Chapter 18 and 19 Lab - Neural Nets and SVMs

This lab will practice the code and discuss concepts from Chapters 18 and 19 on Neural Nets and Support Vector Machines.

A Cautionary Tale

As mentioned in class, there are over 80 packages in R for neural nets. Not all are equal in how well they work, though. If you really want to fit neural nets, you should investigate the *keras* package.

The example here will show you why we can't just use functions blindly. It uses the Boston housing data set we've seen before, and tries to predict medv, the same response variable we had in our Chapter 17 lab. So this is a regression problem.

The code below loads the data, fits the model with only 2 hidden nodes in a single hidden layer on the whole data set, gets the MSE, and plots the predicted values versus the actual values in the data set. (The picture of the net itself is turned off.)

Run the code once and look at your plot of the predicted versus actual values.

```
BostonHousing <- MASS::Boston
nnet.fit <- nnet(medv/50 ~ ., data=BostonHousing, size = 2)
```

```
## # weights:  31
## initial  value 24.523913
## iter  10 value 13.686028
## iter  20 value 12.664561
## iter  30 value 7.828883
## iter  40 value 6.065299
## iter  50 value 4.794837
## iter  60 value 4.311866
## iter  70 value 3.889326
## iter  80 value 3.486055
## iter  90 value 3.421006
## iter 100 value 3.390303
## final  value 3.390303
## stopped after 100 iterations
```

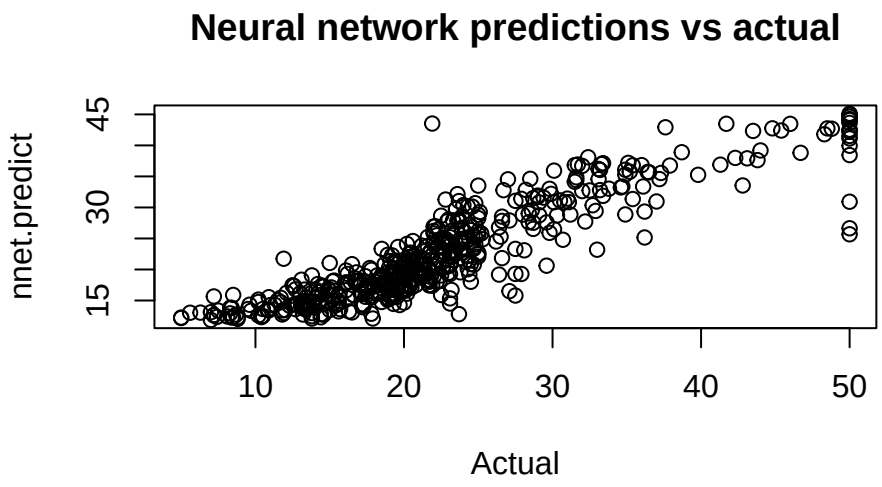
```
 # multiply by 50 to restore original scale
nnet.predict <- predict(nnet.fit)*50

# plotnet(nnet.fit)

# mean squared error: claimed to be 16.40581 from blog
mean((nnet.predict - BostonHousing$medv)^2)
```

```
## [1] 16.75051
```

```
plot(BostonHousing$medv, nnet.predict,
    main="Neural network predictions vs actual",
    xlab="Actual")
```



For some of you, this likely will run "fine", meaning you will see a scatterplot that shows the actual versus predicted values have a fairly strong positive linear relationship. Others of you will get something very different. If you got the fairly strong positive linear relationship, try running the code again. Everyone should run the code chunk several times. What is happening to your predictions / the plot?

Why is this happening?

There is no seed set, and clearly, the results are not reproducible without a seed set. Sometimes we get completely (or almost completely) nonsense predictions that are constant. Even if we set a seed, the different performance we are obtaining indicates there is something off with the function and/or our understanding of the parameters we need to be setting in order for this neural net to run appropriately. Setting a seed allows us to get reproducible results, but we don't want reproducible results that don't make sense! The blog I was following for this doesn't set a seed and doesn't recognize there are these issues with the solution. I tried scaling, changing other tuning parameters, etc. and wasn't able to get a consistent behavior for the solution (for regression).

If you want to really get into neural nets (especially for regression), go use *keras* (or python). *keras* still doesn't have the best documentation, but there is more out there, and it is more stable (it still can take a while to install though!).

For our neural nets below, to keep using the *nnet* package, we'll stick with classification where the behavior seems to be stable. (From what I can tell, anyway.)

Neural Nets for classification

In the Chapter 17 lab, for classification, we looked at the Carseats data, so let's return to that. In these data, `Sales` is a continuous variable, we want to consider a binary classification problem, so we begin by converting it to a binary variable. We use the `ifelse()` function to create a variable, called `High`, which takes on a value of `Yes` if the `Sales` variable exceeds 8, and takes on a value of `No` otherwise:

```
Carseats <- Carseats %>%
  mutate(High = as.factor(ifelse(Sales <= 8, "No", "Yes")))
```

We again split the observations into a training set and a test set:

```
set.seed(495)

n <- nrow(Carseats)
```

```
train_index <- sample(1:n, 0.5 * n)
test_index <- setdiff(1:n, train_index)

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

We now use the `nnet` function to fit a neural net in order to predict `High` using all variables but `Sales` (that would be a little silly...). Besides the model formula, we have to specify the number of nodes in the hidden layer. *nnet* only allows a single hidden layer.

```
set.seed(495)
nnet_carseats <- nnet(High ~ . - Sales, train, size = 4)
```

```
## # weights:  53
## initial  value 134.763466
## iter  10 value 117.716502
## iter  20 value 116.215623
## iter  30 value 116.166218
## final  value 116.166053
## converged
```

This neural net had 4 hidden nodes in the single hidden layer, and with this seed, it converged in less than 100 iterations (for me). Did yours converge?

Let's try increasing the number of nodes in the hidden layer. Does this neural net converge?

```
set.seed(495)
nnet_carseats2 <- nnet(High ~ . - Sales, train, size = 8)
```

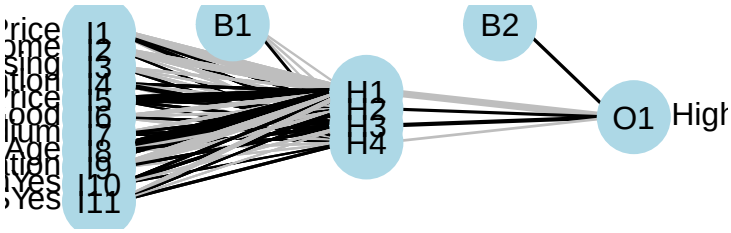
```
## # weights:  105
## initial  value 140.932482
## iter  10 value 134.822204
## iter  20 value 131.976453
## iter  30 value 130.553764
## iter  40 value 126.988916
## iter  50 value 124.388478
## iter  60 value 118.305423
## iter  70 value 116.338662
## iter  80 value 115.266852
## iter  90 value 115.097672
## iter 100 value 115.065479
## final  value 115.065479
## stopped after 100 iterations
```

Try adding the *maxit* parameter to the net with 8 hidden nodes and see if you can find a value for which it converges.

Let's see how we did with the simpler net with 4 hidden nodes (easier to view, etc.)

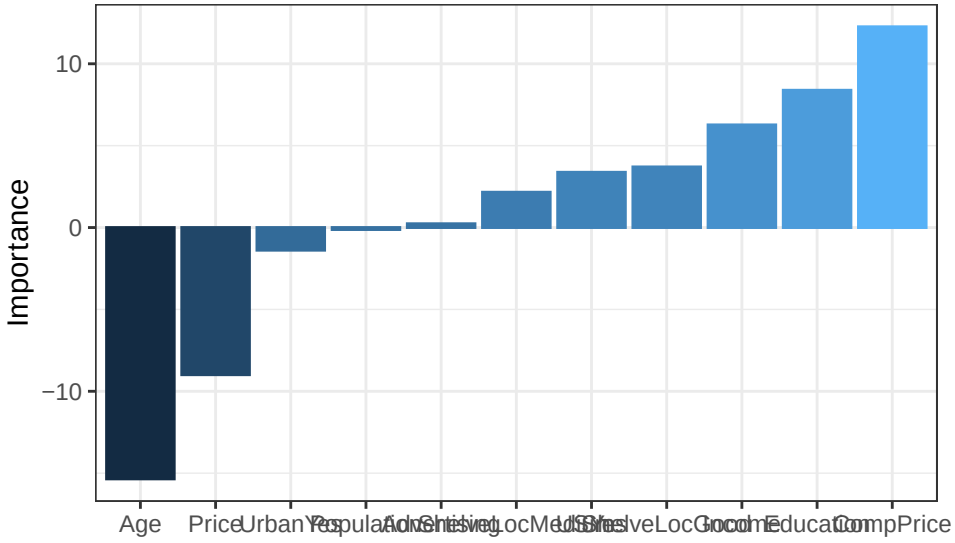
Here's how to plot the neural net:

```
plotnet(nnet_carseats)
```



To look at variable importance, because there are only 2 levels to the response here, we can just run:

```
olden(nnet_carseats)
```



Look at the help page for the *olden* function. Why is this method considered superior to Garson's algorithm?

Finally, let's evaluate the performance of the neural nets. We can check how they do on both the training and the test data sets. Here is code to get predictions for the training and test data using the network with 4 hidden nodes.

```
trainpred <- predict(nnet_carseats, type = "class")
pred <- predict(nnet_carseats, newdata = test, type = "class")
```

Now let's get the confusion matrices for both train and test.

```
tally(High ~ trainpred, data = train)
```

```
##      trainpred
## High   No Yes
##   No  113   5
##   Yes  51  31
```

```
tally(High ~ pred, data = test)
```

```
##      pred
## High   No Yes
##   No  106  12
##   Yes  63  19
```

Get predictions for the model with 8 hidden nodes and enough iterations to converge on both the training and test sets. How well does this model do compared to the model with 4 hidden nodes?

Try one more model with parameters of your choice for at least one new option. How does it do?

Which neural net model do you prefer overall?

SVMs for Classification

The *e1071* package and the *svm* function allows for SVMs to be fit in R.

What else does it say it can do via the help menu?

Fitting a model is easy with the usual formula interface.

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

```
##
## Call:
## svm(formula = High ~ . - Sales, data = train, gamma = 0.75, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  199
##
##  ( 117 82 )
##
##
## Number of Classes:  2
##
## Levels:
##  No Yes
```

The help menu lists the other possible kernels, and there are options that need to be specified for each.

Fit a model called *svm2* using a polynomial kernel of degree 4 with a gamma of your choice. (To have something to compare to the provided svm1.)

Getting predictions is pretty easy using predict, as expected. We can then get confusion matrices to see how well the method is doing.

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

```
##      svm1predtrain
## High   No Yes
##   No  118   0
##   Yes   0  82
```

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

```
##      svm1predtest
## High   No Yes
##   No  111   7
##   Yes  64  18
```

What do you notice about the performance of svm1?

How does your svm2 compare?

Fit another svm to try to improve on the performance on the test set. Can you find a better model than either svm1 or svm2?

Practice - Regression

We use a data set that we previously examined when learning about ridge regression that we are returning to for practice.

```
data(Hitters)
# ?Hitters # MLB data from 1986 and 1987; 322 obs on 20 variables
```

We want to predict Salary, which is missing for 59 players.

```
Hitters <- na.omit(Hitters) #263 obs now
```

Unfortunately, since the *nnet* package doesn't seem to be reliable for regression, we don't want to fit those models here, and instead will just fit some SVMs after checking out options for neural nets.

Do a brief google search. What other packages do you find out there for neural nets in R? Can you find blog entries about regression with neural nets? Do they use any of these packages?

SOLUTION

Use SVMs to explore models for predicting salary. Be sure to explore the SVM model options and the various parameter settings.

SOLUTION