Homework 7 - Stat 495

Justin Papagelis

Due Friday, Nov. 4th 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: Add 1, Add 2, Add 3, Add 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.

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

```
## Joining, by = "Molecule"
```

```
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.

```
# 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.

Add 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 Add 2 and Add 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? Why is this variable not very useful for this analysis?

SOLUTION: This variable is not very useful for this analysis because there are significantly more observations with a value of 0 than a value of 1. Additionally, there are no observations with a value of 1 for `QikProp_.amidine` that "do not cross" the blood-brain barrier and only 1 observation of `QikProp_.amidine` value 1 that "crosses" the blood-brain barrier.

```
tally(bbb2_Class ~ QikProp_.amidine, data = mybbb2)
```

```
##           QikProp_.amidine
## bbb2_Class  0  1
##    Crosses 44  1
##    DoesNot 35  0
```

```
# run once you are ready to remove the variable to proceed
# this variable is not used anywhere below, so overwrite the data set
mybbb2 <- select(mybbb2, -QikProp_.amidine)
```

> part b. Create an appropriate training/test split from mybbb2 with a ratio of 70/30 to use throughout the problem. 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)

# for bagging, mtry = 49 so all the variables are used
class.bag <- randomForest(Class ~ ., data = train, ntree = 1000, mtry = 49)

class.bag
```

```
##
## Call:
##  randomForest(formula = Class ~ ., data = train, ntree = 1000,      mtry = 49)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 49
##
##         OOB estimate of  error rate: 10.71%
## Confusion matrix:
##         Crosses DoesNot class.error
## Crosses      28       3   0.0967742
## DoesNot       3      22   0.1200000
```

```
table(train$Class, predict(class.bag, train))
```

```
##
##           Crosses DoesNot
##   Crosses      31       0
##   DoesNot       0      25
```

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)

# no mtry value so it is defualt random forest
class.rf <- randomForest(Class ~ ., data = train, ntree = 1000)

class.rf
```

```
##
## Call:
##  randomForest(formula = Class ~ ., data = train, ntree = 1000)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 7.14%
## Confusion matrix:
##         Crosses DoesNot class.error
## Crosses      29       2   0.0645161
## DoesNot       2      23   0.0800000
```

```
table(train$Class, predict(class.rf, train))
```

```
##
##           Crosses DoesNot
##   Crosses      31       0
##   DoesNot       0      25
```

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)

# set 1 to be "crosses" and 0 to be "does not cross"
train2 <- mutate(train, Class = ifelse(Class == "Crosses", 1, 0))
test2 <- mutate(test, Class = ifelse(Class == "Crosses", 1, 0))

class.boost <- gbm(Class ~ ., data = train2, distribution = "bernoulli",
                   n.trees = 500)

class.boost
```

```
## gbm(formula = Class ~ ., distribution = "bernoulli", data = train2,
##     n.trees = 500)
## A gradient boosted model with bernoulli loss function.
## 500 iterations were performed.
## There were 49 predictors of which 40 had non-zero influence.
```

```
boost_estimate_train <- predict(class.boost, newdata = train2, n.trees = 500,
                          type = "response")


pred_class_train <- ifelse(boost_estimate_train > 0.5, 1, 0)
tally(~ pred_class_train)
```

```
## pred_class_train
##  0  1
## 25 31
```

```
table(train2$Class, pred_class_train)
```

```
##    pred_class_train
##      0  1
##   0 25  0
##   1  0 31
```

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: In order to compare the performance, we will see how accurate the model can predict the test set.

```
# for the bagging model
table(test$Class, predict(class.bag, newdata = test))
```

```
##
##           Crosses DoesNot
##   Crosses      10       4
##   DoesNot       2       8
```

```
(10+8)/(10+8+2+4)
```

```
## [1] 0.75
```

```
# for random forest model
table(test$Class, predict(class.rf, newdata = test))
```

```
##
##           Crosses DoesNot
##   Crosses       8       6
##   DoesNot       2       8
```

```
(8+8)/(8+8+2+6)
```

```
## [1] 0.666667
```

```
# for boosting model
test2 <- mutate(test, Class = ifelse(Class == "Crosses", 1, 0))
boost_estimate_test <- predict(class.boost, newdata = test2, n.trees = 500,
                           type = "response")
pred_class_test <- ifelse(boost_estimate_test > 0.5, 1, 0)
table(test2$Class, pred_class_test)
```

```
##    pred_class_test
##     0 1
##   0 8 2
##   1 7 7
```

```
(8+7)/(8+7+7+2)
```

```
## [1] 0.625
```

The Bagging model appears to have an accuracy of 75% on the test set. The Random Forest model appears to have an accuracy of 66.7% on the test set and the Boosting model appears to have an accuracy of 62.5% on the test set. Therefore, we would choose the Bagging model for predicting Class because the model had the higest accuracy.

Add 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:

```
set.seed(1)
nnet_class <- nnet(Class ~ ., train, size = 15)
```

```
## # weights:  766
## initial  value 38.831000
## iter  10 value 21.587315
## iter  20 value 17.010459
## iter  30 value 16.980151
## final  value 16.980120
## converged
```

```
trainpred <- predict(nnet_class, type = "class")
tally(Class ~ trainpred, data = train)
```

```
##          trainpred
## Class     Crosses DoesNot
##   Crosses      23       8
##   DoesNot       0      25
```

> 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:

```
set.seed(1)
nnet_class2 <- nnet(Class ~ ., train, size = 15, decay = 5e-4)
```

```
## # weights:  766
## initial  value 38.890644
## iter  10 value 20.413244
## iter  20 value 17.200852
## iter  30 value 17.166510
## iter  40 value 16.304502
## iter  50 value 15.946859
## iter  60 value 13.507398
## iter  70 value 11.505734
## iter  80 value 10.066955
## iter  90 value 8.151368
## iter 100 value 5.954050
## final  value 5.954050
## stopped after 100 iterations
```

```
trainpred2 <- predict(nnet_class2, type = "class")
tally(Class ~ trainpred2, data = train)
```

```
##           trainpred2
## Class      Crosses DoesNot
##   Crosses       30       1
##   DoesNot        2      23
```

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:

```
set.seed(1)
nnet_class3 <- nnet(Class ~ ., train, size = 15, decay = 5e-4, maxit = 1030)
```

```
## # weights:  766
## initial  value 38.890644
## iter  10 value 20.413244
## iter  20 value 17.200852
## iter  30 value 17.166510
## iter  40 value 16.304502
## iter  50 value 15.946859
## iter  60 value 13.507398
## iter  70 value 11.505734
## iter  80 value 10.066955
## iter  90 value 8.151368
## iter 100 value 5.954050
## iter 110 value 4.841553
## iter 120 value 4.172768
## iter 130 value 4.044346
## iter 140 value 3.597450
## iter 150 value 3.124801
## iter 160 value 2.978512
## iter 170 value 2.785024
## iter 180 value 2.555844
## iter 190 value 2.535562
## iter 200 value 2.526391
## iter 210 value 2.508483
## iter 220 value 2.498743
## iter 230 value 2.490085
## iter 240 value 2.369866
## iter 250 value 1.783930
## iter 260 value 0.829670
## iter 270 value 0.583957
## iter 280 value 0.566022
## iter 290 value 0.538523
## iter 300 value 0.533109
## iter 310 value 0.491527
## iter 320 value 0.440394
## iter 330 value 0.386001
```

```
## iter 340 value 0.369386
## iter 350 value 0.318506
## iter 360 value 0.313066
## iter 370 value 0.308847
## iter 380 value 0.299996
## iter 390 value 0.291081
## iter 400 value 0.287434
## iter 410 value 0.283187
## iter 420 value 0.278166
## iter 430 value 0.270208
## iter 440 value 0.262924
## iter 450 value 0.255176
## iter 460 value 0.248745
## iter 470 value 0.241410
## iter 480 value 0.233006
## iter 490 value 0.226084
## iter 500 value 0.218591
## iter 510 value 0.207126
## iter 520 value 0.202702
## iter 530 value 0.195266
## iter 540 value 0.192839
## iter 550 value 0.190092
## iter 560 value 0.189193
## iter 570 value 0.184497
## iter 580 value 0.172596
## iter 590 value 0.166186
## iter 600 value 0.157847
## iter 610 value 0.152611
## iter 620 value 0.150317
## iter 630 value 0.145633
## iter 640 value 0.144149
## iter 650 value 0.141481
## iter 660 value 0.138051
## iter 670 value 0.131357
## iter 680 value 0.126889
## iter 690 value 0.126055
## iter 700 value 0.123703
## iter 710 value 0.122135
## iter 720 value 0.120838
## iter 730 value 0.118925
## iter 740 value 0.116366
## iter 750 value 0.114796
## iter 760 value 0.112372
## iter 770 value 0.111763
## iter 780 value 0.110493
## iter 790 value 0.109869
## iter 800 value 0.109418
## iter 810 value 0.108380
## iter 820 value 0.105488
## iter 830 value 0.102474
## iter 840 value 0.100753
## iter 850 value 0.099825
## iter 860 value 0.099371
## iter 870 value 0.099079
```

```
## iter 880 value 0.098872
## iter 890 value 0.098744
## iter 900 value 0.098588
## iter 910 value 0.098503
## iter 920 value 0.098419
## iter 930 value 0.098350
## iter 940 value 0.098290
## iter 950 value 0.098107
## iter 960 value 0.098040
## iter 970 value 0.098004
## iter 980 value 0.097966
## iter 990 value 0.097938
## iter1000 value 0.097650
## iter1010 value 0.097341
## iter1020 value 0.097124
## iter1030 value 0.096971
## final  value 0.096971
## stopped after 1030 iterations
```

```
trainpred3 <- predict(nnet_class3, type = "class")
tally(Class ~ trainpred3, data = train)
```

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

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: We will do the same as we did before and compare models based on their accuracy in predicting the test set.

```
# neural net 1
pred <- predict(nnet_class, newdata = test, type = "class")
tally(Class ~ pred, data = test)
```

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

```
(8+8)/(8+8+6+2)
```

```
## [1] 0.666667
```

```
# neural net 2
pred2 <- predict(nnet_class2, newdata = test, type = "class")
tally(Class ~ pred2, data = test)
```

```
##          pred2
## Class     Crosses DoesNot
##   Crosses       8       6
##   DoesNot       1       9
```

```
(9+8)/(9+8+1+6)
```

```
## [1] 0.708333
```

```
# neural net 3
pred3 <- predict(nnet_class3, newdata = test, type = "class")
tally(Class ~ pred3, data = test)
```

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

```
(8+8)/(8+8+6+2)
```

```
## [1] 0.666667
```

From the confusion matrices above, it appears the the accuracy in predicting the test set from our first and third neural network model is 66.7% and the accuracy in predicting the test set from the second model is 70.8%. This means that we would choose the second neural network for predicting the test set. However, it is important to note that the difference between the accuracy in all of these models is only one more correct observation which could could change depending on the seed.

Add 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 Add 1, Add 2, and Add 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:

```
set.seed(1)
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:

```
set.seed(1)
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:  28
##
##  ( 9 19 )
##
##
## 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:

```
set.seed(1)
svm3 <- svm(Class ~ ., data = train, gamma = 0.0001, 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:  28
##
##  ( 9 19 )
##
##
## 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: Again, we will use the confusion matrices to determine how accurate each model is at predicting the test set.

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

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

```
(14+0)/(14+10)
```

```
## [1] 0.583333
```

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

```
##          svm2predtest
## Class     Crosses DoesNot
##   Crosses       9       5
##   DoesNot       2       8
```

```
(9+8)/(9+8+5+2)
```

```
## [1] 0.708333
```

```
# for svm2
svm3predtest <- predict(svm3, test)
tally(Class ~ svm3predtest, data = test)
```

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

```
(14+0)/(14+10)
```

```
## [1] 0.583333
```

It appears the that first and third SVM model predict the test set with an accuracy of 58.3% while the second model predicts the test set with a 70.8% accuracy.

> part e. Look over your responses to Add 1, Add 2, and Add 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: Between the three models that we picked as the best for each section, the Bagging model was the most accurate at predicting the test set at 75%. However, this corresponds to 6 mistakes and best models for the Neural Net and SVM sections only had 7 mistakes. Due to the test set being so small, it is difficult to determine which model is the best because even a change of the seed value could cause a different model to have a higher accuracy rate. This means that we would choose the Bagging model as our final model to predict Class with these concerns in mind.

Add 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: For Neural Nets, the non-linearity of the model comes from the the non-linear threshold activation functions that creates a non-linear output from the activation values. These activation functions can include a sigmoid function or a hyperbolic tangent function, as well as others. For SVMs, the non-linearity comes up when the feature space is enlarged to look to optimal separating hyperplanes. The feature space can be grown by using kernals (and the "kernal trick") which map a lower dimensional linear model in the enlarged space to a non-linear model in higher dimensional space.

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

SOLUTION: Random forest and boosting are similar that they both use decision trees to create models. They are also similar in the way that they lose interpretability due to the complexity of the models. Both methods utilize trees which automatically select variables. They differ because random forests build a number of larger and more bushy trees while boosting builds a large number of small trees in which the depth is fixed. The aim of random forests is variance reduction by averaging because each of the large trees has a high variance, but by averaging, the variance is brought down. On the other hand, boosting aims 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: The concepts of backpropagation and the kernel trick are related in the ways that they help make the computation of the model more feasible. Backpropagation uses gradient descent to optimize the calculation and fixing the weights to minimize error. Backpropagation computes the gradient one layer at a time, by starting from the last layer and working backwards to minimize the number of redundant computations. The kernel trick is used when creating the enlarged feature space to perform the computations of inner products quicker. The kernel allows us to do this in a much lower dimension space which causes the computations to be easier. By doing this, it makes the computations for an SVM model more feasible.

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: How are Out-Of-Bag Error estimates used in random forests?

A: The Out-Of-Bag (OOB) Error is used to measure the prediction error of a random forest. The Out-Of-Bag set is the set of all the observations that are not chosen in the bootstrap sampling process to be used in a creating a tree in the random-forest. The OOB Error is calculated by taking the average of each random-forest tree that the observation pair was left out of the bootstrap sample. This is similar to the leave-one-out cross validation error.