Homework 7 - Stat 495

Cassandra Jin

Due Wednesday, Nov. 8th by midnight (11:59 pm)

Practicing Academic Integrity

If you worked with others or used resources outside of provided course material (anything besides our textbook(s), course materials in Moodle/Git repo, R help menu) to complete this assignment, please acknowledge them below using a bulleted list.

I acknowledge the following individuals with whom I worked on this assignment:

Name(s) and corresponding problem(s)

-

I used the following sources to help complete this assignment:

Source(s) and corresponding problem(s)

-

PROBLEMS TO TURN IN: Additional Problems 1-4

The first three problems use the bbb2 data set from the QSARdata package. The final problem covers some concepts from the methods in Chapters 17, 18, and 19, without a data set / application.

For the applied problems, the response variable of interest is bbb2_Class, which can be found in the bbb2_Outcome data set. We have joined the outcome variable to the QuickProp data set that we want to focus on below. You can read the associated help file in R to learn more about the data set.

```r
data(bbb2)
#?bbb2 # for variable reference and information
mybbb2 <- left_join(bbb2_QuickProp, bbb2_Outcome) %>% select(- Molecule)
```

```
## Joining with `by = join_by(Molecule)`
```

```r
tally(~ bbb2_Class, data = mybbb2)
```

```
## bbb2_Class
## Crosses DoesNot
##      45      35
```

Our goal for the applied problems below is to use the recent methods from class (Chapter 17, 18, and 19) to predict the response variable, whether each compound "crosses" the blood-brain barrier or "does not" cross. You should use the *mybbb2* data set going forward. Note that there are 51 variables at the moment, and the last variable "Class" is the target, but if you open the data set to view, it will only show 50 variables by default. Class is there, but you have to use the arrows to see it.

```r
# We loaded a lot of data sets we don't need anymore
# remove them to clean up your workspace
remove(bbb2_AtomPair, bbb2_Daylight_FP, bbb2_Dragon, bbb2_Lcalc, bbb2_moe2D,
       bbb2_moe2D_FP, bbb2_moe3D, bbb2_Outcome, bbb2_PipelinePilot_FP,
       bbb2_QuickProp)
```

To avoid issues with reproducibility, you should set a seed in EACH chunk below where you do a random process, whether that is setting up the train/test split or fitting a model that has some random process involved.

Additional Problem 1

Your task for this problem is to fit the models described in parts c, d and e, and then compare them in part f. Use all available predictor variables (except what is removed in part a), with no re-expressions. The same training/test split will be used in Additional Problems 2 and 3 as well (i.e. you only make this once).

> part a. One variable in the data set, QikProp_.amidine causes issues with some of these methods. We will remove it here, but can you see why it is problematic? Explain why this variable is not very useful for this analysis.

SOLUTION:

```
favstats(mybbb2$QikProp_.amidine)
```

```
##  min Q1 median Q3 max   mean       sd  n missing
##    0  0      0  0   1 0.0125 0.111803 80       0
```

The variable `QikProp_.amidine` only has one non-zero observation of 1. It does not contain any values that would allow us to find a relationship between `QikProp_.amidine` and the other variables.

```
# run once you are ready to remove the variable to proceed
# this variable is not used anywhere below, so overwrite the data set
mybbb2 <- mybbb2 %>%
  select(-QikProp_.amidine) %>%
  dplyr::mutate(class_val = ifelse(Class == "Crosses", 1, 0))
```

> part b. Create an appropriate training/test split from mybbb2 with a ratio of 70/30 to use throughout the problems. As always, be sure your split is reproducible.

SOLUTION:

```
set.seed(495)

n <- nrow(mybbb2)
train_index <- sample(1:n, 0.7 * n)
test_index <- setdiff(1:n, train_index)

train <- mybbb2[train_index, ]
test <- mybbb2[test_index, ]
```

> part c. Create an appropriate model to predict Class with the training set using bagging with 1000 trees and otherwise with default settings for tuning parameter values.

SOLUTION:

```
set.seed(495)
mybbb2_bag <- randomForest(Class ~ ., data = train, mtry = 49, ntree = 1000)
mybbb2_bag
```

```
##
## Call:
##  randomForest(formula = Class ~ ., data = train, mtry = 49, ntree = 1000)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 49
##
##         OOB estimate of  error rate: 0%
## Confusion matrix:
##         Crosses DoesNot class.error
## Crosses      31       0           0
## DoesNot       0      25           0
```

part d. Create an appropriate model to predict Class with the training set using a random forest with 1000 trees and otherwise with default settings for tuning parameter values.

SOLUTION:

```
set.seed(495)
mybbb2_rf <- randomForest(Class ~ ., data = train, mtry = 17, ntree = 1000) # random forest: m = p/3
mybbb2_rf
```

```
##
## Call:
##  randomForest(formula = Class ~ ., data = train, mtry = 17, ntree = 1000)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 17
##
##         OOB estimate of  error rate: 3.57%
## Confusion matrix:
##         Crosses DoesNot class.error
## Crosses      30       1   0.0322581
## DoesNot       1      24   0.0400000
```

part e. Create an appropriate model to predict Class with the training set using boosting with 500 trees and otherwise with default settings for tuning parameter values.

SOLUTION:

```
set.seed(495)
mybbb2_boost <- gbm(class_val ~ .,
                    data = train,
                    distribution = "bernoulli",
                    n.trees = 500,
                    interaction.depth = 4)
mybbb2_boost
```

```
## gbm(formula = class_val ~ ., distribution = "bernoulli", data = train,
##     n.trees = 500, interaction.depth = 4)
## A gradient boosted model with bernoulli loss function.
## 500 iterations were performed.
## There were 50 predictors of which 35 had non-zero influence.
```

part f. Parts c, d, and e only required you to fit models. Now we want to compare their performance. Use an appropriate measure to compare the three models in terms of their model performance based on the test set. Be sure your choice of measure is clear. Summarize your findings. Then discuss which model you would choose for predicting Class. Explain your choice.

SOLUTION:

```
# bagging
mybbb2_bagest <- predict(mybbb2_bag,
                         newdata = test,
                         n.trees = 500, type = "response")
mybbb2_bagpred <- ifelse(mybbb2_bagest == "Crosses", 1, 0)
tally(~ mybbb2_bagpred)
```

```
## mybbb2_bagpred
##  0  1
## 10 14
```

```
table(test$class_val, mybbb2_bagpred)
```

```
##    mybbb2_bagpred
##      0  1
##   0 10  0
##   1  0 14
```

```
0/24 # estimated TER
```

```
## [1] 0
```

```
# random forest
mybbb2_rfest <- predict(mybbb2_rf,
                         newdata = test,
                         n.trees = 500, type = "response")
mybbb2_rfpred <- ifelse(mybbb2_rfest == "Crosses", 1, 0)
tally(~ mybbb2_rfpred)
```

```
## mybbb2_rfpred
##  0  1
## 13 11
```

```
table(test$class_val, mybbb2_rfpred)
```

```
##    mybbb2_rfpred
##      0  1
##   0 10  0
##   1  3 11
```

```
3/24
```

```
## [1] 0.125
```

```
# boosting
mybbb2_boostest <- predict(mybbb2_boost,
                         newdata = test,
                         n.trees = 500, type = "response")
mybbb2_boostpred <- ifelse(mybbb2_boostest >= 0.5, 1, 0)
tally(~ mybbb2_boostpred)
```

```
## mybbb2_boostpred
##  0  1
## 10 14
```

```
table(test$class_val, mybbb2_boostpred)
```

```
##    mybbb2_boostpred
##      0  1
##   0 10  0
##   1  0 14
```

```
0/24
```

```
## [1] 0
```

The bagging TER is 0% (from the confusion matrix) and the estimated TER via the OOB error rate is 0%, and similarly, the TER based on the confusion matrix for boosting is 0%. The random forest model, however, yields a TER (from the confusion matrix) of 12.5% and the estimated TER via the OOB error rate is 3.57%. Based on the three models' predictions for the test set, the bagging and the boosting methods are equally strong. It is useful to see from the boosting output that there were 50 predictors of which 35 had non-zero influence, and the parameter setting of “bernoulli” makes it clearer to me that the outcome is either Crosses or DoesNot (whereas my impression from bagging is that there just happens to be 2 outcomes but it is not necessarily binary), so ultimately, I would choose the boosting model for predicting Class.

Additional Problem 2

Your task for this problem is to fit the models described in parts a, b, and c, and then compare them in part d. Use all available predictor variables, with no re-expressions. Use the same training/test data as above.

> part a. Create an appropriate model to predict Class with the training set using a neural net with a single hidden layer of 15 nodes, and otherwise with default settings for tuning parameter values.

SOLUTION:

```r
set.seed(495)
mybbb2_nnet1 <- nnet(Class ~ ., train, size = 15)
```

```
## # weights:  781
## initial  value 53.108387
## iter  10 value 19.715099
## iter  20 value 19.329266
## iter  30 value 19.328592
## iter  30 value 19.328592
## iter  30 value 19.328592
## final  value 19.328592
## converged
```

> part b. Create an appropriate model to predict Class with the training set using a neural net with a single hidden layer of 15 nodes, and a decay parameter of 5e-4, and otherwise with default settings for tuning parameter values.

SOLUTION:

```r
set.seed(495)
mybbb2_nnet2 <- nnet(Class ~ ., train, size = 15, decay = 5e-4)
```

```
## # weights:  781
## initial  value 53.172138
## iter  10 value 19.820197
## iter  20 value 17.458213
## iter  30 value 17.394875
## iter  40 value 13.944369
## iter  50 value 13.304133
## iter  60 value 11.858260
## iter  70 value 9.669896
## iter  80 value 9.491585
## iter  90 value 8.995502
## iter 100 value 8.032147
## final  value 8.032147
## stopped after 100 iterations
```

> part c. Create an appropriate model to predict Class with the training set using a neural net with a single hidden layer of 15 nodes, a decay parameter of 5e-4, and a value for maxit that allows for convergence, and otherwise with default settings for tuning parameter values.

SOLUTION:

```r
set.seed(495)
mybbb2_nnet3 <- nnet(Class ~ ., train, size = 15, maxit = 1030, decay = 5e-4)
```

```
## # weights:  781
## initial  value 53.172138
## iter  10 value 19.820197
## iter  20 value 17.458213
## iter  30 value 17.394875
## iter  40 value 13.944369
## iter  50 value 13.304133
## iter  60 value 11.858260
## iter  70 value 9.669896
## iter  80 value 9.491585
## iter  90 value 8.995502
## iter 100 value 8.032147
## iter 110 value 5.647028
## iter 120 value 5.177518
## iter 130 value 5.145872
## iter 140 value 4.151611
## iter 150 value 4.115438
## iter 160 value 4.037198
## iter 170 value 2.620296
## iter 180 value 2.530702
## iter 190 value 2.491338
## iter 200 value 2.486392
## iter 210 value 2.476014
## iter 220 value 2.451636
## iter 230 value 2.157568
## iter 240 value 1.824685
## iter 250 value 1.648380
## iter 260 value 1.612384
## iter 270 value 1.603624
## iter 280 value 1.598128
## iter 290 value 1.593874
## iter 300 value 1.591437
## iter 310 value 1.589951
## iter 320 value 1.585839
## iter 330 value 0.654559
## iter 340 value 0.462120
## iter 350 value 0.355634
## iter 360 value 0.321225
## iter 370 value 0.306201
## iter 380 value 0.292285
## iter 390 value 0.284934
## iter 400 value 0.280663
## iter 410 value 0.275076
## iter 420 value 0.270544
## iter 430 value 0.267811
## iter 440 value 0.266434
## iter 450 value 0.264217
## iter 460 value 0.240099
## iter 470 value 0.233325
## iter 480 value 0.218869
## iter 490 value 0.204018
## iter 500 value 0.191685
## iter 510 value 0.186779
## iter 520 value 0.180367
## iter 530 value 0.168161
## iter 540 value 0.149122
## iter 550 value 0.140941
## iter 560 value 0.138753
## iter 570 value 0.137618
## iter 580 value 0.135538
## iter 590 value 0.130894
## iter 600 value 0.128678
## iter 610 value 0.126629
## iter 620 value 0.124106
## iter 630 value 0.122155
## iter 640 value 0.119331
## iter 650 value 0.117746
## iter 660 value 0.115826
## iter 670 value 0.114426
## iter 680 value 0.110140
## iter 690 value 0.109456
## iter 700 value 0.107731
## iter 710 value 0.102680
## iter 720 value 0.101251
## iter 730 value 0.100493
## iter 740 value 0.097461
## iter 750 value 0.095208
## iter 760 value 0.092247
## iter 770 value 0.091993
## iter 780 value 0.090749
## iter 790 value 0.089154
## iter 800 value 0.086755
## iter 810 value 0.084505
## iter 820 value 0.083945
## iter 830 value 0.081989
## iter 840 value 0.079738
## iter 850 value 0.076792
## iter 860 value 0.074926
## iter 870 value 0.073336
## iter 880 value 0.070950
## iter 890 value 0.070553
## iter 900 value 0.070227
## iter 910 value 0.069103
## iter 920 value 0.068386
## iter 930 value 0.068071
## iter 940 value 0.067758
## iter 950 value 0.067193
## iter 960 value 0.066697
## iter 970 value 0.066473
## iter 980 value 0.066198
## iter 990 value 0.065928
## iter1000 value 0.065343
## iter1010 value 0.064709
## iter1020 value 0.064032
## iter1030 value 0.063825
## final  value 0.063825
## stopped after 1030 iterations
```

part d. Parts a, b, and c only required you to fit models. Now we want to compare their performance. Use an appropriate measure to compare the three models in terms of their model performance based on the test set. Be sure your choice of measure is clear. Summarize your findings. Then discuss which model you would choose for predicting Class. Explain your choice.

SOLUTION:

```r
# a) 15 nodes
trainpred1 <- predict(mybbb2_nnet1, type = "class")
pred1 <- predict(mybbb2_nnet1, newdata = test, type = "class")
tally(Class ~ trainpred1, data = train)
```

```
##          trainpred1
## Class     Crosses DoesNot
##   Crosses      24       7
##   DoesNot       1      24
```

```r
(1+7)/56
```

```
## [1] 0.142857
```

```r
tally(Class ~ pred1, data = test)
```

```
##          pred1
## Class     Crosses DoesNot
##   Crosses       8       6
##   DoesNot       1       9
```

```r
(1+6)/24
```

```
## [1] 0.291667
```

```r
# b) 15 nodes, decay parameter of 5e-4
trainpred2 <- predict(mybbb2_nnet2, type = "class")
pred2 <- predict(mybbb2_nnet2, newdata = test, type = "class")
tally(Class ~ trainpred2, data = train)
```

```
##          trainpred2
## Class     Crosses DoesNot
##   Crosses      28       3
##   DoesNot       0      25
```

```r
3/56
```

```
## [1] 0.0535714
```

```r
tally(Class ~ pred2, data = test)
```

```
##          pred2
## Class     Crosses DoesNot
##   Crosses       8       6
##   DoesNot       2       8
```

```r
(6+2)/24
```

```
## [1] 0.333333
```

```r
# c) 15 nodes, decay parameter of 5e-4, value for maxit that allows for convergence
trainpred3 <- predict(mybbb2_nnet3, type = "class")
pred3 <- predict(mybbb2_nnet3, newdata = test, type = "class")
tally(Class ~ trainpred3, data = train)
```

```
##          trainpred3
## Class     Crosses DoesNot
##   Crosses      31       0
##   DoesNot       0      25
```

```
0/56
```

```
## [1] 0
```

```
tally(Class ~ pred3, data = test)
```

```
##          pred3
## Class     Crosses DoesNot
##   Crosses       8       6
##   DoesNot       2       8
```

```
(6+2)/24
```

```
## [1] 0.333333
```

Based on the confusion matrices from all three models predicting on both the training and test sets, we see that the model from part b has the highest error rate for both training and test sets. The model from part a, with the most default settings, yielded 14.3% error on the training set and 29.2% error on the test. Although the model from part c took the longest to run, it predicted perfectly on the training set and had only a slightly higher error rate on the test set compared to that of the part a model. So for predicting Class, I would choose the third neural net model, with a single hidden layer of 15 nodes, a decay parameter of 5e-4, and a value for maxit (1030) that allows for convergence, and otherwise with default settings for tuning parameter values.

Additional Problem 3

Your task for this problem is to fit the models described in parts a, b, and c, and then compare them in part d. Use all available predictor variables, with no re-expressions. Use the same training/test data as above. Finally, part e will have you compare the best models from Additional Problems 1, 2, and 3.

> part a. Create an appropriate model to predict Class with the training set using an SVM with a radial kernel and a gamma of 0.75, and otherwise with default settings for tuning parameter values.

SOLUTION:

```r
svm1 <- svm(Class ~ ., data = train, gamma = 0.75, kernel = "radial")
summary(svm1)
```

```
##
## Call:
## svm(formula = Class ~ ., data = train, gamma = 0.75, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  56
##
##  ( 25 31 )
##
##
## Number of Classes:  2
##
## Levels:
##  Crosses DoesNot
```

> part b. Create an appropriate model to predict Class with the training set using an SVM with a polynomial kernel and a gamma of 0.5, and otherwise with default settings for tuning parameter values.

SOLUTION:

```r
svm2 <- svm(Class ~ ., data = train, gamma = 0.5, kernel = "poly")
summary(svm2)
```

```
##
## Call:
## svm(formula = Class ~ ., data = train, gamma = 0.5, kernel = "poly")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  27
##
```

```
##  ( 11 16 )
##
##
## Number of Classes:  2
##
## Levels:
##  Crosses DoesNot
```

part c. Create an appropriate model to predict Class with the training set using an SVM with a polynomial kernel and a gamma of 0.0001, and otherwise with default settings for tuning parameter values.

SOLUTION:

```
svm3 <- svm(Class ~ ., data = train, gamma = 0.0001, kernel = "poly")
summary(svm3)
```

```
##
## Call:
## svm(formula = Class ~ ., data = train, gamma = 1e-04, kernel = "poly")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  50
##
##  ( 25 25 )
##
##
## Number of Classes:  2
##
## Levels:
##  Crosses DoesNot
```

part d. Parts a, b, and c only required you to fit models. Now we want to compare their performance. Use an appropriate measure to compare the three models in terms of model performance based on the test set. Be sure your choice of measure is clear. Summarize your findings. Then discuss which model you would choose for predicting Class. Explain your choice.

SOLUTION:

```
# a)
svm1predtrain <- predict(svm1, train)
svm1predtest <- predict(svm1, test)
tally(Class ~ svm1predtrain, data = train)
```

```
##          svm1predtrain
## Class     Crosses DoesNot
##   Crosses      31       0
##   DoesNot       0      25
```

```
0/56
```

```
## [1] 0
```

```
tally(Class ~ svm1predtest, data = test)
```

```
##          svm1predtest
## Class     Crosses DoesNot
##   Crosses      14       0
##   DoesNot      10       0
```

```
(10+0)/24
```

```
## [1] 0.416667
```

```
# b)
svm2predtrain <- predict(svm2, train)
svm2predtest <- predict(svm2, test)
tally(Class ~ svm2predtrain, data = train)
```

```
##          svm2predtrain
## Class     Crosses DoesNot
##   Crosses      31       0
##   DoesNot       0      25
```

```
0/56
```

```
## [1] 0
```

```
tally(Class ~ svm2predtest, data = test)
```

```
##          svm2predtest
## Class     Crosses DoesNot
##   Crosses      12       2
##   DoesNot       1       9
```

```
(1+2)/24
```

```
## [1] 0.125
```

```
# c)
svm3predtrain <- predict(svm3, train)
svm3predtest <- predict(svm3, test)
tally(Class ~ svm3predtrain, data = train)
```

```
##          svm3predtrain
## Class     Crosses DoesNot
##   Crosses      31       0
##   DoesNot      25       0
```

```
25/56
```

```
## [1] 0.446429
```

```
tally(Class ~ svm3predtest, data = test)
```

```
##          svm3predtest
## Class     Crosses DoesNot
##   Crosses      14       0
##   DoesNot      10       0
```

```
(10+0)/24
```

```
## [1] 0.416667
```

Based on the confusion matrices from all three SVMs predicting on both the training and test sets, we see that the model from part c does terribly for predicting Class, misclassifying all DoesNot observations for both the training and test sets. The models from part a and c both predicted Class perfectly on the training set, but the part a model yielded 41.7% error on the test set while the part c model had only 12.5% error on the test set. Thus, I would choose the second SVM a polynomial kernel and a gamma of 0.5, and otherwise with default settings for tuning parameter values.

> part e. Look over your responses to Additional Problems 1, 2, and 3 in terms of your final model from each method/problem. Compare these three models, and explain which you would choose as an overarching final model to predict Class. Explain your choice. Your final choice may be determined by performance in conjunction with any factors you think are relevant.

SOLUTION:

```r
# boosting
table(test$class_val, mybbb2_boostpred)
```

```
##    mybbb2_boostpred
##      0  1
##   0 10  0
##   1  0 14
```

```r
0/24
```

```
## [1] 0
```

```r
# neural net: 15 nodes, decay parameter of 5e-4, value for maxit that allows for convergence
tally(Class ~ trainpred3, data = train)
```

```
##          trainpred3
## Class     Crosses DoesNot
##   Crosses      31       0
##   DoesNot       0      25
```

```r
0/56
```

```
## [1] 0
```

```r
tally(Class ~ pred3, data = test)
```

```
##          pred3
## Class     Crosses DoesNot
##   Crosses       8       6
##   DoesNot       2       8
```

```r
(6+2)/24
```

```
## [1] 0.333333
```

```r
# SVM: polynomial kernel, gamma of 0.5
tally(Class ~ svm2predtrain, data = train)
```

```
##          svm2predtrain
## Class     Crosses DoesNot
##   Crosses      31       0
##   DoesNot       0      25
```

```r
0/56
```

```
## [1] 0
```

```r
tally(Class ~ svm2predtest, data = test)
```

```
##          svm2predtest
## Class     Crosses DoesNot
##   Crosses      12       2
##   DoesNot       1       9
```

```r
(1+2)/24
```

```
## [1] 0.125
```

Based on the TERs from the confusion matrices of the three final models (boosting, part c neural net, part b svm), we see that the boosting model is the only one that predicts perfectly on the test set. The boosting method was also the easiest to implement, yielding a 0% error rate with the fewest parameters compared to the other two models, so I would choose the boosting model for predicting Class.

Additional Problem 4

part a. Neural nets and SVMs are both discussed as nonlinear models for prediction. Discuss where the "nonlinearity" is in both of these models.

SOLUTION:

In neural nets, the linear transformations $z_l^{(2)}$ of the x_j from the nonlinear transformation of these are separated, and the "nonlinearity" is in the layer-specific nonlinear transformations $g^{(k)}$ (activation functions), meaning that as data passes through each layer, the network learns hierarchical representations of the input and can capture nonlinear patterns that may exist. SVMs enrich the feature space through nonlinear transformations and basis expansions. A linear model in the enlarged space leads to a nonlinear model in the ambient space via the "kernel trick," which allows the computations to be performed in the n-dimensional space for an arbitrary number of predictors p, so SVMs achieve nonlinearity by mapping data to a higher-dimensional space using kernel functions.

part b. Compare and contrast random forests and boosting in a few sentences. How are they similar? How are they different?

SOLUTION:

Both random forests and boosting represent the fitted model by a sum of regression trees, but random forests grow many deep regression trees to randomized versions (bootstrap sampling, subsampling of the observations, and/or subsampling of the variables) of the training data and average them, while boosting repeatedly grows shallow trees to the residuals and builds up an additive model consisting of a sum of trees. Random forests seek to reduce variance by averaging, as each deep tree has a high variance, and the averaging brings the variance down. Boosting, on the other hand, works to reduce bias.

part c. The concepts of backpropagation and the kernel trick are related, even though they are for neural nets and SVMs, respectively. What do these concepts have in common?

Hint: It has to do with what they help with in their respective methods.

SOLUTION:

In backpropagation, we compute the gradient in single layers, and since the loss part of the objective is a sum, the overall gradient is the sum of these individual gradient elements over the training pairs. The kernel trick expands the p-dimensional feature vector x into a potentially much larger set by serving as an efficient way to compute the inner products for any x, and we can then compute the SVM solution in this enlarged space just as easily as in the original. Both backpropagation and the kernel trick work with individual, smaller components that can be expanded to the whole objective.

part d. Write your own short answer question relating to a concept from Chapter 17, 18, or 19, and then answer it.

Questions should require at least two sentences to answer reasonably well. True/false questions are not short answer questions.

The motivation here is for you to pick something you are still unclear on and ask a question about it. This will make you review the concept in order to answer your question well.

SOLUTION:

Q: Given the shrinkage element of boosting, how are boosting and lasso related?

A: Both methods are based in the process of forward-stagewise fitting via infinitesimal forward-stagewise regression. Lasso also returns as a post-processor for boosting, since boosting with shrinkage does a good job in building a prediction model, but it can end up generating a lot of trees. Due to the shrinkage, many of these trees could be similar to each other, so lasso is used to select a subset of these trees, reweight them, and produce a prediction model with far fewer trees that maintains good accuracy.