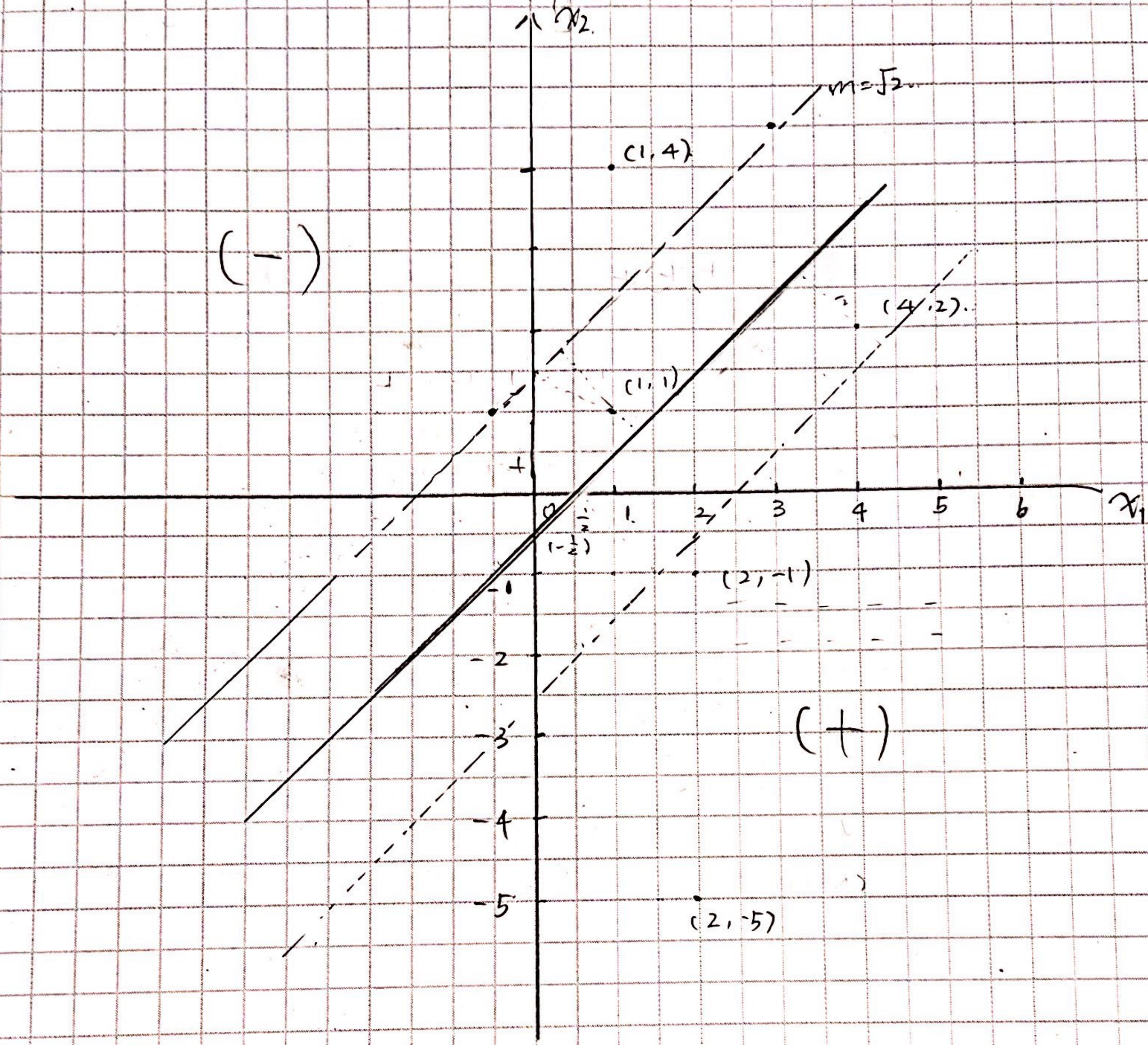
UNIVERSITY OF MICHIGAN
DATA MINING
STATS415

ASSIGNMENT9

Author: *Shu* ZHOU
ID: 19342932

December 2, 2020



(c) Hence (1,4) : (−)

(1,1) : −

(2,−5) : +

(2,−1) : +

(4,2) : +

(d). (1,4); $\xi_1 = 0$

(1,1); $\xi_2 = \frac{\frac{2}{3}\sqrt{2}}{\sqrt{2}} = \frac{2}{3}$

(2,−5); $\xi_3 = 0$

(2,−1) $\xi_4 = \frac{2}{3}$

(4,2) $\xi_5 = \frac{(1+\frac{2}{3})\sqrt{2}}{\sqrt{2}} = \frac{5}{3}$

Q2.

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.0.3
```

```
data(crabs)
head(crabs)
```

```
##   sp sex index   FL  RW   CL   CW  BD
## 1  B   M     1  8.1 6.7 16.1 19.0 7.0
## 2  B   M     2  8.8 7.7 18.1 20.8 7.4
## 3  B   M     3  9.2 7.8 19.0 22.4 7.7
## 4  B   M     4  9.6 7.9 20.1 23.1 8.2
## 5  B   M     5  9.8 8.0 20.3 23.0 8.2
## 6  B   M     6 10.8 9.0 23.0 26.5 9.8
```

```
set.seed(6789)
inTrain <- createDataPartition(crabs$sp, p = 0.8, list = FALSE)
training <- crabs[inTrain,]
testing <- crabs[-inTrain,]
```

a

For the linear Svm model

```
set.seed(1)

ranges = c(0.001, 0.01, 0.1, 1, 5, 10, 100)

svmTrainErr = vector(length = length(ranges))
svmTestErr = vector(length = length(ranges))


for (i in 1:length(ranges)) {
svm.model = svm(sp ~ .-index, data = training, kernel = "linear", cost = ranges[i], scale = FALSE)
pred.train <- predict(svm.model, training)
pred.test<-predict(svm.model,testing)
svmTrainErr[i] = mean(pred.train != training$sp)
svmTestErr[i] = mean(pred.test != testing$sp)
}

tune.out <- tune(svm,  sp~ .-index, data = training, kernel = "linear",
ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0
```

```
##
## - Detailed performance results:
##    cost   error dispersion
## 1 1e-03 0.56250 0.12842529
## 2 1e-02 0.35625 0.11043632
## 3 1e-01 0.13750 0.07095578
## 4 1e+00 0.00000 0.00000000
## 5 5e+00 0.00000 0.00000000
## 6 1e+01 0.00000 0.00000000
## 7 1e+02 0.00000 0.00000000
```

```
tune.out$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = sp ~ . - index, data = training,
##     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:  1
##
## Number of Support Vectors:  49
```

```
svmTestErr
```

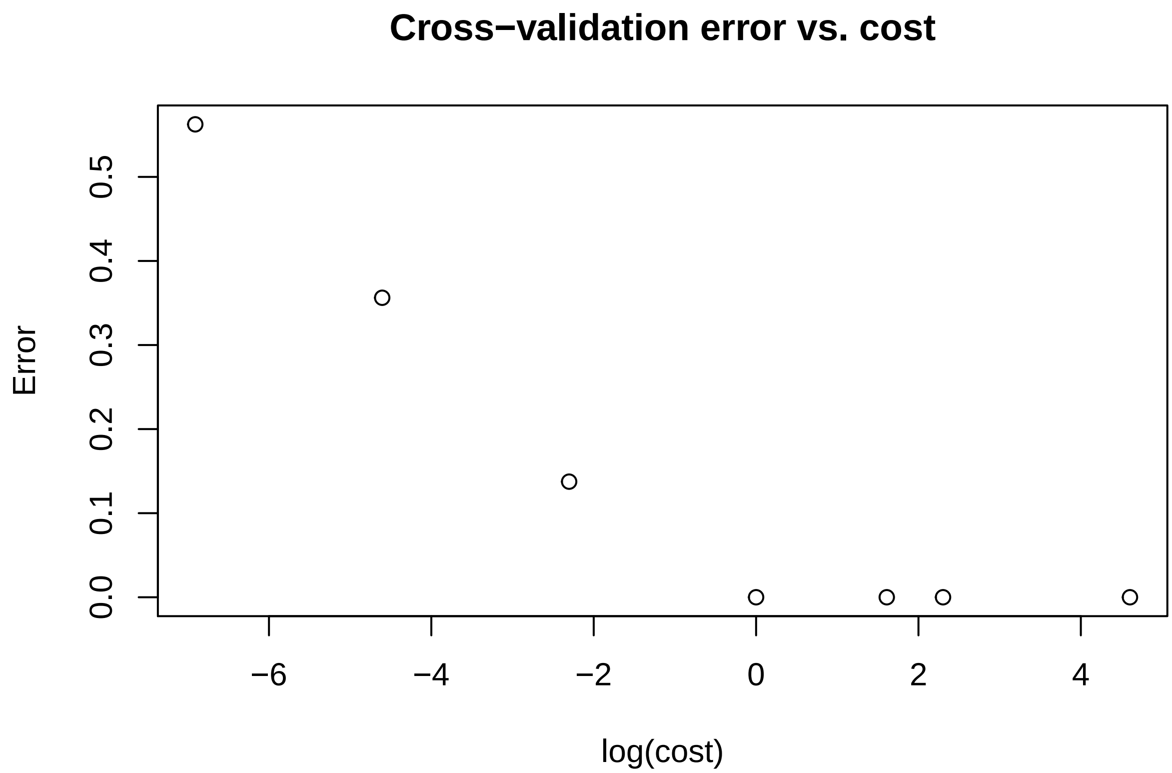
```
## [1] 0.4 0.0 0.0 0.0 0.0 0.0 0.0
```

```
svmTrainErr
```

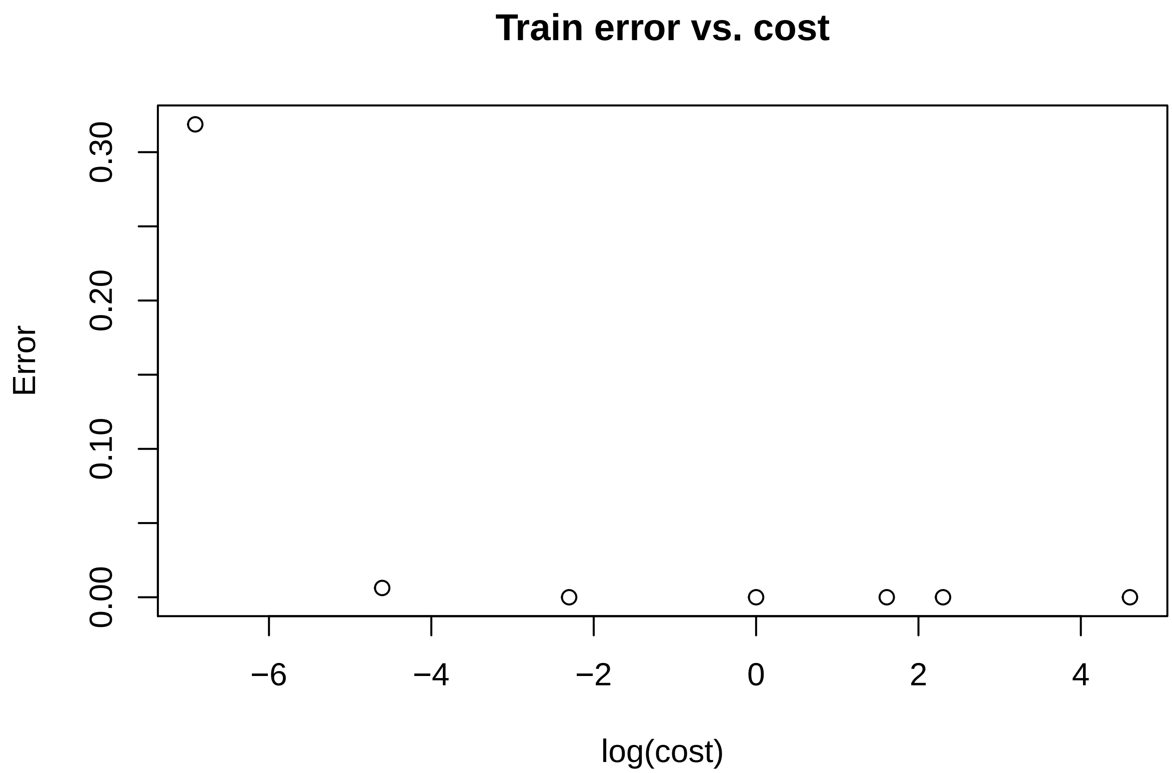
```
## [1] 0.31875 0.00625 0.00000 0.00000 0.00000 0.00000 0.00000
```

We take the lograithm of the cost to make plots for better visualization

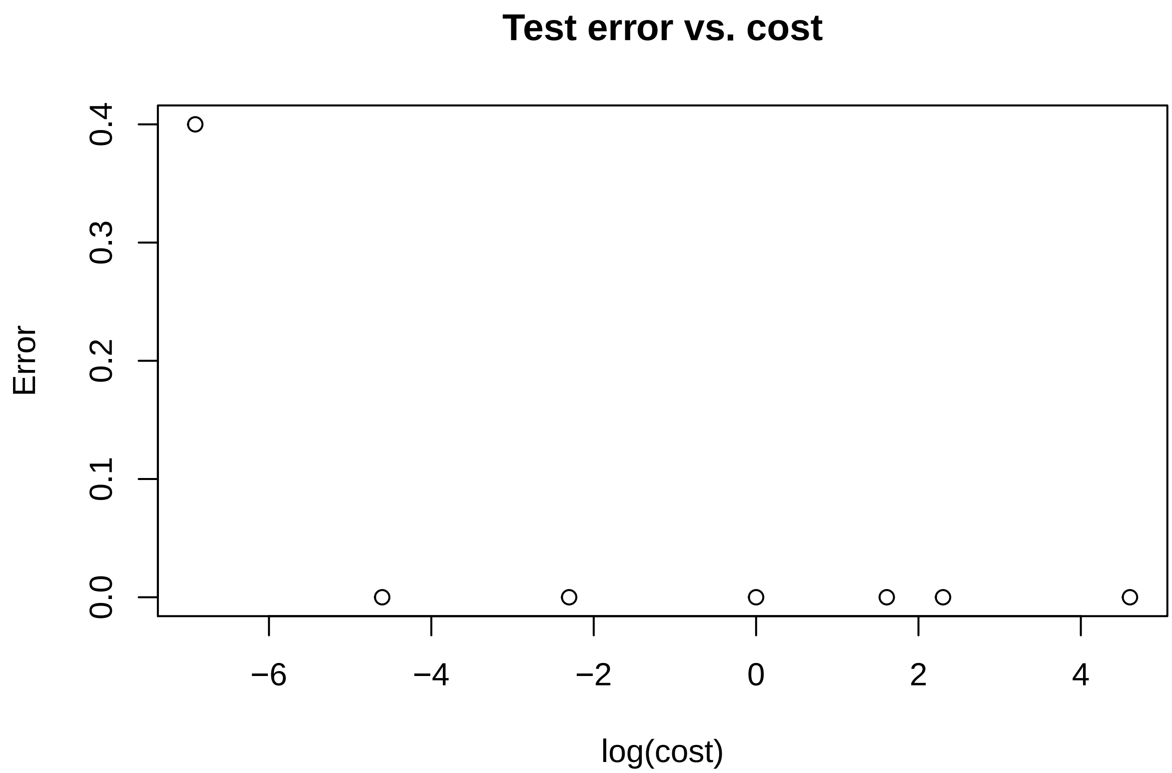
```
plot(log(tune.out$performances$cost),tune.out$performances$error,main = "Cross-validation error vs. cos
```




```
plot(log(ranges),svmTrainErr,main = "Train error vs. cost", xlab = "log(cost)",ylab = "Error" )
```




```
plot(log(ranges),svmTestErr,main = "Test error vs. cost", xlab = "log(cost)",ylab = "Error" )
```



The best model is obtained by $cost \geq 1$, Since the cv-error, train error and test error are all minimized.

b

For the non-linear Svm model

```
ranges =  c(0.1, 1, 10, 100, 1000)

set.seed(1)
tune.out <- tune(svm,sp ~ .-index, data = training, kernel = "radial",
ranges = list(cost = c(0.1, 1, 10, 100, 1000),
gamma = c(0.5, 1, 2, 3, 4),degree = c(1,2,3,4,5)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma degree
##    10   0.5      1
##
## - best performance: 0.00625
##
## - Detailed performance results:
##       cost gamma degree   error dispersion
## 1   1e-01   0.5      1 0.33750 0.15080801
```

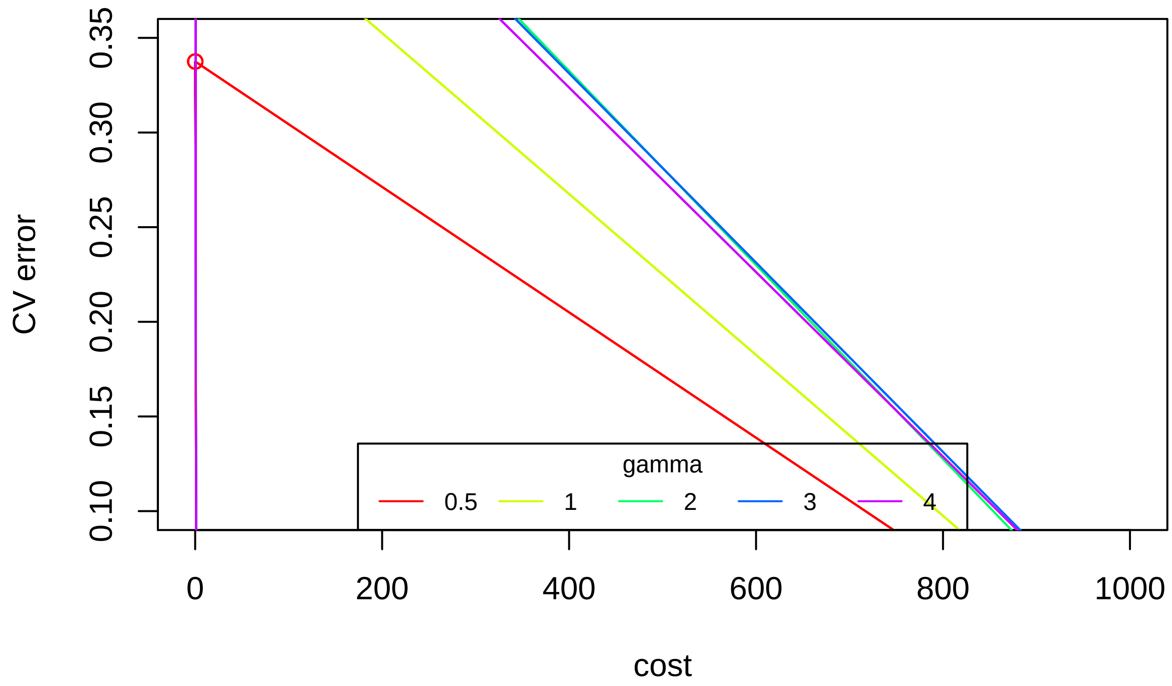
```
## 2   1e+00   0.5      1 0.07500 0.06454972
## 3   1e+01   0.5      1 0.00625 0.01976424
## 4   1e+02   0.5      1 0.00625 0.01976424
## 5   1e+03   0.5      1 0.00625 0.01976424
## 6   1e-01   1.0      1 0.43750 0.13501543
## 7   1e+00   1.0      1 0.06250 0.05892557
## 8   1e+01   1.0      1 0.01250 0.02635231
## 9   1e+02   1.0      1 0.01250 0.02635231
## 10  1e+03   1.0      1 0.01250 0.02635231
## 11  1e-01   2.0      1 0.53750 0.17969882
## 12  1e+00   2.0      1 0.06875 0.05472469
## 13  1e+01   2.0      1 0.02500 0.03227486
## 14  1e+02   2.0      1 0.02500 0.03227486
## 15  1e+03   2.0      1 0.02500 0.03227486
## 16  1e-01   3.0      1 0.53125 0.20252315
## 17  1e+00   3.0      1 0.06250 0.05892557
## 18  1e+01   3.0      1 0.03125 0.03294039
## 19  1e+02   3.0      1 0.03125 0.03294039
## 20  1e+03   3.0      1 0.03125 0.03294039
## 21  1e-01   4.0      1 0.51875 0.22831280
## 22  1e+00   4.0      1 0.05625 0.06215181
## 23  1e+01   4.0      1 0.03125 0.03294039
## 24  1e+02   4.0      1 0.03125 0.03294039
## 25  1e+03   4.0      1 0.03125 0.03294039
## 26  1e-01   0.5      2 0.33750 0.15080801
## 27  1e+00   0.5      2 0.07500 0.06454972
## 28  1e+01   0.5      2 0.00625 0.01976424
## 29  1e+02   0.5      2 0.00625 0.01976424
## 30  1e+03   0.5      2 0.00625 0.01976424
## 31  1e-01   1.0      2 0.43750 0.13501543
## 32  1e+00   1.0      2 0.06250 0.05892557
## 33  1e+01   1.0      2 0.01250 0.02635231
## 34  1e+02   1.0      2 0.01250 0.02635231
## 35  1e+03   1.0      2 0.01250 0.02635231
## 36  1e-01   2.0      2 0.53750 0.17969882
## 37  1e+00   2.0      2 0.06875 0.05472469
## 38  1e+01   2.0      2 0.02500 0.03227486
## 39  1e+02   2.0      2 0.02500 0.03227486
## 40  1e+03   2.0      2 0.02500 0.03227486
## 41  1e-01   3.0      2 0.53125 0.20252315
## 42  1e+00   3.0      2 0.06250 0.05892557
## 43  1e+01   3.0      2 0.03125 0.03294039
## 44  1e+02   3.0      2 0.03125 0.03294039
## 45  1e+03   3.0      2 0.03125 0.03294039
## 46  1e-01   4.0      2 0.51875 0.22831280
## 47  1e+00   4.0      2 0.05625 0.06215181
## 48  1e+01   4.0      2 0.03125 0.03294039
## 49  1e+02   4.0      2 0.03125 0.03294039
## 50  1e+03   4.0      2 0.03125 0.03294039
## 51  1e-01   0.5      3 0.33750 0.15080801
## 52  1e+00   0.5      3 0.07500 0.06454972
## 53  1e+01   0.5      3 0.00625 0.01976424
## 54  1e+02   0.5      3 0.00625 0.01976424
## 55  1e+03   0.5      3 0.00625 0.01976424
```

```
## 56  1e-01   1.0       3 0.43750 0.13501543
## 57  1e+00   1.0       3 0.06250 0.05892557
## 58  1e+01   1.0       3 0.01250 0.02635231
## 59  1e+02   1.0       3 0.01250 0.02635231
## 60  1e+03   1.0       3 0.01250 0.02635231
## 61  1e-01   2.0       3 0.53750 0.17969882
## 62  1e+00   2.0       3 0.06875 0.05472469
## 63  1e+01   2.0       3 0.02500 0.03227486
## 64  1e+02   2.0       3 0.02500 0.03227486
## 65  1e+03   2.0       3 0.02500 0.03227486
## 66  1e-01   3.0       3 0.53125 0.20252315
## 67  1e+00   3.0       3 0.06250 0.05892557
## 68  1e+01   3.0       3 0.03125 0.03294039
## 69  1e+02   3.0       3 0.03125 0.03294039
## 70  1e+03   3.0       3 0.03125 0.03294039
## 71  1e-01   4.0       3 0.51875 0.22831280
## 72  1e+00   4.0       3 0.05625 0.06215181
## 73  1e+01   4.0       3 0.03125 0.03294039
## 74  1e+02   4.0       3 0.03125 0.03294039
## 75  1e+03   4.0       3 0.03125 0.03294039
## 76  1e-01   0.5       4 0.33750 0.15080801
## 77  1e+00   0.5       4 0.07500 0.06454972
## 78  1e+01   0.5       4 0.00625 0.01976424
## 79  1e+02   0.5       4 0.00625 0.01976424
## 80  1e+03   0.5       4 0.00625 0.01976424
## 81  1e-01   1.0       4 0.43750 0.13501543
## 82  1e+00   1.0       4 0.06250 0.05892557
## 83  1e+01   1.0       4 0.01250 0.02635231
## 84  1e+02   1.0       4 0.01250 0.02635231
## 85  1e+03   1.0       4 0.01250 0.02635231
## 86  1e-01   2.0       4 0.53750 0.17969882
## 87  1e+00   2.0       4 0.06875 0.05472469
## 88  1e+01   2.0       4 0.02500 0.03227486
## 89  1e+02   2.0       4 0.02500 0.03227486
## 90  1e+03   2.0       4 0.02500 0.03227486
## 91  1e-01   3.0       4 0.53125 0.20252315
## 92  1e+00   3.0       4 0.06250 0.05892557
## 93  1e+01   3.0       4 0.03125 0.03294039
## 94  1e+02   3.0       4 0.03125 0.03294039
## 95  1e+03   3.0       4 0.03125 0.03294039
## 96  1e-01   4.0       4 0.51875 0.22831280
## 97  1e+00   4.0       4 0.05625 0.06215181
## 98  1e+01   4.0       4 0.03125 0.03294039
## 99  1e+02   4.0       4 0.03125 0.03294039
## 100 1e+03   4.0       4 0.03125 0.03294039
## 101 1e-01   0.5       5 0.33750 0.15080801
## 102 1e+00   0.5       5 0.07500 0.06454972
## 103 1e+01   0.5       5 0.00625 0.01976424
## 104 1e+02   0.5       5 0.00625 0.01976424
## 105 1e+03   0.5       5 0.00625 0.01976424
## 106 1e-01   1.0       5 0.43750 0.13501543
## 107 1e+00   1.0       5 0.06250 0.05892557
## 108 1e+01   1.0       5 0.01250 0.02635231
## 109 1e+02   1.0       5 0.01250 0.02635231
```

```
## 110 1e+03   1.0      5 0.01250 0.02635231
## 111 1e-01   2.0      5 0.53750 0.17969882
## 112 1e+00   2.0      5 0.06875 0.05472469
## 113 1e+01   2.0      5 0.02500 0.03227486
## 114 1e+02   2.0      5 0.02500 0.03227486
## 115 1e+03   2.0      5 0.02500 0.03227486
## 116 1e-01   3.0      5 0.53125 0.20252315
## 117 1e+00   3.0      5 0.06250 0.05892557
## 118 1e+01   3.0      5 0.03125 0.03294039
## 119 1e+02   3.0      5 0.03125 0.03294039
## 120 1e+03   3.0      5 0.03125 0.03294039
## 121 1e-01   4.0      5 0.51875 0.22831280
## 122 1e+00   4.0      5 0.05625 0.06215181
## 123 1e+01   4.0      5 0.03125 0.03294039
## 124 1e+02   4.0      5 0.03125 0.03294039
## 125 1e+03   4.0      5 0.03125 0.03294039
```

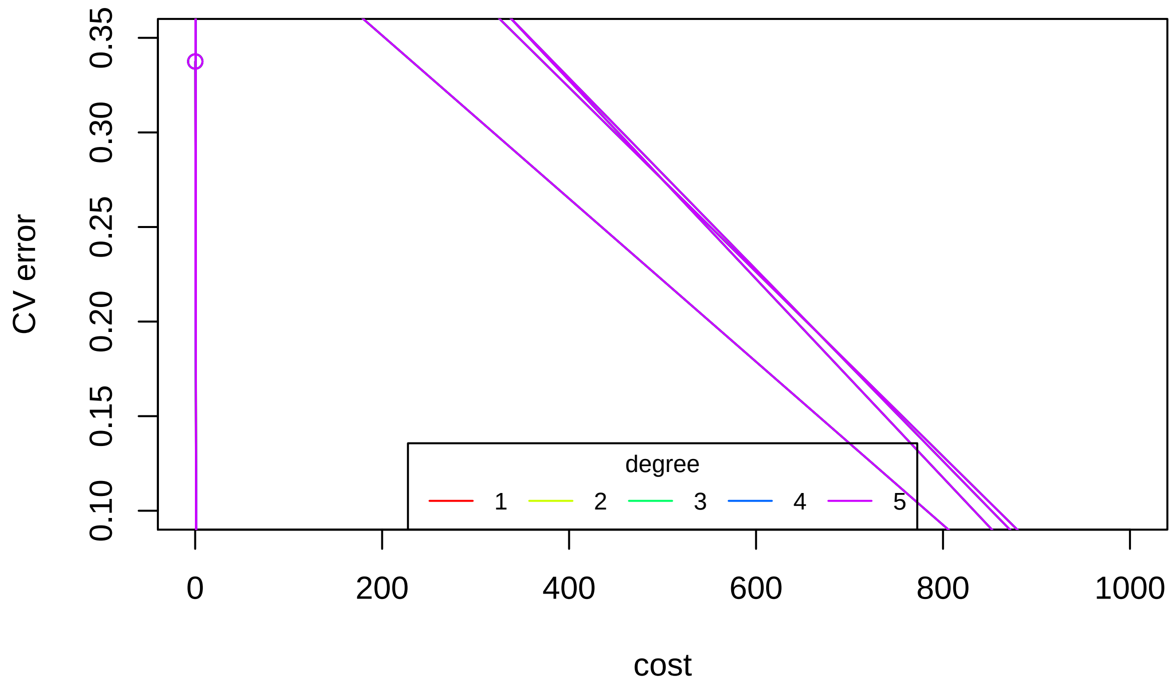
We first pick the value of gamma

```r
with(tune.out$performances, {
plot(error[gamma == 0.5] ~ cost[gamma == 0.5], ylim = c(.1, .35),
type = "o", col = rainbow(5)[1], ylab = "CV error", xlab = "cost")
lines(error[gamma == 1] ~ cost[gamma == 1],
type = "o", col = rainbow(5)[2])
lines(error[gamma == 2] ~ cost[gamma == 2],
type = "o", col = rainbow(5)[3])
lines(error[gamma == 3] ~ cost[gamma == 3],
type = "o", col = rainbow(5)[4])
lines(error[gamma == 4] ~ cost[gamma == 4],
type = "o", col = rainbow(5)[5])
})
legend("bottom", horiz = T, legend = c(0.5, 1:4), col = rainbow(5),
lty = 1, cex = .75, title = "gamma")
```



Hence the best choice of gamma is 0.5. Then we pick the value of degree

```
with(tune.out$performances, {
plot(error[degree == 5] ~ cost[degree == 5], ylim = c(.1, .35),
type = "o", col = rainbow(5)[1], ylab = "CV error", xlab = "cost")
lines(error[degree == 1] ~ cost[degree == 1],
type = "o", col = rainbow(5)[2])
lines(error[degree == 2] ~ cost[degree == 2],
type = "o", col = rainbow(5)[3])
lines(error[degree == 3] ~ cost[degree == 3],
type = "o", col = rainbow(5)[4])
lines(error[degree == 4] ~ cost[degree == 4],
type = "o", col = rainbow(5)[5])
})
legend("bottom", horiz = T, legend = c(1:5), col = rainbow(5),
lty = 1, cex = .75, title = "degree")
```



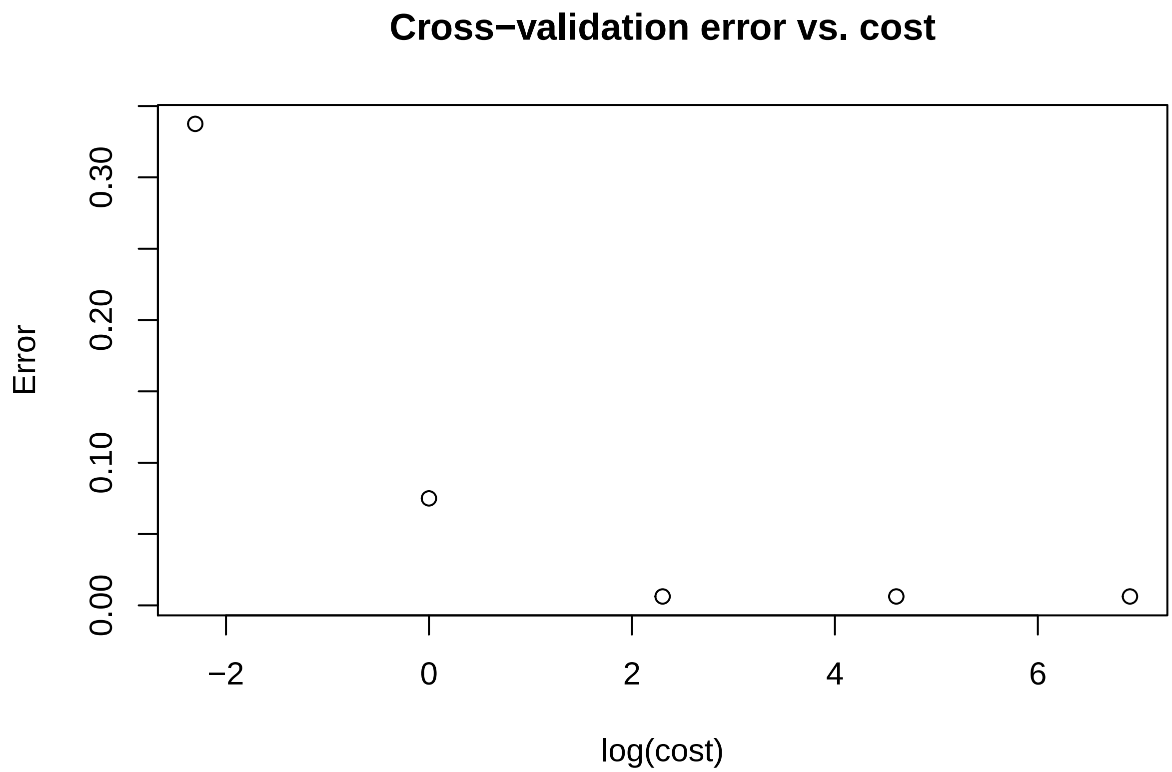
Hence the best choice of degree is 1.

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

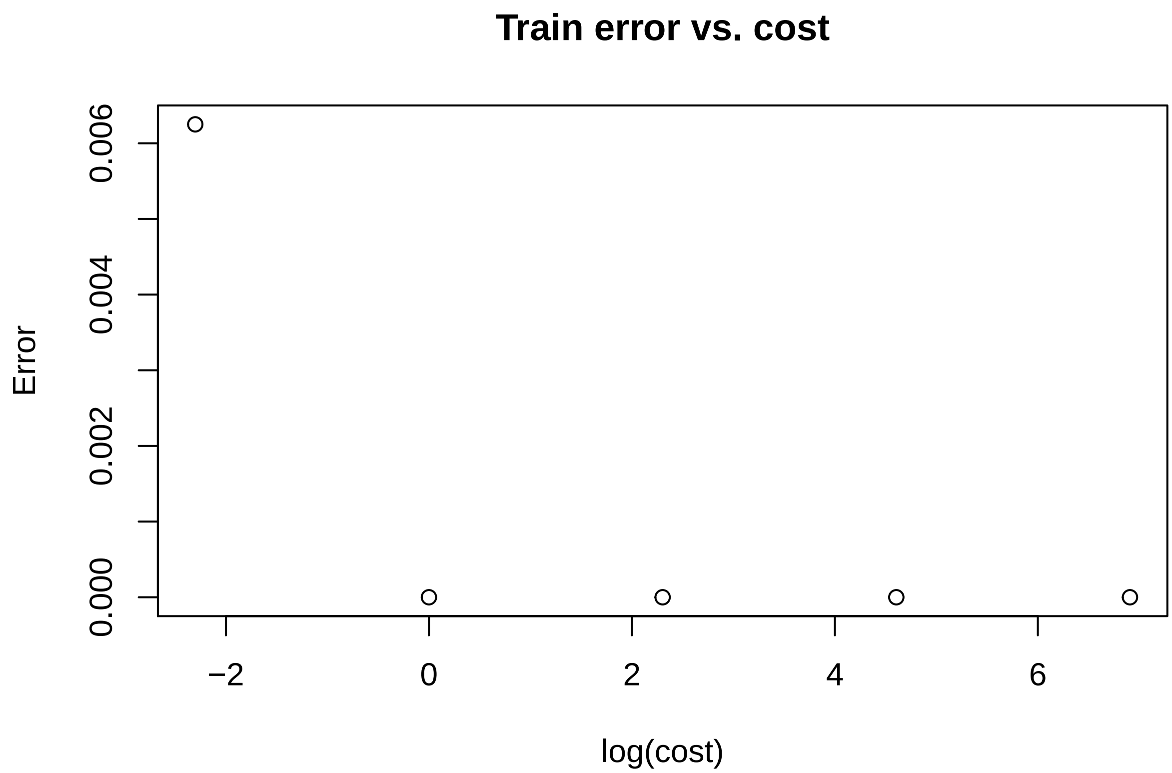
```
##
## Call:
## best.tune(method = svm, train.x = sp ~ . - index, data = training,
##     ranges = list(cost = c(0.1, 1, 10, 100, 1000), gamma = c(0.5,
##         1, 2, 3, 4), degree = c(1, 2, 3, 4, 5)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##
## Number of Support Vectors:  43
##
##  ( 21 22 )
##
##
## Number of Classes:  2
##
## Levels:
##  B O
```

We take the lograithm of the cost to make plots for better visualization

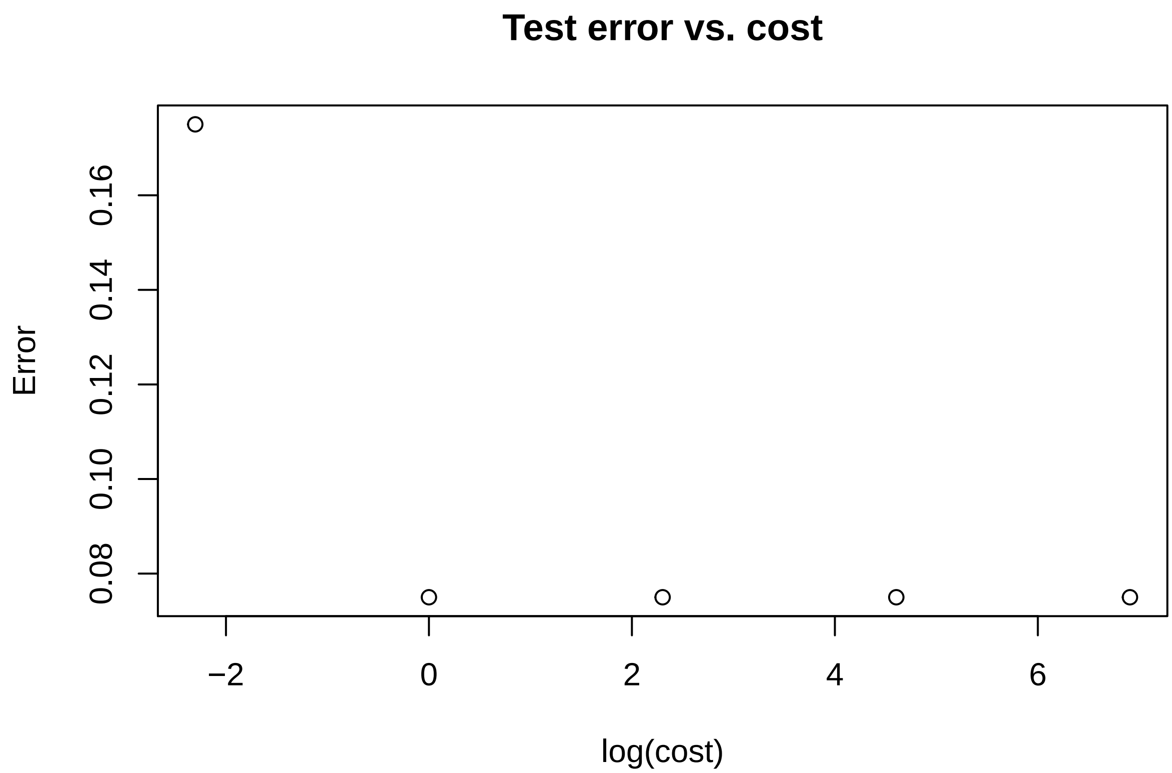
```r
ranges =  c(0.1, 1, 10, 100, 1000)

svmTrainErr = vector(length = length(ranges))
svmTestErr = vector(length = length(ranges))


for (i in 1:length(ranges)) {
svm.model = svm(sp ~ .-index, data = training, kernel = "radial", cost = ranges[i],degree = 1, gamma = 
pred.train <- predict(svm.model, training)
pred.test<-predict(svm.model,testing)
svmTrainErr[i] = mean(pred.train != training$sp)
svmTestErr[i] = mean(pred.test != testing$sp)
}


selected <-tune.out$performances$gamma == 0.5 & tune.out$performances$degree == 1
plot(log(ranges),subset(tune.out$performances,selected)$error,main = "Cross-validation error vs. cost",
```




```r
plot(log(ranges),svmTrainErr,main = "Train error vs. cost", xlab = "log(cost)",ylab = "Error" )
```




```
plot(log(ranges),svmTestErr,main = "Test error vs. cost", xlab = "log(cost)",ylab = "Error" )
```



The best model is obtained by $cost \geq 10$, Since the cv-error, train error and test error are all minimized.

The best model is obtained by $cost \geq 1$, Since the cv-error, train error and test error are all minimized.