# 5241 HW3P3

*Mengjia Huang*

*2018/3/20*

## R Markdown

```
setwd("/Users/minta/Desktop/5241 HW3 3.22")
d6=read.table("train.6.txt",header = F,sep=',')
d5=read.table("train.5.txt",header = F,sep=',')
x=rbind(as.matrix(d5),as.matrix(d6))
y=rep(c(-1,1),c(nrow(d5),nrow(d6)))

#Randomly select about 20% of the data and set it aside as a test set.
set.seed(1089)
h=nrow(x)*0.2
h
```

```
## [1] 244
```

```
test.index<-sort(sample(1:nrow(x),h))
xtest=x[test.index,]
ytest=y[test.index]

xtrain=x[-test.index,]
ytrain=y[-test.index]
```

### Linear Kernel

```
set.seed(1089)
###Linear Kernel
#Train a linear SVM with soft margin. Cross-validate the margin parameter.
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.3.2
```

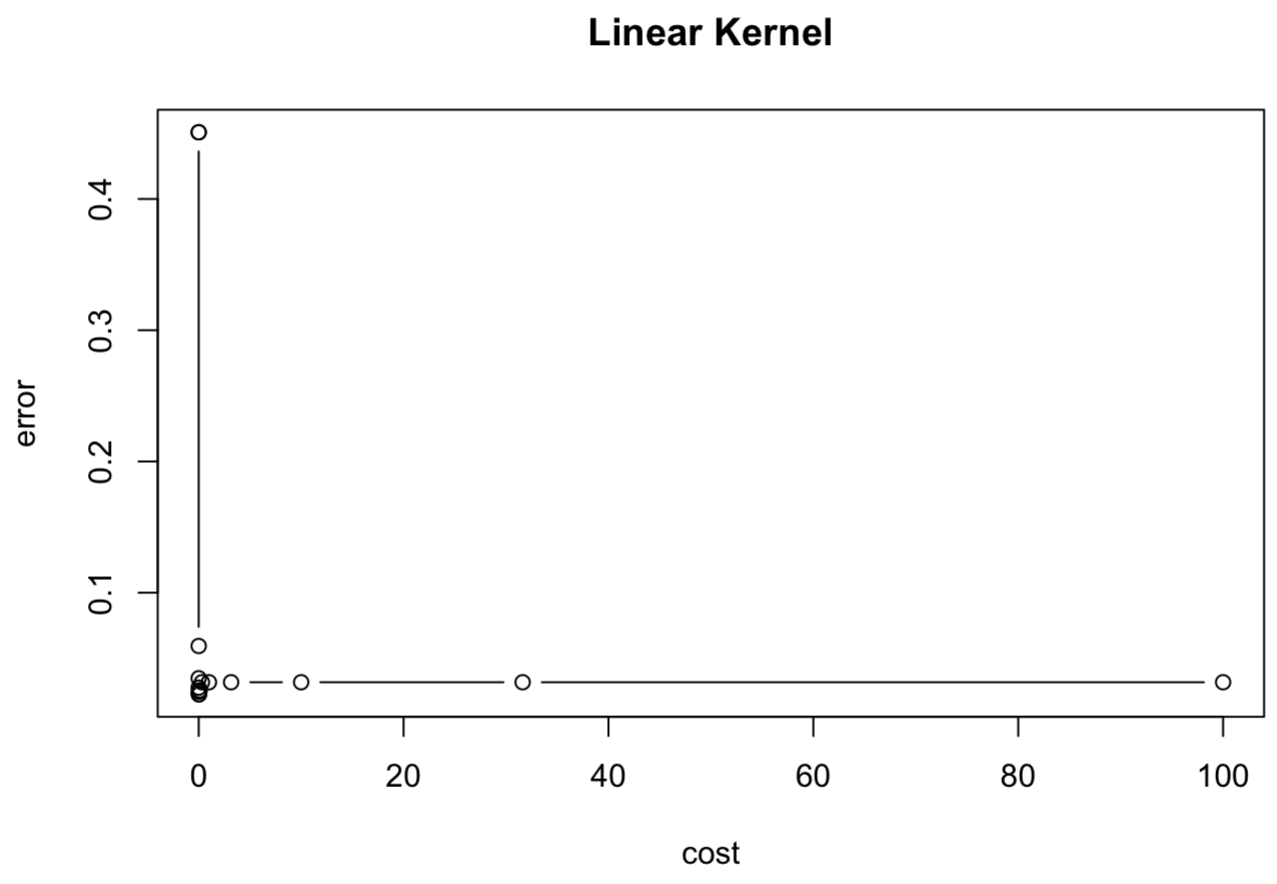
```
traindata=data.frame(x=xtrain, y=as.factor(ytrain))
dim(traindata)
```

```
## [1] 976 257
```

```
#By default, tune() performs ten-fold cross-validation
tune.out=tune(svm, y~.,data=traindata,kernel="linear",
              ranges=list(cost=10^(seq(-5, 2, 0.5))),scale=FALSE)
#cross-validation error rates
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##        cost
##  0.03162278
##
## - best performance: 0.0225647
##
## - Detailed performance results:
##            cost      error dispersion
## 1  1.000000e-05 0.45084157 0.05511213
## 2  3.162278e-05 0.45084157 0.05511213
## 3  1.000000e-04 0.05940459 0.02503456
## 4  3.162278e-04 0.03482011 0.01686796
## 5  1.000000e-03 0.02765622 0.01810399
## 6  3.162278e-03 0.02457395 0.01942805
## 7  1.000000e-02 0.02257522 0.02267001
## 8  3.162278e-02 0.02256470 0.02046425
## 9  1.000000e-01 0.02562592 0.02065170
## 10 3.162278e-01 0.03177993 0.02536686
## 11 1.000000e+00 0.03177993 0.02536686
## 12 3.162278e+00 0.03177993 0.02536686
## 13 1.000000e+01 0.03177993 0.02536686
## 14 3.162278e+01 0.03177993 0.02536686
## 15 1.000000e+02 0.03177993 0.02536686
```

```
#plot the misclassification rates as a function of the margin parameter in the linear
 case
plot(tune.out,main="Linear Kernel")
```



```
#best model
bestmod=tune.out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = traindata, ranges = list(cost = 10
^(seq(-5,
##     2, 0.5))), kernel = "linear", scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.03162278
##       gamma:  0.00390625
##
## Number of Support Vectors:  97
##
##  ( 44 53 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```
#predict the class label on a set of test observations
testdata=data.frame(x=xtest, y=as.factor(ytest))
#train with the Cost parameter=0.03162278 selected via CV
svmfit=svm(y~., data=traindata, kernel="linear", cost=0.03162278, scale=FALSE)
ypred1=predict(svmfit,testdata) #method1
ypred2=predict(bestmod,testdata) #method2
#test error
sum(ypred1 != ytest)/length(ytest)
```

```
## [1] 0.01639344
```

```
sum(ypred2 != ytest)/length(ytest) #1.64% of test observations are misclassified by t
his SVM.
```

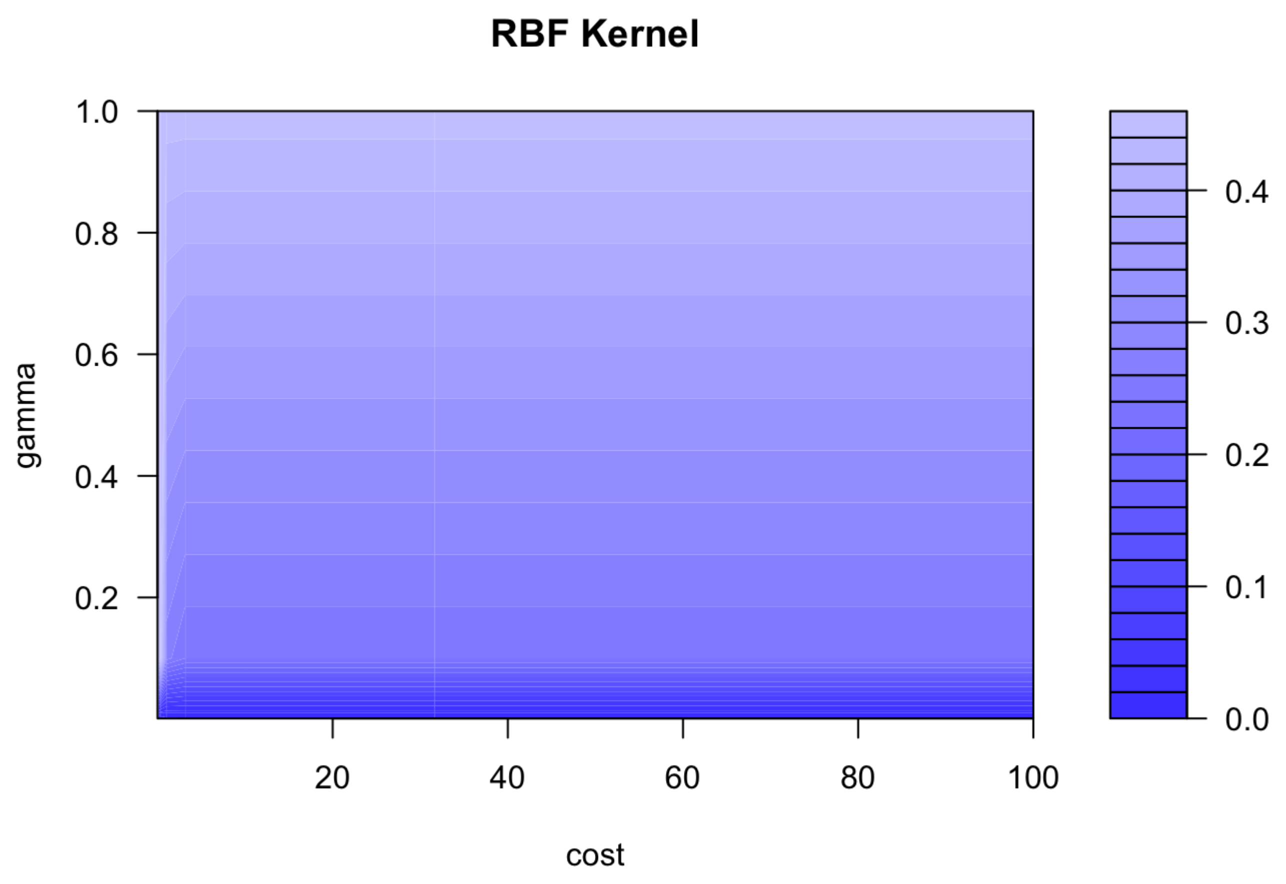
```
## [1] 0.01639344
```

### RBF Kernel

```
set.seed(1089)
###RBF Kernel
#Train an SVM with soft margin and RBF kernel.
#cross-validate both the soft-margin parameter and the kernel bandwidth.
traindata=data.frame(x=xtrain, y=as.factor(ytrain))
tune.out1=tune(svm, y~., data=traindata, kernel="radial",
ranges=list(cost=10^(seq(-5, 2, 0.5)),gamma=c(0.001,0.01,0.1,1)),scale=FALSE)
#cross-validation error rates
summary(tune.out1)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1  0.01
##
## - best performance: 0.009236272
##
## - Detailed performance results:
##             cost gamma       error  dispersion
## 1  1.000000e-05 0.001 0.450841574 0.055112128
## 2  3.162278e-05 0.001 0.450841574 0.055112128
## 3  1.000000e-04 0.001 0.450841574 0.055112128
## 4  3.162278e-04 0.001 0.450841574 0.055112128
## 5  1.000000e-03 0.001 0.450841574 0.055112128
## 6  3.162278e-03 0.001 0.450841574 0.055112128
## 7  1.000000e-02 0.001 0.450841574 0.055112128
## 8  3.162278e-02 0.001 0.229507679 0.053443362
## 9  1.000000e-01 0.001 0.043025458 0.019239211
## 10 3.162278e-01 0.001 0.025604881 0.015451405
## 11 1.000000e+00 0.001 0.022533137 0.018585931
## 12 3.162278e+00 0.001 0.019471912 0.018382352
## 13 1.000000e+01 0.001 0.015369240 0.015470432
## 14 3.162278e+01 0.001 0.016410688 0.017594324
## 15 1.000000e+02 0.001 0.017420576 0.017439841
## 16 1.000000e-05 0.010 0.450841574 0.055112128
## 17 3.162278e-05 0.010 0.450841574 0.055112128
## 18 1.000000e-04 0.010 0.450841574 0.055112128
## 19 3.162278e-04 0.010 0.450841574 0.055112128
## 20 1.000000e-03 0.010 0.450841574 0.055112128
## 21 3.162278e-03 0.010 0.450841574 0.055112128
## 22 1.000000e-02 0.010 0.449810646 0.057252222
## 23 3.162278e-02 0.010 0.028718704 0.016660317
## 24 1.000000e-01 0.010 0.016421208 0.013060827
## 25 3.162278e-01 0.010 0.012329055 0.012661712
## 26 1.000000e+00 0.010 0.009236272 0.009021575
## 27 3.162278e+00 0.010 0.009236272 0.009021575
## 28 1.000000e+01 0.010 0.009236272 0.009021575
## 29 3.162278e+01 0.010 0.009236272 0.009021575
## 30 1.000000e+02 0.010 0.009236272 0.009021575
## 31 1.000000e-05 0.100 0.450841574 0.055112128
## 32 3.162278e-05 0.100 0.450841574 0.055112128
## 33 1.000000e-04 0.100 0.450841574 0.055112128
## 34 3.162278e-04 0.100 0.450841574 0.055112128
## 35 1.000000e-03 0.100 0.450841574 0.055112128
## 36 3.162278e-03 0.100 0.450841574 0.055112128
## 37 1.000000e-02 0.100 0.450841574 0.055112128
## 38 3.162278e-02 0.100 0.450841574 0.055112128
## 39 1.000000e-01 0.100 0.450841574 0.055112128
## 40 3.162278e-01 0.100 0.450841574 0.055112128
## 41 1.000000e+00 0.100 0.267662529 0.113211298
## 42 3.162278e+00 0.100 0.240006312 0.099845981
## 43 1.000000e+01 0.100 0.240006312 0.099845981
## 44 3.162278e+01 0.100 0.240006312 0.099845981
## 45 1.000000e+02 0.100 0.240006312 0.099845981
## 46 1.000000e-05 1.000 0.450841574 0.055112128
## 47 3.162278e-05 1.000 0.450841574 0.055112128
## 48 1.000000e-04 1.000 0.450841574 0.055112128
## 49 3.162278e-04 1.000 0.450841574 0.055112128
## 50 1.000000e-03 1.000 0.450841574 0.055112128
## 51 3.162278e-03 1.000 0.450841574 0.055112128
## 52 1.000000e-02 1.000 0.450841574 0.055112128
## 53 3.162278e-02 1.000 0.450841574 0.055112128
## 54 1.000000e-01 1.000 0.450841574 0.055112128
## 55 3.162278e-01 1.000 0.450841574 0.055112128
## 56 1.000000e+00 1.000 0.450841574 0.055112128
## 57 3.162278e+00 1.000 0.450841574 0.055112128
## 58 1.000000e+01 1.000 0.450841574 0.055112128
## 59 3.162278e+01 1.000 0.450841574 0.055112128
## 60 1.000000e+02 1.000 0.450841574 0.055112128
```

```
#plot the misclassification rates a function of the margin parameter and the kernel b
andwidth
plot(tune.out1,main="RBF Kernel")
```



```
#best model
bestmod1=tune.out1$best.model
summary(bestmod1)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = traindata, ranges = list(cost = 10
^(seq(-5,
##     2, 0.5)), gamma = c(0.001, 0.01, 0.1, 1)), kernel = "radial",
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.01
##
## Number of Support Vectors:  254
##
##  ( 135 119 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```
#predict the class label on a set of test observations
testdata=data.frame(x=xtest, y=as.factor(ytest))
#train with the parameters(cost=1  gamma=0.01) selected via CV
svmfit1=svm(y~., data=traindata, kernel="radial",gamma=0.01, cost=1, scale=FALSE)
ypred11=predict(svmfit1,testdata) #method1
ypred22=predict(bestmod1,testdata) #method2
#test error
sum(ypred11 != ytest)/length(ytest)
```

```
## [1] 0.004098361
```

```
sum(ypred22 != ytest)/length(ytest) #0.41% of test observations are misclassified by
 this SVM.
```

```
## [1] 0.004098361
```

Since 1.64% of test observations are misclassified by using Linear SVM, and 0.41% of test observations are misclassified by using non-linear SVM, we should use the non-linear one.