Sam Lin
Hwk. 4
Stats 202
Chapter 8, Exercise 10 (p. 334)

a)

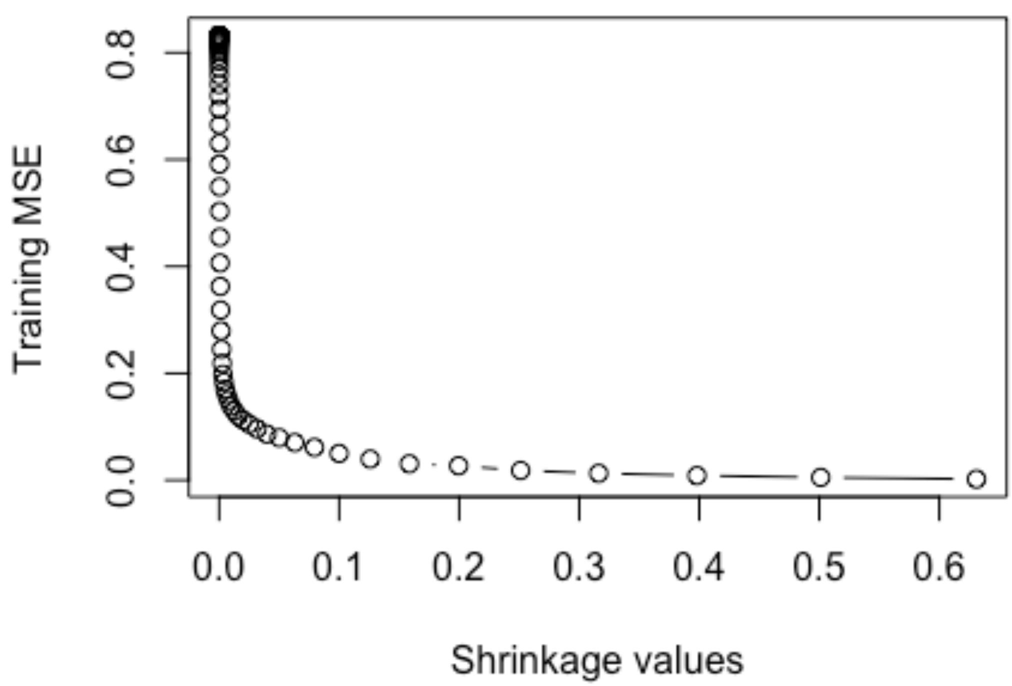
```
> Hitters <- na.omit(Hitters)
> Hitters$Salary <- log(Hitters$Salary)
```

b)

```
> train <- 1:200
> Hitters.train <- Hitters[train, ]
> Hitters.test <- Hitters[-train, ]
```

c)

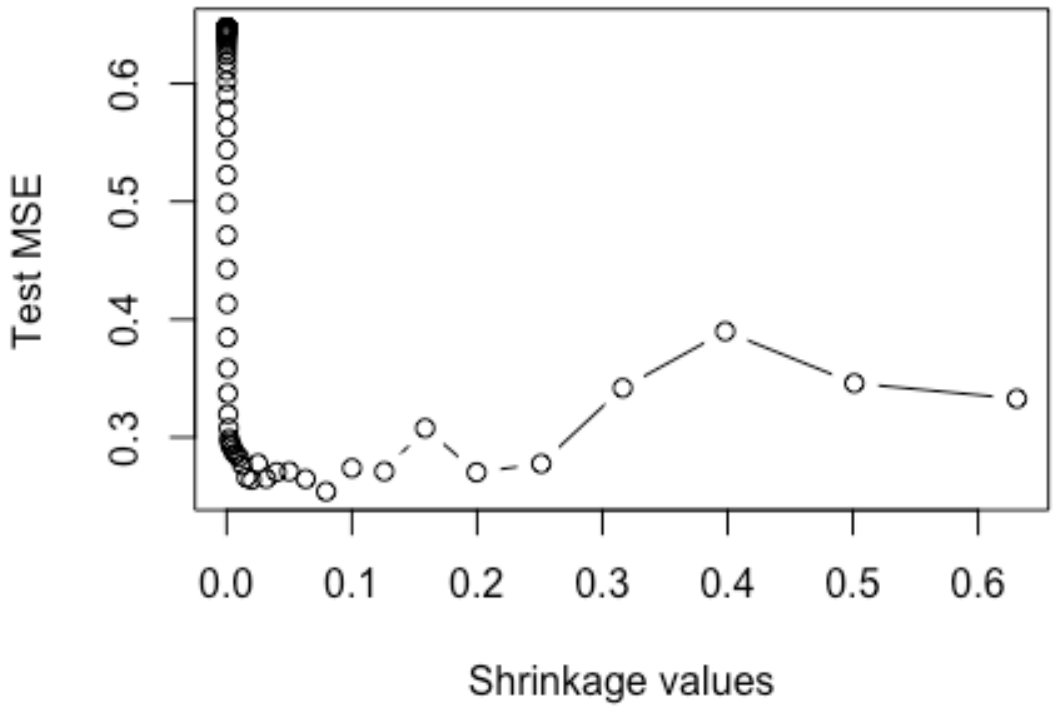
```
> pows <- seq(-10, -0.2, by = 0.1)
> lambdas <- 10^pows
> train.err <- rep(NA, length(lambdas))
> for (i in 1:length(lambdas)) {
+   boost.hitters <- gbm(Salary ~ ., data = Hitters.train, distribution = "gaussian", n.trees = 1000, shrin
kage = lambdas[i])
+   pred.train <- predict(boost.hitters, Hitters.train, n.trees = 1000)
+   train.err[i] <- mean((pred.train - Hitters.train$Salary)^2)
+ }
> plot(lambdas, train.err, type = "b", xlab = "Shrinkage values", ylab = "Training MSE")
```



d)

```
> for (i in 1:length(lambdas)) {
+   boost.hitters <- gbm(Salary ~ ., data = Hitters.train, distribution = "gaussian", n.trees = 1000, shrin
kage = lambdas[i])
+   yhat <- predict(boost.hitters, Hitters.test, n.trees = 1000)
+   test.err[i] <- mean((yhat - Hitters.test$Salary)^2)
+ }
> plot(lambdas, test.err, type = "b", xlab = "Shrinkage values", ylab = "Test MSE")
```

Sam Lin
Hwk. 4
Stats 202




```
> min(test.err)
[1] 0.2540265
> lambdas[which.min(test.err)]
[1] 0.07943282
```

Test MSE for boosting is 0.25 and lambda = 0.08

e)

```
> fit1 <- lm(Salary ~ ., data = Hitters.train)
> pred1 <- predict(fit1, Hitters.test)
> mean((pred1 - Hitters.test$Salary)^2)
[1] 0.4917959
> x <- model.matrix(Salary ~ ., data = Hitters.train)
> x.test <- model.matrix(Salary ~ ., data = Hitters.test)
> y <- Hitters.train$Salary
> fit2 <- glmnet(x, y, alpha = 0)
> pred2 <- predict(fit2, s = 0.01, newx = x.test)
> mean((pred2 - Hitters.test$Salary)^2)
[1] 0.4570283
```

Test MSE for boosting (0.25) is lower than test MSE for linear and ridge regression (0.49, 0.45 respectively)

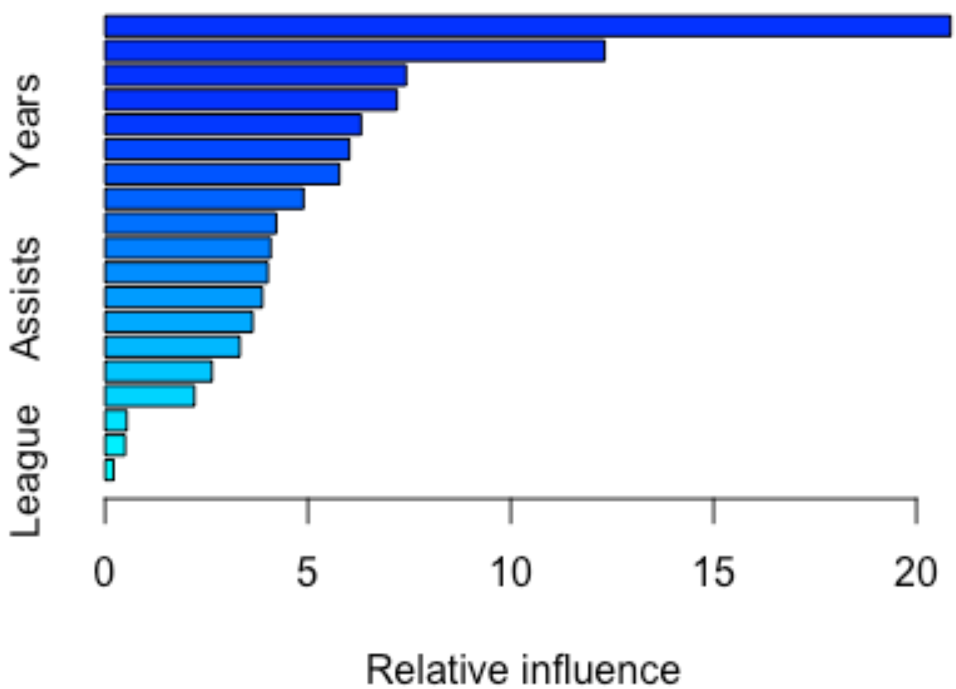
f)

Sam Lin
Hwk. 4
Stats 202

```
> boost.hitters <- gbm(Salary ~ ., data = Hitters.train, distribution = "gaussian", n.trees = 1000, shrinka
ge = lambdas[which.min(test.err)])
> summary(boost.hitters)
                var    rel.inf
CAtBat       CAtBat 20.8404970
CRBI           CRBI 12.3158959
Walks         Walks  7.4186037
PutOuts     PutOuts  7.1958539
Years         Years  6.3104535
CWalks       CWalks  6.0221656
CHmRun       CHmRun  5.7759763
CHits         CHits  4.8914360
AtBat         AtBat  4.2187460
RBI             RBI  4.0812410
Hits           Hits  4.0117255
Assists     Assists  3.8786634
HmRun         HmRun  3.6386178
CRuns         CRuns  3.3230296
Errors       Errors  2.6369128
Runs           Runs  2.2048386
Division   Division  0.5347342
NewLeague NewLeague  0.4943540
League       League  0.2062551
```

Catbat is the most important by far (rel. inf is highest)

```
> bag.hitters <- randomForest(Salary ~ ., data = Hitters.train, mtry = 19, ntree = 500)
> yhat.bag <- predict(bag.hitters, newdata = Hitters.test)
> mean((yhat.bag - Hitters.test$Salary)^2)
[1] 0.2313593
```



g)

```
> bag.hitters <- randomForest(Salary ~ ., data = Hitters.train, mtry = 19, ntree = 500)
> yhat.bag <- predict(bag.hitters, newdata = Hitters.test)
> mean((yhat.bag - Hitters.test$Salary)^2)
[1] 0.2313593
```

Test MSE is 0.23, which is lower than test MSE for boosting (0.25)

Sam Lin
Hwk. 4
Stats 202
Chapter 8, Exercise 11 (p. 335)

a)

```
> train <- 1:1000
> Caravan$Purchase <- ifelse(Caravan$Purchase == "Yes", 1, 0)
> Caravan.train <- Caravan[train, ]
> Caravan.test <- Caravan[-train, ]
```

.b)

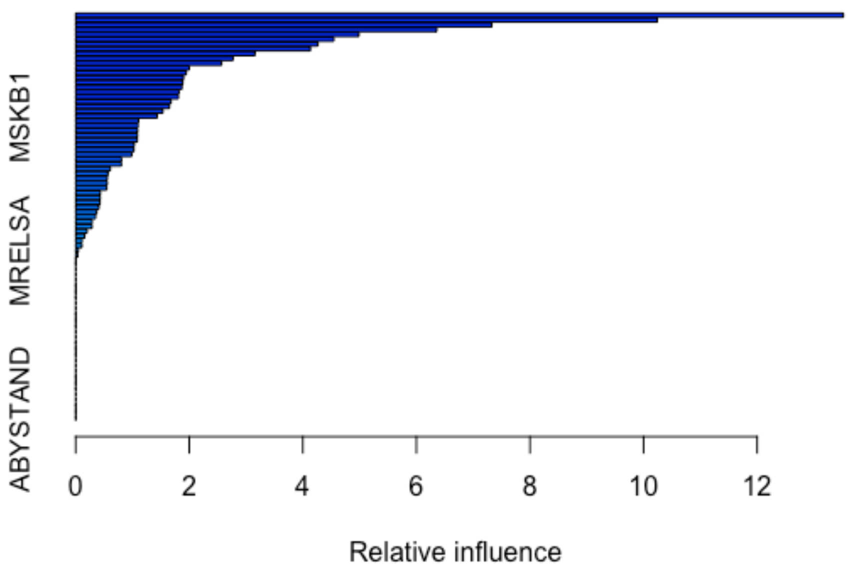
```
> boost.caravan <- gbm(Purchase ~ ., data = Caravan.train, distribution = "gaussian", n.trees = 1000, shrin
kage = 0.01)
```

c)

Sam Lin
Hwk. 4
Stats 202

```
> summary(boost.caravan)
              var     rel.inf
PPERSAUT PPERSAUT 13.51824557
MKOOPKLA MKOOPKLA 10.24062778
MOPLHOOG MOPLHOOG  7.32689780
MBERMIDD MBERMIDD  6.35820558
PBRAND     PBRAND  4.98826360
ABRAND     ABRAND  4.54504653
MGODGE     MGODGE  4.26496875
MINK3045 MINK3045  4.13253907
PWAPART   PWAPART  3.15612877
MAUT1       MAUT1  2.76929763
MOSTYPE   MOSTYPE  2.56937935
MAUT2       MAUT2  1.99879666
MSKA         MSKA  1.94618539
MBERARBG MBERARBG  1.89917331
PBYSTAND PBYSTAND  1.88591514
MINKGEM   MINKGEM  1.87131472
MGODOV     MGODOV  1.81673309
MGODPR     MGODPR  1.80814745
MFWEKIND MFWEKIND  1.67884570
MSKC         MSKC  1.65075962
MBERHOOG MBERHOOG  1.53559951
MSKB1       MSKB1  1.43339514
MOPLMIDD MOPLMIDD  1.10617074
MHHUUR     MHHUUR  1.09608784
MRELGE     MRELGE  1.09039794
MINK7512 MINK7512  1.08772012
MZFONDS   MZFONDS  1.08427551
MGODRK     MGODRK  1.03126657
MINK4575 MINK4575  1.02492795
MZPART     MZPART  0.98536712
MRELOV     MRELOV  0.80356854
MFGEKIND MFGEKIND  0.80335689
MBERARBO MBERARBO  0.60909852
APERSAUT APERSAUT  0.56707821
MGEMOMV   MGEMOMV  0.55589456
MOSHOOFD MOSHOOFD  0.55498375
MAUT0       MAUT0  0.54748481
PMOTSCO   PMOTSCO  0.43362597
MSKB2       MSKB2  0.43075446
MSKD         MSKD  0.42751490
MINK123M MINK123M  0.40920707
MINKM30   MINKM30  0.36996576
```

Sam Lin
Hwk. 4
Stats 202
The top 3 most important variables are PPERSAUT, MKOOPKLA, and MOPLHOOG (due to rel. inf)



c)

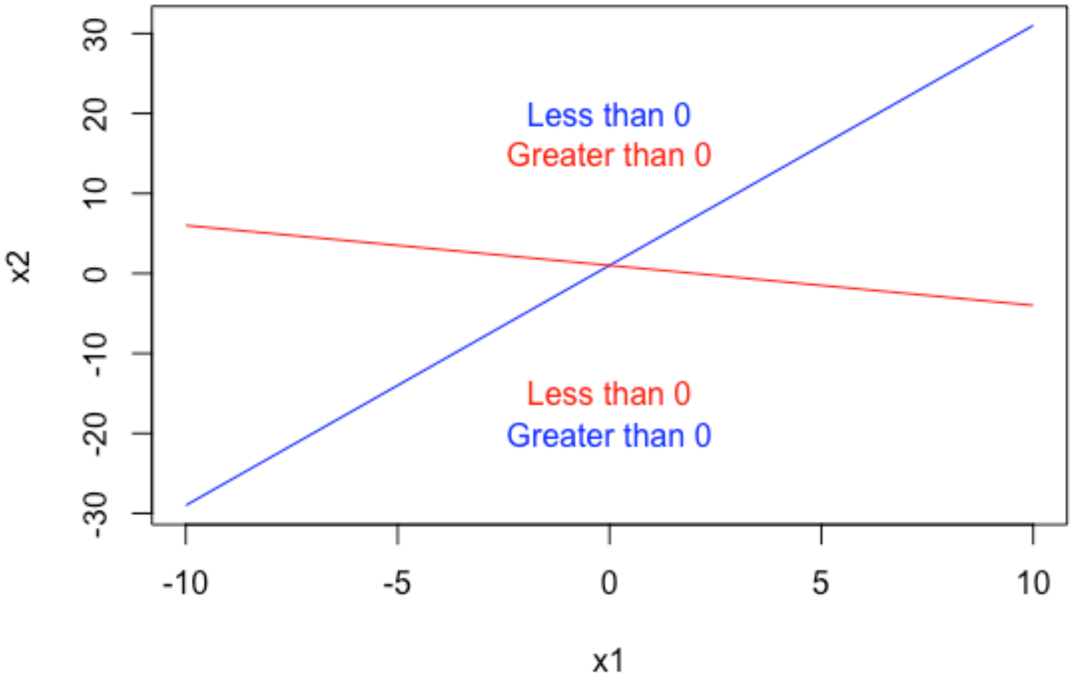
```
> probs.test <- predict(boost.caravan, Caravan.test, n.trees = 1000, type = "response")
> pred.test <- ifelse(probs.test > 0.2, 1, 0)
> table(Caravan.test$Purchase, pred.test)
   pred.test
       0    1
  0 4493   40
  1  278   11
```

For boosting, the percentage of people buying is 11/(11+40) = 0.2156

```
> logit.caravan <- glm(Purchase ~ ., data = Caravan.train, family = "binomial")
Warning message:
glm.fit: fitted probabilities numerically 0 or 1 occurred
> probs.test2 <- predict(logit.caravan, Caravan.test, type = "response")
Warning message:
In predict.lm(object, newdata, se.fit, scale = 1, type = ifelse(type ==  :
  prediction from a rank-deficient fit may be misleading
> pred.test2 <- ifelse(probs.test > 0.2, 1, 0)
> table(Caravan.test$Purchase, pred.test2)
   pred.test2
       0    1
  0 4493   40
  1  278   11
```

For logistic regression, the percentage of people buying is 11/(11+40) = 0.2156
Chapter 9, Exercise 1 (p. 368)
a)

Sam Lin
Hwk. 4
Stats 202



Chapter 9, Exercise 8 (p. 371)
a)

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

Sam Lin
Hwk. 4
Stats 202
b)

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

Call:
svm(formula = Purchase ~ ., data = OJ.train, kernel = "linear", cost = 0.01)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01
      gamma:  0.05555556

Number of Support Vectors:  432

 ( 215 217 )


Number of Classes:  2

Levels:
 CH MM
```

432 support vectors out of 800 observations/training points. 215 support vectors belong to CH and 217 belong to MM.

c)

```
> train.pred <- predict(svm.linear, OJ.train)
> table(OJ.train$Purchase, train.pred)
    train.pred
      CH  MM
  CH 439  55
  MM  78 228
```

Training error rate is (78+55)/(439+78+55+228) = 133/800 = 0.16625

```
> test.pred <- predict(svm.linear, OJ.test)
> table(OJ.test$Purchase, test.pred)
    test.pred
      CH  MM
  CH 141  18
  MM  31  80
```

Test error rate is (31+18)/(141+18+31+80) = 49/270 = 0.18148

d) Best cost is 0.1

Sam Lin
Hwk. 4
Stats 202

```
> tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "linear", ranges = list(cost = 10^seq(-2, 1, by = 0.25)))
> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.1

- best performance: 0.1625

- Detailed performance results:
          cost   error dispersion
1   0.01000000 0.17125 0.05172376
2   0.01778279 0.16500 0.05197489
3   0.03162278 0.16625 0.04604120
4   0.05623413 0.16500 0.04594683
5   0.10000000 0.16250 0.04787136
6   0.17782794 0.16250 0.04249183
7   0.31622777 0.16875 0.04379958
8   0.56234133 0.16625 0.03998698
9   1.00000000 0.16500 0.03670453
10  1.77827941 0.16625 0.03682259
11  3.16227766 0.16500 0.03717451
12  5.62341325 0.16500 0.03525699
13 10.00000000 0.16750 0.03917553
```

e)

```
> test.pred <- predict(svm.linear, OJ.test)
> table(OJ.test$Purchase, test.pred)
    test.pred
      CH  MM
  CH 140  19
  MM  32  79
```

With best cost, the test error rate is (32+19)/(32+19+79+140) = 51/270 = 0.188889

```
> svm.linear <- svm(Purchase ~ ., kernel = "linear", data = OJ.train, cost = tune.out$best.parameter$cost)
> train.pred <- predict(svm.linear, OJ.train)
> table(OJ.train$Purchase, train.pred)
    train.pred
      CH  MM
  CH 438  56
  MM  71 235
```

With best cost, training error rate is (71+56)/(71+56+235+438) = 127/800 = 0.15875

f)

Sam Lin
Hwk. 4
Stats 202

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

Call:
svm(formula = Purchase ~ ., data = OJ.train, kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.05555556

Number of Support Vectors:  379

 ( 188 191 )


Number of Classes:  2

Levels:
 CH MM
> train.pred <- predict(svm.radial, OJ.train)
> table(OJ.train$Purchase, train.pred)
    train.pred
      CH  MM
  CH 455  39
  MM  77 229
> test.pred <- predict(svm.radial, OJ.test)
> table(OJ.test$Purchase, test.pred)
    test.pred
      CH  MM
  CH 141  18
  MM  28  83
```

Created 379 support vectors out of the total observations with 188 belonging to CH and 191 belonging to MM. The classifier has training error of (77+39)/(455+39+77+229) = 116/800 = 0.171 and test error = (18+28)/(141+18+28+83) = (46/270) = 0.1704 which is better than linear with 15% and 18% training and testing error respectively.

Sam Lin
Hwk. 4
Stats 202

```
> set.seed(2)
> tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "radial", ranges = list(cost = 10^seq(-2,
+                                                                                                      1, by = 0.25)))
> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
    1

- best performance: 0.16625

- Detailed performance results:
          cost   error dispersion
1   0.01000000 0.38250 0.04533824
2   0.01778279 0.38250 0.04533824
3   0.03162278 0.37500 0.04894725
4   0.05623413 0.21500 0.05886661
5   0.10000000 0.17875 0.04860913
6   0.17782794 0.17875 0.05497790
7   0.31622777 0.17875 0.05981743
8   0.56234133 0.17250 0.05458174
9   1.00000000 0.16625 0.05001736
10  1.77827941 0.16875 0.05008673
11  3.16227766 0.17500 0.04787136
12  5.62341325 0.18000 0.05244044
13 10.00000000 0.18250 0.05596378
```

```
> svm.radial <- svm(Purchase ~ ., kernel = "radial", data = OJ.train, cost = tune.out$best.parameter$cost)
> summary(svm.radial)

Call:
svm(formula = Purchase ~ ., data = OJ.train, kernel = "radial", cost = tune.out$best.parameter$cost)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.05555556

Number of Support Vectors:  379

 ( 188 191 )


Number of Classes:  2

Levels:
 CH MM
```

```
> train.pred <- predict(svm.radial, OJ.train)
> table(OJ.train$Purchase, train.pred)
    train.pred
      CH  MM
  CH 455  39
  MM  77 229
```

Sam Lin
Hwk. 4
Stats 202

```
> test.pred <- predict(svm.radial, OJ.test)
> table(OJ.test$Purchase, test.pred)
    test.pred
      CH  MM
  CH 141  18
  MM  28  83
```

Train error: (77+39)/(455+39+77+229) = 116/800
Test error :(28+18)/(141+18+28+83) = 46/270
Both are same as above, so we can conclude tuning does not reduce the error rates.

g)

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

Call:
svm(formula = Purchase ~ ., data = OJ.train, kernel = "polynomial", degree = 2)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  1
     degree:  2
      gamma:  0.05555556
     coef.0:  0

Number of Support Vectors:  454

 ( 224 230 )


Number of Classes:  2

Levels:
 CH MM
```

Sam Lin
Hwk. 4
Stats 202

```
> test.pred <- predict(svm.poly, OJ.test)
> table(OJ.test$Purchase, test.pred)
    test.pred
      CH  MM
  CH 149  10
  MM  41  70
```

```
> train.pred <- predict(svm.poly, OJ.train)
> table(OJ.train$Purchase, train.pred)
    train.pred
      CH  MM
  CH 461  33
  MM 105 201
```

Polynomial kernel with default gamma creates 454 support vectors with 224 belonging to CH and 230 belonging to MM. The classifier has a test error of (41+10)/(149+10+70+41) = 51/270 = 0.1725 and training error of (105+33)/(461+33+105+201) = 138/800 = 0.1889. This is the same as the linear kernel.

```
> set.seed(2)
> tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "polynomial", degree = 2, ranges = list(cost = 10^seq(-2,
+                                                                                                            1, by = 0.25)))
> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
   10

- best performance: 0.18125

- Detailed performance results:
          cost   error dispersion
1   0.01000000 0.38250 0.04533824
2   0.01778279 0.36750 0.04972145
3   0.03162278 0.36500 0.05458174
4   0.05623413 0.33375 0.05070681
5   0.10000000 0.32500 0.04677072
6   0.17782794 0.25875 0.05952649
7   0.31622777 0.21250 0.06123724
8   0.56234133 0.21250 0.05743354
9   1.00000000 0.19750 0.06687468
10  1.77827941 0.19375 0.05376453
11  3.16227766 0.19625 0.05653477
12  5.62341325 0.18375 0.05434266
13 10.00000000 0.18125 0.05245699
```

Sam Lin
Hwk. 4
Stats 202

```
> svm.poly <- svm(Purchase ~ ., kernel = "polynomial", degree = 2, data = OJ.train, cost = tune.out$best.parameter$cost)
> summary(svm.poly)

Call:
svm(formula = Purchase ~ ., data = OJ.train, kernel = "polynomial", degree = 2, cost = tune.out$best.parameter$cost)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  10
     degree:  2
      gamma:  0.05555556
     coef.0:  0

Number of Support Vectors:  342

 ( 170 172 )


Number of Classes:  2

Levels:
 CH MM
```

```
> train.pred <- predict(svm.poly, OJ.train)
> table(OJ.train$Purchase, train.pred)
    train.pred
      CH  MM
  CH 450  44
  MM  72 234
> test.pred <- predict(svm.poly, OJ.test)
> table(OJ.test$Purchase, test.pred)
    test.pred
      CH  MM
  CH 140  19
  MM  31  80
```

Training error rate: (72+44)/(450+44+72+234) = 96/800 = 0.145
Test error rate: (31+19)/(140+80+31+19) = 50/270 = 0.1851

Conclusion: tuning reduces training and training error compared to 0.1725 and 0.18889 training and test error.

h)
The radial basis kernel produces the minimum classification error on test and training errors compared to others.