Stats202 Homework 4

By Fang Lin (Stanford ID # 06166564)

July 30, 2016

Problem 1 (p.334, ex10)

- (a)

  Answer:

```
> library(ISLR)
> sum(is.na(Hitters$Salary))
[1] 59
> Hitters = Hitters[-which(is.na(Hitters$Salary)), ]
> sum(is.na(Hitters$Salary))
[1] 0
> Hitters$Salary = log(Hitters$Salary)
```

- (b)

  Answer:

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

- (c)

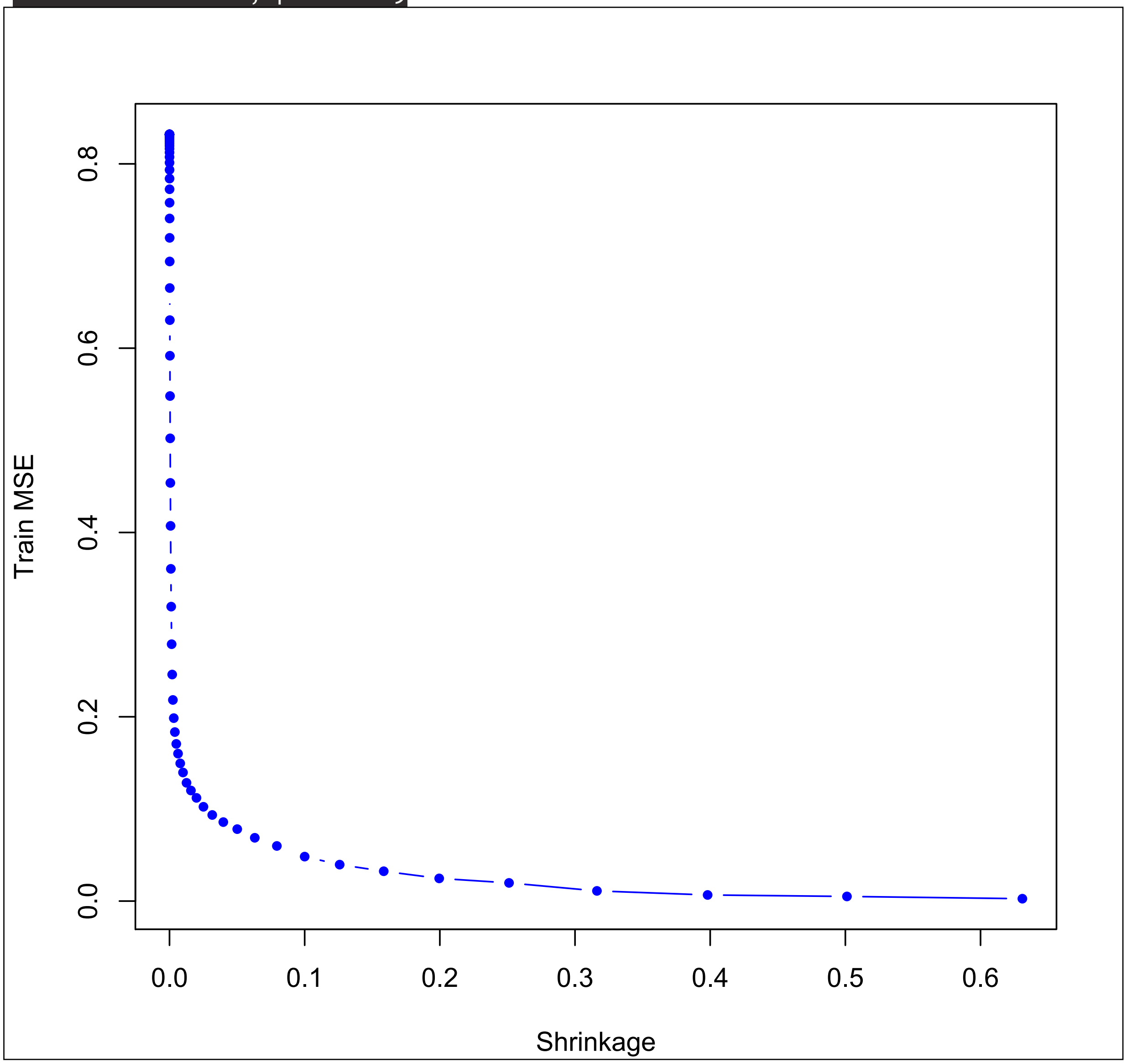
  Answer: >

  library(gbm)

```
Loading required package: survival
Loading required package: lattice
Loading required package: splines
Loading required package: parallel
Loaded gbm 2.1.1
> set.seed(103)
> pows = seq(-10, -0.2, by = 0.1)
> lambdas = 10^pows
> length.lambdas = length(lambdas)
> train.errors = rep(NA, length.lambdas)
> test.errors = rep(NA, length.lambdas)
> for (i in 1:length.lambdas) {
+     boost.hitters = gbm(Salary ~ ., data = Hitters.train, distribution =
"gaussian",
+         n.trees = 1000, shrinkage = lambdas[i])
+     train.pred = predict(boost.hitters, Hitters.train, n.trees = 1000)
+     test.pred = predict(boost.hitters, Hitters.test, n.trees = 1000)
+     train.errors[i] = mean((Hitters.train$Salary - train.pred)^2)
+     test.errors[i] = mean((Hitters.test$Salary - test.pred)^2)
+ }
```

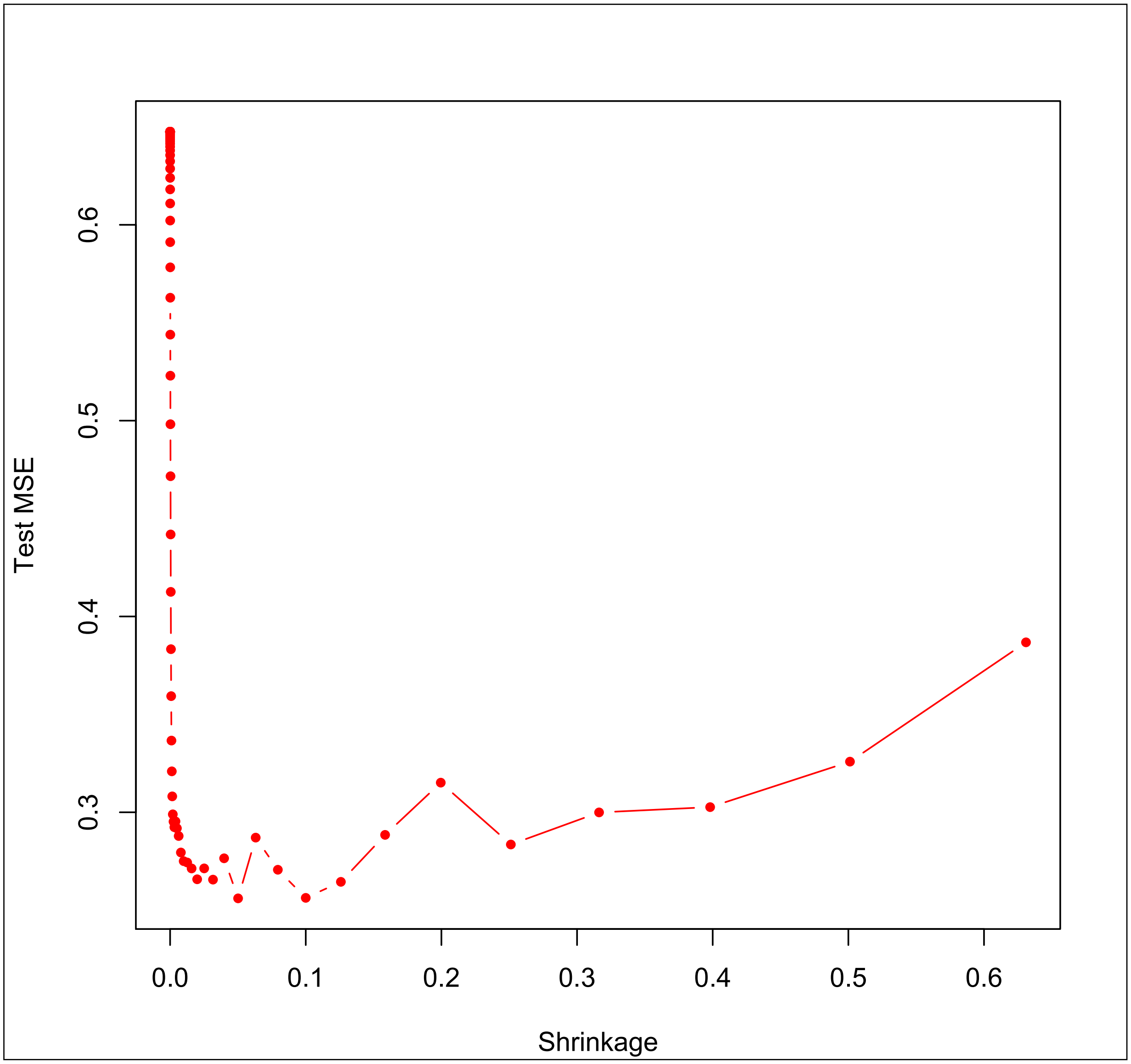
```
plot(lambdas, train.errors, type = "b", xlab = "Shrinkage", ylab = "Train
MSE",
    col = "blue", pch = 20)
```

```
>
> plot(lambdas, train.errors, type = "b", xlab = "Shrinkage", ylab = "Train
MSE",
+      col = "blue", pch = 20)
```



- (d)

  Answer:




```
> min(test.errors)
[1] 0.2560507
> lambdas[which.min(test.errors)]
[1] 0.05011872
```

When numbda=0.05011872, the minium test error got 0.2561

- (e)

Answer:

```
> lm.fit = lm(Salary ~ ., data = Hitters.train)
> lm.pred = predict(lm.fit, Hitters.test)
> mean((Hitters.test$Salary - lm.pred)^2)
[1] 0.4917959
> library(glmnet)
Loading required package: Matrix
Loading required package: foreach
```

```
foreach: simple, scalable parallel programming from Revolution Analytics
Use Revolution R for scalability, fault tolerance and more.
http://www.revolutionanalytics.com
Loaded glmnet 2.0-5
```

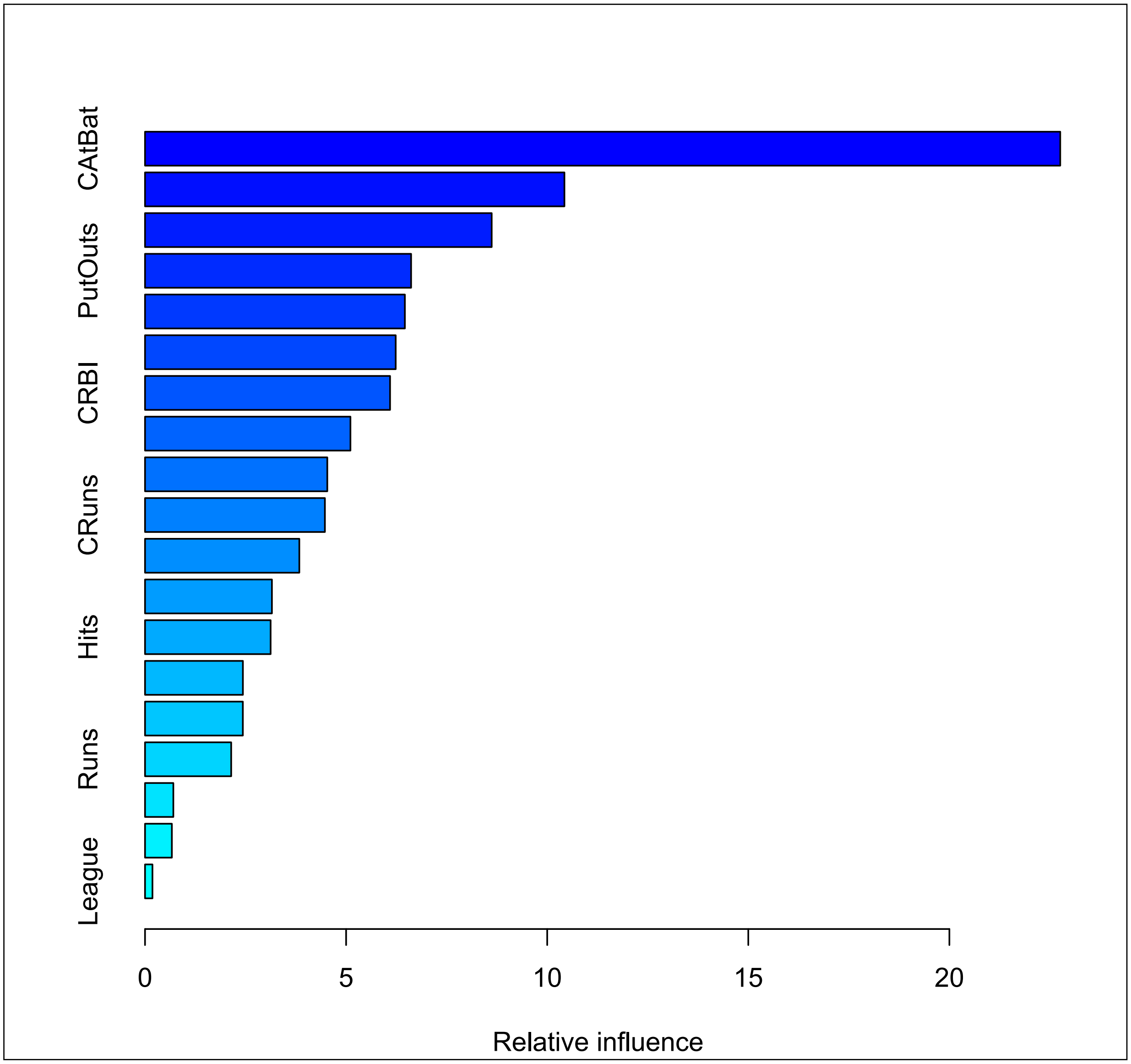
```
> set.seed(134)
> x = model.matrix(Salary ~ ., data = Hitters.train)
> y = Hitters.train$Salary
> x.test = model.matrix(Salary ~ ., data = Hitters.test)
> lasso.fit = glmnet(x, y, alpha = 1)
> lasso.pred = predict(lasso.fit, s = 0.01, newx = x.test)
> mean((Hitters.test$Salary - lasso.pred)^2)
[1] 0.4700537
```

Both linear and regularization models have higher test MSE than boosting.

- (f)

  Answer:

```
> boost.best = gbm(Salary ~ ., data = Hitters.train, distribution = 
"gaussian", 
+     n.trees = 1000, shrinkage = lambdas[which.min(test.errors)])
> summary(boost.best)
                var    rel.inf
CAtBat       CAtBat 22.7562681
CWalks       CWalks 10.4279674
CHits         CHits  8.6198109
PutOuts     PutOuts  6.6159325
Years         Years  6.4611683
Walks         Walks  6.2331148
CRBI           CRBI  6.0926744
CHmRun       CHmRun  5.1076104
RBI             RBI  4.5321678
CRuns         CRuns  4.4728132
Assists     Assists  3.8366575
HmRun         HmRun  3.1554038
Hits           Hits  3.1229284
AtBat         AtBat  2.4338530
Errors       Errors  2.4324185
Runs           Runs  2.1425481
Division   Division  0.7041949
NewLeague NewLeague  0.6675446
League       League  0.1849234
```



CRBI, CWalks and CAtBat are the three most important variables.

- (g)

  Answer:

```
> library(randomForest)
randomForest 4.6-12
Type rfNews() to see new features/changes/bug fixes.
> set.seed(21)
> rf.hitters = randomForest(Salary ~ ., data = Hitters.train, ntree =
500, mtry = 19)
> rf.pred = predict(rf.hitters, Hitters.test)
> mean((Hitters.test$Salary - rf.pred)^2)
[1] 0.231884.
```

  Bagging produce 0.23 Test MSE, which is lower than the best test MSE of boosting.

Problem 2 (p.335, ex11)

- (a)

  Answer:

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

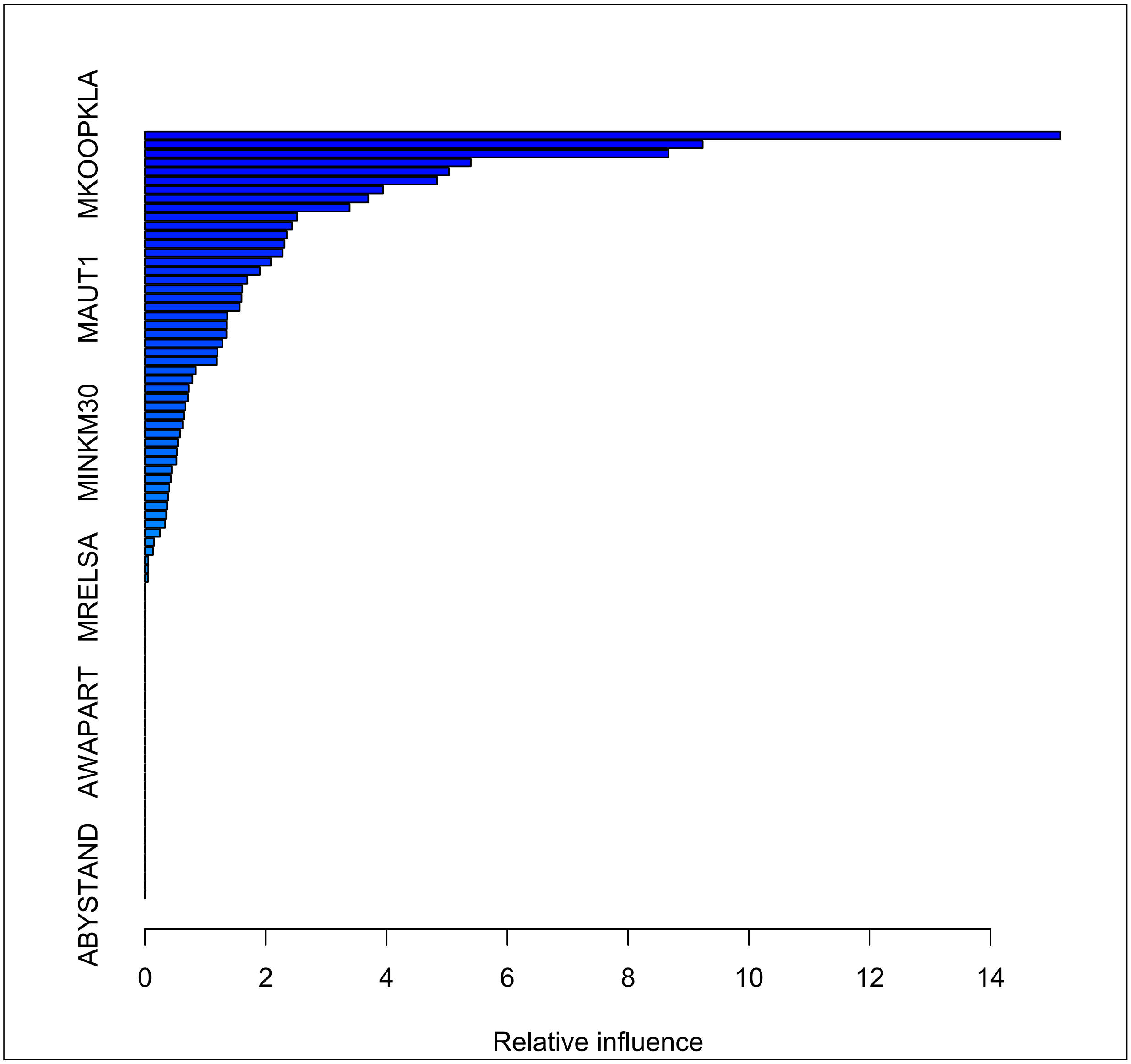
- (b)

  Answer:

```
> library(gbm)
> set.seed(342)
> boost.caravan = gbm(Purchase ~ ., data = Caravan.train, n.trees = 1000,
shrinkage = 0.01,
+     distribution = "bernoulli")
Warning messages:
1: In gbm.fit(x, y, offset = offset, distribution = distribution, w = w,  :
  variable 50: PVRAAUT has no variation.
2: In gbm.fit(x, y, offset = offset, distribution = distribution, w = w,  :
  variable 71: AVRAAUT has no variation.
> summary(boost.caravan)
              var     rel.inf
PPERSAUT PPERSAUT 15.15534009
MKOOPKLA MKOOPKLA  9.23499526
MOPLHOOG MOPLHOOG  8.67017024
MBERMIDD MBERMIDD  5.39403655
MGODGE     MGODGE  5.03047673
PBRAND     PBRAND  4.83740038
MINK3045 MINK3045  3.94305387
ABRAND     ABRAND  3.69692919
MOSTYPE   MOSTYPE  3.38768960
PWAPART   PWAPART  2.51970169
MGODPR     MGODPR  2.43689096
MSKC         MSKC  2.34594774
MAUT2       MAUT2  2.30973409
MFWEKIND MFWEKIND  2.27959503
MBERARBG MBERARBG  2.08245286
MSKA         MSKA  1.90020973
PBYSTAND PBYSTAND  1.69481877
MGODOV     MGODOV  1.61147668
MAUT1       MAUT1  1.59879109
MBERHOOG MBERHOOG  1.56791308
MINK7512 MINK7512  1.36255296
MSKB1       MSKB1  1.35071475
```

```
MINKGEM    MINKGEM  1.34913011
MRELGE      MRELGE  1.28204167
MAUT0        MAUT0  1.19929798
MHHUUR      MHHUUR  1.19158719
MFGEKIND MFGEKIND  0.84203310
MRELOV      MRELOV  0.78554535
MZPART      MZPART  0.72191139
MINK4575 MINK4575  0.70935967
MSKB2        MSKB2  0.66694112
APERSAUT APERSAUT  0.64644681
MGODRK      MGODRK  0.62380797
MSKD          MSKD  0.58168337
MINKM30    MINKM30  0.54392696
PMOTSCO    PMOTSCO  0.52708603
MOPLMIDD MOPLMIDD  0.52091706
MGEMOMV    MGEMOMV  0.44231264
MZFONDS    MZFONDS  0.43037800
PLEVEN      PLEVEN  0.39901552
MHKOOP      MHKOOP  0.37672230
MBERARBO MBERARBO  0.36653424
MBERBOER MBERBOER  0.35290257
MINK123M MINK123M  0.33559225
MGEMLEEF MGEMLEEF  0.24937634
MFALLEEN MFALLEEN  0.14898856
MOSHOOFD MOSHOOFD  0.13265308
MOPLLAAG MOPLLAAG  0.05654615
MBERZELF MBERZELF  0.05589282
MAANTHUI MAANTHUI  0.05047841
MRELSA      MRELSA  0.00000000
PWABEDR    PWABEDR  0.00000000
PWALAND    PWALAND  0.00000000
PBESAUT    PBESAUT  0.00000000
PVRAAUT    PVRAAUT  0.00000000
PAANHANG PAANHANG  0.00000000
PTRACTOR PTRACTOR  0.00000000
PWERKT      PWERKT  0.00000000
PBROM        PBROM  0.00000000
PPERSONG PPERSONG  0.00000000
PGEZONG    PGEZONG  0.00000000
PWAOREG    PWAOREG  0.00000000
PZEILPL    PZEILPL  0.00000000
PPLEZIER PPLEZIER  0.00000000
PFIETS      PFIETS  0.00000000
PINBOED    PINBOED  0.00000000
AWAPART    AWAPART  0.00000000
AWABEDR    AWABEDR  0.00000000
```

```
AWALAND   AWALAND  0.00000000
ABESAUT   ABESAUT  0.00000000
AMOTSCO   AMOTSCO  0.00000000
AVRAAUT   AVRAAUT  0.00000000
AAANHANG AAANHANG  0.00000000
ATRACTOR ATRACTOR  0.00000000
AWERKT     AWERKT  0.00000000
ABROM       ABROM  0.00000000
ALEVEN     ALEVEN  0.00000000
APERSONG APERSONG  0.00000000
AGEZONG   AGEZONG  0.00000000
AWAOREG   AWAOREG  0.00000000
AZEILPL   AZEILPL  0.00000000
APLEZIER APLEZIER  0.00000000
AFIETS     AFIETS  0.00000000
AINBOED   AINBOED  0.00000000
ABYSTAND ABYSTAND  0.00000000
```



PPERSAUT, MKOOPKLA and MOPLHOOG are three most important variables in that order

- (c)

Answer:

```
> boost.prob = predict(boost.caravan, Caravan.test, n.trees = 1000,
type = "response")
> boost.pred = ifelse(boost.prob > 0.2, 1, 0)
> table(Caravan.test$Purchase, boost.pred)
   boost.pred
       0    1
  0 4396  137
  1  255   34
> 34/(137 + 34)
```

```
[1] 0.1988304
```

19.9% people predicted to make purchease will actually do.

```
> lm.caravan = glm(Purchase ~ ., data = Caravan.train, family = binomial)
Warning message:
glm.fit: fitted probabilities numerically 0 or 1 occurred
> lm.prob = predict(lm.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
> lm.pred = ifelse(lm.prob > 0.2, 1, 0)
> table(Caravan.test$Purchase, lm.pred)
   lm.pred
       0    1
  0 4183  350
  1  231   58
> 58/(350 + 58)
[1] 0.1421569
```

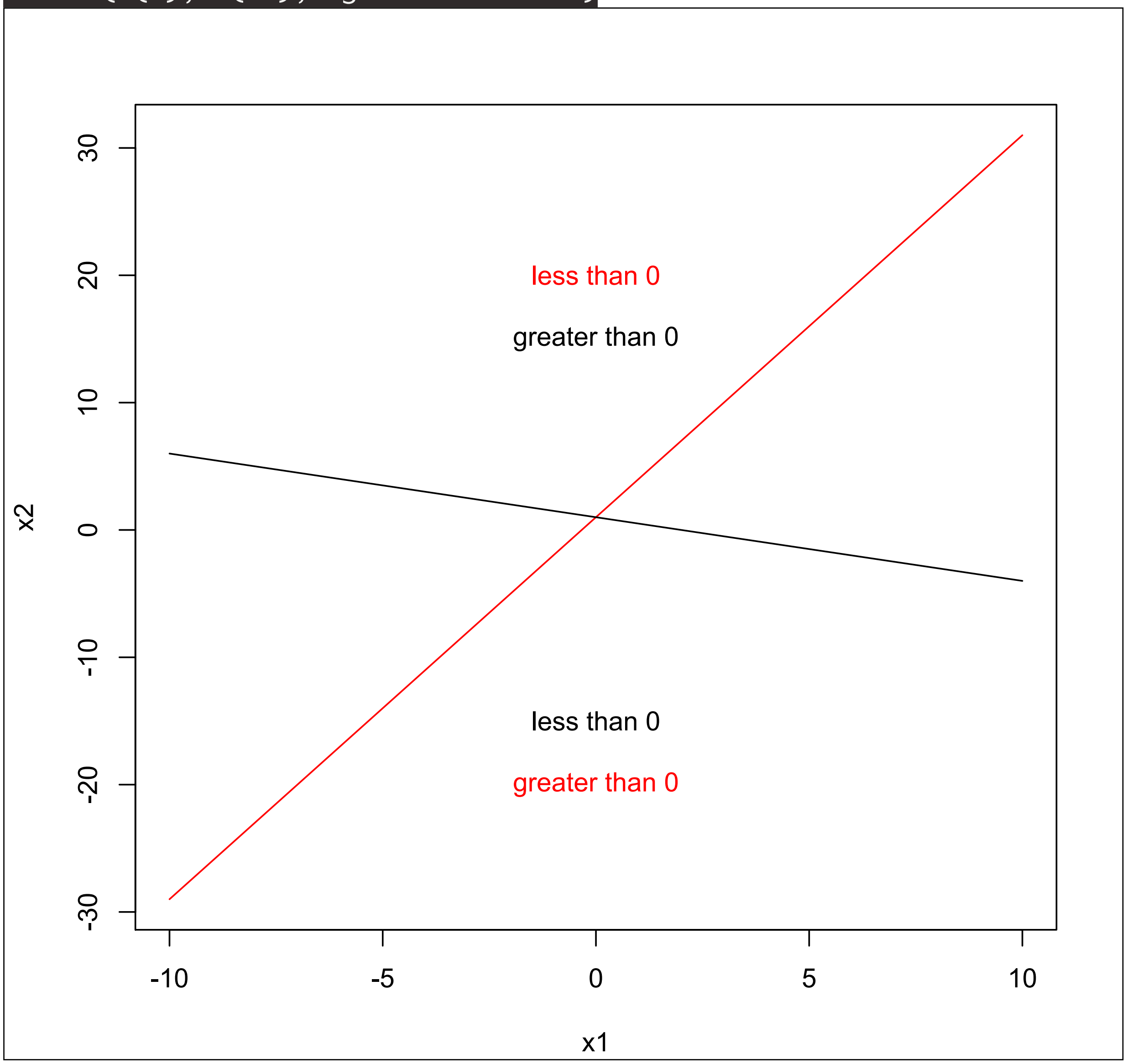
Logistic regression about 14% people predicted to make purchase will do.

Problem 3 (p.368, ex1)

-

Answer:

```
> x1 = -10:10
> x2 = 1 + 3 * x1
> plot(x1, x2, type = "l", col = "red")
> text(c(0), c(-20), "greater than 0", col = "red")
> text(c(0), c(20), "less than 0", col = "red")
> lines(x1, 1 - x1/2)
> text(c(0), c(-15), "less than 0")
> text(c(0), c(15), "greater than 0")
```



Problem 4 (p.371, ex8)

(a)

Answer:

```
> library(ISLR)
> set.seed(9004)
> train = sample(dim(OJ)[1], 800)
> OJ.train = OJ[train, ]
> OJ.test = OJ[-train, ]
```

(b)

```
> library(e1071)
> svm.linear = svm(Purchase ~ ., kernel = "linear", data = OJ.train, 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

 ( 217 215 )

Number of Classes:  2

Levels:
 CH MM
```

Answer:

SVC generates 432 support vectors out of 800 training points. 217 belong to level CH and 215 belng to level MM.

(c)

```
> train.pred = predict(svm.linear, OJ.train)
> table(OJ.train$Purchase, train.pred)
    train.pred
      CH  MM
  CH 439  53
  MM  82 226
> (82 + 53)/(439 + 53 + 82 + 226)
```

```
[1] 0.16875
> test.pred = predict(svm.linear, OJ.test)
> table(OJ.test$Purchase, test.pred)
    test.pred
      CH  MM
  CH 142  19
  MM  29  80
> (19 + 29)/(142 + 19 + 29 + 80)
[1] 0.1777778
```

Answer: The Training error rate is 16.9% and test error rate is 17.8%.

(d)

```
> set.seed(1554)
> tune.out = tune(svm, Purchase ~ ., data = OJ.train, kernel = "linear",
ranges = list(cost = 10^seq(-2,
+      1, by = 0.25)))
summary(tune.out)
> summary(tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
      cost
 0.3162278

- best performance: 0.16875

- Detailed performance results:
          cost   error dispersion
1   0.01000000 0.16875 0.03691676
2   0.01778279 0.16875 0.03397814
3   0.03162278 0.17125 0.03230175
4   0.05623413 0.17250 0.03162278
5   0.10000000 0.17000 0.03291403
6   0.17782794 0.17125 0.03335936
7   0.31622777 0.16875 0.03498512
8   0.56234133 0.17000 0.03129164
9   1.00000000 0.16875 0.03397814
10  1.77827941 0.16875 0.03240906
11  3.16227766 0.16875 0.03294039
12  5.62341325 0.17125 0.03120831
13 10.00000000 0.17125 0.03283481
```

Answer:

We can see taht the optimal cost is 0.3162278.

(e)

```
> svm.linear = svm(Purchase ~ ., kernel = "linear", data = OJ.train,
cost = tune.out$best.parameters$cost)
> train.pred = predict(svm.linear, OJ.train)
> table(OJ.train$Purchase, train.pred)
    train.pred
      CH  MM
  CH 435  57
  MM  71 237
> (57 + 71)/(435 + 57 + 71 + 237)
[1] 0.16
> test.pred = predict(svm.linear, OJ.test)
> table(OJ.test$Purchase, test.pred)
    test.pred
      CH  MM
  CH 141  20
  MM  29  80
> (29 + 20)/(141 + 20 + 29 + 80)
[1] 0.1814815
```

Answer:
Using the above cost, the training error decreases to 16% while test error increases to 18.1%.

(f)

```
> set.seed(410)
> svm.radial = svm(Purchase ~ ., data = OJ.train, kernel = "radial")
> 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:  367

 ( 184 183 )
```

```
Number of Classes:  2

Levels: 
 CH MM
```

```
> train.pred = predict(svm.radial, OJ.train)
> table(OJ.train$Purchase, train.pred)
    train.pred
      CH  MM
  CH 452  40
  MM  78 230
> (40 + 78)/(452 + 40 + 78 + 230)
[1] 0.1475
> test.pred = predict(svm.radial, OJ.test)
> table(OJ.test$Purchase, test.pred)
    test.pred
      CH  MM
  CH 146  15
  MM  27  82
> (27 + 15)/(146 + 15 + 27 + 82)
[1] 0.1555556
```

Answer:
We see that the radial basis kernel with default gamma creates 367 support vectors, out of which, 184 belong to level CH and remaining 183 belong to lecvel MM. The classifier has a training error of 14.7% and a test error of 15.6% which is a slight improvement over linear kernel. To find optimal gamma:

```
> set.seed(755)
> tune.out = tune(svm, Purchase ~ ., data = OJ.train, kernel = "radial",
ranges = list(cost = 10^seq(-2, 
+     1, by = 0.25)))
summary(tune.out)
> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation 

- best parameters:
      cost
 0.5623413

- best performance: 0.165 

- Detailed performance results:
```

```
            cost   error dispersion
1    0.01000000 0.38500 0.06258328
2    0.01778279 0.38500 0.06258328
3    0.03162278 0.37625 0.06908379
4    0.05623413 0.21000 0.03855011
5    0.10000000 0.18625 0.03143004
6    0.17782794 0.18375 0.03230175
7    0.31622777 0.17125 0.03438447
8    0.56234133 0.16500 0.03763863
9    1.00000000 0.17500 0.03584302
10   1.77827941 0.17375 0.04059026
11   3.16227766 0.17625 0.03747684
12   5.62341325 0.17625 0.03839216
13 10.00000000 0.17375 0.03458584
```

```
> svm.radial = svm(Purchase ~ ., data = OJ.train, kernel = "radial", cost =
tune.out$best.parameters$cost)
> train.pred = predict(svm.radial, OJ.train)
> table(OJ.train$Purchase, train.pred)
    train.pred
      CH  MM
  CH 452  40
  MM  77 231
> (77 + 40)/(452 + 40 + 77 + 231)
[1] 0.14625
> test.pred = predict(svm.radial, OJ.test)
> table(OJ.test$Purchase, test.pred)
    test.pred
      CH  MM
  CH 146  15
  MM  28  81
> (28 + 15)/(146 + 15 + 28 + 81)
[1] 0.1592593
```

Tuning slightly decreases training error to 14.6% and increases test error to 16% which is still better than linear kernel.

(g)

```
> set.seed(8112)
> svm.poly = svm(Purchase ~ ., data = OJ.train, kernel = "poly", degree = 2)
> summary(svm.poly)

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


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

Number of Support Vectors:  452

 ( 232 220 )


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 460  32
  MM 105 203
> (12 + 37)/(149 + 12 + 37 + 72)
[1] 0.1814815
```

According to the summary, polynomial kernel produces 452 support vectors, out of which, 232 are level CH and 220 are level MM. This kernel produces a train error of 17.1% and a test error of 18.1% which are slightly higher than the errors produces by radial kernel but lower than the errors produced by linear kernel.

```
> set.seed(322)
> tune.out = tune(svm, Purchase ~ ., data = OJ.train, kernel = "poly",
degree = 2,
```

```
+     ranges = list(cost = 10^seq(-2, 1, by = 0.25)))
summary(tune.out)
> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
     cost
 5.623413

- best performance: 0.18375

- Detailed performance results:
          cost   error dispersion
1   0.01000000 0.38500 0.05426274
2   0.01778279 0.36750 0.05075814
3   0.03162278 0.35750 0.05177408
4   0.05623413 0.34250 0.04937104
5   0.10000000 0.31500 0.05230785
6   0.17782794 0.24875 0.03928617
7   0.31622777 0.20875 0.05684103
8   0.56234133 0.20875 0.05653477
9   1.00000000 0.20000 0.06095308
10  1.77827941 0.19375 0.04497299
11  3.16227766 0.18625 0.04185375
12  5.62341325 0.18375 0.03335936
13 10.00000000 0.18375 0.04041881

> svm.poly = svm(Purchase ~ ., data = OJ.train, kernel = "poly", degree = 2,
cost = tune.out$best.parameters$cost)
> train.pred = predict(svm.poly, OJ.train)
> table(OJ.train$Purchase, train.pred)
    train.pred
      CH  MM
  CH 455  37
  MM  84 224
> (37 + 84)/(455 + 37 + 84 + 224)
[1] 0.15125
> test.pred = predict(svm.poly, OJ.test)
> table(OJ.test$Purchase, test.pred)
    test.pred
      CH  MM
  CH 148  13
  MM  34  75
```

```
> (13 + 34)/(148 + 13 + 34 + 75)
[1] 0.1740741
```

We can see that tuning reduces the training error to 15.12% and test error to 17.4% which is worser than radial kernel but better than linear kernel.

(h)

Radial basis kernel is the best since it produces minimum misclassification error on training and test data set.