Lab: 支持向量机

李文东

最后编译于 11/25/2022

我们使用 `R` 中的 `e1071` 库来演示支持向量机和SVM。此外，还可以使用 `LiblineaR` 库，其更适用于大数据集的线性分类问题。

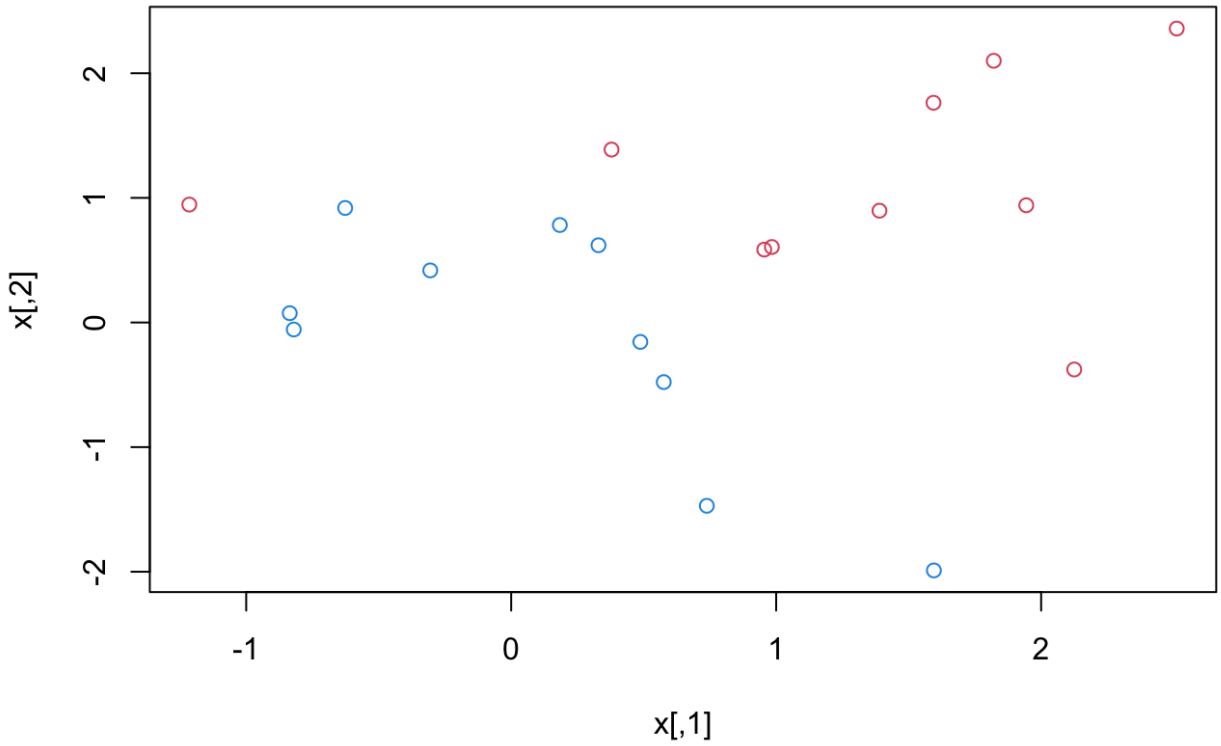
1 支持向量分类器

`e1071` 库包含许多统计学习方法的函数实现。例如，可以使用参数为 `kernel = "linear"` 的 `svm()` 函数来拟合支持向量分类器。

对于支持向量分类器， `cost` 参数可以指定超出边界的损失幅度（即课上所学的 `C` ）。当 `cost` 参数很小时，间隔会很大，并且很多支持向量会在间隔的错误一侧。当 `cost` 参数较大时，间隔会很小，在间隔上或在其错误一侧的支持向量会很少。

我们现在使用 `svm()` 函数在给定 `cost` 参数下来拟合支持向量分类器。这里我们在二维情况下来演示一下此函数的使用，并绘制出决策边界。我们首先生成属于两个类的观测值，并检查这两类是否线性可分。

```
set.seed(1)
x <- matrix(rnorm(20 * 2), ncol = 2)
y <- c(rep(-1, 10), rep(1, 10))
x[y == 1, ] <- x[y == 1, ] + 1
plot(x, col = (3 - y))
```



很明显是非线性可分。

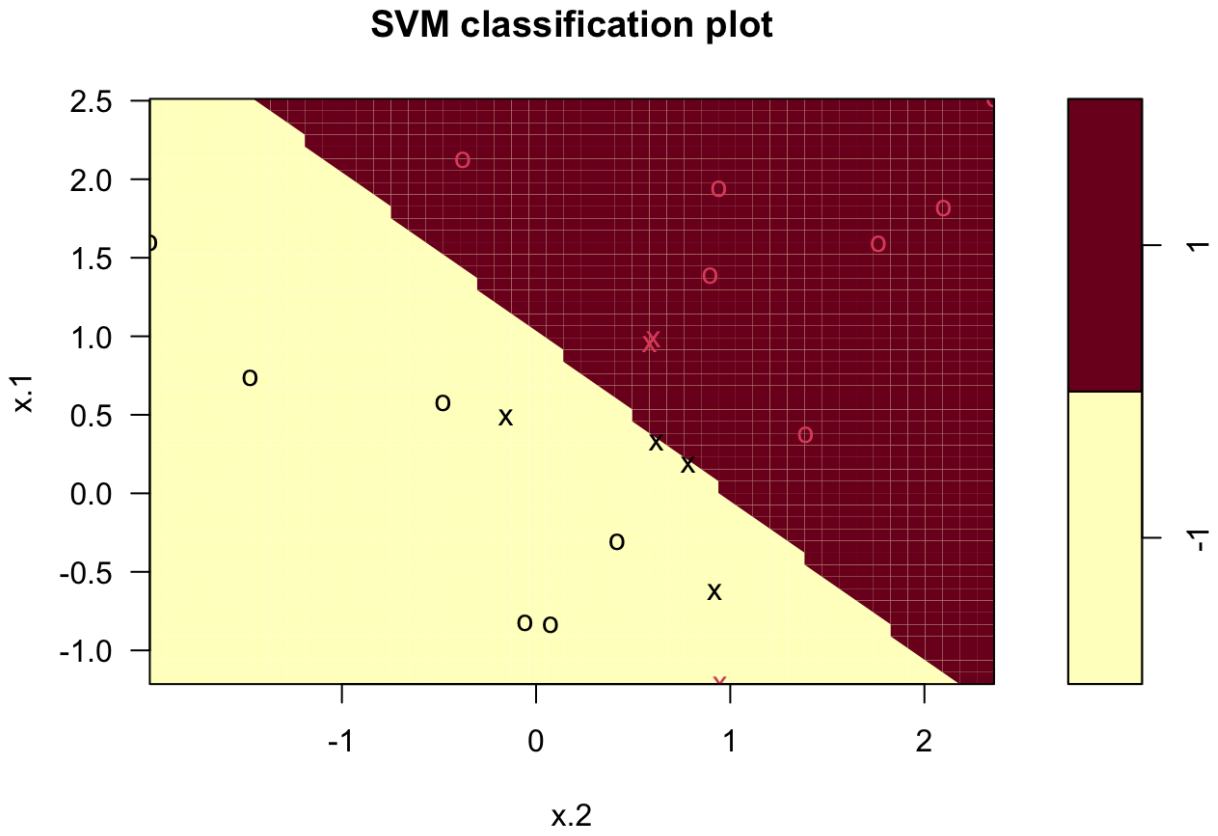
现在我们拟合支持向量分类器。注意到，如果想让 `svm()` 执行分类（而不是支持向量回归），我们必须将响应变量转变为因子型。我们现在创建一个数据框，其响应变量是因子型。

```
dat <- data.frame(x = x, y = as.factor(y))
library(e1071)
svmfit <- svm(y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)
```

参数 `scale = FALSE` 使得 `svm()` 函数不会把每个特征标准化为均值为0，方差为1。根据应用场景，可能更应当将参数选取为 `scale = TRUE` 。

现在可以绘制出所得的支持向量分类器：

```
plot(svmfit, dat)
```



注意到SVM `plot()` 函数的两个参数分别是 `svm()` 函数调用的输出以及所用的数据。被分配到-1类的特征空间的区域显示为浅黄色，而$+1$类则是红色。这两个类之间的决策边界是线性的（因为我们使用了参数 `kernel = "linear"`），但由于在这个库中实现绘图函数的方式，决策边界在图中看起来有些参差不齐。（需要注意，这里第二个特征绘制在x-轴上，第一个特征绘制于y-轴，与 `R` 中常用的 `plot()` 函数相反。） 支持向量绘制为交叉符号，其余观察值绘制为圆圈。我们在这里看到有七个支持向量。我们可以用如下方式定位它们：

```
svmfit$index
```

```
## [1]  1  2  5  7 14 16 17
```

我们可以使用 `summary()` 函数获得有关支持向量分类器的拟合的一些基本信息：

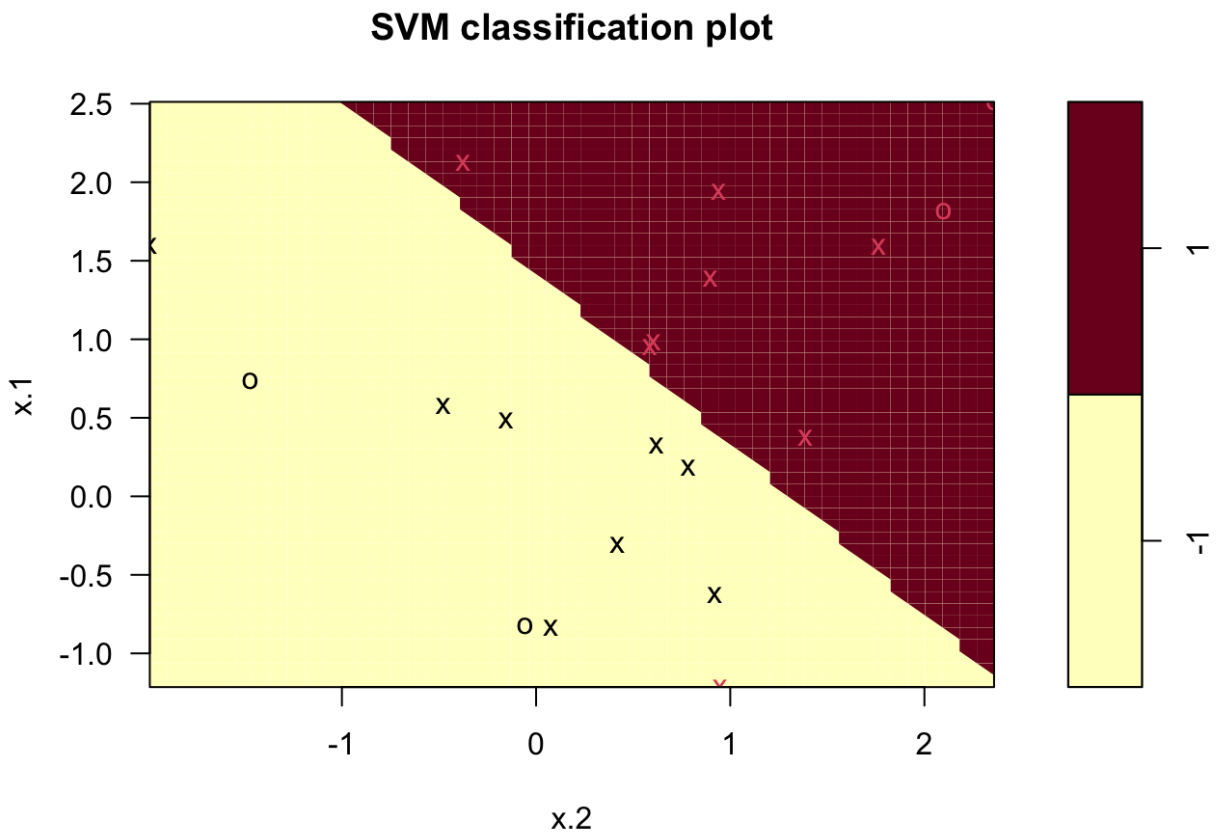
```
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  7
##
##  ( 4 3 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

例如，这个结果告诉我们，使用了线性核参数且 `cost = 10`，最终有七个支持向量，一个类中有四个，另一个类中有三个。

如果使用一个更小的 `cost` 参数会怎样?

```
svmfit <- svm(y ~ ., data = dat, kernel = "linear", cost = 0.1, scale = FALSE)
plot(svmfit, dat)
```




```
svmfit$index
```

```
##  [1]  1  2  3  4  5  7  9 10 12 13 14 15 16 17 18 20
```

在使用了更小的 cost 参数后，决策边界的间隔更大了，有了更多的支持向量。然而不好的一面是， svm() 函数 没有显式输出当拟合支持向量分类器时获得的线性决策边界的系数，也没有输出间隔的宽度。

e1071 库包含一个内置函数 tune() ，用于执行交叉验证。默认情况下， tune() 对一组模型执行十折交叉验证。为了使用此函数，我们输入有关正在考虑的模型集的相关信息。下面的命令表示我们要对比有着不同 cost 参数且基于线性核函数的支持向量分类器。

```
set.seed(1)
tune.out <- tune(svm, y ~ ., data = dat, kernel = "linear",
    ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
```

我们可以很容易地使用 summary() 函数访问每个模型的交叉验证误差：

```
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.05
##
## - Detailed performance results:
##    cost error dispersion
## 1 1e-03  0.55  0.4377975
## 2 1e-02  0.55  0.4377975
## 3 1e-01  0.05  0.1581139
## 4 1e+00  0.15  0.2415229
## 5 5e+00  0.15  0.2415229
## 6 1e+01  0.15  0.2415229
## 7 1e+02  0.15  0.2415229
```

可以看出 cost = 0.1 时交叉验证的误差最小。 tune() 函数保存了最佳的模型，可以通过如下方式访问：

```
bestmod <- tune.out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = dat, ranges = list(cost = c(0.001,
##     0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##
## Number of Support Vectors:  16
##
##  ( 8 8 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

`predict()` 函数可以用来预测任意cost参数下的测试观测值的分类，我们首先生成一个测试数据集：

```
xtest <- matrix(rnorm(20 * 2), ncol = 2)
ytest <- sample(c(-1, 1), 20, rep = TRUE)
xtest[ytest == 1, ] <- xtest[ytest == 1, ] + 1
testdat <- data.frame(x = xtest, y = as.factor(ytest))
```

现在我们预测这些测试观测值的分类标签。我们使用通过交叉验证获得的最佳模型进行预测：

```
ypred <- predict(bestmod, testdat)
table(predict = ypred, truth = testdat$y)
```

```
##        truth
## predict -1 1
##      -1  9 1
##      1   2 8
```

因此，使用这个 `cost` 参数时，一共有17个测试观测值被正确分类，有3个被错误分类。如果我们改用 `cost = 0.01` 会怎样？

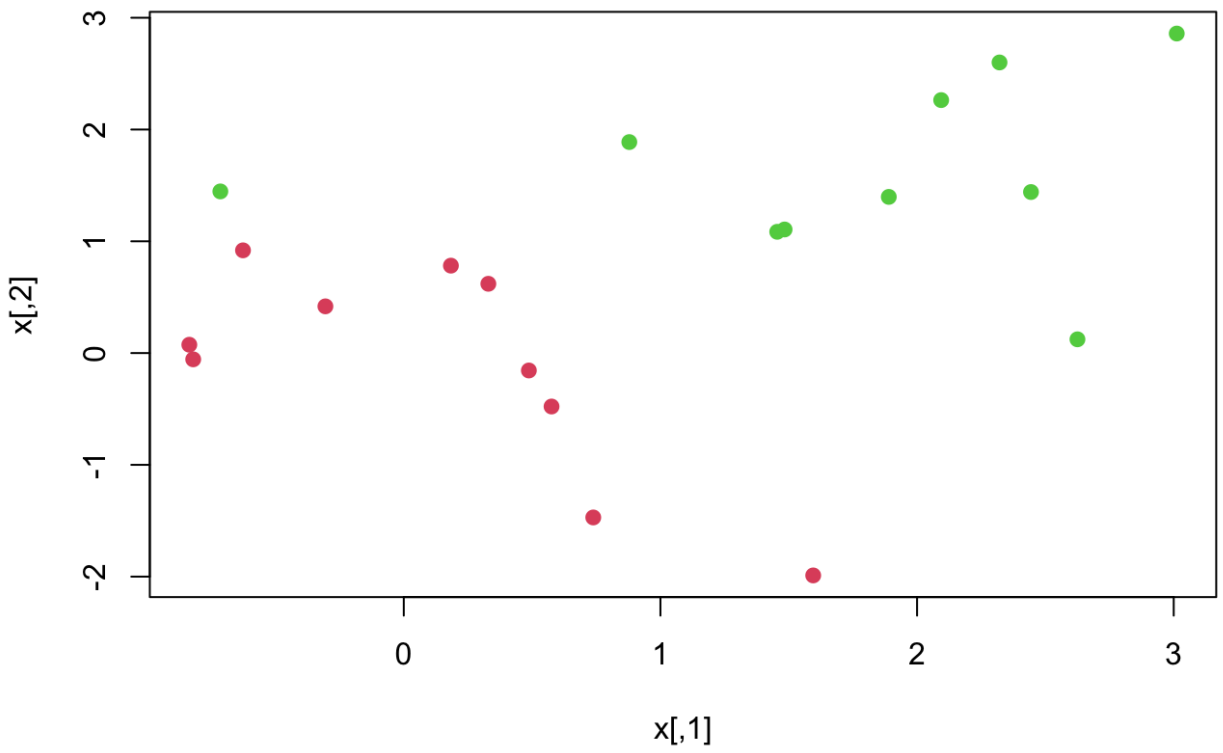
```
svmfit <- svm(y ~ ., data = dat, kernel = "linear", cost = .01, scale = FALSE)
ypred <- predict(svmfit, testdat)
table(predict = ypred, truth = testdat$y)
```

```
##        truth
## predict -1  1
##      -1 11  6
##      1   0  3
```

在这种情况下，有6个观测值被错误分类。

现在考虑两个类是线性可分的情况。然后我们可以使用 `svm()` 函数找到分离超平面。我们首先在模拟数据中进一步分离两个类，以使得它们是线性可分的：

```
x[y == 1, ] <- x[y == 1, ] + 0.5
plot(x, col = (y + 5) / 2, pch = 19)
```

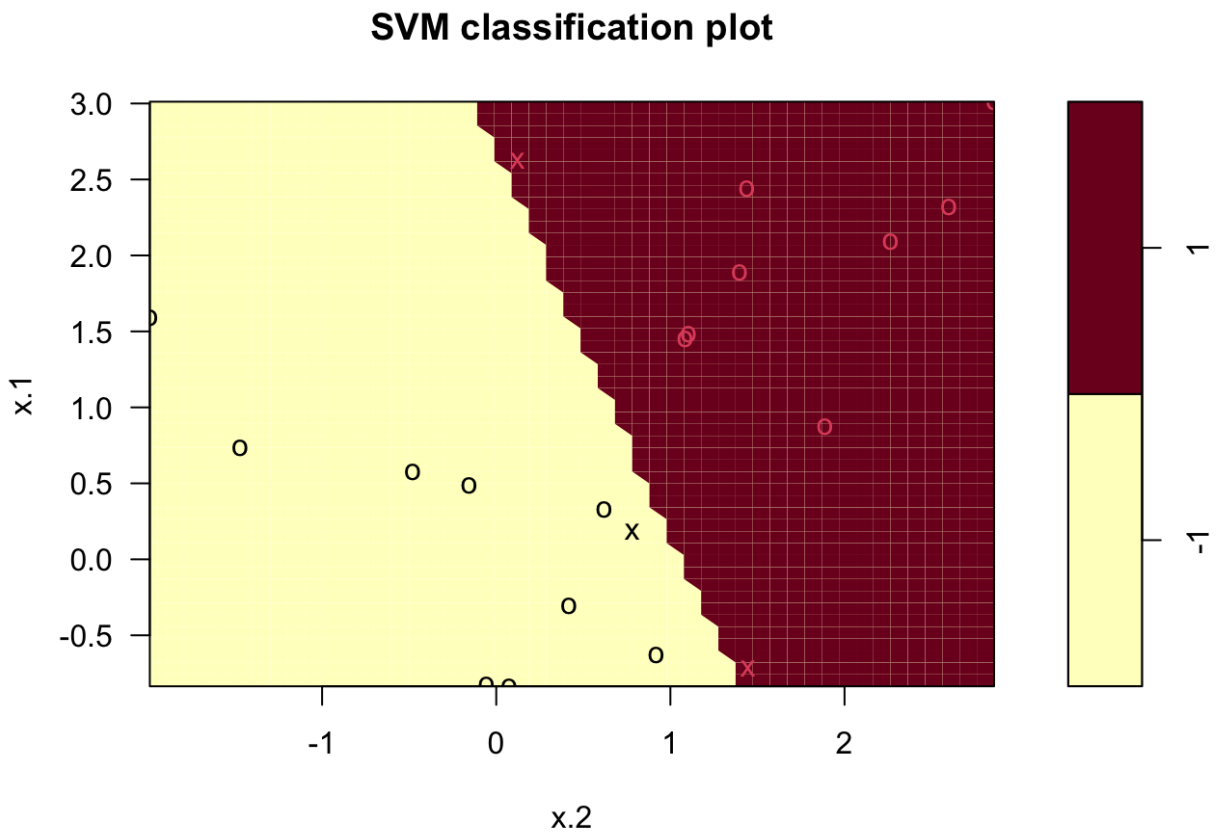


现在观测值几乎是线性可分的。我们拟合了支持向量分类器并绘制了生成的超平面，并且使用了非常大的 `cost` 值，这样就不会对观测值进行错误分类。

```
dat <- data.frame(x = x, y = as.factor(y))
svmfit <- svm(y ~ ., data = dat, kernel = "linear", cost = 1e5)
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1e+05)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1e+05
##
## Number of Support Vectors:  3
##
##  ( 1 2 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```
plot(svmfit, dat)
```

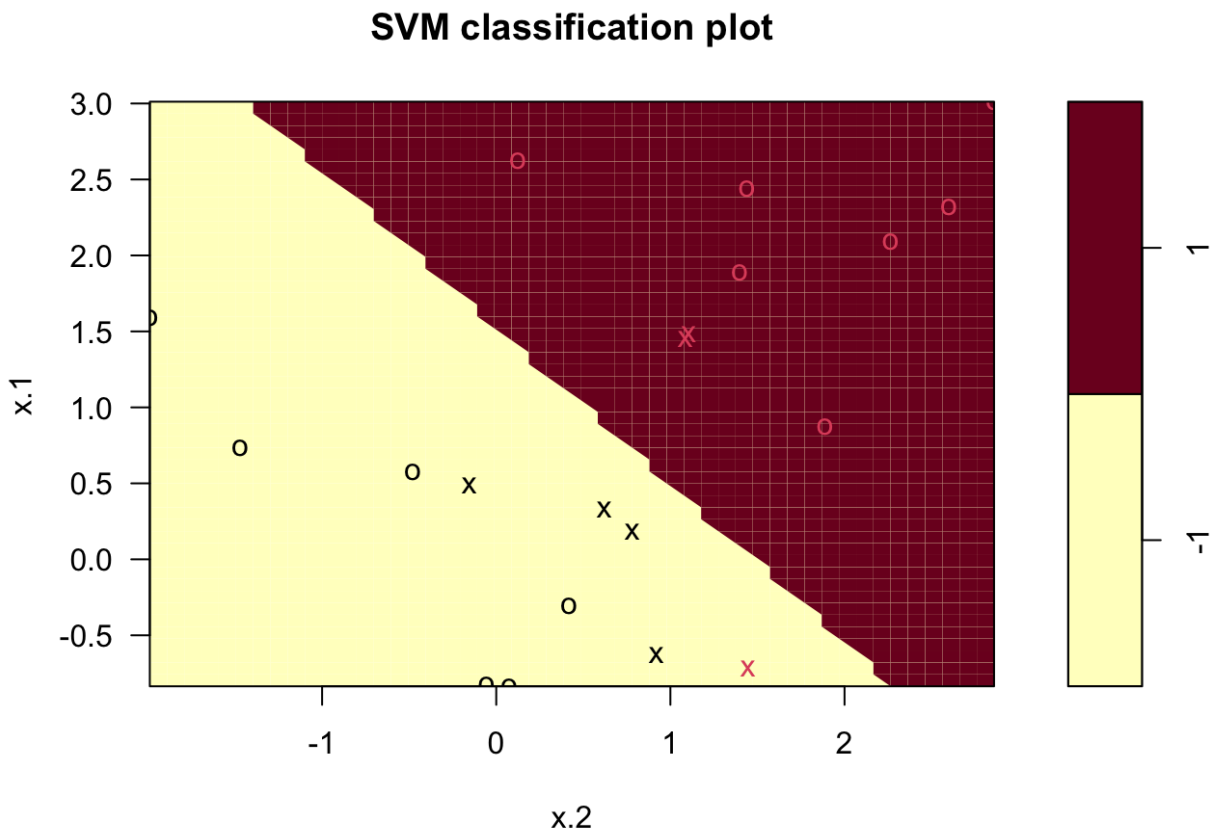


没有出现训练错误，并且只使用了三个支持向量。 然而，我们可以从图中看到，决策边界非常窄（因为不是支持向量的观测值（用圆圈表示）非常接近决策边界）。这个模型在测试数据上的表现似乎可能会很差。我们现在尝试使用一个更小的 `cost` 值：

```
svmfit <- svm(y ~ ., data = dat, kernel = "linear", cost = 1)
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  7
##
##  ( 4 3 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```
plot(svmfit, dat)
```



使用 `cost = 1`，我们对一个训练观测值进行了错误分类，但我们也获得了更大的决策边界，并使用了七个支持向量。 看起来这个模型比在 `cost = 1e5` 时表现更好。

2 支持向量机

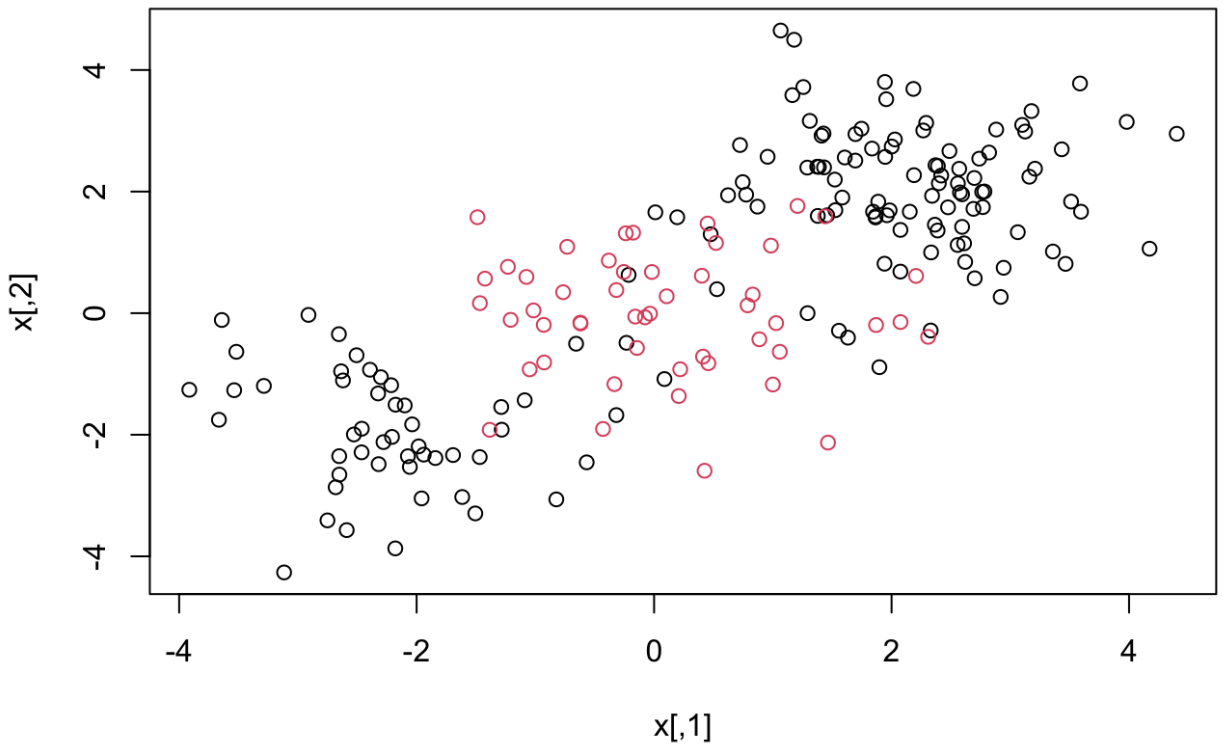
为使用非线性核函数拟合SVM，我们再次使用 `svm()` 函数。但是，现在我们使用了不同的参数 `kernel` 。为使用多项式核拟合SVM，我们选取 `kernel = "polynomial"` ；为使用高斯核，我们选取 `kernel = "radial"` 。不同的核函数有着自己的参数需要指定，参见帮助文档。

我们首先生成一些有着非线性类边界的数据如下：

```
set.seed(1)
x <- matrix(rnorm(200 * 2), ncol = 2)
x[1:100, ] <- x[1:100, ] + 2
x[101:150, ] <- x[101:150, ] - 2
y <- c(rep(1, 150), rep(2, 50))
dat <- data.frame(x = x, y = as.factor(y))
```

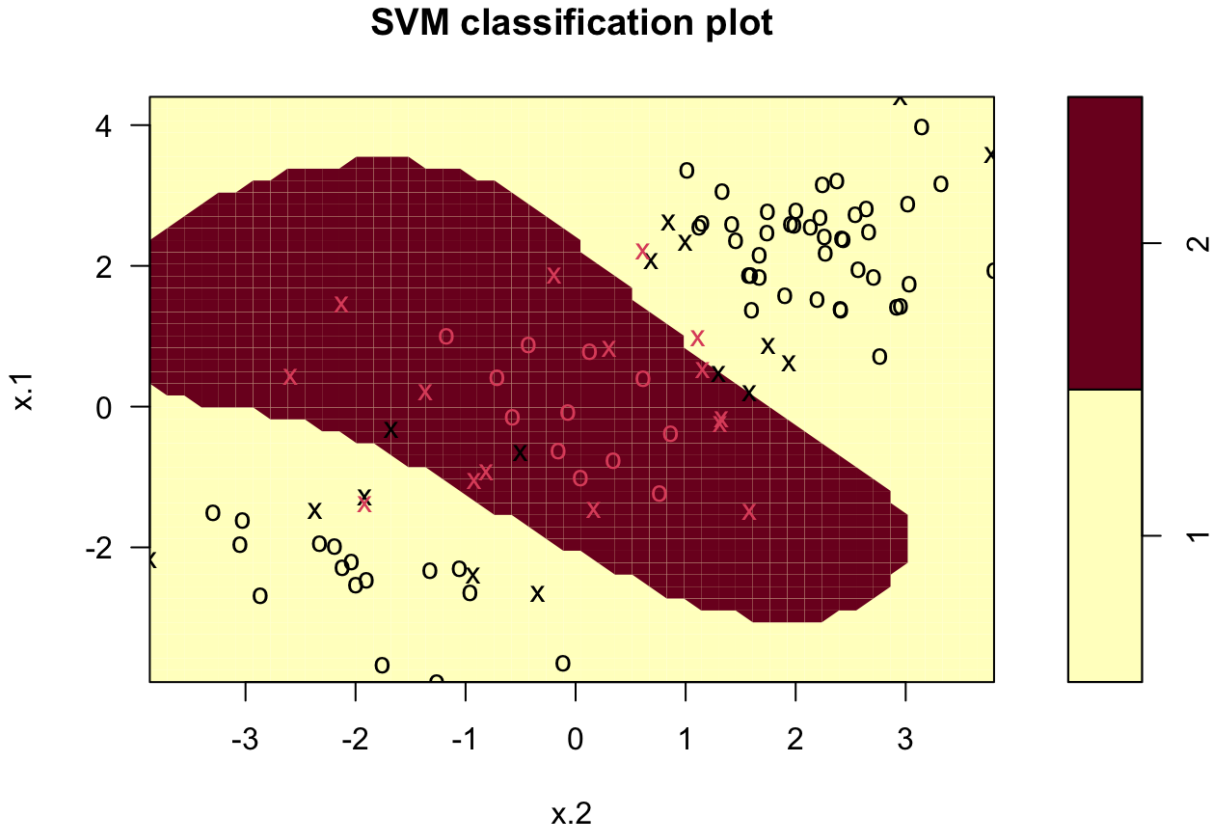
数据的图表明，类边界确实是非线性的：

```
plot(x, col = y)
```



数据被随机分为训练集和测试集。我们用训练集来拟合基于高斯核函数，参数为$\gamma = 1$的 `svm()` ：

```
train <- sample(200, 100)
svmfit <- svm(y ~ ., data = dat[train, ], kernel = "radial", gamma = 1, cost = 1)
plot(svmfit, dat[train, ])
```



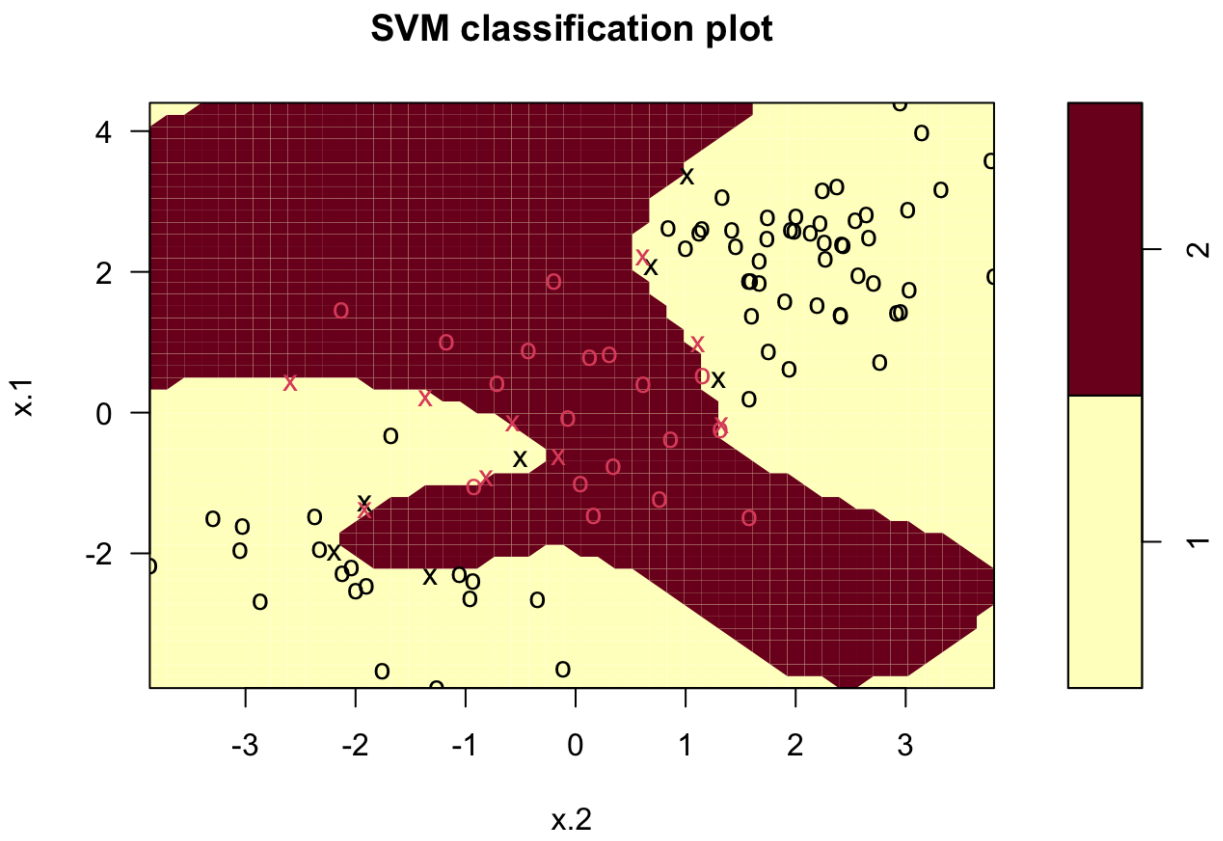
该图显示，生成的SVM具有明显的非线性边界。可使用 `summary()` 函数来获取该SVM的一些信息：

```
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat[train, ], kernel = "radial", gamma = 1,
##     cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  31
##
##  ( 16 15 )
##
##
## Number of Classes:  2
##
## Levels:
##  1 2
```

我们可以从图中看到，在这种SVM拟合中存在大量的训练错误。如果我们增加 `cost` 的值，我们可以减少训练错误的数量。然而，这是以更不规则的决策边界为代价的，这似乎有过度拟合数据的风险。

```
svmfit <- svm(y ~ ., data = dat[train, ], kernel = "radial", gamma = 1, cost = 1e5)
plot(svmfit, dat[train, ])
```



我们使用 `tune()` 来执行交叉验证，来选择一个最优的SVM的γ 和 `cost`：

```
set.seed(1)
tune.out <- tune(svm, y ~ ., data = dat[train, ],
    kernel = "radial",
    ranges = list(
      cost = c(0.1, 1, 10, 100, 1000),
      gamma = c(0.5, 1, 2, 3, 4)
    )
  )
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.07
##
## - Detailed performance results:
##    cost gamma error dispersion
## 1  1e-01   0.5  0.26 0.15776213
## 2  1e+00   0.5  0.07 0.08232726
## 3  1e+01   0.5  0.07 0.08232726
## 4  1e+02   0.5  0.14 0.15055453
## 5  1e+03   0.5  0.11 0.07378648
## 6  1e-01   1.0  0.22 0.16193277
## 7  1e+00   1.0  0.07 0.08232726
## 8  1e+01   1.0  0.09 0.07378648
## 9  1e+02   1.0  0.12 0.12292726
## 10 1e+03   1.0  0.11 0.11005049
## 11 1e-01   2.0  0.27 0.15670212
## 12 1e+00   2.0  0.07 0.08232726
## 13 1e+01   2.0  0.11 0.07378648
## 14 1e+02   2.0  0.12 0.13165612
## 15 1e+03   2.0  0.16 0.13498971
## 16 1e-01   3.0  0.27 0.15670212
## 17 1e+00   3.0  0.07 0.08232726
## 18 1e+01   3.0  0.08 0.07888106
## 19 1e+02   3.0  0.13 0.14181365
## 20 1e+03   3.0  0.15 0.13540064
## 21 1e-01   4.0  0.27 0.15670212
## 22 1e+00   4.0  0.07 0.08232726
## 23 1e+01   4.0  0.09 0.07378648
## 24 1e+02   4.0  0.13 0.14181365
## 25 1e+03   4.0  0.15 0.13540064
```

因此最优的参数为 `cost = 1` 和 `gamma = 0.5`。 我们可以通过对数据使用 `predict()` 函数来查看此模型的测试集预测。需要注意，我们使用 `-train` 作为索引集来获取测试数据。

```
table(
    true = dat[-train, "y"],
    pred = predict(tune.out$best.model, newdata = dat[-train, ])
    )
```

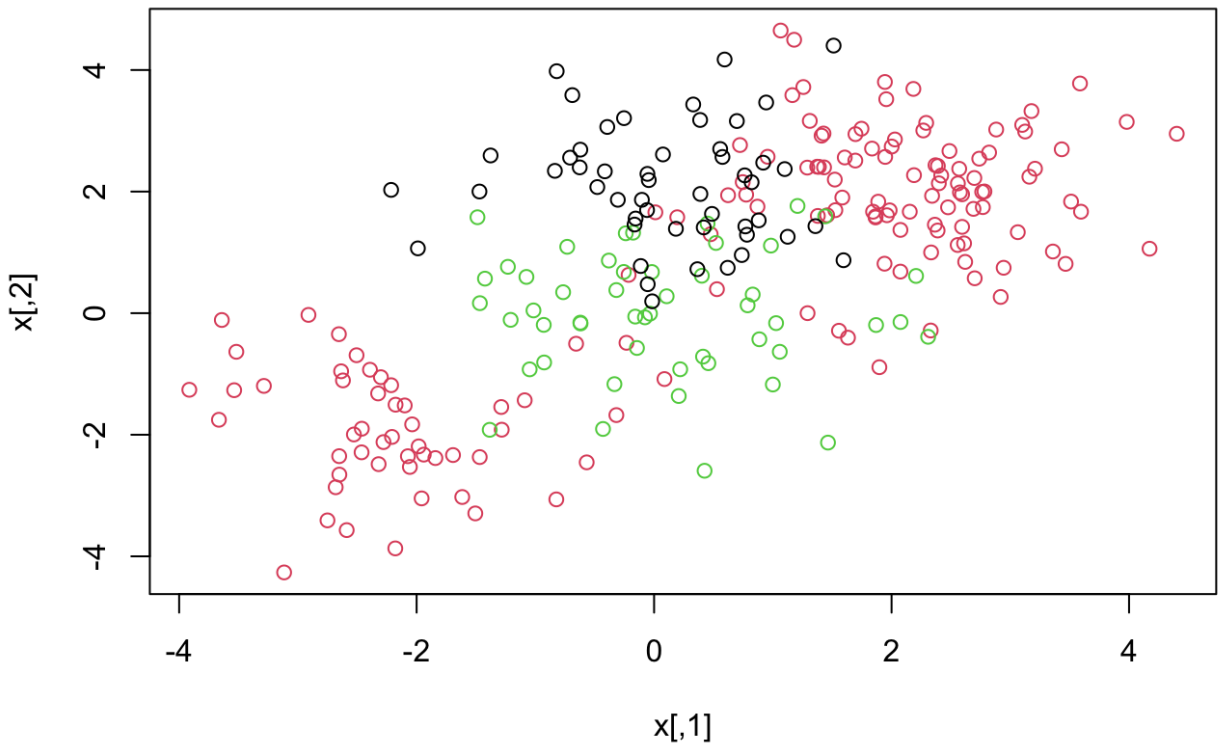
```
##     pred
## true  1  2
##    1 67 10
##    2  2 21
```

该SVM模型错误分类了12 % 的测试集观测值。

3 多分类SVM

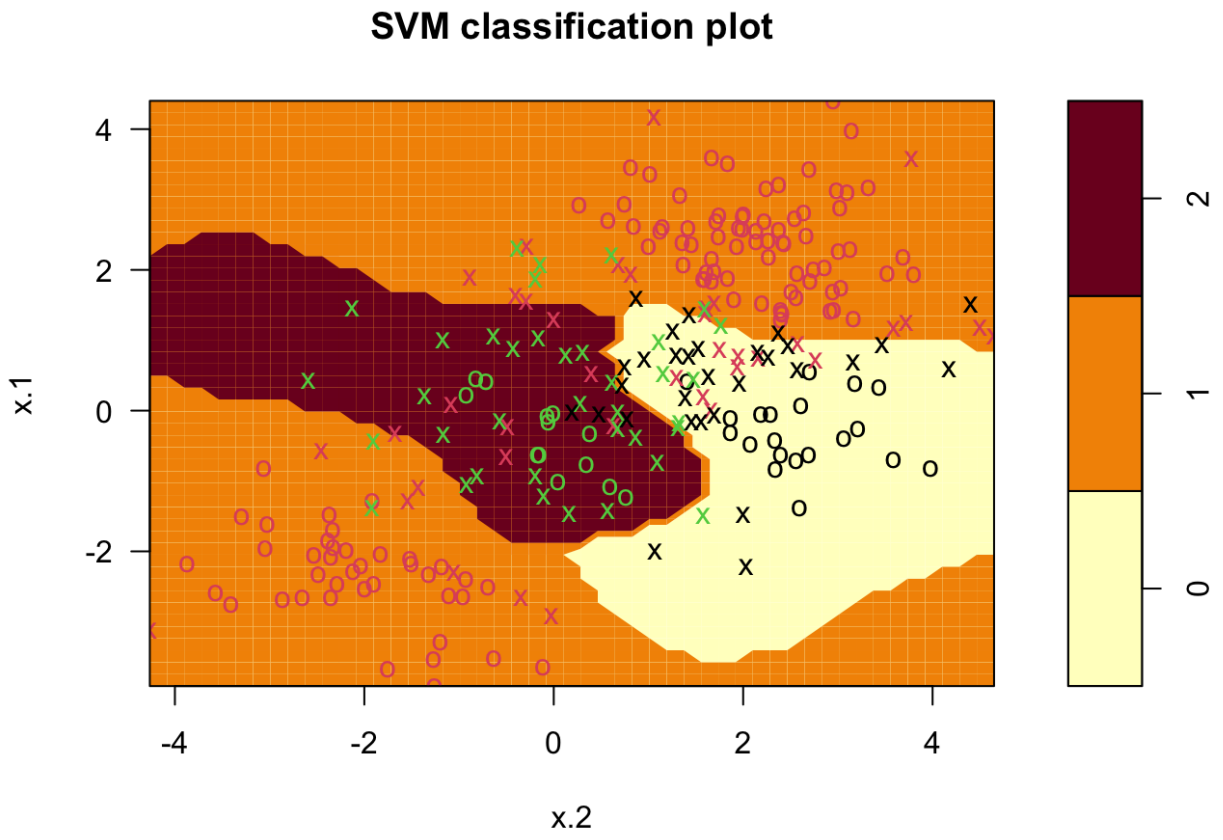
如果响应变量包含两个以上的因子类别，则 svm() 函数将使用一对一方法执行多分类。我们在这里通过生成第三类观测值来探索这种设定。

```
set.seed(1)
x <- rbind(x, matrix(rnorm(50 * 2), ncol = 2))
y <- c(y, rep(0, 50))
x[y == 0, 2] <- x[y == 0, 2] + 2
dat <- data.frame(x = x, y = as.factor(y))
plot(x, col = (y + 1))
```



我们现在用这组数据来拟合SVM：

```
svmfit <- svm(y ~ ., data = dat, kernel = "radial", cost = 10, gamma = 1)
plot(svmfit, dat)
```



4 基因表达数据的应用

我们现在检查 Khan 数据集，该数据集由与四种不同类型的小圆蓝色细胞肿瘤对应的大量组织样本组成。对于每个组织样本，基因表达测度都是可获取的。 数据集由训练数据 xtrain 和 ytrain 以及测试数据 xtest 和 ytest 组成。

我们先查看数据集的维度：

```
library(ISLR2)
names(Khan)
```

```
## [1] "xtrain" "xtest"  "ytrain" "ytest"
```

```
dim(Khan$xtrain)
```

```
## [1]   63 2308
```

```
dim(Khan$xtest)
```

```
## [1]   20 2308
```

数据集包含了2,308个基因的表达测度。训练集和测试集分别包含了63和20个观测值。

```
table(Khan$ytrain)
```

```
##
##  1  2  3  4
##  8 23 12 20
```

```
table(Khan$ytest)
```

```
##
## 1 2 3 4
## 3 6 6 5
```

我们将使用支持向量方法通过基因表达测度来预测癌症亚型。 在这个数据集中，相对于观测数量，特征的数量很庞大。这表明我们应该使用线性核，因为使用多项式或高斯核所带来的额外的灵活性并不十分必要。

```
dat <- data.frame(
    x = Khan$xtrain,
    y = as.factor(Khan$ytrain)
  )
out <- svm(y ~ ., data = dat, kernel = "linear", cost = 10)
summary(out)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  58
##
##  ( 20 20 11 7 )
##
##
## Number of Classes:  4
##
## Levels:
##  1 2 3 4
```

```
table(out$fitted, dat$y)
```

```
##
##      1  2  3  4
##   1  8  0  0  0
##   2  0 23  0  0
##   3  0  0 12  0
##   4  0  0  0 20
```

我们看到模型在训练集上完全无误差。事实上这并不奇怪，因为相对于观测数量而言，特征的数量很大，这意味着很容易找到分离超平面。我们最感兴趣的不是支持向量分类器对训练集的表现，而是它对测试集的表现。

```
dat.te <- data.frame(
    x = Khan$xtest,
    y = as.factor(Khan$ytest))
pred.te <- predict(out, newdata = dat.te)
table(pred.te, dat.te$y)
```

```
##
## pred.te 1 2 3 4
##       1 3 0 0 0
##       2 0 6 2 0
##       3 0 0 4 0
##       4 0 0 0 5
```

我们可以看出使用 `cost = 10` 在该测试集上产生了两个错误。

5 练习 & 第三次作业

我们已经看到，可以用非线性核拟合SVM，以便使用非线性决策边界执行分类。我们现在将看到，我们还可以通过使用特征的非线性变换执行逻辑回归来获得非线性决策边界。

a. 生成一个$n = 500$且$p = 2$的数据集，使得观测值属于两个类，它们之间具有二次决策边界。例如，可以按如下方式执行：

```
x1 <- runif(500) - 0.5
x2 <- runif(500) - 0.5
y <- 1 * (x1^2 - x2^2 > 0)
```

b. 绘制观测值并按标签赋颜色。在x轴上标注X1，在y轴上标注X2。

c. 使用X1和X2作为预测变量，对数据拟合逻辑回归模型。

d. 将该模型应用于训练数据集，以获得每个训练观测值的预测类标签。绘制观测值，并根据预测的类标签着色。决策边界应为线性。

e. 使用X1和X2的非线性函数作为预测因子，用逻辑回归模型拟合数据（例如，$X1^2$，$X1 * X2$，$log(X2)$,以此类推）。

f. 将该模型应用于训练数据，以获得每个训练观测值的预测类标签。绘制观测值，根据类标签着色。决策边界应明显为非线性的。如果不是，那么重复(a)-(e)，直到找到一个预测的类标签明显是非线性的例子。

g. 用支持向量分类器拟合数据，并将X1和X2作为预测变量。获得每个训练观测值的分类预测。绘制观测值，并根据预测的类标签着色。

h. 使用非线性核的SVM拟合数据。获得每个训练观测值的分类预测。绘制观测值，并根据预测的类标签着色。

i. 描述你获得的结果。

要求

- 12月4日周日晚24点截止上交，上交pdf文件（一定要pdf，否则无法批改，可以Knit直接生成或html转存）至邮箱：lyfsufe@163.com (mailto:lyfsufe@163.com)

- 务必创建一个新的Rmd文件，不要使用我们的教学文档直接上交作业