Support Vector Machines

Linear SVM Classification

Outline

- **Linear SVM Classification**
- Nonlinear SVM Classification
- SVM Regression

Support Vector Machines (Classifiers) (SVC)

- *A Support Vector Machine (SVM)* is a powerful and versatile Machine Learning model, capable of performing linear or nonlinear classification, regression, and even outlier detection.

- SVM classifier as fitting the widest possible street (represented by the parallel dashed lines) between the classes. This is called *large margin classification*.

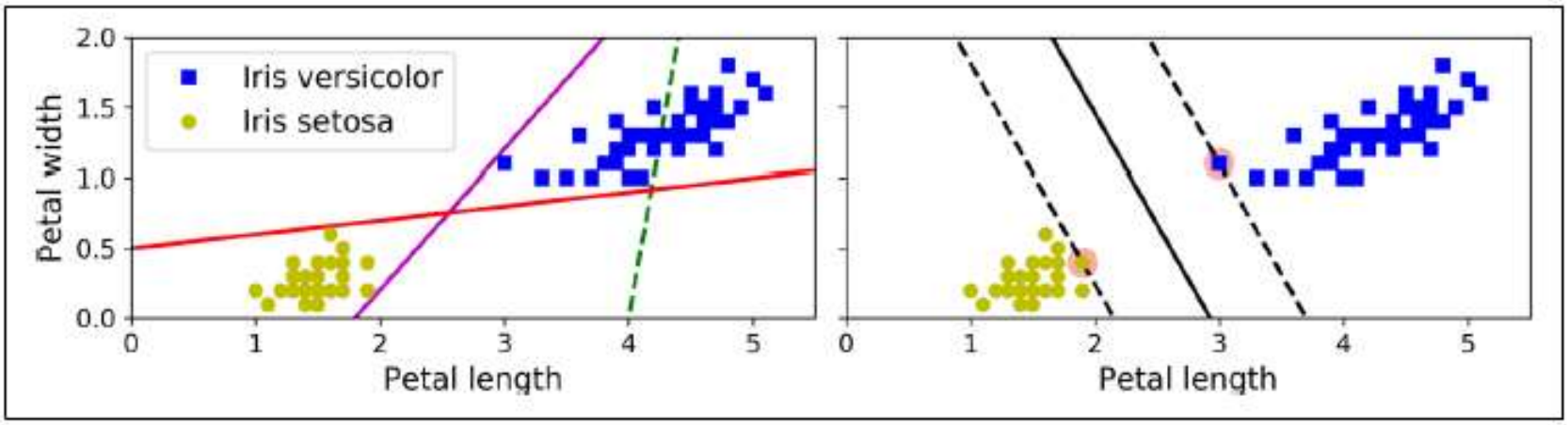


Figure 5-1. Large margin classification

Support Vector Machines (Classifiers) (SVC)

- *Unscaled vs. Scaled*

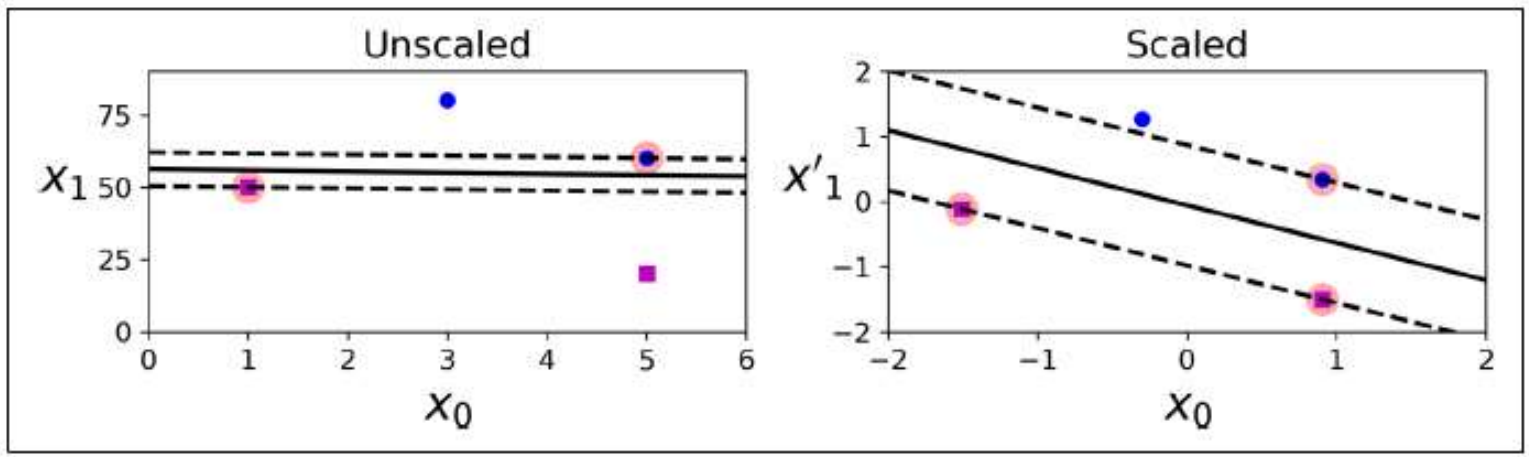


Figure 5-2. Sensitivity to feature scales

Soft Margin Classification

- The objective is to find a good balance between keeping the street as large as possible and limiting the *margin violations* (i.e., instances that end up in the middle of the street or even on the wrong side). This is called *soft margin classification*.

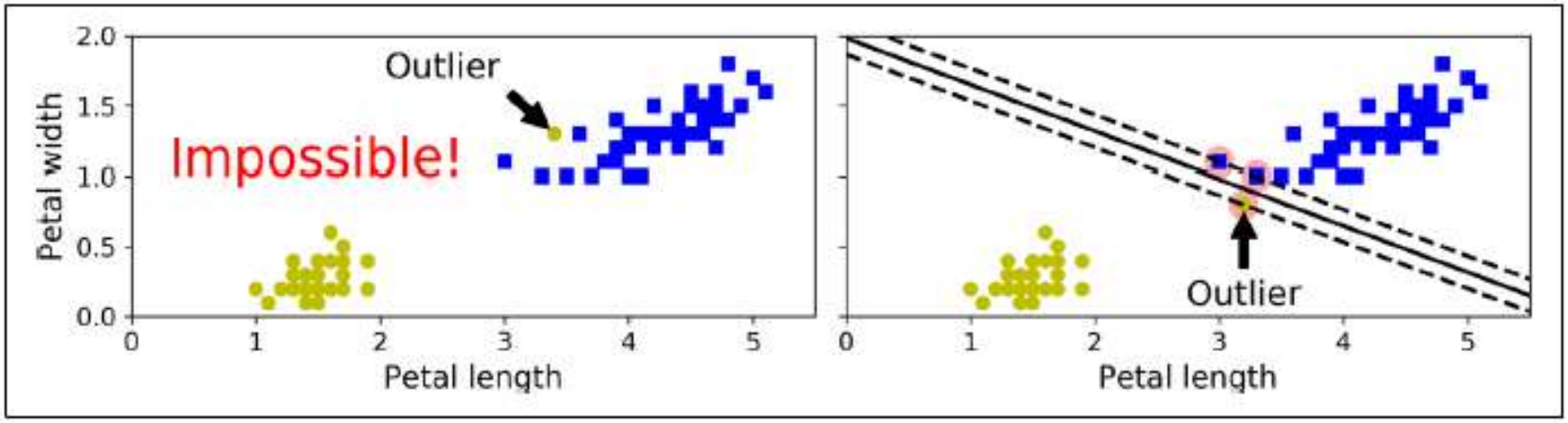


Figure 5-3. Hard margin sensitivity to outliers

Soft Margin Classification: Hyperparameter - C

- The objective is to find a good balance between keeping the street as large as possible and limiting the *margin violations* (i.e., instances that end up in the middle of the street or even on the wrong side). This is called *soft margin classification*.

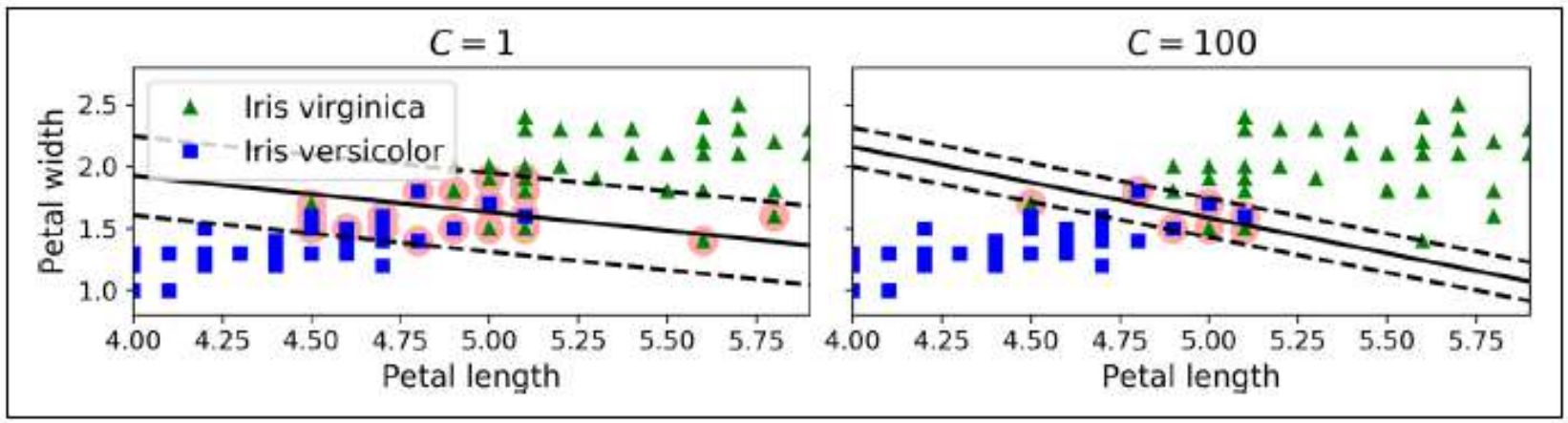


Figure 5-4. Large margin (left) versus fewer margin violations (right)

Decision Functions and Predictions

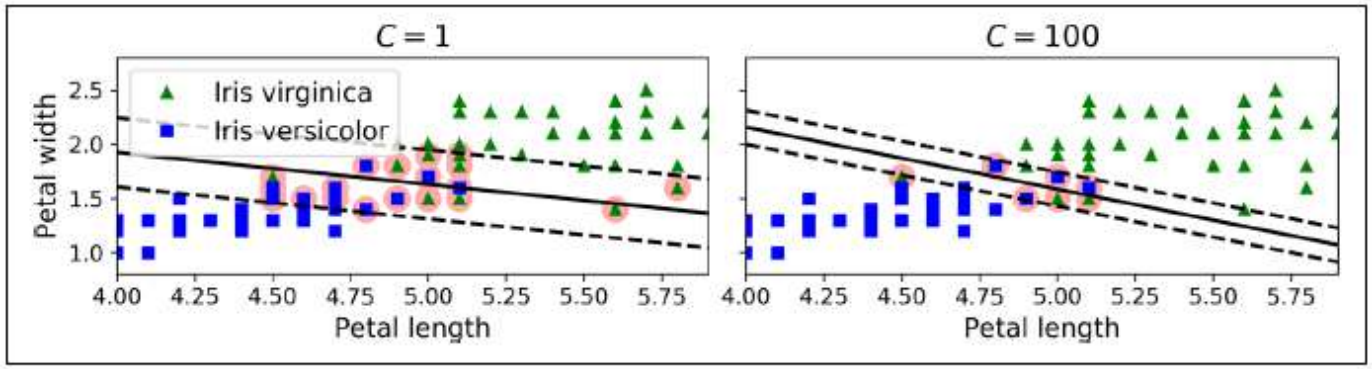


Figure 5-4. Large margin (left) versus fewer margin violations (right)

Equation 5-2. Linear SVM classifier prediction

$$\hat{y} = \begin{cases} 0 \text{ if } \mathbf{w}^\intercal \mathbf{x} + b < 0, \\ 1 \text{ if } \mathbf{w}^\intercal \mathbf{x} + b \geq 0 \end{cases}$$

Decision Functions and Predictions

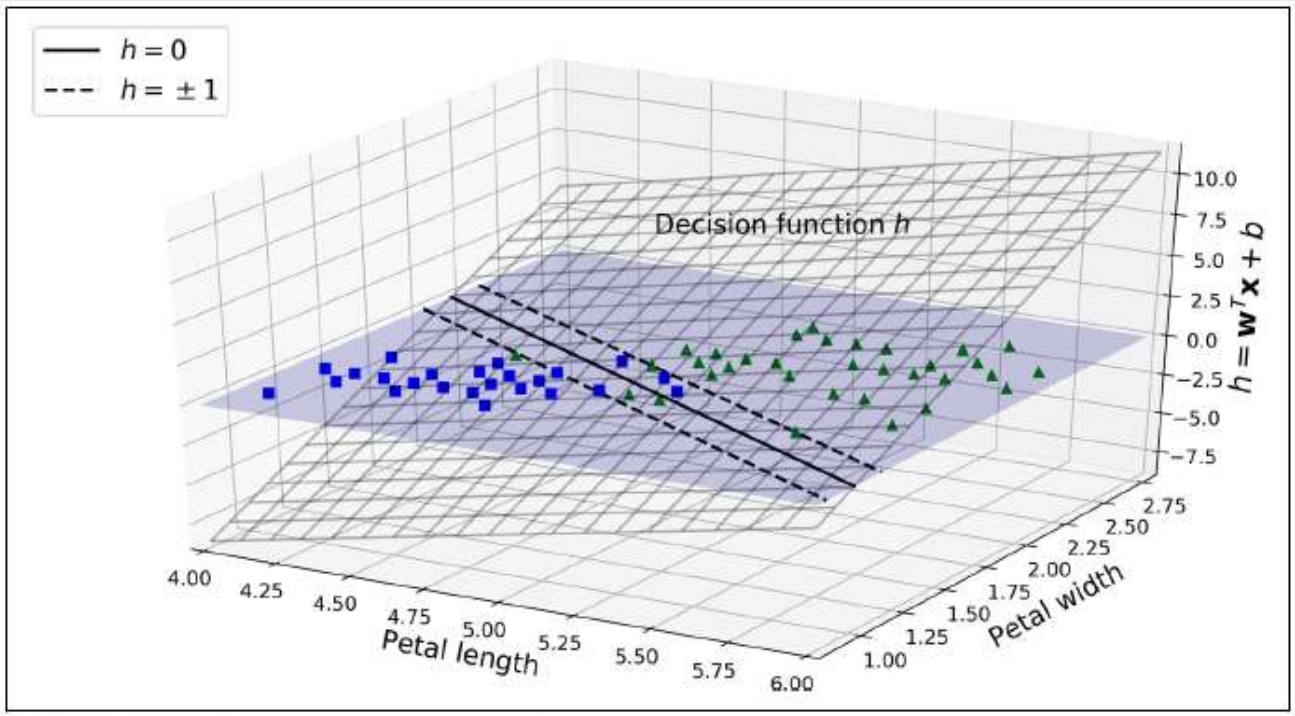


Figure 5-12. Decision function for the iris dataset

Training Objective

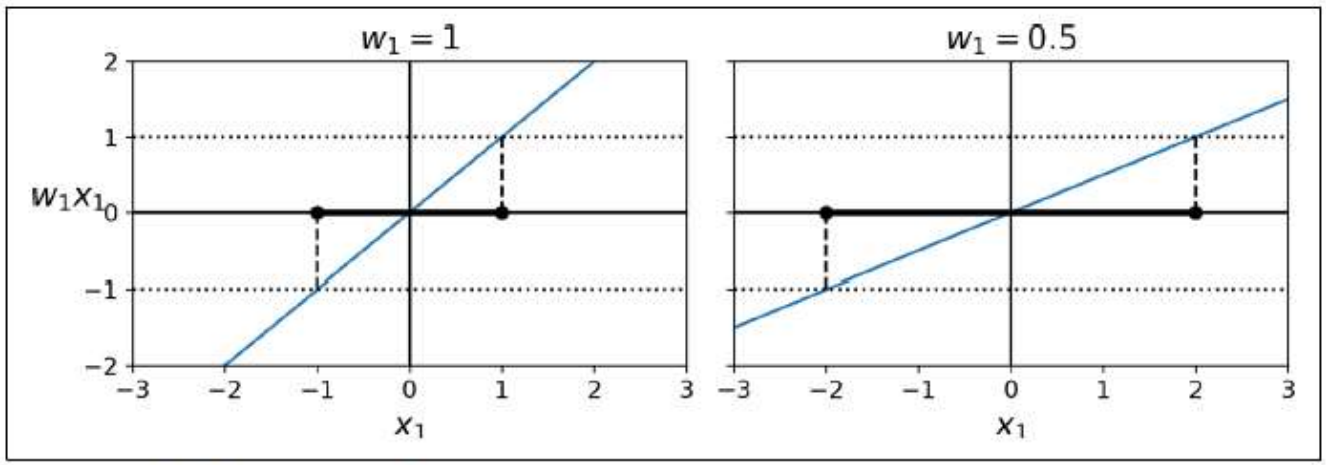


Figure 5-13. A smaller weight vector results in a larger margin

instances. If we define $t^{(i)} = -1$ for negative instances (if $y^{(i)} = 0$) and $t^{(i)} = 1$ for positive instances (if $y^{(i)} = 1$), then we can express this constraint as $t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1$ for all instances.

Equation 5-3. Hard margin linear SVM classifier objective

$$\underset{\mathbf{w}, b}{\text{minimize}} \quad \frac{1}{2}\mathbf{w}^\top\mathbf{w}$$

$$\text{subject to} \quad t^{(i)}\left(\mathbf{w}^\top\mathbf{x}^{(i)} + b\right) \geq 1 \quad \text{for } i = 1, 2, \cdots, m$$

Training Objective: Soft margin linear SVM classifier objective

Equation 5-4. Soft margin linear SVM classifier objective

$$\underset{\mathbf{w}, b, \zeta}{\text{minimize}} \quad \frac{1}{2}\mathbf{w}^\intercal\mathbf{w} + C\sum_{i=1}^{m}\zeta^{(i)}$$

$$\text{subject to} \quad t^{(i)}\left(\mathbf{w}^\intercal\mathbf{x}^{(i)} + b\right) \geq 1 - \zeta^{(i)} \quad \text{and} \quad \zeta^{(i)} \geq 0 \quad \text{for } i = 1, 2, \cdots, m$$

To get the soft margin objective, we need to introduce a *slack variable* $\zeta^{(i)} \geq 0$ for each instance:[4] $\zeta^{(i)}$ measures how much the i^{th} instance is allowed to violate the margin. We

SVM: Quadratic Programming

Equation 5-5. Quadratic Programming problem

$$\begin{aligned} &\underset{\mathbf{p}}{\text{Minimize}} && \frac{1}{2}\mathbf{p}^{\intercal}\mathbf{H}\mathbf{p} + \mathbf{f}^{\intercal}\mathbf{p} \\ &\text{subject to} && \mathbf{A}\mathbf{p} \le \mathbf{b} \\ &\text{where} && \begin{cases} \mathbf{p} & \text{is an } n_p\text{-dimensional vector } (n_p = \text{number of parameters}), \\ \mathbf{H} & \text{is an } n_p \times n_p \text{ matrix}, \\ \mathbf{f} & \text{is an } n_p\text{-dimensional vector}, \\ \mathbf{A} & \text{is an } n_c \times n_p \text{ matrix } (n_c = \text{number of constraints}), \\ \mathbf{b} & \text{is an } n_c\text{-dimensional vector}. \end{cases} \end{aligned}$$

Note that the expression $\mathbf{A}\,\mathbf{p} \le \mathbf{b}$ defines n_c constraints: $\mathbf{p}^{\intercal}\,\mathbf{a}^{(i)} \le b^{(i)}$ for $i = 1, 2, \cdots, n_c$, where $\mathbf{a}^{(i)}$ is the vector containing the elements of the i^{th} row of $\mathbf{A}$ and $b^{(i)}$ is the i^{th} element of $\mathbf{b}$.

You can easily verify that if you set the QP parameters in the following way, you get the hard margin linear SVM classifier objective:

- $n_p = n + 1$, where n is the number of features (the +1 is for the bias term).
- $n_c = m$, where m is the number of training instances.
- $\mathbf{H}$ is the $n_p \times n_p$ identity matrix, except with a zero in the top-left cell (to ignore the bias term).
- $\mathbf{f} = 0$, an n_p-dimensional vector full of 0s.
- $\mathbf{b} = -1$, an n_c-dimensional vector full of −1s.
- $\mathbf{a}^{(i)} = -t^{(i)}\,\dot{\mathbf{x}}^{(i)}$, where $\dot{\mathbf{x}}^{(i)}$ is equal to $\mathbf{x}^{(i)}$ with an extra bias feature $\dot{\mathbf{x}}_0 = 1$.

SVM: Dual Problem

Equation 5-6. Dual form of the linear SVM objective

$$\underset{\alpha}{\text{minimize}} \quad \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha^{(i)}\alpha^{(j)}t^{(i)}t^{(j)}\mathbf{x}^{(i)\top}\mathbf{x}^{(j)} - \sum_{i=1}^{m}\alpha^{(i)}$$

$$\text{subject to } \alpha^{(i)} \geq 0 \text{ for all } i = 1, 2, \ldots, m \text{ and } \sum_{i=1}^{m}\alpha^{(i)}t^{(i)} = 0$$

Once you find the vector $\hat{\alpha}$ that minimizes this equation (using a QP solver), use Equation 5-7 to compute $\hat{\mathbf{w}}$ and $\hat{b}$ that minimize the primal problem.

Equation 5-7. From the dual solution to the primal solution

$$\hat{\mathbf{w}} = \sum_{i=1}^{m}\hat{\alpha}^{(i)}t^{(i)}\mathbf{x}^{(i)}$$

$$\hat{b} = \frac{1}{n_s}\sum_{\substack{i=1 \\ \hat{\alpha}^{(i)}>0}}^{m}\left(t^{(i)} - \hat{\mathbf{w}}^{\top}\mathbf{x}^{(i)}\right)$$

Where n_s is the number of support vectors.

Soft Margin Classification: Scikit-Learn

- LinearSVC class with C=1 and the hinge loss function

```
import numpy as np
from sklearn import datasets
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC

iris = datasets.load_iris()
X = iris["data"][:, (2, 3)]  # petal length, petal width
y = (iris["target"] == 2).astype(np.float64)  # Iris virginica

svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("linear_svc", LinearSVC(C=1, loss="hinge")),
    ])

svm_clf.fit(X, y)
```

```
SVC(kernel="linear", C=1)
SGDClassifier(loss="hinge", alpha=1/(m*C))
```

```
>>> svm_clf.predict([[5.5, 1.7]])
array([1.])
```

Nonlinear SVM Classification

- Adding more features to make the data linearly separable

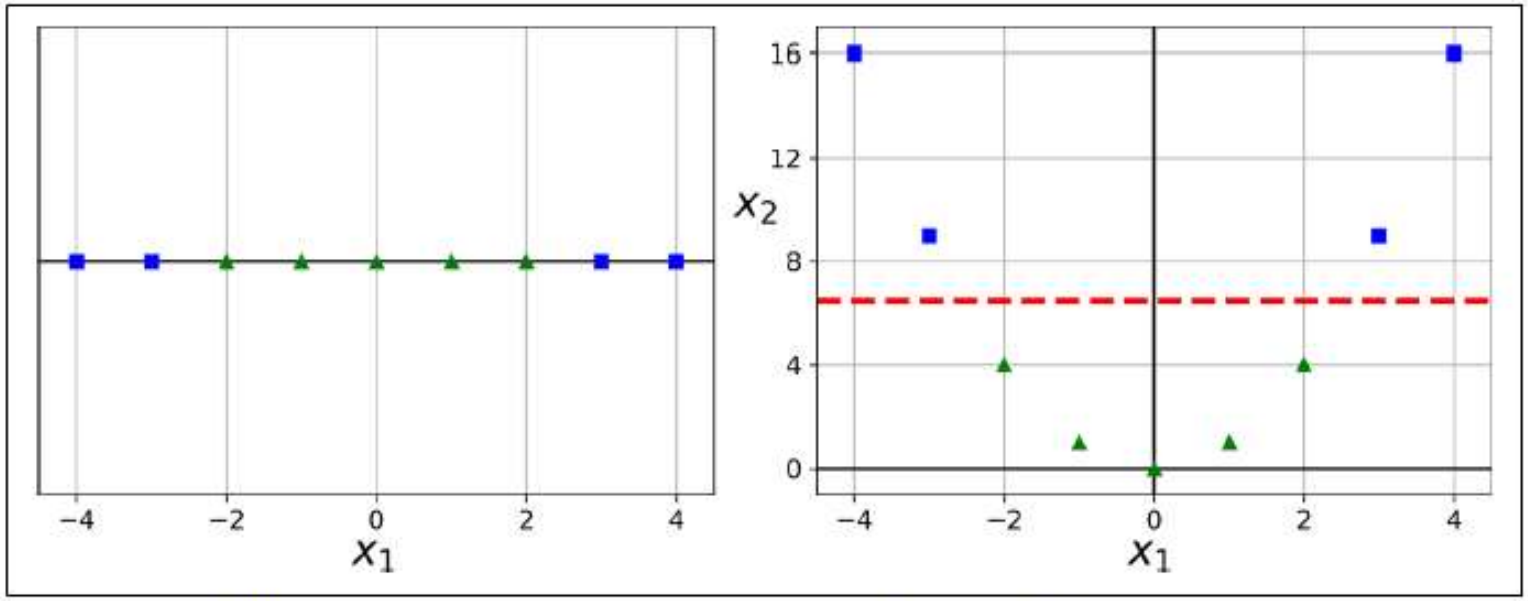


Figure 5-5. Adding features to make a dataset linearly separable

Nonlinear SVM Classification: Polynomial Feature Transformer

```
from sklearn.datasets import make_moons
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

X, y = make_moons(n_samples=100, noise=0.15)
polynomial_svm_clf = Pipeline([
        ("poly_features", PolynomialFeatures(degree=3)),
        ("scaler", StandardScaler()),
        ("svm_clf", LinearSVC(C=10, loss="hinge"))
    ])

polynomial_svm_clf.fit(X, y)
```

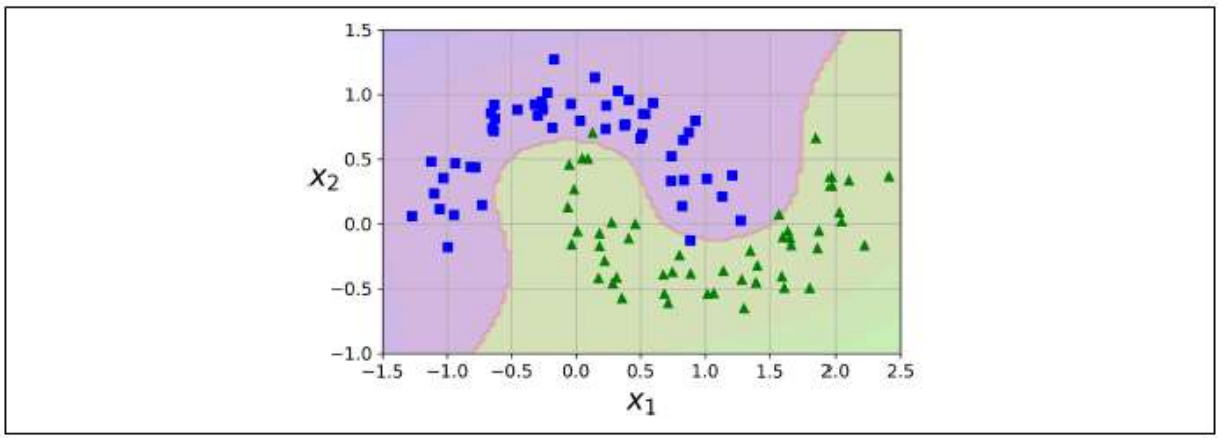


Figure 5-6. Linear SVM classifier using polynomial features

Polynomial Kernel

- Kernel trick: makes it possible to get the same result as if you had added many polynomial features, even with very high-degree polynomials, without actually having to add them.

- The hyperparameter coef0 controls how much the model is influenced by high-degree polynomials versus low-degree polynomials.

```
from sklearn.svm import SVC
poly_kernel_svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("svm_clf", SVC(kernel="poly", degree=3, coef0=1, C=5))
    ])
poly_kernel_svm_clf.fit(X, y)
```

Polynomial Kernel

- d: polynomial degrees
- r: higher-degree weight (coef0)
- C: hyperparameter to control margin violations

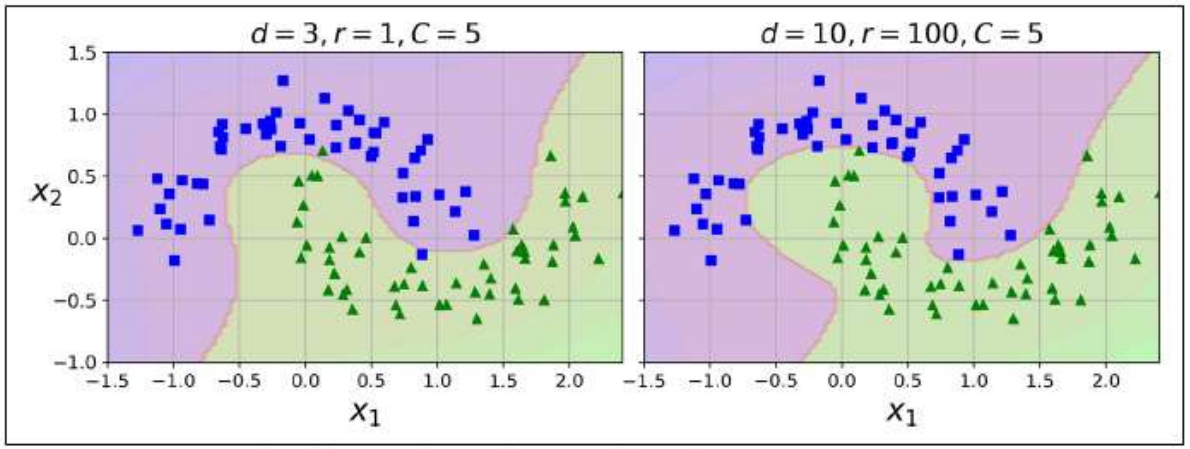


Figure 5-7. SVM classifiers with a polynomial kernel

Similarity Features

- For example, let's look at the instance $x1 = -1$: it is located at a distance of 1 from the first landmark and 2 from the second landmark. Therefore its new features are $x2 = \exp(-0.3 \times 12) \approx 0.74$ and $x3 = \exp(-0.3 \times 22) \approx 0.30$.

Equation 5-1. Gaussian RBF

$$\phi_\gamma(\mathbf{x}, \ell) = \exp\left(-\gamma \| \mathbf{x} - \ell \|^2\right)$$

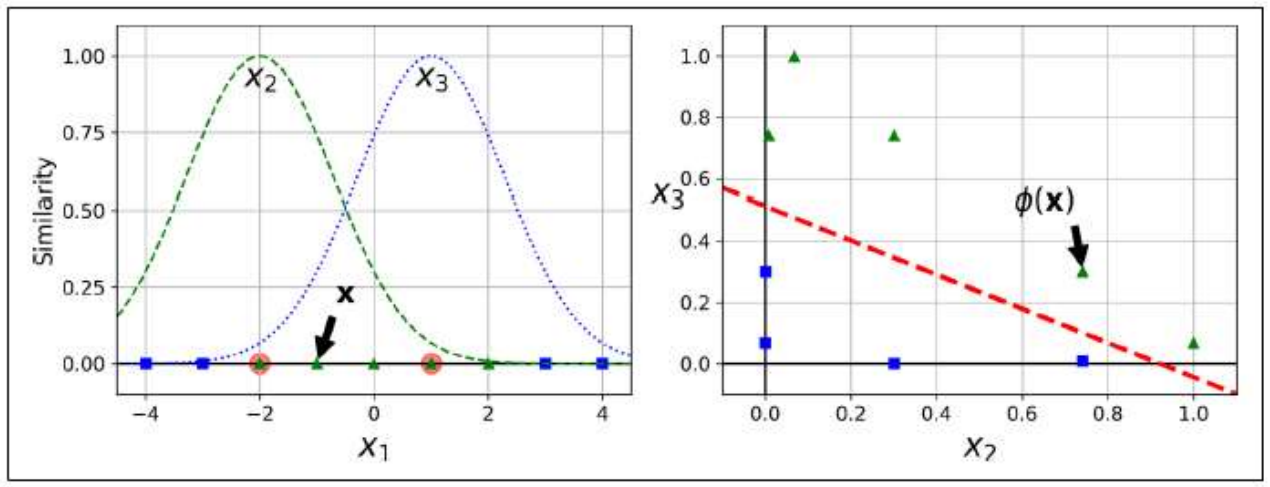


Figure 5-8. Similarity features using the Gaussian RBF

Gaussian RBF Kernel

- The kernel trick does its SVM magic, making it possible to obtain a similar result as if you had added many similarity features.

- Gamma works as a hyperparameter. If your model is overfitting, you can reduce gamma. If it is underfitting, you should increase it (similar to C hyperparameter)

```
rbf_kernel_svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("svm_clf", SVC(kernel="rbf", gamma=5, C=0.001))
    ])
rbf_kernel_svm_clf.fit(X, y)
```

Gaussian RBF Kernel

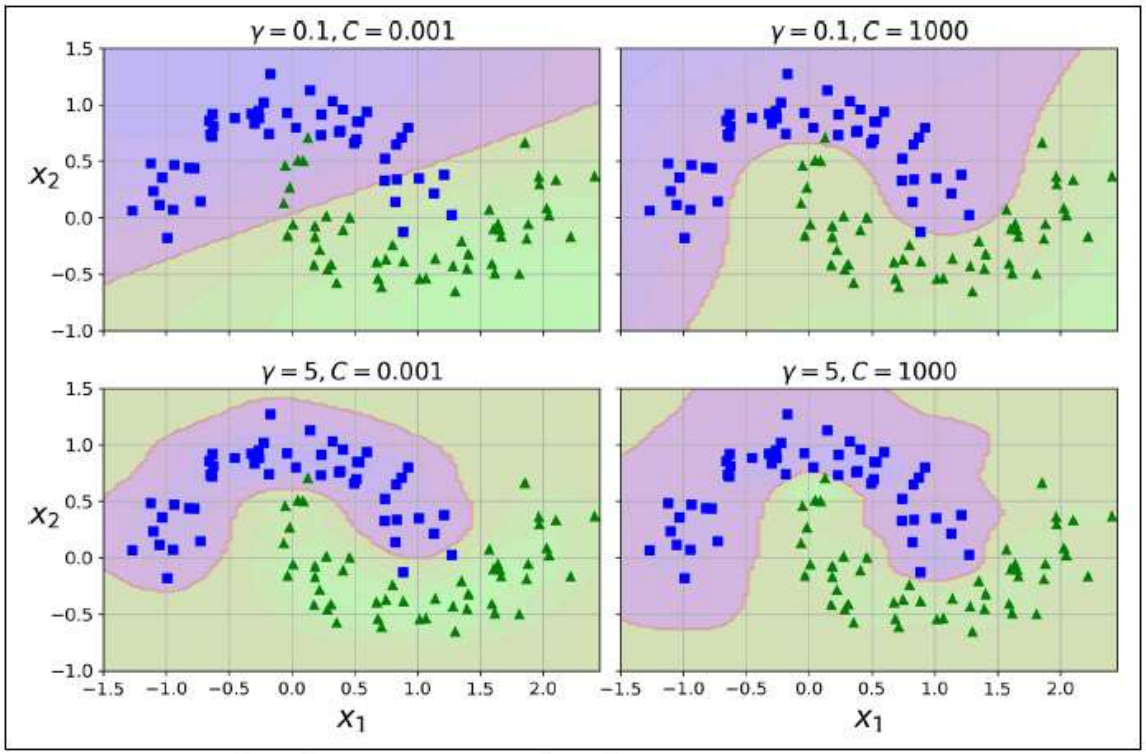


Figure 5-9. SVM classifiers using an RBF kernel

Kernel SVM: Dual Problem

- So, you don't need to transform the training instances at all; just replace the dot product by its square in Equation 5-6. The result will be strictly the same as if you had gone through the trouble of transforming the training set then fitting a linear SVM algorithm, but this trick makes the whole process much more computationally efficient.

$$\left(\mathbf{x}^{(i)\mathsf{T}}\mathbf{x}^{(j)}\right)^2$$

Equation 5-6. Dual form of the linear SVM objective

$$\underset{\alpha}{\text{minimize}} \quad \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha^{(i)}\alpha^{(j)}t^{(i)}t^{(j)}\mathbf{x}^{(i)\mathsf{T}}\mathbf{x}^{(j)} - \sum_{i=1}^{m}\alpha^{(i)}$$

$$\text{subject to } \alpha^{(i)} \geq 0 \text{ for all } i = 1, 2, \ldots, m \text{ and } \sum_{i=1}^{m}\alpha^{(i)}t^{(i)} = 0$$

Once you find the vector $\hat{\alpha}$ that minimizes this equation (using a QP solver), use Equation 5-7 to compute $\hat{\mathbf{w}}$ and $\hat{b}$ that minimize the primal problem.

Equation 5-7. From the dual solution to the primal solution

$$\hat{\mathbf{w}} = \sum_{i=1}^{m}\hat{\alpha}^{(i)}t^{(i)}\mathbf{x}^{(i)}$$

$$\hat{b} = \frac{1}{n_s}\sum_{\substack{i=1 \\ \hat{\alpha}^{(i)}>0}}^{m}\left(t^{(i)} - \hat{\mathbf{w}}^{\mathsf{T}}\mathbf{x}^{(i)}\right)$$

Where n_s is the number of support vectors.

Kernel SVM

Equation 5-10. Common kernels

$$
\begin{aligned}
\text{Linear:} \quad & K(\mathbf{a}, \mathbf{b}) = \mathbf{a}^\top \mathbf{b} \\
\text{Polynomial:} \quad & K(\mathbf{a}, \mathbf{b}) = (\gamma \mathbf{a}^\top \mathbf{b} + r)^d \\
\text{Gaussian RBF:} \quad & K(\mathbf{a}, \mathbf{b}) = \exp\left(-\gamma \| \mathbf{a} - \mathbf{b} \|^2\right) \\
\text{Sigmoid:} \quad & K(\mathbf{a}, \mathbf{b}) = \tanh(\gamma \mathbf{a}^\top \mathbf{b} + r)
\end{aligned}
$$

Kernel SVM

Equation 5-11. Making predictions with a kernelized SVM

$$h_{\hat{\mathbf{w}},\hat{b}}\left(\phi\left(\mathbf{x}^{(n)}\right)\right) = \hat{\mathbf{w}}^{\mathsf{T}}\phi\left(\mathbf{x}^{(n)}\right) + \hat{b} = \left(\sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} \phi\left(\mathbf{x}^{(i)}\right)\right)^{\mathsf{T}} \phi\left(\mathbf{x}^{(n)}\right) + \hat{b}$$

$$= \sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} \left(\phi\left(\mathbf{x}^{(i)}\right)^{\mathsf{T}} \phi\left(\mathbf{x}^{(n)}\right)\right) + \hat{b}$$

$$= \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \hat{\alpha}^{(i)} t^{(i)} K\left(\mathbf{x}^{(i)}, \mathbf{x}^{(n)}\right) + \hat{b}$$

Note that since $\alpha^{(i)} \neq 0$ only for support vectors, making predictions involves computing the dot product of the new input vector $\mathbf{x}^{(n)}$ with only the support vectors, not all the training instances. Of course, you need to use the same trick to compute the bias term $\hat{b}$ (Equation 5-12).

Equation 5-12. Using the kernel trick to compute the bias term

$$\hat{b} = \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \left(t^{(i)} - \hat{\mathbf{w}}^{\mathsf{T}} \phi\left(\mathbf{x}^{(i)}\right)\right) = \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \left(t^{(i)} - \left(\sum_{j=1}^{m} \hat{\alpha}^{(j)} t^{(j)} \phi\left(\mathbf{x}^{(j)}\right)\right)^{\mathsf{T}} \phi\left(\mathbf{x}^{(i)}\right)\right)$$

$$= \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \left(t^{(i)} - \sum_{\substack{j=1 \\ \hat{\alpha}^{(j)} > 0}}^{m} \hat{\alpha}^{(j)} t^{(j)} K\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)\right)$$

If you are starting to get a headache, it's perfectly normal: it's an unfortunate side effect of the kernel trick.

Computational Complexity

- M is the training instances and N is the number of features

Table 5-1. Comparison of Scikit-Learn classes for SVM classification

Class	Time complexity	Out-of-core support	Scaling required	Kernel trick
`LinearSVC`	$O(m \times n)$	No	Yes	No
`SGDClassifier`	$O(m \times n)$	Yes	Yes	No
`SVC`	$O(m^2 \times n)$ to $O(m^3 \times n)$	No	Yes	Yes

SVM Regression

- Instead of trying to fit the largest possible street between two classes while limiting margin violations, SVM Regression tries to fit as many instances as possible *on* the street while limiting margin violations (i.e., instances *off* the street) (sigma is a hyperparameter to control the width of the street)

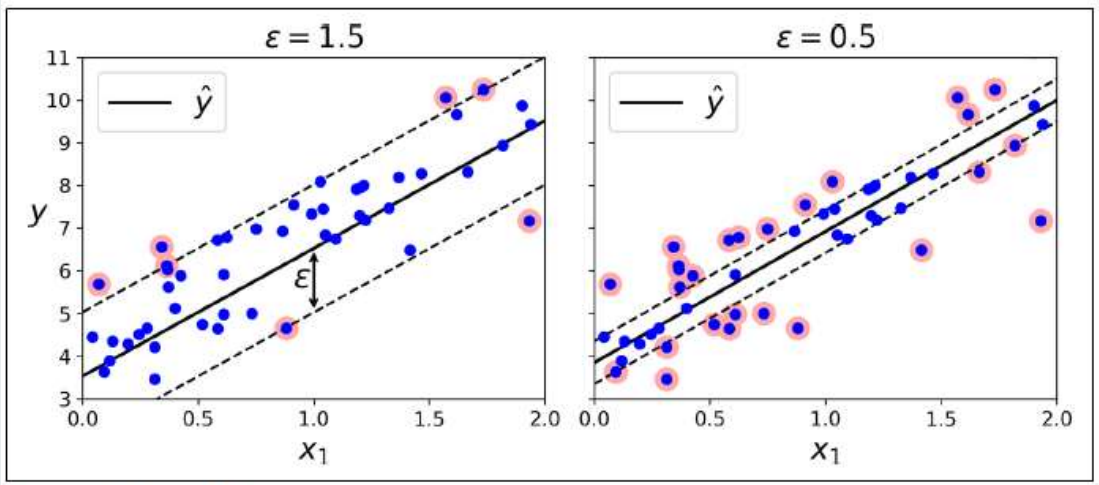


Figure 5-10. SVM Regression

SVM Regression : Scikit-Learn

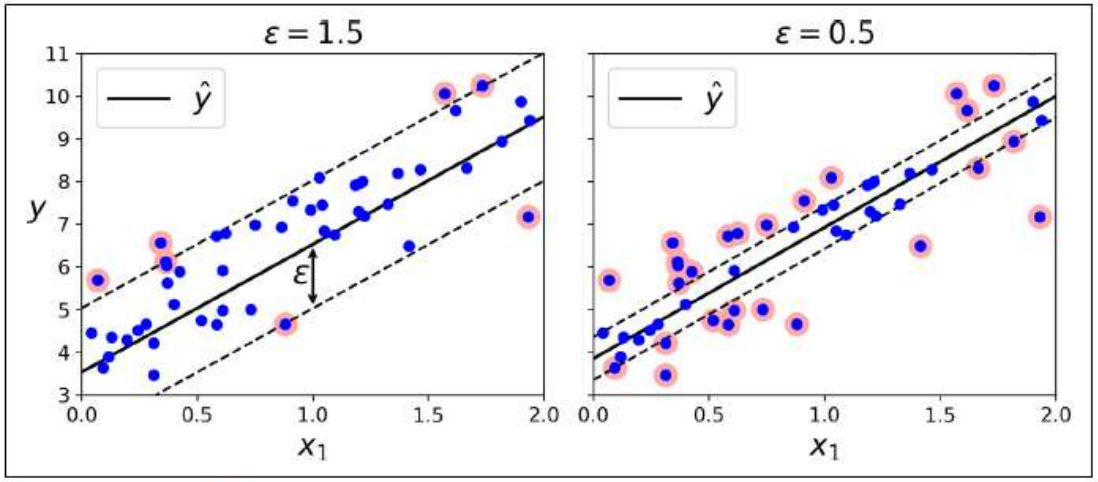


Figure 5-10. SVM Regression

```
from sklearn.svm import LinearSVR

svm_reg = LinearSVR(epsilon=1.5)
svm_reg.fit(X, y)
```

SVM Nonlinear Regression : Scikit-Learn

- More regularization in the right plot smaller C

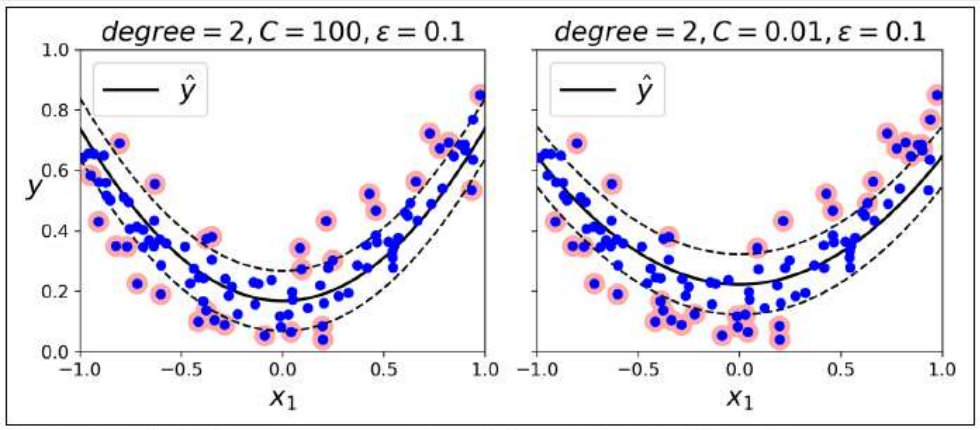


Figure 5-11. SVM Regression using a second-degree polynomial kernel

```
from sklearn.svm import SVR

svm_poly_reg = SVR(kernel="poly", degree=2, C=100, epsilon=0.1)
svm_poly_reg.fit(X, y)
```

SVM: Decision Function and Predictions

- More regularization in the right plot smaller C

Equation 5-2. Linear SVM classifier prediction

$$\hat{y} = \begin{cases} 0 \text{ if } \mathbf{w}^T\mathbf{x} + b < 0, \\ 1 \text{ if } \mathbf{w}^T\mathbf{x} + b \geq 0 \end{cases}$$

```
from sklearn.svm import SVR

svm_poly_reg = SVR(kernel="poly", degree=2, C=100, epsilon=0.1)
svm_poly_reg.fit(X, y)
```

Online SVM

- The first sum in the cost function will push the model to have a small weight vector **w**, leading to a larger margin. The second sum computes the total of all margin violations. An instance's margin violation is equal to 0 if it is located off the street and on the correct side, or else it is proportional to the distance to the correct side of the street. Minimizing this term ensures that the model makes the margin violations as small and as few as possible.

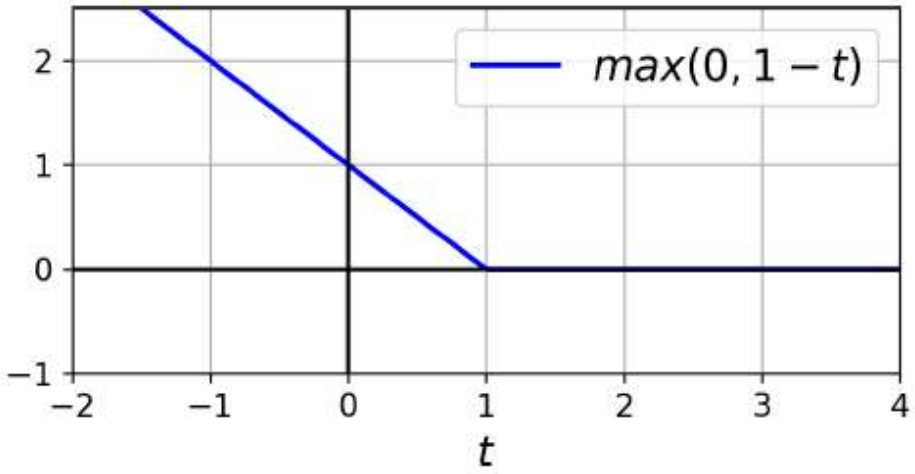
Equation 5-13. Linear SVM classifier cost function

$$J(\mathbf{w}, b) = \frac{1}{2}\mathbf{w}^T\mathbf{w} \quad + \quad C\sum_{i=1}^{m} max\left(0, 1 - t^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right)\right)$$

Online SVM

Hinge Loss

The function $max(0, 1 - t)$ is called the *hinge loss* function (see the following image). It is equal to 0 when $t \geq 1$. Its derivative (slope) is equal to -1 if $t < 1$ and 0 if $t > 1$. It is not differentiable at $t = 1$, but just like for Lasso Regression (see "Lasso Regression" on page 137), you can still use Gradient Descent using any *subderivative* at $t = 1$ (i.e., any value between -1 and 0).



Equation 5-13. Linear SVM classifier

$$J(\mathbf{w}, b) = \frac{1}{2}\mathbf{w}^\intercal\mathbf{w} \quad + \quad C\sum_{i=1}^{m} max\left(0, 1 - t^{(i)}\left(\mathbf{w}^\intercal\mathbf{x}^{(i)} + b\right)\right)$$

Online SVM

- It is also possible to implement online kernelized SVMs, as described in the papers "Incremental and Decremental Support Vector Machine Learning"
- "Fast Kernel Classifiers with Online and Active Learning".
- These kernelized SVMs are implemented in Matlab and C++.
- For large-scale nonlinear problems, you may want to consider using neural networks instead (see Part II).