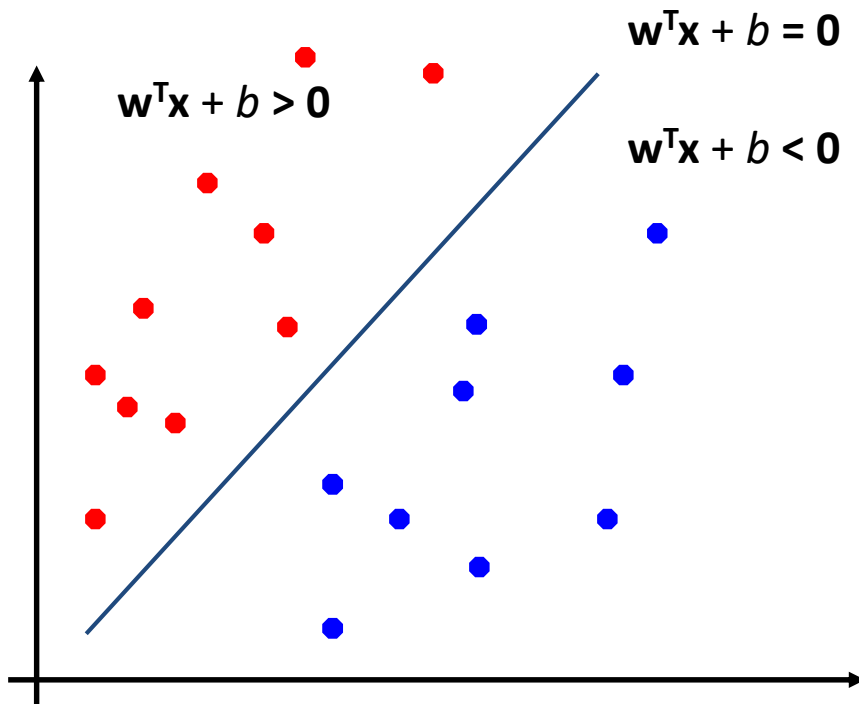


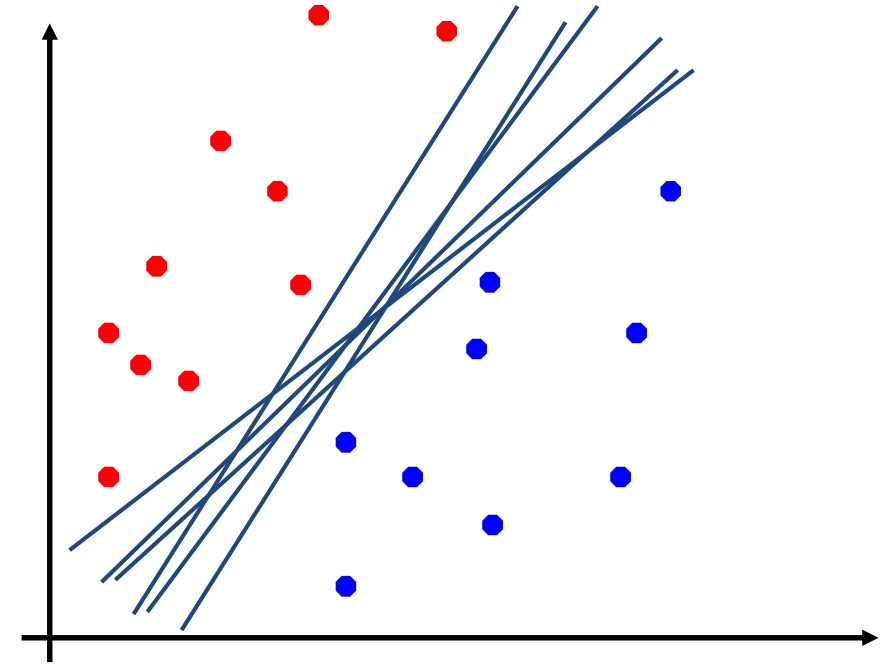
Support Vector Machine

Support Vector Machine

Binary classification can be viewed as the task of separating classes in feature space:



$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x} + b)$$



The objective of the support vector machine algorithm is to find a hyperplane in an N -dimensional space (N — the number of features) that distinctly classifies the data points.

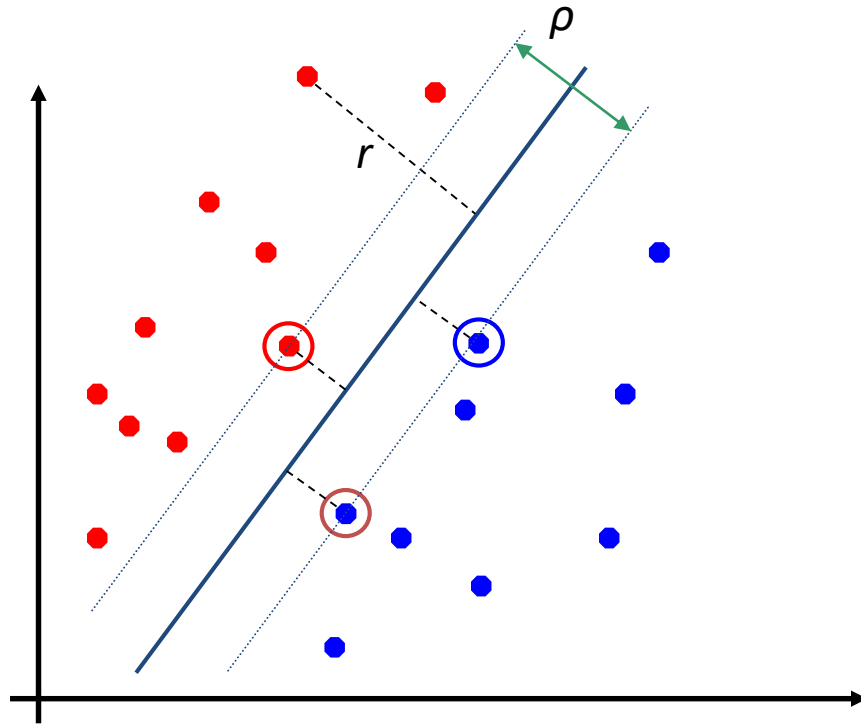
Support Vector Machine

Binary classification can be viewed as the task of separating classes in feature space:

Distance from example \mathbf{x}_i to the separator is $r = \frac{\mathbf{w}^T \mathbf{x}_i + b}{\|\mathbf{w}\|}$

Examples closest to the hyperplane are **support vectors**.

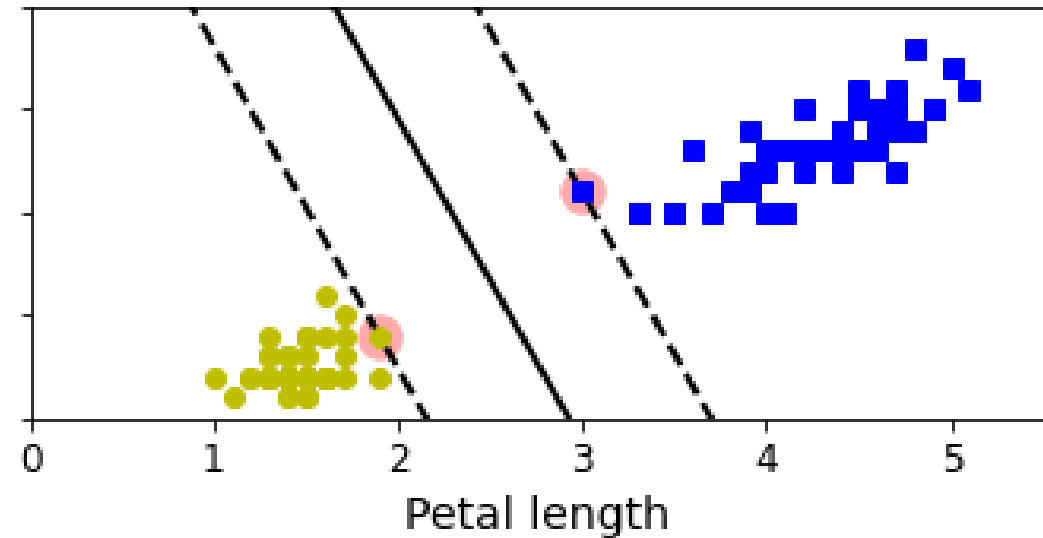
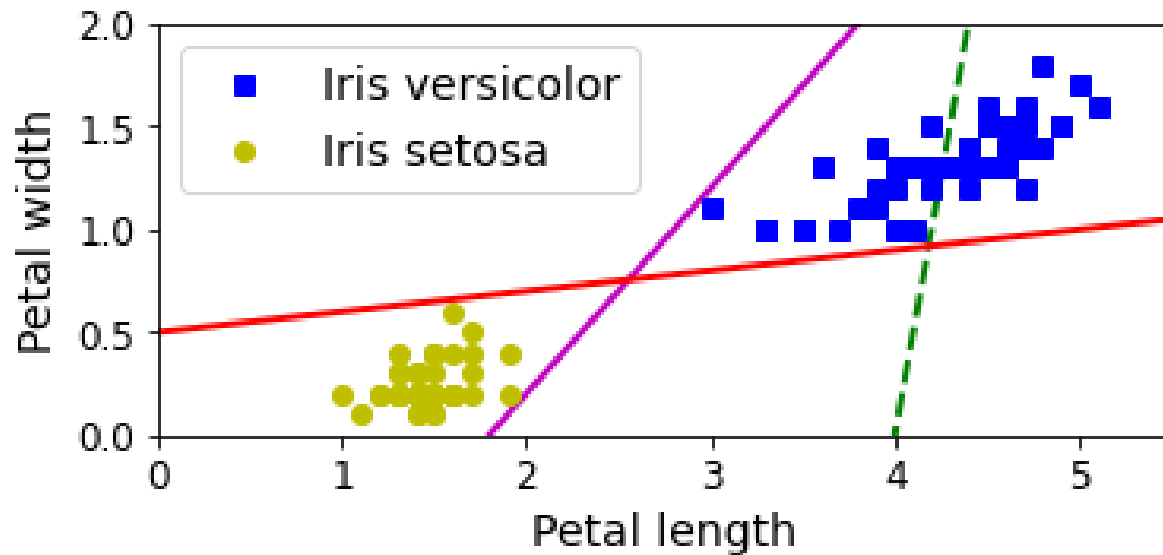
Margin ρ of the separator is the distance between support vectors



Support Vector Machine

A Support Vector Machine (SVM) is a very powerful and versatile Machine Learning model, capable of performing linear or nonlinear classification, regression, and even outlier detection. It is one of the most popular models in Machine Learning, and any-one interested in Machine Learning should have it in their toolbox. SVMs are particularly well suited for classification of complex but small- or medium-sized datasets.

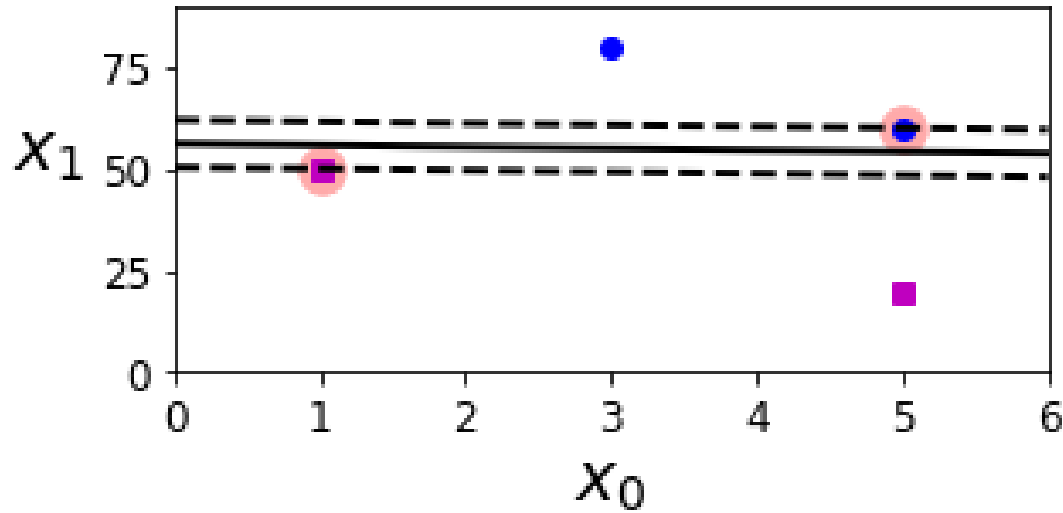
Linear SVM Classification



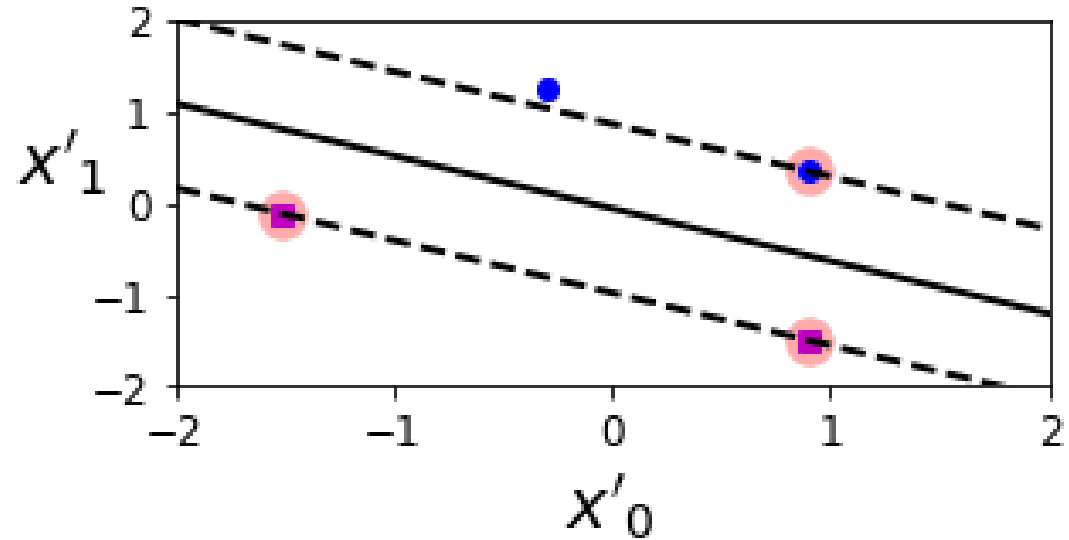
Support Vector Machine

Linear SVM Classification

Unscaled



Scaled



SVMs are sensitive to the feature scales; 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)

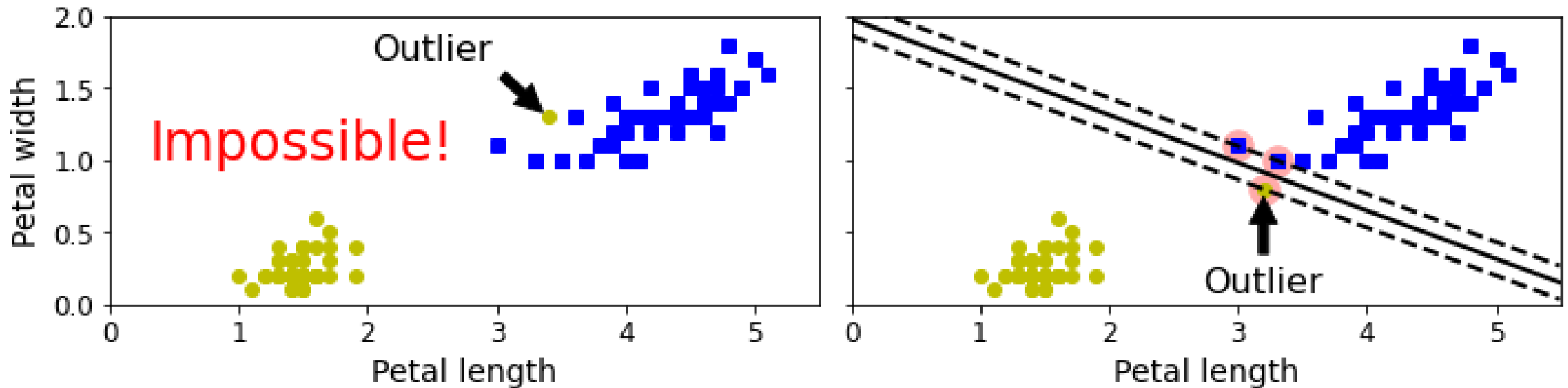
Support Vector Machine

Linear SVM Classification

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 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 without the outlier, and it will probably not generalize as well



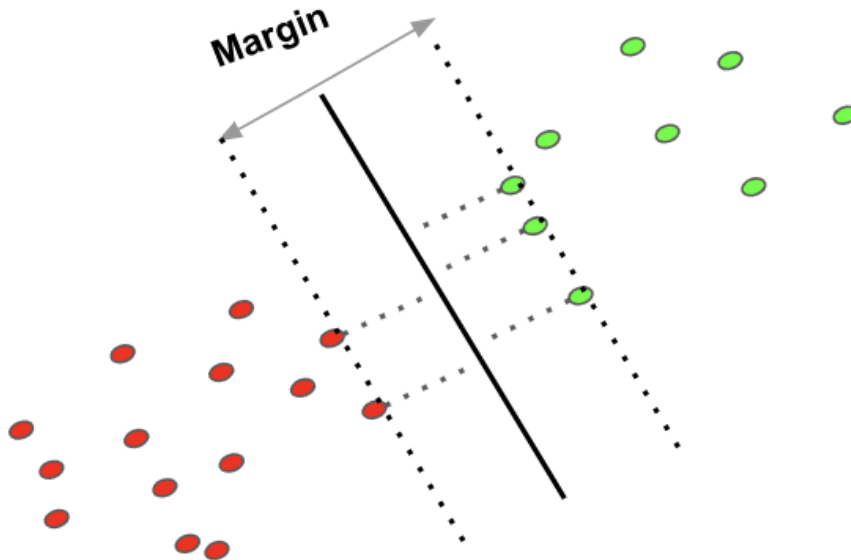
Support Vector Machine

Linear SVM Classification

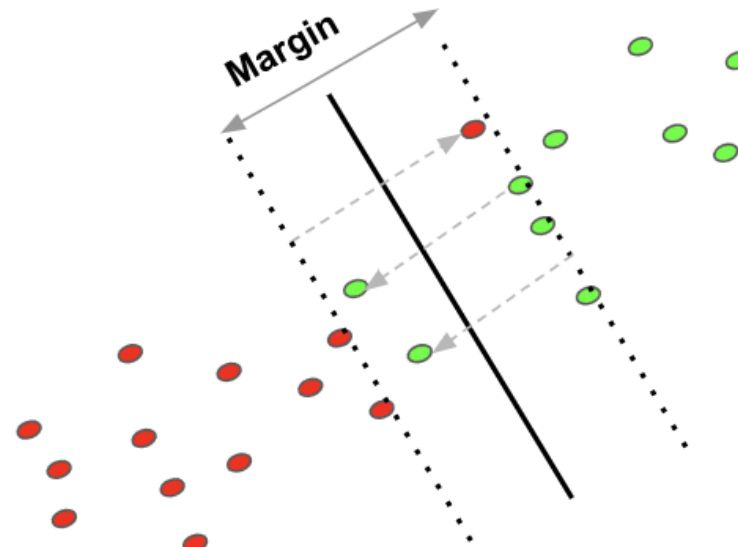
Soft Margin Classification

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

Hard Margin

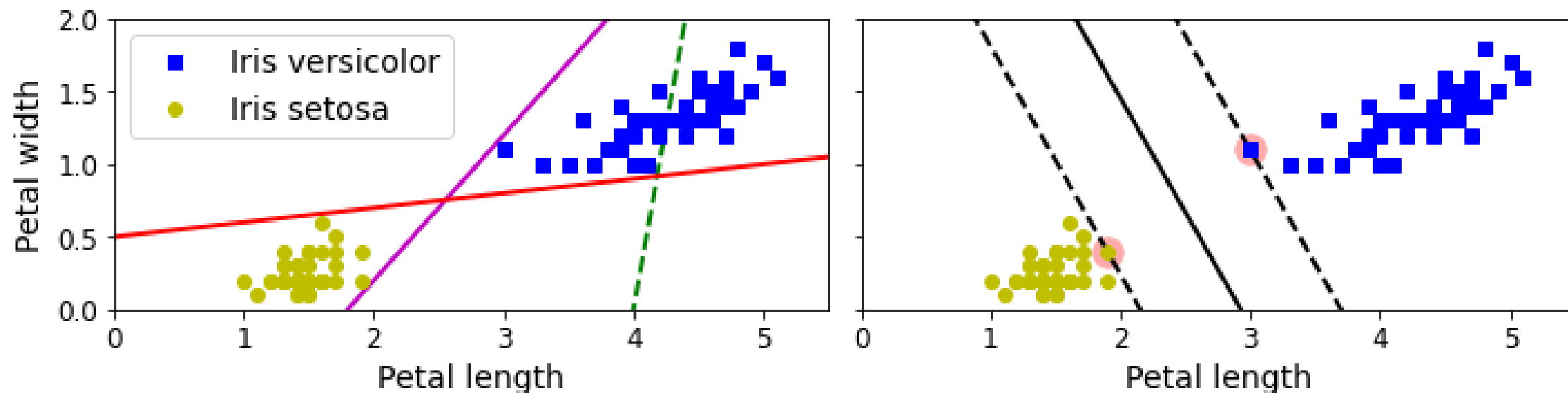


Soft Margin



Support Vector Machine

Linear SVM Classification



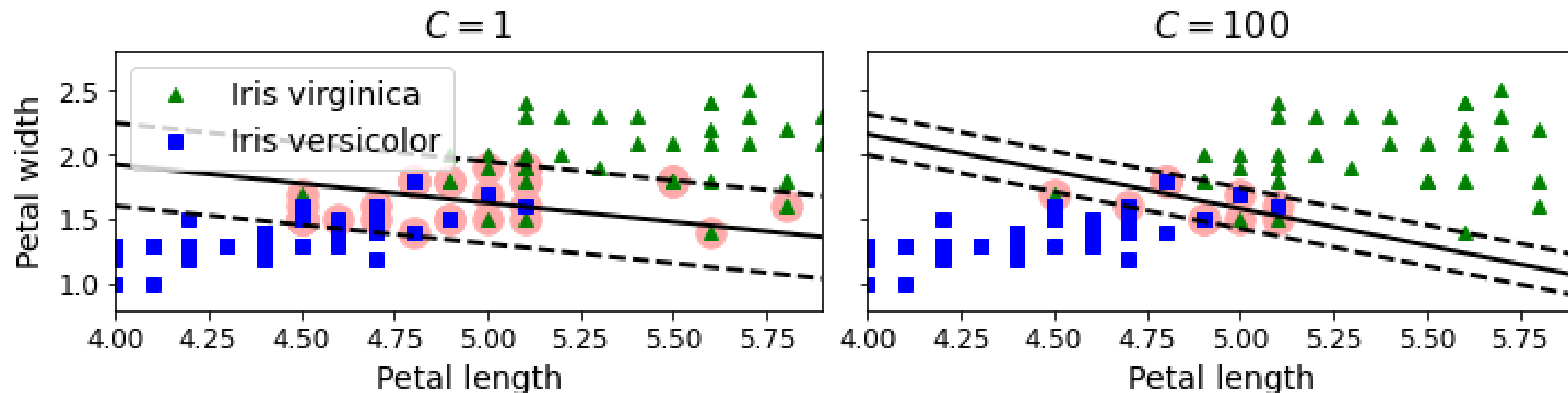
Notice that adding more training instances “off the street” will not affect the decision boundary at all: it is fully determined (or “supported”) by the instances located on the edge of the street. These instances are called the support vectors

Support Vector Machine

Linear SVM 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 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.



Support Vector Machine

Linear SVM Classification

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 function, described shortly) to detect Iris-Virginica flowers.

```
✓ 0s ▶ 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

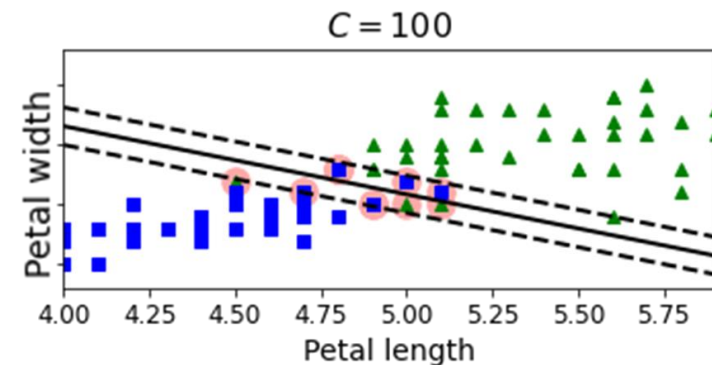
svm_clf = Pipeline([
    ("scaler", StandardScaler()),
    ("linear_svc", LinearSVC(C=1, loss="hinge", random_state=42)),
])

svm_clf.fit(X, y)

➞ Pipeline(steps=[('scaler', StandardScaler()),
                   ('linear_svc', LinearSVC(C=1, loss='hinge', random_state=42))])
```

```
✓ 0s ▶ svm_clf.predict([[5.5, 1.7]])

array([1.])
```



the hinge loss is a loss function used for training classifiers. The hinge loss is used for "maximum-margin" classification.

For an intended output $t = \pm 1$ and a classifier score y , the hinge loss of the prediction y is defined as $\ell(y) = \max(0, 1 - t \cdot y)$

Support Vector Machine

Linear SVM Classification

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 function, described shortly) to detect Iris-Virginica flowers.

```
✓ 0s ▶ 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

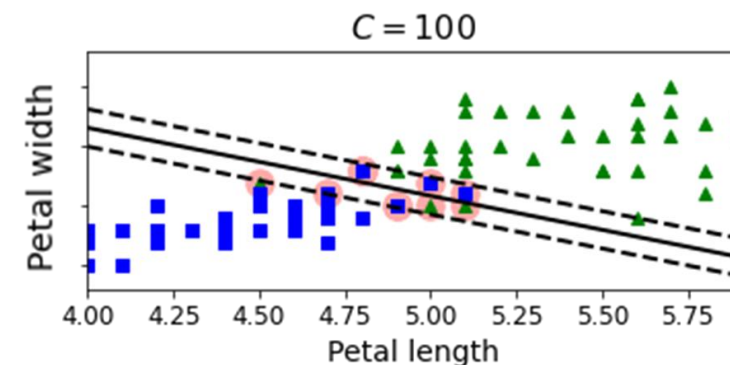
svm_clf = Pipeline([
    ("scaler", StandardScaler()),
    ("linear_svc", LinearSVC(C=1, loss="hinge", random_state=42)),
])

svm_clf.fit(X, y)
```

```
↳ Pipeline(steps=[('scaler', StandardScaler()),
                   ('linear_svc', LinearSVC(C=1, loss='hinge', random_state=42))])
```

```
✓ 0s ▶ svm_clf.predict([[5.5, 1.7]])

array([1.])
```



NOTE: 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.

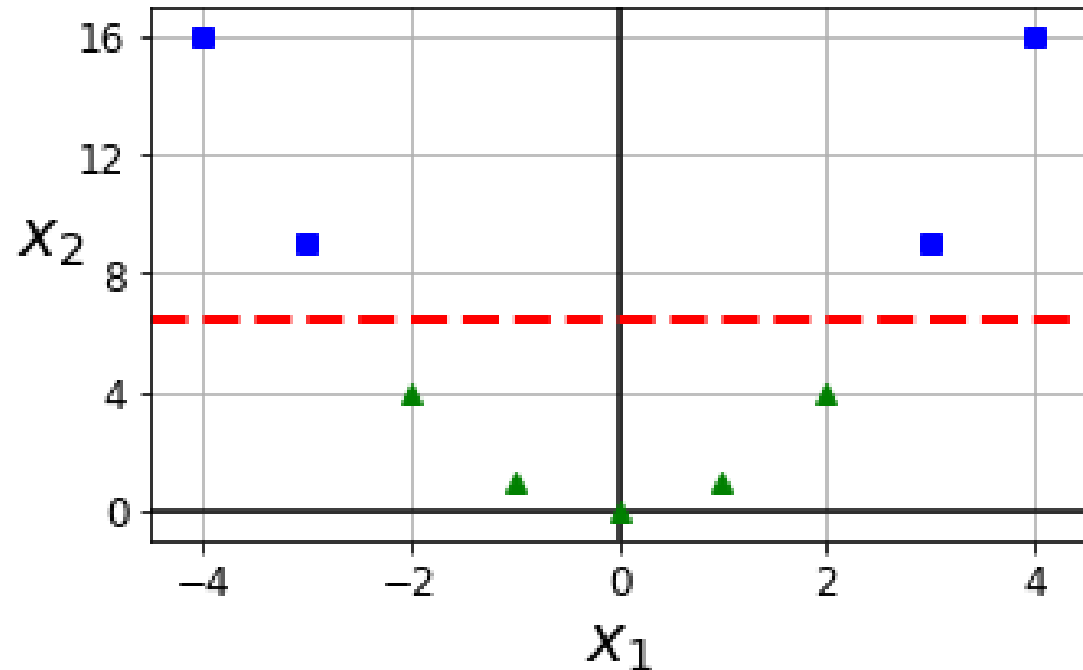
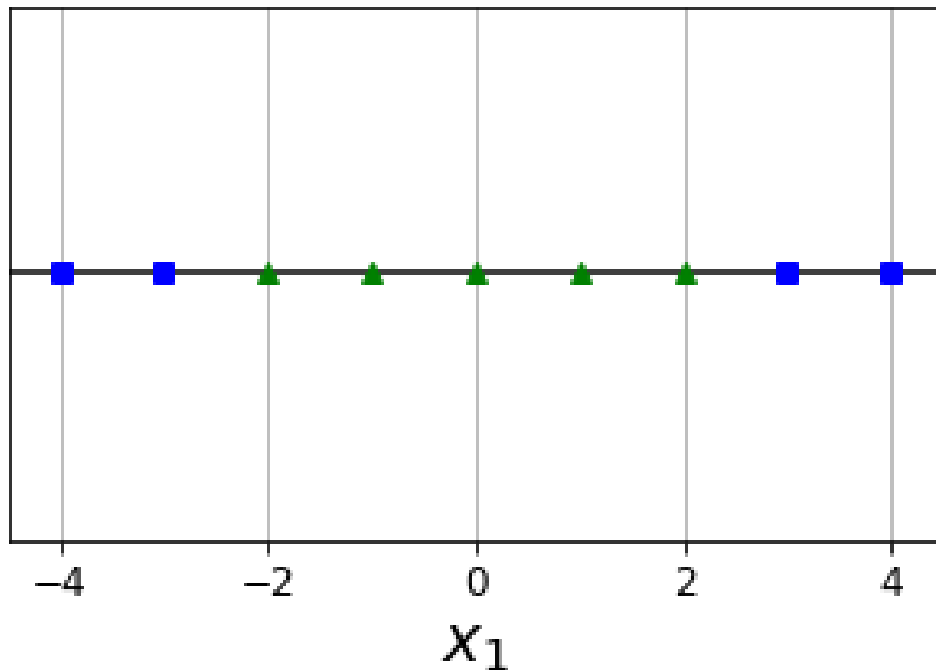
Support Vector Machine

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.

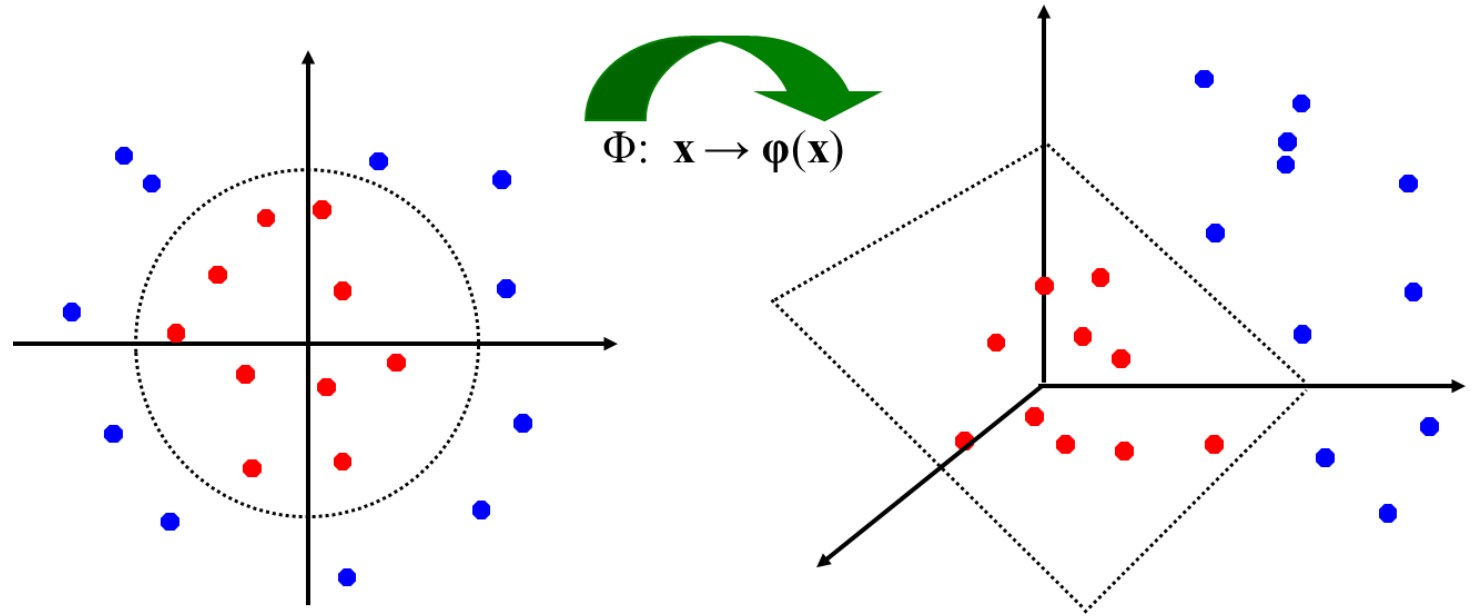
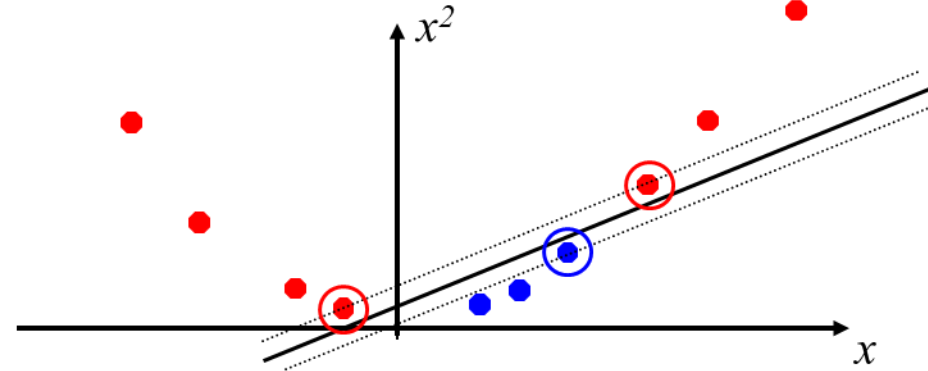
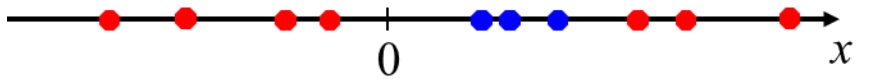
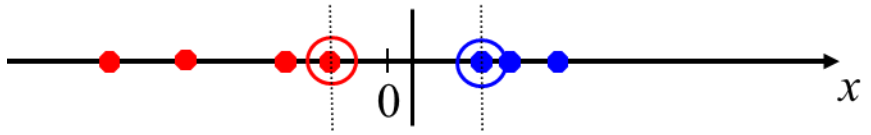
One approach to handling nonlinear datasets is to add more features, such as polynomial features in some cases this can result in a linearly separable dataset. Consider the left plot in Figure 5-5: it represents a simple dataset with just one feature x_1 .

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.



Support Vector Machine

Nonlinear SVM Classification



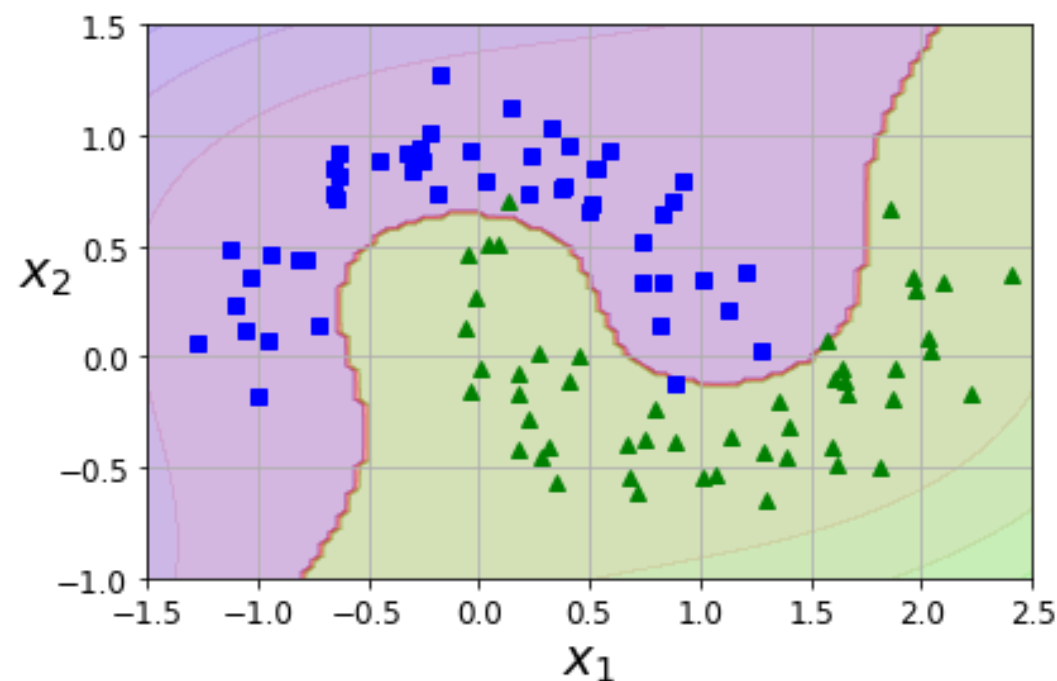
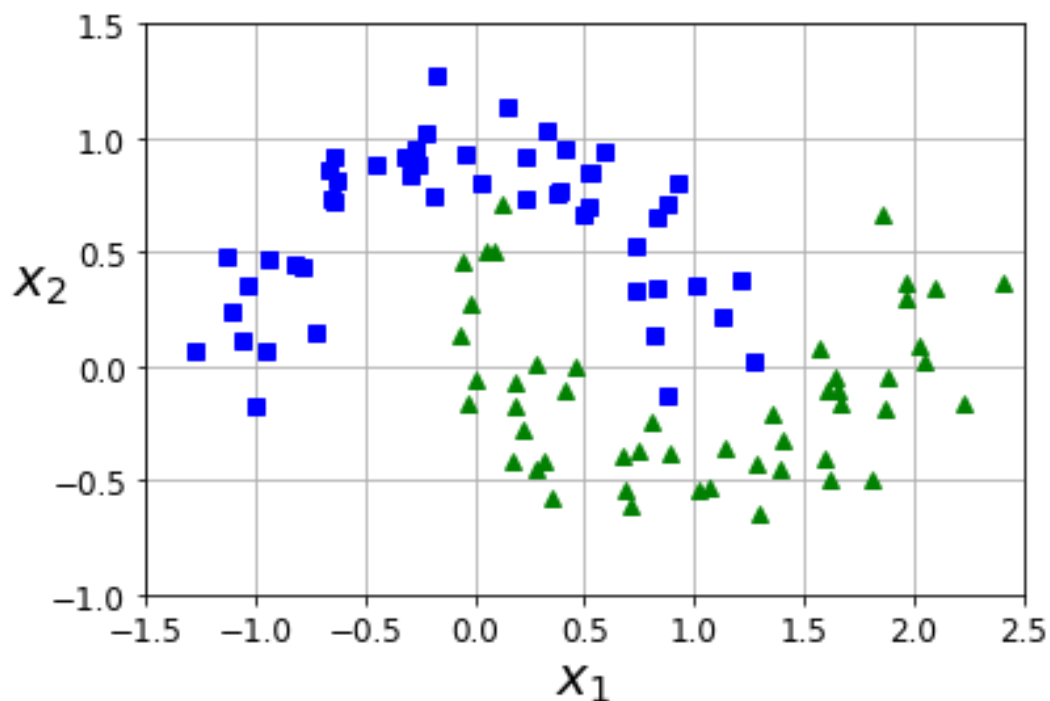
Support Vector Machine

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 in some cases this can result in a linearly separable dataset. Consider the left plot in Figure : 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.



Support Vector Machine

Nonlinear SVM Classification (Linear classifier using polynomial features)



```
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", random_state=42))
])

polynomial_svm_clf.fit(X, y)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:1208: ConvergenceWarning:
ConvergenceWarning,
Pipeline(steps=[('poly_features', PolynomialFeatures(degree=3)),
                ('scaler', StandardScaler()),
                ('svm_clf', LinearSVC(C=10, loss='hinge', random_state=42))])
```

Support Vector Machine

Nonlinear SVM Classification (Linear classifier using polynomial features)

```
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", random_state=42))
])
```

```
polynomial_svm_clf.fit(X, y)
```

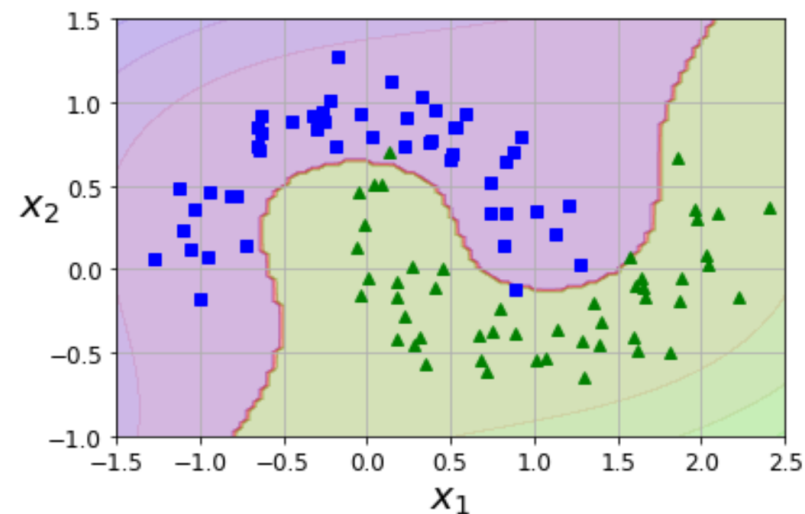
```
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:1208: ConvergenceWarning:
ConvergenceWarning,
Pipeline(steps=[('poly_features', PolynomialFeatures(degree=3)),
                 ('scaler', StandardScaler()),
                 ('svm_clf', LinearSVC(C=10, loss='hinge', random_state=42))])
```

```
[28] def plot_predictions(clf, axes):
    x0s = np.linspace(axes[0], axes[1], 100)
    x1s = np.linspace(axes[2], axes[3], 100)
    x0, x1 = np.meshgrid(x0s, x1s)
    X = np.c_[x0.ravel(), x1.ravel()]
    y_pred = clf.predict(X).reshape(x0.shape)
    y_decision = clf.decision_function(X).reshape(x0.shape)
    plt.contourf(x0, x1, y_pred, cmap=plt.cm.brg, alpha=0.2)
    plt.contourf(x0, x1, y_decision, cmap=plt.cm.brg, alpha=0.1)
```

```
plot_predictions(polynomial_svm_clf, [-1.5, 2.5, -1, 1.5])
plot_dataset(X, y, [-1.5, 2.5, -1, 1.5])
```

```
save_fig("moons_polynomial_svc_plot")
plt.show()
```

Saving figure moons_polynomial_svc_plot



Support Vector Machine

Nonlinear SVM Classification (Polynomial Kernel)

Adding polynomial features is simple to implement and can work great with all sorts of Machine Learning algorithms (not just SVMs), but at a low polynomial degree it cannot deal with very complex datasets, and with a high polynomial degree it creates a huge number of features, making the model too slow.

Fortunately, when using SVMs you can apply an almost miraculous mathematical Technique called the kernel trick (it is explained in a moment). It makes it possible to get the same result as if you added many polynomial features, even with very high degree polynomials, without actually having to add them. So there is no combinatorial explosion of the number of features since you don't actually add any features. This trick is implemented by the SVC class.

```
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)
```

Support Vector Machine

Nonlinear SVM Classification (Polynomial Kernel)

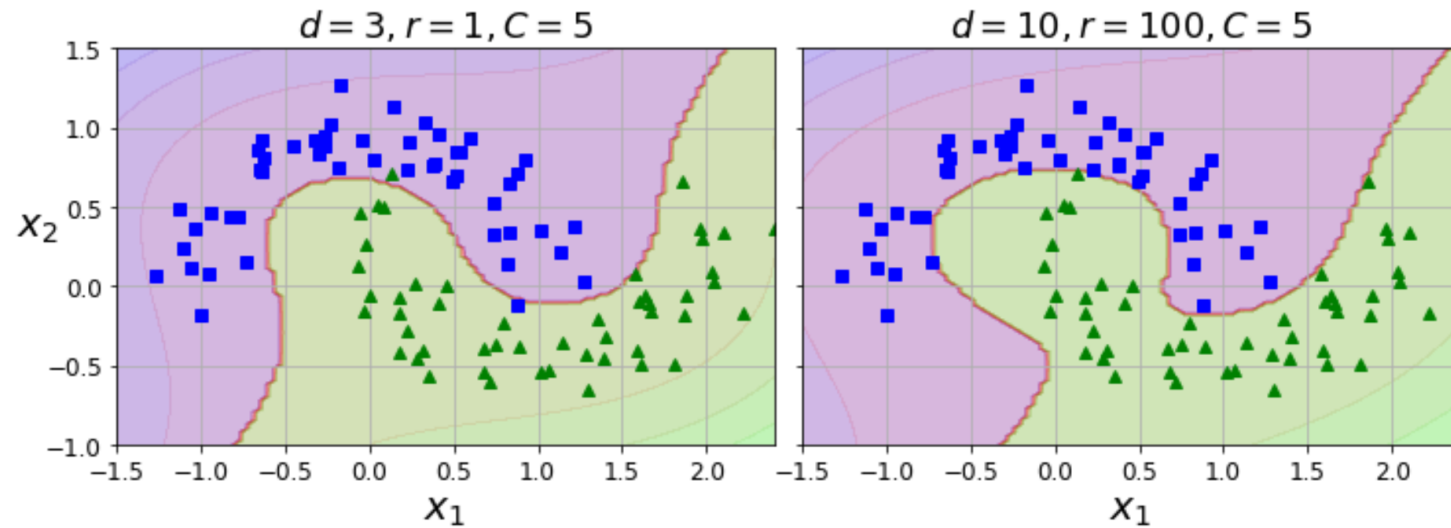
```
✓ 1s [▶] fig, axes = plt.subplots(ncols=2, figsize=(10.5, 4), sharey=True)

plt.sca(axes[0])
plot_predictions(poly_kernel_svm_clf, [-1.5, 2.45, -1, 1.5])
plot_dataset(X, y, [-1.5, 2.4, -1, 1.5])
plt.title(r"$d=3, r=1, C=5$", fontsize=18)

plt.sca(axes[1])
plot_predictions(poly100_kernel_svm_clf, [-1.5, 2.45, -1, 1.5])
plot_dataset(X, y, [-1.5, 2.4, -1, 1.5])
plt.title(r"$d=10, r=100, C=5$", fontsize=18)
plt.ylabel("")

save_fig("moons_kernelized_polynomial_svc_plot")
plt.show()
```

☞ Saving figure moons_kernelized_polynomial_svc_plot



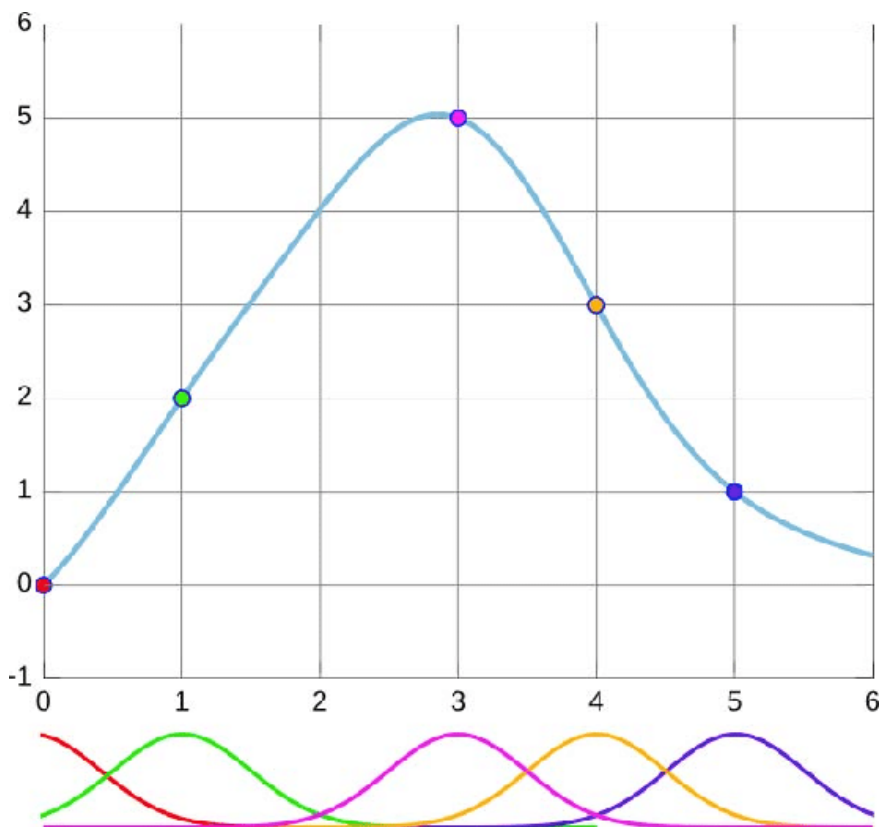
```
from sklearn.svm import SVC
```

```
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)
```

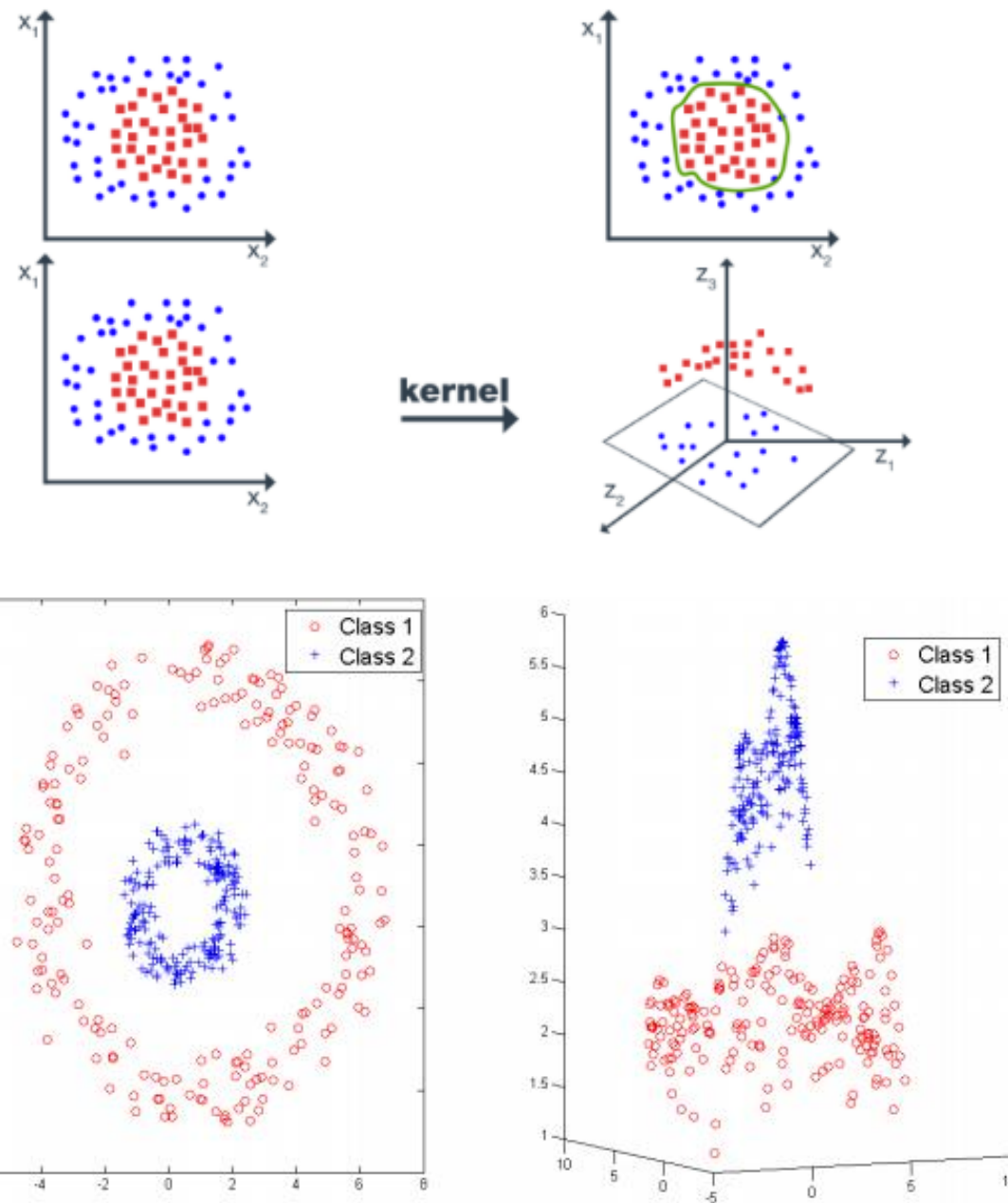
Support Vector Machine

Nonlinear SVM Classification (Adding Similarity Features)

Another technique to tackle nonlinear problems is to add features computed using a similarity function that measures how much each instance resembles a particular landmark.



Gaussian Radial Basis Function (RBF)

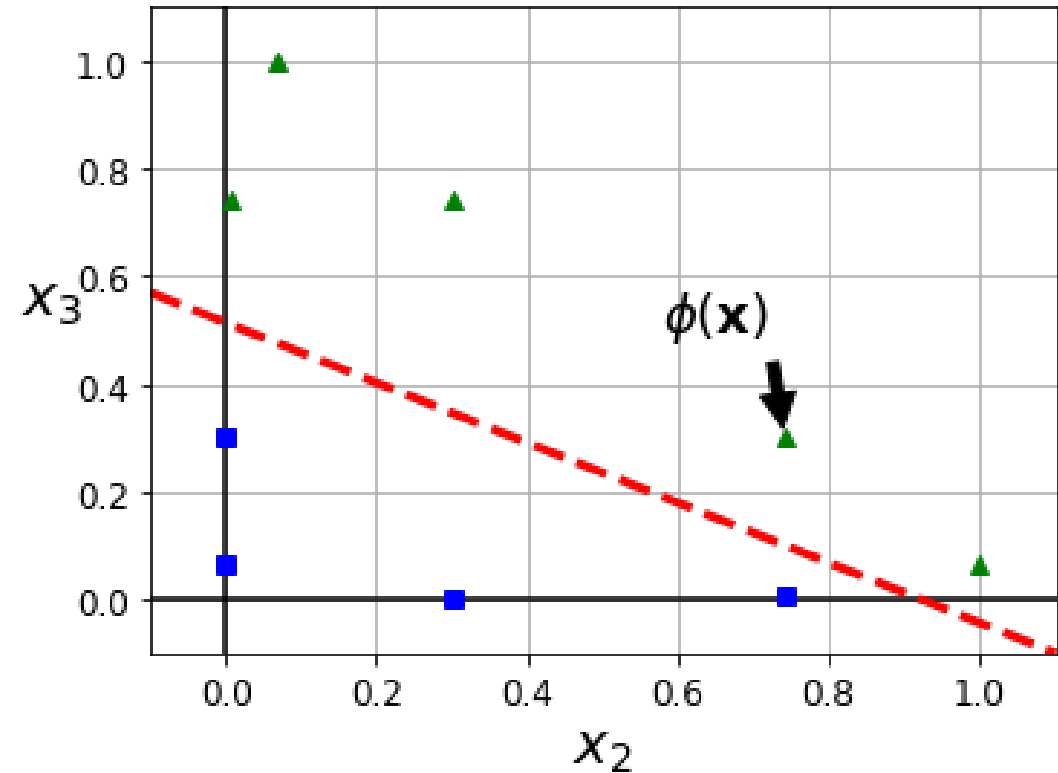
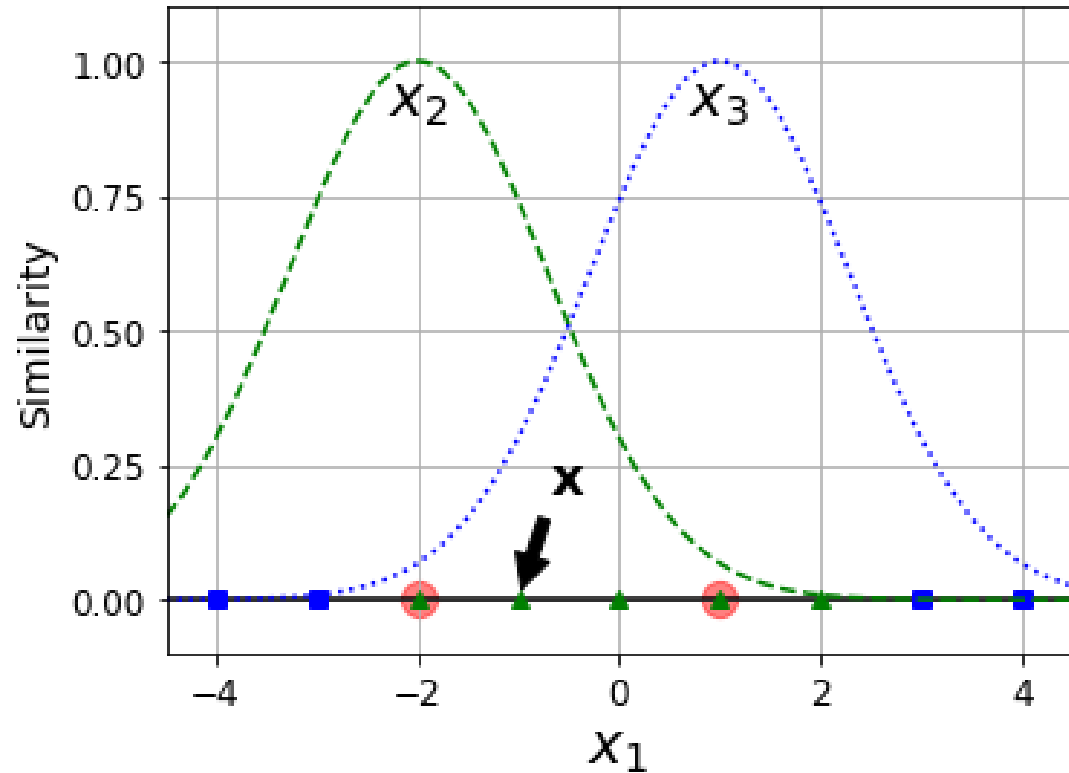


Support Vector Machine

Nonlinear SVM Classification (Adding Similarity Features)

Another technique to tackle nonlinear problems is to add features computed using a similarity function that measures how much each instance resembles a particular landmark.

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



Gaussian Radial Basis Function (RBF)

Support Vector Machine

Nonlinear SVM Classification (Adding Similarity Features)

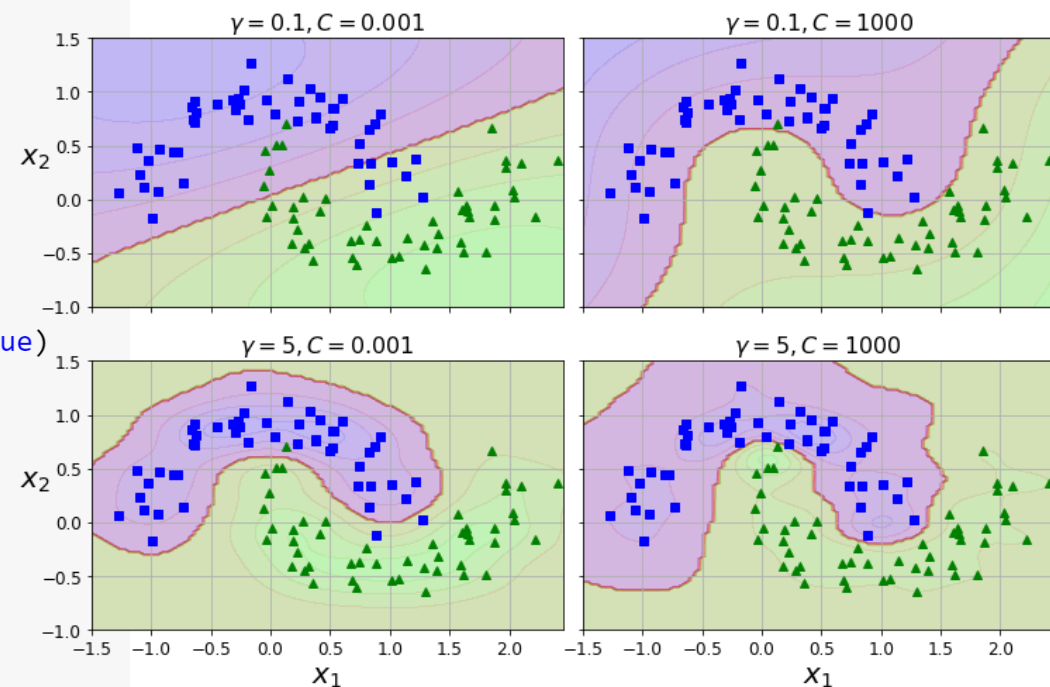
```
gamma1, gamma2 = 0.1, 5
C1, C2 = 0.001, 1000
hyperparams = (gamma1, C1), (gamma1, C2), (gamma2, C1), (gamma2, C2)

svm_clfs = []
for gamma, C in hyperparams:
    rbf_kernel_svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("svm_clf", SVC(kernel="rbf", gamma=gamma, C=C))
    ])
    rbf_kernel_svm_clf.fit(X, y)
    svm_clfs.append(rbf_kernel_svm_clf)

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10.5, 7), sharex=True, sharey=True)

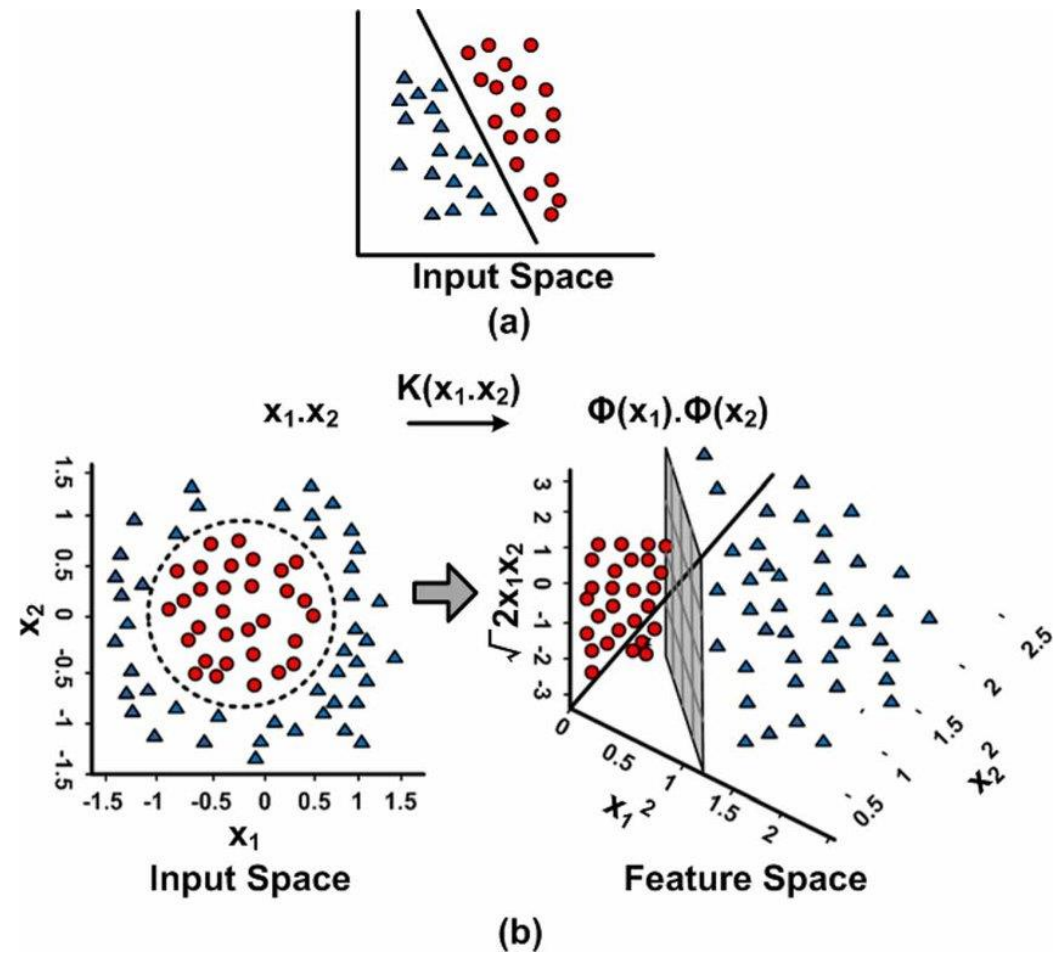
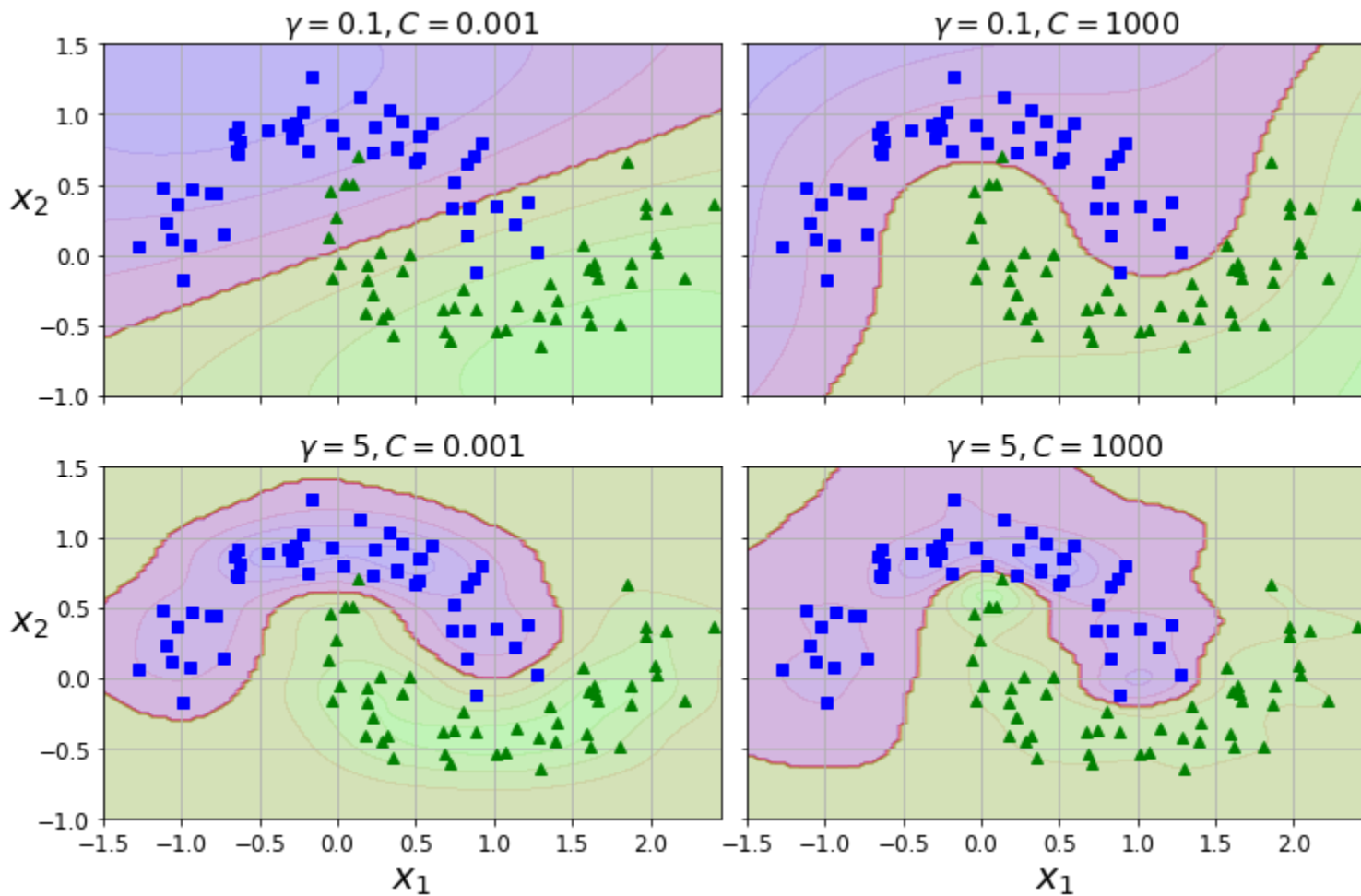
for i, svm_clf in enumerate(svm_clfs):
    plt.sca(axes[i // 2, i % 2])
    plot_predictions(svm_clf, [-1.5, 2.45, -1, 1.5])
    plot_dataset(X, y, [-1.5, 2.45, -1, 1.5])
    gamma, C = hyperparams[i]
    plt.title(r"$\gamma = {}, C = {}".format(gamma, C), fontsize=16)
    if i in (0, 1):
        plt.xlabel("")
    if i in (1, 3):
        plt.ylabel("")

save_fig("moons_rbf_svc_plot")
plt.show()
```



Support Vector Machine

Nonlinear SVM Classification



Support Vector Machine

Example

iris setosa



petal

sepal

iris versicolor



petal

sepal

iris virginica



petal

sepal

Support Vector Machine

Example

We are going to create a model for classifying the the type of iris based on the variables of the dataset.

iris setosa



petal

sepal

iris versicolor



petal

sepal

iris virginica



petal

sepal

Support Vector Machine

Example

In first place, we're going to identifying the variables

Sepal

The sepal is the part that forms the calyx of a flower, typically function as protection for the flower in bud, and often as support for the petals when in bloom.

We have two variables

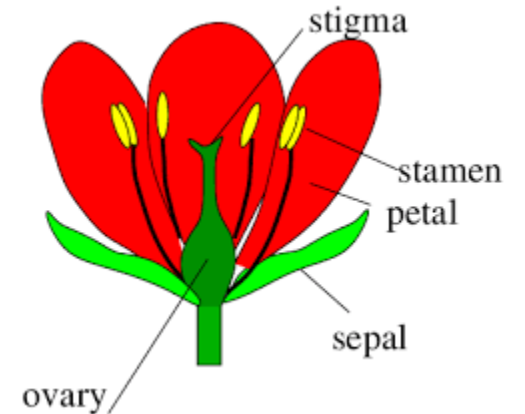
- The Sepal length on centimeters
- The Sepal Width on centimeters

Petal

Petals are modified leaves that surround the reproductive parts of flowers.

We have two variables

- The Petal length on centimeters
- The Petal Width on centimeters



Support Vector Machine

Example

The Types of Flowers

Iris is a genus of 260–300 species of flowering plants with showy flowers. It takes its name from the greek word for a rainbow,Iris.

In the dataset we have three types of iris:

- the Iris Setosa
- Iris Versicolour
- Iris Virginica

Support Vector Machine

Example Data Preparation

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
#Define the col names
colnames=["sepal_length_in_cm", "sepal_width_in_cm","petal_length_in_cm","petal_width_in_cm", "class"]

#Read the dataset
dataset = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data", header = None, names=
colnames )

#Data
dataset.head()
```

	sepal_length_in_cm	sepal_width_in_cm	petal_length_in_cm	petal_width_in_cm	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Support Vector Machine

Example Data Preparation

```
#Encoding the categorical column  
dataset = dataset.replace({"class": {"Iris-setosa":1,"Iris-versicolor":2, "Iris-virginica":3}})  
#Visualize the new dataset  
dataset.head()
```

	sepal_length_in_cm	sepal_width_in_cm	petal_length_in_cm	petal_width_in_cm	class
0	5.1	3.5	1.4	0.2	1
1	4.9	3.0	1.4	0.2	1
2	4.7	3.2	1.3	0.2	1
3	4.6	3.1	1.5	0.2	1
4	5.0	3.6	1.4	0.2	1

Support Vector Machine

Example

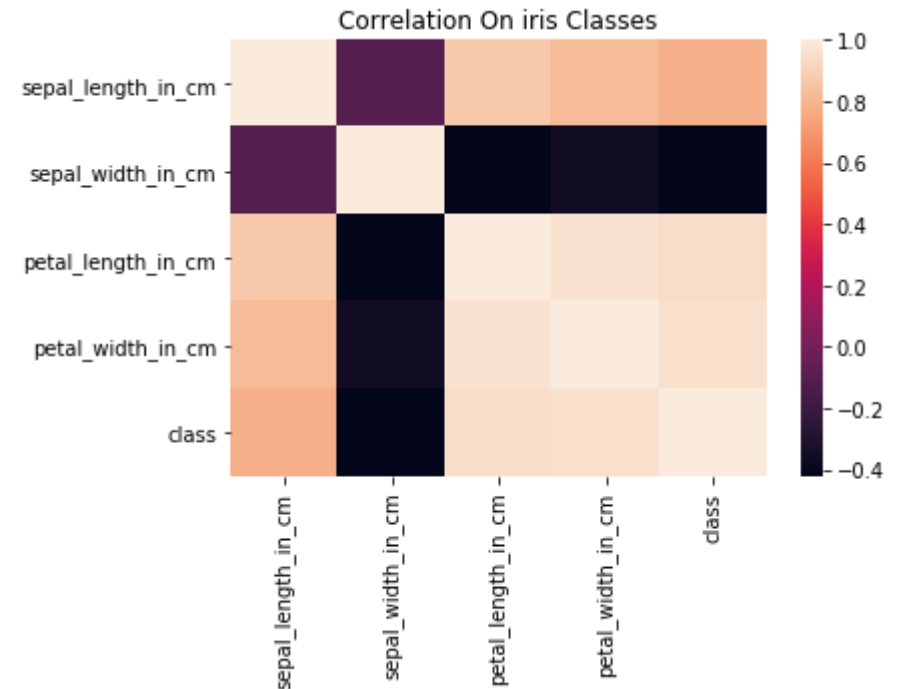
Data Visualization

Now we're going to analyze our data

In first place, all the dataset is organized equally, there is not any type of flower with more data, there are 50 rows for each flower so trying to count any quantity will be unuseful.

So let's look the correlation between the columns, in this way will see how important is a column for choosing wich type of flower.

```
plt.figure(1)
sns.heatmap(dataset.corr())
plt.title('Correlation On iris Classes')
```



Support Vector Machine

Example

Splitting dataset

```
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

Create SVM Model

```
#Create the SVM model
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
#Fit the model for the data

classifier.fit(X_train, y_train)

#Make the prediction
y_pred = classifier.predict(X_test)
```

Support Vector Machine

Example

Splitting dataset

```
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

Create SVM Model

```
#Create the SVM model
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
#Fit the model for the data

classifier.fit(X_train, y_train)

#Make the prediction
y_pred = classifier.predict(X_test)
```

Support Vector Machine

Example

Model Validation

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
Accuracy: 98.18 %
Standard Deviation: 3.64 %
```

		Predicted Values		
		Setosa	Versicolor	Virginica
Actual Values	Setosa	16 (cell 1)	0 (cell 2)	0 (cell 3)
	Versicolor	0 (cell 4)	17 (cell 5)	1 (cell 6)
	Virginica	0 (cell 7)	0 (cell 8)	11 (cell 9)

		True Class		
		A	B	C
Predicted Class	A	TP _A	E _{BA}	E _{CA}
	B	E _{AB}	TP _B	E _{CB}
	C	E _{AC}	E _{BC}	TP _C

Support Vector Machine

When Should I use SVM

Advantages:

- 1.SVM works relatively well when there is a clear margin of separation between classes.
- 2.SVM is more effective in high dimensional spaces.
- 3.SVM is effective in cases where the number of dimensions is greater than the number of samples.
- 4.SVM is relatively memory efficient

Disadvantages:

- 1.SVM algorithm is not suitable for large data sets.
- 2.SVM does not perform very well when the data set has more noise i.e. target classes are overlapping.
- 3.In cases where the number of features for each data point exceeds the number of training data samples, the SVM will underperform.
4. As the support vector classifier works by putting data points, above and below the classifying hyperplane there is no probabilistic explanation for the classification.

Support Vector Machine

Nonlinear 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

Support Vector Machine Regressor

Support Vector Machine

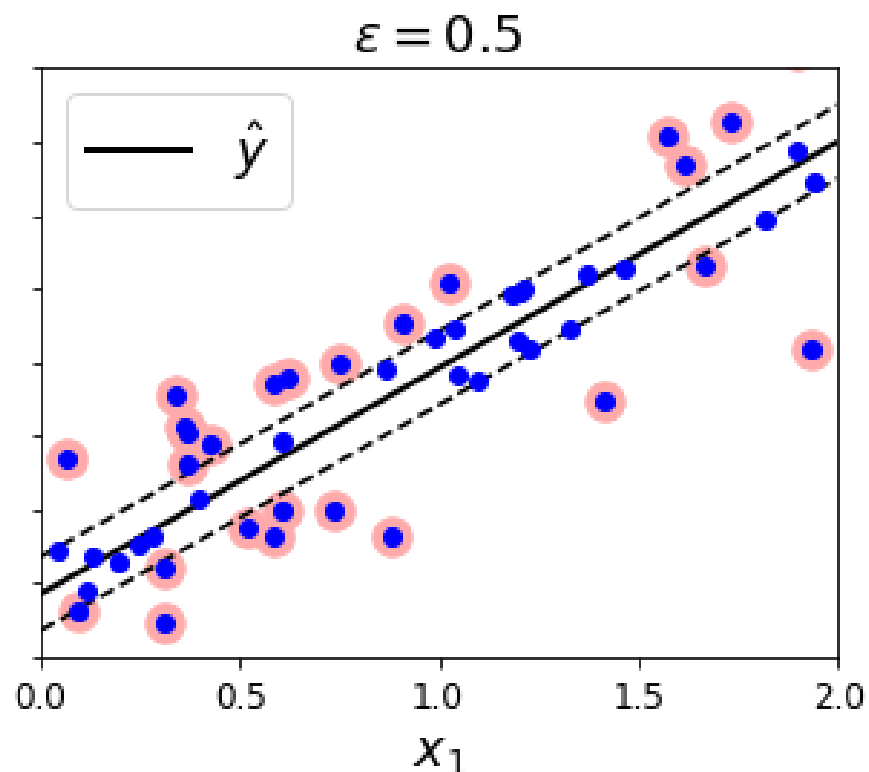
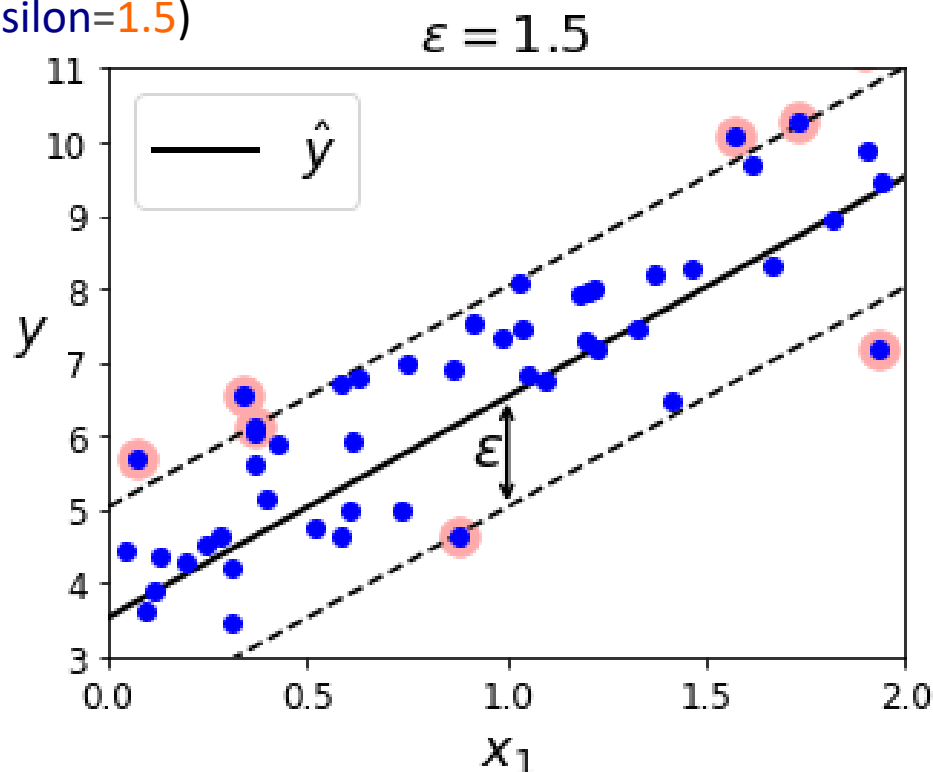
Linear Regression

As we mentioned earlier, the SVM algorithm is quite versatile: not only does it support linear and nonlinear classification, but it also supports linear and nonlinear regression. The trick is to reverse the objective: 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). The width of the street is controlled by a hyperparameter ϵ .

```
from sklearn.svm import LinearSVR
```

```
svm_reg = LinearSVR(epsilon=1.5)
```

```
svm_reg.fit(X, y)
```

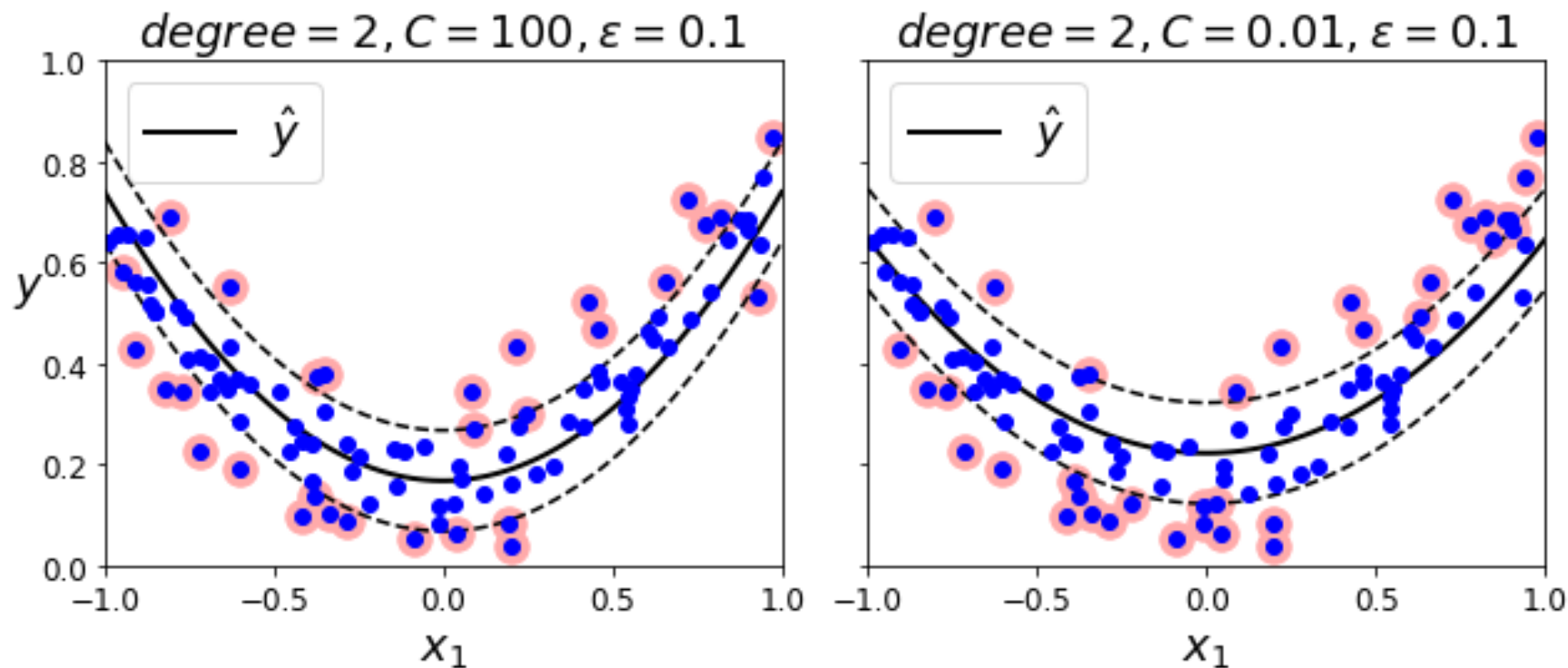


Support Vector Machine

Non-Linear Regression

To tackle nonlinear regression tasks, you can use a kernelized SVM model. For example, Figure shows SVM Regression on a random quadratic training set, using a 2nd-degree polynomial kernel. There is little regularization on the left plot (i.e., a large C value), and much more regularization on the right plot (i.e., a small C value)

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



Support Vector Machine

Non-Linear Regression

To tackle nonlinear regression tasks, you can use a kernelized SVM model. For example, Figure shows SVM Regression on a random quadratic training set, using a 2nd-degree polynomial kernel. There is little regularization on the left plot (i.e., a large C value), and much more regularization on the right plot (i.e., a small C value)

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

