

Figure 5-1. Large margin classification

Notice that adding more training instances “off the street” will not affect the decision boundary at all: it is fully determined (or “supported”) by the instances located on the edge of the street. These instances are called the *support vectors* (they are circled in Figure 5-1).



SVMs are sensitive to the feature scales, as you can see in Figure 5-2: on the left plot, the vertical scale is much larger than the horizontal scale, so the widest possible street is close to horizontal. After feature scaling (e.g., using Scikit-Learn’s `StandardScaler`), the decision boundary looks much better (on the right plot).

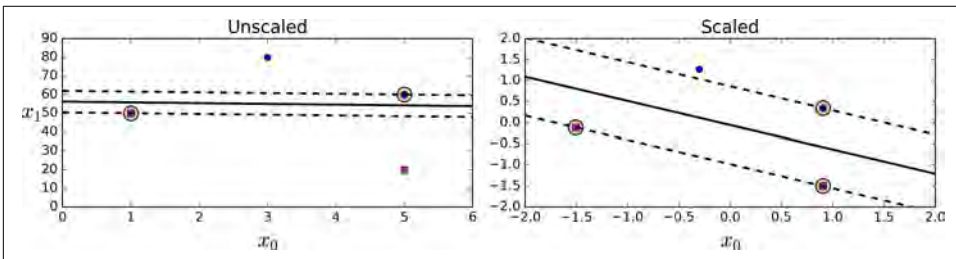


Figure 5-2. Sensitivity to feature scales

Soft Margin Classification

If we strictly impose that all instances be off the street and on the right side, this is called *hard margin classification*. There are two main issues with hard margin classification. First, it only works if the data is linearly separable, and second it is quite sensitive to outliers. Figure 5-3 shows the iris dataset with just one additional outlier: on the left, it is impossible to find a hard margin, and on the right the decision boundary ends up very different from the one we saw in Figure 5-1 without the outlier, and it will probably not generalize as well.

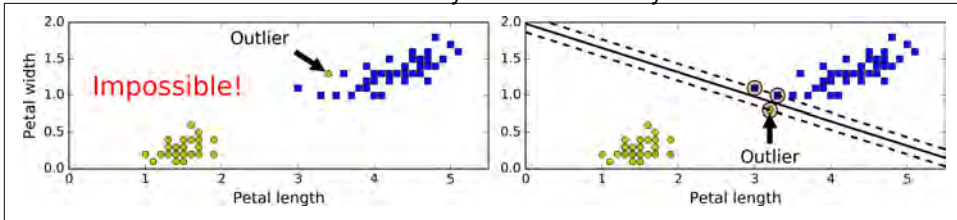


Figure 5-3. Hard margin sensitivity to outliers

To avoid these issues it is preferable to use a more flexible model. 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*.

In Scikit-Learn's SVM classes, you can control this balance using the C hyperparameter: a smaller C value leads to a wider street but more margin violations. Figure 5-4 shows the decision boundaries and margins of two soft margin SVM classifiers on a nonlinearly separable dataset. On the left, using a high C value the classifier makes fewer margin violations but ends up with a smaller margin. On the right, using a low C value the margin is much larger, but many instances end up on the street. However, it seems likely that the second classifier will generalize better: in fact even on this training set it makes fewer prediction errors, since most of the margin violations are actually on the correct side of the decision boundary.

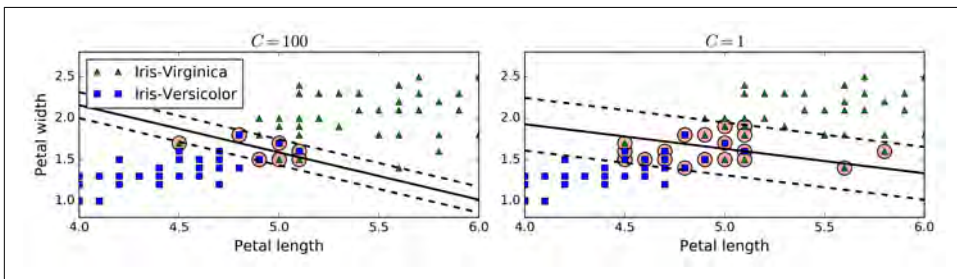


Figure 5-4. Fewer margin violations versus large margin



If your SVM model is overfitting, you can try regularizing it by reducing C .

The following Scikit-Learn code loads the iris dataset, scales the features, and then trains a linear SVM model (using the `LinearSVC` class with $C = 0.1$ and the *hinge loss*

Download from [finelybook www.finelybook.com](http://finelybook.com)
function, described shortly) to detect Iris-Virginica flowers. The resulting model is represented on the right of [Figure 5-4](#).

```
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_scaled, y)
```

Then, as usual, you can use the model to make predictions:

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



Unlike Logistic Regression classifiers, SVM classifiers do not output probabilities for each class.

Alternatively, you could use the SVC class, using `SVC(kernel="linear", C=1)`, but it is much slower, especially with large training sets, so it is not recommended. Another option is to use the `SGDClassifier` class, with `SGDClassifier(loss="hinge", alpha=1/(n*C))`. This applies regular Stochastic Gradient Descent (see [Chapter 4](#)) to train a linear SVM classifier. It does not converge as fast as the `LinearSVC` class, but it can be useful to handle huge datasets that do not fit in memory (out-of-core training), or to handle online classification tasks.



The `LinearSVC` class regularizes the bias term, so you should center the training set first by subtracting its mean. This is automatic if you scale the data using the `StandardScaler`. Moreover, make sure you set the `loss` hyperparameter to "hinge", as it is not the default value. Finally, for better performance you should set the `dual` hyperparameter to `False`, unless there are more features than training instances (we will discuss duality later in the chapter).

Nonlinear SVM Classification

Although linear SVM classifiers are efficient and work surprisingly well in many cases, many datasets are not even close to being linearly separable. One approach to handling nonlinear datasets is to add more features, such as polynomial features (as you did in [Chapter 4](#)); in some cases this can result in a linearly separable dataset. Consider the left plot in [Figure 5-5](#): it represents a simple dataset with just one feature x_1 . This dataset is not linearly separable, as you can see. But if you add a second feature $x_2 = (x_1)^2$, the resulting 2D dataset is perfectly linearly separable.

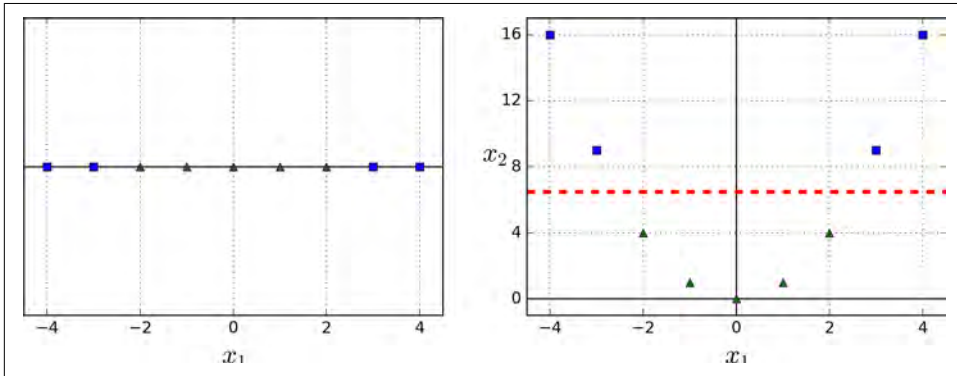


Figure 5-5. Adding features to make a dataset linearly separable

To implement this idea using Scikit-Learn, you can create a Pipeline containing a PolynomialFeatures transformer (discussed in “[Polynomial Regression](#)” on [page 121](#)), followed by a StandardScaler and a LinearSVC. Let’s test this on the moons dataset (see [Figure 5-6](#)):

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

polynomial_svm_clf = Pipeline((
    ("poly_features", PolynomialFeatures(degree=3)),
    ("scaler", StandardScaler()),
    ("svm_clf", LinearSVC(C=10, loss="hinge"))
))

polynomial_svm_clf.fit(X, y)
```