`

7.1 SVMs: An Introduction

The classical SVM is a supervised linear technique for solving binary classification problems. For illustration, consider Figure 7.1a. Here, in a 2D system, the training data-points belong to two distinct (positive and negative) classes. The task is to find a line/linear boundary that separates these 2 classes. Two sample candidate lines are also shown. While these lines clearly do the stated job, something seems amiss. Each of them passes very close to some of the training data-points. This can cause poor generalization: for example, the shown test observation 'A' lies closer to the positive samples but will get classified as negative class by boundary L2. This clearly is undesirable.

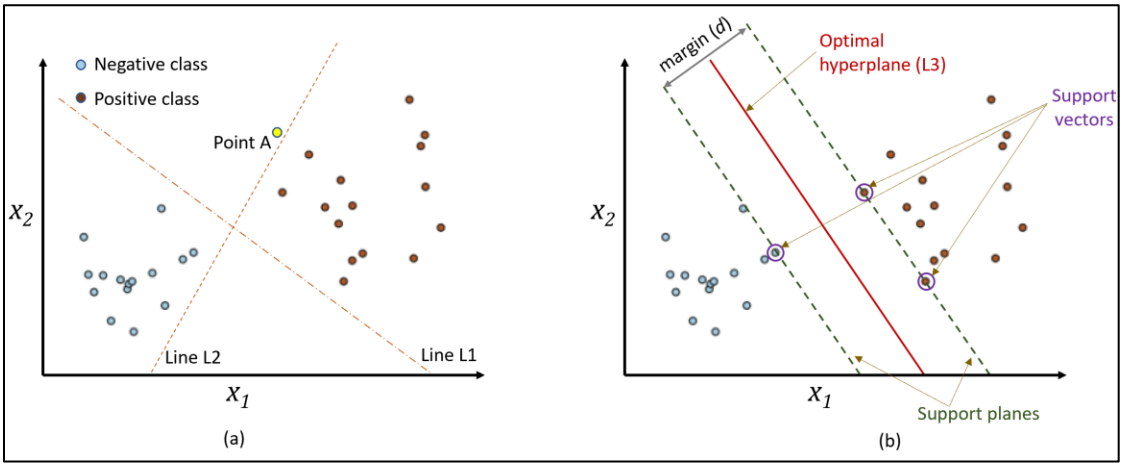


Figure 7.1: (a) Training data distribution with test sample A (b) Optimal separating boundary

The optimal separating line/decision boundary, line L3 in Figure 7.1b, lies as far away as possible from either class of data. L3, as shown, lies midway of the support planes (planes that pass-through training points closest to the separating boundary). During model fitting, SVM simply finds this optimal boundary that corresponds to the maximum margin (distance between the support planes). In Figure 7.1, any other orientation or position of L3 will reduce the margin and will make L3 closer to one class than to the other. Large margins make model predictions robust to small perturbations in the training samples.

Points that lie on the support planes are called support vectors[34] and completely determine the optimal boundary, and hence the name, support vector machines. In Figure 7.1, if support vectors are moved, line L3 may change. However, if any non-support vectors are removed, L3 won't get affected at all. We will see later how the sole dependency on the support vectors imparts computational advantage to the SVMs.

[34] Calling data-points as vectors may seem weird. While this terminology is commonly used in general SVM literature, support vectors refer to the vectors originating from origin with the data-points on support planes as their tips.

`

Mathematical background

Let there be *N* training samples ($\boldsymbol{x}$, *y*) where $\boldsymbol{x}$ is an input vector in *m*-dimensional input space and *y* is the class label (± 1). Let the optimal separating hyperplane (a line in 2D space) be represented as $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ where the model parameters ($\boldsymbol{w}, b$) are found such that

$$\begin{aligned} \boldsymbol{w}^T\boldsymbol{x}_i + b &\geq 1 \quad && \text{for positive samples } (y_i = 1) \\ \boldsymbol{w}^T\boldsymbol{x}_i + b &\leq -1 \quad && \text{for negative samples } (y_i = -1) \end{aligned}$$

The supporting planes get represented as $\boldsymbol{w}^T\boldsymbol{x} + b = 1$ and $\boldsymbol{w}^T\boldsymbol{x} + b = -1$. The above equations are simply stating the requirements that the data points must lie on the correct side of their corresponding support planes. Using simple trigonometry, it can be shown that the margin is given by $\frac{2}{||\boldsymbol{w}||}$ where $||\boldsymbol{w}||$ is vector norm. Therefore, model parameters are found by maximizing $\frac{2}{||\boldsymbol{w}||}$ (or equivalently, minimizing $||\boldsymbol{w}||$) while meeting the above constraints. Specifically, the following optimization problem is solved

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\left|\left|\boldsymbol{w}\right|\right|^2 \\ \text{s.t.} \quad & y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1, \qquad i = 1, \cdots N \end{aligned} \qquad \text{eq. 1}$$

Once the model has been fitted, class predictions for test sample, $\boldsymbol{x_t}$, are made as follows.

$$\hat{y}_t = sign(\boldsymbol{w}^T\boldsymbol{x}_t + b) \quad \Rightarrow \quad \boldsymbol{x}_t \text{ belongs to positive class if it lies on positive side of separating hyperplane}$$

The expression inside the sign function is also called decision function and therefore, positive decision function results in positive class prediction and vice-versa.

The optimization formulation in Eq. 1 and all the others that we will see in this chapter share a very favorable property of possessing a unique global minimum. This is a huge advantage when compared to other powerful ML methods like neural networks where the issue of local minimums can be an inconvenience.

`

Simple linear classification illustration

We will use the toy dataset in Figure 7.1 to illustrate SVM-based classifier modeling in Sklearn.

```
# read data
import numpy as np
data = np.loadtxt('toyDataset.csv', delimiter=',')
X = data[:, [0, 1]]; y = data[:, 2]

# scale model inputs
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# fit SVM model
from sklearn.svm import SVC # for large datasets LinearSVC class is preferable
model = SVC(kernel='linear', C=100)
model.fit(X_scaled, y)

# get details of support vectors
print('# of support vectors:', len(model.support_))

>> # of support vectors: 3
```

The above code provides us the optimal separating boundary shown in Figure 7.1[35]. As with other Sklearn estimators, `predict()` method can be used to predict the class of any test observation. We will soon cover the hyperparameters (`kernel` and `C`) used in the above code.

Hard margin vs soft margin classification

The toy dataset represents an ideal data where all the training samples can be correctly classified via a linear boundary. However, in real-life, some data impurities always creep in as shown in Figure 7.2. Our SVM formulation (in Eq. 1) will fail to find a solution in this case

[35] Check out the online code to see how the separating boundary and support planes are plotted

`

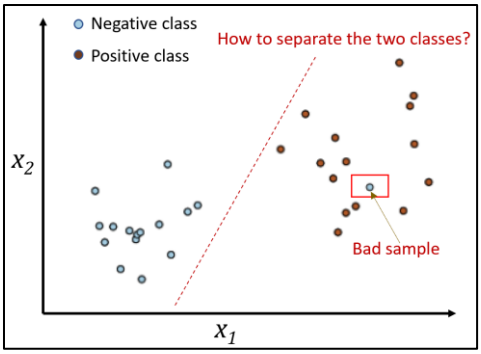


Figure 7.2: Presence of the shown bad sample makes perfect linear separation infeasible

To deal with such scenarios, we add a little flexibility into our SVM optimization program by modifying the constraints as shown below

$$\boldsymbol{w}^T\boldsymbol{x}_i + b \geq 1 - \xi_i \quad \text{for } y_i = 1$$
$$\boldsymbol{w}^T\boldsymbol{x}_i + b \leq -1 + \xi_i \quad \text{for } y_i = -1$$

Here, we use slack variables (ξ_i) to allow each sample the freedom to end up on the wrong side of the support plane and potentially be misclassified during model fitting. However, we would like to keep the number of such violations low as well which we can achieve by penalizing the violations. The revised SVM formulation looks like this

$$\min_{\boldsymbol{w},b,\xi} \quad \frac{1}{2}\left|\left|\boldsymbol{w}\right|\right|^2 + C\sum_{i=1}^{N}\xi_i \qquad \text{eq. 2}$$
$$\text{s.t.} \quad y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \xi_i, \quad i = 1, \cdots N$$
$$\xi_i \geq 0$$

The above formulation is called soft margin classification (as opposed to the previous hard margin classification). Sklearn implements soft margin formulation. The positive constant, C, is a hyperparameter (C=1 in Sklearn by default) and corresponds to the hyperparameter we saw in the previous code. For our toy dataset 2 (in Figure 7.2), with the previously shown code, we end up with the same separating boundary as shown in Figure 7.1. Class prediction expression remains the same as $\hat{y}_t = sign(\boldsymbol{w}^T\boldsymbol{x}_t + b)$.

C as regularization hyperparameter

The slack variables not only help find a solution in the presence of gross impurity, but they also help to avoid overfitting noisy data. For example, consider the scenario in Figure 7.3. If no misclassifications are allowed, we end up with a very small margin, while with a single

misclassification we get a much better margin with potentially better generalization. Therefore, we see that there is a trade-off between margin maximization and training accuracy.

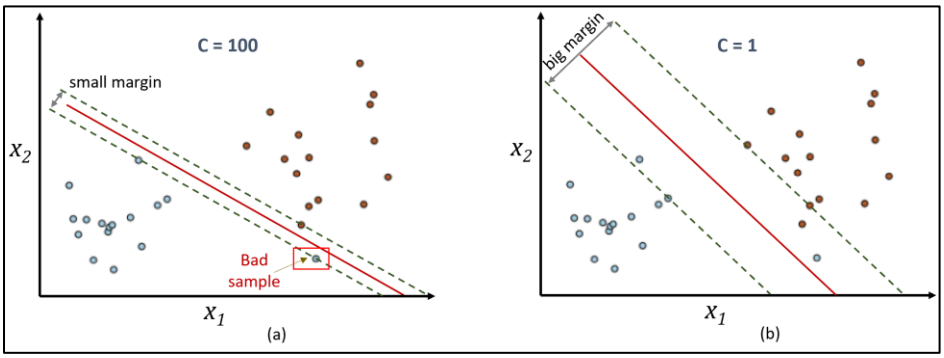


Figure 7.3: (a) Overfitted unregularized boundary (b) Regularized fitting

The hyperparameter C is the knob to control the trade-off. A large value of C implies heavy penalization of the constraint violations which will prevent misclassifications, while small value of C allows more misclassifications during model fitting for better margin.

7.2 The Kernel Trick for Nonlinear Data

While soft margin classification formulation is quite flexible, it won't work for nonlinear classification problems where curved boundaries are warranted. Consider the dataset in Figure 7.4. It is clear that a linear boundary is inappropriate here.

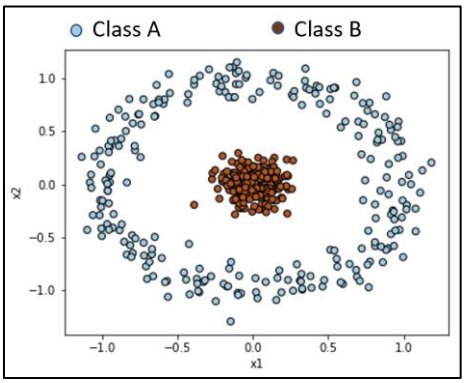


Figure 7.4: Nonlinearly distributed samples

However, all is not lost here. One idea to circumvent this issue is to map the original input variables/features into a higher dimensional space where they become linearly separable. For the data in Figure 7.4, the following transformation would work quite well

$$\varphi(\boldsymbol{x}) = \varphi(x_1, x_2) = [z_1, z_2, z_3] = [x_1, x_2, x_1^2 + x_2^2]$$

`

As we can see in Figure 7.5, in the 3D space, the data is easily linearly separable!

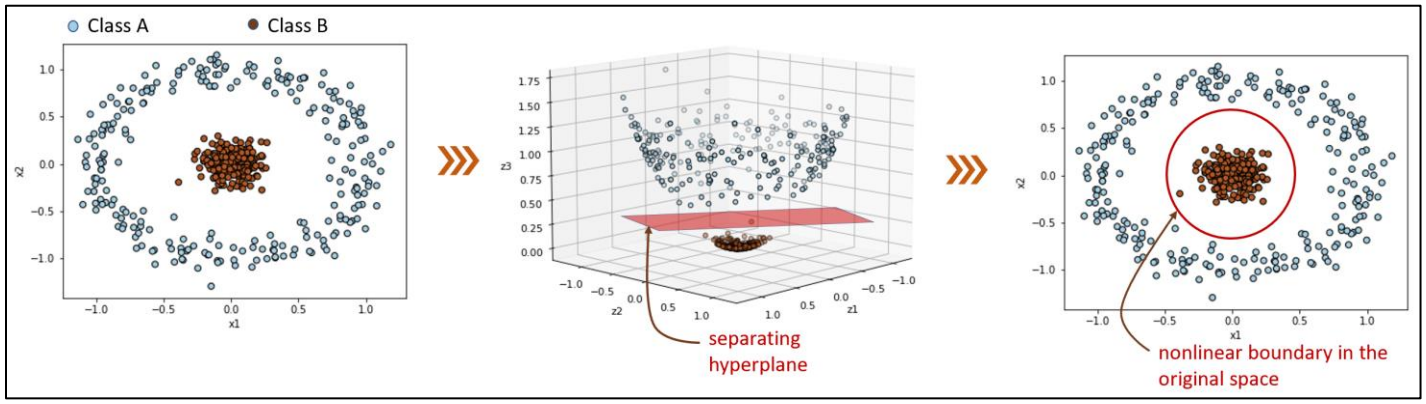


Figure 7.5: Nonlinear mapping to higher dimensional enabling linear segregation

SVM can be trained on the new feature space to obtain the optimal separating hyperplane. Any new test data point can be transformed via the same mapping function, φ, for its class determination via the fitted SVM model. While this solution looks great, there remains a small issue. How do we find an appropriate mapping for a high-dimensional input dataset? As it turns out, you don't need to find this mapping explicitly and this is made possible by a neat 'kernel trick'. Let's learn what is this trick and how it is used. For this we will revisit the mathematical background of SVMs.

Mathematical background (revisited)

SVMs are not solved in the form shown in Eq. 2. Instead, the following equivalent form is solved

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_i \boldsymbol{x}_i^T \boldsymbol{x}_j \; - \; \sum_{i=1}^{N}\alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^{N} y_i \alpha_i \; = \; 0 \\ & 0 \; \leq \; \alpha_i \leq C \quad i = 1, \cdots N \end{aligned} \qquad \text{eq. 3}$$

Here, the optimization parameters $\boldsymbol{w}, b$ have been replaced by α_s (also called Lagrange multipliers). This equivalent form is called dual form (which you may remember from your optimization classes; it's perfectly fine if you have not encountered this term before). Once α_s have been found, $\boldsymbol{w}$ and b can be computed via the following

$$\boldsymbol{w} \; = \sum_{i=1}^{N} \alpha_i \, y_i \boldsymbol{x}_i, \qquad \boldsymbol{b} = \frac{1}{N_s}\sum_{i\in\{SV\}} 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i$$

`

where N_s is number of support vectors and {SV} is the set of support vector indices. Any test data point can be classified as

$$\hat{y}_t = sign(\sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i^T \boldsymbol{x}_t + b) \quad \text{eq. 4}$$

In the dual formulation, it is found that α_s are non-zero for only the support vectors and zero for the rest of the training samples. This implies that Eq. 4 can be reduced to

$$\hat{y}_t = sign(\sum_{i\in\{SV\}} \alpha_i y_i \boldsymbol{x}_i^T \boldsymbol{x}_t + b) \quad \text{eq. 5}$$

Strictly speaking, support vectors need not lie on the separating hyperplane. For soft margin classification, data-points with non-zero slacks are also support vectors and their $\boldsymbol{\alpha_s}$ are non-zero (defining characteristic of the support vectors). The presence/absence of the support vectors impacts the solution (the objective function and/or the model parameters).

At this point, you may be wondering why have we made things more complicated; why not solve the problem in the original form (Eq. 2) which seemed more interpretable? The reason for doing this will become clear to you very soon. For now, imagine that you are solving the nonlinear problem where SVM finds a separating hyperplane in the higher dimension. Eq. 3 will look like the following

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_i \varphi(\boldsymbol{x}_i)^{\boldsymbol{T}} \varphi(\boldsymbol{x}_j) - \sum_{i=1}^{N} \alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^{N} y_i \alpha_i = 0 \\ & 0 \le \alpha_i \le C \quad i = 1, \cdots N \end{aligned} \quad \text{eq. 6}$$

and Eq. 5 becomes

$$\hat{y}_t = sign(\sum_{i\in\{SV\}} \alpha_i y_i \varphi(\boldsymbol{x}_i)^T \varphi(\boldsymbol{x}_j) + b) \quad \text{eq. 7}$$

The most crucial observation here is that the transformed variables ($\varphi(\boldsymbol{x})$) appear only as inner (dot) products. Now, if somehow, we knew the values of these dot products then we would not need to know the exact form of $\varphi(\boldsymbol{x})$ or the mapping at all and we can fit our SVM model. This is made possible via kernel functions, $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, which provide the relationship

`

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \varphi(\boldsymbol{x}_i)^T \varphi(\boldsymbol{x}_j)$$

As you can see, kernel functions allow us to compute dot products in the 'unknown' transformed space as a function of vectors in the original space! There are several forms of kernel functions to choose from and thus this choice becomes a model hyperparameter. Once a kernel function is chosen, Eq. 6 becomes

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_{i=1}^{N} \alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^{N} y_i \alpha_i = 0 \\ & 0 \le \alpha_i \le C \quad i = 1, \cdots N \end{aligned} \qquad \text{eq. 8}$$

Above is the form in which a SVM model is fitted and predictions are made as follows

$$\hat{y_t} = sign(\sum_{i \in \{SV\}} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}_t) + b) \qquad \text{eq. 9}$$

These kernel functions allow us to obtain powerful nonlinear classifiers while retaining all the benefits of the original linear SVM method!

Types of kernel functions

Table below lists some commonly used kernel functions. The first one is simply the familiar dot product of two vectors, while the 2nd one, RBF (radial basis function) or Gaussian kernel is the most popular (and usually the default choice) kernel for nonlinear problems.

Function	Equation	Hyperparameter
Linear	$K(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x}^T \boldsymbol{z}$	
Gaussian	$K(\boldsymbol{x}, \boldsymbol{z}) = exp(-\frcd{\|\boldsymbol{x} - \boldsymbol{z}\|^2}{2\sigma^2})$	σ
Polynomial	$K(\boldsymbol{x}, \boldsymbol{z}) = (\gamma \boldsymbol{x}^T \boldsymbol{z} + r)^d$	γ, r, d
Sigmoid	$K(\boldsymbol{x}, \boldsymbol{z}) = tanh(\gamma \boldsymbol{x}^T \boldsymbol{z} + r)$	γ, r

`

Let's use the polynomial kernel to illustrate how using kernel functions amounts to higher dimensional mapping. Assume that we use the following kernel

$$K(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x}^T \boldsymbol{z} + 1)^2$$

where $\boldsymbol{x} = [x_1, x_2]^T$ and $\boldsymbol{z} = [z_1, z_2]^T$ are two vectors in the original 2D space. We claim that the above kernel is equivalent to the following mapping

$$\varphi(\boldsymbol{x}) = [x_1, x_2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, 1]$$

To see how, just compute $\varphi(\boldsymbol{x})^T \varphi(\boldsymbol{z})$

$$\begin{aligned} \varphi(\boldsymbol{x})^T \varphi(\boldsymbol{z}) &= x_1^2 z_1^2 + x_2^2 z_2^2 + 2x_1 z_1 + 2x_2 z_2 + 2x_1 x_2 z_1 z_2 + 1 \\ &= (x_1 z_1 + x_2 z_2 + 1)^2 \\ &= (\boldsymbol{x}^T \boldsymbol{z} + 1)^2 \\ &= K(\boldsymbol{x}, \boldsymbol{z}) \end{aligned}$$

Therefore, if you use the above polynomial kernel, you are implicitly assuming that your dataset is linearly separable in the shown 6th dimensional feature space! In general, you will have to experiment with the kernel hyperparameters to determine what exact form of polynomial kernel works best for your problem.

If you were amazed by the previous illustration, you will find it more interesting to know that Gaussian kernels map original space into an infinite dimensional feature space!. Luckily, we don't need to know the form of this feature space.

Sklearn Implementation

Let's try to find the nonlinear classifier boundary for the toy dataset in Figure 7.4 using Gaussian kernel. We will also find optimal values of C and σ via grid-search and cross-validation.

```
# generate data
from sklearn.datasets import make_circles

X, y = make_circles(500, factor=.08, noise=.1, random_state=1)
# note that y = 0,1 here and need not be ±1; SVM does internal transformation accordingly
```

`

```
# find optimal hyperparameter via GridSearchCV
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

param_grid = {'C':[0.1, 1, 10, 100, 1000], 'gamma':[0.01, 0.1, 1, 10, 100]}
gs = GridSearchCV(SVC(), param_grid, cv=5).fit(X, y) # no scaling as inputs are already scaled

print('Optimal hyperparameter:', gs.best_params_)

>>> Optimal hyperparameter: {'C': 0.1, 'gamma': 1}
```

You will notice that Sklearn uses the hyperparameter gamma which is simply $1/2\sigma^2$. Optimal C and gamma come out to be 0.1 and 1, respectively, with the classifier solution shown in Figure 7.6. The figure also shows the boundary regions for low and high values of the hyperparameters. As we saw before, large C leads to overfitting (boundary impacted by the noise). As far as gamma is concerned, large value (or small σ) also leads to overfitting.

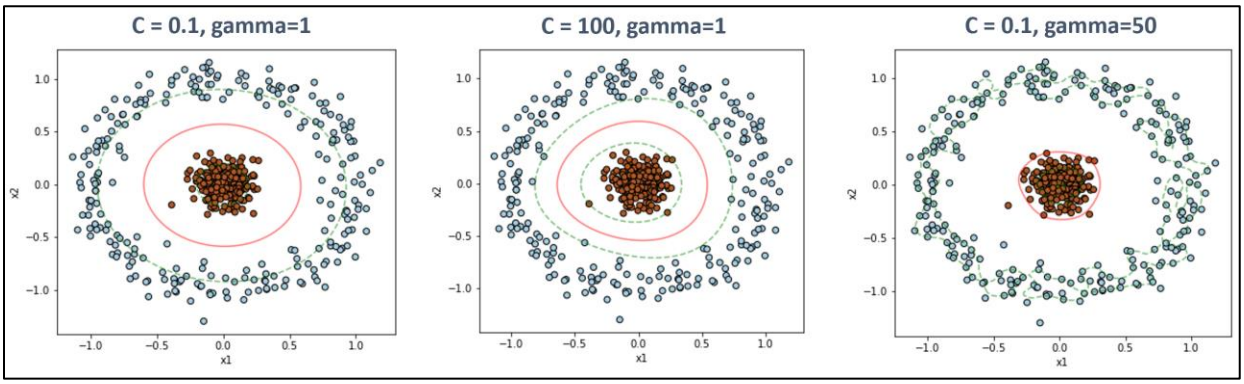


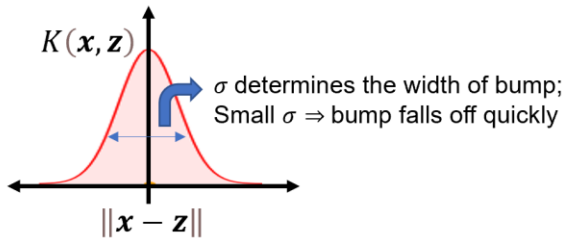
Figure 7.6: Nonlinear binary classification via kernel SVM and impact of hyperparameters. [Code for plotting the boundaries is provided online]

`

Closer look on Gaussian kernels

A better intuition behind the kernels can help us understand the impact of kernel hyperparameters on the classification boundaries. Kernels provide an indirect measure of similarity between any two points in the high-dimensional space. Consider again the Gaussian kernel

$$K(\boldsymbol{x}, \boldsymbol{z}) = exp(-\frac{\|\boldsymbol{x} - \boldsymbol{z}\|^2}{2\sigma^2})$$

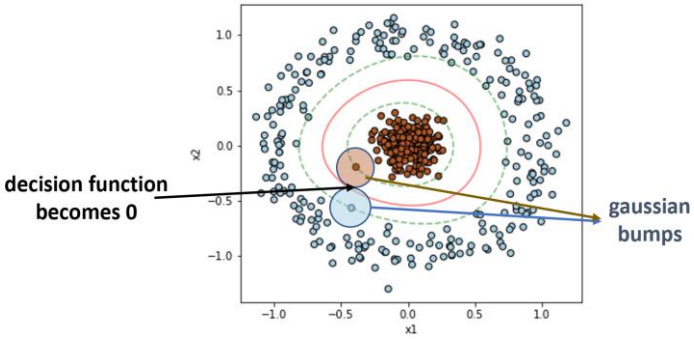


Here, if two points ($\boldsymbol{x}$, $\boldsymbol{z}$) are close to each-other in the original space, then their similarity (or kernel value) in the mapped space will be higher, compared to when $\boldsymbol{x}$ and $\boldsymbol{z}$ are far away from each-other. Now, let's look at the classifier prediction formula for a sample $\boldsymbol{z}$

$$\hat{y} = sign(\sum_{i \in \{SV\}} \alpha_i y_i K(\boldsymbol{x_i}, \boldsymbol{z}) + b)$$

Sum of gaussian bumps

Therefore, the classifier is nothing but a sum of Gaussian bumps from the support vectors (plus an offset *b*)!



Given bandwidth (σ), during training, SVM tries to find the optimal values of the bump multipliers (α_s) such that the training samples get correct labels while keeping maximum separation between classification boundary and training samples. The boundary is simply the set of points where the net summations of bumps and offset become zero. Small values of $\boldsymbol{\sigma}$ lead to very localized bumps near any support vector, resulting in higher number of support vectors with too much 'wiggles' in the separating boundary which often indicates overfitting.

`

7.3 SVDD: An Introduction

Support vector data description (SVDD) is the unsupervised form of SVM algorithm used for dealing with problems where training samples belong to only one class and the model objective is to determine if any test/new sample belongs to the same class of data or not. Consider the motivating example in Figure 7.7. Here, how do we obtain a model of the training data to be able to call sample *X* an outlier / abnormal? Such problems are quite common in process industry. Process monitoring and equipment health monitoring are some example areas where most / all the available process data may belong to only normal plant operations class and the modeling objective is to identify any abnormal data-point.

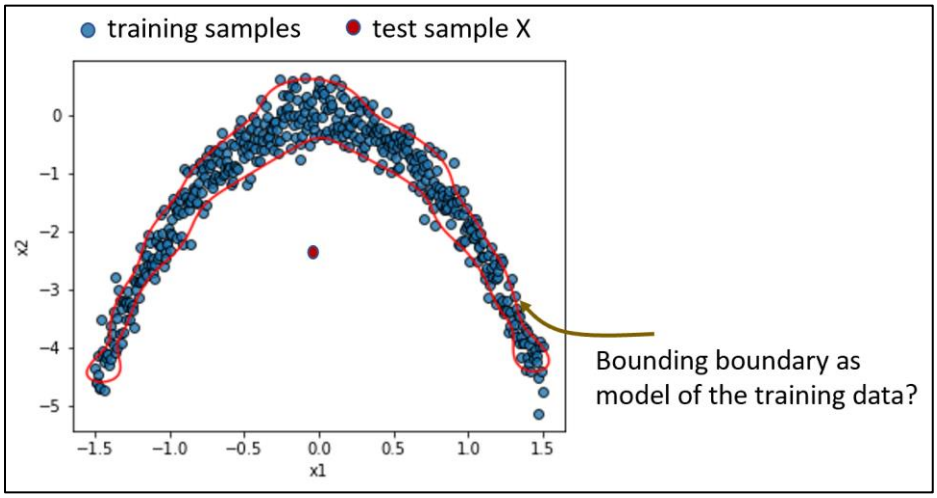


Figure 7.7: 2D training dataset with only one class. Boundary in red shows a potential description model of the dataset that can be used to distinguish the test sample *X* from training samples.

The idea behind SVDD is to envelop training data by a hypersphere (circle in 2D, sphere in 3D) containing maximum number of data-points within a smallest volume. Any new observation that lies farther than the hypersphere radius from hypersphere center can be regarded as abnormal observation. But the data in Figure 7.7 don't look like it can be suitably enveloped with a circle? That is correct and our recourse is to use kernel functions to implicitly project original data onto a higher dimensional space where data can be adequately enveloped within a compact hypersphere. The projection of the optimal hyperplane onto the original space will show up as a tight nonlinear boundary around the dataset!

Just like classical 2-class SVM, only a small set of training samples get to completely define the hypersphere. These data-points or support vectors lie on the circumference or outside of the hypersphere (or the nonlinear boundary in the original space).

`

Mathematical background

Assume again that $\varphi(\boldsymbol{x})$ represents a data-point in the higher dimensional feature space. In this space, the optimal hypersphere is found via the following optimization problem

$$\begin{aligned} \min_{R,a,\xi} \quad & R^2 + C\sum_{i=1}^{N}\xi_i \\ \text{s.t.} \quad & \left||\varphi(\boldsymbol{x_i}) - a|\right|^2 \leq R^2 + \xi_i, \quad i = 1, \cdots N \\ & \xi_i \geq 0 \end{aligned}$$

As is evident, the above program is trying to minimize the radius (*R*) of the hypersphere centered at '***a***' such that most of the data-points lie within the hypersphere. Slack variables, ξ, allow certain samples to fall outside and the number of such violations is tuned via the hyperparameter *C*. As before, the problem is solved in its dual form

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j K\left(\boldsymbol{x_i}, \boldsymbol{x_j}\right) - \sum_{i=1}^{N}\alpha_i K(\boldsymbol{x_i}, \boldsymbol{x_i}) \\ \text{s.t.} \quad & \sum_{i=1}^{N}\alpha_i = 1 \\ & 0 \leq \alpha_i \leq C \quad i = 1, \cdots N \end{aligned}$$

Like SVM, the alphas indicate the position of training samples w.r.t. the optimal boundary. The following relationships hold true

Inside the boundary:	$\left\|\|\varphi(\boldsymbol{x_i}) - a\|\right\| < R \Rightarrow \alpha_i = 0$	
On the boundary:	$\left\|\|\varphi(\boldsymbol{x_i}) - a\|\right\| = R \Rightarrow 0 < \alpha_i < C$	Support Vectors
Outside the boundary:	$\left\|\|\varphi(\boldsymbol{x_t}) - a\|\right\| > R \Rightarrow \alpha_i = C$	Support Vectors

The optimal solution satisfies the following expression

$$R = \sqrt{K(\boldsymbol{x_s}, \boldsymbol{x_s}) - 2\sum_{i\in\{SV\}}\alpha_i K(\boldsymbol{x_s}, \boldsymbol{x_i}) + \sum_{i\in\{SV\}}\sum_{j\in\{SV\}}\alpha_i\alpha_j K\left(\boldsymbol{x_i}, \boldsymbol{x_j}\right)}$$

where $\boldsymbol{x_s}$ is any support vector lying on the boundary. Any test observation $\boldsymbol{x_t}$ is abnormal if its distance from center ***a*** in the mapped space is greater than *R* where the distance is given as follows

`

$$Dist(\varphi(\boldsymbol{x_t}), \boldsymbol{a})^2 = \left|\left|\varphi(\boldsymbol{x_t}) - \boldsymbol{a}\right|\right|^2$$
$$= K(\boldsymbol{x_t}, \boldsymbol{x_t}) - 2\sum_{i\in\{SV\}} \alpha_i K(\boldsymbol{x_t}, \boldsymbol{x_i}) + \sum_{i\in\{SV\}}\sum_{j\in\{SV\}} \alpha_i\alpha_j K(\boldsymbol{x_i}, \boldsymbol{x_j})$$

As you can see, specifications of kernel functions and other model hyperparameters is all that is needed; no knowledge of mapping φ is required.

OC-SVM vs SVDD

There is another technique closely related to SVDD, called one class SVM (OC-SVM). Infact, OC-SVM is the unsupervised SVM algorithm currently available in Sklearn. OC-SVM finds a separating hyperplane that best separates the training data from the origin. Its kernelized dual form is given by

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \tfrac{1}{2}\textstyle\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j K\left(\boldsymbol{x_i}, \boldsymbol{x_j}\right) \\ \text{s.t.} \quad & \textstyle\sum_{i=1}^{N} \alpha_i = 1 \\ & 0 \le \alpha_i \le C \quad i = 1, \cdots N \end{aligned}$$

You will notice that for Gaussian kernel, OC-SVM formulation becomes equivalent to that of SVDD because $K(\boldsymbol{x_i}, \boldsymbol{x_i}) = 1$ and we end up with the same values of the multipliers. The decision boundaries are the same as well. For other kernels with $K(\boldsymbol{x_i}, \boldsymbol{x_i}) \neq 1$, results would be different.

Bandwidth parameter & SVDD illustration

In this sub-section, we will attempt to use Sklearn's OneClassSVM to build the bounding boundary for the dataset in Figure 7.7. However, before we do that, we need to specify our strategy for determination of SVDD model hyperparameters, C and σ. If sufficient validation data samples belonging to both 'normal' and 'abnormal' classes are available, then cross-validation can be used to choose C and σ that give the best validation accuracy. However, very often this is not possible and like training dataset, validation dataset also only contains 'normal' class samples! To enable educated hyperparameter selection, let's take another look at them.

Previously, we saw that *C* controls the trade-off between volume of hypersphere and the number of misclassifications in the training dataset. *C* can also be written as

`

$$C = \frac{1}{Nf}$$

Where *N* is the number of samples and *f* is the expected fraction of outliers in the training dataset. Smaller value of *f* (correspondingly larger *C)* will lead to less samples being put outside the hypersphere. Infact if *C* is set to 1 (or greater) the hypersphere will include all the samples (as $\sum \alpha_i = 1$ and $\alpha = C$ outside the hypersphere). Therefore, *C* can be set with some educated presumptions on the outlier fractions. In absence of any advance knowledge, *f* = 0.01 is often specified to exclude away 1% of sample lying farthest from hypersphere center.

As far as σ is concerned, we previously saw that at low value of σ, data boundary becomes very wiggly with high number of support vectors, resulting in overfitting. Conversely, at high value of σ, boundary tends to become spherical in the original space itself resulting in underfitting (or non-compact bounding of data). One approach for bandwidth selection is to use empirical methods which are based on obtaining a kernel matrix (whose *i,j*th element is $K(\boldsymbol{x_i}, \boldsymbol{x_j})$) with favorable properties. One such method, modified mean criterion[36], gives bandwidth as follows

$$\sigma = \sqrt{\frac{\bar{D}^2}{\ln\left(\frac{N-1}{\delta^2}\right)}}$$

$$\bar{D}^2 = \frac{\sum_{i<j} \left|\left|\boldsymbol{x_i} - \boldsymbol{x_j}\right|\right|^2}{\frac{N(N-1)}{2}}$$

$$\delta = -0.14818008\emptyset^4 + 0.2846623624\emptyset^3 - 0.252853808\emptyset^2 + 0.159059498\emptyset - 0.001381145$$

$$\emptyset = \frac{1}{\ln(N-1)}$$

Another approach for bandwidth selection is to choose largest value of σ that gives the desired confidence level on the validation dataset. For example, for a confidence level of 99%, σ is increased until 99% of validation samples are correctly classified as inliers. Any higher value of σ will include more validation samples within the hypersphere. The modified mean criterion can be used as the initial guess with subsequent search made around it. Let's now find the nonlinear boundary for the dataset in Figure 7.7.

```
# read data
import numpy as np
X = np.loadtxt('SVDD_toyDataset.csv', delimiter=',')
```

[36] Kalde & Sadek, The mean and median criterion for kernel bandwidth selection for support vector data description, IEEE 2017

`

```
# compute bandwidth via modified mean criteria
import scipy

N = X.shape[0]
phi = 1/np.log(N-1)
delta = -0.14818008*np.power(phi,4) + 0.2846623624*np.power(phi,3) - 0.252853808*np.power(phi,2)
        + 0.159059498*phi - 0.001381145
D2 = np.sum(scipy.spatial.distance.pdist(X, 'sqeuclidean'))/(N*(N-1)/2)
sigma = np.sqrt(D2/np.log((N-1)/delta*delta))
gamma = 1/(2*sigma*sigma)

# SVM fit
from sklearn.svm import OneClassSVM
model = OneClassSVM(nu=0.01, gamma=gamma).fit(X) # nu corresponds to f
```

Figure 7.8 shows the bounding boundary for different values of gamma with *f* (or nu in Sklearn) kept at 0.01. A value of gamma (= 1) close to that given by modified mean criterion method (= 0.58) provided a satisfactory boundary.

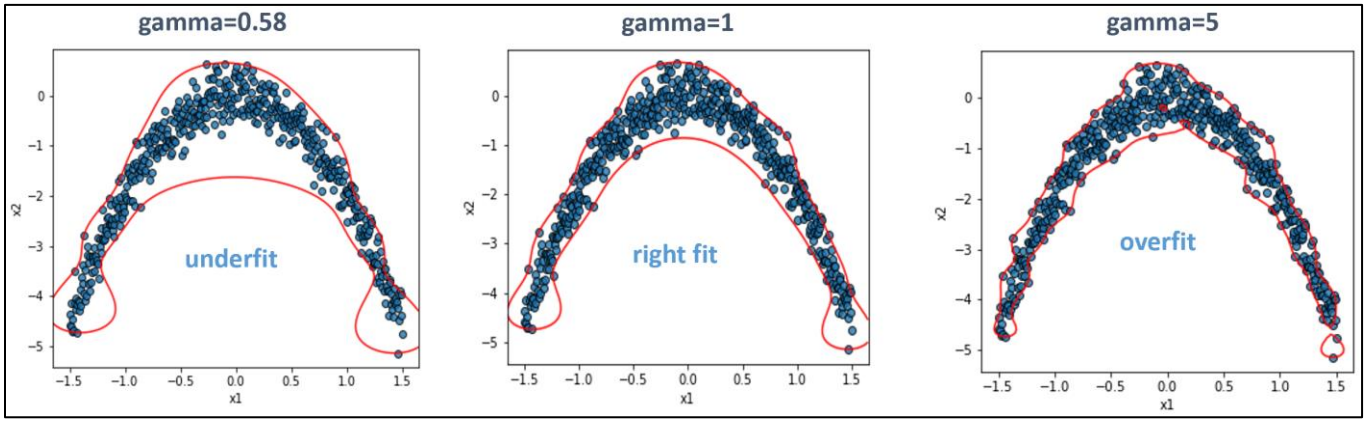


Figure 7.8: SVDD application for data description and impact of model hyperparameter.

We hope that by now you are convinced of the powerful capabilities of SVM for discriminating between different classes of data and compact bounding of normal operational data. A big requirement for successful application of SVM is that the training dataset should be very representative of the 'normal' dataset and fully characterize all the expected variations. Next, we will look at a case study with real process data and then move onto another variant of SVM utilized for regression applications.

`

PCA vs SVDD for process monitoring

In the previous chapters, we studied dimensionality reduction-based methods (PCA, ICA) for process monitoring. We have now added SVDD to our toolkit. Although it is difficult to say beforehand which method will work best for given scenarios, some rough guidelines do exist. If variables in your dataset are linearly (or only slightly nonlinearly) related and measurements are noisy, PCA/ICA would work best. PCA, however, assumes unimodal distribution of data. SVDD makes no assumption on underlying data distribution and is very-well suited for very nonlinear dataset. However, relationships between different process variables are difficult to interpret in SVM/SVDD models compared to PCA/ICA models.

7.4 Process Fault Detection via SVDD

To illustrate a practical application of SVDD for process monitoring, we will use data from a semiconductor manufacturing process. This batch process dataset contains 19 process variables measured over the course of 108 normal batches and 21 faulty batches. The batch durations range from 95 to 112 seconds (see appendix for more details). Figure 7.9 shows the training samples and the faulty test samples in the principal component space.

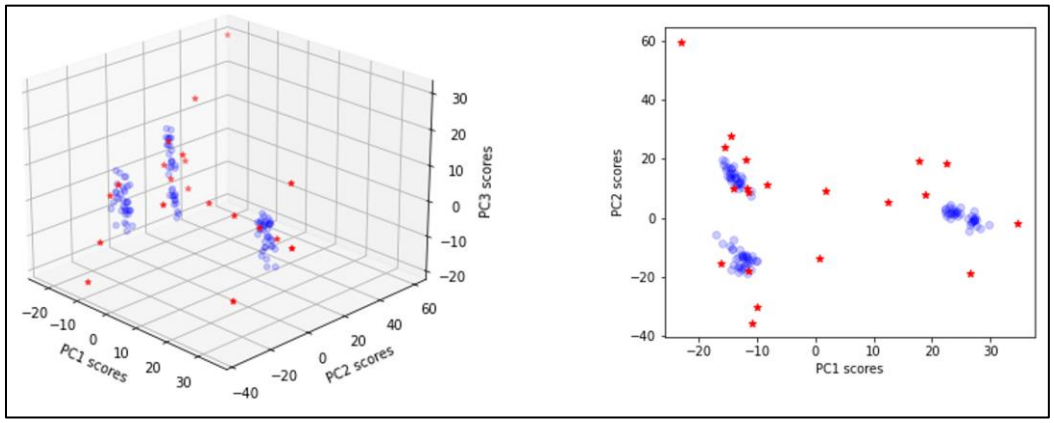


Figure 7.9: Normal (in blue) and faulty (in red) batches in PCA score space

For this illustration, the raw data has been processed using multiway PCA and the transformed 2D (score) data is provided in the *Metal_etch_2DPCA_trainingData.csv* file. Note that we could also implement SVDD in the original input space but pre-processing via PCA to remove variable correlation is generally a good practice. Moreover, we use the 2D PC space for our analysis just for the ease of illustrating the SVDD boundary. In actual deployment, you would work in higher dimensional PC space for better accuracy. Let's see if our model can identify the faulty samples as outliers or not in the multi-clustered dataset.

`

```
# read data
import numpy as np
X_train = np.loadtxt('Metal_etch_2DPCA_trainingData.csv', delimiter=',')

# fit SVM
from sklearn.svm import OneClassSVM
model = OneClassSVM(nu=0.01, gamma=0.025).fit(X_train) # gamma from modified mean criterion = 0.0025

# predict for test data
X_test = np.loadtxt('Metal_etch_2DPCA_testData.csv', delimiter=',')
y_test = model.predict(X_test)  # y=-1 for outliers

print('Number of faults identified: ', np.sum(y_test == -1), ' out of ', len(y_test))

>>> Number of faults identified:  17  out of  20
```

Figure 7.10 shows the boundary around the training samples and the faulty samples labeled according to their correct or incorrect identification. Seventeen out of twenty faulty data samples have correctly been identified as outliers. This example illustrates the power of SVDD for compactly describing clustered datasets.

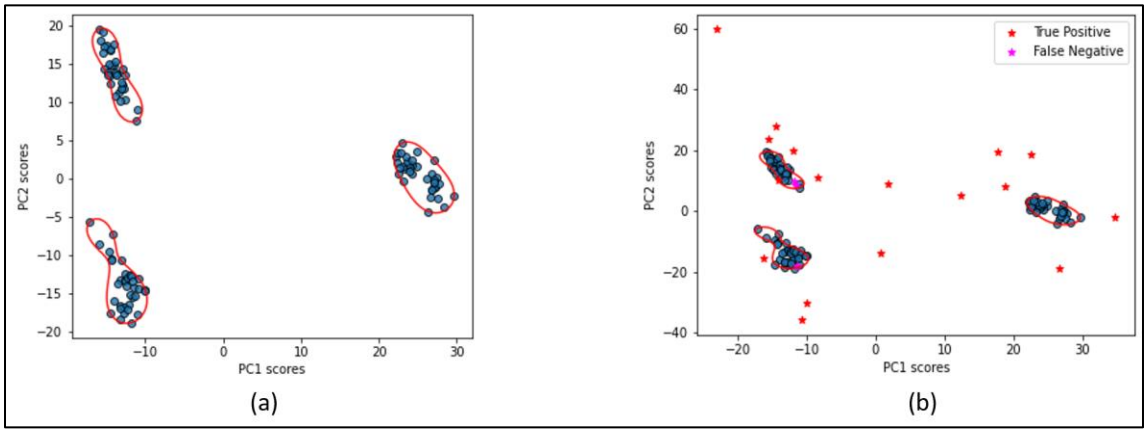


Figure 7.10: (a) SVDD / OC-SVM boundary (in red) around metal-etch training dataset in 2D PC space (b) Position of correctly and incorrectly diagnosed faulty samples

We should get the same results if we use the distances from the hypersphere center for fault detection. The results from SVDD and OC-SVM will differ if RBFs are not used as kernel. Unfortunately, Sklearn currently does not provide SVDD implementation. Nonetheless, a SVDD package is available on GitHub[37].

[37] https://github.com/iqiukp/SVDD

`

7.5 SVR: An Introduction

Support vector regression (SVR) is another variant of SVM used for linear and nonlinear regressions. SVR attempts to find a regression curve that is as close as possible to the training observations. Geometrically, as shown in Figure 7.11 for kernelized-SVR, a tube of pre-specified width (ε) is fitted to the data and any sample lying outside the tube is penalized. You will see later that SVR's optimization program is designed to obtain a good balance between model generalization capabilities and the ability to fit the training data.

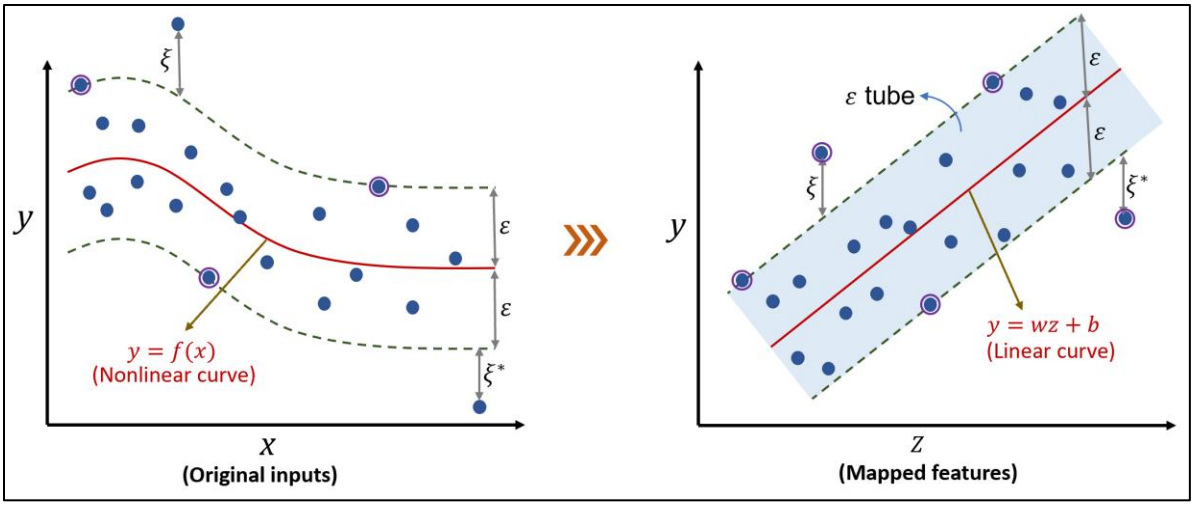


Figure 7.11: Kernelized-SVR illustration for nonlinear curve fitting

SVR generates a linear hyperplane that best describes the output as a function of the inputs. For kernelized SVR, the hyperplane translates into an appropriate nonlinear curve in the original measurement space. As shown in Figure 7.11, in feature space, most of the data-points lie within ε distance to the optimal hyperplane. Slack variables, ξ, allow some samples to fall outside the ε tube; however, such violations are penalized. Samples lying on the edge or outside the tube are the support vectors which completely determine the optimal solution for a given ε. Adding or removing any non-support vector (or samples inside the tube), does not affect the solution.

SVR has been found to provide performance superior to ANNs for small and medium-sized high-dimensional datasets. We will see a couple of applications later in the chapter.

Mathematical Background

SVR's optimization program for parameter estimation takes the following form in the feature space

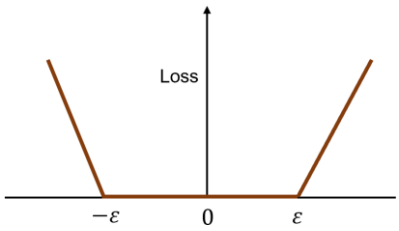
`

$$\underset{\boldsymbol{w},b,\xi,\xi^*}{\text{Min}} \quad \frac{1}{2}\left|\left|\boldsymbol{w}\right|\right|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*)$$

$$\text{s.t.} \quad y_i - \boldsymbol{w}^T\varphi(\boldsymbol{x_i}) - b \leq \varepsilon + \xi_i$$
$$\boldsymbol{w}^T\varphi(\boldsymbol{x_i}) + b - y_i \leq \varepsilon + \xi_i^*$$
$$\xi_i, \xi_i^* \geq 0, \quad i = 1, \cdots N$$

The first part of the objective ($\frac{1}{2}\left|\left|\boldsymbol{w}\right|\right|^2$) attempts to keep the model as less complex as possible or the model output as flat as possible. For example, for $y=\boldsymbol{w}^T\boldsymbol{x}+b$, w=0 gives the simplest model y=b with least dependency on inputs. The 2nd part tries to keep the number of samples outside of the ε tube as low as possible. The hyperparameter *C* controls the trade-off between the two.

In the SVR formulations, there are no errors associated with samples that lie within the $\boldsymbol{\varepsilon}$ tube. In SVM literature, this is referred to as implementing $\boldsymbol{\varepsilon}$-insensitive loss function as shown below.



Compare this to an ordinary least squares regression where error from the regression curve for every sample is penalized.

The dual form is given by

$$\underset{\boldsymbol{\alpha},\boldsymbol{\alpha}^*}{\max} \quad \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)K\left(\boldsymbol{x_i}, \boldsymbol{x_j}\right) -$$
$$\varepsilon\sum_{i=1}^{N}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{N} y_i(\alpha_i - \alpha_i^*)$$

$$\text{s.t.} \quad \sum_{i=1}^{N}(\alpha_i - \alpha_i^*) = 0$$
$$0 \leq \alpha_i, \alpha_i^* \leq C \quad i = 1, \cdots N$$

If you notice carefully, here we assign two multipliers ($\boldsymbol{\alpha}, \boldsymbol{\alpha}^*$) for each data-point. Both the multipliers end up being 0 for non-support vectors (training samples lying within the ε tube). Once multipliers have been found, predictions can be made by

`

$$\hat{y}_t = \sum_{i \in \{SV\}} (\alpha_i - \alpha_i^*) K(\boldsymbol{x_i}, \boldsymbol{x_t}) + b$$

The specification of the kernel function and hyperparameters allow us to fit the model (find multipliers) and make predictions. Figure 7.12 shows the impact of the SVR hyperparameters (C, γ, ε) with RBF kernel for a SISO system.

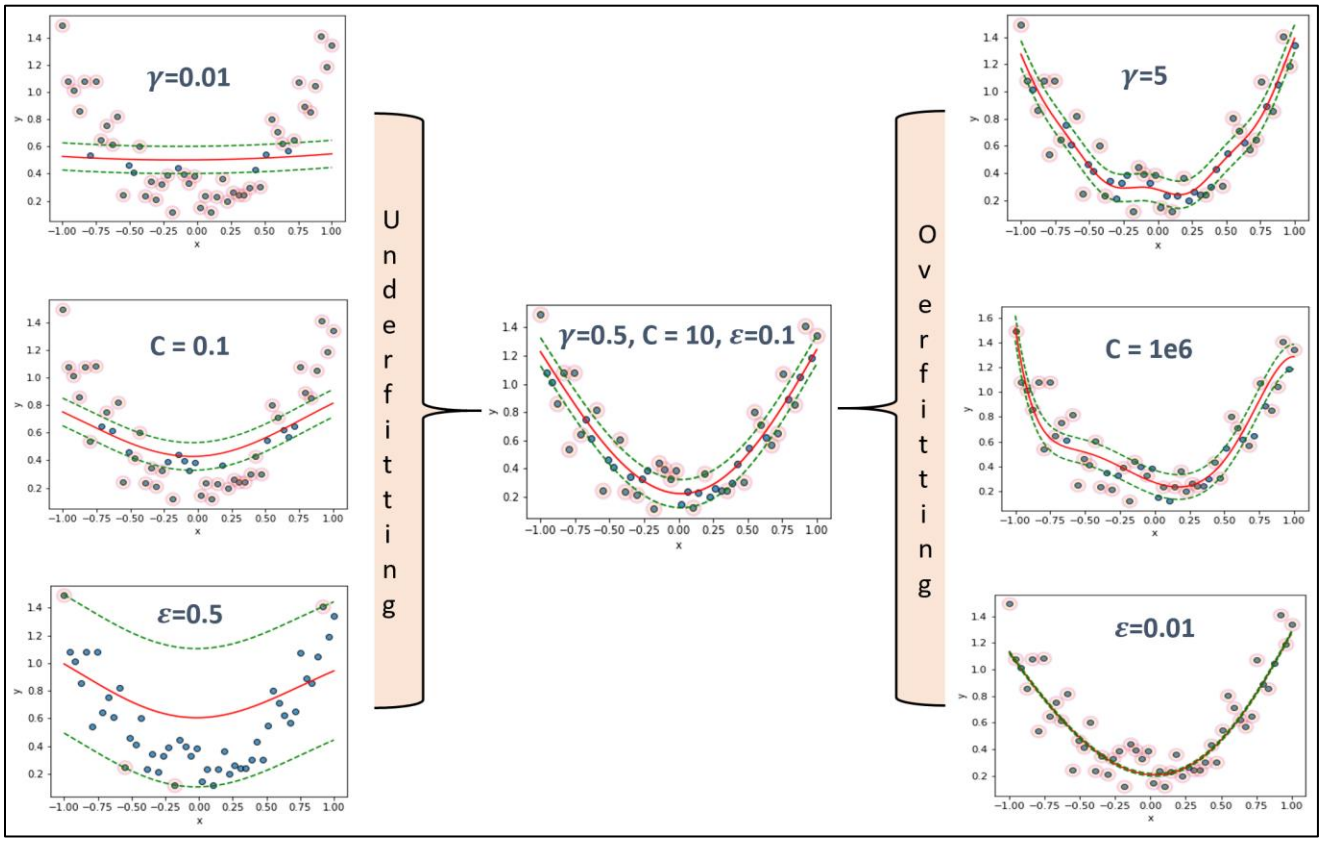


Figure 7.12: Impact of SVR hyperparameters on model fitting quality. Encircled training samples denote the support vectors selected by the respective models.

Figure 7.12 is along the expected lines w.r.t. gamma and C. Let's understand the impact of ε. Large ε makes the ε tube big allowing SVR to keep the model 'flat' leading to underfitting. On the other hand, very small ε makes the tube very small which results in high number of support vectors making the fit prone to overfitting. Nonetheless, it is not uncommon to specify small ε and control overfitting via C. To see SVR implementations using Sklearn, let's check out a couple of industrial applications in the next sections.

`

SVR vs ANN

Due to the powerful nonlinear modeling capabilities of SVR, it is natural to compare it to artificial neural networks (ANNs). The equation for SVR model prediction can be represented in a network form as shown in figure below. This representation makes the similarities between the two regression methods evident. However, unlike ANNs, the hidden units are automatically determined without any explicit specification of the network structure in SVR. This makes SVR easier to train as there are fewer hyperparameters.

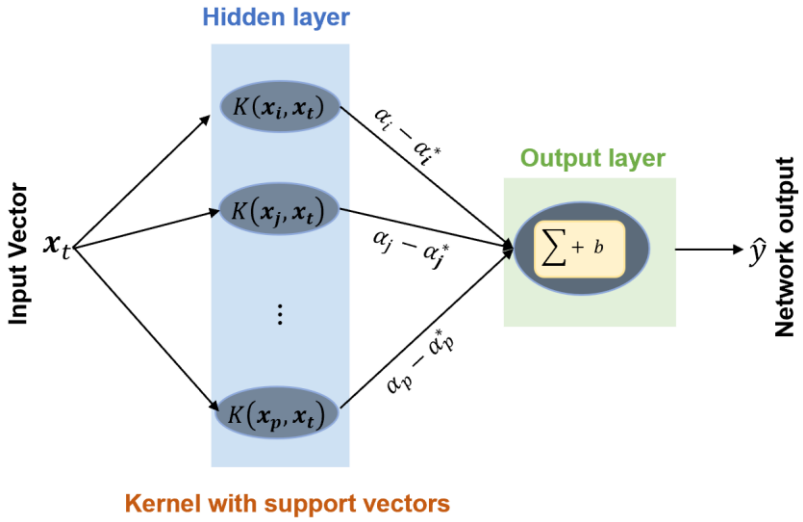


Fig: Network representation of SVR prediction

[We will learn about ANNs in detail in Chapter 11. If you are not familiar with them, you can re-visit this note later. But we figured that if you already know, this comparison could throw some more insights into SVR algorithm.]

In ANNs, the number of fitted model parameters (and therefore the tendency to overfit) increases with increase in dimensionality of input vectors. But in SVR, number of parameters depend on the number of support vectors and not on input dimensionality. Adding more input variables in a SVR model doesn't increase model complexity if the number of support vectors don't change. This greatly helps SVRs overcome overfitting issue arising from high input dimensionality. Moreover, through support vectors, SVR models provide an explicit knowledge of training data-points which are important in defining the regression prediction; this can often help in rationalizing model predictions. Overall, SVRs are as powerful as ANNs for nonlinear modeling with some unique advantageous features.

`

7.6 Soft Sensing via SVR in a Polymer Processing Plant

For our first illustration, we will apply SVR for soft sensing for predicting plant outputs in a polymer plant. The dataset obtained from Dupont consists of 61 samples of 14 variables (10 predictors, 4 plant outputs). We chose this dataset to illustrate the utility of SVRs for small-sized datasets. Figure 7.13 provides a quick glimpse into the distribution of some of the process inputs and a process output. The sparse nature of data distribution is immediately evident.

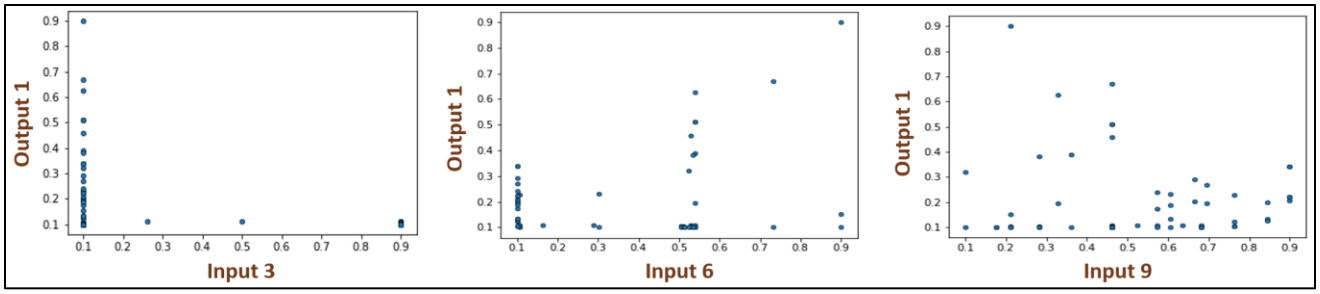


Figure 7.13: Distribution of polymer process inputs and output 1

Sparse data modeling is known to be a difficult task in machine learning. Nonetheless, let's see how well our SVR model can handle this challenge. The code below generates an SVR-based soft sensor model to predict each of the process output one at a time. Hyperparameters are determined via grid-search and k-fold CV.

```
# read data
import numpy as np
data = np.loadtxt('polymer.dat')

X, Y = data[:, 0:10], data[:, 10:]
y = Y[:, 0] # modeling the 1st output

# fit SVR model using grid-search
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV

model = SVR(epsilon=0.01) # default epsilon = 0.1
param_grid = [{'gamma': np.linspace(0.1e-05,5,100), 'C': np.linspace(0.01,5000,100)}]
gs = GridSearchCV(model, param_grid, scoring='neg_mean_squared_error', cv=10)
```

`

```
gs.fit(X, y)
print('Optimal hyperparameter:', gs.best_params_)

>>> Optimal hyperparameter: {'C': 50.51, 'gamma': 0.1}

# predict using the best model
y_predicted_SVR = gs.predict(X)

# plots of raw and predicted data
import matplotlib.pyplot as plt

plt.figure()
plt.plot(y, y_predicted_SVR, '.', markeredgecolor='k', markeredgewidth=0.5, ms=9)
plt.plot(y, y, '-r', linewidth=0.5)
plt.xlabel('measured data'), plt.ylabel('predicted data ')
```

Note that all the variables in the raw dataset are already scaled between 0.1 and 0,9 and therefore no further scaling is done before model fitting. Figure below shows the model predictions. Predictions from PLS models are provided for comparison. The first 2 outputs proved difficult for the linear PLS models, but SVR models could provide good fit. Outputs 3 and 4 seem to be more linearly related to the inputs with both PLS and SVR models providing similar fits.

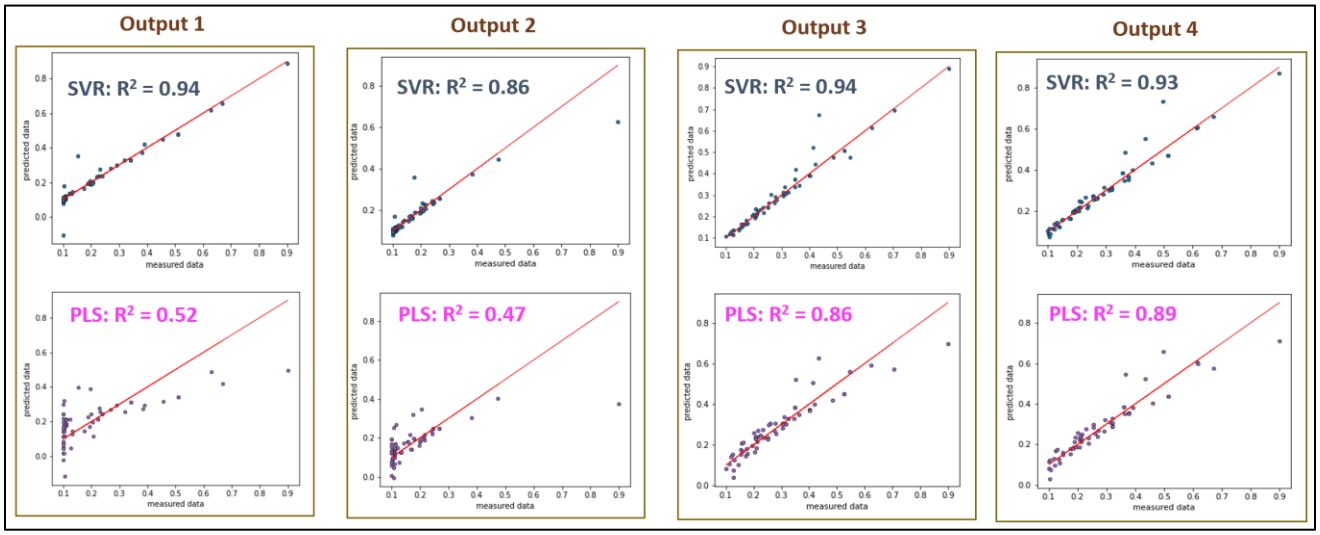


Figure 7.14: SVR and PLS predictions for polymer plant dataset. The red line denotes the ideal $y_{prediction} = y_{measured}$ reference.

`

7.7 Soft Sensing via SVR for Debutanizer Column in a Petroleum Refinery

In this illustration we will use a medium-sized dataset that comes from a debutanizer column operation in a petroleum refinery (see Appendix for system details). The butane (C4) content in gasoline bottoms product of the debutanizer column is not available in real-time and therefore, is required to be predicted using other process data around the column. The dataset contains 2394 samples of input-output process values. Seven process (pressures, temperatures, flows around the column) variables are used as predictors. Figure 7.15 shows that dataset has decent process variability. Note that data have been provided in normalized form.

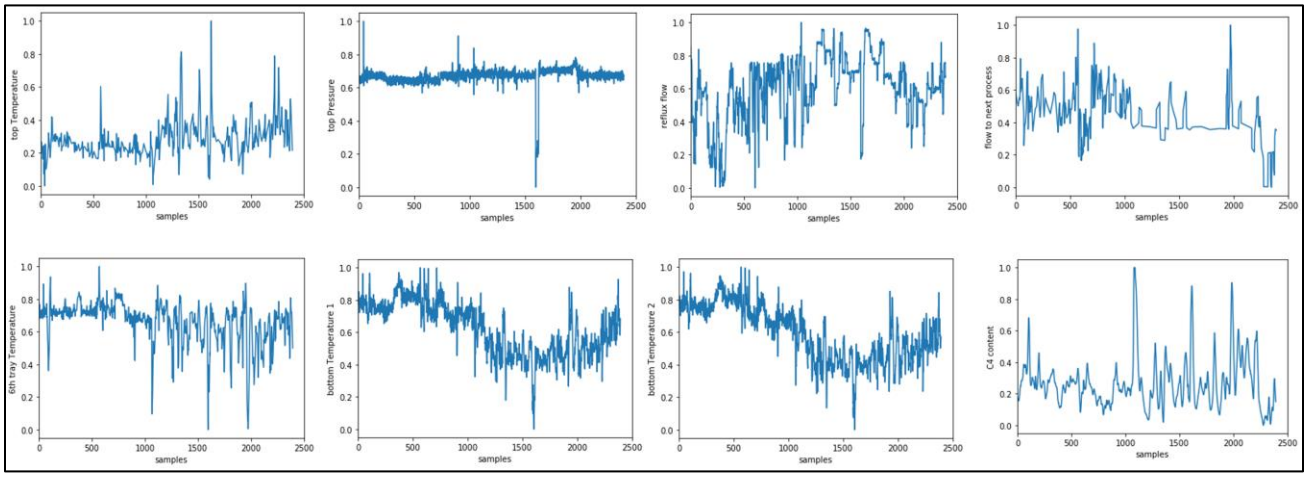


Figure 7.15: Plots of input and output (last plot) variables for the debutanizer column dataset

The output variable shows strong nonlinearity and therefore PLS model fails miserably as shown in Figure 7.16. Let's build an SVR-based soft sensor.

```
# read data
import numpy as np
data = np.loadtxt('debutanizer_data.txt', skiprows=5)

# separate train and test data
from sklearn.model_selection import train_test_split
X, y = data[:,0:-1], data[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_state = 100)

# fit SVR model via grid-search
from sklearn.svm import SVR
```

`

```
from sklearn.model_selection import GridSearchCV

model = SVR(epsilon=0.05)
param_grid = [{'gamma': np.linspace(1,10,10), 'C': np.linspace(0.01,500,10)}]
gs = GridSearchCV(model, param_grid, scoring='neg_mean_squared_error', cv=10)

gs.fit(X_train, y_train)
```

As shown in Figure 7.16, with a very coarse hyperparameter grid, we are able to obtain a SVR model that provides reasonable accuracy on test data. Difference in training and test accuracies suggest that some overfitting, which may be overcome with more exhaustive hyperparameter search.

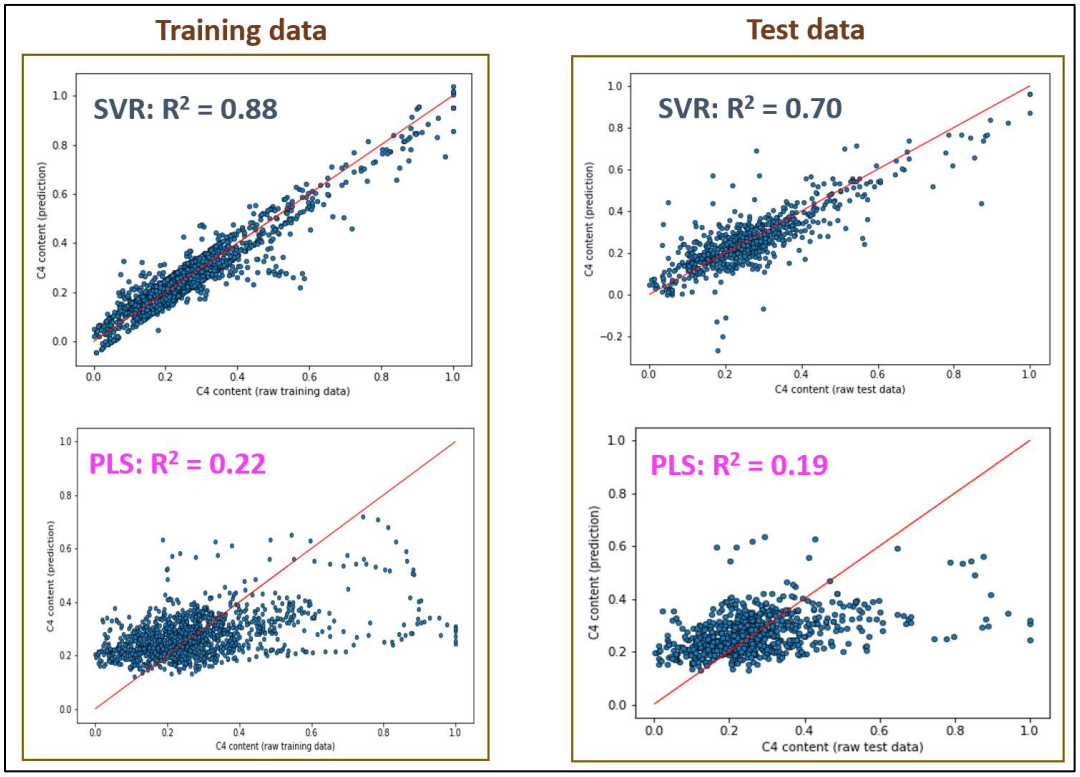


Figure 7.16: SVR and PLS model performance comparison for debutanizer dataset

This concludes our look into support vector machines. SVMs are in a league of their own and are well-suited for industrial processes with difficult to estimate process parameters. With elegant mathematical background, just a few hyperparameters, excellent generalization capabilities, and guaranteed unique global optimum, SVMs are among the best ML algorithms.

`

Summary

In this chapter we studied the support vector machine algorithm and its varied forms for supervised and unsupervised classification and regression. We saw its applications for binary classification, process monitoring, fault detection, and soft sensing. Through kernelized learning, we learned the art of nonlinear modeling. In summary, we have added a powerful tool to your data science toolkit. Next, we will continue building our toolkit and learn how to find clusters/groups in process dataset.