第九章支持向量机

支持向量机 (support vector machine) 是一类上世纪 90 年代发展起来的分类器，被认为是最好的“开箱即用”的分类器之一

本章讨论的概念涉及**最大间隔分类器 (maximal margin clssifier)**(不能用于大数据集)、**支持向量分类器 (support vector classifier)** 和支持向量机及其扩展。人们通常把上面三类分类器统称为支持向量机，本章对它们作出了详细区分

9.1 最大间隔分类器

9.1.1 超平面

在 p 维空间中，一个**超平面 (hyperplane)** 是一个维度为 $p-1$ 的平坦仿射子空间。当 $p=2$ 时，超平面是一条直线；$p=3$ 时，超平面是一个平面；而当 $p>3$ 时，超平面很难可视化

二维空间中的超平面可以这样定义：

$$\beta_0+\beta_1X_1+\beta_2X_2=0$$

也就是说，所有 $X=(X_1,X_2)^T$ 满足上面方程的点都在这个超平面上 (实际上是一条直线)。上面的式子可以扩展为一个 p 维的超平面：

$$\beta_0+\sum_{i=1}^{p}\beta_iX_i=0$$

对于不满足超平面的点，例如：

$$\beta_0+\sum_{i=1}^{p}\beta_iX_i>0$$

表示点在超平面的一侧，而：

$$\beta_0+\sum_{i=1}^{p}\beta_iX_i<0$$

则表示点在超平面的另一侧。通过比较不等符号，我们可以区分点在超平面的哪一侧

> 超平面的特性是，可以将对应的空间分为两个不相交的子空间

9.1.2 用分隔超平面分类

对于一个 $n\times p$ 的数据矩阵 $\mathbf{X}$，它包含 n 个 p 维向量作为观测值：

$$x_1=\begin{pmatrix}x_{11}\\\vdots\\x_{1p}\end{pmatrix},\cdots,x_n=\begin{pmatrix}x_{n1}\\\vdots\\x_{np}\end{pmatrix}$$

另有一个测试数据 $x^* = (x_1^*, \cdots, x_p^*)^T$。现在，我们将用一种基于**分隔超平面 (separating hyperplane)** 的方法对数据进行分类

一个分隔超平面满足下面的特性。对于分别被标记为 1 和-1 的两类，分隔超平面有：

$$\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} > 0 \quad if\ y_i = 1$$

和：

$$\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} < 0 \quad if\ y_i = -1$$

等价的，在上面的分类条件下，一个分隔超平面满足：

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) > 0$$

对于任意 $i = 1, 2, \cdots, n$ 成立

假设我们能找到这样的一个超平面，则可以根据：

$$f(x^*) = \beta_0 + \sum_{i=1}^{p} \beta_i x_i^*$$

的正负来判断它在超平面的哪个方向，且这个函数值的大小可以判断它距离超平面的距离。显然，超平面分类器会导致一个线性的分类边界

9.1.3 最大间隔分类器

如果数据能用超平面完美分开，那么实际上存在无穷个这样的超平面

为了选择唯一的超平面，一个自然的方法是**最大化间隔超平面 (maximal margin hyperplane)**，也被称为**最佳分隔超平面 (optimal separating hyperplane)**。称训练数据上的点到超平面的垂直距离为**间隔 (margin)**，则最大化间隔超平面会最大化所有训练数据到其的间隔，使得分类的准确性最大化。这样的分类器就是**最大化间隔分类器 (maximal margin classifier)**。然而，当 p 过大时，最大化间隔超平面容易导致过拟合

> 某种意义上来说，最大化间隔就是我们能够插入的最小超平面 (刚好分开某类数据的超平面) 的中线

对于位于分割超平面间隔附近的数据点 (即最小分隔超平面内的点)，我们称这些观测值为**支持向量 (support vetors)**，他们位于的超平面被称为**间隔边界 (margin)**。这是因为它们固定了最大化间隔超平面，如果它们略微移动，则这个超平面也会跟着移动 (其他的点则不会)

> 这个分类的一个特点是，最大间隔超平面直接依赖于观测值的一小部分子集

9.1.4 建立最大间隔分类器

建立最大化间隔分类器实际上是一个优化问题：

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,M}{\text{maximize}}\ M \tag{1}$$

$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1 \tag{2}$$

$$y_i(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}) \geq M,\ \forall i = 1, \ldots, n \tag{3}$$

这个式子没有它看起来那么困难。首先，对于式 (3)：

$$y_i(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}) \geq M,\ \forall i = 1, \ldots, n$$

它保证了所有的数据点都位于超平面的某侧，存在一定的缓冲距离

而式子 (2) 则为 (3) 添加了意义，在此约束条件下，第 i 哥观测值到超平面的距离就是：

$$y_i(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij})$$

最后，式子 (1) 要求最大化这个平面到所有点的距离

> 优化问题的细节涉及拉格朗日乘数法，本书未提及

9.1.5 不可分情况

许多问题中的观测值不能找到可分的超平面，也就是说，不存在一个式子，能够满足所有的解都有 $M > 0$

不可分情况下的最大化间隔分类器不可用。后面的模型将使用**软间隔 (soft margin)** 来进行几乎可分的分类。这种分类器就是支持向量分类器

9.2 支持向量分类器

9.2.1 支持向量分类器简介

最大间隔分类器存在如下的问题：

1. 观测数据不一定线性可分
2. 对单个数据变化敏感，容易发生过拟合

本章开发的分类器有如下改进：

1. 对少量观测值的更高稳健性
2. 对大多数观测值的更好分类效果

为了实现这些改进，对于少量的观测值，误分类是值得的

支持向量分类器，又称为**软间隔分类器 (soft margin classifier)**，能顾实现这些要求。对于部分数据，它可以存在于间隔的错误一侧，也可以存在于超平面的错误一侧

> 这就是说数据可能靠近边界但未被正确分类，也可能完全错误分类 (远离超平面)

9.2.2 支持向量分类器的细节

为了实现上一节的改进，我们对式子进行如下修改：

$$\underset{\beta_0,\ldots,\beta_p,\epsilon_0,\ldots,\epsilon_n,M}{\text{maximize}} M \tag{1}$$

$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1 \tag{2}$$

$$y_i(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}) \geq M(1-\epsilon_i) \tag{3}$$

$$\epsilon_i \geq 0, \quad , \sum_{i=1}^{n} \epsilon_i \leq C \tag{4}$$

其中 C 是一个非负调谐参数；ϵ_i 是一个非负的**松弛变量 (slack variable)**，允许数被误分类 ($\epsilon_i = 0$ 时，表示 x_i 被正确分类，且到超平面距离至少为 M)。当上式被确定下来后，就可用于分类观测数据 x^* 了

松弛变量告诉我们第 i 个观测值位于何处，$\epsilon_i > 0$ 则说明它被误分类了。参数 C 用于控制我们对误分类的容忍程度，实践中常用交叉检验选择

优化式子 $(1) \sim (4)$ 指出，严格位于正确分类的边缘一侧的数据不影响支持向量分类器，位于超平面上，或者被错误分类的观测值 (被称为支持向量) 才会对支持向量分类器产生影响

> 这意味着支持向量分类器对远离超平面的观测值的行为具有相当的稳健性

9.3 支持向量机

9.3.1 非线性决策边界的分类

支持向量分类器无法简单地处理非线性决策边界。为了解决这个问题，我们考虑使用：

- 添加交互项，如 X_1X_2
- 引入高次特征，如 X_1^2

来扩展特征空间，并处理非线性决策边界。对于引入二次特征，我们的约束函数组变为：

$$\begin{aligned}
&\underset{\beta_0,\beta_{11},\beta_{12},\ldots,\beta_{1p},\beta_{2p},\epsilon_1,\ldots,\epsilon_n,M}{\text{maximize}} M \\
&\text{subject to} y_i \left(\beta_0 + \sum_{j=1}^{p} \beta_{j1} x_{ij} + \sum_{j=1}^{p} \beta_{j2} x_{ij}^2 \right) \geq M(1-\epsilon_i) \\
&\sum_{i=1}^{n} \epsilon_i \leq C, \epsilon_i \geq 0, \ , \sum_{j=1}^{p} \sum_{k=1}^{2} \beta_{jk}^2 = 1
\end{aligned}$$

当我们将这些扩展后的特征映射回原来的函数中时，高次项和交互项会导致决策边界的非线性

然而，当 p 很大时，引入过多的高次项和交互项使计算变得异常复杂。下面提到的支持向量机通过核函数技术，可以在不显示扩展特征空间的情况下，高效处理非线性问题

9.3.2 支持向量机

支持向量机使用**核函数 (kernels)** 进行特征空间的拓展化，其细节具有相当的技术性

首先，我们引入内积的概念：

$$< x_i, x_{i'} >= \sum_{j=1}^{p} x_{ij} x_{i'j}$$

内积就是两个向量的点积。则线性支持向量分类器的一个输出函数则可以表述为：

$$f(x) = \beta_0 + \sum_{i=1}^{p} \alpha_i < x, x_i >$$

其中参数 α_i 对应每个训练数据向量和观测向量之间的内积。为了估计 α_i 和 β_0，我们需要计算 C_n^2 对向量 $< x_i, x_{i'} >$ 之间的内积

数学上证明，为了计算 $f(x)$，我们计算点 x 和训练数据 x_i 之间的内积有如下性质：若 x_i 不是支持向量，则它的观测 $\alpha_i = 0$。这些系数 α_i 非 0 的点就是支持向量。我们设支持向量集合为 S，则输出函数可以变为：

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i < x, x_i >$$

这比之前的形式减少了很多项

现在，我们将式中的内积替换为广义内积：

$$K(x_i, x_{i'})$$

其中 K 就是核函数。核函数以一种刻画两个观测值之间相似性的函数，例如一种核函数：

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

这被称为线性核，与原来的支持向量分类器等价，它使用皮尔逊相关系数来刻画两个向量之间的相关性。我们可以替换为另一种核：

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d$$

这被称为**多项式核 (polynomial kernel)**，其中 d 是一个正整数。使用这个 $d > 1$ 的核，我们可以刻画非常灵活的决策边界，即通过把特征转换到高维空间中

当支持向量分类器使用非线性核时，它就成为了所谓的支持向量机，此时它的输出函数具有如下形式：

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i K(x, x_i)$$

一种流行的核叫做**径向核 (radial kernel)**，它的形式如下：

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2)$$

其中 γ 是一个正常数。这个核使用指数衰减的形式来刻画一个训练数据对预测数据的影响，当二者的欧氏距离很近时，这个影响很大，否则很小。这个特性使得它能捕捉数据的局部结构

使用核函数相比于简单的原始特征扩大的好处在于：

- 不显示地扩大特征空间，这意味这我们不需要计算和存储特征的高维表示
- 对于 n 个训练样本，我们计算的核函数值 C_n^2 相比于许多高维度的数据扩展来说已经很小，尤其是某些扩展后维度为无限的数据

9.4 多分类下的 SVM

目前我们的讨论仅局限于二分类。虽然 SVM 基于的分离超平面的概念并不自然的适用于两个以上的类别，但是目前存在一些扩展分类的方法

9.4.1 一对一分类

设 $K > 2$ 个类别，我们构造 C_n^2 个 SVM 来对这个数量的类别进行**一对一 (one-versus-one)** 分类，最终以最多被分类的类为最终分类。这个方法有些繁琐

9.4.2

一对全 (one-versuis-all) 方法，也称为**一对余 (one-versus-rest)** 方法，一共拟合 K 个 SVM 模型，每个模型将一个类别 (编码为+1) 和其他所有类别 (编码为-1) 进行比较。令 $\beta_{0k}, \beta_{1k}, \cdots, \beta_{pk}$ 表示第 k 类 (+1) 与其他类 (-1) 拟合 SVM 得到的参数，设实验值 x^*，则输出函数 $f(x^*) = \beta_{0k} + \beta_{1k} x_1^* + \cdots + \beta_{pk} x_p^*$ 最大的分类为最终分类

> 这种方法在类别不平衡时表现不佳

9.5 SVM 与逻辑斯蒂回归的关系

支持向量分类器的输出函数可以重写为：

$$\underset{\beta_0, \beta_1, \ldots, \beta_p}{minimize} \left\{ \sum_{i=1}^{n} \max[0, 1 - y_i f(x_i)] + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$$

其中 λ 是一个非负调谐参数，当 λ 足够大时可以容纳更高的偏差。其中，小 λ 值与支持向量分类器中的 C 值相等。注意，$\lambda \sum_{j=1}^{p} \beta_j^2$ 是我们之前提到过的正则化项

这种损失+乘法的组合重复出现了多次：

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{minimize}\{L(\mathbf{X},\mathbf{y},\beta)+\lambda P(\beta)\}$$

其中 L 是用于刻画模型与训练数据的损失函数，量化了以 β 为参数的模型对数据 (X,y) 的拟合程度，$\lambda P(\beta)$ 是对这组参数的惩罚项

> 岭回归和 lasso 回归都采用了上面的形式

对于支持向量分类器的损失函数，我们可以写为：

$$L(\mathbf{X},\mathbf{y},\beta)=\sum_{i=1}^{n}\max[0,1-y_i(\beta_0+\sum_{j=1}^{p}\beta_j x_{ij})]$$

这也被称为**合页损失 (hinge loss)**。合页损失与逻辑斯蒂回归使用的损失函数非常近似

支持向量分类器的特性是位于分类边缘的正确分类数据对模型没有影响，损失函数为 0；而逻辑斯蒂回归的损失函数在任何地方都不为 0，即是远离决策边界的观测值影响非常小。因此二者常给出相似的分类结果

> 当类间分类较好时，支持向量机的效果好于逻辑回归；在更多重叠的区域中，逻辑回归更受青睐

SVM 的超参数 C 在它出现初期被认为不重要，但是现在已经证明它用于控制模型的偏差-方差权衡，有必要通过交叉检验等方式进行验证

SVM 并不以使用核函数扩大特征维度来显得特殊，事实上，我们之前介绍的许多方法都可以用核函数。只是由于历史原因，SVM 使用非线性核的情况广泛的多

支持向量回归 (support vector regression) 是 SVM 在回归领域中的扩展，它寻求使不同类型损失最小化的系数，只有绝对值大于某个常数的残差对损失函数有贡献

9.6 实验: 支持向量机

本章数据包：

```
import numpy as np
from matplotlib.pyplot import subplots, cm
import sklearn.model_selection as skm
from ISLP import load_data, confusion_table
from sklearn.svm import SVC
from ISLP.svm import plot as plot_svm
from sklearn.metrics import RocCurveDisplay
```

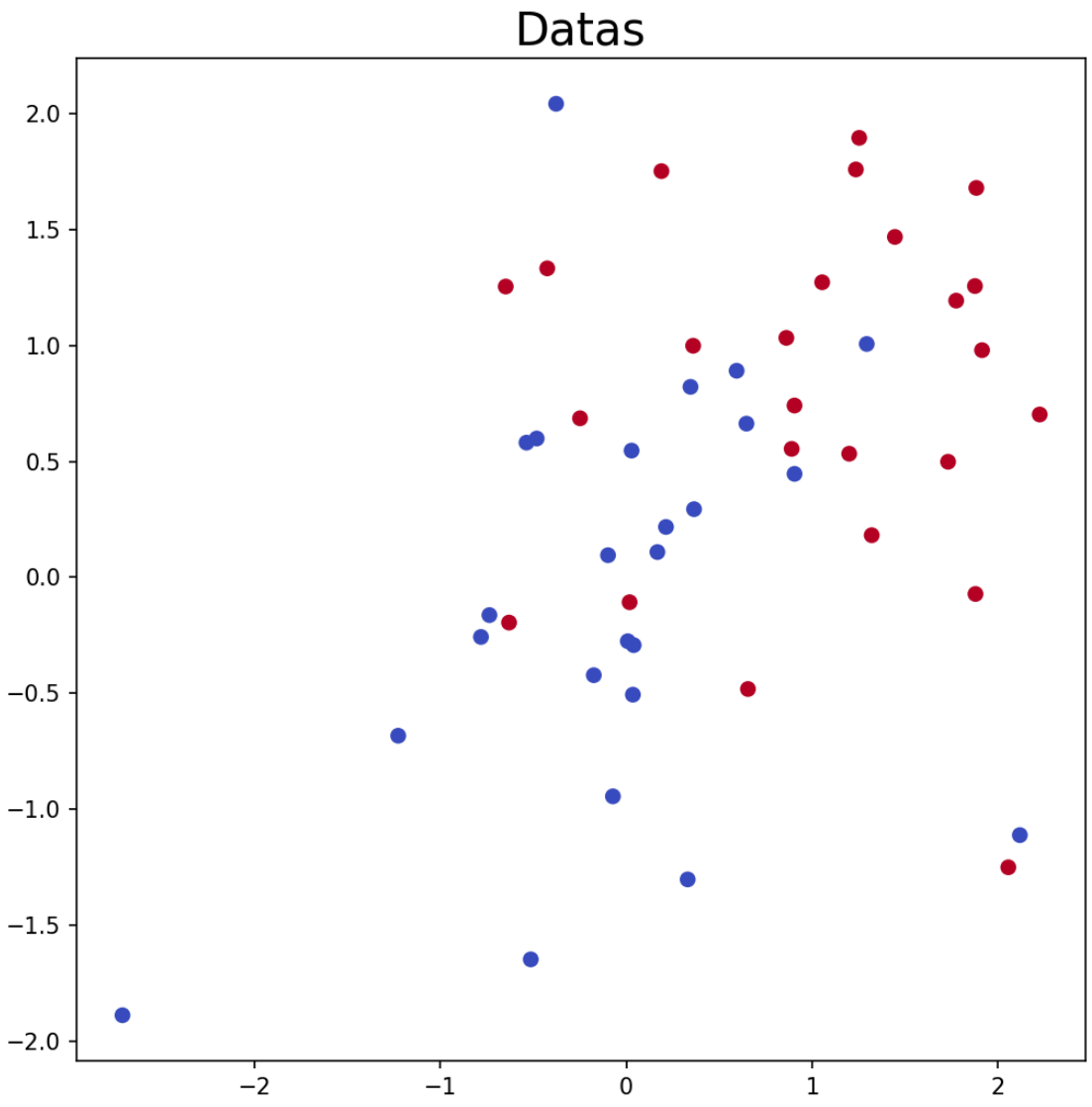
其中 `RocCurveDisplay.from_estimator()` 缩写为 `rov_curve`

9.6.1 支持向量分类器

我们使用 `skl` 包的 `SupportVectorClassifier()` 函数来拟合支持向量分类器，其中需要的超参数是 `C`

下面构造一个线性不可分的数据：

```
#生成随机数据
rng = np.random.default_rng(1)
X = rng.standard_normal((50, 2)) #二维数组
y = np.array([-1]*25 + [1]*25) #前25个值为-1，后25个值为1的一维数组
X[y==1] += 1 #类别为1的值加一，以便更好分类
fig, ax = plt.subplots(figsize=(8, 8))
ax.scatter(X[:, 0],
           X[:, 1],
           c=y,
           cmap=plt.cm.coolwarm)
ax.set_title("Datas", fontsize=20)
```



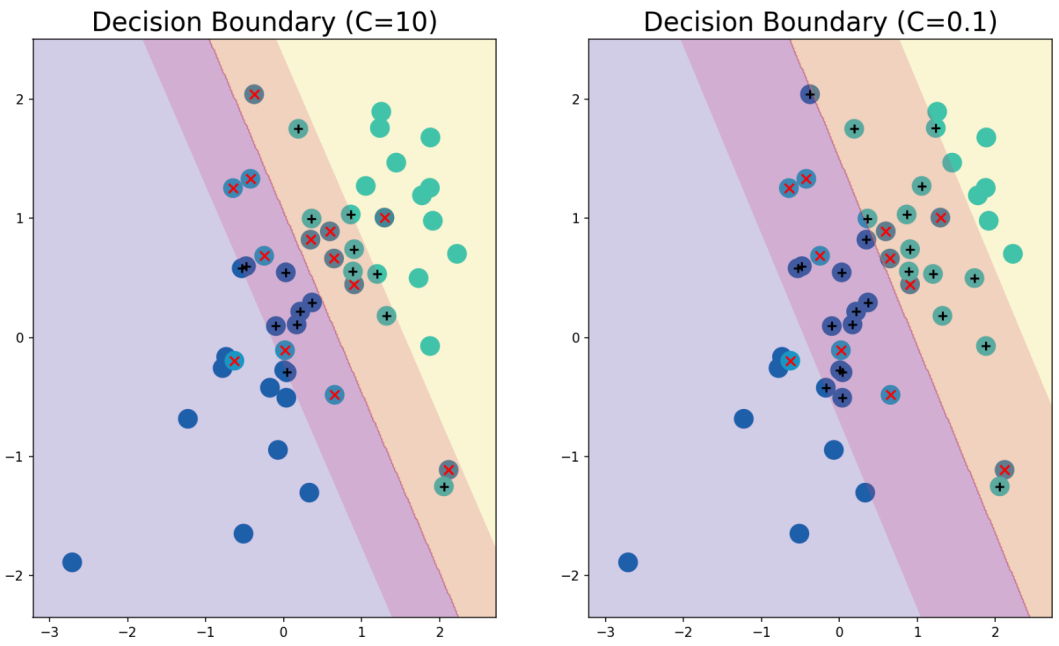
容易从图像发现其线性不可分。现在拟合一个线性SVM模型，指定一个较大的`C`：

```
#生成SVC分类器
svm_linear = SVC(C=10, kernel='linear')
svm_linear.fit(X, y)

#显示决策边界
fig, axes = plt.subplots(1, 2, figsize=(14, 8))
ax = axes[0]
plot_svm(X,
         y,
         svm_linear,
         ax=ax)
ax.set_title('Decision Boundary (C=10)', fontsize=20)
```

`ISLP` 包中的 `plot_svm` 可用于绘制超平面和决策边界，同时显示间隔边界和支持向量

现在更改 `C` 的大小，我们发现当 `C` 越大，则间隔边界越窄：

```
#更改C大小，重新显示决策边界
svm_linear_small = SVC(C=0.1, kernel='linear')
svm_linear_small.fit(X, y)
ax = axes[1]
plot_svm(X,
         y,
         svm_linear_small,
         ax=ax)
ax.set_title('Decision Boundary (C=0.1)', fontsize=20)

#显示线性决策边界
print(svm_linear.coef_)
```



用交叉检验调优参数 C：

```
#用交叉检验寻找最优C
kfold = skm.KFold(5,
                  random_state=0,
                  shuffle=True)
grid = skm.GridSearchCV(svm_linear,
                        {'C': [0.001, 0.01, 0.1, 1, 5, 10, 100]}, #待调优参数
取值列表
                        refit=True,
                        cv=kfold,
                        scoring='accuracy')
grid.fit(X, y)
print(grid.best_params_) #输出最优参数
print(grid.cv_results_[('mean_test_score')]) #输出测试得分
```

上面，我们用到了 `grid.cv_results_` 来显示网格中的参数列表对应模型的测试得分

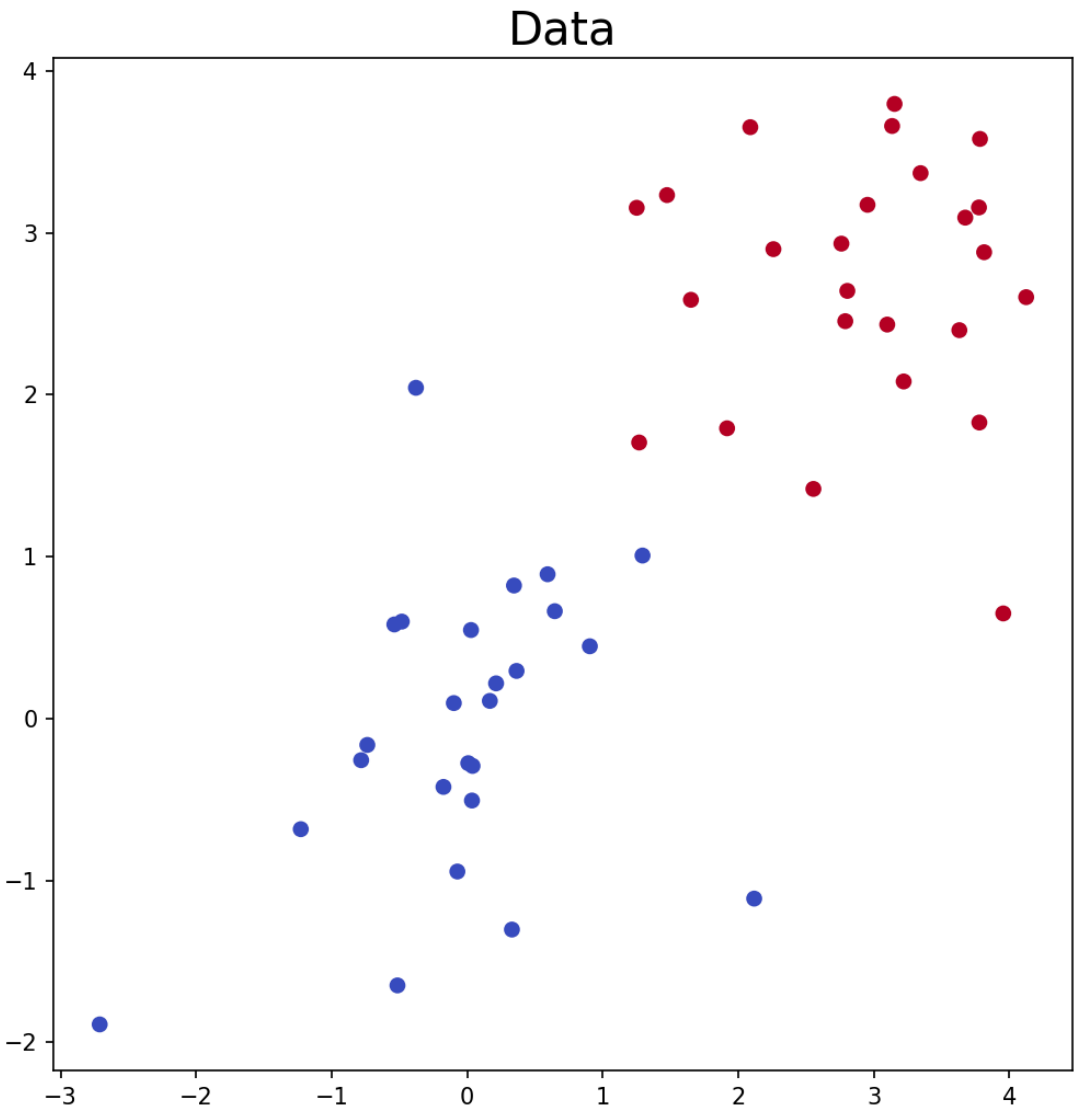
现在用预测集来测试模型，并显示混淆矩阵：

```
#生成测试集，用最优预测器来预测
X_test = rng.standard_normal((20, 2))
y_test = np.array([-1]*10+[1]*10)
X_test[y_test==1] += 1
best_ = grid.best_estimator_ #获取最优预测模型
y_test_hat = best_.predict(X_test)
print(confusion_table(y_test_hat, y_test)) #输出混淆矩阵

#尝试使用C=0.001的预测结果
svm_ = SVC(C=0.001 , kernel='linear').fit(X, y)
```

```
y_test_hat = svm_.predict(X_test)
print(confusion_table(y_test_hat, y_test))
```

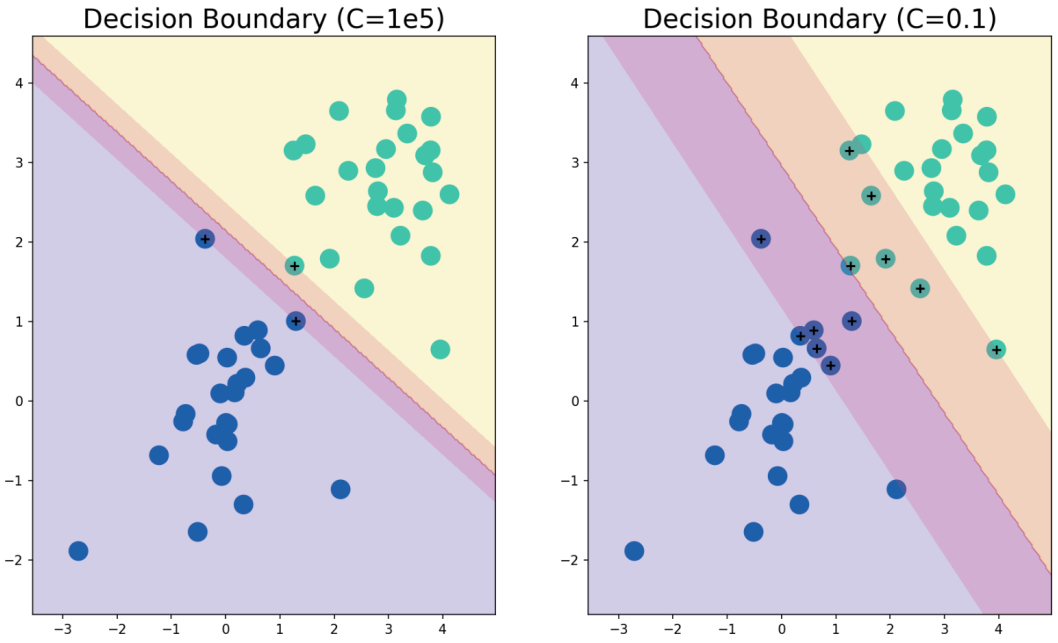
我们尝试构造线性可分的数据集，并用线性SVM来预测：

```
#生成线性可分数据
X[y==1] += 1.9
fig, ax = plt.subplots(figsize=(8,8))
ax.scatter(X[:,0], X[:,1], c=y, cmap=plt.cm.coolwarm)
ax.set_title('Data', fontsize=20)

#拟合线性模型
svm_ = SVC(C=1e5, kernel='linear').fit(X, y) #一个很大的C
y_hat = svm_.predict(X)
print(confusion_table(y_hat, y))

#绘出分类图像
fig, axes = plt.subplots(1, 2, figsize=(14, 8))
ax = axes[0]
plot_svm(X,
         y,
         svm_,
         ax=ax)
ax.set_title('Decision Boundary (C=1e5)', fontsize=20)
```



改变 C 的 大小，此时决策边界更宽了，但是分类效果不变。实际上，用更大的测试集可以发现，更宽决策边界的模型稳健性较好：

```
#改用一个更小的C
svm_ = SVC(C=0.1, kernel='linear').fit(X, y)
y_hat = svm_.predict(X)
print(confusion_table(y_hat, y))
ax = axes[1]
plot_svm(X,
         y,
         svm_,
         ax=ax)
ax.set_title('Decision Boundary (C=0.1)', fontsize=20)
plt.show()
```



9.6.2 支持向量机

改变 SVC() 的核为非线性核就能实现支持向量机。对于多项式核，我们指定 `kernel="poly"`，对于径向核则是 `kernel='rbf'`；前者使用 `degree` 参数指定幂次，后者使用 `gamma` 参数指定γ超参数

构建一个二分类数据集，其中数据的分布不能找到线性超平面分割：

```
#生成非线性边界数据
rng = np.random.default_rng(1)
X = rng.standard_normal((200, 2))
X[:100] += 2
X[100:150] -= 2
y = np.array([1]*150+[2]*50)
fig, ax = plt.subplots(figsize=(8, 8))
ax.scatter(X[:, 0],
           X[:, 1],
           c=y,
           cmap=plt.cm.coolwarm)

#划分数据集
(X_train,
 X_test,
 y_train,
 y_test) = skm.train_test_split(X, y, test_size=0.5, random_state=0)
```

拟合一个径向核SVM：

```
#拟合径向核SVM
svm_rbf = SVC(kernel="rbf", gamma=1, C=1)
svm_rbf.fit(X_train, y_train)
fig, ax = plt.subplots(figsize=(8, 8))
plot_svm(X_train,
         y_train,
         svm_rbf,
         ax=ax)
ax.set_title("RBF kernel SVM (C=1)", fontsize=20)
```

当我们尝试一个极大的`C`时，模型会出现非常不规则的决策边界，这意味着过拟合：

```
#尝试更大的C, 显示不均匀的决策边界
svm_rbf = SVC(kernel="rbf", gamma=1, C=1e5)
svm_rbf.fit(X_train, y_train)
fig, ax = plt.subplots(figsize=(8,8))
plot_svm(X_train,
         y_train,
         svm_rbf,
         ax=ax)
ax.set_title('RBF kernel SVM (C=1e5)', fontsize=20)
plt.show()
```

交叉检验和网格搜索能为我们调优`C`和`gamma`两个参数：

```
#用交叉检验选择最优的C
kfold = skm.KFold(5,
                  random_state=0,
                  shuffle=True)
grid = skm.GridSearchCV(svm_rbf,
                        {'C':[0.1,1,10,100,1000],
                         'gamma':[0.5,1,2,3,4]}, #同时调优两组参数
                        refit=True,
                        cv=kfold,
                        scoring='accuracy')
grid.fit(X_train, y_train)
print(grid.best_params_) #显示最优参数

#显示最优模型的混淆曲线
best_svm = grid.best_estimator_
fig, ax = plt.subplots(figsize=(8,8))
plot_svm(X_train,
         y_train,
         best_svm,
         ax=ax)
```

```
y_hat_test = best_svm.predict(X_test)
print(confusion_table(y_hat_test, y_test))
```

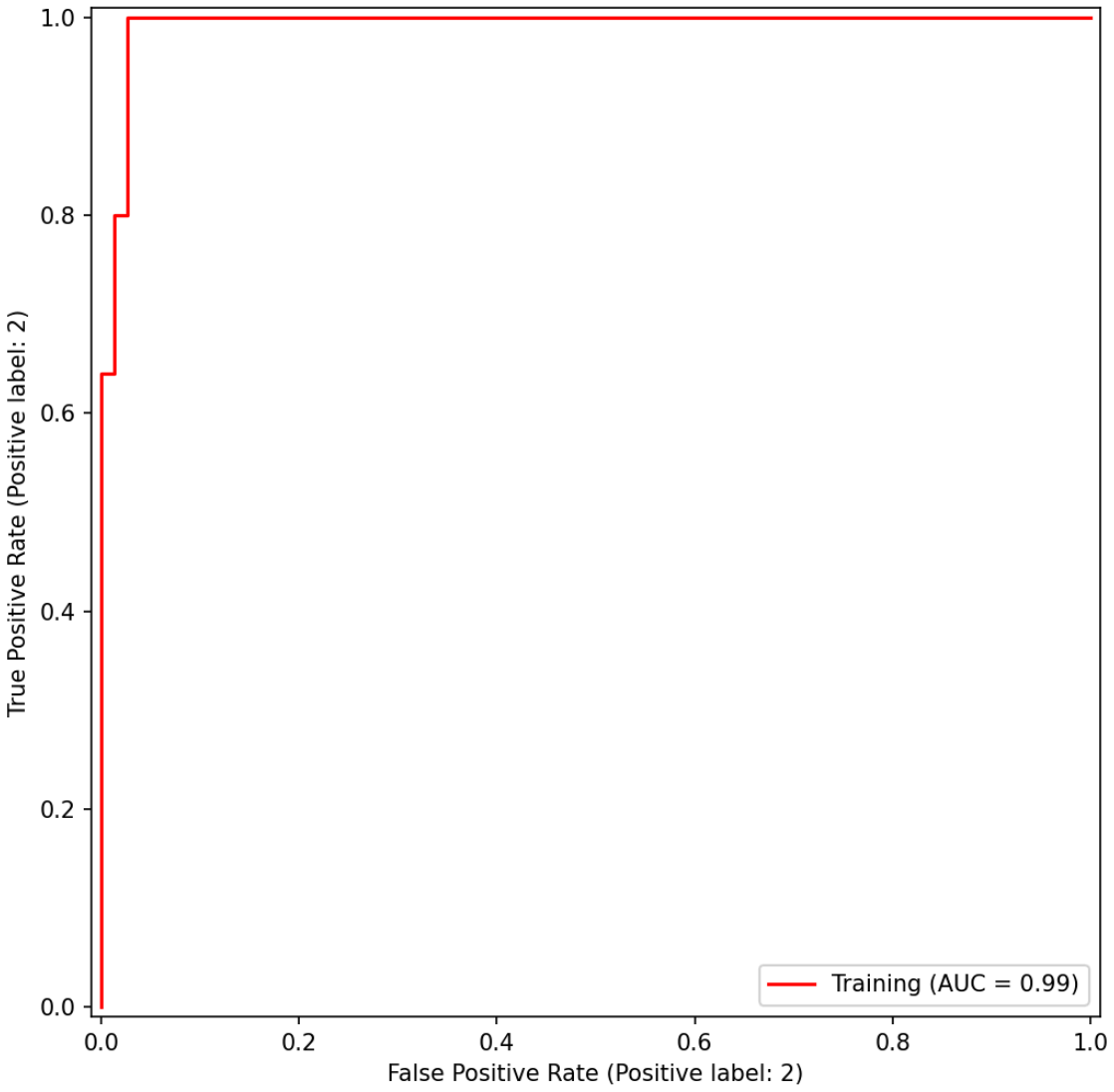
混淆矩阵的结果表明仅有12%的测试数据被误分类

9.6.3 ROC曲线

SVM除了直接对观测数据分类之外，还可以计算观测值的分类得分，这是通过输出函数$f(\mathbf{X})$实现的。默认情况下，若$f(\mathbf{X})$的值小于0，则分类为一类；若$f(\mathbf{X})$的值大于0，则分类为另一类。通过调整阈值0为其他的正数或负数，我们可以改变分类结果。一系列的改变可以绘制ROC曲线，我们使用`decision_function()`来获取这些得分

我们使用之前创建的对象`roc_curve`展示ROC曲线：

```
#绘制ROC曲线
fig, ax = plt.subplots(figsize=(8,8))
roc_curve(best_svm,
          X_train,
          y_train,
          name='Training',
          color='r',
          ax=ax)
```

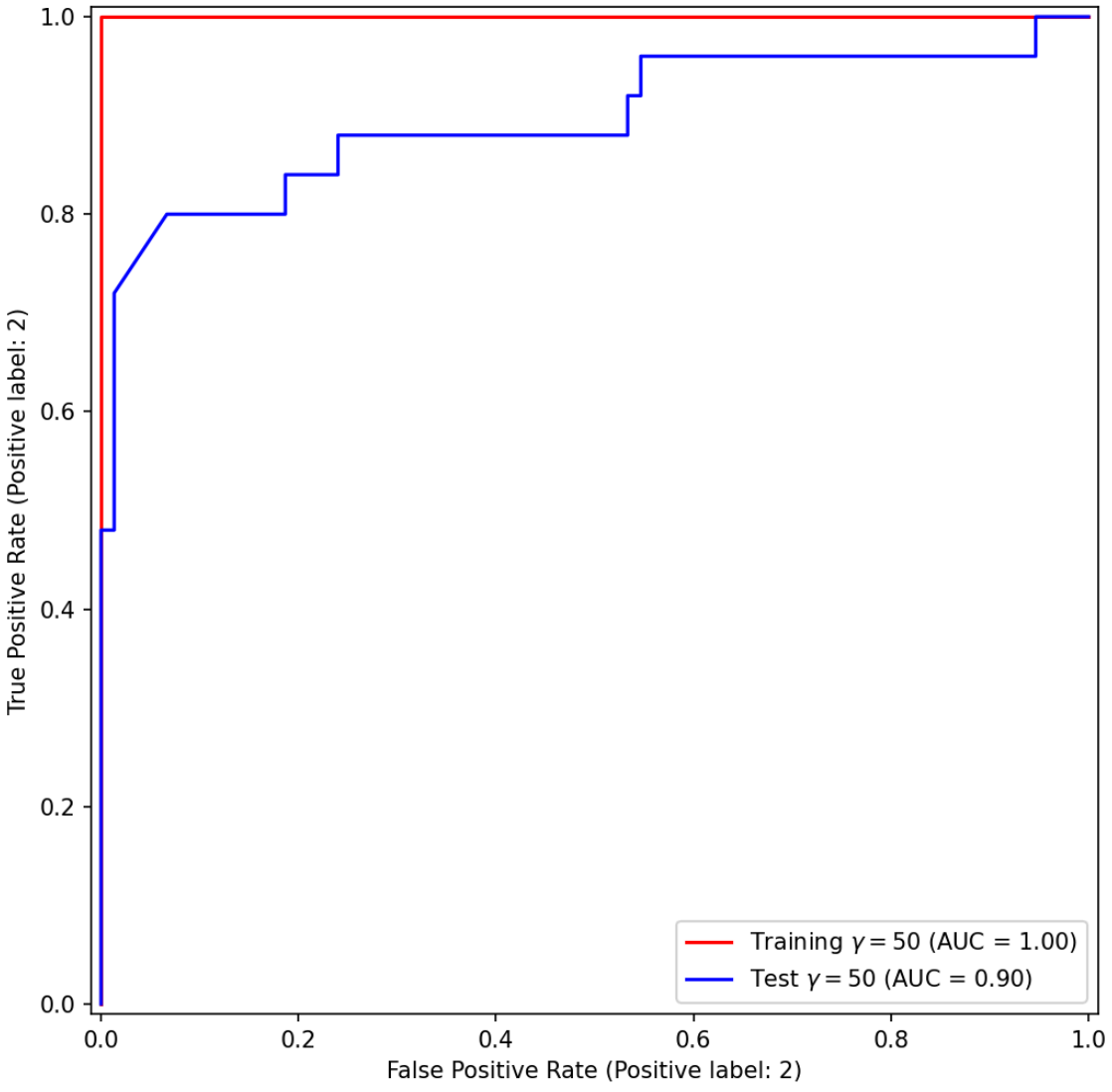


尝试一个新的γ值，获得更灵活模型：

```
#尝试改变gamma获得更灵活的模型
svm_flex = SVC(kernel="rbf",
               gamma=50,
               C=1)
svm_flex.fit(X_train, y_train)
fig, ax = plt.subplots(figsize=(8,8))
roc_curve(svm_flex,
          X_train,
          y_train,
          name='Training $\gamma=50$',
          color='r',
          ax=ax)
```

使用测试集数据，获得更加真实的ROC曲线：

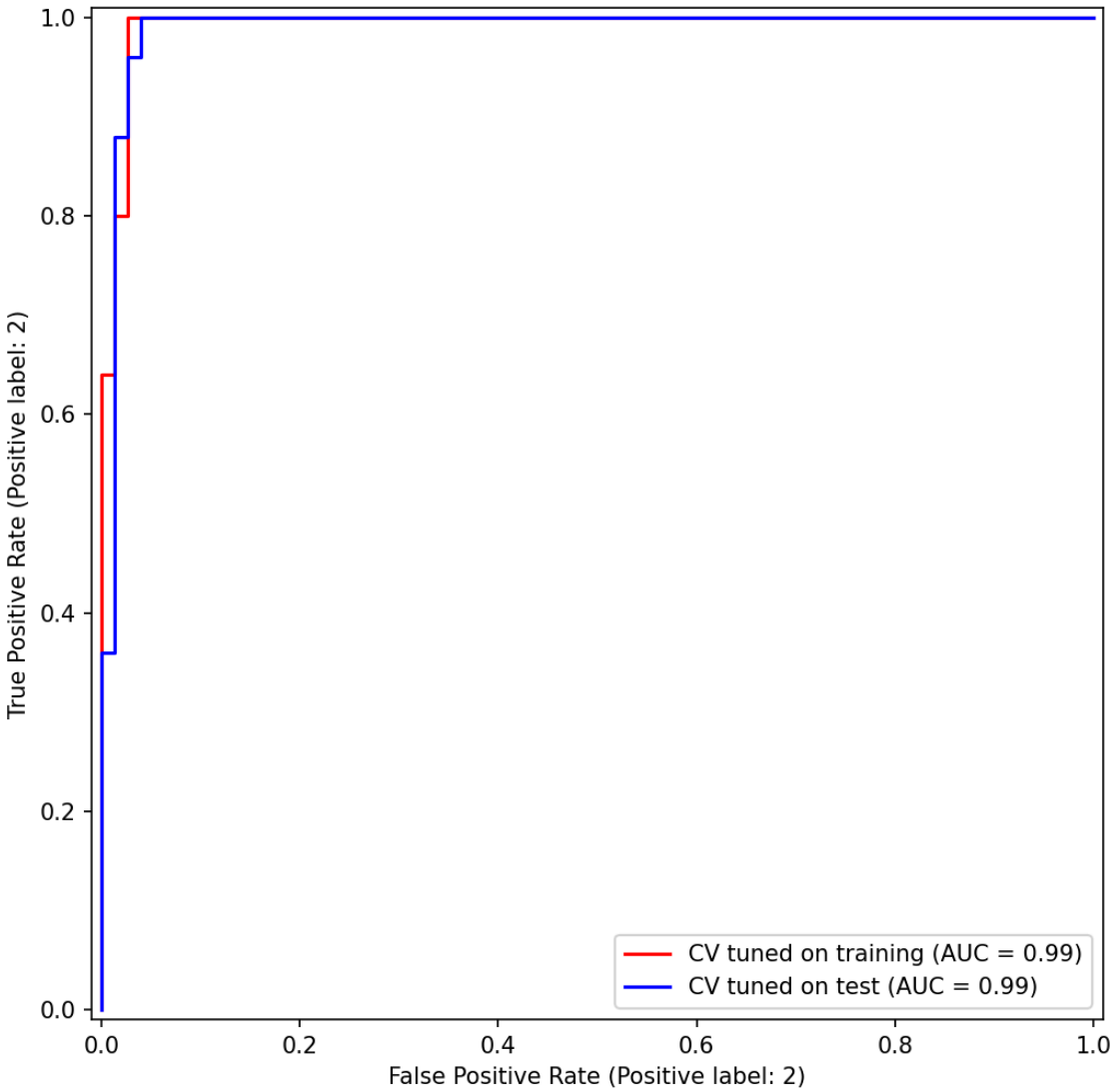
```
#使用预测值
roc_curve(svm_flex,
          X_test,
          y_test,
          name='Test $\gamma=50$',
          color='b',
          ax=ax)
plt.show()
```



使用调优后的模型：

```
#使用调优的模型
fig, ax = plt.subplots(figsize=(8,8))
for (X_, y_, c, name) in zip(
     (X_train, X_test),
     (y_train, y_test),
     ('r', 'b'),
```

```
      ('CV tuned on training',
       'CV tuned on test')):
    roc_curve(best_svm,
              X_,
              y_,
              name=name,
              ax=ax,
              color=c)
plt.show()
```

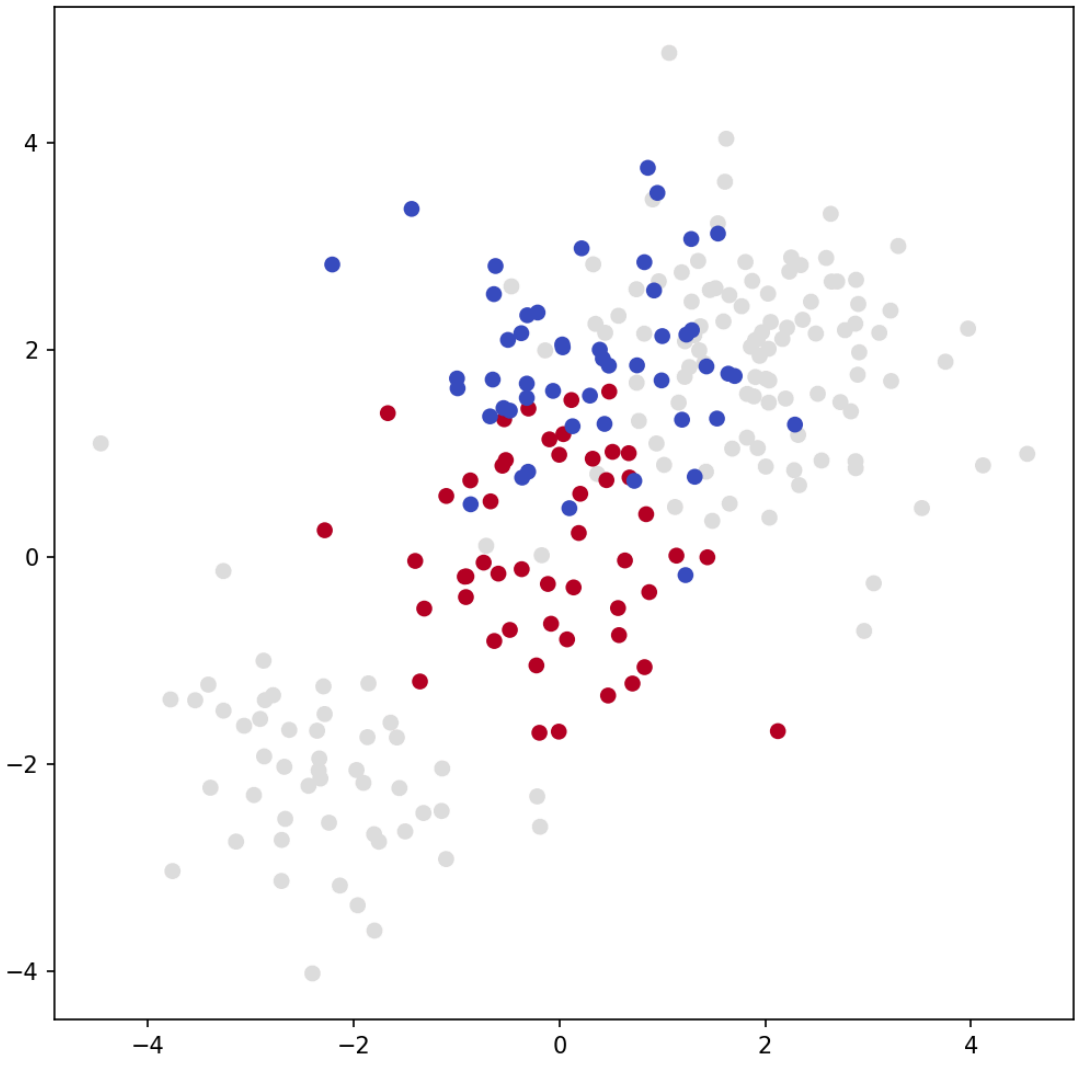


9.6.4 多分类SVM

对于具有多个分类的y，`SVC()`函数会通过指定`decision_function_shape`参数来决定多分类方案，其中`'ovo'`表示一对一方案，而`'ove'`表示一对多方案

我们向之前的数据添加一类：

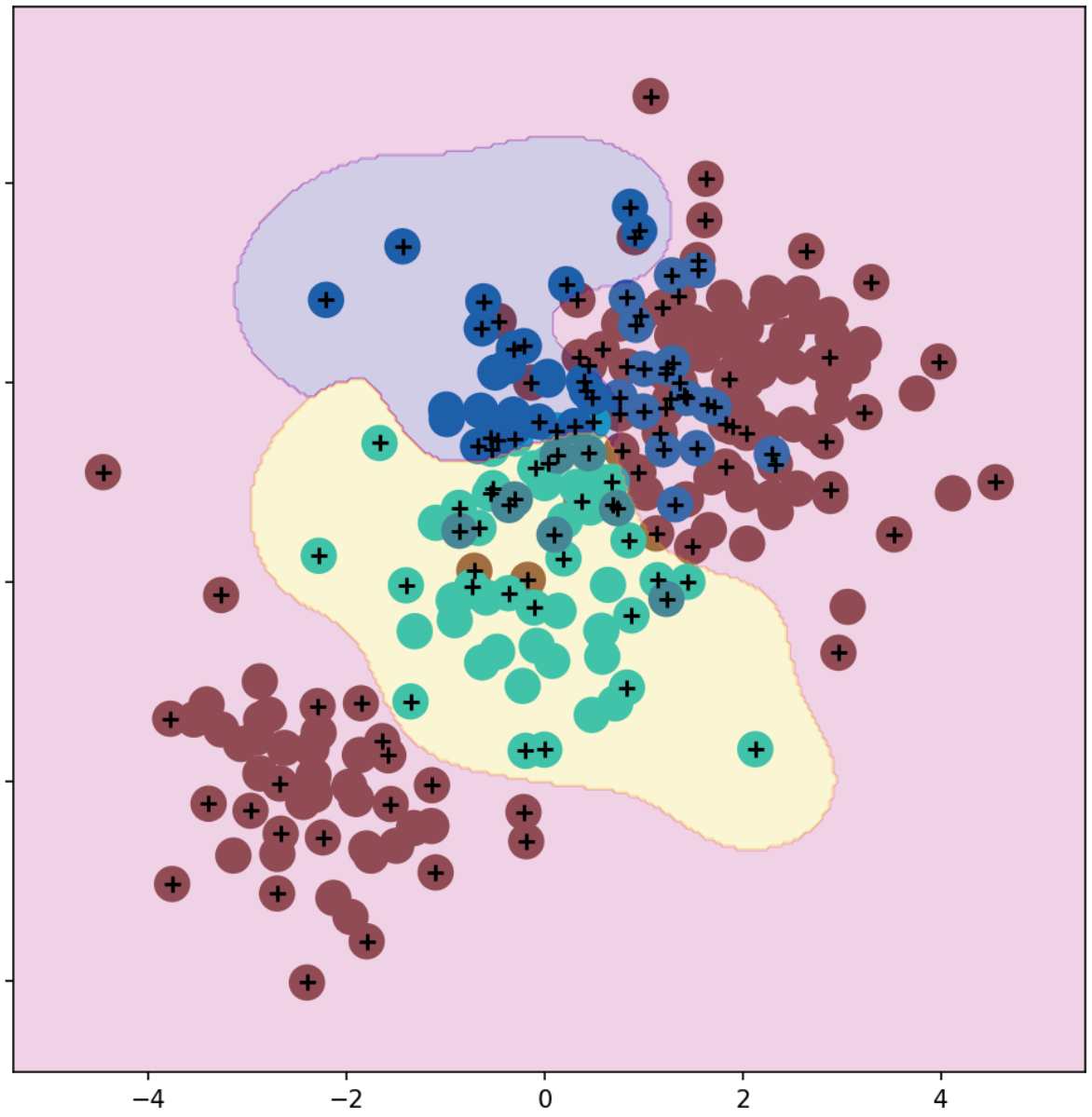
```
#生成数据，多个分类
rng = np.random.default_rng(123)
X = np.vstack([X, rng.standard_normal([50, 2])])
y = np.hstack([y, [0]*50])
X[y==0, 1] += 2
fig, ax = plt.subplots(figsize=(8, 8))
ax.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
```



现在拟合多分类的SVM：

```
#指定一对一多分类SVM
svm_rbf_3 = SVC(kernel="rbf",
                C=10,
                gamma=1,
                decision_function_shape='ovo');
svm_rbf_3.fit(X, y)
```

```
fig, ax = plt.subplots(figsize=(8,8))
plot_svm(X,
         y,
         svm_rbf_3,
         scatter_cmap=plt.cm.tab10,
         ax=ax)
plt.show()
```



使用 `skleran.svm` 中的 `SupportVectorRegression()` 可以执行支持向量回归模型

9.6.5 SVM在基因表达数据中的应用

由于资料的缺失，我们没有找到`Khan`数据集，此处用SVC拟合了一个超大特征数量p的数据集

#CS
#ML