Data 3 HW 6

Sean Duan

11/9/2020

1.

A

```
#A
tune.out=tune(svm ,Purchase~.,data=OJ[train,] ,kernel ="linear", ranges=list(cost=c(0.01, 0.05, 0.1, 0.5
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.17125
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.17375 0.03884174
## 2  0.05 0.18000 0.03073181
## 3  0.10 0.17875 0.03064696
## 4  0.50 0.17875 0.03064696
## 5  1.00 0.17500 0.03061862
## 6  5.00 0.17250 0.03322900
## 7 10.00 0.17125 0.03488573
```

```
#best cost para 0.1
bestmod=tune.out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = OJ[train,
##     ], ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 5, 10)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
```

```
##
## Number of Support Vectors:  326
##
##  ( 162 164 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
#training error
ypred=predict(bestmod ,OJ[train,])
table(predict=ypred , truth=OJ[train,]$Purchase )
```

```
##        truth
## predict  CH  MM
##      CH 423  69
##      MM  62 246
```

```r
1-mean(ypred==OJ[train,]$Purchase)
```

```
## [1] 0.16375
```

```r
#16.5% error

#test error
ypred=predict(bestmod ,OJ[-train,])
table(predict=ypred , truth=OJ[-train,]$Purchase )
```

```
##        truth
## predict  CH  MM
##      CH 156  28
##      MM  12  74
```

```r
1-mean(ypred==OJ[-train,]$Purchase)
```

```
## [1] 0.1481481
```

```r
# wow even lower test err, 16.3%!
```

Looking at our support vector classifier, we were able to find a training error of 16.5%, and a test error of 16.3%. Our best values for the cost parameter was 0.1.

B

```r
#B
tune.out=tune(svm ,Purchase~.,data=OJ[train,] ,kernel ="polynomial", degree=2, ranges=list(cost=c(0.01,
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     5
```

```
##
## - best performance: 0.17625
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.39375 0.06568284
## 2  0.05 0.34375 0.06325928
## 3  0.10 0.32000 0.05809475
## 4  0.50 0.20125 0.05665747
## 5  1.00 0.19875 0.06248611
## 6  5.00 0.17625 0.04656611
## 7 10.00 0.17750 0.04116363
```

```r
#best cost para 10
bestmod=tune.out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = OJ[train,
##     ], ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 5, 10)), kernel = "polynomial",
##     degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  5
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  368
##
##  ( 188 180 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
#training error
ypred=predict(bestmod ,OJ[train,])
table(predict=ypred , truth=OJ[train,]$Purchase )
```

```
##        truth
## predict  CH  MM
##      CH 447  86
##      MM  38 229
```

```r
1-mean(ypred==OJ[train,]$Purchase)
```

```
## [1] 0.155
```

```r
#15% error

#test error
```

```r
ypred=predict(bestmod ,OJ[-train,])
table(predict=ypred , truth=OJ[-train,]$Purchase )
```

```
##        truth
## predict  CH  MM
##      CH 155  36
##      MM  13  66
```

```r
1-mean(ypred==OJ[-train,]$Purchase)
```

```
## [1] 0.1814815
```

```r
#18.9% err
```

Looking at our support vector machine, we were able to find a training error of 15%, and a test error of 18.9%. Our best values for the cost parameter was 10.

C

```r
#C
tune.out=tune(svm ,Purchase~.,data=OJ[train,] ,kernel ="radial",
              ranges=list(cost=c(0.01, 0.05, 0.1, 0.5, 1, 5, 10),
                          gamma=c(0.001, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 100)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1  0.01
##
## - best performance: 0.17125
##
## - Detailed performance results:
##     cost gamma   error dispersion
## 1   0.01 1e-03 0.39375 0.08191501
## 2   0.05 1e-03 0.39375 0.08191501
## 3   0.10 1e-03 0.39375 0.08191501
## 4   0.50 1e-03 0.32000 0.08211611
## 5   1.00 1e-03 0.18625 0.05905800
## 6   5.00 1e-03 0.17625 0.03458584
## 7  10.00 1e-03 0.17375 0.03356689
## 8   0.01 1e-02 0.39375 0.08191501
## 9   0.05 1e-02 0.36875 0.08625101
## 10  0.10 1e-02 0.20625 0.06594663
## 11  0.50 1e-02 0.18250 0.03689324
## 12  1.00 1e-02 0.17125 0.04168749
## 13  5.00 1e-02 0.17125 0.04168749
## 14 10.00 1e-02 0.17500 0.03952847
## 15  0.01 5e-02 0.39375 0.08191501
## 16  0.05 5e-02 0.21125 0.07417369
## 17  0.10 5e-02 0.18625 0.03928617
```

```
## 18  0.50 5e-02 0.17750 0.03944053
## 19  1.00 5e-02 0.17250 0.04281744
## 20  5.00 5e-02 0.19250 0.04830459
## 21 10.00 5e-02 0.19250 0.04866267
## 22  0.01 1e-01 0.39375 0.08191501
## 23  0.05 1e-01 0.23125 0.06488505
## 24  0.10 1e-01 0.19625 0.04489571
## 25  0.50 1e-01 0.18125 0.04973890
## 26  1.00 1e-01 0.19000 0.04362084
## 27  5.00 1e-01 0.20375 0.05466120
## 28 10.00 1e-01 0.20875 0.04896498
## 29  0.01 5e-01 0.39375 0.08191501
## 30  0.05 5e-01 0.39375 0.08191501
## 31  0.10 5e-01 0.28000 0.06406377
## 32  0.50 5e-01 0.20750 0.04972145
## 33  1.00 5e-01 0.22000 0.04495368
## 34  5.00 5e-01 0.22625 0.05118390
## 35 10.00 5e-01 0.21875 0.05111602
## 36  0.01 1e+00 0.39375 0.08191501
## 37  0.05 1e+00 0.39375 0.08191501
## 38  0.10 1e+00 0.34750 0.08096639
## 39  0.50 1e+00 0.21250 0.05000000
## 40  1.00 1e+00 0.22250 0.04518481
## 41  5.00 1e+00 0.22375 0.05084358
## 42 10.00 1e+00 0.22625 0.04875178
## 43  0.01 5e+00 0.39375 0.08191501
## 44  0.05 5e+00 0.39375 0.08191501
## 45  0.10 5e+00 0.39375 0.08191501
## 46  0.50 5e+00 0.25250 0.06476453
## 47  1.00 5e+00 0.23125 0.06325928
## 48  5.00 5e+00 0.24000 0.05974483
## 49 10.00 5e+00 0.25125 0.05510407
## 50  0.01 1e+01 0.39375 0.08191501
## 51  0.05 1e+01 0.39375 0.08191501
## 52  0.10 1e+01 0.39375 0.08191501
## 53  0.50 1e+01 0.28625 0.07082108
## 54  1.00 1e+01 0.25375 0.06375136
## 55  5.00 1e+01 0.26250 0.06208194
## 56 10.00 1e+01 0.27000 0.05779514
## 57  0.01 1e+02 0.39375 0.08191501
## 58  0.05 1e+02 0.39375 0.08191501
## 59  0.10 1e+02 0.39375 0.08191501
## 60  0.50 1e+02 0.36375 0.07533490
## 61  1.00 1e+02 0.30875 0.06181570
## 62  5.00 1e+02 0.30750 0.05927806
## 63 10.00 1e+02 0.30750 0.05927806
```

```r
#best cost para 5, gamma .001
bestmod=tune.out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = OJ[train,
##     ], ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 5, 10), gamma = c(0.001,
```

```
##     0.01, 0.05, 0.1, 0.5, 1, 5, 10, 100)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  406
##
##  ( 204 202 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
#training error
ypred=predict(bestmod ,OJ[train,])
table(predict=ypred , truth=OJ[train,]$Purchase )
```

```
##        truth
## predict  CH  MM
##      CH 430  71
##      MM  55 244
```

```
1-mean(ypred==OJ[train,]$Purchase)
```

```
## [1] 0.1575
```

```
#17.4% error

#test error
ypred=predict(bestmod ,OJ[-train,])
table(predict=ypred , truth=OJ[-train,]$Purchase )
```

```
##        truth
## predict  CH  MM
##      CH 153  34
##      MM  15  68
```

```
1-mean(ypred==OJ[-train,]$Purchase)
```

```
## [1] 0.1814815
```

```
#18.1% err
```

Looking at our support vector machine, we were able to find a training error of 17.4%, and a test error of 18.1%. Our best values for the cost parameter was 5, and 0.001 for the gamma parameter.

D

```
#D
tune.out=tune(svm ,Purchase~.,data=OJ[train,] ,kernel ="sigmoid",
              ranges=list(cost=c(0.01, 0.05, 0.1, 0.5, 1, 5, 10),
                          gamma=c(0.001, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 100)))
```

```r
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10 0.001
##
## - best performance: 0.1725
##
## - Detailed performance results:
##     cost gamma   error dispersion
## 1   0.01 1e-03 0.39375 0.06852868
## 2   0.05 1e-03 0.39375 0.06852868
## 3   0.10 1e-03 0.39375 0.06852868
## 4   0.50 1e-03 0.39375 0.06852868
## 5   1.00 1e-03 0.31500 0.07745967
## 6   5.00 1e-03 0.17375 0.04387878
## 7  10.00 1e-03 0.17250 0.04518481
## 8   0.01 1e-02 0.39375 0.06852868
## 9   0.05 1e-02 0.39375 0.06852868
## 10  0.10 1e-02 0.31875 0.07822910
## 11  0.50 1e-02 0.17500 0.04487637
## 12  1.00 1e-02 0.17250 0.04518481
## 13  5.00 1e-02 0.17375 0.04267529
## 14 10.00 1e-02 0.17750 0.04556741
## 15  0.01 5e-02 0.39375 0.06852868
## 16  0.05 5e-02 0.19250 0.05006940
## 17  0.10 5e-02 0.19625 0.04332131
## 18  0.50 5e-02 0.28000 0.04338138
## 19  1.00 5e-02 0.29875 0.04693746
## 20  5.00 5e-02 0.27375 0.06781562
## 21 10.00 5e-02 0.28375 0.07972392
## 22  0.01 1e-01 0.38750 0.07430231
## 23  0.05 1e-01 0.22375 0.04619178
## 24  0.10 1e-01 0.29375 0.05535554
## 25  0.50 1e-01 0.32750 0.05263871
## 26  1.00 1e-01 0.29125 0.07023028
## 27  5.00 1e-01 0.29250 0.06072479
## 28 10.00 1e-01 0.29875 0.06276333
## 29  0.01 5e-01 0.28125 0.08212668
## 30  0.05 5e-01 0.35375 0.07729139
## 31  0.10 5e-01 0.35500 0.06487167
## 32  0.50 5e-01 0.38125 0.07246886
## 33  1.00 5e-01 0.38750 0.08312474
## 34  5.00 5e-01 0.39000 0.08032054
## 35 10.00 5e-01 0.38875 0.07981097
## 36  0.01 1e+00 0.29375 0.07412686
## 37  0.05 1e+00 0.36125 0.07393926
## 38  0.10 1e+00 0.37000 0.07932003
## 39  0.50 1e+00 0.37750 0.07066156
```

```
## 40  1.00 1e+00 0.37125 0.06923440
## 41  5.00 1e+00 0.39250 0.08232726
## 42 10.00 1e+00 0.38000 0.06977145
## 43  0.01 5e+00 0.30250 0.07472171
## 44  0.05 5e+00 0.38000 0.07910085
## 45  0.10 5e+00 0.40125 0.07915570
## 46  0.50 5e+00 0.42375 0.09288022
## 47  1.00 5e+00 0.41375 0.08788605
## 48  5.00 5e+00 0.42125 0.09736993
## 49 10.00 5e+00 0.41000 0.09257129
## 50  0.01 1e+01 0.30250 0.07945124
## 51  0.05 1e+01 0.37125 0.07218081
## 52  0.10 1e+01 0.39000 0.07610300
## 53  0.50 1e+01 0.40500 0.08522845
## 54  1.00 1e+01 0.40500 0.08998457
## 55  5.00 1e+01 0.40500 0.08842762
## 56 10.00 1e+01 0.40500 0.08842762
## 57  0.01 1e+02 0.30125 0.07417369
## 58  0.05 1e+02 0.37750 0.05974483
## 59  0.10 1e+02 0.40000 0.06373774
## 60  0.50 1e+02 0.40875 0.07096801
## 61  1.00 1e+02 0.41250 0.07728015
## 62  5.00 1e+02 0.41375 0.07625114
## 63 10.00 1e+02 0.41375 0.07625114
```

```r
#why is this fitting so weirdly???
#best cost para 10, gamma .001
bestmod=tune.out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = OJ[train,
##     ], ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 5, 10), gamma = c(0.001,
##     0.01, 0.05, 0.1, 0.5, 1, 5, 10, 100)), kernel = "sigmoid")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  sigmoid
##        cost:  10
##      coef.0:  0
##
## Number of Support Vectors:  435
##
##  ( 219 216 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
#training error
ypred=predict(bestmod ,OJ[train,])
```

```r
table(predict=ypred , truth=OJ[train,]$Purchase )
```

```
##        truth
## predict  CH  MM
##      CH 420  75
##      MM  65 240
```

```r
1-mean(ypred==OJ[train,]$Purchase)
```

```
## [1] 0.175
```

```r
#17.5% error

#test error
ypred=predict(bestmod ,OJ[-train,])
table(predict=ypred , truth=OJ[-train,]$Purchase )
```

```
##        truth
## predict  CH  MM
##      CH 153  33
##      MM  15  69
```

```r
1-mean(ypred==OJ[-train,]$Purchase)
```

```
## [1] 0.1777778
```

```r
#17.8% err
```

Looking at our support vector machine, we were able to find a training error of 17.5%, and a test error of 17.8%. Our best values for the cost parameter was 10, and 0.001 for the gamma parameter.

E

```r
#E
glm1<-glm(Purchase~., data = OJ[train,], family = binomial)
summary(glm1)
```

```
##
## Call:
## glm(formula = Purchase ~ ., family = binomial, data = OJ[train,
##     ])
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8328  -0.5251  -0.2307   0.5388   2.7279
##
## Coefficients: (5 not defined because of singularities)
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)      3.68460    2.36076   1.561  0.11858
## WeekofPurchase  -0.01733    0.01270  -1.365  0.17235
## StoreID         -0.17599    0.15690  -1.122  0.26203
## PriceCH          5.33061    2.11897   2.516  0.01188 *
## PriceMM         -2.95257    1.02348  -2.885  0.00392 **
## DiscCH          -7.93779   22.11783  -0.359  0.71968
## DiscMM          20.25555   10.52014   1.925  0.05418 .
## SpecialCH        0.16968    0.39019   0.435  0.66366
## SpecialMM        0.44348    0.32099   1.382  0.16710
```

```
## LoyalCH         -6.03395    0.44640 -13.517  < 2e-16 ***
## SalePriceMM            NA         NA      NA       NA
## SalePriceCH            NA         NA      NA       NA
## PriceDiff              NA         NA      NA       NA
## Store7Yes         0.08618    0.81124   0.106  0.91540
## PctDiscMM       -37.69732   22.00928  -1.713  0.08675 .
## PctDiscCH         6.66251   41.54761   0.160  0.87260
## ListPriceDiff          NA         NA      NA       NA
## STORE                  NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1072.63  on 799  degrees of freedom
## Residual deviance:  608.19  on 787  degrees of freedom
## AIC: 634.19
##
## Number of Fisher Scoring iterations: 6
```

```
#training err
glm.probs=predict(glm1 ,OJ[train,], type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
glm.pred=rep("CH",800)
glm.pred[glm.probs >.5]="MM"
table(predict=glm.pred , truth=OJ[train,]$Purchase )
```

```
##        truth
## predict  CH  MM
##      CH 420  73
##      MM  65 242
```

```
1-mean(glm.pred==OJ[train,]$Purchase)
```

```
## [1] 0.1725
```

```
#17.3% err

#test err
glm.probs=predict(glm1 ,OJ[-train,], type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
glm.pred=rep("CH",270)
glm.pred[glm.probs >.5]="MM"
table(predict=glm.pred , truth=OJ[-train,]$Purchase )
```

```
##        truth
## predict  CH  MM
##      CH 155  29
##      MM  13  73
```

```
1-mean(glm.pred==OJ[-train,]$Purchase)
```

```
## [1] 0.1555556
```

```r
#15.6% err
```

Looking at our logistic regression, we were able to find a training error of 17.3%, and a test error of 15.6%.

F - Training Data

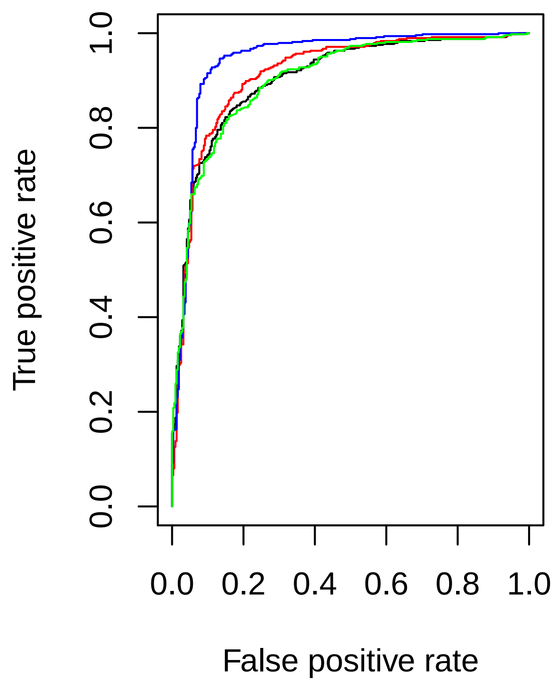
```r
#F
rocplot =function (pred , truth , ...){
  predob = prediction (pred , truth)
  perf = performance (predob , "tpr", "fpr")
  plot(perf ,...)}

par(mfrow=c(1,2))

## ROC for training data
#m1
svmfit.opt=svm(Purchase~., data=OJ[train,], kernel ="linear",gamma=2, cost=0.1, decision.values =T)
fitted =attributes (predict (svmfit.opt ,OJ[train ,], decision.values=TRUE))$decision.values

##try alternative code from Wikle
predob<-prediction(fitted, OJ[train,]$Purchase, label.ordering = c("MM","CH"))
perf=performance(predob, "tpr","fpr")
plot(perf)
#this works correctly!


#m2
svmfit.2=svm(Purchase~., data=OJ[train,], kernel ="polynomial",degree=2, cost=10, decision.values =T)
fitted =attributes (predict (svmfit.2 ,OJ[train ,], decision.values=TRUE))$decision.values
predob<-prediction(fitted, OJ[train,]$Purchase, label.ordering = c("MM","CH"))
perf=performance(predob, "tpr","fpr")
plot(perf, add = T, col = "red")



#m3
svmfit.3=svm(Purchase~., data=OJ[train,], kernel ="radial",gamma=5, cost=0.001, decision.values =T)
fitted =attributes (predict (svmfit.3 ,OJ[train ,], decision.values=TRUE))$decision.values
predob<-prediction(fitted, OJ[train,]$Purchase, label.ordering = c("MM","CH"))
perf=performance(predob, "tpr","fpr")
plot(perf, add = T, col = "blue")


#m4
svmfit.4=svm(Purchase~., data=OJ[train,], kernel ="sigmoid",gamma=0.001, cost=10, decision.values =T)
fitted =attributes (predict (svmfit.4 ,OJ[train ,], decision.values=TRUE))$decision.values
predob<-prediction(fitted, OJ[train,]$Purchase, label.ordering = c("MM","CH"))
perf=performance(predob, "tpr","fpr")
plot(perf, add = T, col = "green")

#m5
test_prob = predict(glm1, newdata = OJ[train,], type = "response")
```

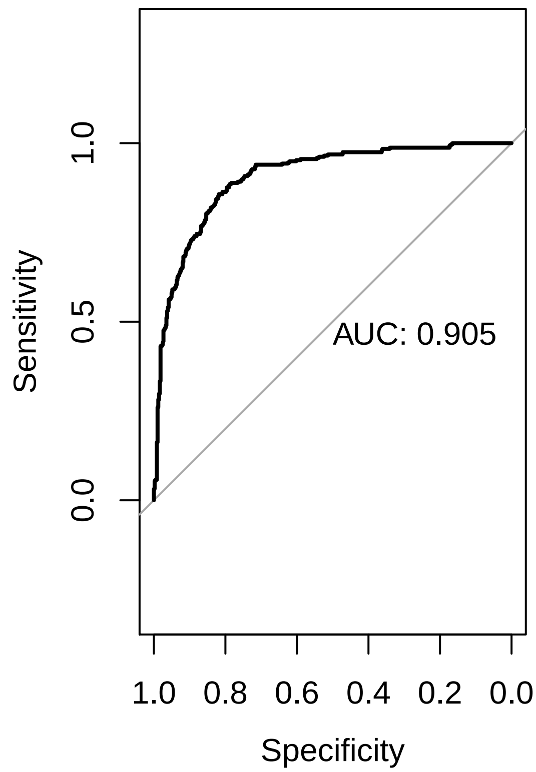
```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```r
test_roc = roc(OJ[train,]$Purchase ~ test_prob, plot = TRUE, print.auc = TRUE)
```

```
## Setting levels: control = CH, case = MM
```

```
## Setting direction: controls < cases
```





F - Testing Data

```r
par(mfrow=c(1,2))


##ROC for test data
#m1
svmfit.opt=svm(Purchase~., data=OJ[-train,], kernel ="linear",gamma=2, cost=0.1, decision.values =T)
fitted =attributes (predict (svmfit.opt ,OJ[train ,], decision.values=TRUE))$decision.values
predob<-prediction(fitted, OJ[train,]$Purchase, label.ordering = c("MM","CH"))
perf=performance(predob, "tpr","fpr")
plot(perf)


#m2
svmfit.2=svm(Purchase~., data=OJ[-train,], kernel ="polynomial",degree=2, cost=10, decision.values =T)
fitted =attributes (predict (svmfit.2 ,OJ[train ,], decision.values=TRUE))$decision.values
predob<-prediction(fitted, OJ[train,]$Purchase, label.ordering = c("MM","CH"))
```

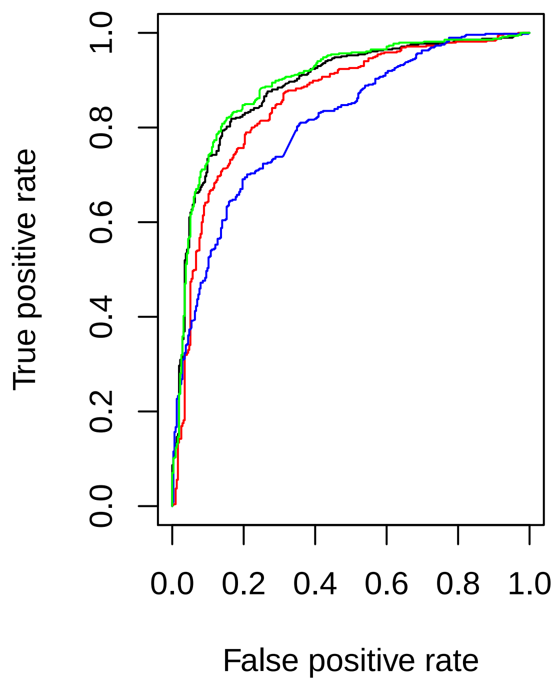
```r
perf=performance(predob, "tpr","fpr")
plot(perf, add = T, col = "red")


#m3
svmfit.3=svm(Purchase~., data=OJ[-train,], kernel ="radial",gamma=5, cost=0.001, decision.values =T)
fitted =attributes (predict (svmfit.3 ,OJ[train ,], decision.values=TRUE))$decision.values
predob<-prediction(fitted, OJ[train,]$Purchase, label.ordering = c("MM","CH"))
perf=performance(predob, "tpr","fpr")
plot(perf, add = T, col = "blue")


#m4
svmfit.4=svm(Purchase~., data=OJ[-train,], kernel ="sigmoid",gamma=0.001, cost=10, decision.values =T)
fitted =attributes (predict (svmfit.4 ,OJ[train ,], decision.values=TRUE))$decision.values
predob<-prediction(fitted, OJ[train,]$Purchase, label.ordering = c("MM","CH"))
perf=performance(predob, "tpr","fpr")
plot(perf, add = T, col = "green")

#m5
test_prob = predict(glm1, newdata = OJ[-train,], type = "response")
```

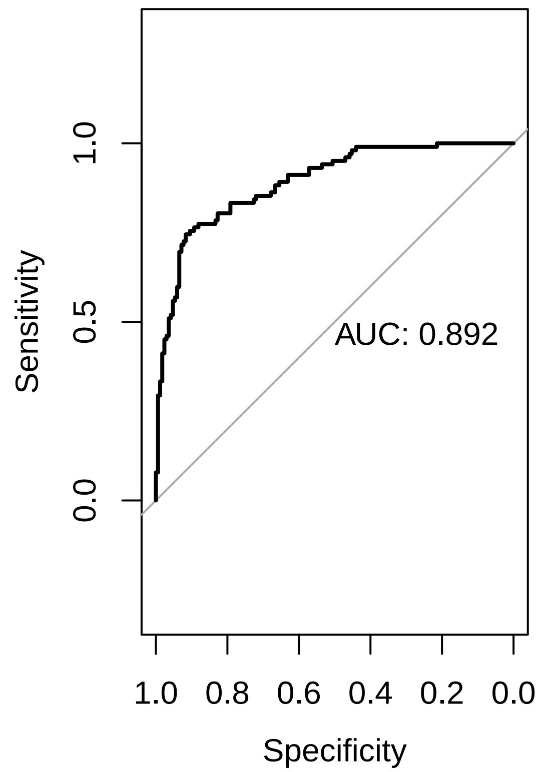
```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```r
test_roc = roc(OJ[-train,]$Purchase ~ test_prob, plot = TRUE, print.auc = TRUE)
```

```
## Setting levels: control = CH, case = MM
```

```
## Setting direction: controls < cases
```





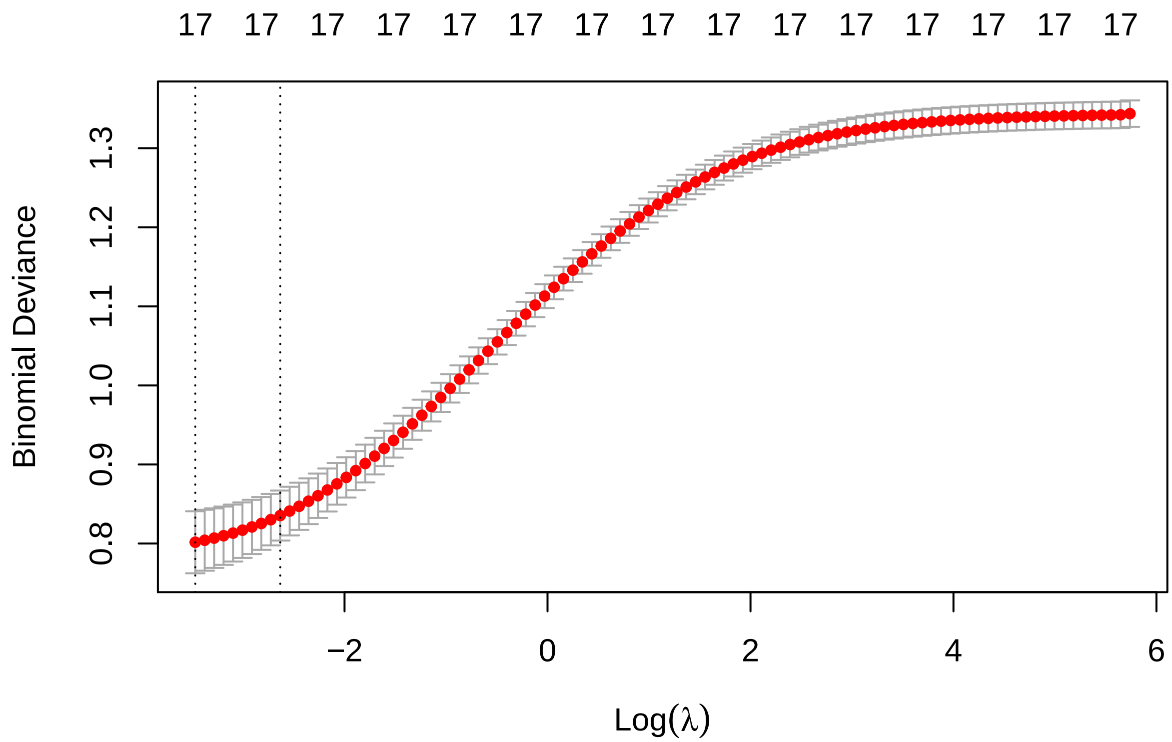
Looking at our ROC curves, it seems like the sigmoid kernal or logistic regression work best looking at AUC.

G

```
#G
###ridge regression
x=model.matrix(Purchase~.,OJ)[,-1]
y=OJ$Purchase
grid=10^seq(10,-2, length =100)
length(grid)
```

```
## [1] 100
```

```
ridge.mod=glmnet (x,y,alpha=0, lambda=grid, family = "binomial")
#finding best lambda
set.seed(1)
cv.out=cv.glmnet(x[train ,],y[ train],alpha=0, family = "binomial")
plot(cv.out)
```




```
bestlam =cv.out$lambda.min
#bestlam is 0.0311
#training data
ridge.mod=glmnet(x[train ,],y[ train],alpha=0, lambda=0.0311,thresh =1e-12, family = "binomial")
ridge.pred=predict(ridge.mod ,s=4, newx=x[train,])
ridge.pred2=rep("CH",800)
ridge.pred2[ridge.pred >0]="MM"
table(predict=ridge.pred2 , truth=OJ[train,]$Purchase )
```

```
##        truth
## predict  CH  MM
##      CH 423  76
##      MM  62 239
```

```
1-mean(ridge.pred2==OJ[train,]$Purchase)
```

```
## [1] 0.1725
```

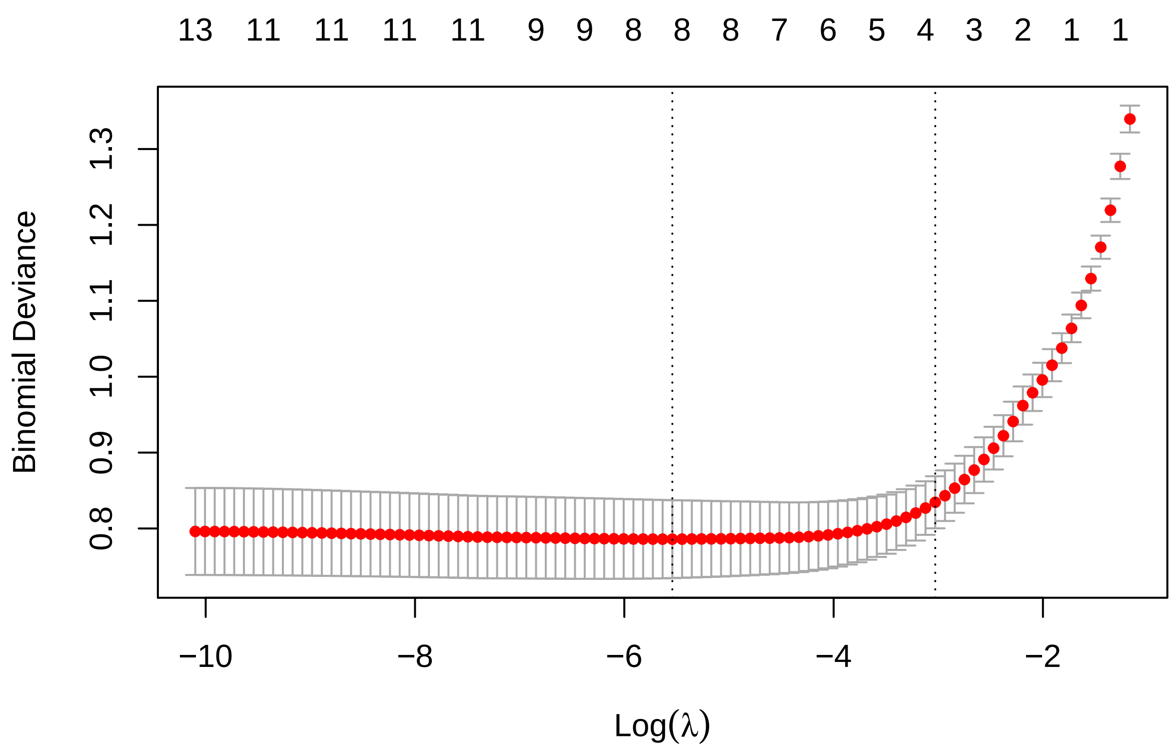
```
#17.2% err w/ RR
ridge.pred=predict(ridge.mod ,s=4, newx=x[-train,])
ridge.pred2=rep("CH",270)
ridge.pred2[ridge.pred >0]="MM"
table(predict=ridge.pred2 , truth=OJ[-train,]$Purchase )
```

```
##        truth
## predict  CH  MM
##      CH 155  34
##      MM  13  68
```

```
1-mean(ridge.pred2==OJ[-train,]$Purchase)
```

```
## [1] 0.1740741
```

```
#17.4 err w/ RR

###LASSO
#finding best lambda
set.seed(1)
cv.out=cv.glmnet(x[train ,],y[ train],alpha=1, family = "binomial")
plot(cv.out)
```




```
bestlam =cv.out$lambda.min
#bestlam is 0.00392
#training data
ridge.mod=glmnet(x[train ,],y[ train],alpha=1, lambda=0.00392,thresh =1e-12, family = "binomial")
ridge.pred=predict(ridge.mod ,s=4, newx=x[train,])
ridge.pred2=rep("CH",800)
ridge.pred2[ridge.pred >0]="MM"
table(predict=ridge.pred2 , truth=OJ[train,]$Purchase )
```

```
##        truth
## predict  CH  MM
##      CH 422  70
##      MM  63 245
```

```r
1-mean(ridge.pred2==OJ[train,]$Purchase)
```

```
## [1] 0.16625
```

```r
#16.6% err w/ RR
ridge.pred=predict(ridge.mod ,s=4, newx=x[-train,])
ridge.pred2=rep("CH",270)
ridge.pred2[ridge.pred >0]="MM"
table(predict=ridge.pred2 , truth=OJ[-train,]$Purchase )
```

```
##        truth
## predict  CH  MM
##      CH 155  33
##      MM  13  69
```

```r
1-mean(ridge.pred2==OJ[-train,]$Purchase)
```

```
## [1] 0.1703704
```

```r
#17% err w/ RR
```

Our ridge regression method has a training error rate of 17.2%, and a test error rate of 17.4%

Our LASSO method has a training error rate of 16.6%, and a test error rate of 17%.

Compared to the other methods we have looked at in this homework, we can conclude that RR and LASSO are comparable to our Support Vector Machines, Support Vector Classifiers, and our standard Logistic Regression Methods.