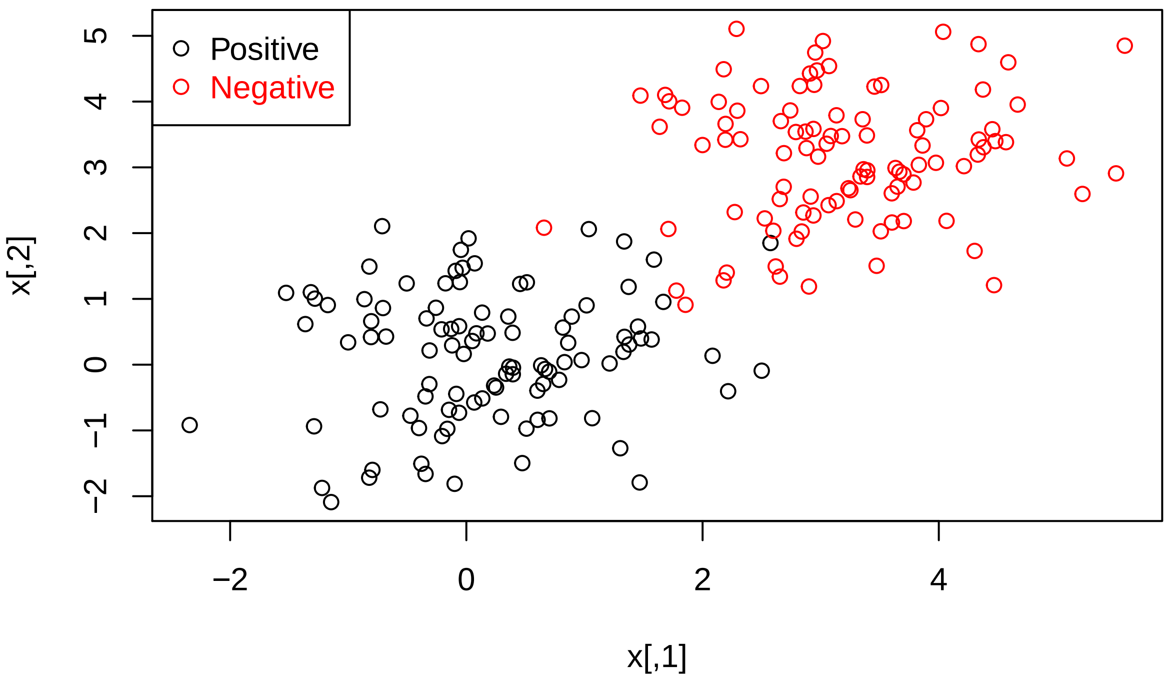
Week 8 Lab Solutions

Generate a linearly separable dataset.

```
set.seed(300)
x.pos <- matrix(rnorm(100*2,mean=0), nrow = 100, ncol = 2)
set.seed(300)
x.neg <- matrix(rnorm(100*2,mean=3), nrow = 100, ncol = 2)
y <- c(rep(1,100),rep(-1,100))
x <- rbind(x.pos,x.neg)
dat <- data.frame(x = x, y = as.factor(y))

plot(x, col = ifelse(y>0,1,2))
legend("topleft",c("Positive","Negative"),col=seq(2),pch=1,text.col=seq(2))
```




```
#####Produce training and testing datasets#####
set.seed(300)
train <-  sample(200, 200*0.7)
dat.train <-  dat[train,]
x.test <- x[-train,]
y.test <- y[-train]
```

1) Install and load the library `e1071`.

```
#install.packages("e1071")
library(e1071)
```

2) Build a linear SVM model with `cost = 1`.

```r
svmfit.train.C1 <- svm(y ~ .,
                       data = dat.train,
                       kernel = "linear",
                       cost = 1,
                       scale = FALSE)
```

3) Report how many support vectors are from each class respectively.

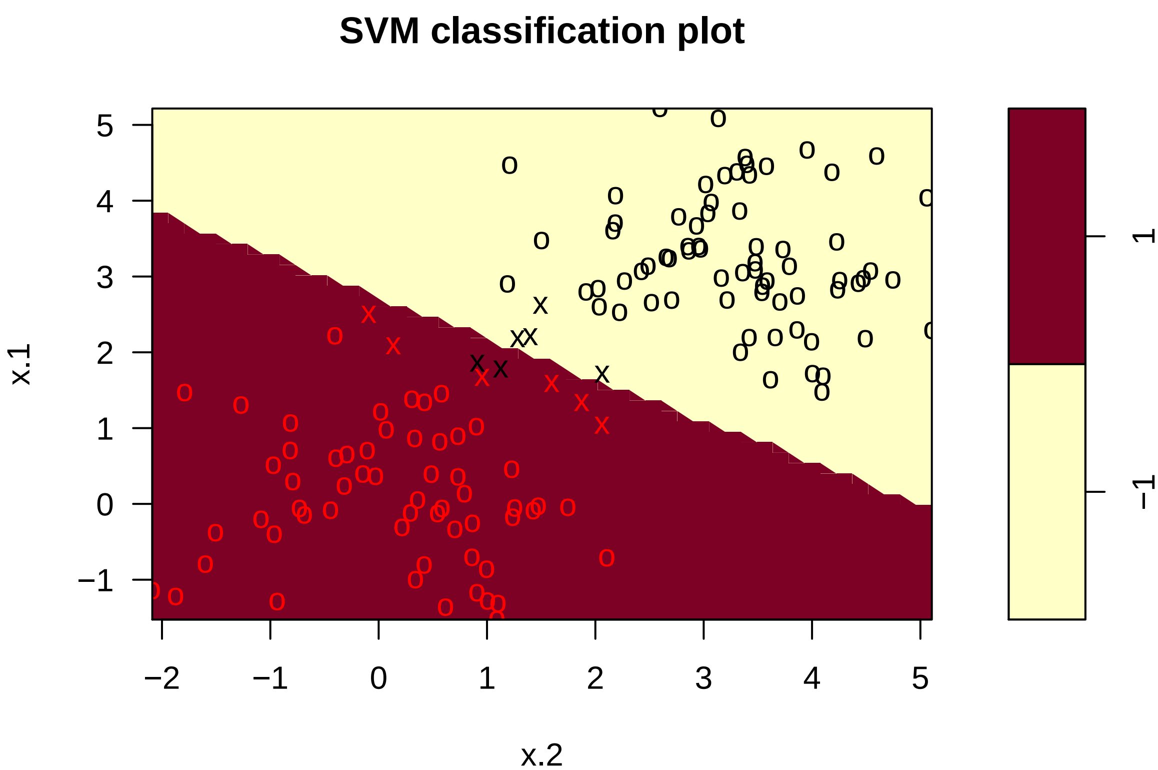
```r
summary(svmfit.train.C1)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat.train, kernel = "linear", cost = 1,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  12
##
##  ( 6 6 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

There are 12 support vectors, 6 from the class of `y = -1` and 6 from the class of`y = 1`.

4) Plot the svm model and check how many support vectors are on the wrong side of the boundary and how many data points are very close to the margin.

```r
plot(svmfit.train.C1, dat.train)
```



There are 2 points on the wrong side of the boundary.

At least 2 data points on the red side and at least 1 point on the yellow side are very close to the margin.

5) Predict the class label of y on the testing set and estimate the testing error rate.

```
y.pred.C1 <- predict(svmfit.train.C1, newdata = x.test)
mean(y.pred.C1 != y.test)
```

```
## [1] 0.03333333
```

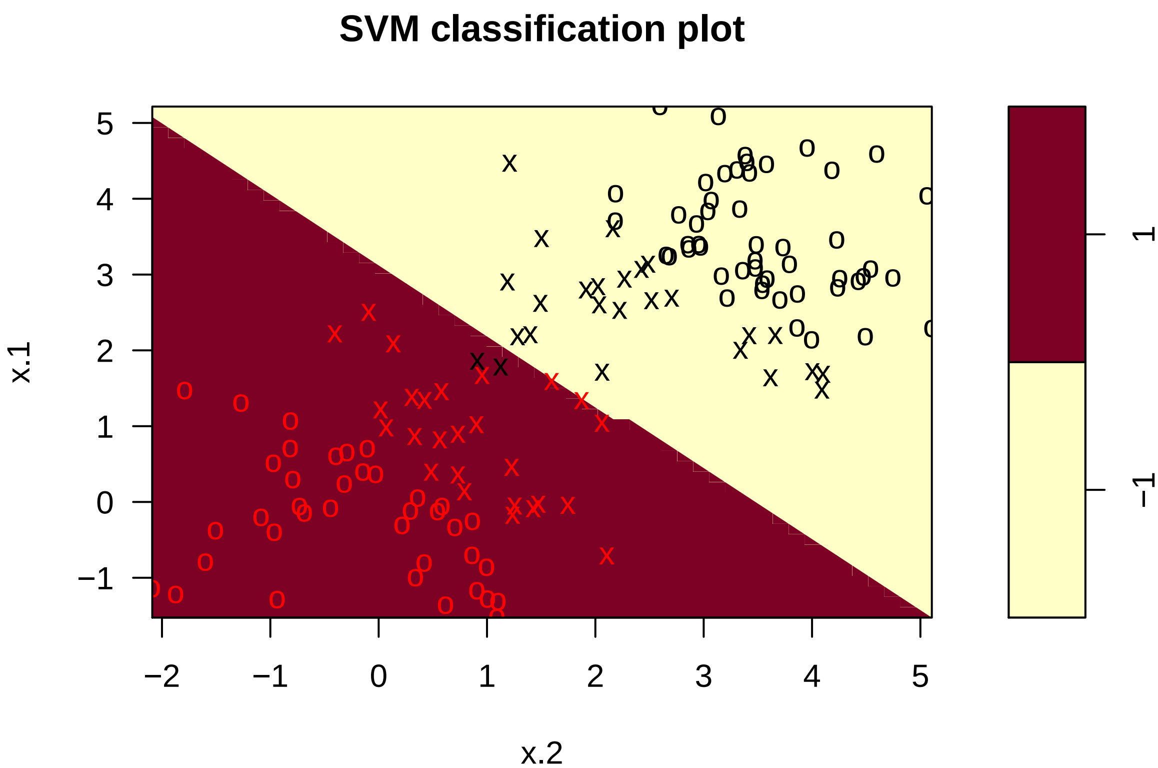
6) Now try a smaller cost = 0.01 and a larger cost = 1e5 and repeat step 2)-5).

```
#######cost = 0.01########
svmfit.train.C001 <- svm(y ~ .,
                         data = dat.train,
                         kernel = "linear",
                         cost = 0.01,
                         scale = FALSE)
summary(svmfit.train.C001)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat.train, kernel = "linear", cost = 0.01,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
```

```
##        cost:  0.01
##
## Number of Support Vectors:  52
##
##  ( 26 26 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```
plot(svmfit.train.C001,dat.train)
```




```
y.pred.C001 <- predict(svmfit.train.C001, newdata = x.test)
mean(y.pred.C001 != y.test)
```

```
## [1] 0.03333333
```

```
#######cost = 1e5########
svmfit.train.C1e5 <- svm(y ~ .,
                         data = dat.train,
                         kernel = "linear",
                         cost = 1e5,
                         scale = FALSE)
summary(svmfit.train.C1e5)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat.train, kernel = "linear", cost = 1e+05,
##     scale = FALSE)
```

```
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1e+05
##
## Number of Support Vectors:  6
##
##  ( 3 3 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```r
plot(svmfit.train.C1e5,dat.train)
```


```r
y.pred.C1e5 <- predict(svmfit.train.C1e5, newdata = x.test)
mean(y.pred.C1e5 != y.test)
```

```
## [1] 0.03333333
```

7) Use `tune()` function to select the best model (tune the parameter `C` or `cost`). Set seed to be 1.

```r
set.seed(1)
tune.out <- tune(svm,
                 y ~ .,
```

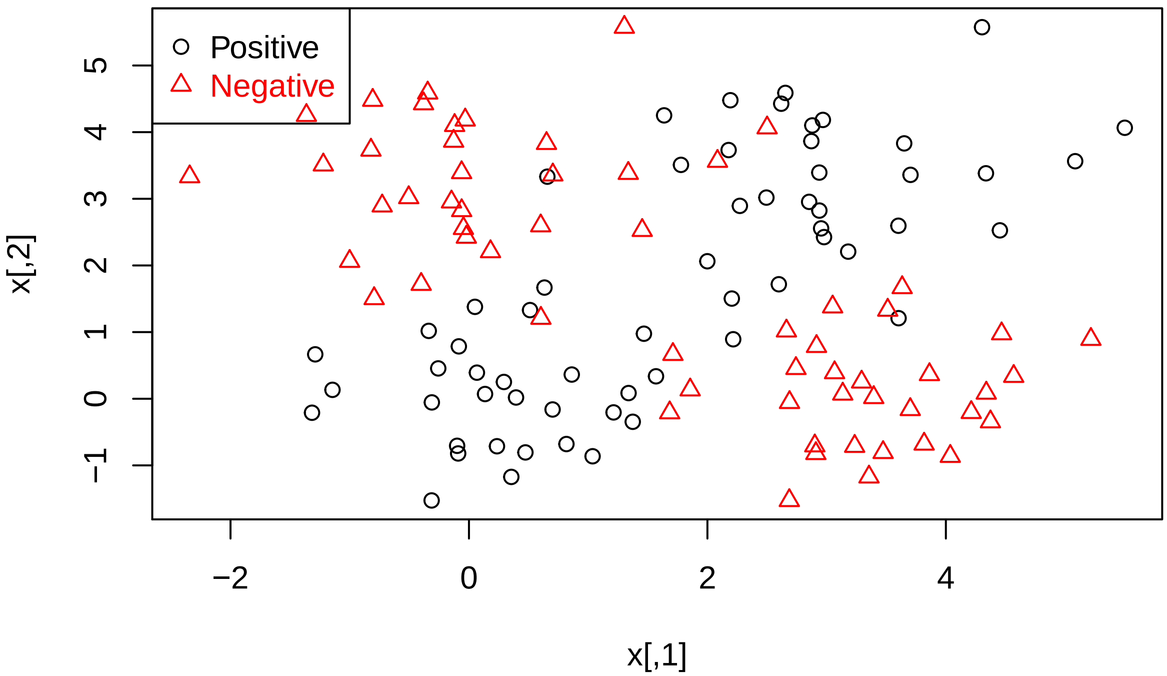
```
                 data = dat.train,
                 kernel = "linear",
                 ranges = list(cost = c(0.001,0.01,0.1,1,5,10,100,1000,10000,1e5)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.01428571
##
## - Detailed performance results:
##     cost      error dispersion
## 1  1e-03 0.45714286 0.16903085
## 2  1e-02 0.02142857 0.03450328
## 3  1e-01 0.01428571 0.03011693
## 4  1e+00 0.02857143 0.04994328
## 5  5e+00 0.04285714 0.04994328
## 6  1e+01 0.01428571 0.03011693
## 7  1e+02 0.01428571 0.03011693
## 8  1e+03 0.02142857 0.03450328
## 9  1e+04 0.02142857 0.03450328
## 10 1e+05 0.02142857 0.03450328
```

Generate a linearly inseparable dataset.

```
set.seed(300)
x.pos1 <- matrix(rnorm(30*2,mean=0), nrow=30, ncol=2)
x.pos2 <- matrix(rnorm(30*2,mean=3), nrow=30, ncol=2)
set.seed(300)
x.neg1 <- matrix(c(rnorm(30,mean=0)+3,rnorm(30,mean=0)),nrow=30,ncol=2)
x.neg2<- matrix(c(rnorm(30,mean=3)-3,rnorm(30,mean=3)),nrow=30,ncol=2)
y <- c(rep(1,60),rep(-1,60))
x <- rbind(x.pos1, x.pos2, x.neg1, x.neg2)
dat2 <- data.frame(x = x, y = as.factor(y))


plot(x,
     col = ifelse(y > 0, 1, 2),
     pch = ifelse(y > 0, 1, 2))
legend("topleft",
       c("Positive","Negative"),
       col = seq(2),
       pch = 1:2,
       text.col = seq(2))
```




```
#####Produce training and testing datasets#####
set.seed(300)
train <- sample(120, 120*0.7)
dat2.train <- dat2[train,]
x.test <- x[-train,]
y.test <- y[-train]
```

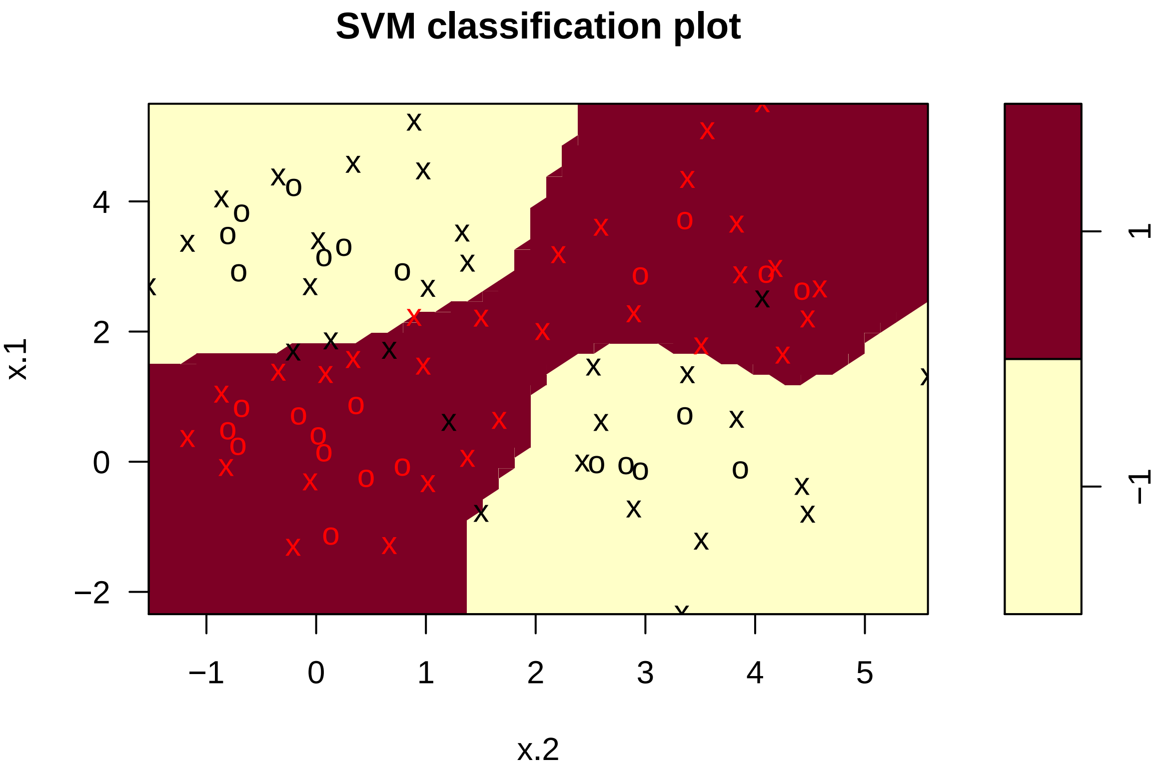
8) Build an SVM model with a radial kernel, `gamma = 1` and `cost = 1`. In this model, a) see the summary; b) plot the svm and c) estimate the testing error rate.

```
svmfit.radial.G1C1 <- svm(y ~ ., data = dat2.train,
                          kernel = "radial",
                          gamma = 1, cost = 1,
                          scale = F)
summary(svmfit.radial.G1C1)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat2.train, kernel = "radial", gamma = 1,
##     cost = 1, scale = F)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  58
##
##  ( 29 29 )
##
##
```

```
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```r
plot(svmfit.radial.G1C1, data = dat2.train)
```




```r
y.pred.radial.G1C1 <- predict(svmfit.radial.G1C1, newdata = x.test)
mean(y.pred.radial.G1C1 != y.test)
```

```
## [1] 0.08333333
```

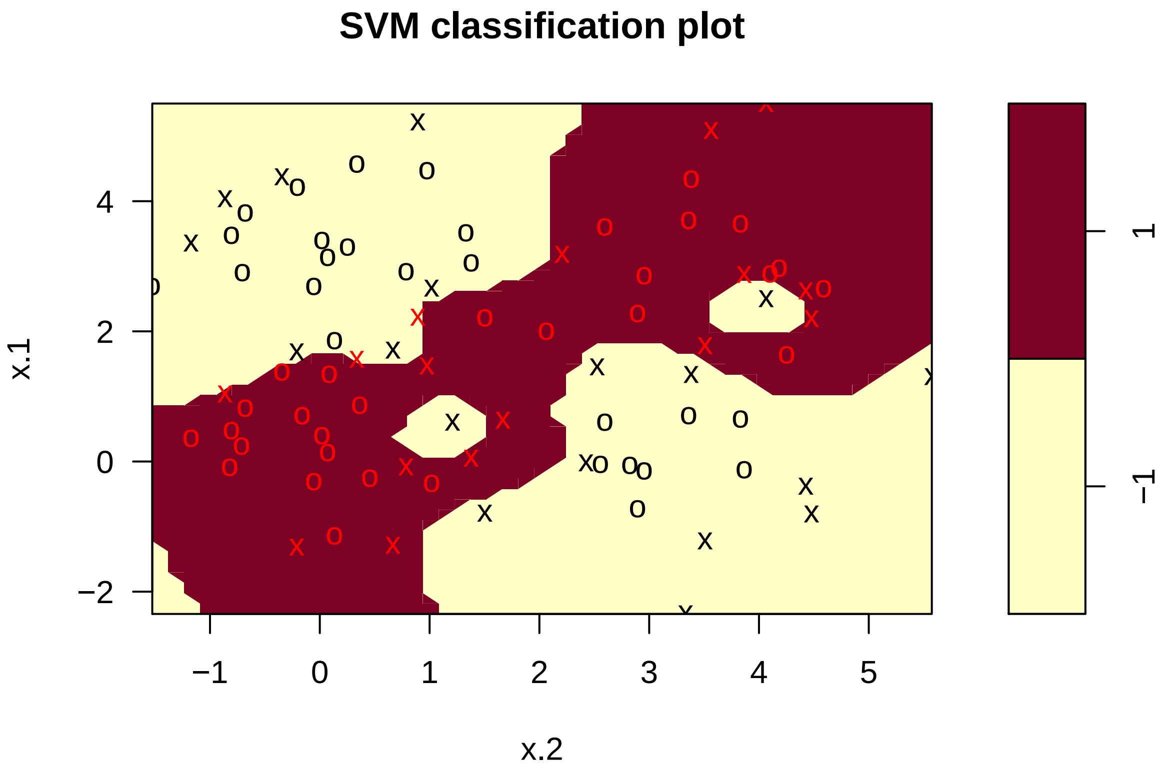
9) Build an SVM model with a radial kernel, `gamma = 1` and `cost = 1e5`. In this model, a) see the summary; b) plot the svm and c) estimate the testing error rate.

```r
svmfit.radial.G1C1e5 <- svm(y ~ .,data = dat2.train,
                            kernel = "radial",
                            gamma = 1,cost = 1e5,
                            scale = F)
summary(svmfit.radial.G1C1e5)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat2.train, kernel = "radial", gamma = 1,
##     cost = 1e+05, scale = F)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1e+05
```

```
##
## Number of Support Vectors:  34
##
##  ( 18 16 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```r
plot(svmfit.radial.G1C1e5,data = dat2.train)
```




```r
y.pred.radial.G1C1e5 <- predict(svmfit.radial.G1C1e5, newdata = x.test)
mean(y.pred.radial.G1C1e5 != y.test)
```

```
## [1] 0.1944444
```

10) Choose the best choice of gamma and cost for an SVM with a radial kernel.

```r
set.seed(1)
tune.out.radial <- tune(svm, y ~ .,
                        data = dat2.train, kernel = "radial",
                        ranges = list(cost = c(0.1,1,10,100,1000),
                                      gamma = c(0.5,1,2,3,4)))

summary(tune.out.radial)
```

```
##
## Parameter tuning of 'svm':
##
```

```
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1     1
##
## - best performance: 0.1027778
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1  1e-01   0.5 0.1972222  0.1262455
## 2  1e+00   0.5 0.1402778  0.1197366
## 3  1e+01   0.5 0.1513889  0.1218657
## 4  1e+02   0.5 0.1166667  0.1204060
## 5  1e+03   0.5 0.1388889  0.1281329
## 6  1e-01   1.0 0.1263889  0.1249914
## 7  1e+00   1.0 0.1027778  0.1109567
## 8  1e+01   1.0 0.1500000  0.1522578
## 9  1e+02   1.0 0.1263889  0.1355225
## 10 1e+03   1.0 0.1750000  0.1477425
## 11 1e-01   2.0 0.2722222  0.1406067
## 12 1e+00   2.0 0.1375000  0.1571962
## 13 1e+01   2.0 0.1250000  0.1521452
## 14 1e+02   2.0 0.1638889  0.1479744
## 15 1e+03   2.0 0.1888889  0.1379909
## 16 1e-01   3.0 0.4777778  0.1446639
## 17 1e+00   3.0 0.1486111  0.1590397
## 18 1e+01   3.0 0.1500000  0.1546324
## 19 1e+02   3.0 0.1750000  0.1354829
## 20 1e+03   3.0 0.1750000  0.1354829
## 21 1e-01   4.0 0.5597222  0.1387422
## 22 1e+00   4.0 0.1611111  0.1507583
## 23 1e+01   4.0 0.1388889  0.1528479
## 24 1e+02   4.0 0.1986111  0.1309539
## 25 1e+03   4.0 0.1986111  0.1309539
```

11) Use the best model to estimate the testing error rate.

```r
y.pred.radial.best <- predict(tune.out.radial$best.model, newdata = x.test)
mean(y.pred.radial.best != y.test)
```

```
## [1] 0.08333333
```