**Chapter 9: Support Vector Machines**

**1(a&b)**

> x1 = c(-5:5)

> x2 = 1 + 3 \* x1

> plot(x1, x2, xlab = "x1", ylab = "x2", type = "l", col = "red")

> text(x = -4, y = -5, labels = "1 + 3X1 − X2 < 0", col = "red")

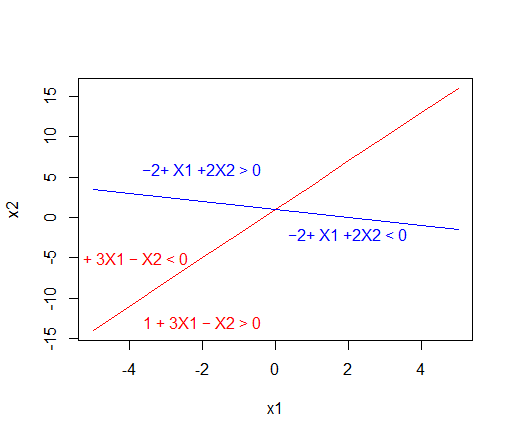
> text(x = -2, y = -13, labels = "1 + 3X1 − X2 > 0", col = "red")

> x3 = -1/2 \* (x1 - 2)

> lines(x1, x3, col = "blue")

> text(x = -2, y = 6, labels = "−2+ X1 +2X2 > 0", col = "blue")

> text(x = 2, y = -2, labels = "−2+ X1 +2X2 < 0", col = "blue")

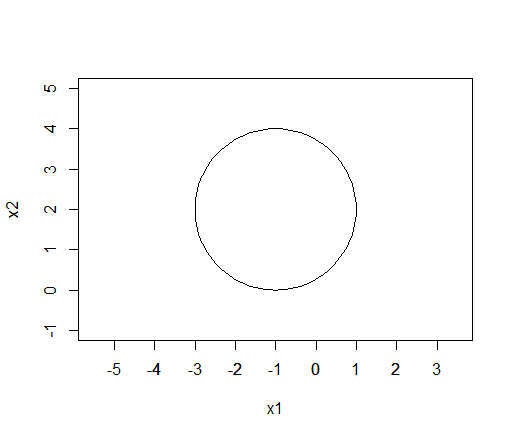


**2(a)**

> plot(NA,NA,xlim = c(-4,2),ylim = c(-1,5),xlab = "x1",ylab = "x2",asp = 1)

> axis(1,c(-5:3))

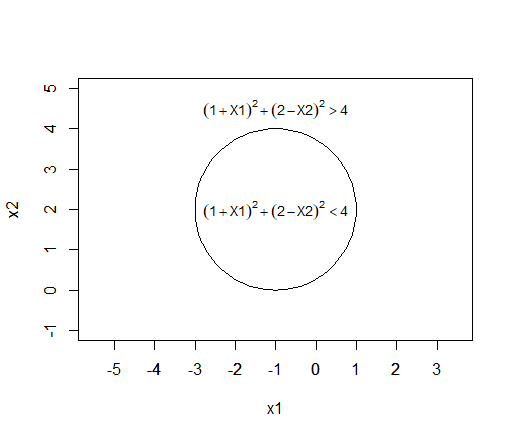
> draw.circle(-1,2,2)



**2(b)**

> text(x = -1, y = 2, expression((1 + X1)^2 + (2 - X2)^2 < 4),cex = 0.9)

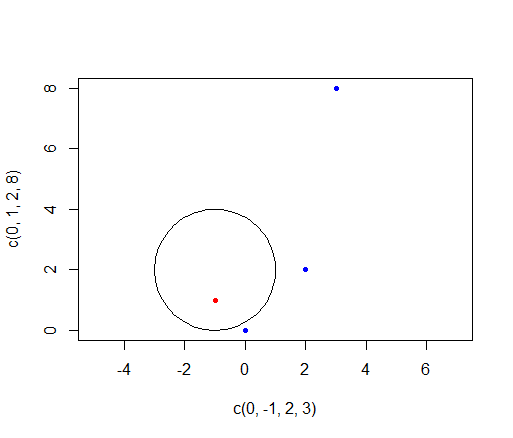
> text(x = -1, y = 4.5, expression((1 + X1)^2 + (2 - X2)^2 > 4),cex = 0.9)



**2(c)**

> plot(c(0, -1, 2, 3), c(0, 1, 2, 8), col = c("blue", "red", "blue", "blue"),pch = 20,asp = 1)

> draw.circle(-1,2,2)



如上图所示，除了(-1,1)属于红类，剩下的三个点都属于蓝类。

**2(d)**

将展开为



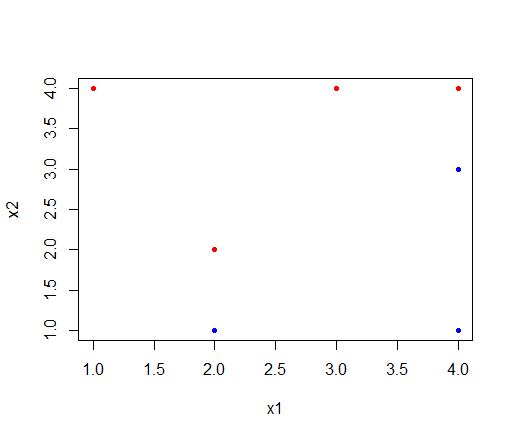
可以看作是关于，，及的线性边界。

**3(a)**

> x1 = c(3, 2, 4, 1, 2, 4, 4)

> x2 = c(4, 2, 4, 4, 1, 3, 1)

> plot(x1, x2, col = c("red", "red", "red", "red", "blue", "blue", "blue"),pch = 20)



**3(b)**

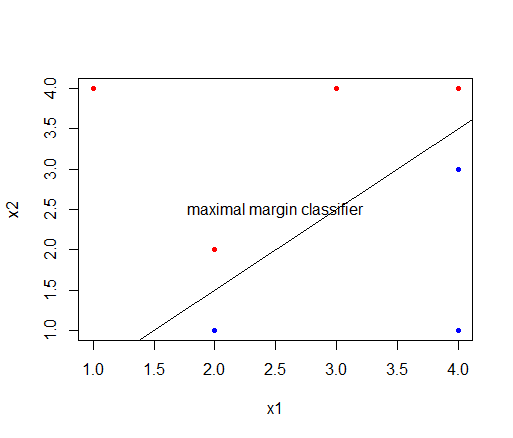
超平面需要经过点(2,1)，(2,2)，(4,3)及(4,4)之间。点(2,1)和(4,3)确定的直线与点(2,2)与点(4,4)确定的直线平行，则最优的超平面过点(2,1.5)和点(4,3.5)。斜率为1，截距为-0.5。

超平面的表达式为



> abline(-0.5, 1)

> text(2.5,2.5,labels = "maximal margin classifier")



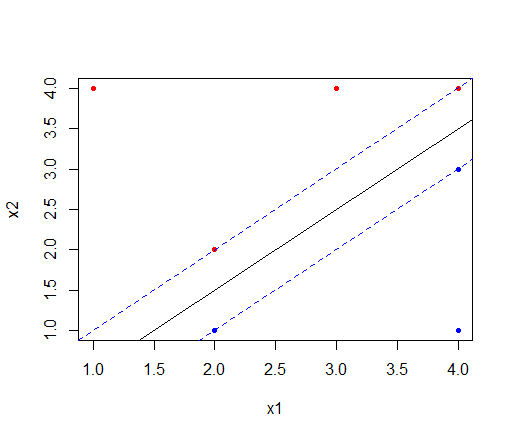
**3(c)**

如果则属于蓝类，否则属于红类。

**3(d)**

> abline(-1, 1, lty = 2, col = "blue")

> abline(0, 1, lty = 2, col = "blue")



两条蓝色虚线表示的是maximal margin classifier的margin。

**3(e)**

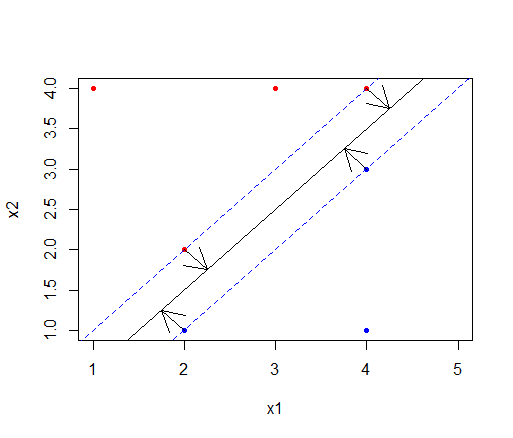
过点(2,1)，(2,2)，(4,3)及(4,4)于超平面垂直的直线与超平面的交点分别为(1.75,1.25)，(2.25,1.75)，(3.75,3.25)及(4.25,3.75)。

> arrows(2, 1, 1.75, 1.25)

> arrows(2, 2, 2.25, 1.75)

> arrows(4, 3, 3.75, 3.25)

> arrows(4, 4, 4.25, 3.75)



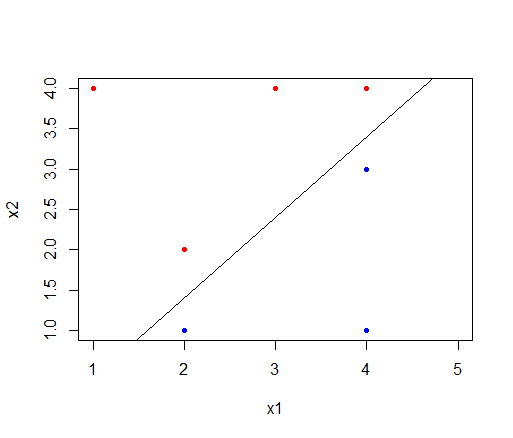
**3(f)**

第七点的坐标是(4,1)，不是支持向量，只要这个点没有移动到maximal margin classifier的margin里面就不会对maximal margin hyperplane有影响。

**3(g)**

> plot(x1, x2, col = c("red", "red", "red", "red", "blue", "blue", "blue"),pch = 20,xlim = c(1,5))

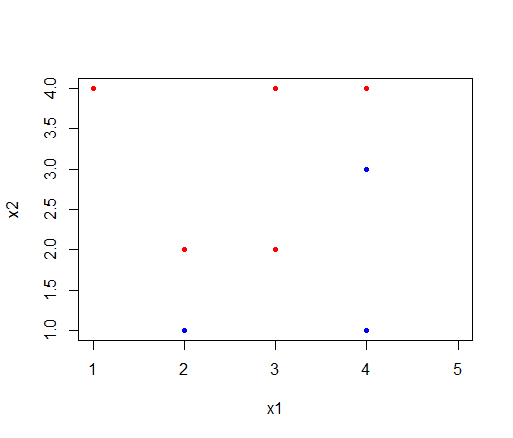
> abline(-0.6, 1)



**3(h)**

> plot(x1, x2, col = c("red", "red", "red", "red", "blue", "blue", "blue"),pch = 20,xlim = c(1,5))

> points(3,2,pch = 20,col = "red")



**4**

> set.seed(100)

> x = rnorm(100)

> y = 5 \* x^2 + 10 \* x + 8 + rnorm(100)

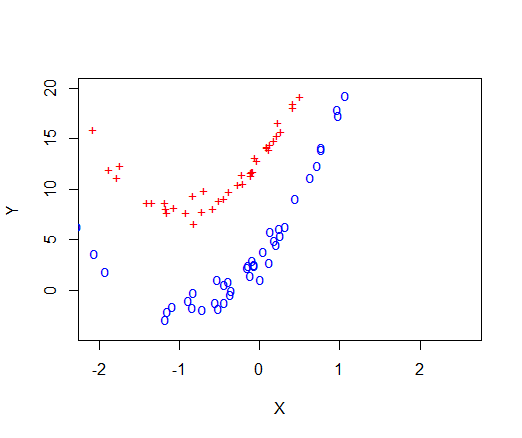
> train = sample(100, 50)

> y[train] = y[train] + 5

> y[-train] = y[-train] - 5

> plot(x[train], y[train], pch="+", lwd=4, col="red", ylim=c(-4, 20), xlab="X", ylab="Y")

> points(x[-train], y[-train], pch="o", lwd=4, col="blue")



> library(e1071)

> set.seed(10)

> z = rep(0, 100)

> z[train] = 1

> adjusted\_train = c(sample(train, 25), sample(setdiff(1:100, train), 25))

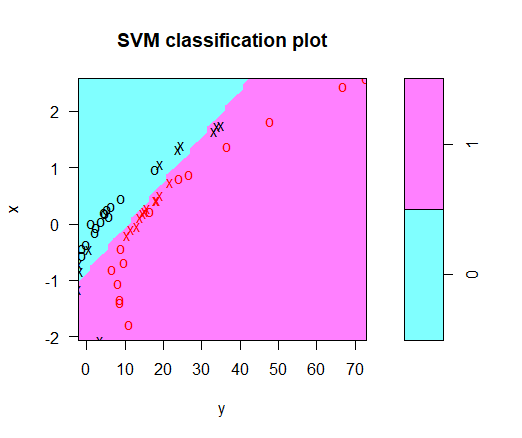
> data.train = data.frame(x=x[adjusted\_train], y=y[adjusted\_train], z=as.factor(z[adjusted\_train]))

> data.test = data.frame(x=x[-adjusted\_train], y=y[-adjusted\_train], z=as.factor(z[-adjusted\_train]))

首先是训练集。对于支持向量分类器：

> svm.linear = svm(z~., data=data.train, kernel="linear", cost=10)

> plot(svm.linear, data.train)



> table(z[adjusted\_train], predict(svm.linear, data.train))

0 1

0 20 5

1 0 25

> mean(z[adjusted\_train] == predict(svm.linear, data.train))

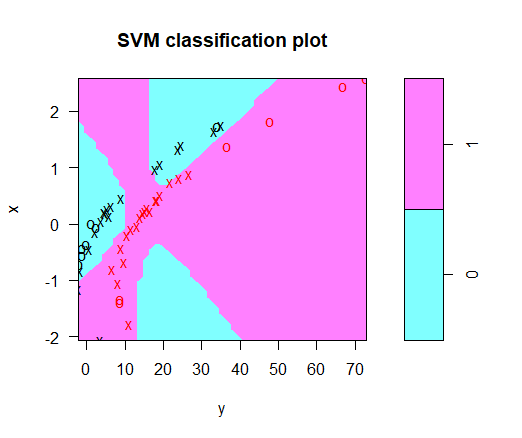
[1] 0.9

支持向量分类器的准确率为90%。

对于多项式核函数支持向量机：

> svm.poly = svm(z~., data=data.train, kernel="polynomial", cost=10)

> plot(svm.poly, data.train)



> table(z[adjusted\_train], predict(svm.poly, data.train))

0 1

0 23 2

1 0 25

> mean(z[adjusted\_train] == predict(svm.poly, data.train))

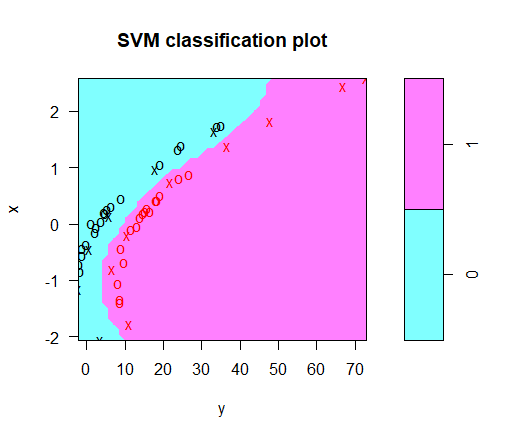
[1] 0.96

多项式核函数支持向量机的准确率为96%。

对于径向核函数支持向量机：

> svm.radial = svm(z~., data=data.train, kernel="radial", gamma=1, cost=10)

> plot(svm.radial, data.train)



> table(z[adjusted\_train], predict(svm.radial, data.train))

0 1

0 25 0

1 0 25

> mean(z[adjusted\_train] == predict(svm.radial, data.train))

[1] 1

径向核函数支持向量机的准确率为100%。

可以看到，对于训练集，多项式核函数支持向量机和径向核函数支持向量机的表现比支持向量分类器要好。下面再来比较一下测试集。

对于支持向量分类器：

> table(z[-adjusted\_train], predict(svm.linear, data.test))

0 1

0 17 8

1 0 25

> mean(z[-adjusted\_train] == predict(svm.linear, data.test))

[1] 0.84

> table(z[-adjusted\_train], predict(svm.poly, data.test))

0 1

0 14 11

1 1 24

> mean(z[-adjusted\_train] == predict(svm.poly, data.test))

[1] 0.76

> table(z[-adjusted\_train], predict(svm.radial, data.test))

0 1

0 24 1

1 0 25

> mean(z[-adjusted\_train] == predict(svm.radial, data.test))

[1] 0.98

对于测试集，径向核函数支持向量机的准确率为98%，表现最好；多项式核函数支持向量机的准确率为76%，表现最差；支持向量分类器的准确率为84%。

**5(a)**

> set.seed(250)

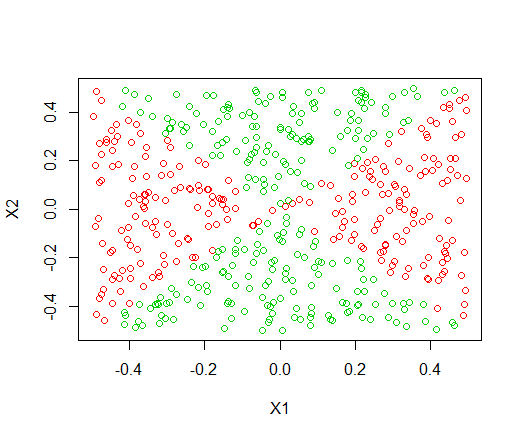
> x1 = runif(500) - 0.5

> x2 = runif(500) - 0.5

> y = 1 \* (x1^2 - x2^2 > 0)

**5(b)**

plot(x1, x2, col = (3-y), xlab = "X1", ylab = "X2")



**5(c)**

> lm.fit = glm(y ~ x1 + x2, family = binomial)

**5(d)**

> data = data.frame(x1 = x1, x2 = x2, y = y)

> lm.prob = predict(lm.fit, data, type = "response")

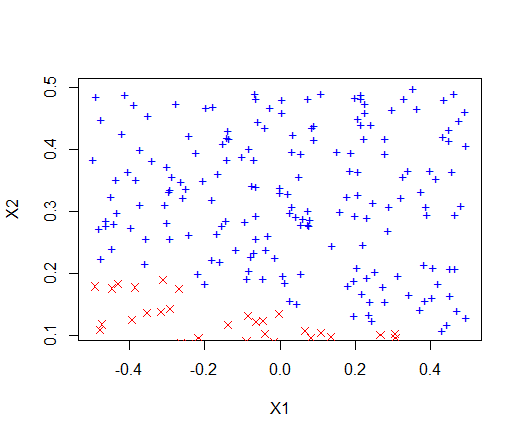
> lm.pred = ifelse(lm.prob > 0.48, 1, 0)

> data\_1 = data[lm.pred == 1, ]

> data\_0 = data[lm.pred == 0, ]

> plot(data\_1$x1, data\_1$x2, col = "blue", xlab = "X1", ylab = "X2", pch = "+")

> points(data\_0$x1, data\_0$x2, col = "red", pch = 4)



**5(e)**

> lm.fit = glm(y ~ I(x1^2) +I(x2^2)+ I(x1 \* x2) + log(x1+1)+ log(x2+1), data = data, family = binomial)

**5(f)**

> lm.prob = predict(lm.fit, data, type = "response")

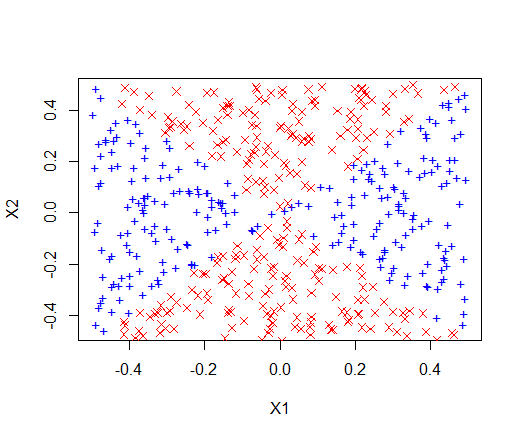
> lm.pred = ifelse(lm.prob > 0.5, 1, 0)

> data\_1 = data[lm.pred == 1, ]

> data\_0 = data[lm.pred == 0, ]

> plot(data\_1$x1, data\_1$x2, col = "blue", xlab = "X1", ylab = "X2", pch = "+")

> points(data\_0$x1, data\_0$x2, col = "red", pch = 4)



**5(g)**

> library(e1071)

> svm.fit = svm(as.factor(y) ~ x1 + x2, data, kernel = "linear", cost = 1)

> svm.pred = predict(svm.fit, data)

> data\_1 = data[svm.pred == 1, ]

> data\_0 = data[svm.pred == 0, ]

> plot(data\_1$x1, data\_1$x2, col = "blue", xlab = "X1", ylab = "X2", pch = "+")

Error in plot.window(...) : 'xlim'值不能是无限的

In addition: Warning messages:

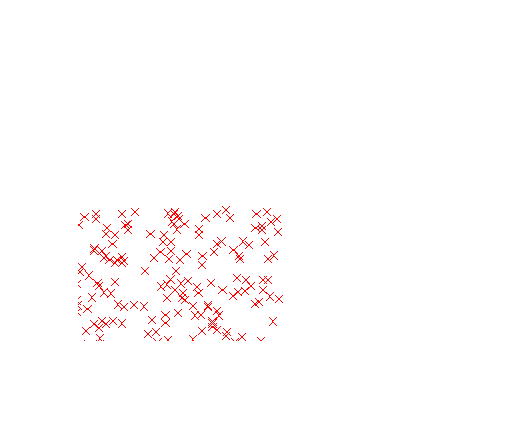
1: In min(x) : no non-missing arguments to min; returning Inf

2: In max(x) : no non-missing arguments to max; returning -Inf

3: In min(x) : no non-missing arguments to min; returning Inf

4: In max(x) : no non-missing arguments to max; returning -Inf

> points(data\_0$x1, data\_0$x2, col = "red", pch = 4)



无法画出正常的图，支持向量分类器只能把训练集所有数据分到一个类里面。

**5(h)**

多项式核函数支持向量机：

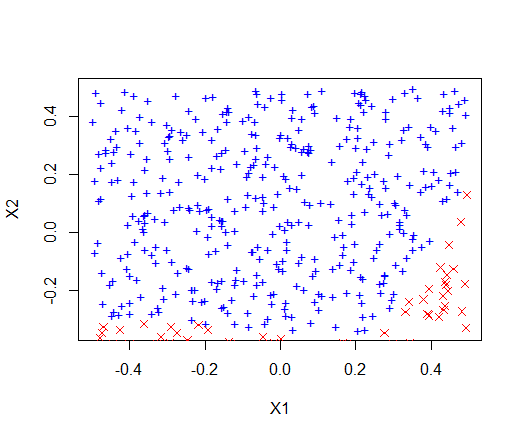
> svm.pred = predict(svm.fit, data)

> data\_1 = data[svm.pred == 1, ]

> data\_0 = data[svm.pred == 0, ]

> plot(data\_1$x1, data\_1$x2, col = "blue", xlab = "X1", ylab = "X2", pch = "+")

> points(data\_0$x1, data\_0$x2, col = "red", pch = 4)



> table(data$y,svm.pred)

svm.pred

0 1

0 73 190

1 34 203

效果不是很好。

径向核函数支持向量机：

> svm.fit = svm(as.factor(y) ~ x1 + x2, data, kernel = "radial",cost = 1,gamma = 1)

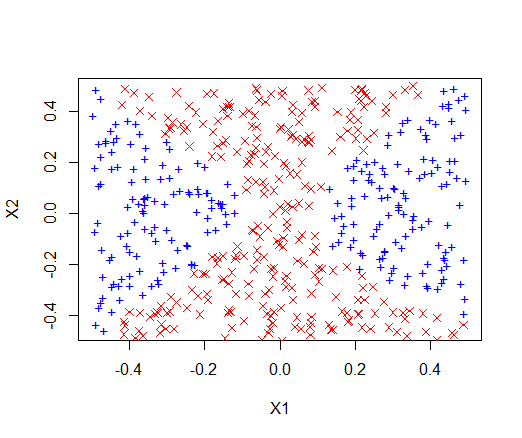
> svm.pred = predict(svm.fit, data)

> data\_1 = data[svm.pred == 1, ]

> data\_0 = data[svm.pred == 0, ]

> plot(data\_1$x1, data\_1$x2, col = "blue", xlab = "X1", ylab = "X2", pch = "+")

> points(data\_0$x1, data\_0$x2, col = "red", pch = 4)



> table(data$y,svm.pred)

svm.pred

0 1

0 255 8

1 14 223

径向核函数支持向量机的预测结果好多了。

**5(i)**

径向核函数支持向量机很擅长寻找非线性边界。无论是不存在交互项的Logistic回归还是支持向量分类器都无法很好地找到非线性决策边界。通过调整参数（如分类的概率阈值），含交互项的Logistic回归也可以找出非线性边界，不过相比于径向核函数支持向量机就要麻烦一些了。

**6(a)**

> set.seed(96)

> x1 = runif(500, 5, 95)

> y1 = runif(500, x1, 95)

> x1\_noise = runif(50, 20, 80)

> y1\_noise = 5/4\*(x1\_noise - 10)+1

> x0 = runif(500, 10, 100)

> y0 = runif(500, 5, x0-5)

> x0\_noise = runif(50, 20, 80)

> y0\_noise = 5/4\*(x0\_noise - 10)-1

>

> class1 = seq(1, 550)

> x = c(x1, x1\_noise, x0, x0\_noise)

> y = c(y1, y1\_noise, y0, y0\_noise)

> plot(x[class1], y[class1], col = "blue", pch = "+", ylim = c(0, 100),xlab = "x1",ylab = "x1")

> points(x[-class1], y[-class1], col = "red", pch = 4)

**6(b)**

> library(e1071)

> set.seed(960)

> z = rep(0, 1100)

> z[class1] = 1

> data = data.frame(x = x, y = y, z = z)

> tune.out = tune(svm, as.factor(z) ~ ., data = data, kernel = "linear", ranges = list(cost = c(0.01,0.1, 1, 5, 10, 100, 1000)))

> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:

cost

5

- best performance: 0.04181818

- Detailed performance results:

cost error dispersion

1 1e-02 0.11272727 0.02975258

2 1e-01 0.06818182 0.02357023

3 1e+00 0.04727273 0.01808053

4 5e+00 0.04181818 0.01670794

5 1e+01 0.04181818 0.01670794

6 1e+02 0.04181818 0.01670794

7 1e+03 0.04181818 0.01670794

> data.frame(cost = tune.out$performances$cost, misclass = tune.out$performances$error \* 1100)

cost misclass

1 1e-02 125

2 1e-01 75

3 1e+00 54

4 5e+00 45

5 1e+01 45

6 1e+02 45

7 1e+03 45

cost与misclass对应如上表，总数（1100）与error相乘就得到了错误分类即misclass的数量。

**6(c)**

> set.seed(9600)

> x\_test = runif(1000, 0, 100)

> class1 = sample(1000, 500)

> y\_test = rep(0, 1000)

> for (i in class1) {

+ y\_test[i] = runif(1, x\_test[i], 100)

+ }

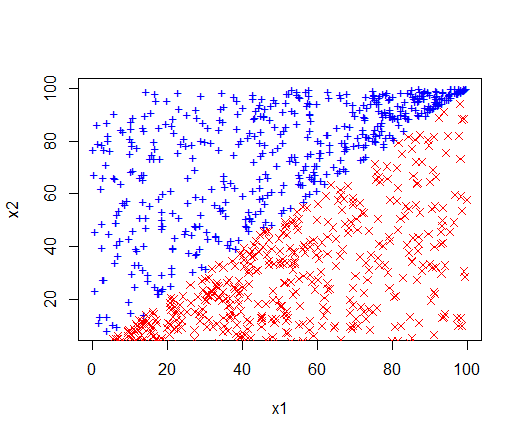
> for (i in setdiff(1:1000, class1)) {

+ y\_test[i] = runif(1, 0, x\_test[i])

+ }

> plot(x\_test[class1], y\_test[class1], col = "blue", pch = "+",xlab = "x1",ylab = "x1")

> points(x\_test[-class1], y\_test[-class1], col = "red", pch = 4)



> set.seed(960)

> z\_test = rep(0, 1000)

> z\_test[class1] = 1

> costs = c(0.01, 0.1, 1, 5, 10, 100, 1000)

> test\_errors = rep(0, 7)

> test\_data = data.frame(x = x\_test, y = y\_test, z = z\_test)

> for (i in 1:length(costs)) {

+ svm.fit = svm(as.factor(z) ~ ., data = data, kernel = "linear", cost = costs[i])

+ svm.predict = predict(svm.fit, test\_data)

+ test\_errors[i] = sum(svm.predict != test\_data$z)

+ }

> data.frame(cost = costs, misclass = test\_errors)

cost misclass

1 1e-02 67

2 1e-01 32

3 1e+00 17

4 5e+00 36

5 1e+01 39

6 1e+02 39

7 1e+03 42

cost取值为1时，test error最小，而之前cost取值为5及以上时，training error最小。

**6(d)**

cost取值过大造成支持向量分类器在训练集上出现过拟合，导致其在测试集上表现并不好。相反，一个较小的cost对应的支持向量分类器在训练集上会受到一些噪声的干扰导致结果受到一定的影响，不过却能在测试集上表现得更好。

**7(a)**

> library(ISLR)

> Auto$mpglevel = as.factor(ifelse(Auto$mpg > median(Auto$mpg), 1, 0))

**7(b)**

> library(e1071)

> set.seed(97)

> tune.out = tune(svm, mpglevel ~ ., data = Auto, kernel = "linear", ranges = list(cost = c(0.01, 0.1, 1, 5, 10, 100,1000)))

> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:

cost

1

- best performance: 0.01282051

- Detailed performance results:

cost error dispersion

1 1e-02 0.07423077 0.04909306

2 1e-01 0.04865385 0.04268849

3 1e+00 0.01282051 0.02179068

4 5e+00 0.02051282 0.03151981

5 1e+01 0.02307692 0.03715738

6 1e+02 0.03833333 0.03671460

7 1e+03 0.03833333 0.03671460

> data.frame(cost = tune.out$performances$cost, cv\_errors = tune.out$performances$error)

cost cv\_errors

1 1e-02 0.07423077

2 1e-01 0.04865385

3 1e+00 0.01282051

4 5e+00 0.02051282

5 1e+01 0.02307692

6 1e+02 0.03833333

7 1e+03 0.03833333

cost取值为1时，支持向量分类器的表现最好。

**7(c)**

首先是多项式核函数支持向量机：

> set.seed(970)

> tune.out = tune(svm, mpglevel ~ ., data = Auto, kernel = "polynomial", ranges = list(cost = c(0.01, 0.1, 1, 5, 10, 100,1000), degree = c(2, 3, 4, 5)))

> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:

cost degree

1000 3

- best performance: 0.2527564

- Detailed performance results:

cost degree error dispersion

1 1e-02 2 0.5765385 0.05539688

2 1e-01 2 0.5765385 0.05539688

3 1e+00 2 0.5765385 0.05539688

4 5e+00 2 0.5765385 0.05539688

5 1e+01 2 0.5175641 0.13086109

6 1e+02 2 0.2936538 0.07638609

7 1e+03 2 0.2707051 0.07750613

8 1e-02 3 0.5765385 0.05539688

9 1e-01 3 0.5765385 0.05539688

10 1e+00 3 0.5765385 0.05539688

11 5e+00 3 0.5765385 0.05539688

12 1e+01 3 0.5765385 0.05539688

13 1e+02 3 0.3369872 0.09007674

14 1e+03 3 0.2527564 0.07718146

15 1e-02 4 0.5765385 0.05539688

16 1e-01 4 0.5765385 0.05539688

17 1e+00 4 0.5765385 0.05539688

18 5e+00 4 0.5765385 0.05539688

19 1e+01 4 0.5765385 0.05539688

20 1e+02 4 0.5765385 0.05539688

21 1e+03 4 0.5534615 0.09580240

22 1e-02 5 0.5765385 0.05539688

23 1e-01 5 0.5765385 0.05539688

24 1e+00 5 0.5765385 0.05539688

25 5e+00 5 0.5765385 0.05539688

26 1e+01 5 0.5765385 0.05539688

27 1e+02 5 0.5765385 0.05539688

28 1e+03 5 0.5765385 0.05539688

cost取值为1000，degree取值为3时表现最好。交叉验证误差为0.2527564。

然后是径向核函数支持向量机：

> set.seed(970)

> tune.out = tune(svm, mpglevel ~ ., data = Auto, kernel = "radial", ranges = list(cost = c(0.01, 0.1, 1, 5, 10, 100,1000), gamma = c(0.01, 0.1, 1, 5, 10, 100)))

> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:

cost gamma

100 0.01

- best performance: 0.01025641

- Detailed performance results:

cost gamma error dispersion

1 1e-02 1e-02 0.57653846 0.05539688

2 1e-01 1e-02 0.08679487 0.05678071

3 1e+00 1e-02 0.07153846 0.05224802

4 5e+00 1e-02 0.05108974 0.04510931

5 1e+01 1e-02 0.02544872 0.02076512

6 1e+02 1e-02 0.01025641 0.01792836

7 1e+03 1e-02 0.02551282 0.02093755

8 1e-02 1e-01 0.20166667 0.10984951

9 1e-01 1e-01 0.07666667 0.05657125

10 1e+00 1e-01 0.05108974 0.04174500

11 5e+00 1e-01 0.03576923 0.03236852

12 1e+01 1e-01 0.03583333 0.03012207

13 1e+02 1e-01 0.03833333 0.03249241

14 1e+03 1e-01 0.03833333 0.03249241

15 1e-02 1e+00 0.57653846 0.05539688

16 1e-01 1e+00 0.57653846 0.05539688

17 1e+00 1e+00 0.05858974 0.04969058

18 5e+00 1e+00 0.06115385 0.05277752

19 1e+01 1e+00 0.06115385 0.05277752

20 1e+02 1e+00 0.06115385 0.05277752

21 1e+03 1e+00 0.06115385 0.05277752

22 1e-02 5e+00 0.57653846 0.05539688

23 1e-01 5e+00 0.57653846 0.05539688

24 1e+00 5e+00 0.51525641 0.04701806

25 5e+00 5e+00 0.51012821 0.05001443

26 1e+01 5e+00 0.51012821 0.05001443

27 1e+02 5e+00 0.51012821 0.05001443

28 1e+03 5e+00 0.51012821 0.05001443

29 1e-02 1e+01 0.57653846 0.05539688

30 1e-01 1e+01 0.57653846 0.05539688

31 1e+00 1e+01 0.53826923 0.05700128

32 5e+00 1e+01 0.53057692 0.05022599

33 1e+01 1e+01 0.53057692 0.05022599

34 1e+02 1e+01 0.53057692 0.05022599

35 1e+03 1e+01 0.53057692 0.05022599

36 1e-02 1e+02 0.57653846 0.05539688

37 1e-01 1e+02 0.57653846 0.05539688

38 1e+00 1e+02 0.57653846 0.05539688

39 5e+00 1e+02 0.57653846 0.05539688

40 1e+01 1e+02 0.57653846 0.05539688

41 1e+02 1e+02 0.57653846 0.05539688

42 1e+03 1e+02 0.57653846 0.05539688

cost取值为100，gamma取值为0.01时表现最好。交叉验证误差为0.01025641。径向核函数支持向量机的表现更好。

**7(d)**

> svm.linear = svm(mpglevel ~ ., data = Auto, kernel = "linear", cost = 1)

> svm.poly = svm(mpglevel ~ ., data = Auto, kernel = "polynomial", cost = 100,degree = 3)

> svm.radial = svm(mpglevel ~ ., data = Auto, kernel = "radial", cost = 100, gamma = 0.01)

> extand\_plot = function(fit) {

+ for (name in names(Auto)[!(names(Auto) %in% c("mpg", "mpglevel", "name"))]) {

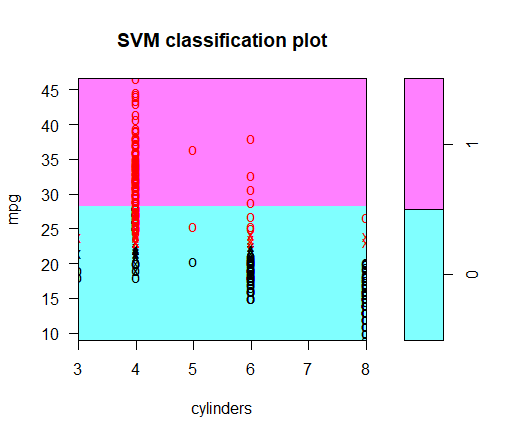
+ plot(fit, Auto, as.formula(paste("mpg~", name, sep = "")))

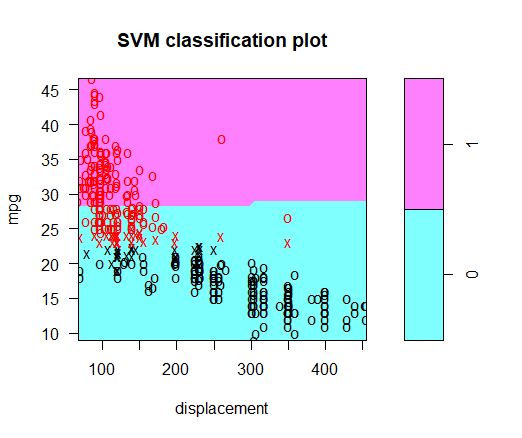
+ }

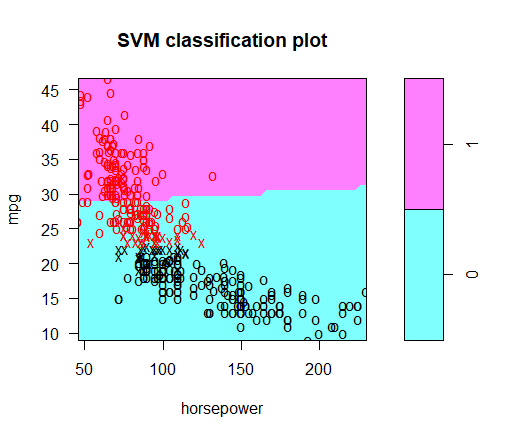
+ }

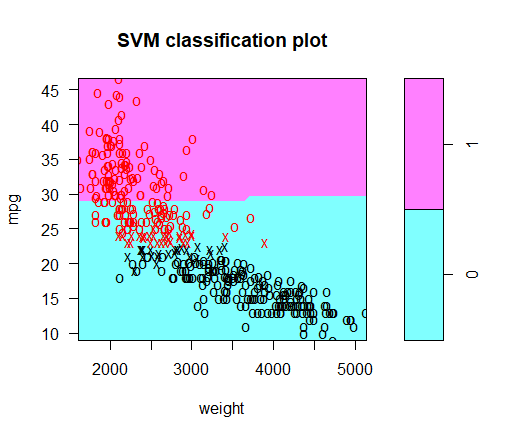
支持向量分类器：

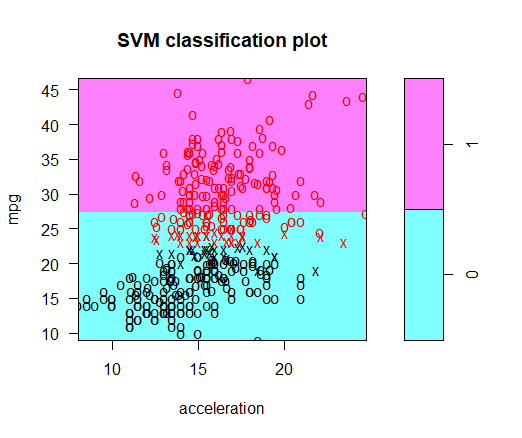
> extand\_plot(svm.linear)

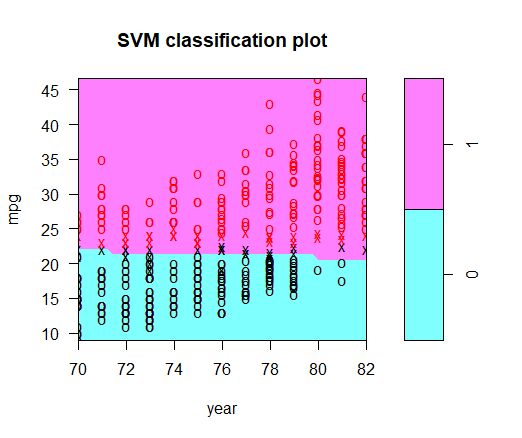


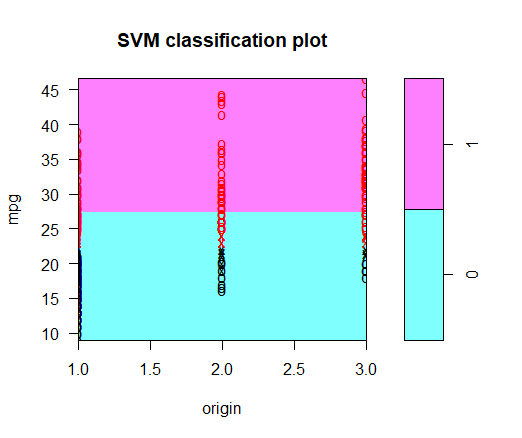






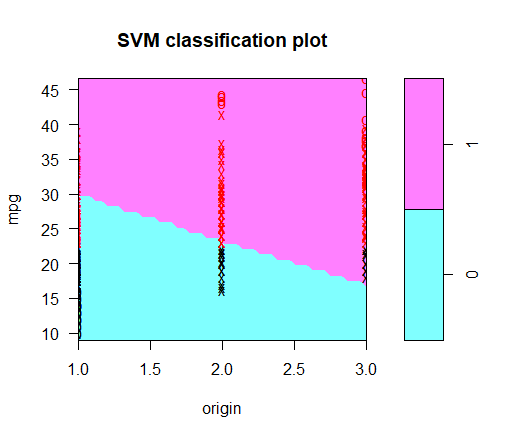
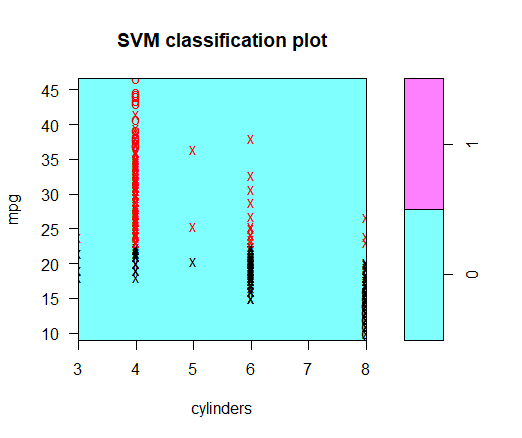
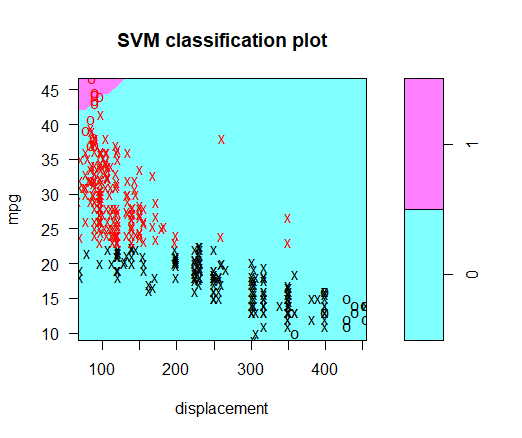
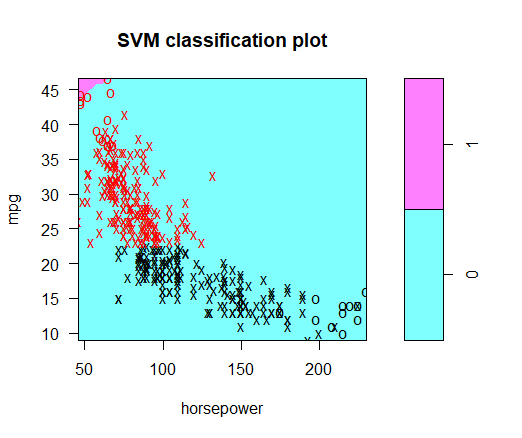
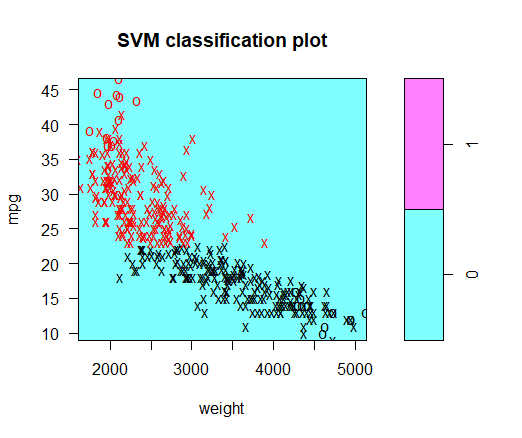
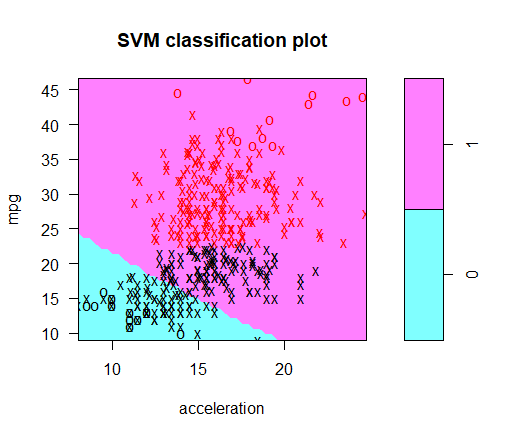
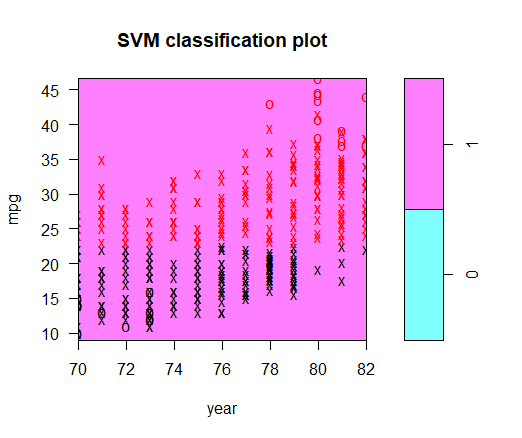






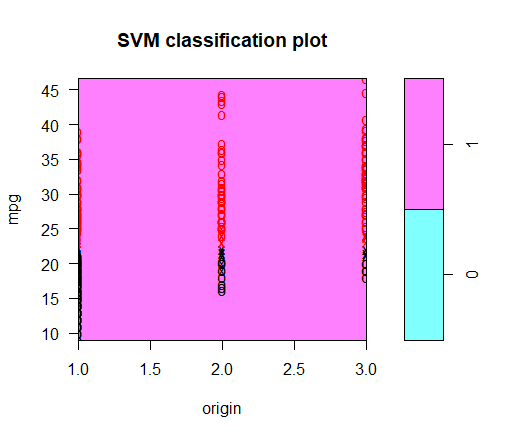
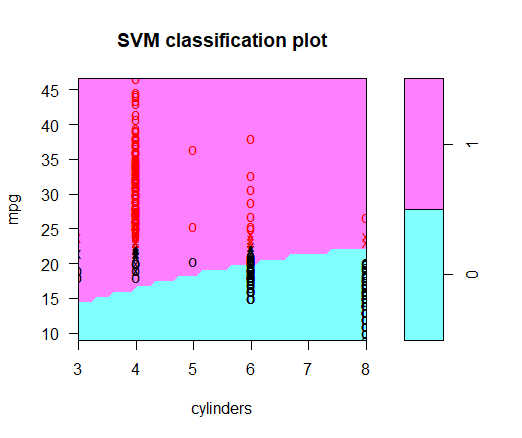
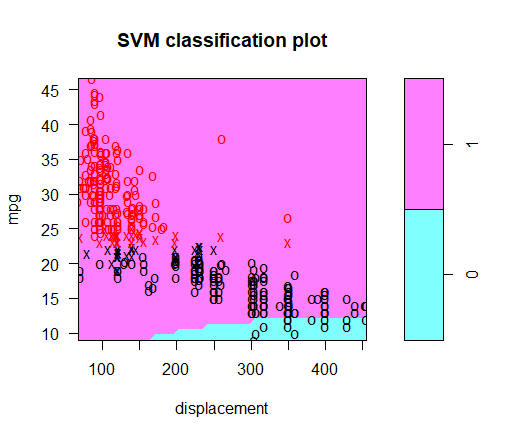
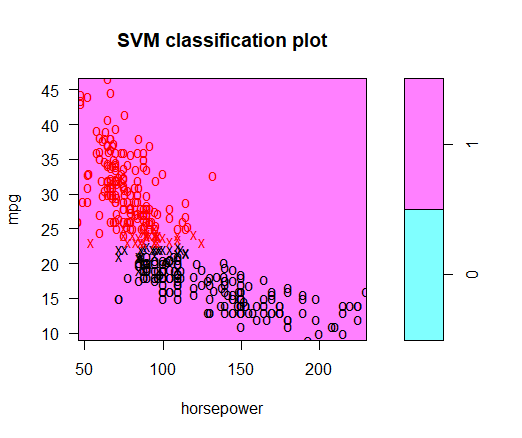
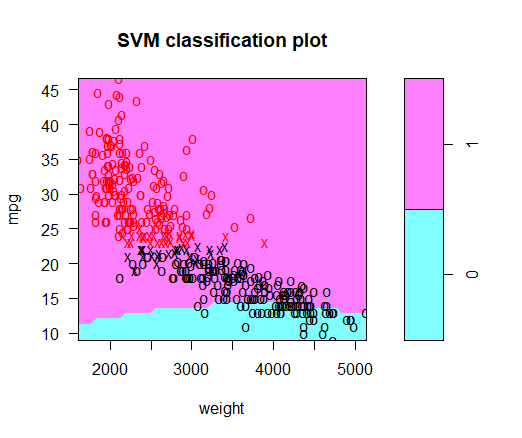
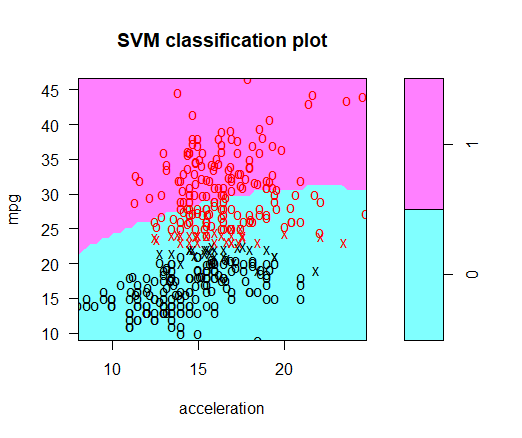
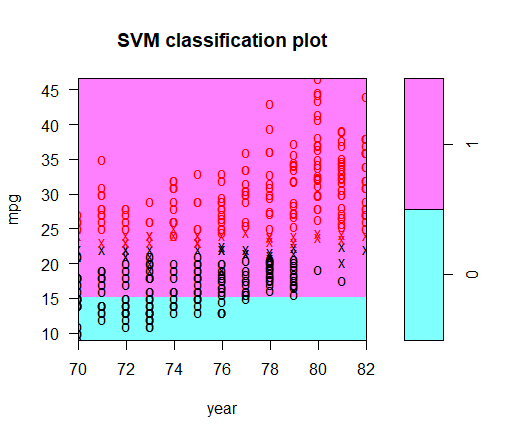
多项式核函数支持向量机：

> extand\_plot(svm.poly)



径向核函数支持向量机：

> extand\_plot(svm.radial)



**8(a)**

> library(ISLR)

> set.seed(98)

> train = sample(dim(OJ)[1], 800)

> OJ.train = OJ[train, ]

> OJ.test = OJ[-train, ]

**8(b)**

> library(e1071)

> svm.linear = svm(Purchase ~ ., kernel = "linear", data = OJ.train, cost = 0.01)

> summary(svm.linear)

Call:

svm(formula = Purchase ~ ., data = OJ.train, kernel = "linear", cost = 0.01)

Parameters:

SVM-Type: C-classification

SVM-Kernel: linear

cost: 0.01

gamma: 0.05555556

Number of Support Vectors: 426

( 213 213 )

Number of Classes: 2

Levels:

CH MM

支持向量有426个，CH和MM各有213个。

**8(c)**

> train.pred = predict(svm.linear, OJ.train)

> table(OJ.train$Purchase, train.pred)

train.pred

CH MM

CH 434 54

MM 72 240

> mean(OJ.train$Purchase != train.pred)

[1] 0.1575

>

> test.pred = predict(svm.linear, OJ.test)

> table(OJ.test$Purchase, test.pred)

test.pred

CH MM

CH 140 25

MM 30 75

> mean(OJ.test$Purchase != test.pred)

[1] 0.2037037

Training error rate为0.1575，test error rate为0.2037037。

**8(d)**

> set.seed(980)

> tune.out = tune(svm, Purchase ~ ., data = OJ.train, kernel = "linear", ranges = list(cost = seq(0.01,10, by = 0.1)))

> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:

cost

0.01

- best performance: 0.1625

- Detailed performance results:

cost error dispersion

1 0.01 0.16250 0.06319063

2 0.11 0.16750 0.05439056

3 0.21 0.16750 0.05277047

4 0.31 0.16750 0.05749396

5 0.41 0.16750 0.05688683

6 0.51 0.16750 0.05688683

7 0.61 0.16500 0.05797509

8 0.71 0.16500 0.05797509

9 0.81 0.16625 0.05834821

10 0.91 0.16625 0.05834821

11 1.01 0.16500 0.05552777

12 1.11 0.16750 0.05374838

13 1.21 0.16875 0.05278691

14 1.31 0.16750 0.05342440

15 1.41 0.16750 0.05342440

16 1.51 0.16750 0.05342440

17 1.61 0.16875 0.05504102

18 1.71 0.16875 0.05504102

19 1.81 0.16875 0.05504102

20 1.91 0.16875 0.05504102

21 2.01 0.16750 0.05596378

22 2.11 0.16750 0.05596378

23 2.21 0.16750 0.05596378

24 2.31 0.16625 0.05434266

25 2.41 0.16625 0.05434266

26 2.51 0.16625 0.05434266

27 2.61 0.16750 0.05596378

28 2.71 0.16875 0.05504102

29 2.81 0.16875 0.05504102

30 2.91 0.16875 0.05504102

31 3.01 0.16875 0.05504102

32 3.11 0.16875 0.05504102

33 3.21 0.16875 0.05504102

34 3.31 0.16875 0.05504102

35 3.41 0.16875 0.05504102

36 3.51 0.16875 0.05504102

37 3.61 0.16875 0.05504102

38 3.71 0.16875 0.05504102

39 3.81 0.16875 0.05504102

40 3.91 0.16875 0.05504102

41 4.01 0.16875 0.05504102

42 4.11 0.16750 0.05533986

43 4.21 0.16750 0.05533986

44 4.31 0.16875 0.05472469

45 4.41 0.16750 0.05533986

46 4.51 0.16750 0.05533986

47 4.61 0.16875 0.05472469

48 4.71 0.17000 0.05407043

49 4.81 0.17000 0.05407043

50 4.91 0.17000 0.05407043

51 5.01 0.17000 0.05407043

52 5.11 0.17000 0.05407043

53 5.21 0.17000 0.05407043

54 5.31 0.17125 0.05337563

55 5.41 0.17125 0.05337563

56 5.51 0.17125 0.05337563

57 5.61 0.17125 0.05337563

58 5.71 0.17000 0.05407043

59 5.81 0.17125 0.05337563

60 5.91 0.17000 0.05407043

61 6.01 0.17000 0.05407043

62 6.11 0.17000 0.05407043

63 6.21 0.17000 0.05407043

64 6.31 0.17000 0.05407043

65 6.41 0.17125 0.05205833

66 6.51 0.17125 0.05205833

67 6.61 0.17250 0.05027701

68 6.71 0.17250 0.05027701

69 6.81 0.17250 0.05027701

70 6.91 0.17250 0.05027701

71 7.01 0.17250 0.05027701

72 7.11 0.17250 0.05027701

73 7.21 0.17250 0.05027701

74 7.31 0.17250 0.05027701

75 7.41 0.17250 0.05027701

76 7.51 0.17250 0.05027701

77 7.61 0.17250 0.05027701

78 7.71 0.17250 0.05027701

79 7.81 0.17250 0.05027701

80 7.91 0.17250 0.05027701

81 8.01 0.17250 0.05027701

82 8.11 0.17250 0.05027701

83 8.21 0.17250 0.05027701

84 8.31 0.17250 0.05027701

85 8.41 0.17250 0.05027701

86 8.51 0.17250 0.05027701

87 8.61 0.17250 0.05027701

88 8.71 0.17250 0.05027701

89 8.81 0.17250 0.05027701

90 8.91 0.17250 0.05027701

91 9.01 0.17250 0.05027701

92 9.11 0.17250 0.05027701

93 9.21 0.17250 0.05027701

94 9.31 0.17250 0.05027701

95 9.41 0.17250 0.05027701

96 9.51 0.17250 0.05027701

97 9.61 0.17250 0.05027701

98 9.71 0.17250 0.05027701

99 9.81 0.17250 0.05027701

100 9.91 0.17250 0.05027701

cost取值为0.01时表现最好，错误率为0.16250。

**8(e)**

最优的cost还是0.01，略过。

**8(f)**

> svm.radial = svm(Purchase ~ ., data = OJ.train, kernel = "radial", cost = 0.01)

> summary(svm.radial)

Call:

svm(formula = Purchase ~ ., data = OJ.train, kernel = "radial", cost = 0.01)

Parameters:

SVM-Type: C-classification

SVM-Kernel: radial

cost: 0.01

gamma: 0.05555556

Number of Support Vectors: 628

( 312 316 )

Number of Classes: 2

Levels:

CH MM

支持向量有628个，其中CH有312个，MM有316个。

> train.pred = predict(svm.radial, OJ.train)

> table(OJ.train$Purchase, train.pred)

train.pred

CH MM

CH 488 0

MM 312 0

> mean(OJ.train$Purchase != train.pred)

[1] 0.39

>

> test.pred = predict(svm.radial, OJ.test)

> table(OJ.test$Purchase, test.pred)

test.pred

CH MM

CH 165 0

MM 105 0

> mean(OJ.test$Purchase != test.pred)

[1] 0.3888889

Training error rate为0.39，test error rate为0.3888889。

> set.seed(9800)

> tune.out = tune(svm, Purchase ~ ., data = OJ.train, kernel = "radial", ranges = list(cost = seq(0.01,10, by = 0.1)))

> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:

cost

0.51

- best performance: 0.1675

- Detailed performance results:

cost error dispersion

1 0.01 0.39000 0.05916080

2 0.11 0.17250 0.03216710

3 0.21 0.17750 0.03944053

4 0.31 0.17000 0.03593976

5 0.41 0.17000 0.03593976

6 0.51 0.16750 0.03917553

7 0.61 0.16875 0.03830162

8 0.71 0.17000 0.03917553

9 0.81 0.17125 0.04084609

10 0.91 0.17125 0.03821086

11 1.01 0.17375 0.04143687

12 1.11 0.17625 0.03606033

13 1.21 0.17500 0.03864008

14 1.31 0.17500 0.03864008

15 1.41 0.17750 0.03374743

16 1.51 0.17625 0.03458584

17 1.61 0.17625 0.03458584

18 1.71 0.17500 0.03333333

19 1.81 0.17500 0.03333333

20 1.91 0.17625 0.03356689

21 2.01 0.18125 0.03240906

22 2.11 0.18125 0.03240906

23 2.21 0.18125 0.03240906

24 2.31 0.18000 0.03129164

25 2.41 0.18000 0.03291403

26 2.51 0.18125 0.03448530

27 2.61 0.18250 0.03496029

28 2.71 0.18125 0.03346329

29 2.81 0.18250 0.03343734

30 2.91 0.18250 0.03343734

31 3.01 0.18250 0.03343734

32 3.11 0.18375 0.03387579

33 3.21 0.18375 0.03387579

34 3.31 0.18375 0.03387579

35 3.41 0.18375 0.03387579

36 3.51 0.18500 0.03574602

37 3.61 0.18625 0.03653860

38 3.71 0.18625 0.03653860

39 3.81 0.18750 0.03864008

40 3.91 0.18625 0.03356689

41 4.01 0.18625 0.03356689

42 4.11 0.18625 0.03356689

43 4.21 0.18625 0.03356689

44 4.31 0.18625 0.03356689

45 4.41 0.18625 0.03356689

46 4.51 0.18625 0.03356689

47 4.61 0.18750 0.03173239

48 4.71 0.18750 0.03173239

49 4.81 0.18875 0.03251602

50 4.91 0.18875 0.03251602

51 5.01 0.18875 0.03251602

52 5.11 0.19125 0.03438447

53 5.21 0.19000 0.03322900

54 5.31 0.19000 0.03322900

55 5.41 0.19000 0.03322900

56 5.51 0.19000 0.03322900

57 5.61 0.19000 0.03322900

58 5.71 0.19000 0.03322900

59 5.81 0.19125 0.03387579

60 5.91 0.19250 0.03593976

61 6.01 0.19250 0.03593976

62 6.11 0.19500 0.03736085

63 6.21 0.19500 0.03736085

64 6.31 0.19500 0.03736085

65 6.41 0.19500 0.03736085

66 6.51 0.19500 0.03736085

67 6.61 0.19500 0.03736085

68 6.71 0.19500 0.03736085

69 6.81 0.19375 0.03691676

70 6.91 0.19375 0.03691676

71 7.01 0.19375 0.03691676

72 7.11 0.19375 0.03691676

73 7.21 0.19375 0.03691676

74 7.31 0.19375 0.03691676

75 7.41 0.19375 0.03691676

76 7.51 0.19375 0.03691676

77 7.61 0.19500 0.03496029

78 7.71 0.19500 0.03496029

79 7.81 0.19500 0.03496029

80 7.91 0.19500 0.03496029

81 8.01 0.19500 0.03496029

82 8.11 0.19500 0.03496029

83 8.21 0.19500 0.03496029

84 8.31 0.19500 0.03496029

85 8.41 0.19500 0.03496029

86 8.51 0.19500 0.03496029

87 8.61 0.19500 0.03496029

88 8.71 0.19500 0.03496029

89 8.81 0.19500 0.03496029

90 8.91 0.19500 0.03496029

91 9.01 0.19625 0.03335936

92 9.11 0.19625 0.03335936

93 9.21 0.19625 0.03335936

94 9.31 0.19625 0.03335936

95 9.41 0.19625 0.03335936

96 9.51 0.19750 0.03322900

97 9.61 0.19750 0.03322900

98 9.71 0.19750 0.03322900

99 9.81 0.19750 0.03322900

100 9.91 0.19750 0.03322900

最优的cost取值为0.51，对应的错误率为0.1675。

> svm.radial = svm(Purchase ~ ., data = OJ.train, kernel = "radial", cost = 0.51)

>

> train.pred = predict(svm.radial, OJ.train)

> table(OJ.train$Purchase, train.pred)

train.pred

CH MM

CH 444 44

MM 72 240

> mean(OJ.train$Purchase != train.pred)

[1] 0.145

>

> test.pred = predict(svm.radial, OJ.test)

> table(OJ.test$Purchase, test.pred)

test.pred

CH MM

CH 142 23

MM 31 74

> mean(OJ.test$Purchase != test.pred)

[1] 0.2

Training error rate为0.145，test error rate为0.2。

**8(g)**

> svm.poly = svm(Purchase ~ ., data = OJ.train, kernel = "polynomial", cost = 0.01,degree = 2)

> summary(svm.poly)

Call:

svm(formula = Purchase ~ ., data = OJ.train, kernel = "polynomial", cost = 0.01, degree = 2)

Parameters:

SVM-Type: C-classification

SVM-Kernel: polynomial

cost: 0.01

degree: 2

gamma: 0.05555556

coef.0: 0

Number of Support Vectors: 631

( 312 319 )

Number of Classes: 2

Levels:

CH MM

支持向量有631个，其中CH有312个，MM有319个。

> train.pred = predict(svm.poly, OJ.train)

> table(OJ.train$Purchase, train.pred)

train.pred

CH MM

CH 488 0

MM 312 0

> mean(OJ.train$Purchase != train.pred)

[1] 0.39

>

> test.pred = predict(svm.poly, OJ.test)

> table(OJ.test$Purchase, test.pred)

test.pred

CH MM

CH 165 0

MM 105 0

> mean(OJ.test$Purchase != test.pred)

[1] 0.3888889

Training error rate为0.39，test error rate为0.3888889。

> set.seed(98000)

> tune.out = tune(svm, Purchase ~ ., data = OJ.train, kernel = "polynomial", ranges = list(cost = seq(0.01,10, by = 0.1)),degree = 2)

> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:

cost

8.61

- best performance: 0.185

- Detailed performance results:

cost error dispersion

1 0.01 0.39000 0.06422616

2 0.11 0.32375 0.05350558

3 0.21 0.22750 0.05767485

4 0.31 0.21000 0.05130248

5 0.41 0.20500 0.05244044

6 0.51 0.20125 0.05541823

7 0.61 0.20000 0.05921946

8 0.71 0.19625 0.05714565

9 0.81 0.19250 0.05779514

10 0.91 0.19500 0.05596378

11 1.01 0.19500 0.05439056

12 1.11 0.19500 0.05596378

13 1.21 0.19375 0.05311479

14 1.31 0.19375 0.04759858

15 1.41 0.19250 0.04721405

16 1.51 0.19250 0.04684490

17 1.61 0.18875 0.04505013

18 1.71 0.18750 0.04823265

19 1.81 0.18625 0.04875178

20 1.91 0.18875 0.04656611

21 2.01 0.19000 0.04706674

22 2.11 0.19250 0.05109903

23 2.21 0.19000 0.05329426

24 2.31 0.19000 0.05197489

25 2.41 0.19000 0.05197489

26 2.51 0.18875 0.05152197

27 2.61 0.19000 0.05361903

28 2.71 0.18625 0.05219155

29 2.81 0.18750 0.05270463

30 2.91 0.18625 0.05185785

31 3.01 0.18750 0.05237419

32 3.11 0.18875 0.05084358

33 3.21 0.18875 0.05084358

34 3.31 0.18875 0.05084358

35 3.41 0.18875 0.05219155

36 3.51 0.19000 0.05263871

37 3.61 0.19125 0.05337563

38 3.71 0.19000 0.05163978

39 3.81 0.19125 0.05205833

40 3.91 0.19000 0.05096295

41 4.01 0.18875 0.05050096

42 4.11 0.18875 0.05050096

43 4.21 0.18875 0.05050096

44 4.31 0.18875 0.05050096

45 4.41 0.19000 0.04779877

46 4.51 0.19000 0.04779877

47 4.61 0.19000 0.04779877

48 4.71 0.18875 0.05285265

49 4.81 0.18750 0.05303301

50 4.91 0.18750 0.05303301

51 5.01 0.18750 0.05303301

52 5.11 0.18750 0.05303301

53 5.21 0.18750 0.05303301

54 5.31 0.18750 0.05303301

55 5.41 0.18750 0.05303301

56 5.51 0.18750 0.05303301

57 5.61 0.18750 0.05303301

58 5.71 0.19000 0.05394184

59 5.81 0.19125 0.05622685

60 5.91 0.19125 0.05622685

61 6.01 0.19125 0.05622685

62 6.11 0.19125 0.05622685

63 6.21 0.19125 0.05622685

64 6.31 0.19000 0.05583955

65 6.41 0.19000 0.05583955

66 6.51 0.19000 0.05583955

67 6.61 0.19000 0.05583955

68 6.71 0.19125 0.05622685

69 6.81 0.19000 0.05737305

70 6.91 0.19000 0.05737305

71 7.01 0.19000 0.05737305

72 7.11 0.19000 0.05737305

73 7.21 0.19000 0.05737305

74 7.31 0.19125 0.05775006

75 7.41 0.19125 0.05775006

76 7.51 0.19125 0.05775006

77 7.61 0.19125 0.05775006

78 7.71 0.19125 0.05775006

79 7.81 0.19125 0.05775006

80 7.91 0.19125 0.05775006

81 8.01 0.19125 0.05775006

82 8.11 0.19000 0.05767485

83 8.21 0.18750 0.05621141

84 8.31 0.18625 0.05696307

85 8.41 0.18750 0.05773503

86 8.51 0.18625 0.05905800

87 8.61 0.18500 0.06061032

88 8.71 0.18625 0.06136469

89 8.81 0.18625 0.06136469

90 8.91 0.18625 0.06136469

91 9.01 0.18625 0.06136469

92 9.11 0.18500 0.06286007

93 9.21 0.18500 0.06286007

94 9.31 0.18500 0.06286007

95 9.41 0.18500 0.06286007

96 9.51 0.18500 0.06286007

97 9.61 0.18625 0.06467064

98 9.71 0.18625 0.06467064

99 9.81 0.18625 0.06467064

100 9.91 0.18625 0.06467064

最优的cost为8.61，对应的错误率0.185。

> svm.poly = svm(Purchase ~ ., data = OJ.train, kernel = "polynomial", cost = 8.61,degree = 2)

> train.pred = predict(svm.poly, OJ.train)

> table(OJ.train$Purchase, train.pred)

train.pred

CH MM

CH 450 38

MM 84 228

> mean(OJ.train$Purchase != train.pred)

[1] 0.1525

>

> test.pred = predict(svm.poly, OJ.test)

> table(OJ.test$Purchase, test.pred)

test.pred

CH MM

CH 144 21

MM 34 71

> mean(OJ.test$Purchase != test.pred)

[1] 0.2037037

Training error rate为0.1525，test error rate为0. 2037037。

**8(h)**

径向核函数支持向量机能得到最好的结果。