Team 9

Erin DiFabio, Chris Farrell, Kris Hooper, and Jared Shawver

9.6.3 ROC Curves

We begin with a portion of code from 9.6.2, which is required for section 9.6.3.

```
library(e1071)
set.seed(1)
x=matrix(rnorm(200*2), ncol=2)
x[1:100,] <- x[1:100,]+2
x[101:150,] <- x[101:150,]-2
y <- c(rep(1,150),rep(2,50))
dat=data.frame(x=x,y=as.factor(y))
train <- sample(200,100)
```

In order to implement ROC curves in R, we must first import the ROCR package. Additionally, we must write a short function to plot a ROC curve with the following arguments: the vector 'pred' containing a numerical score for each observation and the vector 'truth' containing the class label for each observation.

```
library(ROCR)
```

```
## Warning: package 'ROCR' was built under R version 3.6.3
```

```
## Loading required package: gplots
```

```
## Warning: package 'gplots' was built under R version 3.6.3
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

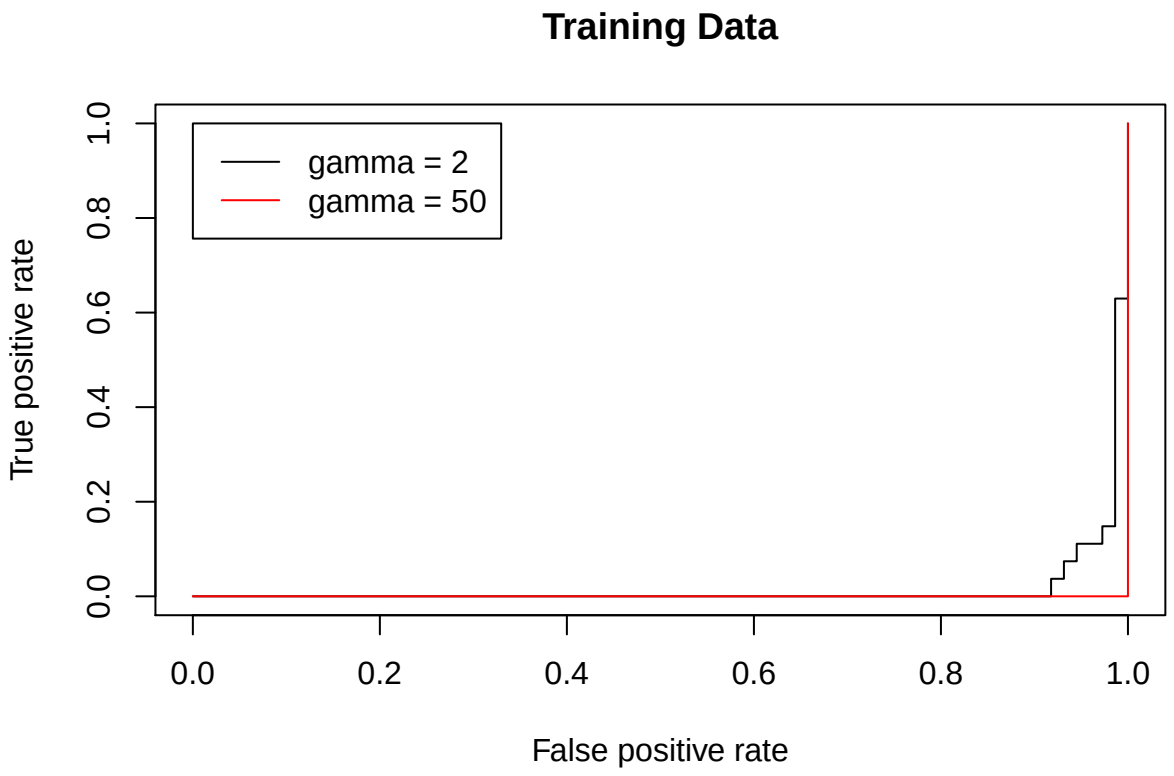
```
rocplot <- function(pred, truth, ...){
  predob <- prediction(pred, truth)
  perf <- performance(predob, "tpr", "fpr")
  plot(perf,...)}
```

By default, support vector classifiers and machines in R yield class labels. For the above function, we require fitted values, which represent the numerical scores used to determine the class label. To retrieve these fitted values, we need to use "decision.values=T" in the svm() function and use predict().

```
svmfit.opt <- svm(y~., data=dat[train,], kernel="radial",gamma=2, cost=1,decision.values=T)
fitted <- attributes(predict(svmfit.opt,dat[train,],decision.values=TRUE))$decision.values
```

The code above provides us what we need to plot the ROC curve. Below, we plot the ROC curve for two different SVMs. The first of which is the SVM we generated earlier, while the second is a support vector machine with a greater value for gamma.

```
par(mfrow=c(1,1))
rocplot(fitted,dat[train,"y"],main="Training Data")
svmfit.flex <- svm(y~., data=dat[train,], kernel="radial",gamma=50, cost=1, decision.values=T)
fitted <- attributes(predict(svmfit.flex,dat[train,],decision.values=T))$decision.values
rocplot(fitted,dat[train,"y"],add=T,col="red")
legend(0,1,legend=c("gamma = 2","gamma = 50"),col=c("black","red"),lty=1)
```



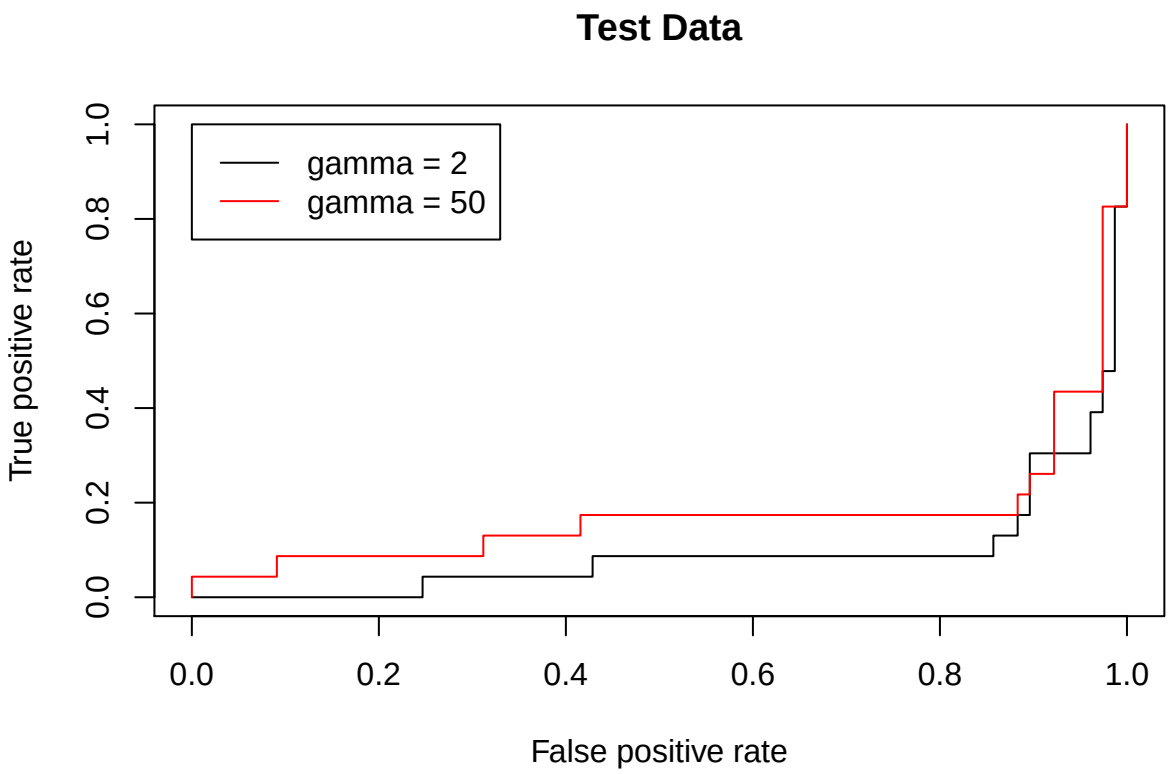
These predictions seem to be incredibly inaccurate. An ideal ROC curve hugs the upper-left of the graph. In this case, the increase in gamma made the fit more flexible, but decreased accuracy.

The above ROC curves pertain to the training data; we are more interested in the results for the test data.

```
fitted.opt <- attributes(predict(svmfit.opt,dat[-train,],decision.values=T))$decision.values
rocplot(fitted.opt,dat[-train,"y"],main="Test Data")

fitted.flex <- attributes(predict(svmfit.flex,dat[-train,],decision.values=T))$decision.values
rocplot(fitted.flex,dat[-train,"y"],add=T,col="red")

legend(0,1,legend=c("gamma = 2","gamma = 50"),col=c("black","red"),lty=1)
```



When we compute the ROC curves on the test data, the more flexible model with gamma = 50 is more accurate.

Below, we take a look at the underlying fitted values and the true classes.

```
test_dat <- dat[-train,]
test_dat[,4] <- fitted.opt
print(test_dat[1:20,])
```

```
##           x.1        x.2 y        1/2
## 2   2.1836433  3.6888733 1  1.0314905
## 3   1.1643714  3.5865884 1  0.9451598
## 4   3.5952808  1.6690922 1  0.9618664
## 5   2.3295078 -0.2852355 1 -0.3613532
## 6   1.1795316  4.4976616 1  0.6354472
## 10  1.6946116  2.5101084 1  1.3341692
## 11  3.5117812  1.8356242 1  0.9807842
## 14 -0.2146999  0.6297921 1 -1.4043977
## 15  3.1249309  2.9878383 1  1.0232205
## 16  1.9550664  3.5197450 1  1.0563091
## 17  1.9838097  1.6912594 1  1.2828115
## 18  2.9438362  0.7467102 1  0.9924712
## 21  2.9189774  0.2667816 1  0.5621606
## 23  2.0745650  1.3696997 1  1.1539827
## 24  0.0106483  1.6590314 1 -0.1255816
## 28  0.5292476  0.3944866 1 -1.0921816
```

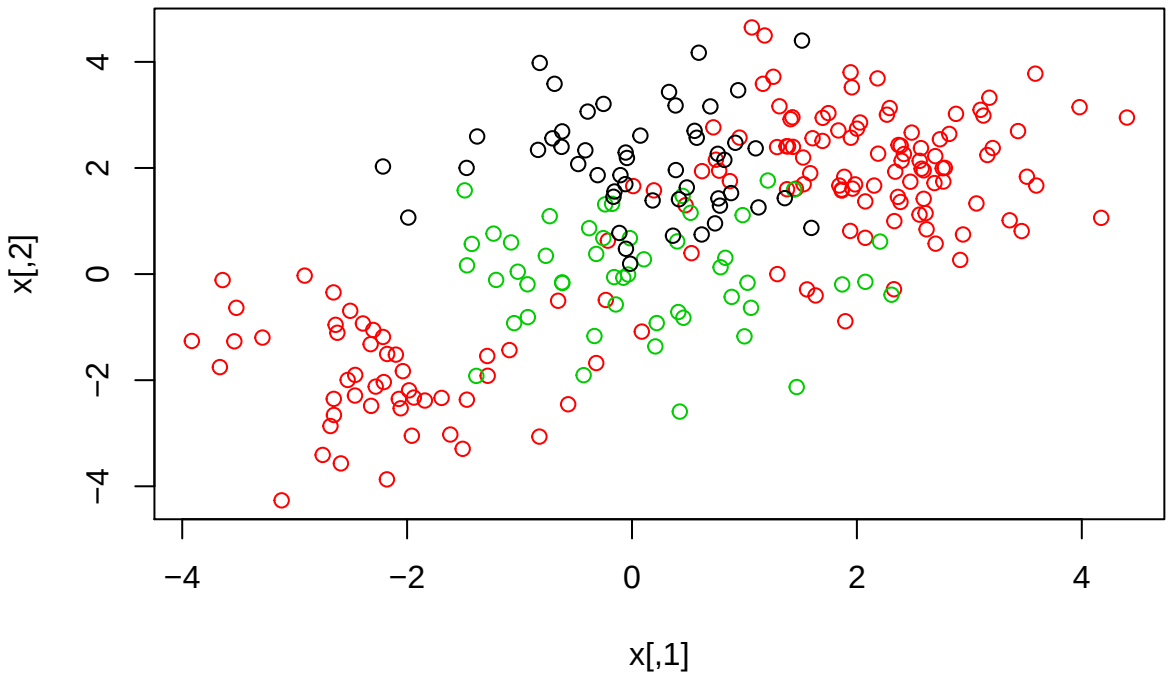
```
## 32  1.8972123 -0.8889207 1 -0.8537011
## 33  2.3876716  1.3595183 1  1.2349400
## 37  1.6057100  2.5608207 1  1.3452319
## 38  1.9406866  0.8135414 1  0.5059613
```

9.6.4 SVM with Multiple Classes

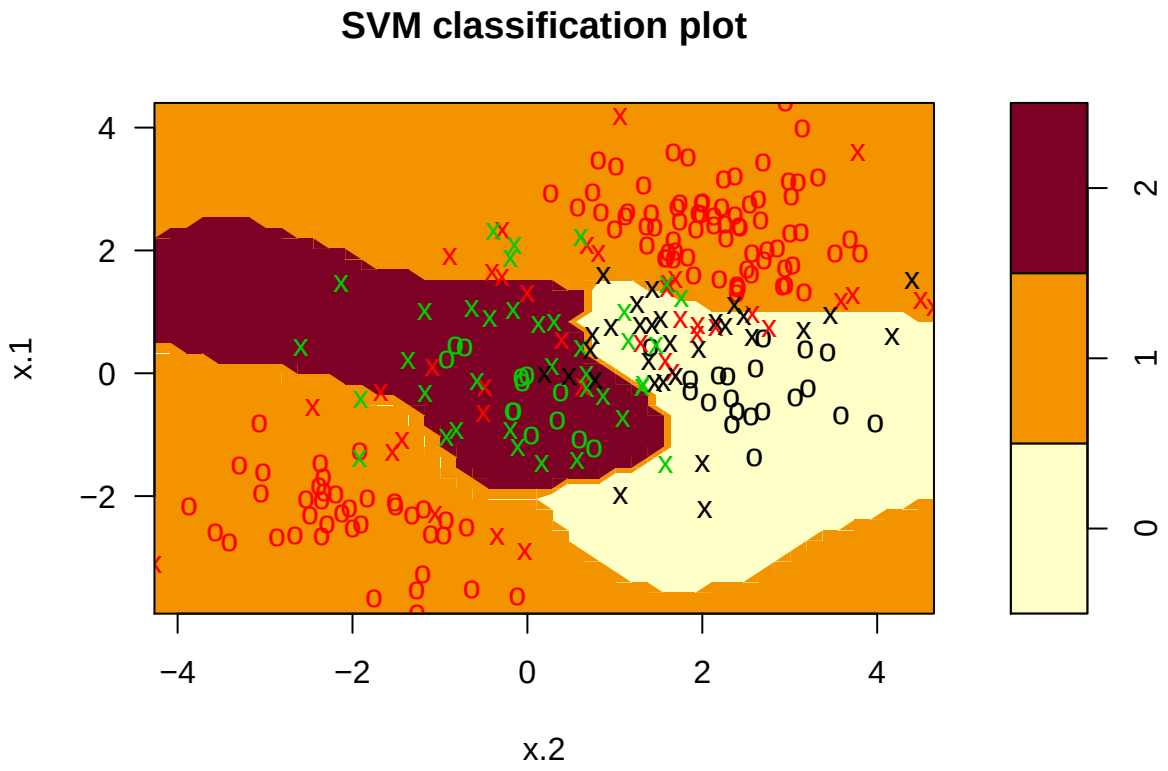
In the case that the response variable contains more than levels, then the svm() function will perform multi-class classification. Multi-class classification in the svm() function uses the one-versus-one (also known as all-pairs) approach. For K classes, K(K-1)/2 SVMs are produced for every pair of classes. The class label results from each model for a given observation are aggregated and used to determine the final test classification.

Below we generate a third class of observations and fit a support vector machine.

```
set.seed(1)
x <- rbind(x, matrix(rnorm(50*2), ncol=2))
y <- c(y, rep(0,50))
x[y==0,2] <- x[y==0,2]+2
dat <- data.frame(x=x, y=as.factor(y))
par(mfrow=c(1,1))
plot(x,col=(y+1))
```



```
svmfit <- svm(y~., data=dat, kernel="radial", cost=10, gamma=1)
plot(svmfit, dat)
```



Exercise 8

This problem uses the OJ dataset from the ISLR pacakge. A seed is set so that the results can be repeated.

```
library(ISLR)
set.seed(123)
```

Part (a)

Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```
pop <- nrow(OJ)
sam <- 800
train <- sample(pop, sam)
OJ.train <- OJ[train, ]
OJ.test <- OJ[-train, ]
```

Part (b)

Fit a support vector classifier to the training data using cost=0.01, with Purchase as the response and the other variables as predictors. Use the summary() function to produce summary statistics, and describe the results obtained.

```
svc <- svm(Purchase~., kernel = "linear", data = OJ.train, cost = 0.01)
summary(svc)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "linear", cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  442
##
##  ( 220 222 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Part (c)

Display the training and test error rates.

```
svc.train.pred <- predict(svc,OJ.train)
svc.train.table <- table(OJ.train$Purchase, svc.train.pred)
svc.train.err <- ((svc.train.table[2] + svc.train.table[3]) /
                  (svc.train.table[1] + svc.train.table[2] + svc.train.table[3] + svc.train.table[4]))
print(paste("Training error: ",svc.train.err))
```

```
## [1] "Training error:  0.165"
```

```
svc.test.pred <- predict(svc,OJ.test)
svc.test.table <- table(OJ.test$Purchase, svc.test.pred)
svc.test.err <- ((svc.test.table[2] + svc.test.table[3]) /
                        (svc.test.table[1] + svc.test.table[2] + svc.test.table[3] + svc.test.table[4]))
print(paste("Test error: ",svc.test.err))
```

```
## [1] "Test error:  0.177777777777778"
```

Part (d)

Use the tune() function to select an optimal cost. Consider values in the range 0.01 to 10.

```
tune.out.svc <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "linear",
                 ranges=list(cost=10^seq(-2,1,0.1)))
summary(tune.out.svc)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost
##  2.511886
##
## - best performance: 0.1625
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.17625 0.03143004
## 2   0.01258925 0.17375 0.03304563
## 3   0.01584893 0.17125 0.03537988
## 4   0.01995262 0.17125 0.03335936
## 5   0.02511886 0.17250 0.03322900
## 6   0.03162278 0.16875 0.03498512
## 7   0.03981072 0.17125 0.03387579
## 8   0.05011872 0.17000 0.03689324
## 9   0.06309573 0.16875 0.03397814
## 10  0.07943282 0.17000 0.03446012
## 11  0.10000000 0.17250 0.03425801
## 12  0.12589254 0.17375 0.03408018
## 13  0.15848932 0.17625 0.03356689
## 14  0.19952623 0.17500 0.03435921
## 15  0.25118864 0.17125 0.03634805
## 16  0.31622777 0.17250 0.03574602
## 17  0.39810717 0.17125 0.03438447
## 18  0.50118723 0.17125 0.03438447
## 19  0.63095734 0.16875 0.03397814
## 20  0.79432823 0.16875 0.03596391
## 21  1.00000000 0.16875 0.03596391
## 22  1.25892541 0.16750 0.03593976
## 23  1.58489319 0.16875 0.03596391
## 24  1.99526231 0.16625 0.03634805
## 25  2.51188643 0.16250 0.03118048
## 26  3.16227766 0.16500 0.02934469
## 27  3.98107171 0.16625 0.03007514
## 28  5.01187234 0.16500 0.03106892
## 29  6.30957344 0.16625 0.02949223
## 30  7.94328235 0.17125 0.02829041
## 31 10.00000000 0.17250 0.02751262
```

Part (e)

Compute the training and test error rates using this new value for cost.

```
bestcost.svc <- tune.out.svc$best.parameters$cost
print(paste("The best cost is",bestcost.svc))
```

```
## [1] "The best cost is 2.51188643150958"
```

```
svc.best <- svm(Purchase~., kernel = "linear", data = OJ.train, cost = bestcost.svc)
svc.best.train.pred <- predict(svc.best,OJ.train)
svc.best.train.table <- table(OJ.train$Purchase, svc.best.train.pred)
svc.best.train.err <- ((svc.best.train.table[2] + svc.best.train.table[3]) /
                         (svc.best.train.table[1] + svc.best.train.table[2] + svc.best.train.table[3] + s
print(paste("Training error after tuning: ",svc.best.train.err))
```

```
## [1] "Training error after tuning:  0.160602258469260"
```

```
svc.best.test.pred <- predict(svc.best,OJ.test)
svc.best.test.table <- table(OJ.test$Purchase, svc.best.test.pred)
svc.best.test.err <- ((svc.best.test.table[2] + svc.best.test.table[3]) /
                        (svc.best.test.table[1] + svc.best.test.table[2] + svc.best.test.table[3] + svc.
print(paste("Test error after tuning: ",svc.best.test.err))
```

```
## [1] "Test error after tuning:  0.155555555555556"
```

Part (f):

Repeat parts (b) through (e) using a support vector machine with a radial kernel. Use the default value for gamma.

```
#f

rad <- svm(Purchase~., kernel = "radial", data = OJ.train)

summary(svc)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "linear", cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  442
##
##  ( 220 222 )
```

```
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
rad.train.pred <- predict(rad,OJ.train)
rad.train.table <- table(OJ.train$Purchase, rad.train.pred)
rad.train.err <- ((rad.train.table[2] + rad.train.table[3]) /
                        (rad.train.table[1] + rad.train.table[2] + rad.train.table[3] + rad.train.table
print(paste("Training error: ",rad.train.err))
```

```
## [1] "Training error:  0.13875"
```

```
rad.test.pred <- predict(rad,OJ.test)
rad.test.table <- table(OJ.test$Purchase, rad.test.pred)
rad.test.err <- ((rad.test.table[2] + rad.test.table[3]) /
                       (rad.test.table[1] + rad.test.table[2] + rad.test.table[3] + rad.test.table[4]))
print(paste("Test error: ",rad.test.err))
```

```
## [1] "Test error:  0.188888888888889"
```

```
tune.out.rad <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "radial",
                 ranges=list(cost=10^seq(-2,1,0.1)))
summary(tune.out.rad)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.1625
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.39125 0.04411554
## 2   0.01258925 0.39125 0.04411554
## 3   0.01584893 0.39125 0.04411554
## 4   0.01995262 0.39125 0.04411554
## 5   0.02511886 0.39125 0.04411554
## 6   0.03162278 0.36875 0.06325928
## 7   0.03981072 0.27000 0.05470883
## 8   0.05011872 0.22500 0.04208127
## 9   0.06309573 0.19125 0.04084609
## 10  0.07943282 0.18250 0.04216370
## 11  0.10000000 0.18125 0.03691676
## 12  0.12589254 0.17500 0.03679900
```

```
## 13  0.15848932 0.17000 0.03016160
## 14  0.19952623 0.17250 0.03374743
## 15  0.25118864 0.17750 0.03525699
## 16  0.31622777 0.17250 0.03476109
## 17  0.39810717 0.16750 0.03446012
## 18  0.50118723 0.16375 0.03197764
## 19  0.63095734 0.16375 0.03304563
## 20  0.79432823 0.16375 0.03508422
## 21  1.00000000 0.16250 0.03486083
## 22  1.25892541 0.16625 0.03175973
## 23  1.58489319 0.16750 0.03291403
## 24  1.99526231 0.16375 0.02913689
## 25  2.51188643 0.16500 0.03270236
## 26  3.16227766 0.16625 0.03283481
## 27  3.98107171 0.16875 0.03346329
## 28  5.01187234 0.16625 0.03175973
## 29  6.30957344 0.17250 0.03809710
## 30  7.94328235 0.17000 0.03961621
## 31 10.00000000 0.16500 0.03622844
```

```
bestcost.rad <- tune.out.rad$best.parameters$cost
print(paste("The best cost is",bestcost.rad))
```

```
## [1] "The best cost is 1"
```

```
rad.best <- svm(Purchase~., kernel = "radial", data = OJ.train, cost = bestcost.rad)
rad.best.train.pred <- predict(rad.best,OJ.train)
rad.best.train.table <- table(OJ.train$Purchase, rad.best.train.pred)
rad.best.train.err <- ((rad.best.train.table[2] + rad.best.train.table[3]) /
                           (rad.best.train.table[1] + rad.best.train.table[2] + rad.best.train.table[3
print(paste("Training error after tunning: ",rad.best.train.err))
```

```
## [1] "Training error after tunning:  0.13875"
```

```
rad.best.test.pred <- predict(rad.best,OJ.test)
rad.best.test.table <- table(OJ.test$Purchase, rad.best.test.pred)
rad.best.test.err <- ((rad.best.test.table[2] + rad.best.test.table[3]) /
                          (rad.best.test.table[1] + rad.best.test.table[2] + rad.best.test.table[3] +
print(paste("Test error after tunning: ",rad.best.test.err))
```

```
## [1] "Test error after tunning:  0.188888888888889"
```

Part (g)

Repeat parts (b) through (e) using a support vector machine with a polynomial kernel. Set degree=2.

```
pol <- svm(Purchase~., kernel = "poly", data = OJ.train, degree=2)

summary(svc)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "linear", cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  442
##
##  ( 220 222 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
pol.train.pred <- predict(pol,OJ.train)
pol.train.table <- table(OJ.train$Purchase, pol.train.pred)
pol.train.err <- ((pol.train.table[2] + pol.train.table[3]) /
                    (pol.train.table[1] + pol.train.table[2] + pol.train.table[3] + pol.train.table[4]))
print(paste("Training error: ",pol.train.err))
```

```
## [1] "Training error:  0.1725"
```

```
pol.test.pred <- predict(pol,OJ.test)
pol.test.table <- table(OJ.test$Purchase, pol.test.pred)
pol.test.err <- ((pol.test.table[2] + pol.test.table[3]) /
                   (pol.test.table[1] + pol.test.table[2] + pol.test.table[3] + pol.test.table[4]))
print(paste("Test error: ",pol.test.err))
```

```
## [1] "Test error:  0.222222222222222"
```

```
tune.out.pol <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "poly",degree=2,
                     ranges=list(cost=10^seq(-2,1,0.1)))
summary(tune.out.pol)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost
##  6.309573
##
## - best performance: 0.165
##
## - Detailed performance results:
```

```
##           cost   error dispersion
## 1   0.01000000 0.38750 0.04289846
## 2   0.01258925 0.37000 0.03593976
## 3   0.01584893 0.36875 0.03448530
## 4   0.01995262 0.36500 0.02813657
## 5   0.02511886 0.36125 0.02853482
## 6   0.03162278 0.36000 0.02687419
## 7   0.03981072 0.35250 0.03162278
## 8   0.05011872 0.34250 0.03343734
## 9   0.06309573 0.32250 0.02266912
## 10  0.07943282 0.31500 0.01645701
## 11  0.10000000 0.30875 0.01772671
## 12  0.12589254 0.29000 0.01645701
## 13  0.15848932 0.25250 0.02622022
## 14  0.19952623 0.22125 0.02285978
## 15  0.25118864 0.21125 0.03197764
## 16  0.31622777 0.20500 0.03736085
## 17  0.39810717 0.20000 0.03864008
## 18  0.50118723 0.19375 0.03830162
## 19  0.63095734 0.19250 0.03827895
## 20  0.79432823 0.18750 0.03435921
## 21  1.00000000 0.18375 0.03335936
## 22  1.25892541 0.18250 0.03689324
## 23  1.58489319 0.17625 0.03557562
## 24  1.99526231 0.17625 0.04059026
## 25  2.51188643 0.17500 0.03535534
## 26  3.16227766 0.17250 0.03944053
## 27  3.98107171 0.17250 0.03899786
## 28  5.01187234 0.17000 0.04090979
## 29  6.30957344 0.16500 0.03809710
## 30  7.94328235 0.16875 0.03738408
## 31 10.00000000 0.17125 0.03729108
```

```
bestcost.pol <- tune.out.pol$best.parameters$cost
print(paste("The best cost is",bestcost.pol))
```

```
## [1] "The best cost is 6.30957344480194"
```

```
pol.best <- svm(Purchase~., kernel = "poly",degree=2, data = OJ.train, cost = bestcost.pol)
pol.best.train.pred <- predict(pol.best,OJ.train)
pol.best.train.table <- table(OJ.train$Purchase, pol.best.train.pred)
pol.best.train.err <- ((pol.best.train.table[2] + pol.best.train.table[3]) /
                         (pol.best.train.table[1] + pol.best.train.table[2] + pol.best.train.table[3] +
print(paste("Training error after tuning: ",pol.best.train.err))
```

```
## [1] "Training error after tuning:  0.147096774193548"
```

```
pol.best.test.pred <- predict(pol.best,OJ.test)
pol.best.test.table <- table(OJ.test$Purchase, pol.best.test.pred)
pol.best.test.err <- ((pol.best.test.table[2] + pol.best.test.table[3]) /
                        (pol.best.test.table[1] + pol.best.test.table[2] + pol.best.test.table[3] + pol
print(paste("Test error after tuning: ",pol.best.test.err))
```

```
## [1] "Test error after tuning:  0.196296296296296"
```

Part (h)

Determine the best approach.

```
print(paste("Best SVC Test error: ",svc.best.test.err))
```

```
## [1] "Best SVC Test error:  0.155555555555556"
```

```
print(paste("Best Rad Test error: ",rad.best.test.err))
```

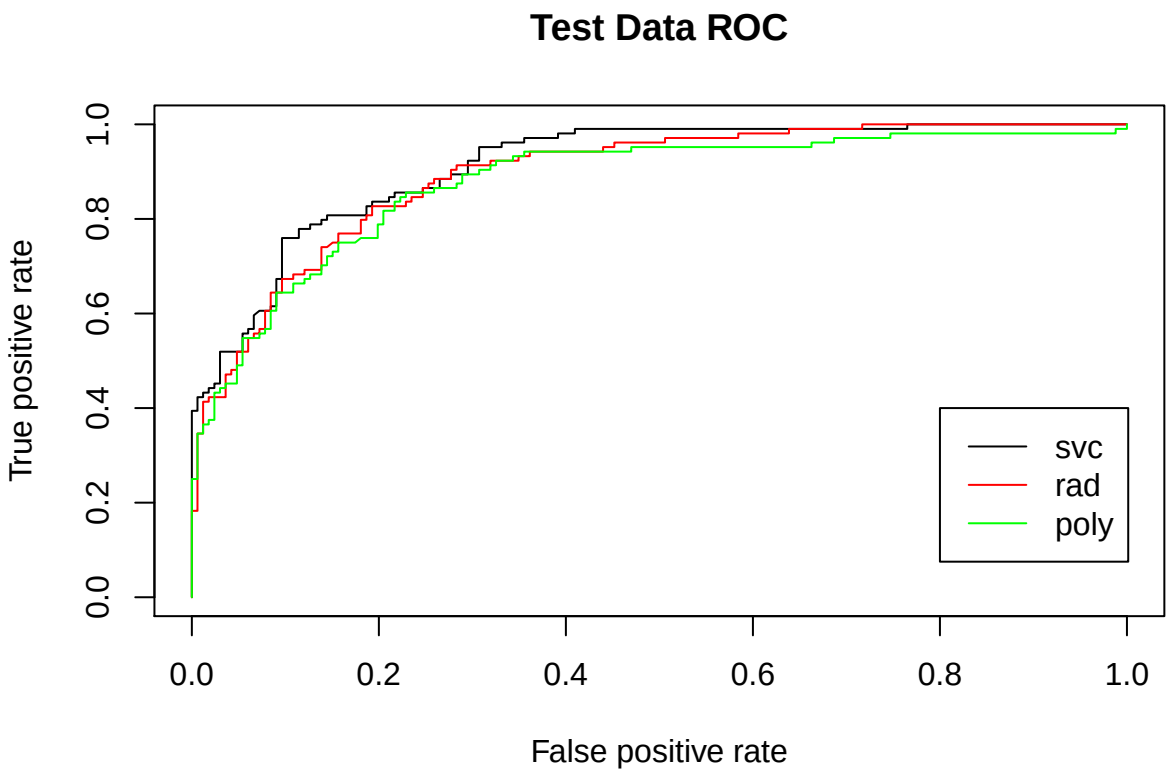
```
## [1] "Best Rad Test error:  0.188888888888889"
```

```
print(paste("Best Polynomial Test error: ",pol.best.test.err))
```

```
## [1] "Best Polynomial Test error:  0.196296296296296"
```

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

fitted.svc <- attributes(predict(svc.best,OJ.test,decision.values=TRUE))$decision.values
rocplot(fitted.svc,OJ.test$Purchase,main="Test Data ROC")
fitted.rad <- attributes(predict(rad.best,OJ.test,decision.values=TRUE))$decision.values
rocplot(fitted.rad,OJ.test$Purchase,add=T,col='red')
fitted.pol <- attributes(predict(pol.best,OJ.test,decision.values=TRUE))$decision.values
rocplot(fitted.pol,OJ.test$Purchase,add=T,col="green")
legend(.8,.4,legend=c("svc","rad","poly"),col=c("black","red","green"),lty=1)
```




```
print("According to test error rate and the ROC plot, the linear SVC is the best model.")
```

```
## [1] "According to test error rate and the ROC plot, the linear SVC is the best model."
```