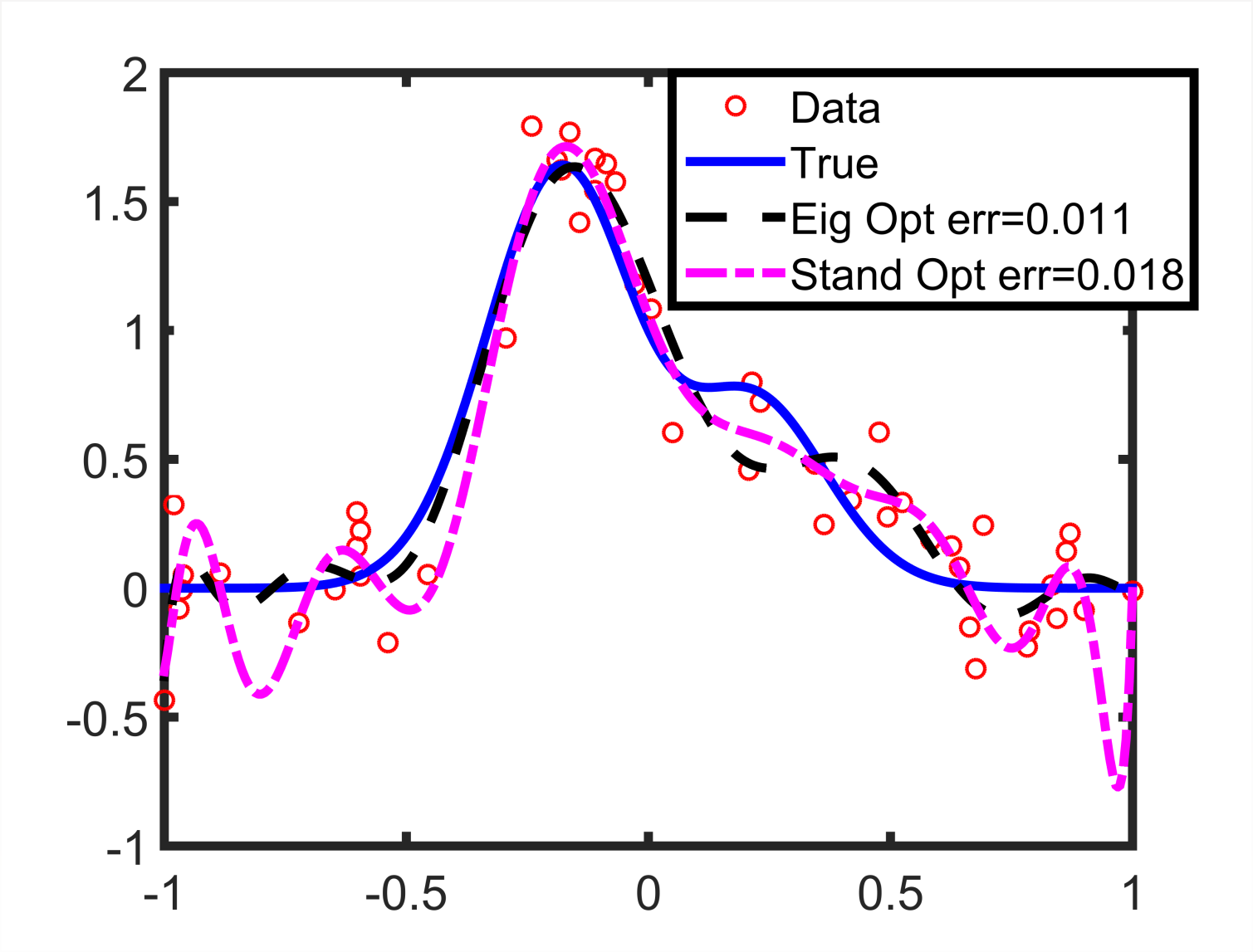
- use of standard kernel basis $\{K(\cdot, z_j)\}$, z_j evenly spaced in $[-1, 1]$,
- use of stable basis $\{\psi_j(\cdot)\}$, z_j evenly spaced in $[-1, 1]$,
- use of eigenfunctions $\{\varphi_n(\cdot)\}$.

MATLAB code for this example is provided in `RBFNetwork2.m`.

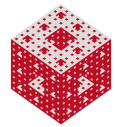




Error compared across various regularization values μ.

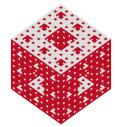


Predictions for "optimal" μ values.



Remark

- *The stable basis outperforms the standard basis for most μ values.*
- *Eigenfunctions and the stable basis essentially overlap.*
- *For large μ values, the prediction is dominated by the regularization component (cf. the overlap near $\mu \approx 10^4$).*
- *The similarity between the stable basis and eigenfunctions is due to the fact that the correction term in the HS-SVD decreases in magnitude as ε decreases.*
- *Even a tiny value of μ has a remarkable stabilization effect in this example.*



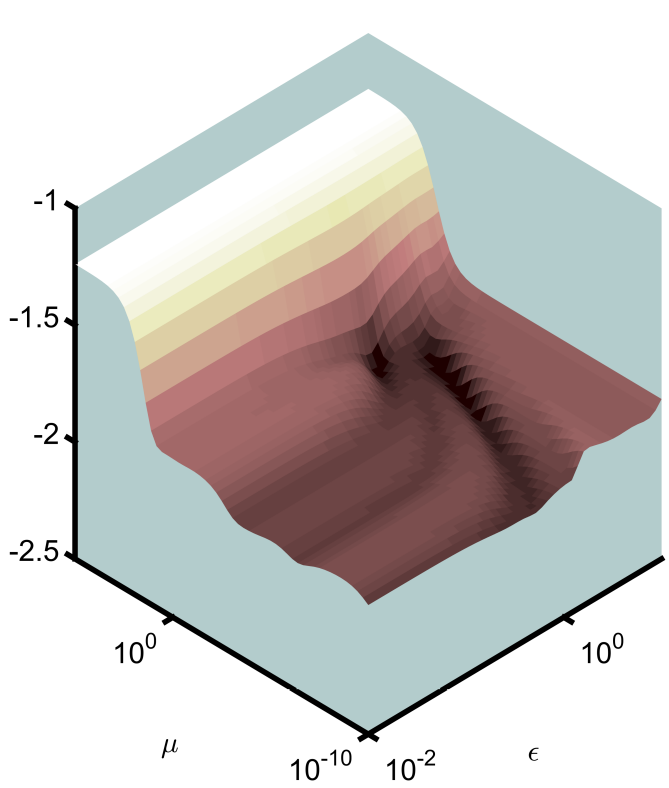
Example: Combined effects of μ and ε

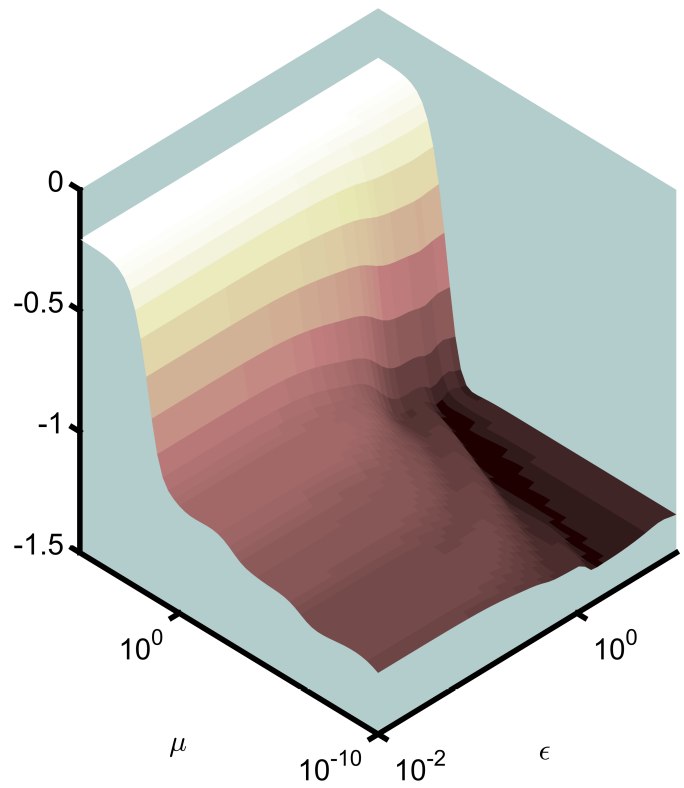
Again, same data as before, but now we look at the effect of allowing both ε and μ to change.

Only the eigenfunction basis is considered.

The errors for $\mu \in [10^{-10}, 10^{5}]$ and $\varepsilon \in [10^{-2}, 10^{1}]$, along with GCV plots are shown below.

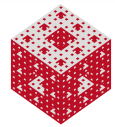
MATLAB code for this example is provided in `RBFNetwork3.m`.





An optimal error exists at $\mu \approx 0.011$ and $\varepsilon \approx 2.8$, but is not predicted by GCV ("optimal" $\mu \approx 0.0017$, $\varepsilon \approx 3.7$).

Outline



We now discuss the two main applications of support vector machines (SVMs) in the context of supervised machine learning:

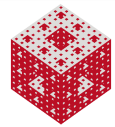
- classification and
- regression.

Both of these applications can be formulated within the regularization framework outlined at the beginning of this chapter.

Standard (binary) classification

Given: a set of training data $\{(\boldsymbol{x}_i, y_i) : \ i = 1, \ldots, N\}$ with

- measurements $\boldsymbol{x}_i \in \mathbb{R}^d$ and
- data values in the form of labels $y_i \in \{-1, +1\}$.

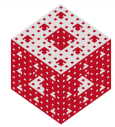


Standard (binary) classification

Given: a set of training data $\{(\boldsymbol{x}_i, y_i) : \ i = 1, \ldots, N\}$ with

- measurements $\boldsymbol{x}_i \in \mathbb{R}^d$ and
- data values in the form of labels $y_i \in \{-1, +1\}$.

Find: a predictor s that will allow us to assign an appropriate label, either -1 or $+1$, to a future measurement $\boldsymbol{x}$.



Example

A predictor might be

$$s(\boldsymbol{x}) = \text{sign}\left(h(\boldsymbol{x})\right),$$

where *h* denotes a hyperplane separating the given measurements.

Example

A predictor might be

$$s(\boldsymbol{x}) = \operatorname{sign}\left(h(\boldsymbol{x})\right),$$

where h denotes a hyperplane separating the given measurements.

A typical loss function is given by the hinge loss (or soft margin loss)

$$L\left(y, h(\boldsymbol{x})\right) = \max\left(1 - yh(\boldsymbol{x}), 0\right)$$

since

$$L\left(y, h(\boldsymbol{x})\right) = 0 \quad \Longleftrightarrow \quad yh(\boldsymbol{x}) \geq 1,$$

i.e., y and $h(\boldsymbol{x})$ have the same sign and $|h(\boldsymbol{x})| \geq 1$ so that we have enough confidence in our prediction (see, e.g., [SS02, Chapter 3]).

Example

A predictor might be

$$s(\boldsymbol{x}) = \operatorname{sign}\left(h(\boldsymbol{x})\right),$$

where h denotes a hyperplane separating the given measurements.

A typical loss function is given by the hinge loss (or soft margin loss)

$$L\left(y, h(\boldsymbol{x})\right) = \max\left(1 - yh(\boldsymbol{x}), 0\right)$$

since

$$L\left(y, h(\boldsymbol{x})\right) = 0 \quad \Longleftrightarrow \quad yh(\boldsymbol{x}) \geq 1,$$

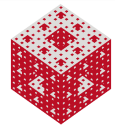
i.e., y and $h(\boldsymbol{x})$ have the same sign and $|h(\boldsymbol{x})| \geq 1$ so that we have enough confidence in our prediction (see, e.g., [SS02, Chapter 3]).

An appropriate regularization term will be given by some norm of h (see below for more details).

Regression

We estimate continuous numeric values as discussed in the previous section.
As for RBF networks, we can use the squared loss

$$L(y, s(\boldsymbol{x})) = (y - s(\boldsymbol{x}))^2 .$$



Regression

We estimate continuous numeric values as discussed in the previous section.
As for RBF networks, we can use the squared loss

$$L(y, s(\boldsymbol{x})) = (y - s(\boldsymbol{x}))^2 .$$

Alternatively, the so-called ϵ-insensitive loss

$$L(y, s(\boldsymbol{x})) = \max(|y - s(\boldsymbol{x})| - \epsilon, 0)$$

is used as a symmetric analogue of the hinge loss.

Regression

We estimate continuous numeric values as discussed in the previous section.
As for RBF networks, we can use the squared loss

$$L(y, s(\boldsymbol{x})) = (y - s(\boldsymbol{x}))^2 .$$

Alternatively, the so-called ϵ-insensitive loss

$$L(y, s(\boldsymbol{x})) = \max(|y - s(\boldsymbol{x})| - \epsilon, 0)$$

is used as a symmetric analogue of the hinge loss.

Remark

According to the ϵ-insensitive loss function, deviations of the predicted value $s(\boldsymbol{x})$ from the correct value y are only penalized if they exceed ϵ, and therefore it will be possible to obtain sparse representations using only a subset of the data referred to as support vectors (more details below).

Linear Classification

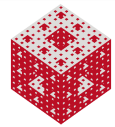
The simplest predictor is given by

$$s(\boldsymbol{x}) = \operatorname{sign}\left(h(\boldsymbol{x})\right),$$

where h denotes a hyperplane — directly in input space — of the form

$$h(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{w} + b = 0, \qquad \boldsymbol{x} \in \mathbb{R}^d,$$

that separates the measurements with label -1 from those with a $+1$.



Linear Classification

The simplest predictor is given by

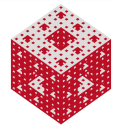
$$s(\boldsymbol{x}) = \operatorname{sign}\left(h(\boldsymbol{x})\right),$$

where h denotes a hyperplane — directly in input space — of the form

$$h(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{w} + b = 0, \qquad \boldsymbol{x} \in \mathbb{R}^d,$$

that separates the measurements with label -1 from those with a $+1$.

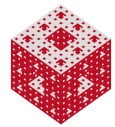
The weights $\boldsymbol{w}$ (which serve as the unit normal vector to the hyperplane) and the bias b can be determined by maximizing the margin or gap to both sides of this hyperplane (see, e.g., [HTF09, Chapter 12]).



Unconstrained minimization

Since the size of the margin is $\frac{1}{\|\boldsymbol{w}\|}$, and we want to maximize this margin, a natural regularization functional is:

$$\text{minimize } \|\boldsymbol{w}\| \qquad \text{(norm of the coefficients of } h\text{)}.$$



Unconstrained minimization

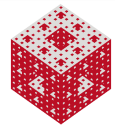
Since the size of the margin is $\frac{1}{\|\boldsymbol{w}\|}$, and we want to maximize this margin, a natural regularization functional is:

$$\text{minimize } \|\boldsymbol{w}\| \qquad \text{(norm of the coefficients of } h\text{)}.$$

Using the hinge loss function and $h(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{w} + b$, we get the unconstrained minimization problem

$$\min_{\boldsymbol{w},b} \left[\frac{1}{N} \sum_{i=1}^{N} \max\left(1 - y_i h(\boldsymbol{x}_i), 0\right) + \mu \frac{1}{2} \boldsymbol{w}^T\boldsymbol{w} \right],$$

where μ is an appropriately chosen regularization parameter.

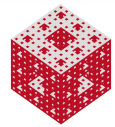


Constrained optimization

The following constrained optimization with slack variables ξ_i is more common since it also allows us to deal with the case where the given measurements are not perfectly separable by h:

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \left[\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{N}\xi_i \right]$$
$$\text{subject to } y_i h(\boldsymbol{x}_i) \geq 1 - \xi_i, \quad i = 1, \ldots, N,$$
$$\xi_i \geq 0,$$

where the regularization parameter μ is transformed into $C = \frac{1}{N\mu}$.



Constrained optimization

The following constrained optimization with slack variables ξ_i is more common since it also allows us to deal with the case where the given measurements are not perfectly separable by h:

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \left[\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{N}\xi_i \right]$$
$$\text{subject to } y_i h(\boldsymbol{x}_i) \geq 1 - \xi_i, \quad i = 1, \ldots, N,$$
$$\xi_i \geq 0,$$

where the regularization parameter μ is transformed into $C = \frac{1}{N\mu}$.

Remark

This formulation is known in the SVM literature as the primal problem (and — ironically — as the dual problem in the optimization literature).

SVM dual problem

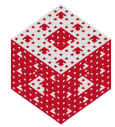
The SVM dual problem can be derived via Lagrange multipliers α_i (see, e.g., [HTF09, Chapter 12]) and is of the form

$$\max_{\boldsymbol{\alpha}} \left(\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j \right)$$

$$\text{subject to } \sum_{i=1}^{N} \alpha_i y_i = 0,$$

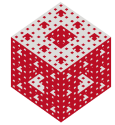
$$0 \leq \alpha_i \leq C,$$

where C is known as a box constraint and $\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i$ (which follows from setting the $\boldsymbol{w}$-gradient of the primal Lagrange multiplier functional equal to zero).
The bias b is given by $b = y_i - \boldsymbol{x}_i^T \boldsymbol{w}$ for any i such that the optimal $\alpha_i > 0$.



Remark

- *For stability purposes we compute the bias by considering all qualifying indices and find b using the mean.*
- *The box constraint C is a free parameter which needs to be either set by the user or determined by an additional parameter optimization methods such as cross validation.*

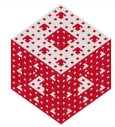


Kernel classification

Feature maps (see Chapter 2) allow us to view kernel values $K(\boldsymbol{x}, \boldsymbol{z})$ as the dot product of the transformed data in feature space, i.e., given $\boldsymbol{x}$ and $\boldsymbol{z}$ in input space and a feature map Φ we have

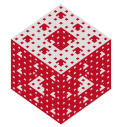
$$K(\boldsymbol{x}, \boldsymbol{z}) = \Phi(\boldsymbol{x})^T \Phi(\boldsymbol{z}).$$

Since the objective function the SVM dual problem is expressed in terms of dot products in input space we can now use the concept of feature maps and related kernels to talk about separating hyperplanes in feature space.



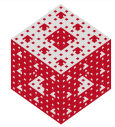
Remark

- *The feature space is potentially infinite-dimensional (as, e.g., in the case of the Gaussian kernel) and therefore offers much more flexibility for separating the data than the finite-dimensional input space.*



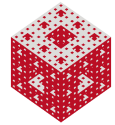
Remark

- *The feature space is potentially infinite-dimensional (as, e.g., in the case of the Gaussian kernel) and therefore offers much more flexibility for separating the data than the finite-dimensional input space.*
- *Cover's theorem [Cov65] provides a theoretical foundation for this. It ensures that data which can not be separated by a hyperplane in input space most likely will be linearly separable after being transformed to feature space by a suitable feature map.*



Remark

- *The feature space is potentially infinite-dimensional (as, e.g., in the case of the Gaussian kernel) and therefore offers much more flexibility for separating the data than the finite-dimensional input space.*
- *Cover's theorem [Cov65] provides a theoretical foundation for this. It ensures that data which can not be separated by a hyperplane in input space most likely will be linearly separable after being transformed to feature space by a suitable feature map.*
- *Thus, support vector machines — and kernel machines, in particular — are a good tool to use in order to tackle difficult data classification problems.*



Algorithms for kernel classification are essentially the same as before; simply replace the measurements $\boldsymbol{x}_i$ in input space by their transformation $\Phi(\boldsymbol{x}_i)$ into feature space.

Algorithms for kernel classification are essentially the same as before; simply replace the measurements $\boldsymbol{x}_i$ in input space by their transformation $\Phi(\boldsymbol{x}_i)$ into feature space.
The separating hyperplane now is

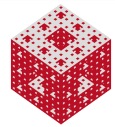
$$h(\boldsymbol{x}) = \Phi(\boldsymbol{x})^T \boldsymbol{w} + b = 0, \qquad \boldsymbol{x} \in \mathbb{R}^d,$$

and the SVM dual problem using the transformed input data is given by

$$\max_{\boldsymbol{\alpha}} \left(\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \Phi(\boldsymbol{x}_i)^T \Phi(\boldsymbol{x}_j) \right)$$

$$\text{subject to } \sum_{i=1}^{N} \alpha_i y_i = 0,$$

$$0 \leq \alpha_i \leq C.$$



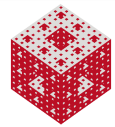
Since we have the kernel decomposition $K(\boldsymbol{x}, \boldsymbol{z}) = \Phi(\boldsymbol{x})^T \Phi(\boldsymbol{z})$ we don't have to compute (possibly infinite) dot products in feature space, but instead just fill the kernel matrix and solve

$$\max_{\boldsymbol{\alpha}} \left(\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \right) \tag{2}$$

$$\text{subject to } \sum_{i=1}^{N} \alpha_i y_i = 0,$$

$$0 \leq \alpha_i \leq C,$$

where, as before, C is the box constraint (which can be viewed as a tuning parameter) and $\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \Phi(\boldsymbol{x}_i)$.

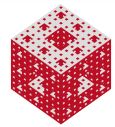


The classifier is now given by

$$s(\boldsymbol{x}) \quad = \quad \text{sign}\left(h(\boldsymbol{x})\right)$$

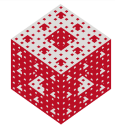
The classifier is now given by

$$\begin{aligned} s(\boldsymbol{x}) &= \text{sign}\left(h(\boldsymbol{x})\right) \\ &= \text{sign}\left(\Phi(\boldsymbol{x})^T \boldsymbol{w} + b\right) \end{aligned}$$



The classifier is now given by

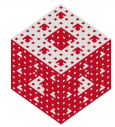
$$\begin{aligned} s(\boldsymbol{x}) &= \operatorname{sign}\left(h(\boldsymbol{x})\right) \\ &= \operatorname{sign}\left(\Phi(\boldsymbol{x})^T \boldsymbol{w} + b\right) \\ &= \operatorname{sign}\left(\Phi(\boldsymbol{x})^T \sum_{j=1}^{N} \alpha_j y_j \Phi(\boldsymbol{x}_j) + b\right) \end{aligned}$$



The classifier is now given by

$$\begin{aligned} s(\boldsymbol{x}) &= \operatorname{sign}\left(h(\boldsymbol{x})\right) \\ &= \operatorname{sign}\left(\Phi(\boldsymbol{x})^T \boldsymbol{w} + b\right) \\ &= \operatorname{sign}\left(\Phi(\boldsymbol{x})^T \sum_{j=1}^{N} \alpha_j y_j \Phi(\boldsymbol{x}_j) + b\right) \\ &= \operatorname{sign}\left(\sum_{j=1}^{N} \alpha_j y_j K(\boldsymbol{x}, \boldsymbol{x}_j) + b\right), \end{aligned}$$

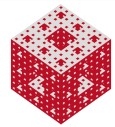
where b is obtained as before, i.e., $b = y_i - \sum_{j=1}^{N} \alpha_j y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ with i denoting the index of an α_i which is strictly between 0 and C.
For stability purposes we can again average over all such candidates.



What does the separating hyperplane in this case look like?

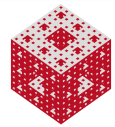
What does the separating hyperplane in this case look like?

- The hyperplane will be linear only in feature space (which we usually have no concrete knowledge of). In the input space the data will be separated by a nonlinear manifold.



What does the separating hyperplane in this case look like?

- The hyperplane will be linear only in feature space (which we usually have no concrete knowledge of). In the input space the data will be separated by a nonlinear manifold.
- The representation of this manifold is sparse in the sense that not all basis functions are needed to specify it.
  - Only those centers $\boldsymbol{x}_j$ whose corresponding α_j are nonzero define meaningful basis functions.
  - These special centers are referred to as support vectors.

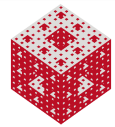


Remark

- *Since the decision boundary can be expressed in terms of a limited number of support vectors, i.e., it has a sparse representation, learning is possible in very high-dimensional input spaces [SC08].*
- *SVMs are*
  - *robust against several types of model violations and outliers,*
  - *computationally efficient, e.g., by using sequential minimal optimization (SMO) [Pla99] to perform the quadratic optimization task required for classification as well as regression.*
- *Another way to make SVMs perform more efficiently is to consider a low-rank representation for the kernel [FS02]. Below we test our own version based on eigenfunctions.*

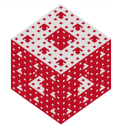
Remark

- *For positive definite kernels one can formulate the separating hyperplane without the bias term b. In that case the equality constraint* $\sum_{i=1}^{N} \alpha_i y_i = 0$ *(which may be somewhat of a nuisance during the optimization process) can be omitted [PMR+01].*



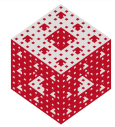
Remark

- *For positive definite kernels one can formulate the separating hyperplane without the bias term b. In that case the equality constraint* $\sum_{i=1}^{N} \alpha_i y_i = 0$ *(which may be somewhat of a nuisance during the optimization process) can be omitted [PMR+01].*
- *The primal and dual formulations each have their advantages.*
  - *The primal formulation (in input space) is good for large amounts of rather low-dimensional data.*
  - *The dual formulation (with kernels in feature space) is good for high-dimensional data (since only the number of support vectors matter).*

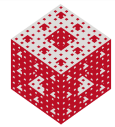


Remark

- *In our numerical experiments we use Gaussian kernels.*
- *Linear SVM uses the dot product kernel* $K(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x}^T \boldsymbol{z}$.
- *Other popular kernels are*
  - *polynomial kernels of degree* β *in the form* $K(\boldsymbol{x}, \boldsymbol{z}) = (1 + \boldsymbol{x}^T \boldsymbol{z})^\beta$,
  - *the sigmoid kernel (or multilayer perceptron)* $K(\boldsymbol{x}, \boldsymbol{z}) = \tanh(1 + \varepsilon \boldsymbol{x}^T \boldsymbol{z})$.
- *Kernels may be defined via the feature map (instead of in closed form), and this feature map can be picked depending on the specific application (e.g., as a string kernel for text mining).*



Outline



Example: Simple kernel classification

This example is from [HTF09, Section 2.3] (see also help for SVMs in MATLAB's Statistics Toolbox).

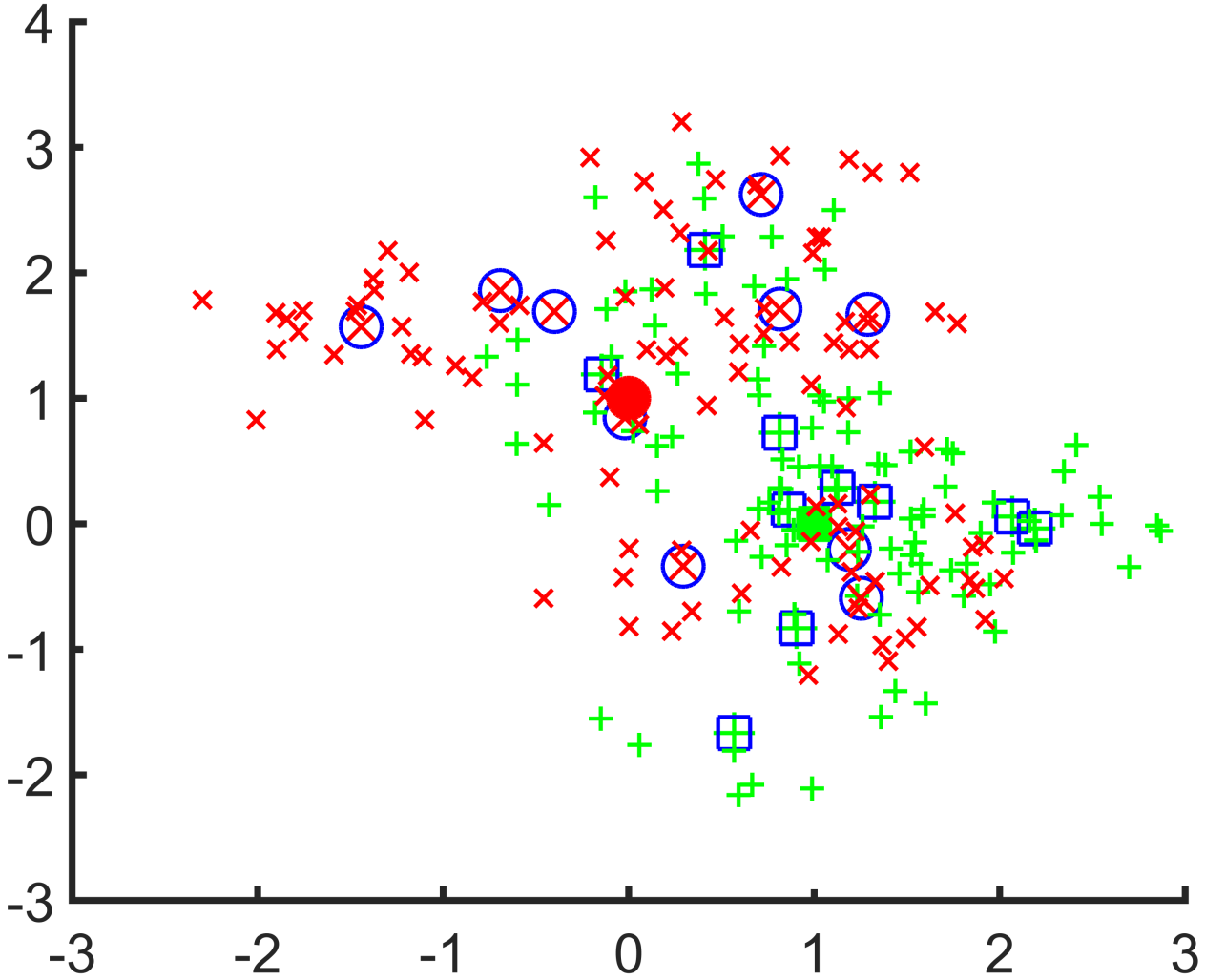
We attempt to learn/classify data coming from two different populations:

- population 1, normally distributed with center at $(1, 0)$ (filled red circle) and identity covariance
- population 2, normally distributed with center at $(0, 1)$ (filled green square) and identity covariance

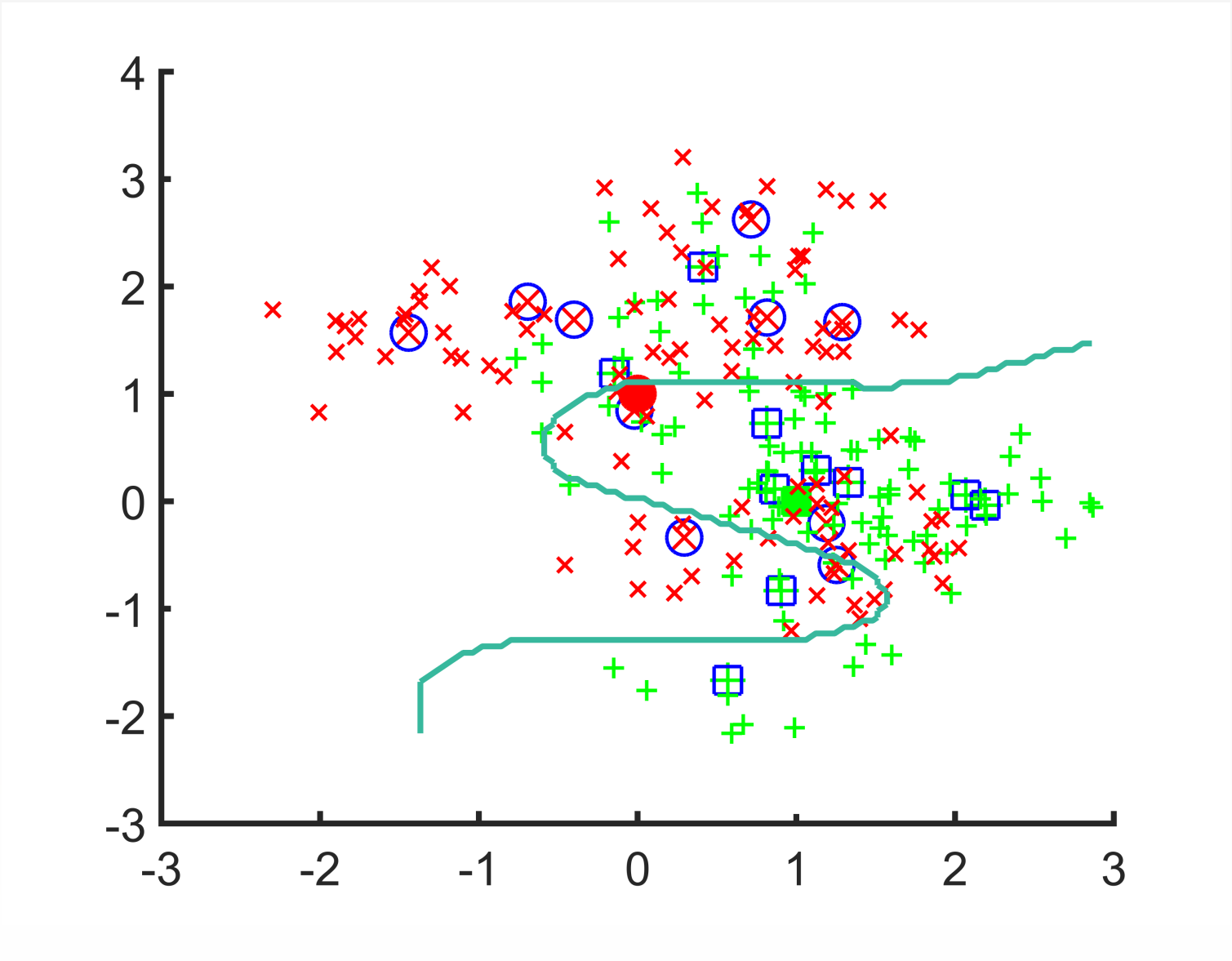
Use Gaussian kernels with varying shape parameter ε and box constraint C.

MATLAB code for this example is provided in `SVM1.m` which uses `SVM_Setup.m` and `gqr_fitsvm.m`.

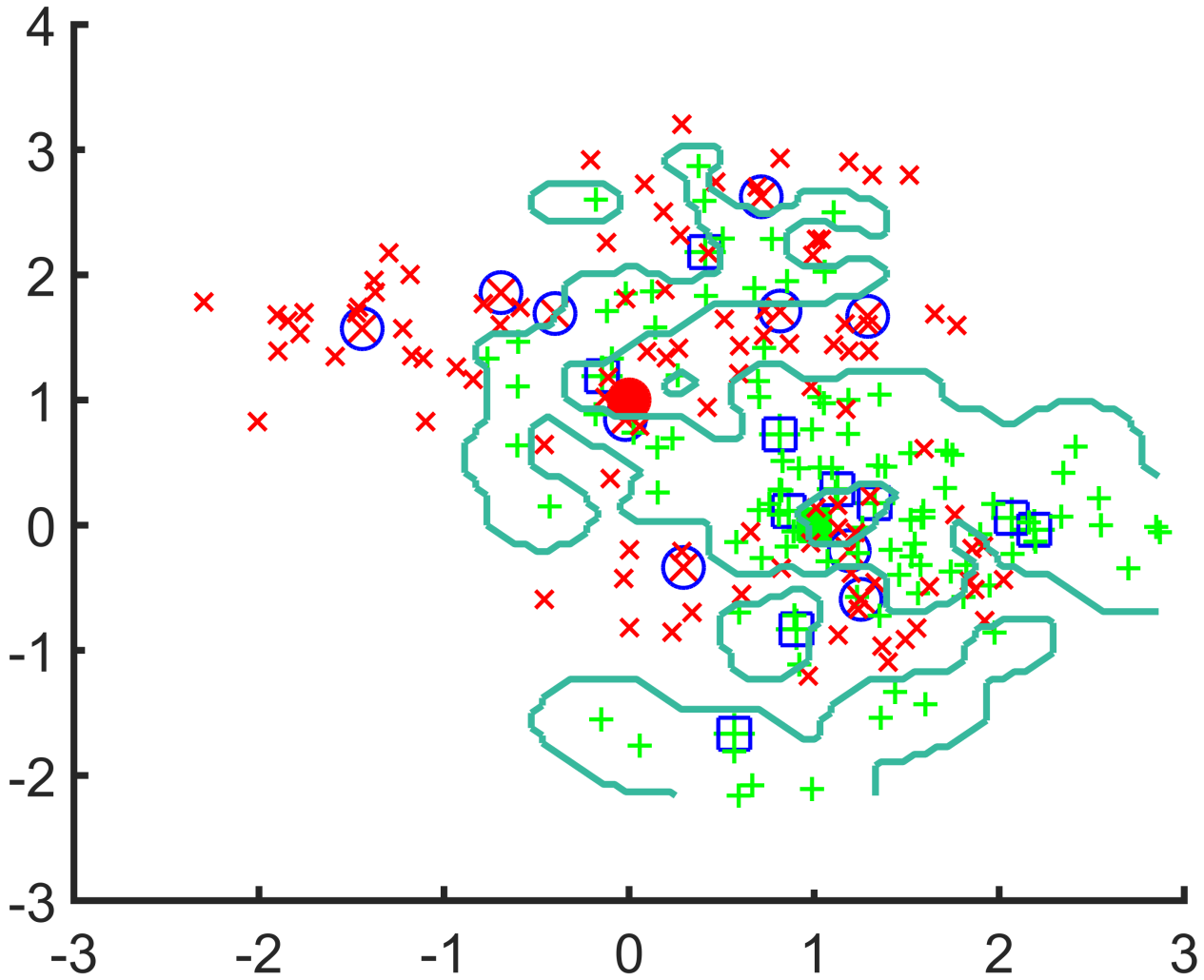
100 training (×, +) and 10 test points (◯, □)



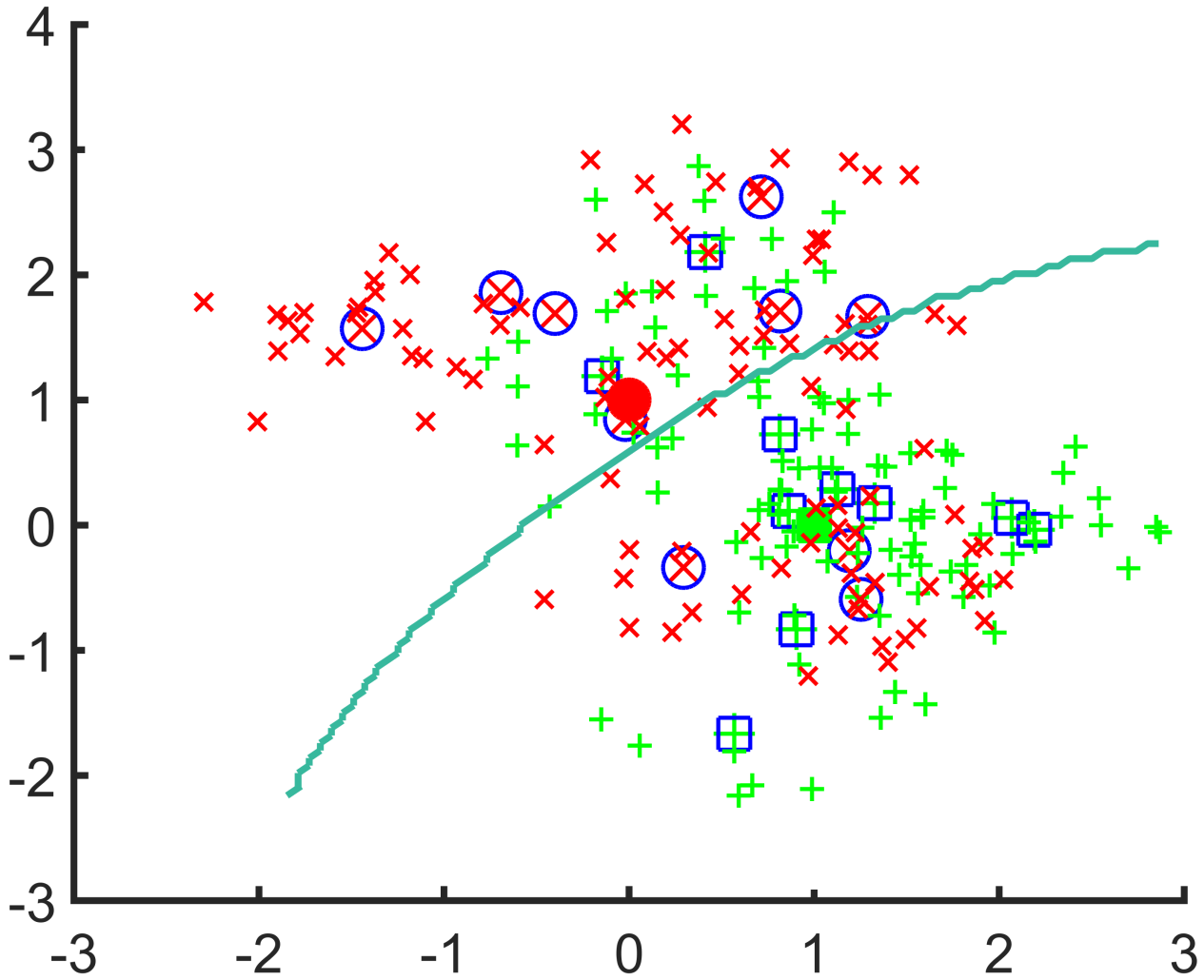
Separating hyperplane, $C = 1$, $\varepsilon = 1$



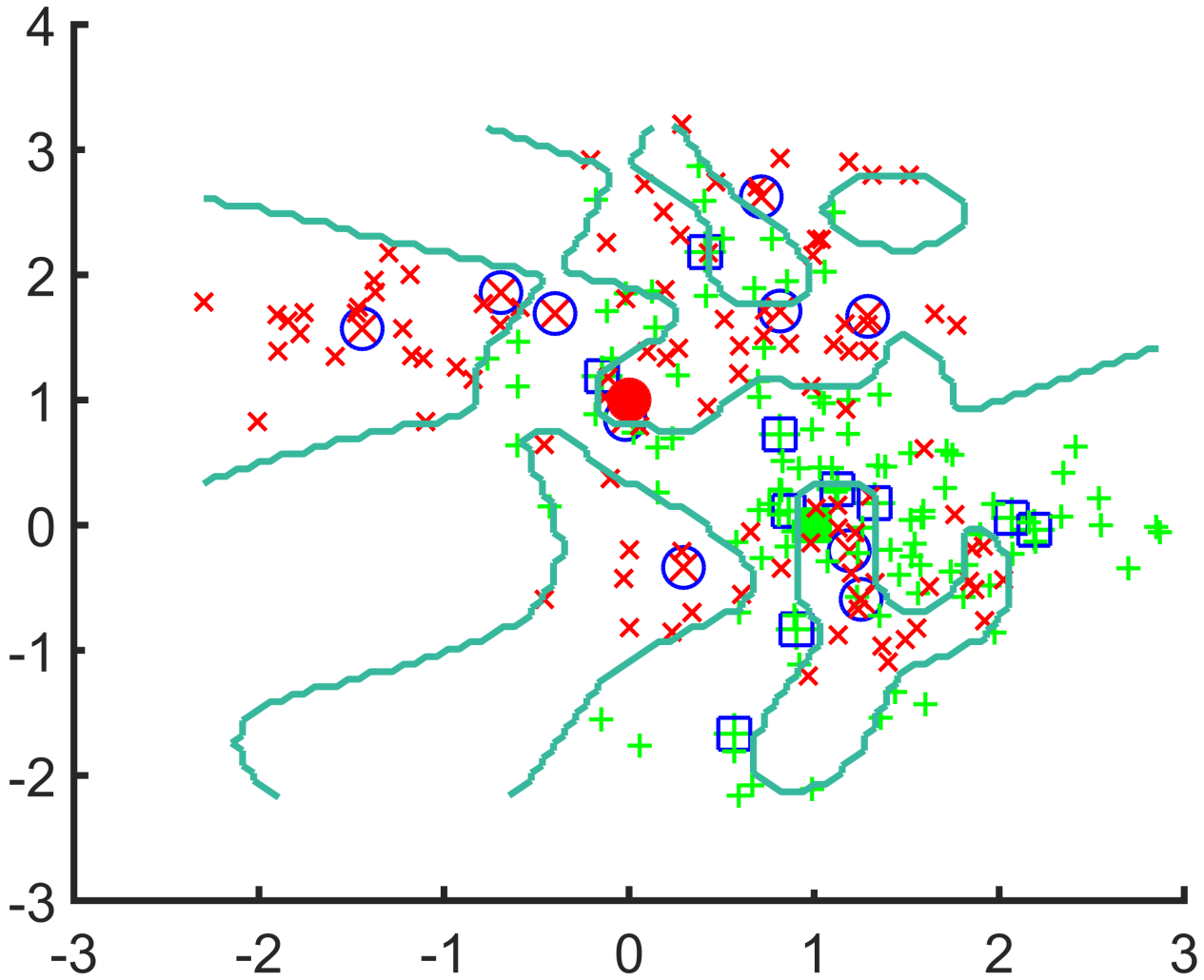
Separating hyperplane, $C = 1$, $\varepsilon = 5$



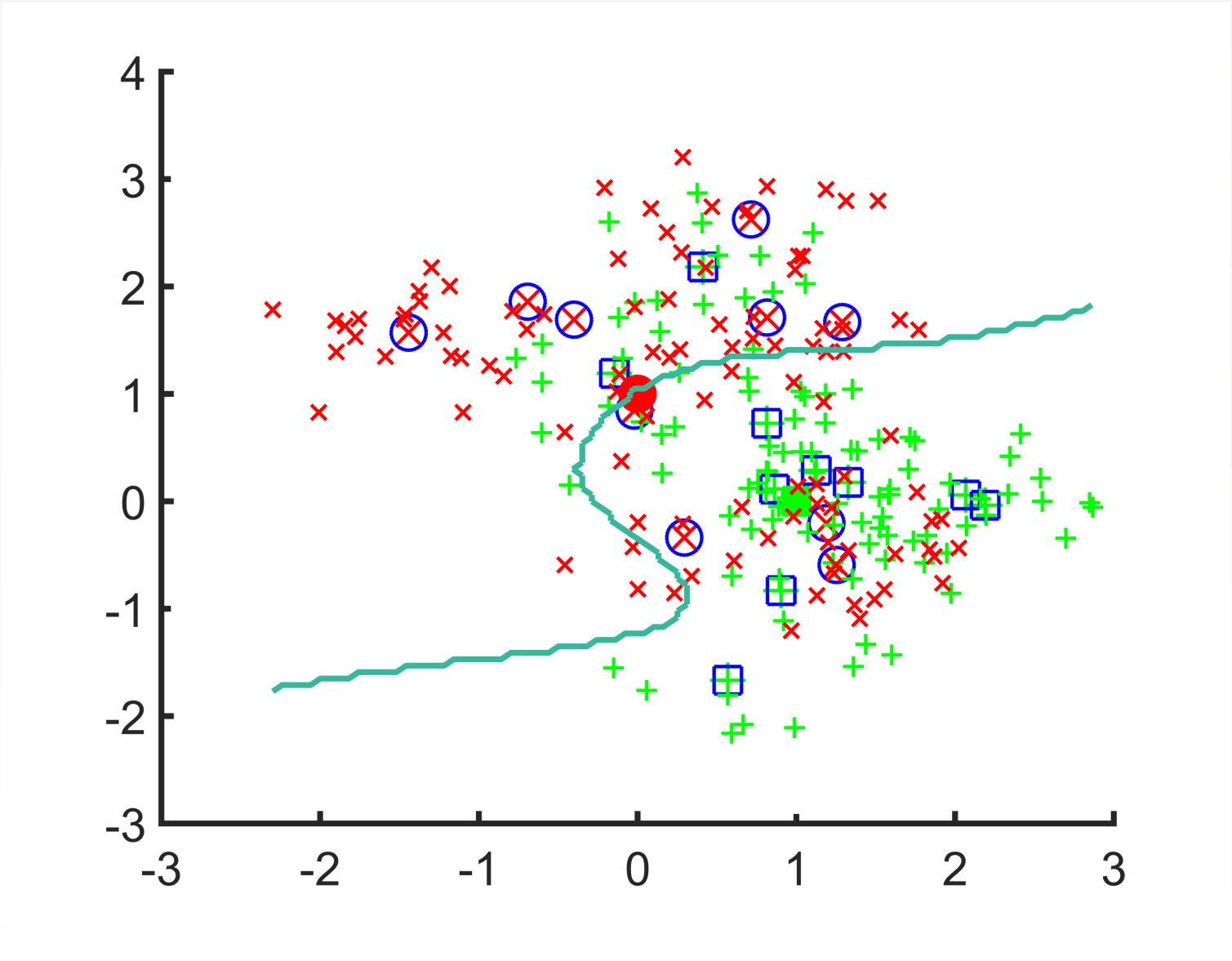
Separating hyperplane, $C = 1$, $\varepsilon = 0.2$



Separating hyperplane, $C = 10000$, $\varepsilon = 5$

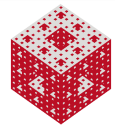


Separating hyperplane, $C = 0.01$, $\varepsilon = 5$



Remark

- *For a fixed C, larger ε produces a more localized/detailed separator*
- *For a fixed ε, larger C produces a more localized/detailed separator*



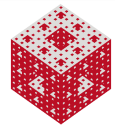
Effects of margin and number of support vectors

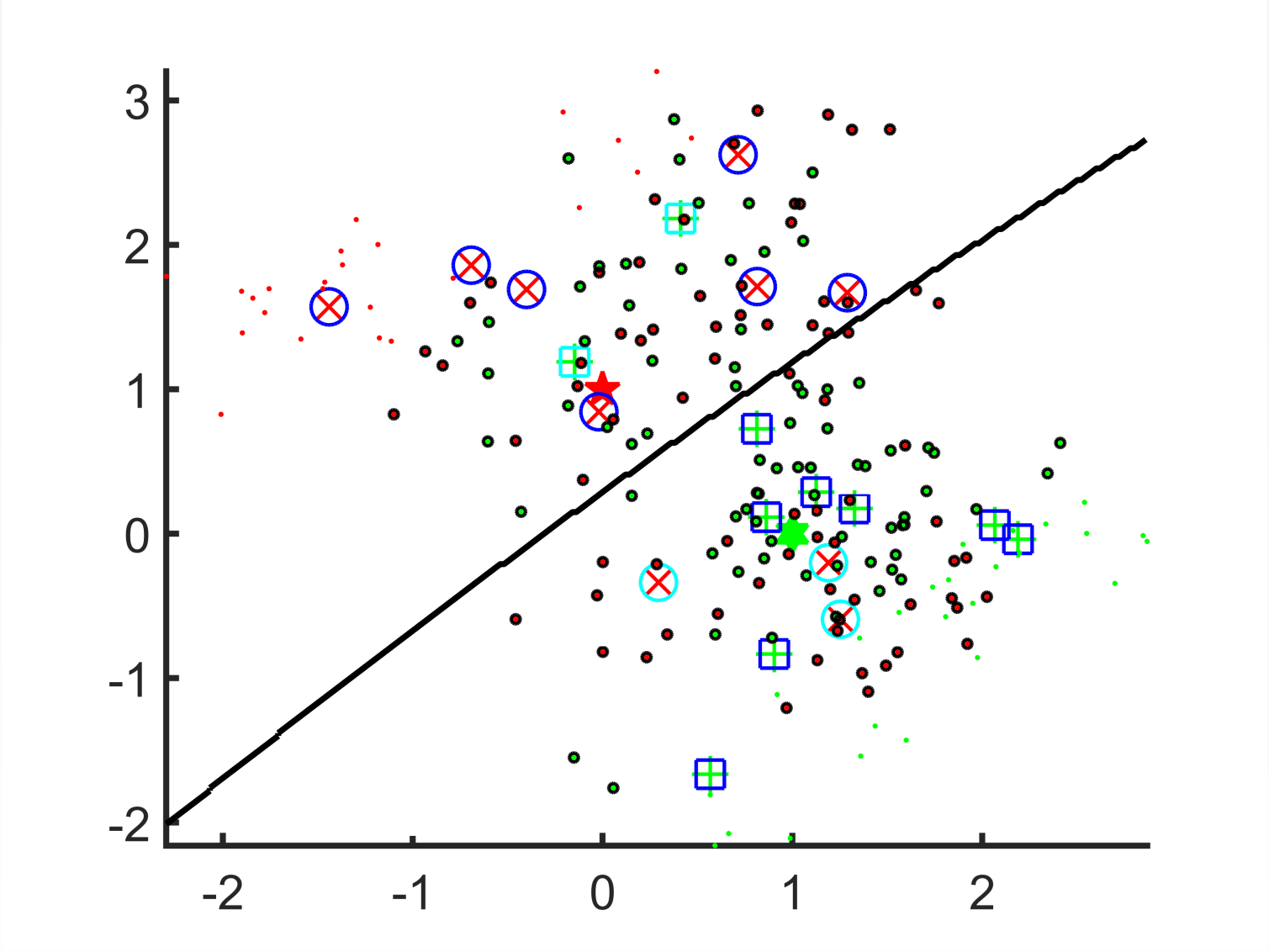
We compare the number of missed classifications (out of 20 total tests) with the margin $1/\|\boldsymbol{w}\|$ and the required number of support vectors.

We look at three experiments:

- fix $C = 10000$ and $\varepsilon = 0.01$,
- fix $C = .6$ and vary ε,
- fix $\varepsilon = 1$ and vary C.

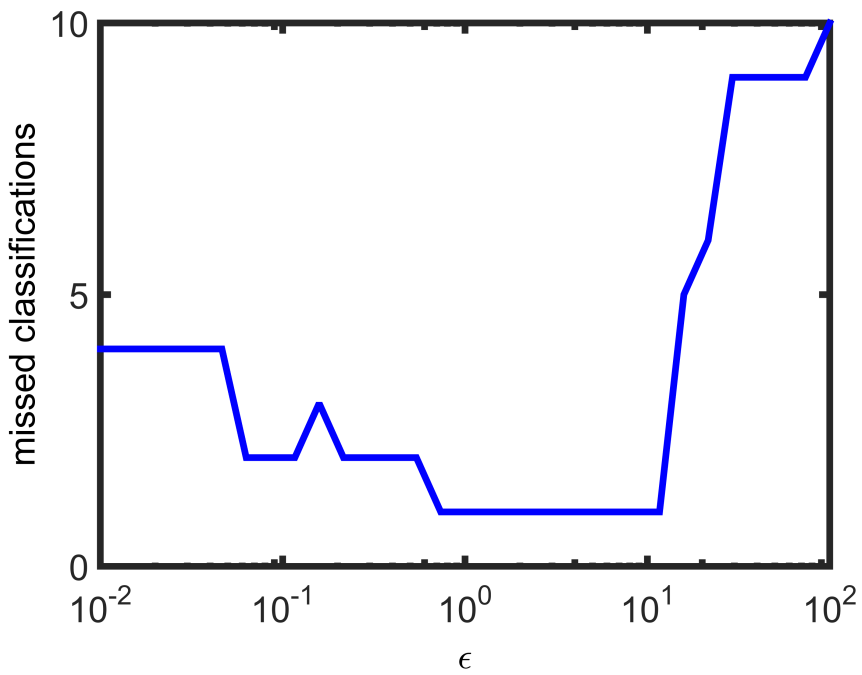
MATLAB code for this example is provided in `SVM2.m` which uses `SVM_Setup.m` and `gqr_fitsvm.m`.

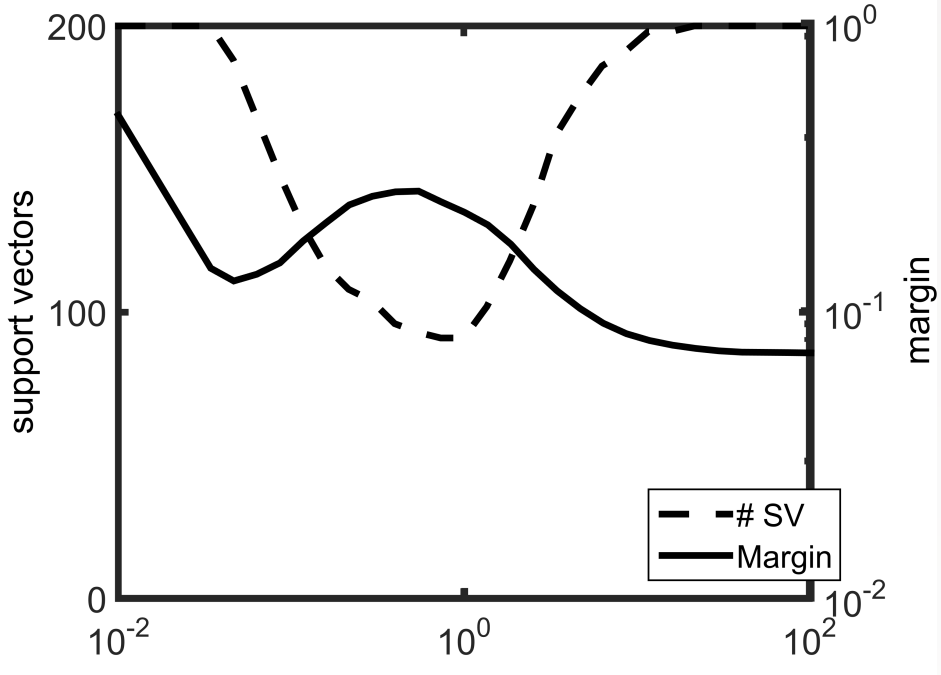




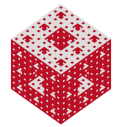
Support vectors marked with ∘, misclassifications with ◯, □.

$C = 0.6$, variable ε

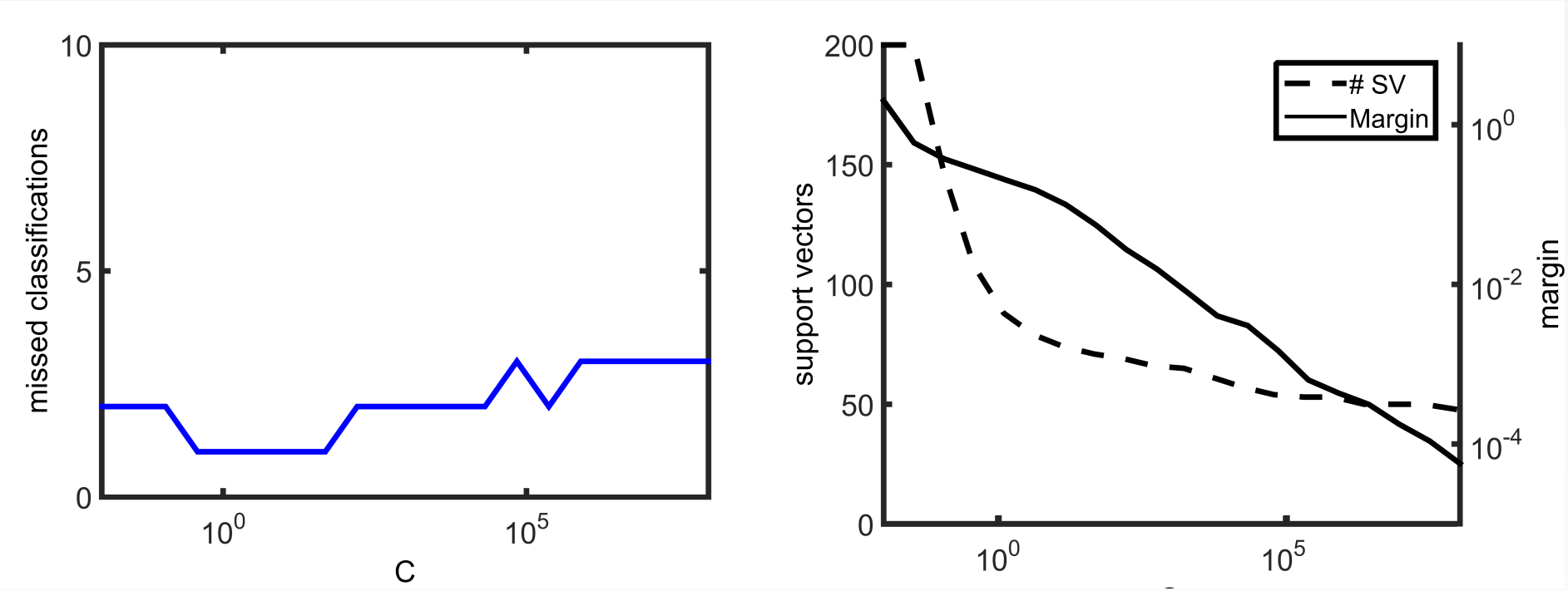




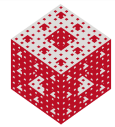
- The margin does not appear to be useful in determining optimal ε value (it grows unboundedly as $\varepsilon \to 0$).
- Minimizing the number of support vectors seems to suggest an optimal region for ε and it helps with computational efficiency.



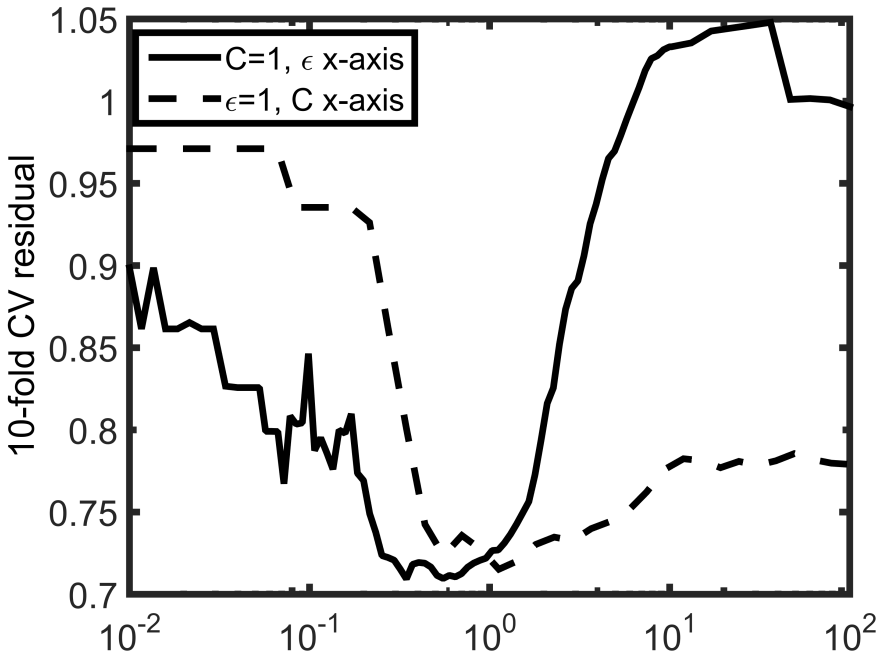
$\varepsilon = 1$, variable C

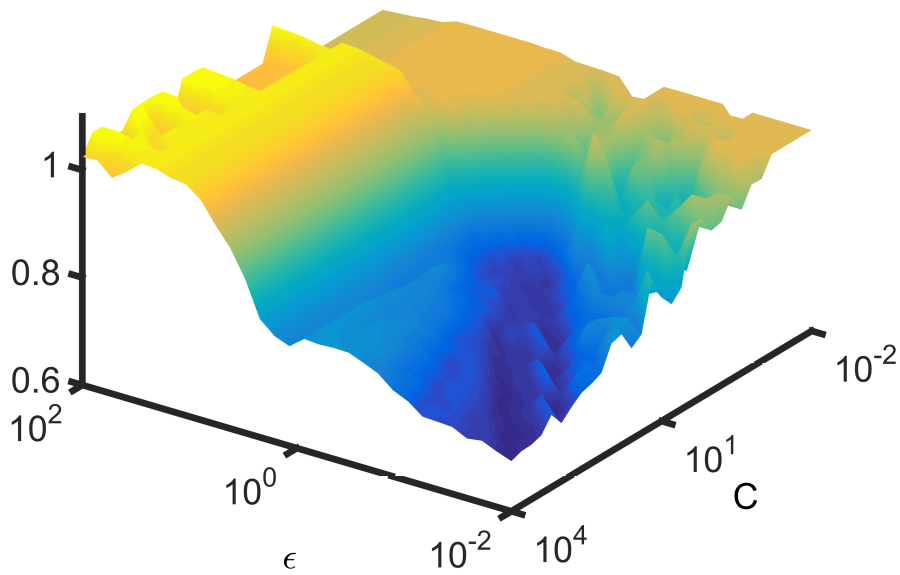


- All C values produce decent results.
- Large C values require fewer support vectors for evaluation (but more time in the optimization solution because a larger search space is used).

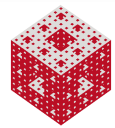


Use of 10-fold CV to estimate C and ε





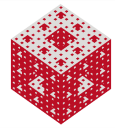
MATLAB code for this example is provided in `SVM3.m` which uses `SVM_Setup.m`, `gqr_svmcv.m` and `gqr_fitsvm.m`.



Remark

The example just discussed uses a linearly separable pattern since the population centers $(0,1)$ *and* $(1,0)$ *are linearly separable.*

Because the $\varepsilon \to 0$ *limit of Gaussians is a polynomial, it is reasonable to conclude that, with infinitely much data drawn from those populations, the optimal SVM would have* $\varepsilon \to 0$ *to produce a line.*

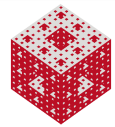


Remark

The example just discussed uses a linearly separable pattern since the population centers $(0,1)$ *and* $(1,0)$ *are linearly separable.*

Because the $\varepsilon \to 0$ *limit of Gaussians is a polynomial, it is reasonable to conclude that, with infinitely much data drawn from those populations, the optimal SVM would have* $\varepsilon \to 0$ *to produce a line.*

We therefore next consider a different pattern which is not linearly separable.



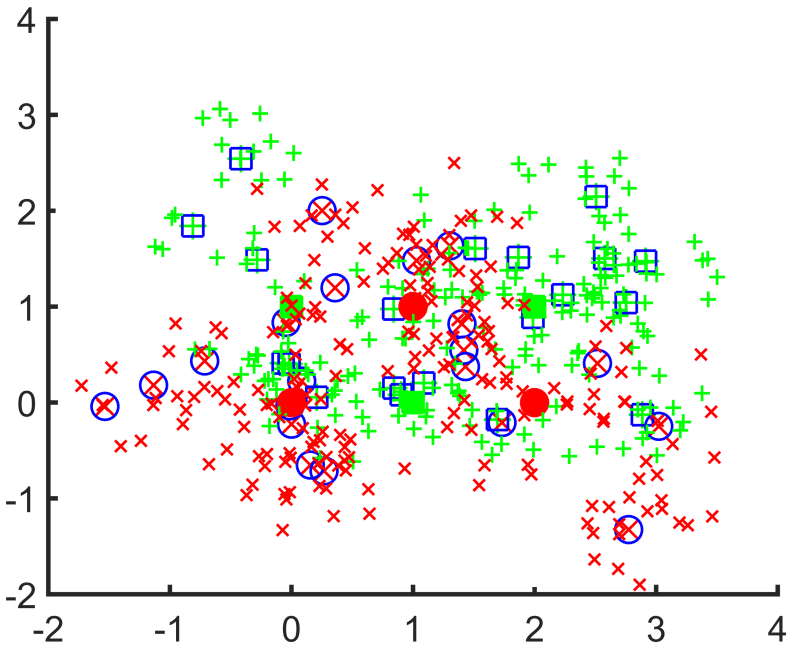
Pattern that is not linearly separable

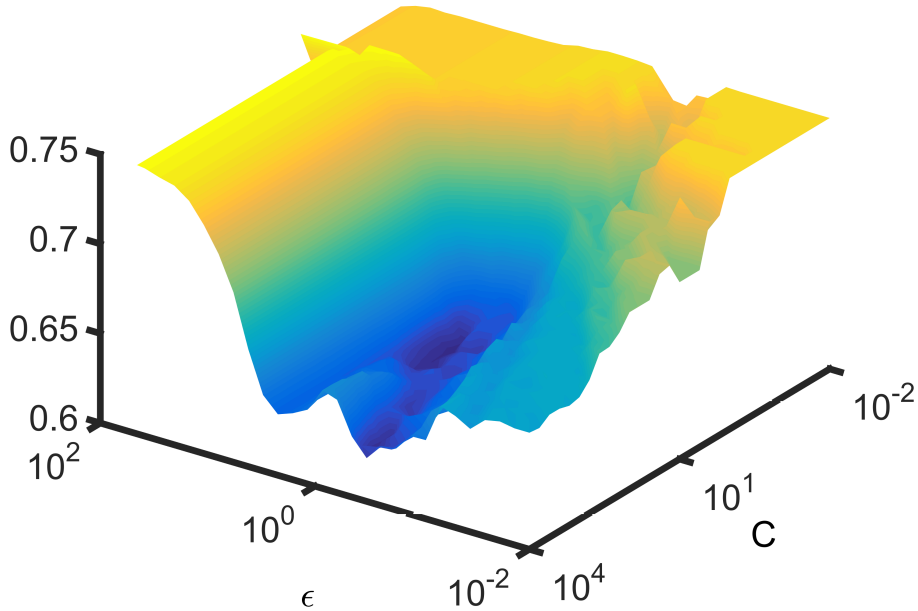
We want to classify data as coming from one of two populations:

- population 1 (denoted by ◯ and $\times$) with centers at $\{(0,0),(1,1),(2,0)\}$ (filled ◯), and
- population 2 (denoted by □ and $+$), with centers at $\{(0,1),(1,0),(2,1)\}$ (filled □).

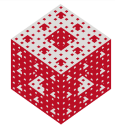
Test points (large $\times$, $+$) and training points (small $\times$, $+$) are shown in the figure.

MATLAB code for this example is provided in `SVM4.m` which uses `SVM_Setup.m`, `gqr_svmcv.m` and `gqr_fitsvm.m`.





- Note that small ε can no longer produce the optimal CV residual.
- Smaller ε causes an increase in the CV residual, likely because the tendency towards polynomial behavior as $\varepsilon \to 0$ is not desirable when learning this pattern.



Computational consideration for classification with kernel SVMs

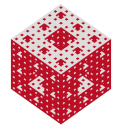
SVMs are generally more popular than RBF networks.

- On the one hand, SVMs may require many fewer kernel centers for evaluation, i.e., they have a spare representation (only the nonzero coefficients must be included).

Computational consideration for classification with kernel SVMs

SVMs are generally more popular than RBF networks.

- On the one hand, SVMs may require many fewer kernel centers for evaluation, i.e., they have a spare representation (only the nonzero coefficients must be included).
- However, solving the quadratic program (2) is more expensive than solving the linear system (1).



Computational consideration for classification with kernel SVMs

SVMs are generally more popular than RBF networks.

- On the one hand, SVMs may require many fewer kernel centers for evaluation, i.e., they have a spare representation (only the nonzero coefficients must be included).
- However, solving the quadratic program (2) is more expensive than solving the linear system (1).

We now

- look at that cost as a function of ε and C, and

Computational consideration for classification with kernel SVMs

SVMs are generally more popular than RBF networks.

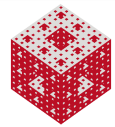
- On the one hand, SVMs may require many fewer kernel centers for evaluation, i.e., they have a spare representation (only the nonzero coefficients must be included).
- However, solving the quadratic program (2) is more expensive than solving the linear system (1).

We now

- look at that cost as a function of ε and C, and
- present a strategy for exploiting the low rank eigenfunction representation for small ε to decrease cost.

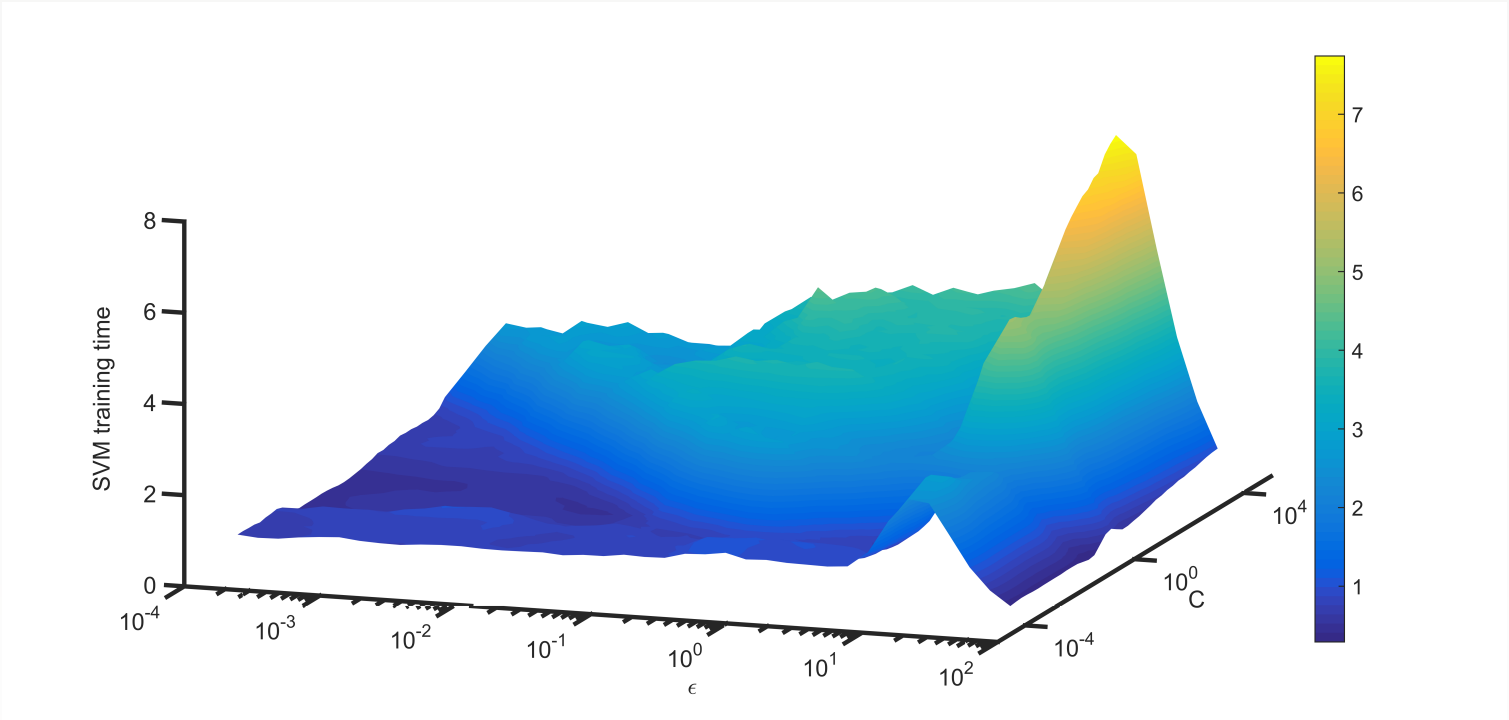
Remark

- *We use the* `quadprog` *solver with the algorithm* `interior-point-convex` *from* MATLAB*'s Optimization Toolbox with initial guess* $C/2$ *times a vector of ones.*
- *As always for iterative solvers, a good initial guess helps speed up convergence.*

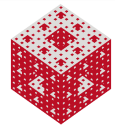


Dependence of training cost of ε and C

We consider the linearly separable example from above, but now use 400 training points.



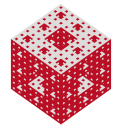
MATLAB code for this example is provided in `SVM4.m` which uses `gqr_fitsvm.m`.



Remark

The solution time for the quadratic program clearly depends on ε and C.

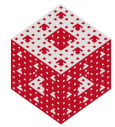
- *Very large ε and very small C seem to be solved quickly:*
  - *large ε because the alertkernel is very localized,*
  - *and small C because the solution domain is very small and quickly searched.*
- *Larger values of C seem to always take longer, likely because the search space is increasing.*



Low-rank approximations via kernel eigenfunctions

Earlier we used the Hilbert–Schmidt SVD to avoid ill-conditioning of the kernel matrix K.

However, this does not help here because the inverse of the kernel matrix is not needed during the quadratic program solution.



Low-rank approximations via kernel eigenfunctions

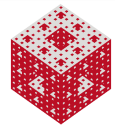
Earlier we used the Hilbert–Schmidt SVD to avoid ill-conditioning of the kernel matrix K.

However, this does not help here because the inverse of the kernel matrix is not needed during the quadratic program solution.

Instead, we may use the eigenfunction expansion

$$K = \Phi \Lambda \Phi^T$$

to produce a low-rank approximation of K and exploit this structure to decrease the cost of the quadratic program.



Low-rank approximations via kernel eigenfunctions

Earlier we used the Hilbert–Schmidt SVD to avoid ill-conditioning of the kernel matrix K.

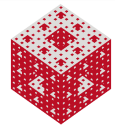
However, this does not help here because the inverse of the kernel matrix is not needed during the quadratic program solution.

Instead, we may use the eigenfunction expansion

$$K = \Phi \Lambda \Phi^T$$

to produce a low-rank approximation of K and exploit this structure to decrease the cost of the quadratic program.

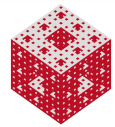
Given N input points and a small ε, only a very low number M of eigenfunctions may be needed to accurately approximate K.



The quadratic program (2) can be written in matrix form as

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \ & \frac{1}{2}\boldsymbol{\alpha}^T \mathrm{D}_{\boldsymbol{y}} \mathrm{K} \mathrm{D}_{\boldsymbol{y}} \boldsymbol{\alpha} - \boldsymbol{e}^T \boldsymbol{\alpha} \\ \text{subject to } & \boldsymbol{y}^T \boldsymbol{\alpha} = 0, \\ & \boldsymbol{\alpha} \in [0, C]^N, \end{aligned} \tag{3}$$

where $\mathrm{D}_{\boldsymbol{y}}$ is a diagonal matrix with $\boldsymbol{y}$ on the diagonal, and $\boldsymbol{e}$ is a vector of all ones.

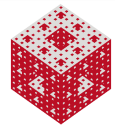


The quadratic program (2) can be written in matrix form as

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \ & \frac{1}{2}\boldsymbol{\alpha}^T \mathrm{D}_{\boldsymbol{y}} \mathrm{K} \mathrm{D}_{\boldsymbol{y}} \boldsymbol{\alpha} - \boldsymbol{e}^T \boldsymbol{\alpha} \\ \text{subject to } & \boldsymbol{y}^T \boldsymbol{\alpha} = 0, \\ & \boldsymbol{\alpha} \in [0, C]^N, \end{aligned} \tag{3}$$

where $\mathrm{D}_{\boldsymbol{y}}$ is a diagonal matrix with $\boldsymbol{y}$ on the diagonal, and $\boldsymbol{e}$ is a vector of all ones.
Using $\mathrm{K} \approx (\Lambda^{1/2}\Phi)^T(\Lambda^{1/2}\Phi)$, we can rephrase this problem as [FS02, ZTK08]

$$\begin{aligned} \min_{\boldsymbol{\eta},\boldsymbol{\alpha}} \ & \frac{1}{2} \begin{pmatrix} \boldsymbol{\eta}^T & \boldsymbol{\alpha}^T \end{pmatrix} \begin{pmatrix} \mathrm{I}_M & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{\eta} \\ \boldsymbol{\alpha} \end{pmatrix} - \begin{pmatrix} 0 & \boldsymbol{e}^T \end{pmatrix} \begin{pmatrix} \boldsymbol{\eta} \\ \boldsymbol{\alpha} \end{pmatrix} \\ \text{subject to } & \begin{pmatrix} 0 & \boldsymbol{y}^T \\ -\mathrm{I}_M & \Lambda^{1/2}\Phi^T \mathrm{D}_{\boldsymbol{y}} \end{pmatrix} \begin{pmatrix} \boldsymbol{\eta} \\ \boldsymbol{\alpha} \end{pmatrix} = 0, \\ & \boldsymbol{\alpha} \in [0, C]^N, \quad \boldsymbol{\eta} \in \mathbb{R}^M. \end{aligned} \tag{4}$$

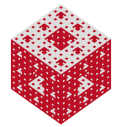


Remark

Although this system is of size $N + M$ (and the original system was only size N), the cost of solving this system may be much lower because of the extremely simple structure of the Hessian.

This sparsity, in comparison to H which may be fully dense, allows for cheap matrix-vector products and decompositions, both of which may all for a faster quadratic program solve.

Note that the η values are inconsequential in making predictions with the SVM.



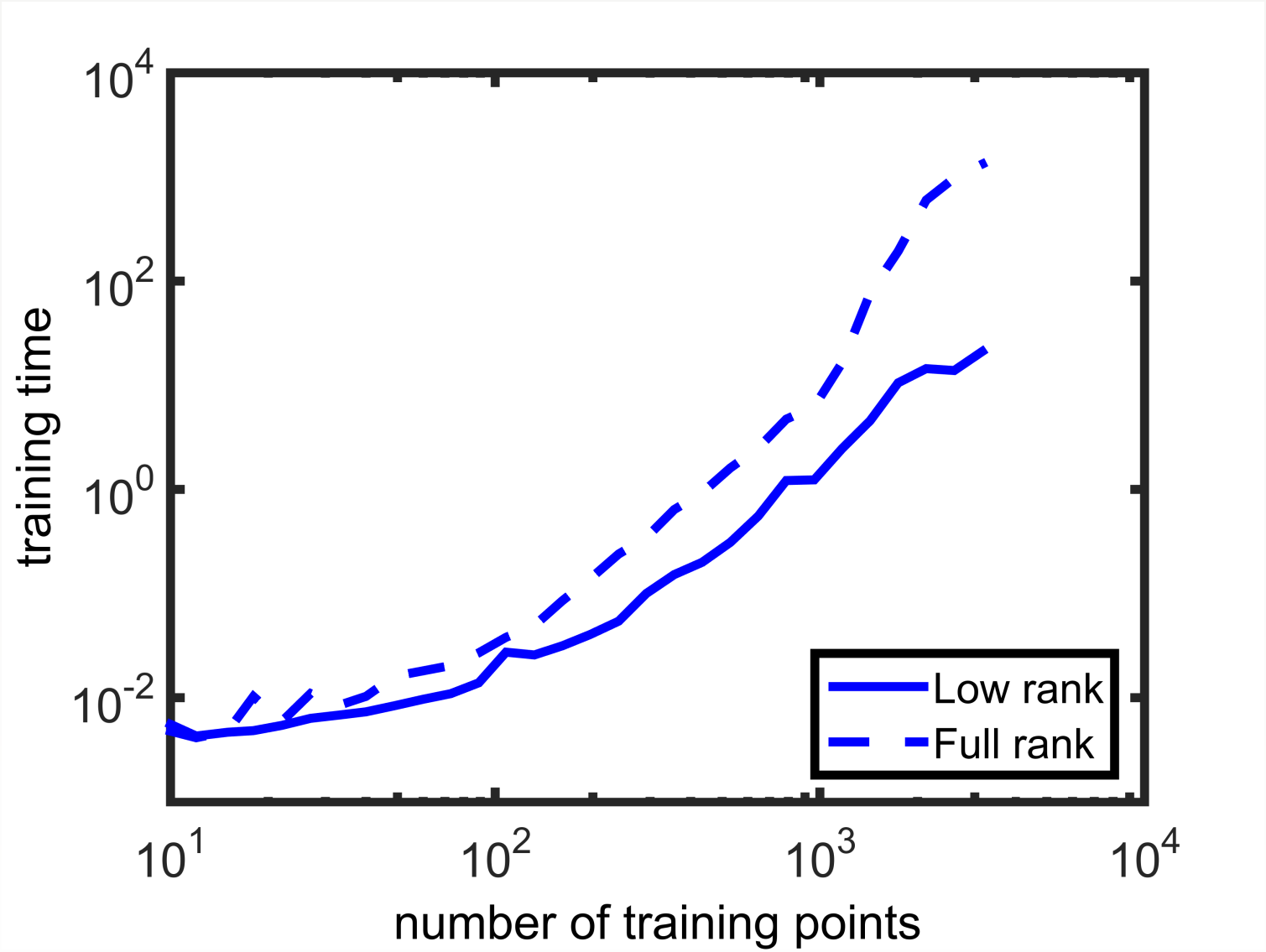
Low-rank vs. full-rank approximation

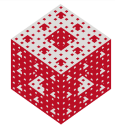
We use the same setup as before (given in `SVM5.m`).

We study the cost of solving the quadratic program and training the SVM. Minimizing this cost is an important topic in machine learning (see, e.g., [YDD04, FL02, LLZ+11]).

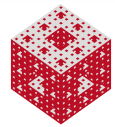
Increasingly large sets of input points are considered and the cost of solving the full rank problem (3) is compared to solving the low rank problem (4).

The kernel is parameterized with $\varepsilon = .01$ and $C = 1$ and the eigenfunctions of the Gaussian use $\alpha = 10^6$.


10^4
10^2
10^0
10^{-2}
training time
10^1
10^2
10^3
10^4
number of training points
Low rank
Full rank



Outline



Linear support vector regression

As for classification, we again start with a linear approximation and assume that

$$s(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{w} + b.$$

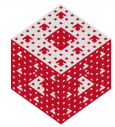
If we use the ϵ-insensitive loss function

$$L(y, s(\boldsymbol{x})) = \max(|y - s(\boldsymbol{x})| - \epsilon, 0)$$

then the primal unconstrained minimization problem is given by

$$\min_{\boldsymbol{w},b} \left[\frac{1}{N} \sum_{i=1}^{N} \max(|y_i - s(\boldsymbol{x}_i)| - \epsilon, 0) + \mu \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w} \right],$$

where, as before, μ is an appropriately chosen regularization parameter.



Constrained minimization problem

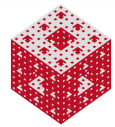
Using slack variables as in the classification case we have the analogous constrained minimization problem

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi},\boldsymbol{\xi}^*} \left[\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) \right]$$

$$\text{subject to } s(\boldsymbol{x}_i) - y_i \leq \epsilon + \xi_i, \quad i = 1,\ldots,N,$$

$$y_i - s(\boldsymbol{x}_i) \leq \epsilon + \xi_i^*, \quad i = 1,\ldots,N,$$

$$\xi_i, \xi_i^* \geq 0.$$



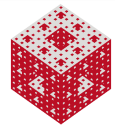
Dual problem

In the dual formulation we need to solve the constrained quadratic programming problem

$$\min_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} \epsilon \sum_{i=1}^{N}(\alpha_i^* + \alpha_i) - \sum_{i=1}^{N} y_i(\alpha_i^* - \alpha_i) + \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)\boldsymbol{x}_i^T\boldsymbol{x}_j$$

subject to $0 \leq \alpha_i, \alpha_i^* \leq C$,

$$\sum_{i=1}^{N}(\alpha_i^* - \alpha_i) = 0.$$



Dual problem

In the dual formulation we need to solve the constrained quadratic programming problem

$$\min_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} \epsilon \sum_{i=1}^{N}(\alpha_i^* + \alpha_i) - \sum_{i=1}^{N} y_i(\alpha_i^* - \alpha_i) + \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)\boldsymbol{x}_i^T\boldsymbol{x}_j$$

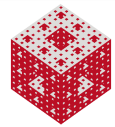
subject to $0 \leq \alpha_i, \alpha_i^* \leq C$,

$$\sum_{i=1}^{N}(\alpha_i^* - \alpha_i) = 0.$$

Once we've found the dual variables α_i and α_i^*, the SVM regression function is given by

$$s(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{w} + b = \sum_{i=1}^{N}(\alpha_i^* - \alpha_i)\boldsymbol{x}^T\boldsymbol{x}_i + b,$$

i.e., $\boldsymbol{w} = \sum_{i=1}^{N}(\alpha_i^* - \alpha_i)\boldsymbol{x}_i$.



Remark

- *The computation of the bias term b follows from the KKT conditions (for details see [SS02]) and is similar in spirit to the classification setting, i.e.,*

$$b = y_i - \boldsymbol{x}_i^T \boldsymbol{w} - \epsilon \quad \text{for } \alpha_i \in (0, C),$$
$$b = y_i - \boldsymbol{x}_i^T \boldsymbol{w} + \epsilon \quad \text{for } \alpha_i^* \in (0, C).$$

As before, any one of these will theoretically suffice, but for stability reasons it is better to compute b via an average over all candidates.

Remark

- *The computation of the bias term b follows from the KKT conditions (for details see [SS02]) and is similar in spirit to the classification setting, i.e.,*

$$b = y_i - \boldsymbol{x}_i^T \boldsymbol{w} - \epsilon \quad \text{for } \alpha_i \in (0, C),$$
$$b = y_i - \boldsymbol{x}_i^T \boldsymbol{w} + \epsilon \quad \text{for } \alpha_i^* \in (0, C).$$

  As before, any one of these will theoretically suffice, but for stability reasons it is better to compute b via an average over all candidates.

- *As in the classification setting, $\alpha_i^* - \alpha_i \neq 0$ only for some i, and the corresponding measurements $\boldsymbol{x}_i$ are called the support vectors.*

Remark

- *The computation of the bias term b follows from the KKT conditions (for details see [SS02]) and is similar in spirit to the classification setting, i.e.,*

$$b = y_i - \boldsymbol{x}_i^T \boldsymbol{w} - \epsilon \quad \text{for } \alpha_i \in (0, C),$$
$$b = y_i - \boldsymbol{x}_i^T \boldsymbol{w} + \epsilon \quad \text{for } \alpha_i^* \in (0, C).$$

  As before, any one of these will theoretically suffice, but for stability reasons it is better to compute b via an average over all candidates.

- *As in the classification setting, $\alpha_i^* - \alpha_i \neq 0$ only for some i, and the corresponding measurements $\boldsymbol{x}_i$ are called the support vectors.*

- *For more details see, e.g., [HTF09, Chapter 12], [SS02, Chapter 9].*

Nonlinear support vector regression

As for classification, we obtain a nonlinear "kernelized" regression fit if we map the data into feature space and then use kernels.

This is straightforward and completely analogous to the classification setting.

The resulting dual problem is

$$\min_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} \epsilon \sum_{i=1}^{N}(\alpha_i^* + \alpha_i) - \sum_{i=1}^{N} y_i(\alpha_i^* - \alpha_i) + \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

subject to $0 \leq \alpha_i, \alpha_i^* \leq C$,

$$\sum_{i=1}^{N}(\alpha_i^* - \alpha_i) = 0,$$

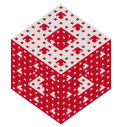
so that

$$s(\boldsymbol{x}) = \Phi(\boldsymbol{x}^T)\boldsymbol{w} + b = \sum_{i=1}^{N}(\alpha_i^* - \alpha_i)K(\boldsymbol{x}, \boldsymbol{x}_i) + b,$$

i.e., $\boldsymbol{w} = \sum_{i=1}^{N}(\alpha_i^* - \alpha_i)\Phi(\boldsymbol{x}_i)$ and

$$b = y_i - \sum_{j=1}^{N}(\alpha_j^* - \alpha_j)K(\boldsymbol{x}_i, \boldsymbol{x}_j) - \epsilon \quad \text{for } \alpha_i \in (0, C),$$

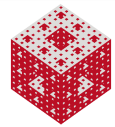
$$b = y_i - \sum_{j=1}^{N}(\alpha_j^* - \alpha_j)K(\boldsymbol{x}_i, \boldsymbol{x}_j) + \epsilon \quad \text{for } \alpha_i^* \in (0, C).$$



Remark

Many more details on all aspects of machine learning can be found, e.g., in the

- *books [Alp09, HTF09, RW06, SS02, STC04, SC08] or*
- *survey papers [EPP00, MMn06, Orr96].*



References I

[AC10] S. Arlot and A. Celisse, *A survey of cross-validation procedures for model selection*, Statistics surveys **4** (2010), 40–79.

[Alp09] Ethem Alpaydin, *Introduction to Machine Learning*, 2nd ed., The MIT Press, December 2009.

[Cov65] Thomas M. Cover, *Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition*, IEEE Transactions on Electronic Computers **14** (1965), 326–334.

[CW79] Peter Craven and Grace Wahba, *Smoothing noisy data with spline functions*, Numerische Mathematik **31** (1979), no. 4, 377–403.

[EPP00] Theodoros Evgeniou, Massimiliano Pontil, and Tomaso Poggio, *Regularization networks and support vector machines*, Advances in Computational Mathematics **13** (2000), no. 1, 1–50.

[Fas07] G. E. Fasshauer, *Meshfree Approximation Methods with* MATLAB, Interdisciplinary Mathematical Sciences, vol. 6, World Scientific Publishing Co., Singapore, 2007.

References II

[FL02] G. W. Flake and S. Lawrence, *Efficient SVM regression training with SMO*, Machine Learning **46** (2002), no. 1-3, 271–290.

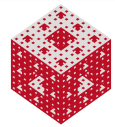
[FS02] Shai Fine and Katya Scheinberg, *Efficient SVM training using low-rank kernel representations*, J. Mach. Learn. Res. **2** (2002), 243–264.

[GHW79] G. H. Golub, M. Heath, and G. Wahba, *Generalized cross-validation as a method for choosing a good ridge parameter*, Technometrics **21** (1979), no. 2, 215–223.

[GVM97] G. H. Golub and U. Von Matt, *Generalized cross-validation for large-scale problems*, Journal of Computational and Graphical Statistics **6** (1997), no. 1, 1–34.

[HTF09] T. Hastie, R. Tibshirani, and J. Friedman, *Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed., Springer Series in Statistics, Springer, New York, 2009.

[KW71] G. Kimeldorf and G. Wahba, *Some results on Tchebycheffian spline functions*, J. Math. Anal. Applic. **33** (1971), 82–95.



References III

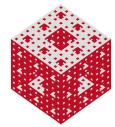
[LLZ+11] Y. Lin, F. Lv, S. Zhu, M. Yang, T. Cour, K. Yu, L. Cao, and T. Huang, *Large-scale image classification: fast feature extraction and SVM training*, 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2011, pp. 1689–1696.

[MMn06] Javier M. Moguerza and Alberto Muñoz, *Support vector machines with applications*, Statistical Science **21** (2006), no. 3, 322–336.

[Orr96] M. J. L. Orr, *Introduction to radial basis function networks*, Tech. report, University of Edinburgh, Centre for Cognitive Sciences, 1996.

[Pla99] John C. Platt, *Fast training of support vector machines using sequential minimal optimization*, Advances in Kernel Methods (Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola, eds.), MIT Press, Cambridge, MA, USA, 1999, pp. 185–208.

[PMR+01] T. Poggio, S. Mukherjee, R. Rifkin, A. Rakhlin, and A. Verri, *b*, Tech. report, MIT AI Memo 2001-011, 2001.



References IV

[RW06] C. E. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*, MIT Press, Cambridge, Massachussetts, 2006.

[SC08] I. Steinwart and A. Christmann, *Support Vector Machines*, Information Science and Statistics, Springer, New York, 2008.

[SS02] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, The MIT Press, 2002.

[STC04] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*, Cambridge University Press, 2004.

[YDD04] C. Yang, R. Duraiswami, and L. S. Davis, *Efficient kernel machines using the improved fast Gauss transform*, Advances in neural information processing systems, 2004, pp. 1561–1568.

[ZTK08] K. Zhang, I. W Tsang, and J. T Kwok, *Improved Nyström low-rank approximation and error analysis*, Proceedings of the 25th international conference on Machine learning, ACM, 2008, pp. 1232–1239.