# SVM-HW

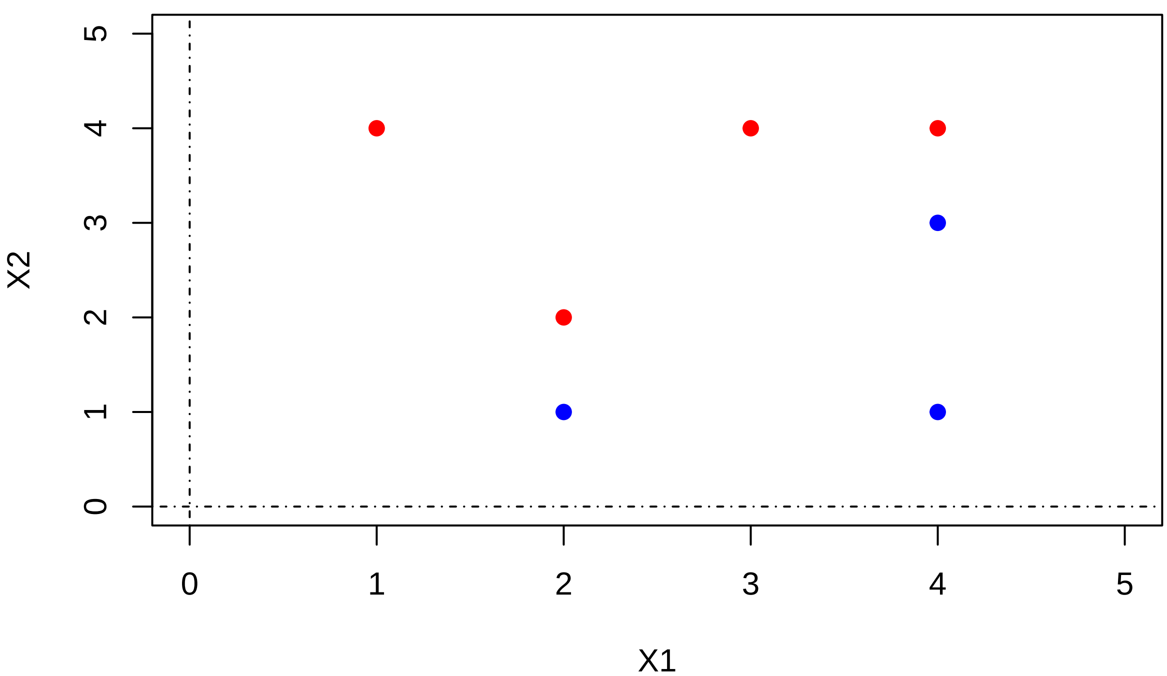
Franky Zhang

3/8/2022

### 9.3

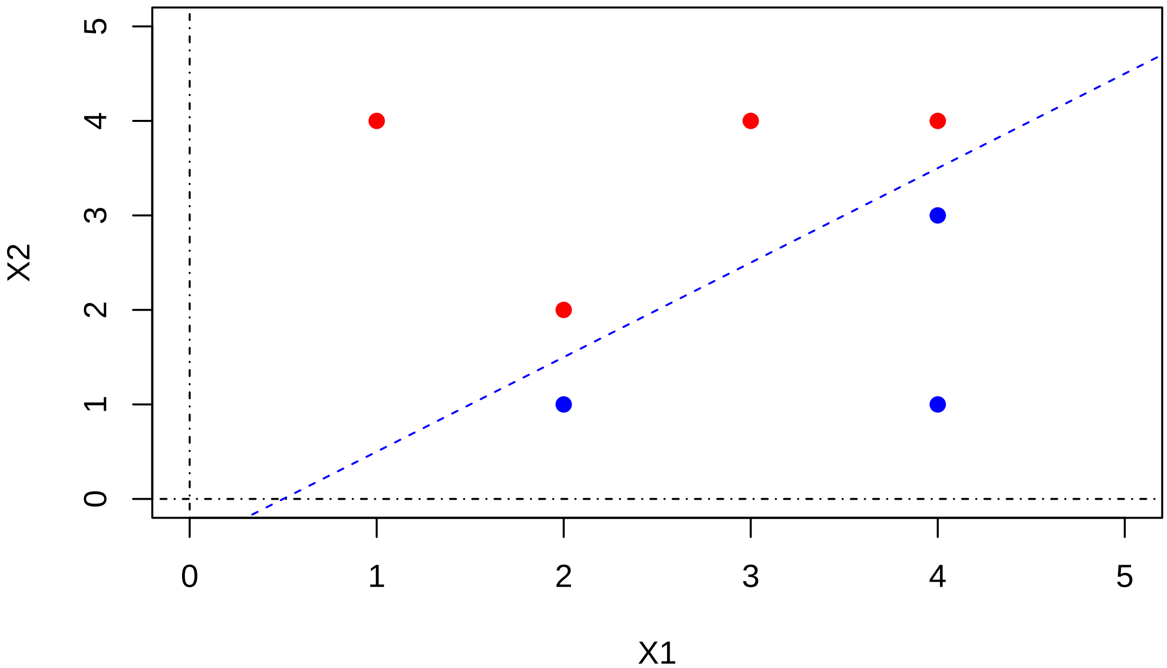
#### (a)

```
dat <- data.frame(
  X1 = c(3, 2, 4, 1, 2, 4, 4),
  X2 = c(4, 2, 4, 4, 1, 3, 1),
  Y = c("red", "red", "red", "red",
               "blue", "blue", "blue")
)
plot(dat[, -3], col = dat$Y, pch = 19, xlim = c(0, 5), ylim = c(0, 5))
abline(h=0,v=0,lty=4)
```




#### (b)

```r
svm.fit <- svm(factor(Y)~., data = dat, type = "C-classification",  kernel = "linear", scale = FALSE)
plot(dat[, -3], col = dat$Y, pch = 19, xlim = c(0, 5), ylim = c(0, 5))
abline(h=0,v=0,lty=4)
# in 2D space, the hyperplane is the line w[1, 1]*x1 + w[1, 2]*x2 + b = 0
abline(a = -.5, b = 1, col="blue", lty=2, lwd = 1)
```



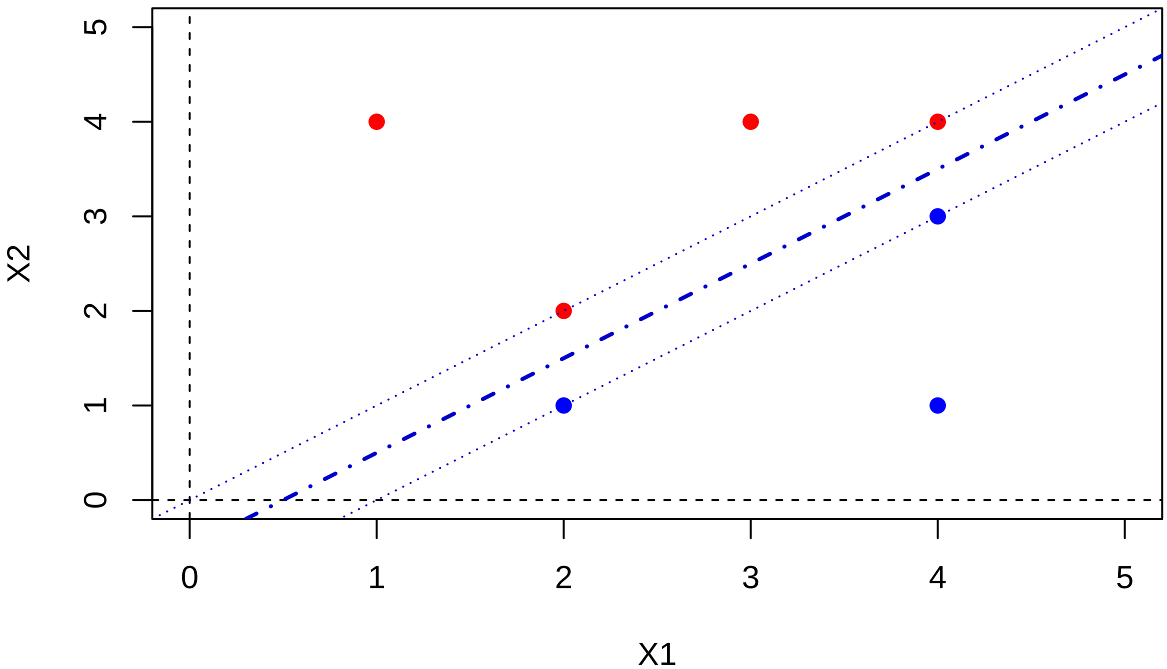

**(c)**

*Answer*:

the classifier is $f(X) = -X_1 + X_2 + 0.5$. when $f(X) < 0$, the observation is classified to Blue; otherwise, Red.

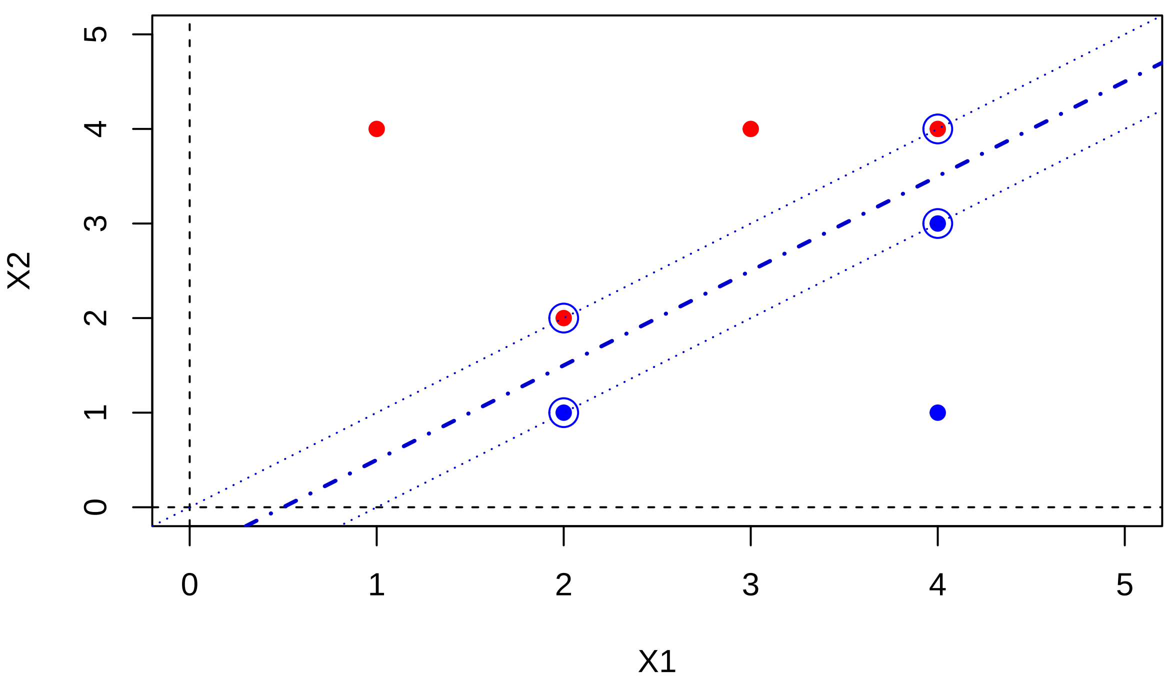
**(d)**

```r
plot(dat[, -3], col = dat$Y, pch = 19, xlim = c(0, 5), ylim = c(0, 5))
abline(h=0,v=0,lty=2)
# in 2D space, the hyperplane is the line w[1, 1]*x1 + w[1, 2]*x2 + b = 0
abline(a = -.5, b = 1, col="blue3", lty=4, lwd = 2)
abline(a = 0  , b = 1, col="blue3", lty=3, lwd = 1)
abline(a = -1 , b = 1, col="blue3", lty=3, lwd = 1)
```




(e)

```
plot(dat[, -3], col = dat$Y, pch = 19, xlim = c(0, 5), ylim = c(0, 5))
abline(h=0,v=0,lty=2)
# in 2D space, the hyperplane is the line w[1, 1]*x1 + w[1, 2]*x2 + b = 0
abline(a = -.5, b = 1, col="blue3", lty=4, lwd = 2)
abline(a = 0  , b = 1, col="blue3", lty=3, lwd = 1)
abline(a = -1 , b = 1, col="blue3", lty=3, lwd = 1)
points(dat[svm.fit$index, c(1, 2)], col = "blue", cex = 2) # circle the support vectors
```



**(f)**

```
svm.fit1 <- svm(factor(Y)~., data = dat, type = "C-classification",  kernel = "linear", scale = FALSE)
svm.fit2 <- svm(factor(Y)~., data = dat[-7, ], type = "C-classification",  kernel = "linear", scale = FA
svm.fit1$SV
```

```
##   X1 X2
## 2  2  2
## 3  4  4
## 5  2  1
## 6  4  3
```

```
svm.fit2$SV
```

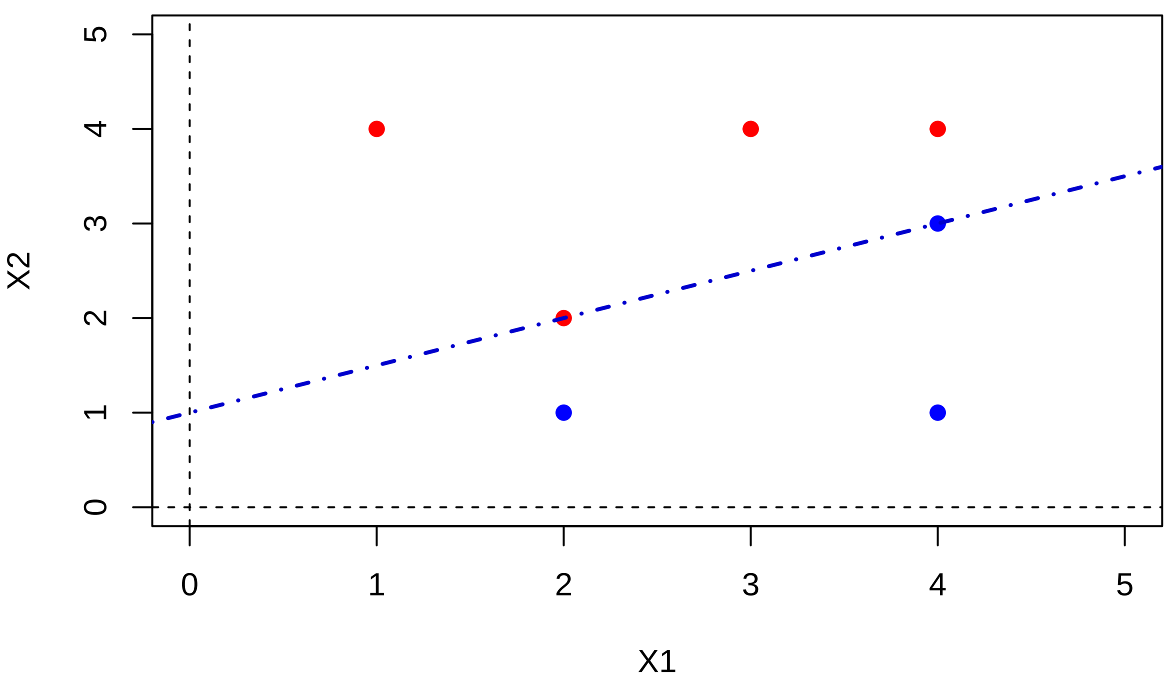
```
##   X1 X2
## 2  2  2
## 3  4  4
## 5  2  1
## 6  4  3
```

*Answer*:

Support vectors of these two models are exactly the same.

**(g)**

```
plot(dat[, -3], col = dat$Y, pch = 19, xlim = c(0, 5), ylim = c(0, 5))
abline(h=0,v=0,lty=2)
# in 2D space, the hyperplane is the line w[1, 1]*x1 + w[1, 2]*x2 + b = 0
abline(a = 1, b = .5, col="blue3", lty=4, lwd = 2)
```



*Answer*:

in this case, the width of margin = 0.

### (h)

*Answer*:

Add a point (X1 = 2, X2 = 4, Y = "blue")

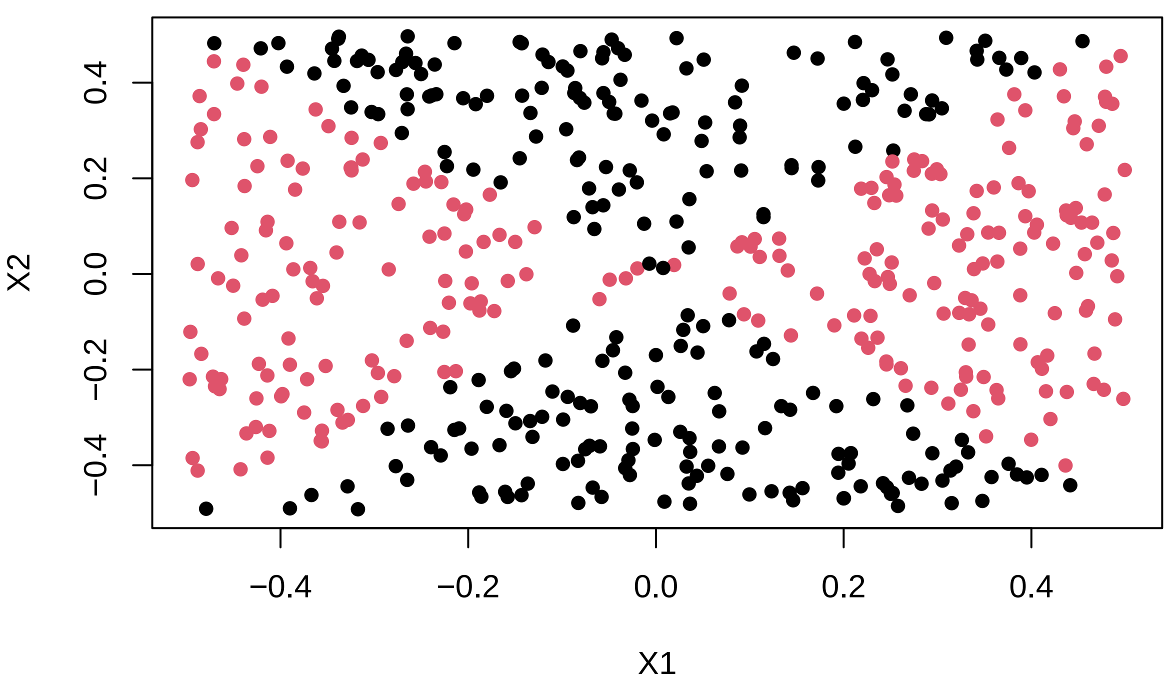
## 9.5

### (a)

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

### (b)

```
dat <- data.frame(
  X1 = x1,
  X2 = x2,
  Y = factor(y)
)
plot(dat[, -3], col = dat$Y, pch = 16)
```



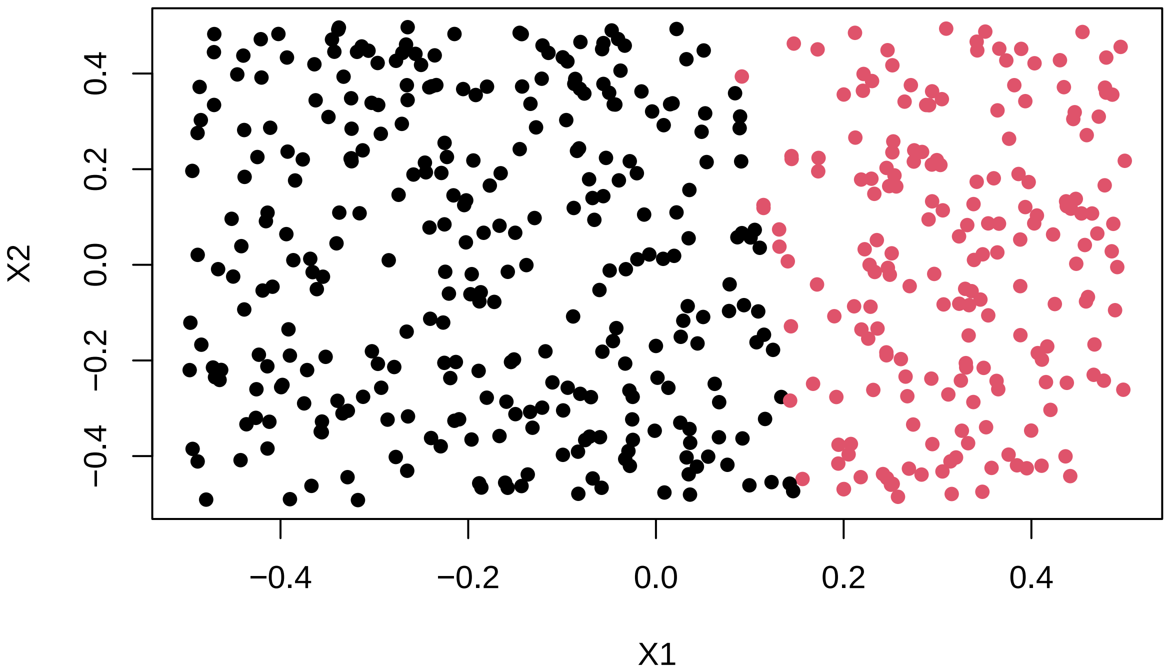
**(c)**

```
log.fit <- glm(Y~., data = dat, family = binomial(link = "logit"))
summary(log.fit)
```

```
##
## Call:
## glm(formula = Y ~ ., family = binomial(link = "logit"), data = dat)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.236  -1.157  -1.100   1.174   1.279
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.04361    0.08967  -0.486    0.627
## X1           0.37175    0.31226   1.191    0.234
## X2           0.02518    0.30161   0.083    0.933
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 692.95  on 499  degrees of freedom
## Residual deviance: 691.52  on 497  degrees of freedom
## AIC: 697.52
##
## Number of Fisher Scoring iterations: 3
```

**(d)**

```
prob.log <- predict(log.fit, newdata = dat, type = "response")
pred.log <- factor(ifelse(prob.log > .5, 1, 0))
plot(dat[, -3], col = pred.log, pch = 16)
```



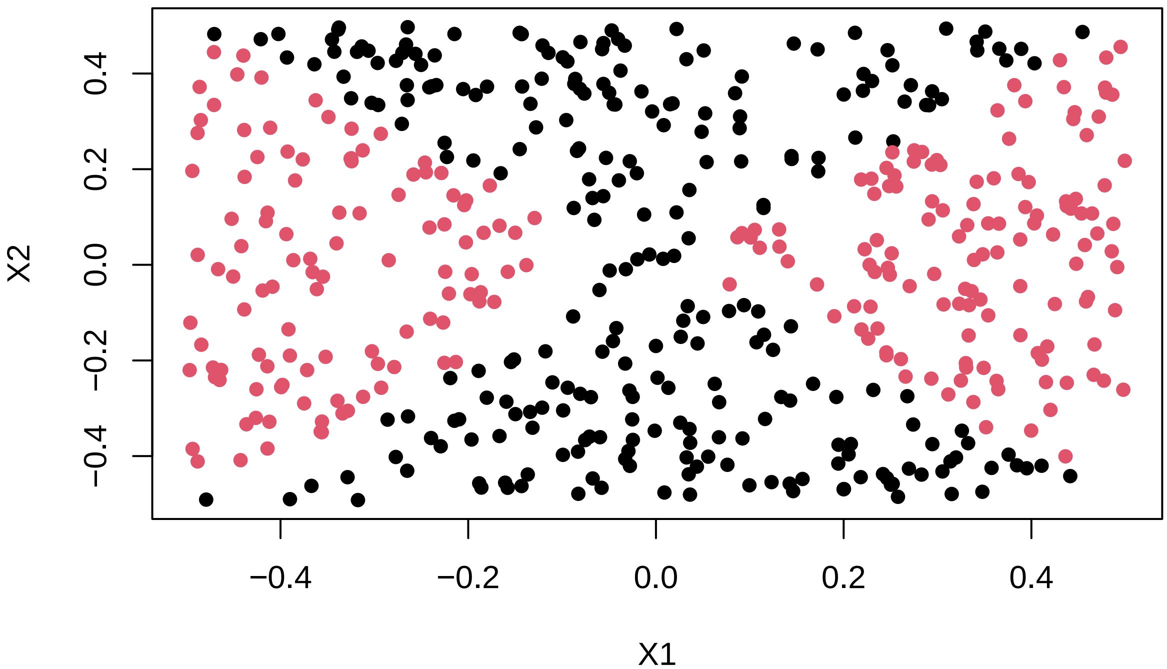

**(e)**

```
log.fit <- glm(Y~ X1 + X2 + I(X1^2) + I(X2^2) + X1:X2, data = dat, family = binomial(link = "logit"))
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

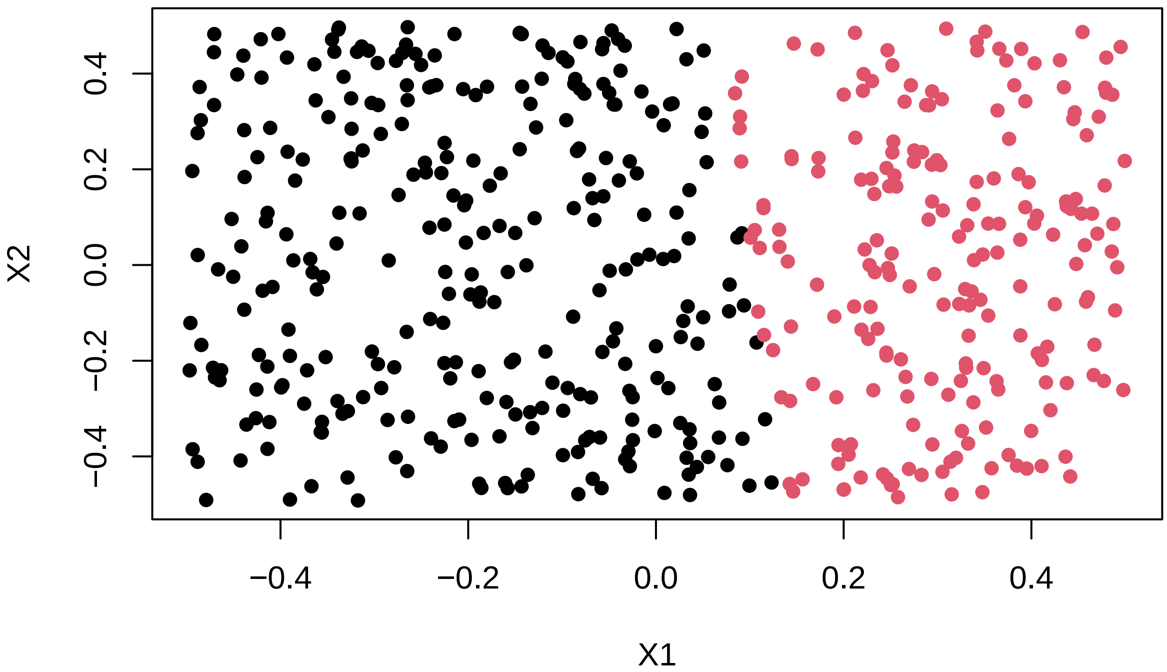
**(f)**

```
prob.log <- predict(log.fit, newdata = dat, type = "response")
pred.log <- factor(ifelse(prob.log > .5, 1, 0))
plot(dat[, -3], col = pred.log, pch = 16)
```



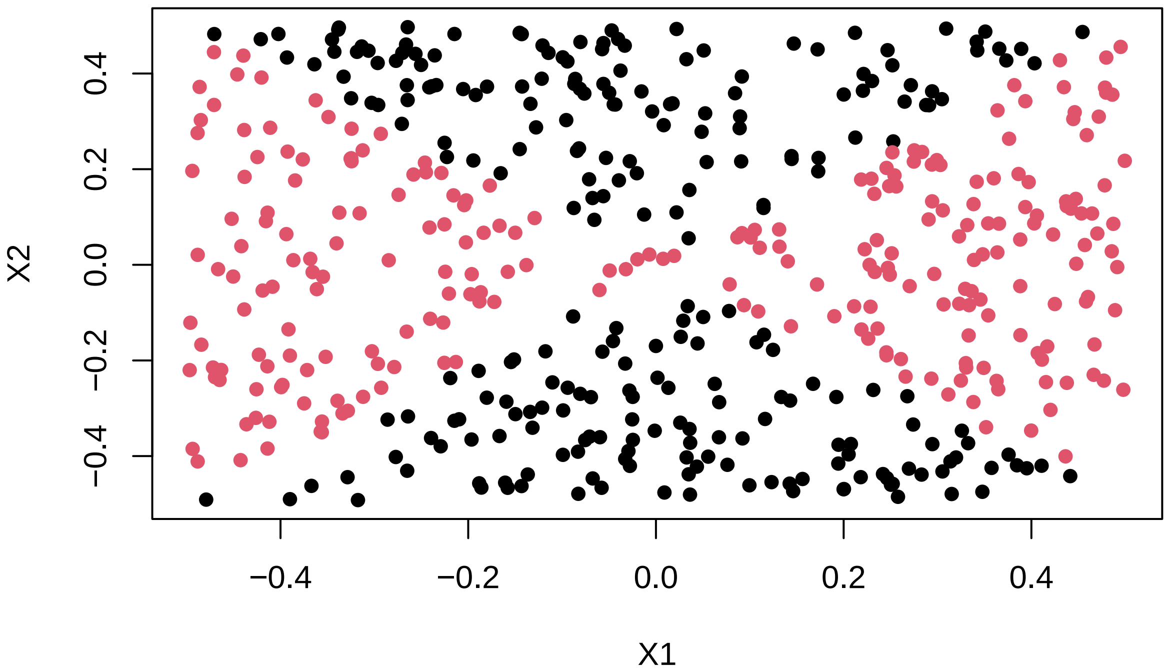

**(g)**

```r
tune.out.linear <- tune(svm, Y~., data = dat,
                        kernel = "linear",
                        ranges = list(
                          cost = c(0.1, 1, 10, 100, 1000)
                        ))
# plot(tune.out.linear$best.model, dat)
pred.svm.linear <- predict(tune.out.linear$best.model, newdata = dat, type = "class")
plot(dat[, -3], col = pred.svm.linear, pch = 16)
```




**(h)**

```
tune.out.radial <- tune(svm, Y~., data = dat,
                        kernel = "radial",
                        ranges = list(
                          cost = c(0.1, 1, 10, 100, 1000),
                          gamma = c(.5, 1, 2, 3, 4)
                        ))
pred.svm.radial <- predict(tune.out.radial$best.model, newdata = dat, type = "class")
plot(dat[, -3], col = pred.svm.radial, pch = 16)
```




**(i)**

whatever the method we use, the prediction boundary greatly depends on the feature space. With more flexible feature space, even logistic regression can grab nonliear and complex boundary.

## 9.7

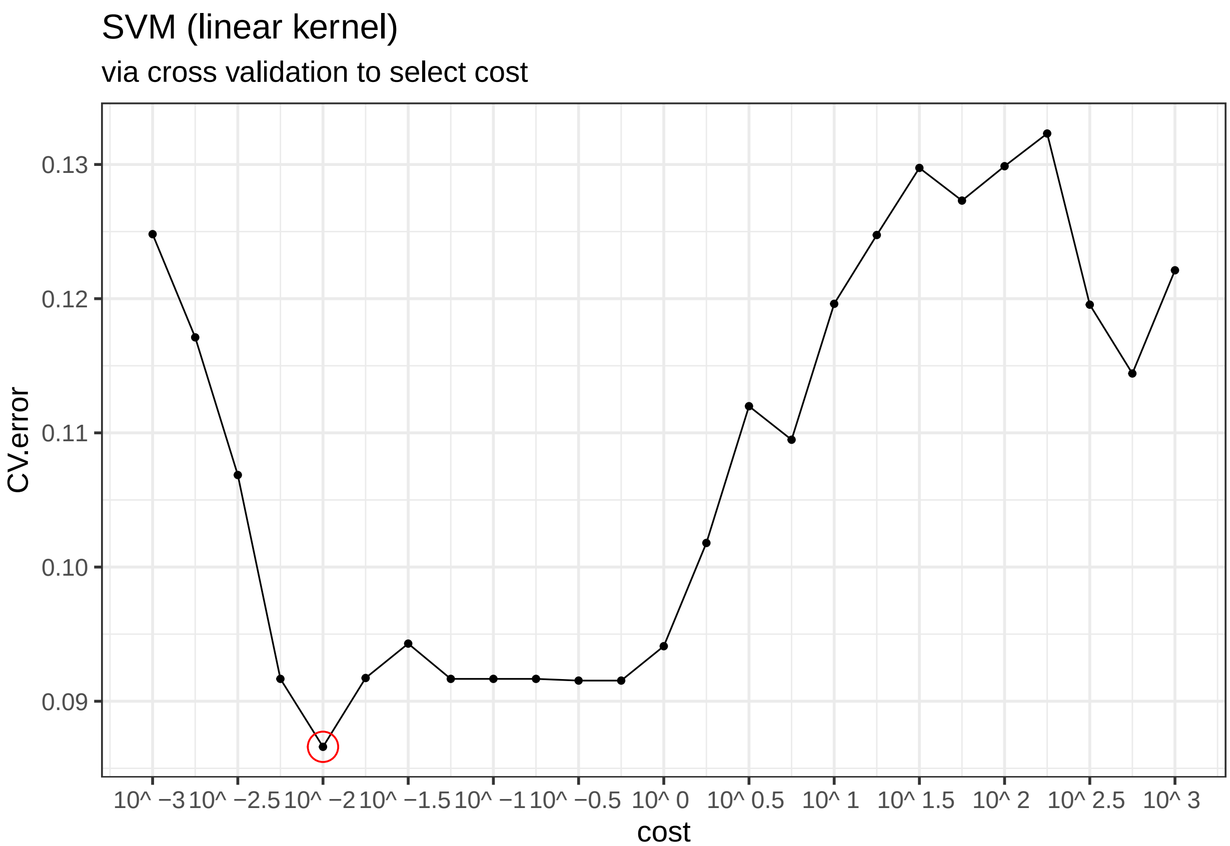
**(a)**

```
dat <- Auto
dat$mpg <- factor(ifelse(dat$mpg > median(Auto$mpg), 1, 0))
```

**(b)**

```
power_range <- seq(-3, 3, by = .25)
cost_range  <- 10^power_range
power_grid <- power_range[seq(1, length(power_range), 2)]
cost_grid  <- cost_range[seq(1, length(cost_range), 2)]
set.seed(2317)
tune.out <- tune(svm, mpg~., data = dat,
                 kernel = "linear",
                 ranges = list(
                   cost = cost_range
                 ))
plot.data <- data.frame(cost = cost_range, CV.error = tune.out$performances$error)
ggplot(plot.data, aes(x = cost, y = CV.error)) +
  geom_point(size = .8) + geom_line(lwd = .3) +
```

```r
  geom_point(data = plot.data[which.min(plot.data$CV.error), ],
             pch = 21, size = 5, color = "red") +
  scale_x_continuous(trans = "log10",
                     breaks = cost_grid,
                     labels = paste("10^", power_grid, seq = "")) + theme_bw() +
  labs(title = "SVM (linear kernel)",
       subtitle = "via cross validation to select cost")
```




*Comments*:

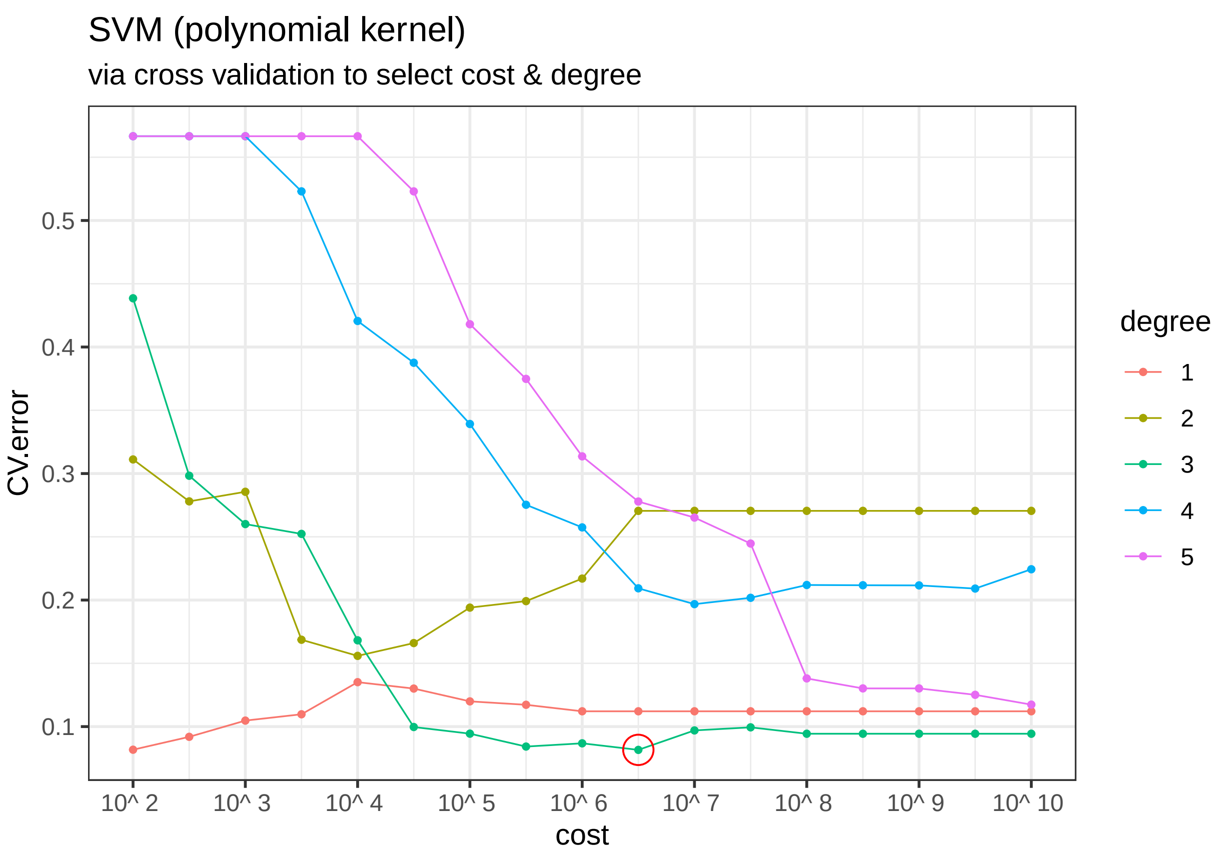
when $cost = \frac{1}{10^2}$, the linear kernel SVM performs best.

**(c)**

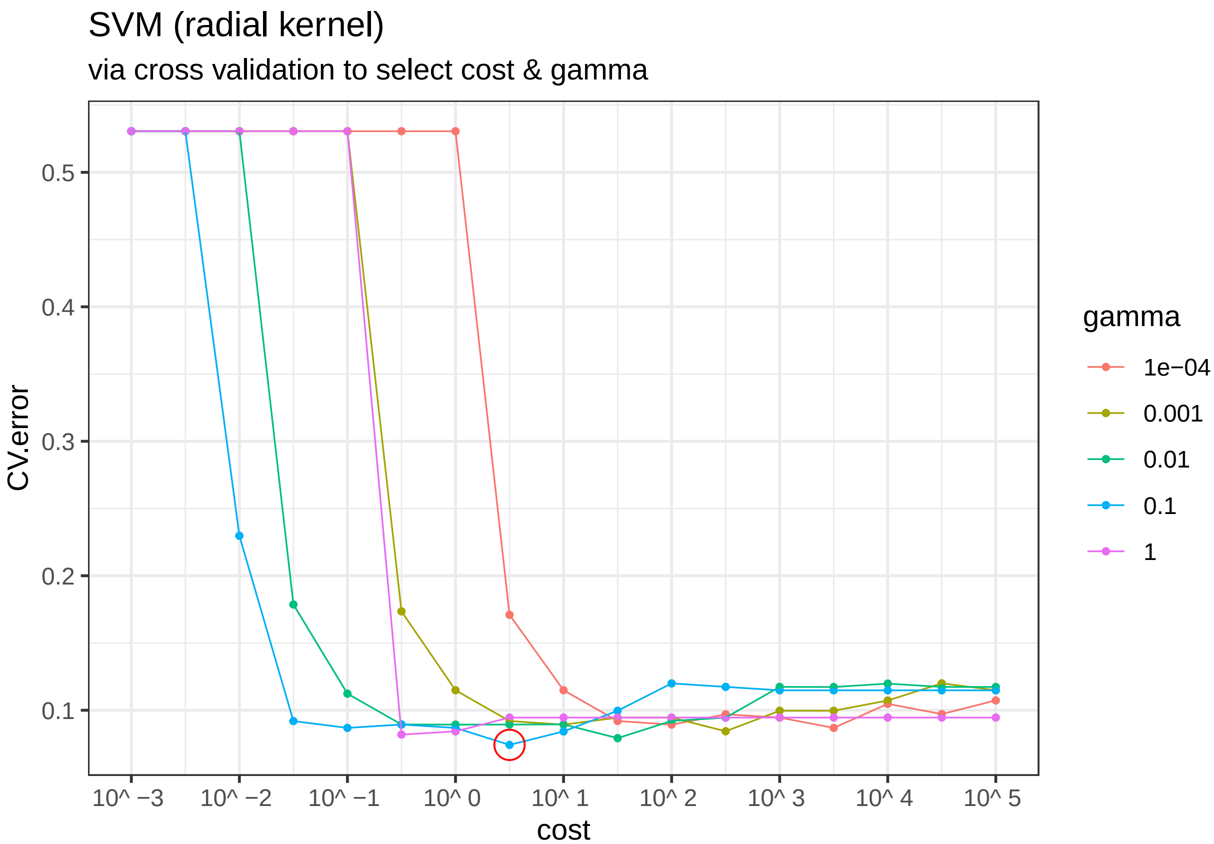
```r
# polynomial kernel
power_range <- seq(2, 10, by = .5)
cost_range  <- 10^power_range
power_grid <- power_range[seq(1, length(power_range), 2)]
cost_grid  <- cost_range[seq(1, length(cost_range), 2)]
degree_range <- 1:5
set.seed(2340)
tune.out.poly <- tune(svm, mpg~., data = dat,
                      kernel = "polynomial",
                      ranges = list(
                        cost = cost_range,
```

```r
                        degree = degree_range
                      ))
plot.data.poly <- data.frame(cost     = tune.out.poly$performances[, 1],
                             degree   = factor(tune.out.poly$performances[, 2]),
                             CV.error = tune.out.poly$performances[, 3])
plot.poly <- ggplot(plot.data.poly, aes(x = cost, y = CV.error, color = degree)) +
  geom_point(size = .8) + geom_line(lwd = .3) +
  geom_point(data = plot.data.poly[which.min(plot.data.poly$CV.error), ],
             pch = 21, size = 5, color = "red") +
  scale_x_continuous(trans = "log10",
                     breaks = cost_grid,
                     labels = paste("10^", power_grid, seq = "")) + theme_bw() +
  labs(title = "SVM (polynomial kernel)",
       subtitle = "via cross validation to select cost & degree")


# Radial kernel
power_range <- seq(-3, 5, by = .5)
power_grid <- power_range[seq(1, length(power_range), 2)]
cost_range  <- 10^power_range
cost_grid  <- cost_range[seq(1, length(cost_range), 2)]
gamma_degree <- -4:0
gamma_range <- 10^gamma_degree
set.seed(2347)
tune.out.radial <- tune(svm, mpg~., data = dat,
                      kernel = "radial",
                      ranges = list(
                        cost = cost_range,
                        gamma = gamma_range
                      ))
plot.data.radial <- data.frame(cost     = tune.out.radial$performances[, 1],
                               gamma    = factor(tune.out.radial$performances[, 2]),
                               CV.error = tune.out.radial$performances[, 3])
plot.radial <- ggplot(plot.data.radial, aes(x = cost, y = CV.error, color = gamma)) +
  geom_point(size = .8) + geom_line(lwd = .3) +
  geom_point(data = plot.data.radial[which.min(plot.data.radial$CV.error), ],
             pch = 21, size = 5, color = "red") +
  scale_x_continuous(trans = "log10",
                     breaks = cost_grid,
                     labels = paste("10^", power_grid, seq = "")) + theme_bw()+
  labs(title = "SVM (radial kernel)",
       subtitle = "via cross validation to select cost & gamma")
plot.poly
```


SVM (polynomial kernel)
via cross validation to select cost & degree
CV.error
0.5
0.4
0.3
0.2
0.1
10^ 2
10^ 3
10^ 4
10^ 5
10^ 6
10^ 7
10^ 8
10^ 9
10^ 10
cost
degree
1
2
3
4
5

```
plot.radial
```




*Comments*:

The parameter selection for polynomial kernel and radial kernel are as follows:

```
poly.parameter <- tune.out.poly$performances[which.min(
  tune.out.poly$performances$error), ]
radial.parameter <- tune.out.radial$performances[which.min(
  tune.out.radial$performances$error), ]
poly.parameter
```

```
##       cost degree      error dispersion
## 44 3162278      3 0.08160256 0.02001271
```

```
radial.parameter
```

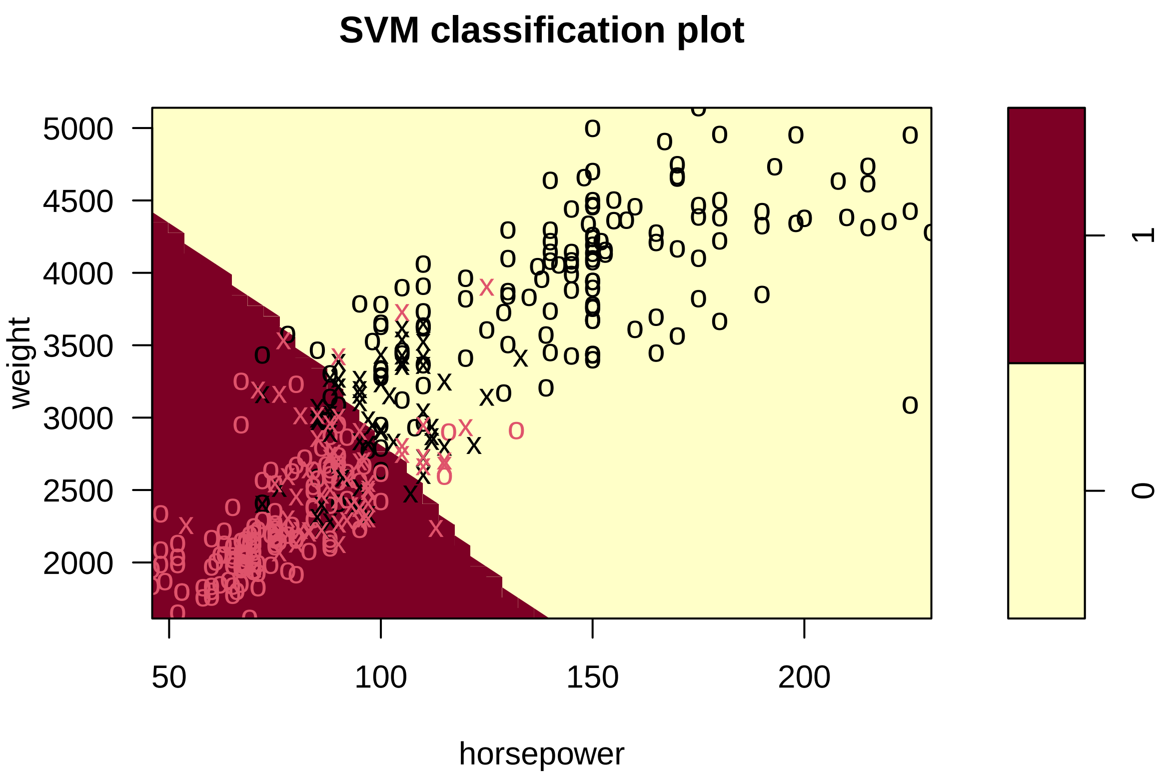
```
##        cost gamma      error dispersion
## 59 3.162278   0.1 0.07423077 0.04916648
```

**(d)**

```
# from r studio example
getmode <- function(v) {
   uniqv <- unique(v)
   uniqv[which.max(tabulate(match(v, uniqv)))]
}
```

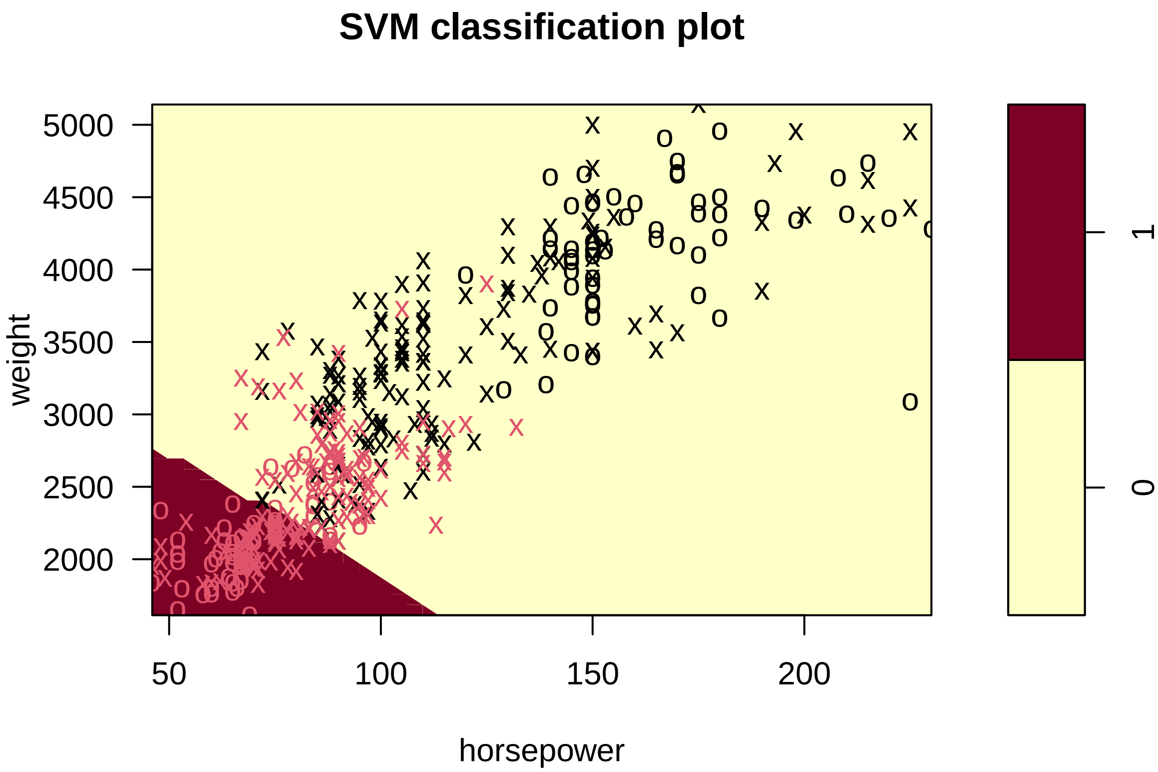
```r
# ?plot.svm # uss slice() to hold other dimension constant

# best polynomial kernel model
svm.poly <- svm(mpg~., data = dat, kernel = "polynomial",
                cost = poly.parameter[1, 1],
                degree = poly.parameter[1, 2], scale = TRUE)
plot(svm.poly, dat, weight ~ horsepower,
     slice = list(
       cylinders = median(dat$cylinders),
       displacement = median(dat$displacement),
       acceleration = median(dat$acceleration),
       year = median(dat$year),
       origin = getmode(dat$origin),
       name = getmode(dat$name)
     ))
```




```r
# comparsion polynomial kernel model
svm.poly <- svm(mpg~., data = dat, kernel = "polynomial",
                cost = poly.parameter[1, 1],
                degree = poly.parameter[1, 2] + 1, scale = TRUE)
plot(svm.poly, dat, weight ~ horsepower,
     slice = list(
       cylinders = median(dat$cylinders),
       displacement = median(dat$displacement),
       acceleration = median(dat$acceleration),
       year = median(dat$year),
       origin = getmode(dat$origin),
       name = getmode(dat$name)
```

```
    ))
```




### 9.8

#### (a)

```
dat <- OJ
dat$Purchase <- factor(dat$Purchase)
set.seed(1113)
train <- sample(nrow(dat), 800)
training <- dat[train, ]
test     <- dat[-train, ]
```

#### (b)

```
set.seed(1114)
svm.fit <- svm(Purchase~., data = training, kernel = "linear", cost = .01)
summary(svm.fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = training, kernel = "linear", cost = 0.01)
##
##
```

```
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  435
##
##  ( 217 218 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

**(c)**

```
training.pred <- predict(svm.fit, newdata = training, response = "class")
test.pred     <- predict(svm.fit, newdata = test,     response = "class")
conf.training <- table(predict = training.pred, truth = training$Purchase)
conf.test     <- table(predict = test.pred,     truth = test$Purchase)
training_error <- round(1 - (conf.training[1, 1] + conf.training[2, 2])/nrow(training), 4)
test_error   <- round(1- (conf.test[1, 1] + conf.test[2, 2])/nrow(test), 4)
cat("the training error is ", training_error, "\n")
```

```
## the training error is  0.1675
```

```
cat("the test error is ", test_error)
```

```
## the test error is  0.1926
```

**(d)**

```
power_range <- seq(-2, 2, by = .25)
power_grid <- power_range[seq(1, length(power_range), 2)] # for plot
cost_range <- 10^power_range
cost_grid  <- cost_range[seq(1, length(cost_range), 2)] # for plot
tune.out <- tune(svm, Purchase~., data = training, kernel = "linear",
                 ranges = list(
                   cost = cost_range
                 ))
tune.out$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = training,
##     ranges = list(cost = cost_range), kernel = "linear")
##
##
```

```
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.05623413
##
## Number of Support Vectors:  357
```

### (e)

```
training.pred <- predict(tune.out$best.model, newdata = training, response = "class")
test.pred     <- predict(tune.out$best.model, newdata = test,     response = "class")
conf.training <- table(predict = training.pred, truth = training$Purchase)
conf.test     <- table(predict = test.pred,     truth = test$Purchase)
training_error <- round(1 - (conf.training[1, 1] + conf.training[2, 2])/nrow(training), 4)
test_error   <- round(1- (conf.test[1, 1] + conf.test[2, 2])/nrow(test), 4)
cat("the training error is ", training_error, "\n")
```

```
## the training error is  0.1587
```

```
cat("the test error is ", test_error)
```

```
## the test error is  0.1815
```

### (f)

```
# first fit
set.seed(1128)
svm.fit <- svm(Purchase~., data = training, kernel = "radial", cost = .01)
summary(svm.fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = training, kernel = "radial", cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##
## Number of Support Vectors:  620
##
##  ( 308 312 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
# first fit result
training.pred <- predict(svm.fit, newdata = training, response = "class")
test.pred     <- predict(svm.fit, newdata = test,     response = "class")
conf.training <- table(predict = training.pred, truth = training$Purchase)
conf.test     <- table(predict = test.pred,     truth = test$Purchase)
training_error <- round(1 - (conf.training[1, 1] + conf.training[2, 2])/nrow(training), 4)
test_error    <- round(1- (conf.test[1, 1] + conf.test[2, 2])/nrow(test), 4)
cat("the training error is ", training_error, "\n")
```

```
## the training error is  0.385
```

```r
cat("the test error is ", test_error)
```

```
## the test error is  0.4037
```

```r
# tuning parameters
power_range <- seq(-3, 3, by = .25)
power_grid <- power_range[seq(1, length(power_range), 2)] # for plot
cost_range <- 10^power_range
cost_grid  <- cost_range[seq(1, length(cost_range), 2)] # for plot
tune.out <- tune(svm, Purchase~., data = training, kernel = "radial",
                 ranges = list(
                   cost = cost_range
                 ))
tune.out$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = training,
##     ranges = list(cost = cost_range), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  360
```

```r
training.pred <- predict(tune.out$best.model, newdata = training, response = "class")
test.pred     <- predict(tune.out$best.model, newdata = test,     response = "class")
conf.training <- table(predict = training.pred, truth = training$Purchase)
conf.test     <- table(predict = test.pred,     truth = test$Purchase)
training_error <- round(1 - (conf.training[1, 1] + conf.training[2, 2])/nrow(training), 4)
test_error    <- round(1- (conf.test[1, 1] + conf.test[2, 2])/nrow(test), 4)
cat("the new training error is ", training_error, "\n")
```

```
## the new training error is  0.1412
```

```r
cat("the new test error is ", test_error)
```

```
## the new test error is  0.2037
```

**(g)**

```r
# first fit
set.seed(1128)
svm.fit <- svm(Purchase~., data = training, kernel = "polynomial",
               degree = 2, cost = .01)
summary(svm.fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = training, kernel = "polynomial",
##     degree = 2, cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  622
##
##  ( 308 314 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
# first fit result
training.pred <- predict(svm.fit, newdata = training, response = "class")
test.pred     <- predict(svm.fit, newdata = test,     response = "class")
conf.training <- table(predict = training.pred, truth = training$Purchase)
conf.test     <- table(predict = test.pred,     truth = test$Purchase)
training_error <- round(1 - (conf.training[1, 1] + conf.training[2, 2])/nrow(training), 4)
test_error    <- round(1- (conf.test[1, 1] + conf.test[2, 2])/nrow(test), 4)
cat("the training error is ", training_error, "\n")
```

```
## the training error is  0.3625
```

```r
cat("the test error is ", test_error)
```

```
## the test error is  0.3926
```

```r
# tuning parameters
power_range <- seq(-3, 3, by = .25)
power_grid <- power_range[seq(1, length(power_range), 2)] # for plot
cost_range <- 10^power_range
cost_grid  <- cost_range[seq(1, length(cost_range), 2)] # for plot
tune.out <- tune(svm, Purchase~., data = training, kernel = "radial",
                 ranges = list(
                   cost = cost_range,
                   degree = 2
                 ))
tune.out$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = training,
##     ranges = list(cost = cost_range, degree = 2), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  360
```

```r
training.pred <- predict(tune.out$best.model, newdata = training, response = "class")
test.pred     <- predict(tune.out$best.model, newdata = test,     response = "class")
conf.training <- table(predict = training.pred, truth = training$Purchase)
conf.test     <- table(predict = test.pred,     truth = test$Purchase)
training_error <- round(1 - (conf.training[1, 1] + conf.training[2, 2])/nrow(training), 4)
test_error    <- round(1- (conf.test[1, 1] + conf.test[2, 2])/nrow(test), 4)
cat("the new training error is ", training_error, "\n")
```

```
## the new training error is  0.1412
```

```r
cat("the new test error is ", test_error)
```

```
## the new test error is  0.2037
```

### (h)

*Comments*:

Suprising, linear kernel SVM seems to give the best result on this data set. Though, more flexible kernel performs best on training set (training error rate down to 14%), but their performance on test set is obviously worse than linear kernel (test error rate = 20%), compare with the test error rate of linear model is 18%.