CITS5508 Machine Learning
Lectures for Semester 1, 2021
Lecture Week 4: Book Chapter 5, SVMs

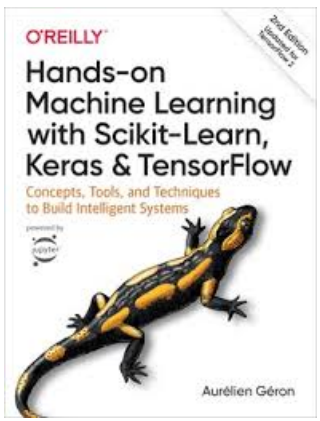
Dr Du Huynh (Unit Coordinator and Lecturer)
UWA

2021

Today

Chapter 5.

Hands-on Machine Learning with Scikit-Learn & TensorFlow



Support Vector Machines

In this chapter, we will explain the core concepts of SVMs, how to use them, and how they work.

Here are the main topics we will go cover:

- Linear SVM Classification
- Nonlinear SVM Classification
- SVM Regression
- Under the Hood

SVMs

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 anyone 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

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

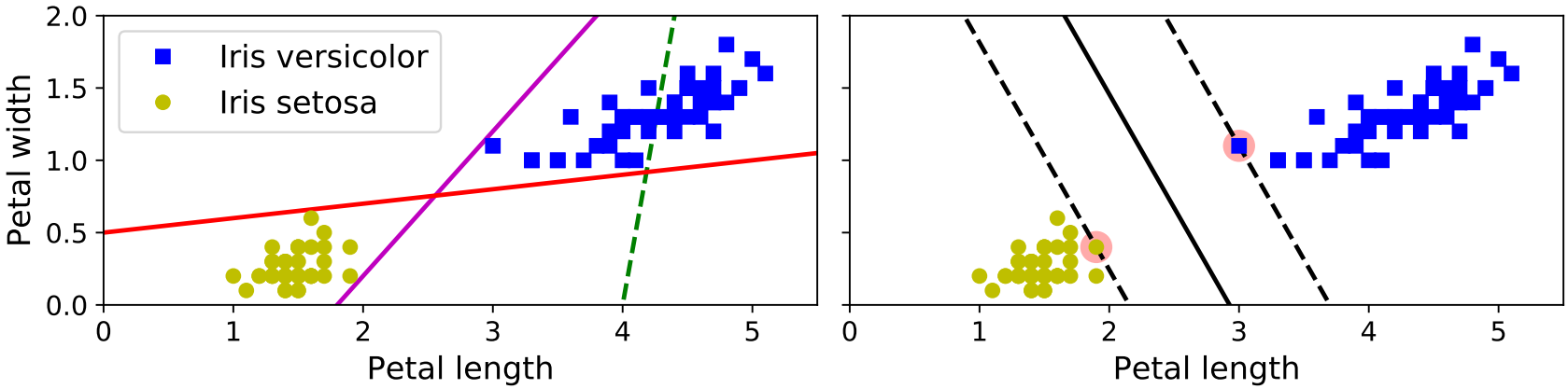


Figure 5-1. Large margin classification

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*.

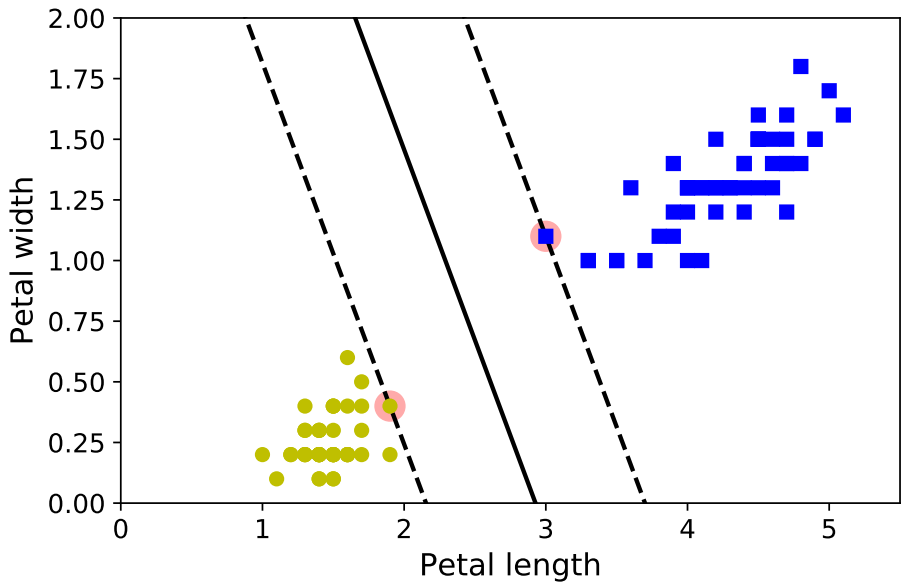


Figure 5-1. Large margin classification

Sensitive to Scale

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).

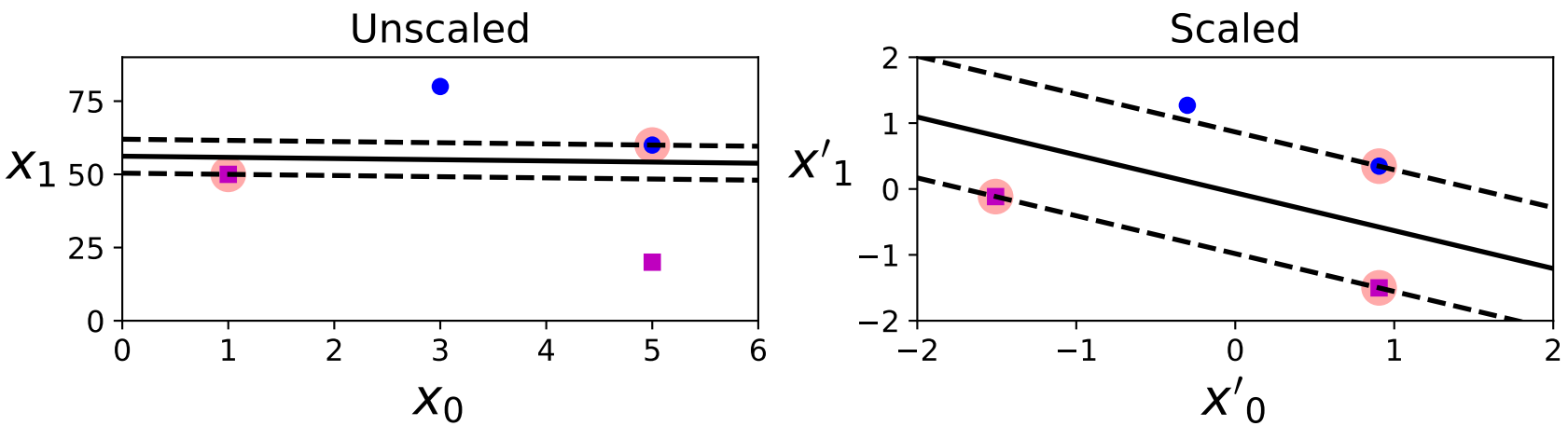
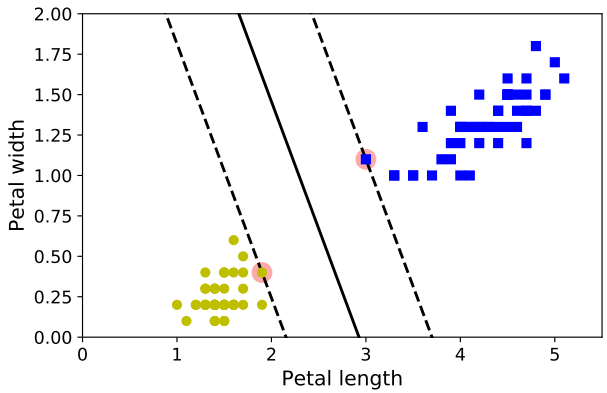


Figure 5-2. Sensitivity to feature scales

Hard Margin Classification

If we strictly impose that all instances be off the street and on the correct side, this is called *hard margin classification*.



The hard margin linear SVM classifier objective can be expressed as a constrained optimzation problem:

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

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

where

$$t^{(i)} = \begin{cases} +1 & \text{if training data } i \text{ is a positive instance} \\ -1 & \text{if training data } i \text{ is a negative instance} \end{cases}$$

Hard Margin Classification (cont.)

- However, *hard margin classification* only works if the data is linearly separable.
- Furthermore, it is quite sensitive to outliers.

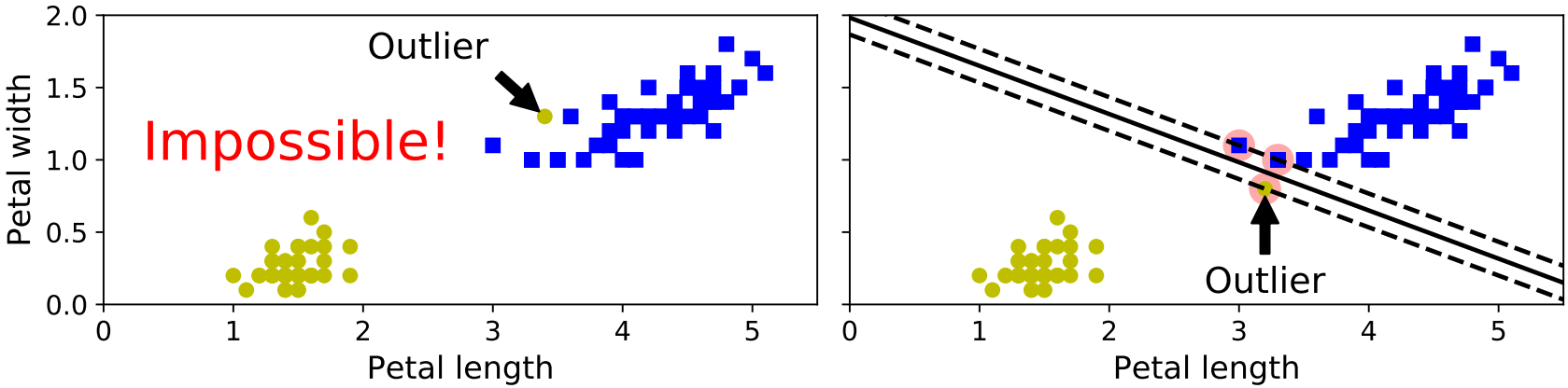


Figure 5-3. Hard margin sensitivity to outliers

Soft Margin Classification

- To avoid the issues mentioned on the previous slide, we can use a more flexible model – soft margin classification.
- The objective of *soft margin classification* is to find a good balance between keeping the street as large as possible and limiting the margin violations:

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

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

$$\text{where} \quad t^{(i)} = \begin{cases} +1 & \text{if training data } i \text{ is a positive instance} \\ -1 & \text{if training data } i \text{ is a negative instance} \end{cases}$$

$\zeta^{(i)}$ is a *slack variable*, which measures how much the i^{th} instance is allowed to violate the margin.

NOTE: C is the hyperparameter which allows us to define the trade-off between the two objectives $\mathbf{w}^\top\mathbf{w}$ and $\sum_{i=1}^{m}\zeta^{(i)}$.

Soft Margin Classification – the C hyperparameter

Use the C hyperparameter to adjust the balance between the two objectives.

- A small C value penalizes less on the points falling onto the wrong side of the street, leading to "wider street" and more *margin violations*.
- A large C value penalizes more on points falling onto the wrong side of the street, leading to "narrower street" and less margin violations.

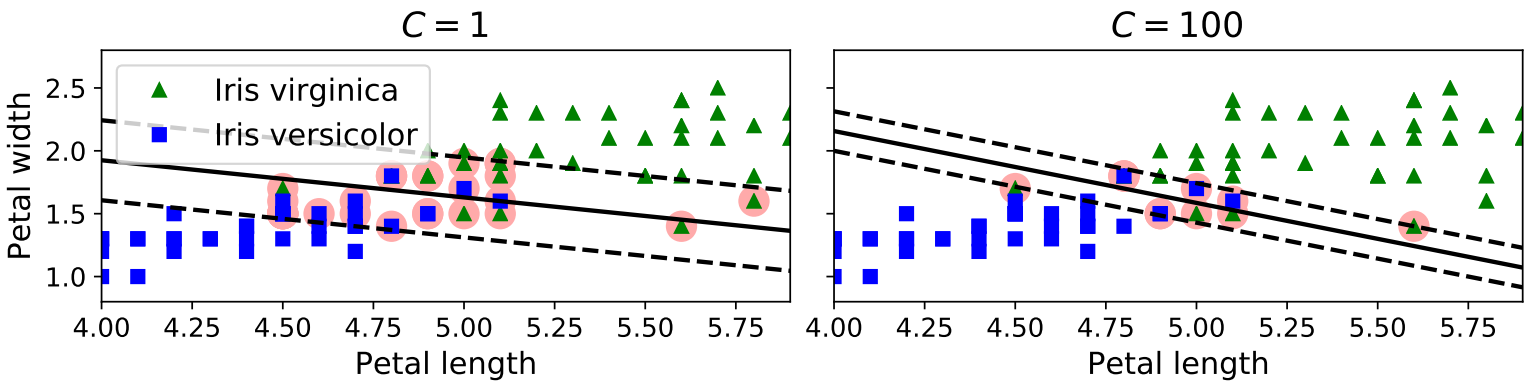


Figure 5-4. Large margin (left) versus fewer margin violations (right)

Soft Margin Classification – An example

```
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

scaler = StandardScaler()
X = scaler.fit_transform(X)      # normalize X

clf = LinearSVC(C=1, loss=hinge) # try a small C value
clf.fit(X,y)                     # train the SVM classifier
```

The *hinge loss* function is a loss function used in maximum-margin classification, typically SVMs. It is defined as $\text{hinge\_loss}(y) = \max(0, 1 - ty)$, where $t = \pm 1$ is the ground truth label and y is the classification score.

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

Nonlinear SVM Classification – using Polynomial Features

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.

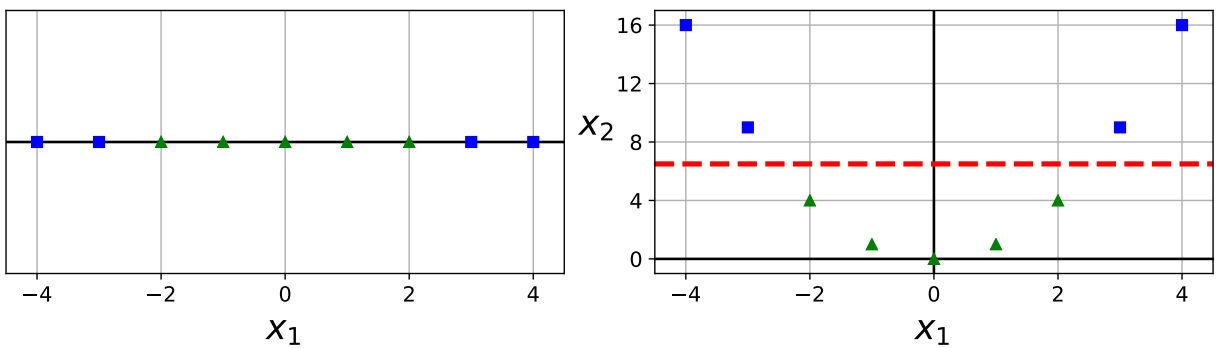


Figure 5-5. Adding features to make a dataset linearly separable

Another example

```
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)
```

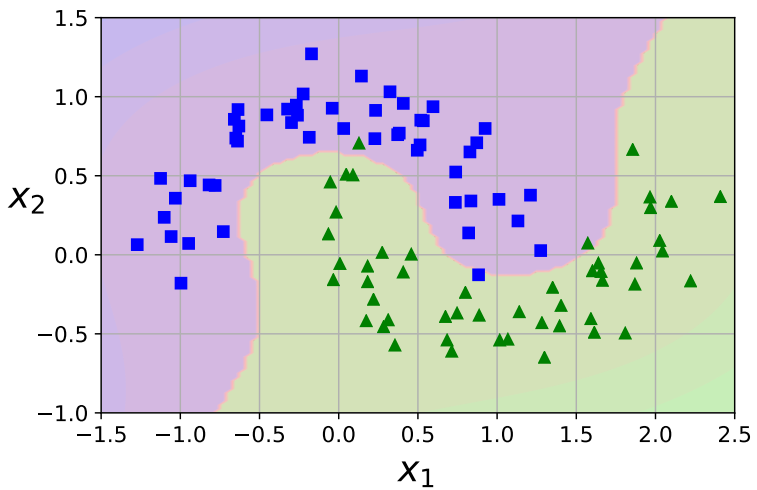


Figure 5-6. Linear SVM classifier using polynomial features

Polynomial Kernel

Using polynomial features can make the model training very slow. Fortunately, when using SVMs you can apply an almost miraculous mathematical technique called the *kernel trick*. 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.

For example, to use a polynomial kernel (of degree 3):

```
SVC(kernel="poly", degree=3, coef0=1, C=5)
```

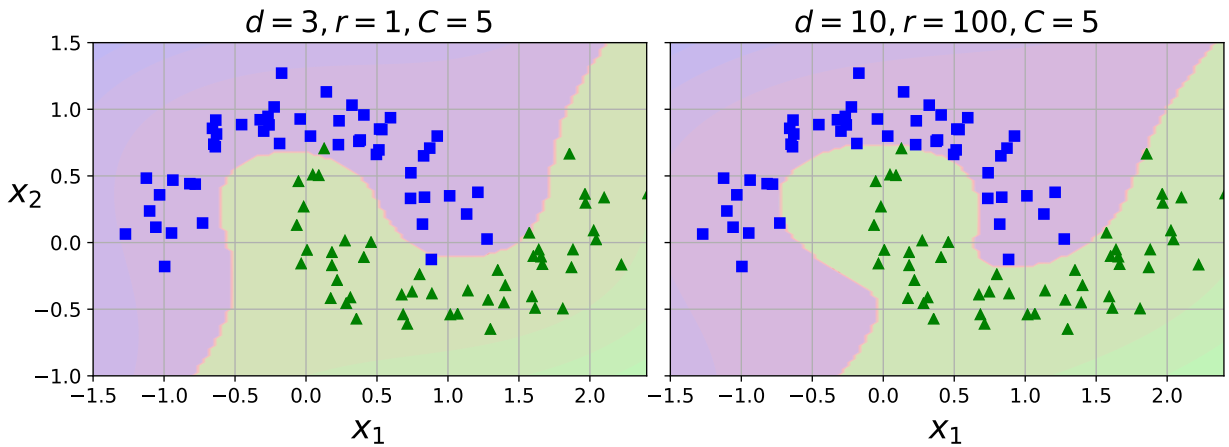


Figure 5-7. SVM classifiers with a polynomial kernel

Adding similarity functions

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. For example, let's take the one-dimensional dataset discussed earlier and add two landmarks to it at $x_1 = -2$ and $x_1 = 1$ (left plot). Next, let's define the similarity function to be the *Gaussian Radial Basis Function* (*RBF*) with $\gamma = 0.3$.

$$\phi_\gamma(x, \ell) = \exp(-\gamma \|x - \ell\|^2)$$

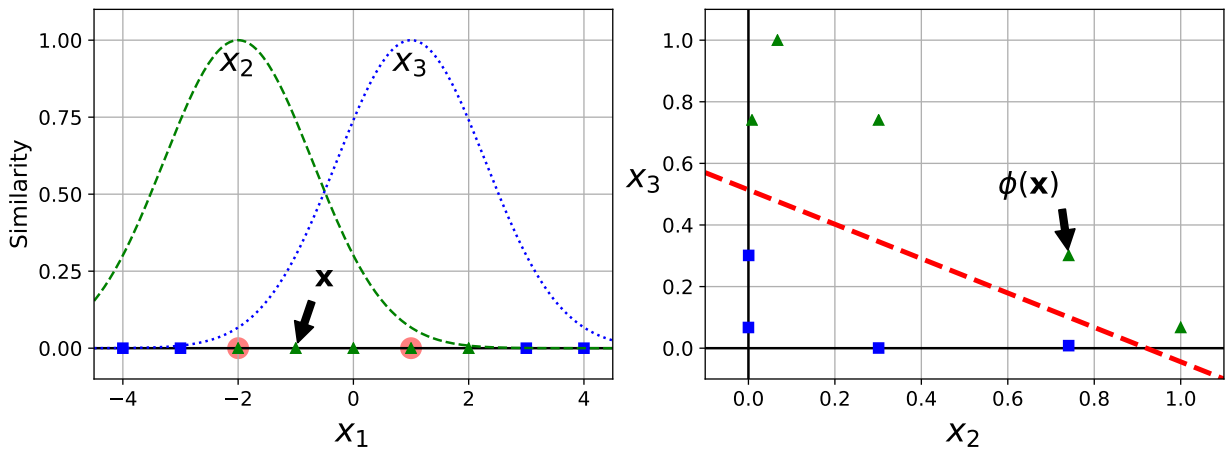


Figure 5-8. Similarity features using the Gaussian RBF

Gaussian RBF Kernel

The kernel trick again makes it possible to obtain a similar result as if you had added many similarity features, without actually having to add them.
Try the Gaussian RBF kernel using the `SVC` class.

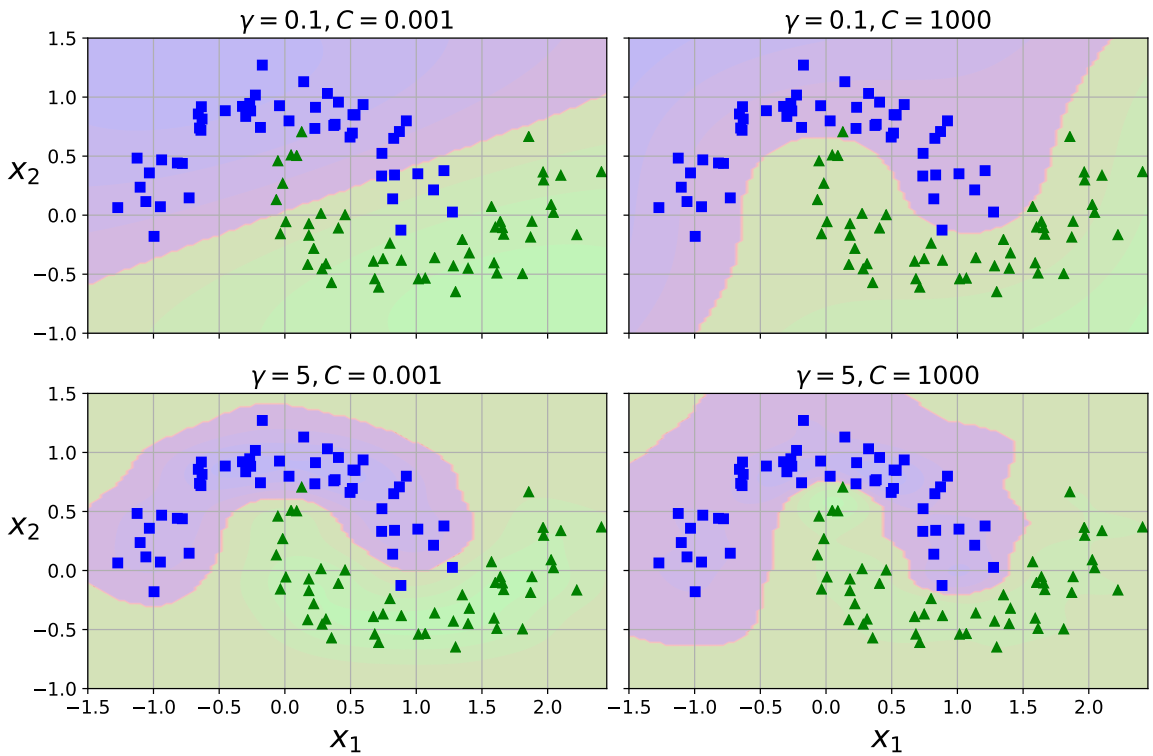


Figure 5-9. SVM classifiers using an RBF kernel

Computational Complexity

- `LinearSVC` implements an optimized algorithm for *linear SVMs*. No kernel trick, but it scales almost linearly with the number of training instances and the number of features.
- `SVC` implements an algorithm that supports the kernel trick. Unfortunately, high training time complexity means that it gets dreadfully slow when the number of training instances gets large.

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).

```
from sklearn.svm import LinearSVR
svm_reg = LinearSVR(epsilon=1.5)
svm_reg.fit(X, y)
```

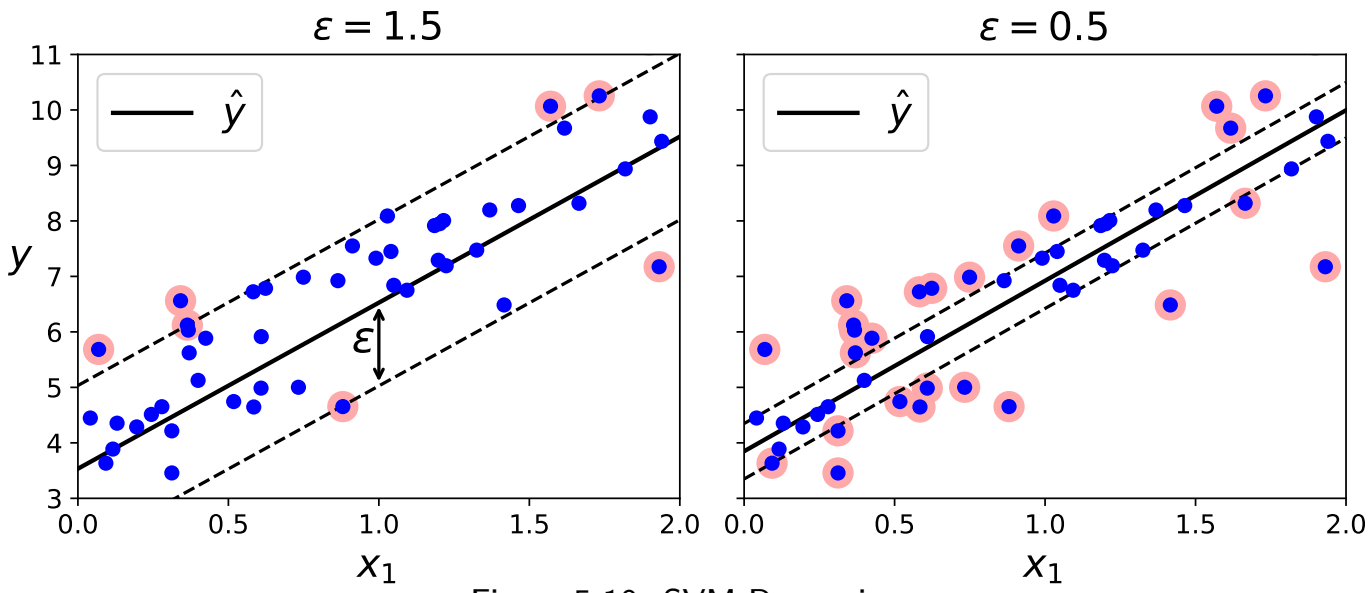


Figure 5-10. SVM Regression

Non-linear Regression

For nonlinear regression use a kernelized SVM model. E.g 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).

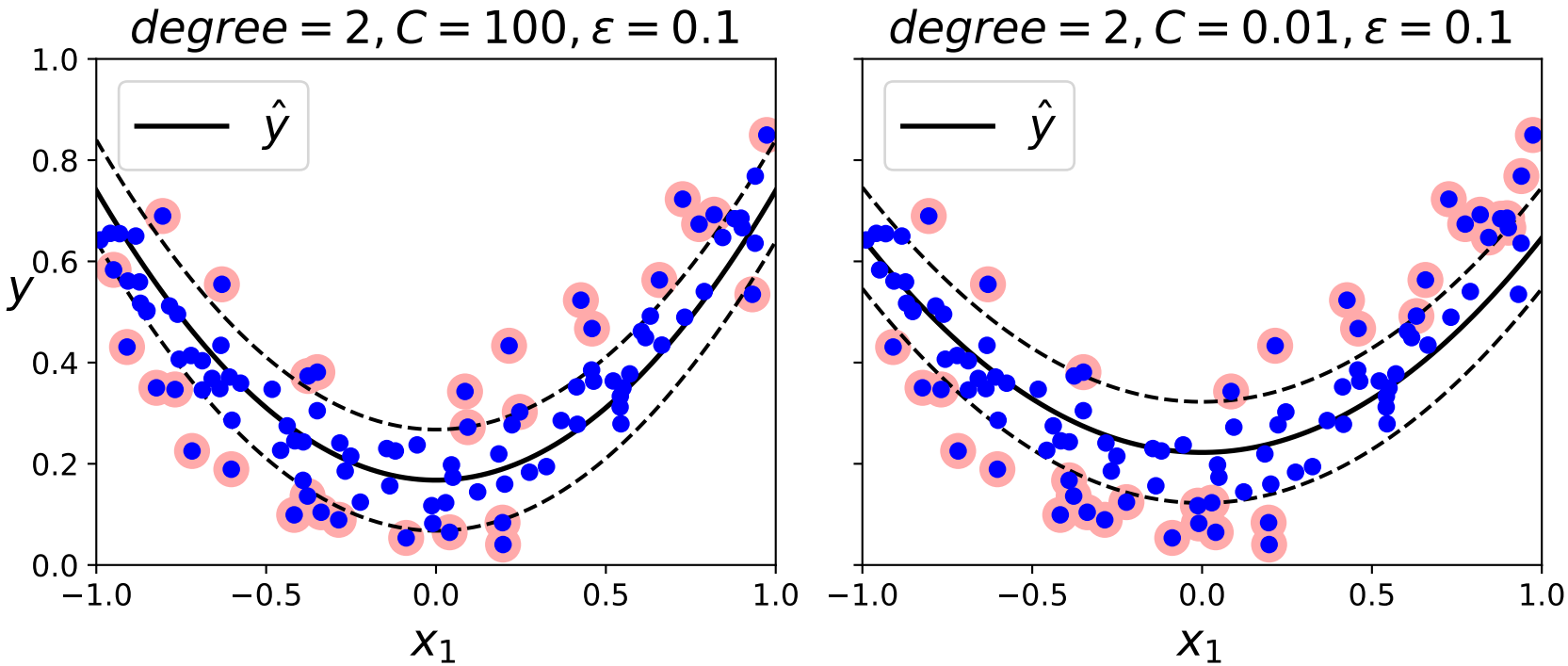


Figure 5-11. SVM Regression using a second-degree polynomial kernel

Under the Hood

Rest of chapter explains how SVMs make predictions and how their training algorithms work.
This has some heavy maths, beyond the scope of the course. We just take a high level overview.

Note: In this chapter, the bias term will be called b and the feature weights vector will be called $\mathbf{w}$. No bias feature ($x_0 = 1$) will be added to the input feature vectors.

Decision Function and Predictions

The linear SVM classifier model predicts the class of a new instance x by simply computing the decision function $\mathbf{w}^T\mathbf{x} + b = w_1x_1 + \cdots + w_nx_n + b$ and predict the class label $\hat{y}$ as follows:

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

Training Objective

We want to choose $\mathbf{w}$ and ζ to

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

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

where the $t^{(i)}$ are ± 1 to indicate positive or negative training instances.

We observe that smaller $\mathbf{w}$ gives wider roads:

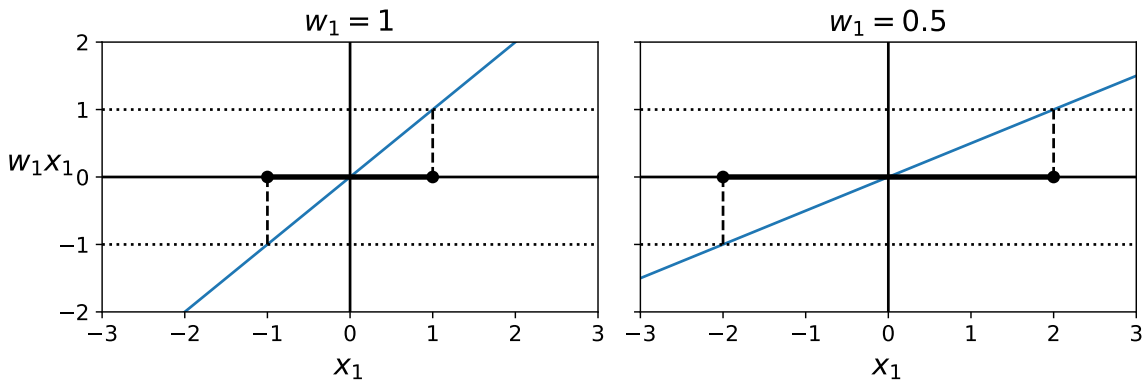


Figure 5-13. A smaller weight vector results in a larger margin

So we minimize $\|\mathbf{w}\|$ to get a larger margin.

Quadratic Programming

The hard margin and soft margin problems are both convex quadratic optimization problems with linear constraints. Such problems are known as *Quadratic Programming (QP) problems*.

Many off-the-shelf solvers are available to solve QP problems using a variety of techniques that are outside the scope of this unit.

Text book shows how to write the hard margin linear SVM classifier problem in QP form, so you can just use an off-the-shelf QP solver. (Soft margin problem is an exercise).

The Dual Problem

Given a constrained optimization problem, known as the *primal problem*, it is possible to express a different but closely related problem, called its *dual problem*.

The dual form of the linear SVM objective is:

$$\min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha^{(i)} \alpha^{(j)} t^{(i)} t^{(j)} \mathbf{x}^{(i)T} \mathbf{x}^{(j)} - \sum_{i=1}^{m} \alpha^{(i)}$$

subject to all $\alpha^{(i)} > 0$, for $i = 1, \cdots, n$.

The dual problem is faster to solve than the primal when the number of training instances is smaller than the number of features. More importantly, it makes the *kernel trick* possible, while the primal does not.

Kernalized SVM, "The Kernel Trick"

Suppose that we want to transform our features $(\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)})$ using the function ϕ (e.g., ϕ can be a 2nd degree polynomial) to a higher dimensional space. We then train a linear SVM classifier on the transformed training set.

Using the dual form on the previous slide, in the new transformed space, it seems that we will need to compute the dot product $\phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$. However, if ϕ is the 2nd-degree polynomial transformation above, then you can simply replace this dot product of the transformed vectors by $(\mathbf{x}^{(i)T} \mathbf{x}^{(j)})^2$.

The function $K(\mathbf{a}, \mathbf{b}) = (\mathbf{a}^T \mathbf{b})^2$ is called a 2nd-degree polynomial kernel. In Machine Learning, a kernel is a function capable of computing the dot product $\phi(\mathbf{a})^T \phi(\mathbf{b})$ based only on the original vectors $\mathbf{a}$ and $\mathbf{b}$, without having to compute (or even to know about) the transformation ϕ.

Common kernels used in SVM

Linear:	$K(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T \mathbf{b}$
Polynomial of degree d:	$K(\mathbf{a}, \mathbf{b}) = (\gamma\, \mathbf{a}^T \mathbf{b} + r)^d$
Gaussian RBF:	$K(\mathbf{a}, \mathbf{b}) = \exp(-\gamma \left\|\mathbf{a} - \mathbf{b}\right\|^2 + r)$
Sigmoid:	$K(\mathbf{a}, \mathbf{b}) = \tanh(\gamma\, \mathbf{a}^\top \mathbf{b} + r)$

where γ, d, and r are hyperparameters.

(The hyperparameter r is the `coef0` parameter in the `SVC` and `SVR` classes of Scikit-learn)

Summary for Chapter 5

- Linear SVM Classification: hard margin vs soft margin
- Nonlinear SVM Classification: polynomial kernels, similarity features, Gaussian RBF, Complexity
- SVM Regression
- Under the Hood: decision function, training objectives, the kernel trick

For next week

Work through the lab sheet and attend the supervised lab. The Unit Coordinator and a casual Teaching Assistant will be there to help most of that time.

Prepare for the Mid-semester test by studying Chapters 1 to 5.

Read Chapter 6 on Decision Trees.

And that's all for the fourth lecture.

Have a good week.